PROJET 2CCPP – LAYING GRASS

Documentation technique

AUTEURS : CUREAU MELVIN ; CHEVALLEREAU AXEL

CAMPUS : TOURS

DATE : 27/11/2023

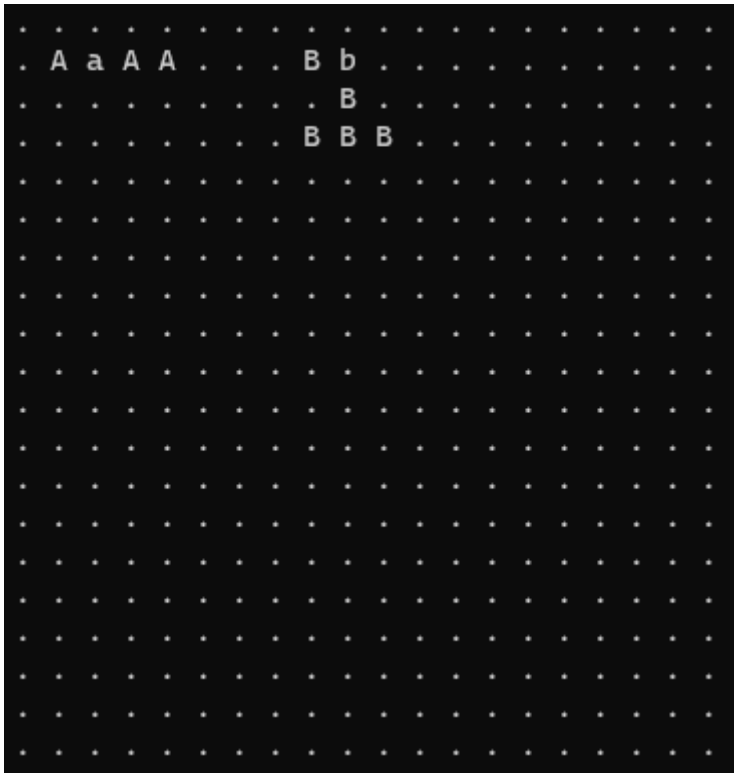
ANNEE SCOLAIRE 2023-2024

Ce document comporte 4 pages :

- Page 1 : Page de garde ; Sommaire
- Page 2 : Introduction ; Architecture
- Page 3 : Architecture ; Choix techniques
- Page 4 : Références ; Conclusion

Introduction/

Le jeu "***Laying Grass***" est une aventure stratégique où votre objectif est de créer le plus vaste territoire herbeux sur un plateau de jeu en constante expansion. Inspiré du jeu du même nom dans l'émission de télé-réalité Netflix "***The Devil's Plan***", ce défi mettra à l'épreuve votre capacité à planifier et à étendre votre influence tout en anticipant les mouvements de vos adversaires.



Capture de la grille générée en console lors de l'exécution du jeu.

Architecture/

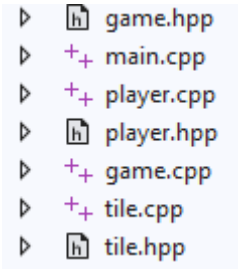
Voici la structuration du dossier de notre projet :

- Un dossier '*Docs*' / contenant toutes les documentations nécessaires
- Un dossier '*Src*' / contenant tous nos fichiers en .c et en .h
- Le fichier 'LICENSE' / contient la licence MIT qui régule l'accès à notre projet
- Le fichier 'README' / explique brièvement notre projet

Structuration du dossier *source* de notre projet :

- Un fichier '*main.cpp*'/ responsable de l'initialisation du jeu et de la boucle du jeu.
- Un fichier '*player.ppc'*/ représente un joueur dans le jeu, avec des méthodes pour gérer les tuiles du joueur, placer des tuiles sur une grille, trouver la taille du plus grand carré formé, etc.
- Un fichier *'tile.cpp'/* représente les tuiles de jeu avec différents motifs définis par des fonctions spécifiques. Ces motifs sont utilisés lorsque les joueurs choisissent et placent leurs tuiles sur la grille de jeu.

- Un fichier ‘game.cpp’/ représente les fonctions globales du jeu.
- Les fichier header propre à chaque fichier C++ du projet.



Choix techniques/

- Langage de programmation : C++
- Editeur de code : Visual Studio 2022
- Système de contrôle de version : Git et GitHub
- Système d’exploitation : Windows / Linux / MacOS

Le code est structuré en plusieurs fichiers pour faciliter la lisibilité et la maintenance. Les fonctions sont découpées de manière modulaire et indépendante pour faciliter la réutilisation du code. L’interface utilisateur est gérée via la console.

Dans l'ensemble, le choix du langage C++ et de l'interface console est adapté à ce type de jeu minimaliste et permet de se concentrer sur la logique du jeu plutôt que sur des considérations graphiques. Cependant, cela peut être considéré comme un choix limitant pour ceux qui souhaitent développer des jeux plus graphiques et interactifs, nécessitant une interface utilisateur plus élaborée.

Structure de données/

Notre jeu utilise plusieurs structures de données pour stocker les informations nécessaires. La grille est stockée dans une matrice à deux dimensions. Chaque case contient un entier représentant son état actuel, qui peut être vide, plein ou une cellule de départ.

Ajout de niveaux/

Le programme ne génère pas automatiquement de nouveaux niveaux. Il est donc limité à un ensemble fini de niveaux prédéfinis (96) et stockés dans des fichiers ‘***tile.cpp’***. Cependant il est possible pour l’utilisateur d’ajouter des niveaux personnalisés dans ce fichier. Chaque niveau suit le format suivant :

- Il doit d’abord être déclarer dans le switch. Le numéro de case ainsi que de la fonction form doit être celui du niveau.
  L’image ci-dessous prend exemple sur le **niveau 1** au sein du fichier **tile.cpp**.

```
// Fonction pour renvoyer la forme de la tuile en fonction de son type
std::vector<std::vector<char>> Tile::Form(char refChar) const {
    switch(type) {
        case 1:
            return form1(refChar);
```

- Il faudra par la suite déclarer la forme au sein d'une fonction, toujours dans notre fichier tile.cpp.
  Les espaces vides sont représenter par '.', la lettre unique du joueur est représenté par ***'refChar'*** et enfin le la lettre unique régissant le placement de la tuile sera représenter par ***'static_cast<char>(std ::tolower(refChar))'***

```
std::vector<std::vector<char>> Tile::form1(char refChar) {
    return {
        {static_cast<char>(std::tolower(refChar)), '.', '.'},
        {refChar, refChar, refChar}
    };
}
```

- Enfin il vous faudra ajouter le prototype de la fonction au sein de la classe ***Tile*** dans le fichier tile.hpp

```
static std::vector<std::vector<char>> form1(char refChar);
```

Limitations/

- **Formes de tuiles :**

Le programme définit un ensemble de formes de tuiles prédéfinies (form1 à form10). L'ajout de nouvelles formes de vignettes nécessite de modifier le code.

- **Version console :**

L'utilisation de la console pour l'interface utilisateur peut également être considérée comme limitante, car elle ne permet pas de réaliser une interface graphique plus élaborée. Cependant, elle est adaptée à ce type de jeu minimaliste et permet de se concentrer sur la logique du jeu plutôt que sur des considérations graphiques.

Références/

Pour la réalisation de ce projet nous avons utilisé VisualStudio 2022 en tant qu'IDE en fonction de la préférence de chaque membre du groupe. Ce code du projet est publié sur GitHub, contenant nos documentations utilisateurs et techniques ainsi que nos fichiers de licence et le readme, en suivant le lien suivant :

https://github.com/MelvinCureau

Conclusion/

En résumé, entreprendre ce projet a été une expérience enrichissante et stimulante. Nous avons pu mettre en pratique les notions de programmation apprises, ainsi que découvrir de

nouveaux outils en CPP. Il nous a également fallu être créatifs et réfléchis dans la conception des niveaux variés et intéressants.

Bien que notre sélection d'outils de gestion de niveau puisse sembler un peu basique, elle fonctionne de manière fiable et permet des mises à jour de niveau faciles. Cependant, il est vrai que nous pouvons explorer d'autres options, comme une gestion automatique des niveaux. Concernant l'architecture, nous avons choisi une structure de code modulaire qui permet une meilleure organisation et maintenance du projet. Nous avons également utilisé des fonctions réutilisables, ce qui contribue à rendre le code plus lisible et plus simple.

En termes d'améliorations possibles, il serait intéressant de travailler sur une version graphique du jeu pour offrir une expérience plus immersive et conviviale. De plus, la génération automatique de niveau serait un excellent ajout pour offrir une jouabilité infinie.

Finalement, nous sommes satisfaits du travail accompli et sommes convaincus que le jeu Laying Grass en C++ offre une expérience de jeu amusante et stimulante pour les amateurs de jeu.