Telecom Saint-Étienne
FISE 1

Mini-Projet

Étude en Filière Ingénieur sous Statut Étudiant
28.04.2024

Lucas Lescure - Tom Paillet

Lettris - Tetris avec des lettres

Table des contenus

1. Specification du jeu

1.1. Principe

Le jeu Lettris est un jeu de type Tetris, c'est-à-dire que des blocs tombent du haut de l'écran et le joueur doit les placer de manière à former des lignes complètes pour les faire disparaître. La particularité de Lettris est que les blocs sont des lettres de l'alphabet et que le joueur doit former des mots pour les faire disparaître.

Le jeu se déroule sur une grille de 10x15 cases et le joueur peut déplacer les blocs à gauche, droite, ou bien les faire tomber plus rapidement. Apres avoir formé un mot de plus de 4 lettres, les lettres disparaissent et le joueur gagne des points en fonction de la taille du mot. Le jeu se termine lorsqu'il n'est plus possible de rajouter des lettres à la grille.

1.2. Fonctionnalités

En accord avec le cahier des charges, lorsque le jouer forme un mot, ceci incrémentera un compteur qui sera affiché dans la colonnes des mots formés. Pour ajouter sur ceci, le joueur aura a sa disposition un journal contenant les 10 derniers mots formés ainsi que les points gagnés par mots. Ces mots sont sauvegardés au fur et à mesure dans un fichier texte placé à `./resources/foundWords.txt`.

De plus pour encourager un air compétitif, le joueur pourra consulter pendant que le jeu se déroule l'état de son score ainsi que le score maximal enregistré sur le jeu. L'historique des scores est sauvegardé à chaque fin de partie dans un fichier texte placé à `./resources/Scores.txt`.

Afin d'être fidèle au jeu Tetris, le joueur pourra voir dans une rubrique dédiée les prochaines lettres qui apparaitront sur l'écran. Et lorsque le jeu se déroule la vitesse de chute des lettres augmentera progressivement pour augmenter la difficulté du jeu. Ceci est réalisé sur 10 niveaux de difficulté, et parmis ces niveaux, 4 stades de difficulté sont définis ayant chacun un thème de couleur différent pour pouvoir les distinguer. Le stade de difficulté et le niveau sur lequel se trouve le joueur sont affichés à l'utilisateur séparément.

Pour faire disparaitre les mots formés, il y aura une pause de 1s et les mots seront colorés en cyan pour que le joueur puisse voir les mots formés. Ces mots peuvent dans les deux sens horizontalement et verticalement. Si la grille de jeu ne peux plus créer de nouvelles lettres le jeu s'arrête et un message de fin de partie est affiché. En fonction de la réponse de l'utilisateur, le jeu à la possibilité de se relancer sur une même instance en mettant à jour les scores et en vidant la liste des mots trouvé de la partie précédente et en réinitialisant l'ensemble des composant affichés.

Si l'on souhaite changer les couleurs des différents stages du jeu il est possible de modifier le fichier `./resources/colorscheme.theme` en attribuant un code hexadecimal sur 32bit sur l'emplacement d'un theme.

2. Diagramme de classes

Le diagramme de classe complet du projet est fournit en annexe. Il ne figurera pas les relation d'association avec les classes SFML par crainte d'encombrer le diagramme avec des classes qui sont peu utilisées.

Sur celui-ci on distingue 3 grande parties:

- Partie interfaces: Composé de `GameLevel`, `GameLogs`, `GameLetter`, `GameScore`. Leur rôle est de gérer et afficher les information récupérées du jeu sur des blocks qui sont affichés à l'utilisateur.
- Partie contrôle: Composé de `Config`, `MainGame` et `Input` Leur rôle est d'assurer le bon fonctionnement et la communication entre les autres classes agrégées. En particulier Config qui regroupe la plupart des paramètres du jeu.
- Partie jeu: Composé de `LetterBlock`, `GameGrid` et `Wordle`. Il permettent de contrôler la logique interne du jeu notamment pour la reconnaissance des mots et la mise à jour de la grille de jeu.

L'une des classes les plus importantes et plus utile à été la classe `Config` qui est possède une relation d'association avec presque toutes les classe du projet. Elle est construite à partir d'une structure de singleton ce qui lui permet d'avoir une seule et unique instance possible. En combinaison avec des attribut public statiques elle permet à toutes les classes associés d'utiliser et accéder les paramètres nécessaires. Ceci est surtout un atout lors de la lisibilité du code car cela evite que chaque classe possède leur propres paramètres de configuration, on centralise tout dans une classe globale.

Il faut noter qu'au niveau du code il y a plus de paramètres que l'on aurai pû ajouter à la classe `Config`.

La classe `MainGame` est responsable de gérer toutes les autres classes par voie d'aggregation. C'est à l'intérieur le jeu entier est contrôlé, grâce a un environment de plus haut niveau que les autres classes. Ceci rend le code plus malléable et facile à debugger.

La classe `GameGrid` sera celle qui permet de gérer la grille de jeu et qui manipulera les `LetterBlock` par composition multiple pour former des combinaisons de caractères, utilisé au niveau du `Wordle` en agrégation pour la detection de mots.

Enfin la classe `LetterBlock` est responsable du contrôle de chaque block de lettres et sera utilisé par la classe `GameGrid` pour former des mots ainsi que de les afficher sur la grille.

3. Description des classes

3.1. Config - Lucas Lescure

Un singleton qui stock les configurations globales du jeu de sorte qu'il puisse partager ces configurations avec d'autres classes externes sans nécessiter une relation d'agrégation, composition ou hérédité.
Cette classe est faite pour être appelée à partir d'un instance partagée et statique.

Cela permet de charger les configuration des configs, telle que les textures et la mise en page qu'une seule fois.

En outre, il améliore la lisibilité du code en stockant les configurations globales en un seul endroit, ce qui évite d'encombrer les classes avec un trop grand nombre de membres constants.

La classe est unique en ce sens qu'elle ne peut être ni copiée ni reconstruite puisqu'elle fonctionne comme un singleton.

On notera aussi que lees attribus de la classe possèdent le mot clé `constexpr` ce qui permet de les initialiser à la compilation. Un aspect qui à voulu être étudié pour améliorer les performances du jeu, même si l'impact est negligeable.

Elle peut être appelée en récupérant l'instance de la classe à l'aide de la méthode `getInstance()`. Cette instance peut ensuite être stockée dans un pointeur de classe pour de multiples appels de configuration.

Exemple d'utilisation :

```
// Stocke l'instance dans un pointeur 'config' en vue d'appels repetes de parametres.
Config* config = Config::getInstance();

// Les membres partages de Config sont accessibles a l'aide du pointeur d'instance
printf("Title: %s \nFramerate: %d",config->window_title, config->window_framerate);

// Utilise pour les appels uniques ne necessitant pas de pointeur pour stocker l'instance
Vector2i Size = Config::getInstance()->window_size;
```

3.2. GameGrid - Tom Paillet

Il s'agit grille en 2D faite avec des `LetterBlocks` qui forment la grille de jeu.C'est ici que la majeure partie du jeu est contrôlée en traitant les données du jeu avec les classes correspondantes. On notera aussi que les dimensions de la grille sont trouvées dans la classe `Config`.

L'agrégation avec `Input` et `GameWordle` est utilisée pour récupérer les données de l'utilisateur et déterminer quels blocs - formant un mot valide - doivent être détruits. D'autres agrégations sont utilisées comme composants connexes qui communiquent avec la grille de jeu pour produire les valeurs de jeu correctes.

3.3. GameLetter - Tom Paillet

La classe permet de donner la lettre suivante au hasard et l'affiche dans l'emplacement prévu sur le côté du jeu. La lettre sera ensuite récupérée par la grille, auquel cas une nouvelle lettre aléatoire est alors chargée.

La classe permet une randomisation simple avec des getters et des setters.
Exemple d'utilisation :

```
GameLetter next = GameLetter();  // initialise avec une lettre aleatoire

char c = next->getLetter();      // recupere la lettre actuellement affichee

c = next->randLetter();          // renvoie une lettre aleatoire

next->changeLetter();            // randomise la lettre affichee
```

3.4. GameLevel - Lucas Lescure

C'est une classe qui gère le niveau et l'étape du jeu. La vitesse du jeu est accélérée au par rapport au score actuel. Il modifie également la palette de couleurs en fonction de l'étape.

Exemple de code :

```
GameLevel level = GameLevel();

// Affiche le niveau et l'etape
level.render(window);

// Augmente le niveau et met a jour le stage consequemment.
level.levelUp();

// Recupere la vitesse du jeu
int speed = level.getSpeed();
```

3.5. GameLogs - Tom Paillet

Stocke et affiche l'historique des mots ainsi que les points gagnés par chacun d'entre eux dans deux colonnes. La colonne des points est alignée à gauche et l'historique contient 10 mots maximum.

Elle possède aussi un attribut `count` qui permet de compter le nombre de mots formés et de les afficher au niveau du titre du module.

Exemple de code :

```
GameLogs logs = GameLogs();           // initialise les composants avec un historique vide
logs->emplace("new text");            // place le mot en haut avec le score correspondant
logs->emplaceLog("newer text");       // place le mot en haut sans mise a jour du score
logs->emplacePoints("newest text")    // place le score en haut sans mise a jour du texte
```

3.6. GameScore - Lucas Lescure

Une classe qui gère et affiche le score du jeu. Elle permet d'afficher le meilleur score ainsi que le score actuel, sauver et charger les scores avec un fichier texte et aussi de donner la valeur d'un mot.

LE modification de la pallete de couleur est réalisant en modifiant l'attributs `Config::colorScheme_` qui est utilisé dans la methode `GameGrid::newBlock()` pour attribuer des couleurs aléatoires au block qui tombent.

3.7. Input - Tom Paillet

Cette classe récupère les évènements (actions de l'utilisateur) qui se produisent à l'aide de la méthode `pollEvent()` et stocke l'entrée dans le membre `input_`. Cette direction peut ensuite être récupérée avec la méthode `getInput()`.

3.8. LetterBlock - Lucas Lescure

Bloc contenant une lettre qui est utilisé en grille afin de créer des mots. La taille par défaut est 33.6x32.4 comme spécifié dans la classe `Config`.

Elle permet des opérations simples tel que cacher le bloc et changer la couleur de remplissage.

```
LetterBlock block();            // Initialise par defaut a 33.6x32.4px avec comme texte un espace (' ')
block.display(false);           // cache le bloc (alpha = 0)
```

Les fonctions get et set permettent de récupérer ou modifier des données.
Exemple de code :

```
LetterBlock block('A');                         // Cree un bloc de 33.6x32.4 avec un A

char letter = block.getLetter();                // Copie l'attibut 'letter'
block.setLetter('B');                           // Mets la lettre a 'B'
block.setPosition({0,0});                       // Change la position du LetterBlock
block.setState(State::Falling);                 // Change l'etat du bloc
if (block.isState(State::Falling)) {            // Verifie l'etat du bloc
     block.setColor(sf::Color::Red)             // Change la couleur de remplsisage du bloc
}
if (block.isHidden()) {                         // Verifie si le bloc est cache
     block.getPosition(grid);                   // Renvoi la position du bloc
}
State state = block.getState();                 // Recupere l'etat du bloc
sf::RectangleShape rect = block.getBlock();     // Recupere une copie de l'attribut du bloc
```

Pour son utilisation dans un grille 2D, une surcharge de l'opérateur '=' a été faite pour changer les propriétés d'un bloc à l'autre, permettant ainsi d'avoir une illusion de déplacement sans le déplacer graphiquement. Exemple de code :

```
LetterBlock block1('A');
Letterblock block2('B');
block1.setColor(sf::Color::Black);
block2.setColor(sf::Color::Red);
block1 = block2                         // les proprietes de block2 sont transferees au block1
```

3.9. MainGame - Lucas Lescure

Elle est utilisé pour afficher et contrôler la logique du jeu en utilisant les classes précédentes. Elle est responsable de l'initialisation des classes et de la gestion des événements de jeu. Ceci est fait principalement dans la méthode `update()` qui est appelée en boucle pour mettre à jour les éléments du jeu.

3.10. Wordle - Tom Paillet

Cette classe est utilisée pour vérifier si un mot est valide. Elle est utilisée par la classe `GameGrid` pour vérifier si un mot est valide et doit être détruit. La detection des mots est faite en parcourant la grille de jeu et en vérifiant si une combinaison de lettres forme un mot contenu dans la map de mot créée à partir d'un dictionnaire de mots.

4. Annexe

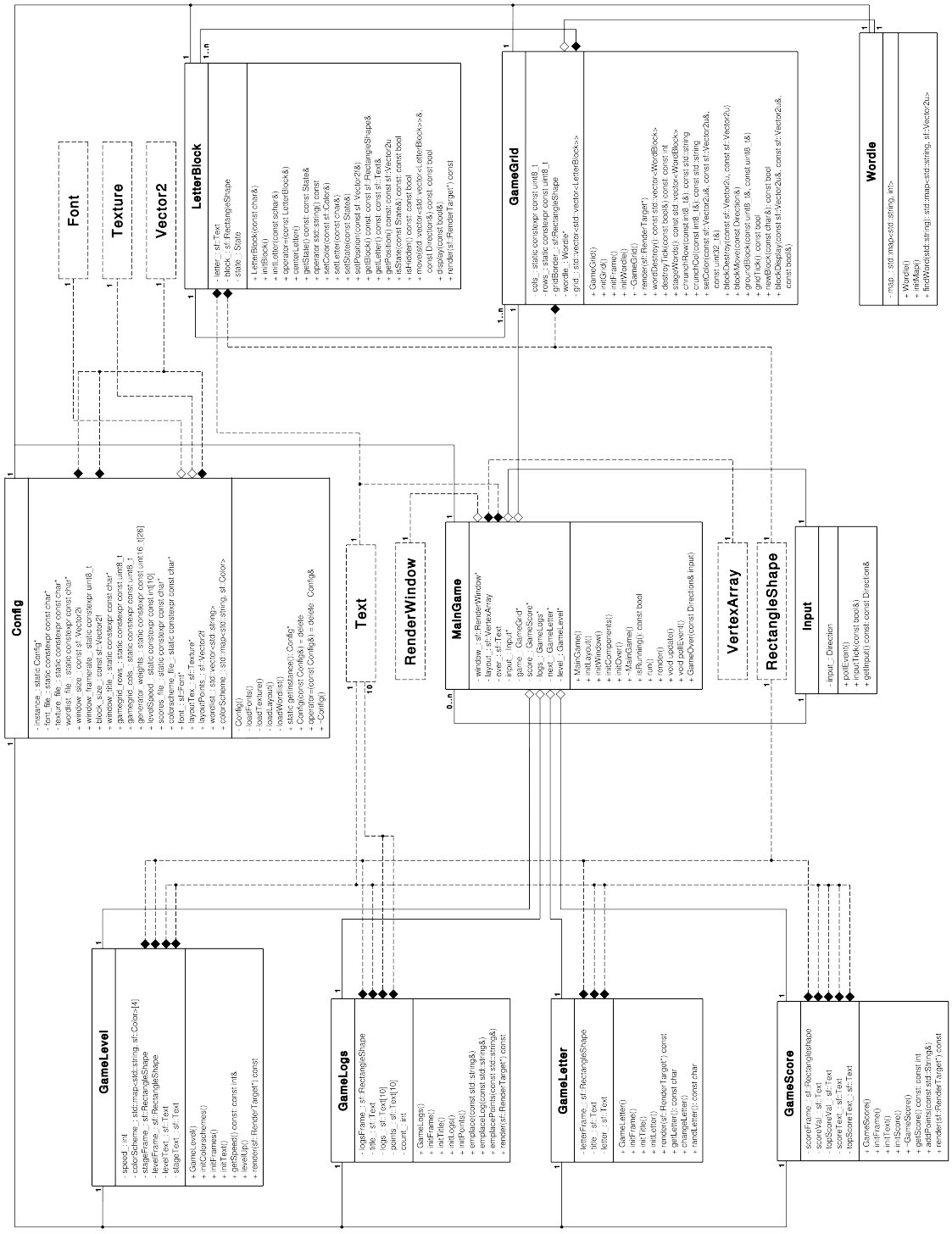

Figure 4.1. Diagramme de classe du projet