

Manuel Développeur



1. Présentation du jeu

2048 : Le défi du puzzle stratégique !

Le jeu 2048 est un puzzle captivant qui met à l'épreuve ta réflexion et ta stratégie. Sur une grille de 4x4, l'objectif est de fusionner des tuiles de même valeur pour atteindre la fameuse tuile 2048. Pour y parvenir, utilise les touches **z**, **q**, **s**, **d** ou les flèches directionnelles afin de choisir la direction du mouvement : **z** pour aller vers le haut, **q** pour aller à gauche, **s** pour descendre et **d** pour aller à droite. Lorsque deux tuiles identiques se rencontrent, elles fusionnent et doublent leur valeur. Chaque mouvement génère une nouvelle tuile aléatoire, il est donc essentiel de bien gérer l'espace pour éviter de bloquer la grille.

L'objectif ultime est d'atteindre la tuile 2048 ! 🎉 Mais le jeu ne s'arrête pas là : il est possible de continuer à jouer pour battre son propre record et maximiser son score. Chaque décision compte et la gestion de l'espace est cruciale. Le jeu est à la fois simple à comprendre mais difficile à maîtriser, rendant chaque partie unique et addictive. Plus tu joues, plus tu affines ta stratégie et découvres des techniques pour optimiser tes déplacements. Que tu aies quelques minutes devant toi ou plusieurs heures, notre 2048 s'adapte parfaitement à ton rythme de jeu.

Avec son interface fluide et son concept addictif, notre 2048 est un jeu qui met au défi ton esprit logique et ta capacité à anticiper les mouvements futurs. Il favorise la réflexion et la patience, tout en offrant une expérience de jeu gratifiante et engageante.

Vers une analyse approfondie : Conception et développement de 2048

Maintenant que tu connais les bases du jeu, il est temps de plonger plus profondément dans son fonctionnement. La suite de ce manuel détaille la conception et l'analyse du jeu 2048, en explorant ses mécanismes, son algorithme et son développement technique.

Table des matières

1.	Présentation du jeu	2
2.	Analyse et conception du logiciel	4
2.1.	Diagramme de cas d'utilisation	4
2.2.	Diagramme de séquences	5
2.3.	Diagramme de Classes	8
3.	Programmation du logiciel	10
3.1.	L'architecture du logiciel	10
3.2.	Points importants du code	12

2. Analyse et conception du logiciel

2.1. Diagramme de cas d'utilisation

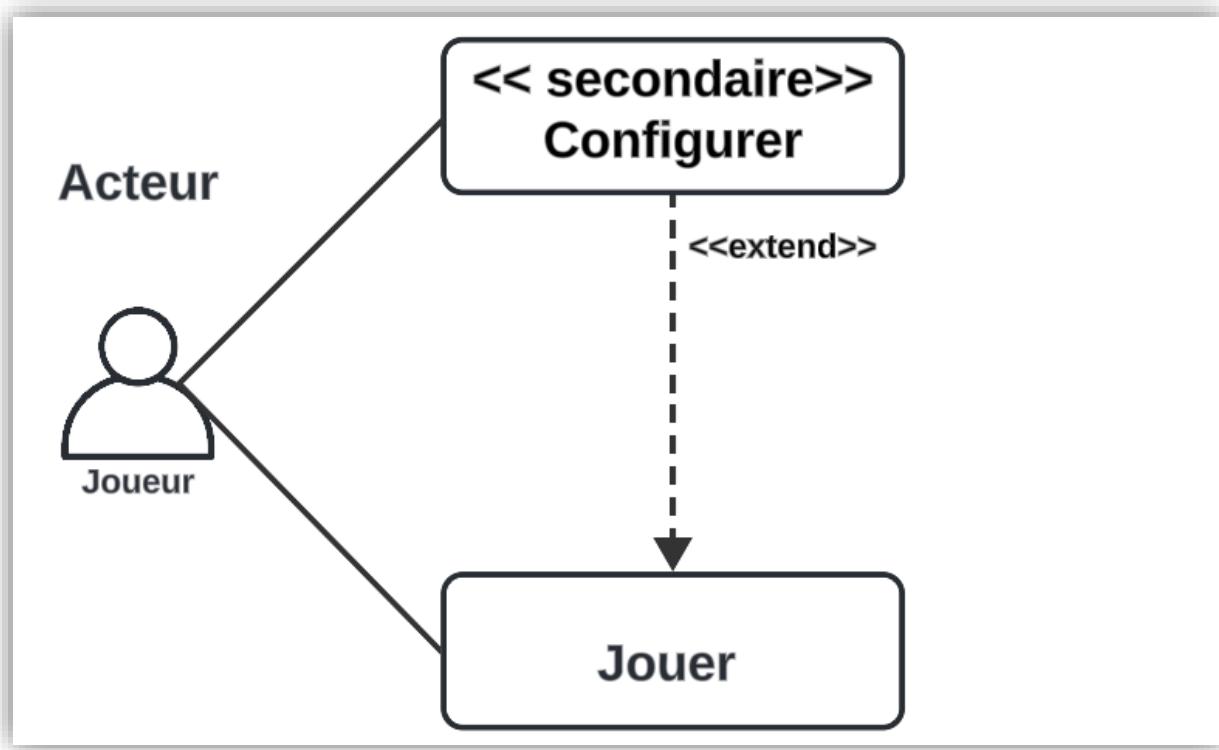


FIGURE 1 - DIAGRAMME DE CAS D'UTILISATION

Le diagramme de cas d'utilisation du jeu 2048 (cf. figure 1) illustre les interactions possibles entre l'utilisateur et le système. L'acteur principal est le joueur humain, qui interagit avec le jeu pour effectuer différentes actions. Le système, quant à lui, représente le jeu 2048 en tant qu'entité regroupant l'ensemble des fonctionnalités mises à disposition.

L'utilisation principale du jeu est de permettre au joueur de jouer une partie. Cette action constitue le cœur du système, car elle englobe toutes les interactions essentielles, comme le déplacement des tuiles et la tentative d'atteindre la tuile 2048. Autour de cette action centrale, d'autres fonctionnalités secondaires viennent enrichir l'expérience utilisateur.

Parmi ces fonctionnalités secondaires, on trouve la possibilité de configurer le jeu en recommençant une partie, ce qui permet au joueur de réinitialiser la grille et de débuter un nouveau jeu. Cette action est étroitement liée à la partie en cours, car elle peut être exécutée à tout moment si le joueur souhaite retenter sa chance.

Dans l'ensemble, ce modèle met en évidence la flexibilité du jeu 2048 et la fluidité des interactions entre le joueur et le système. Il permet d'avoir une vision claire des fonctionnalités essentielles tout en montrant comment elles s'articulent pour offrir une expérience intuitive et agréable.

2.2. Diagramme de séquences

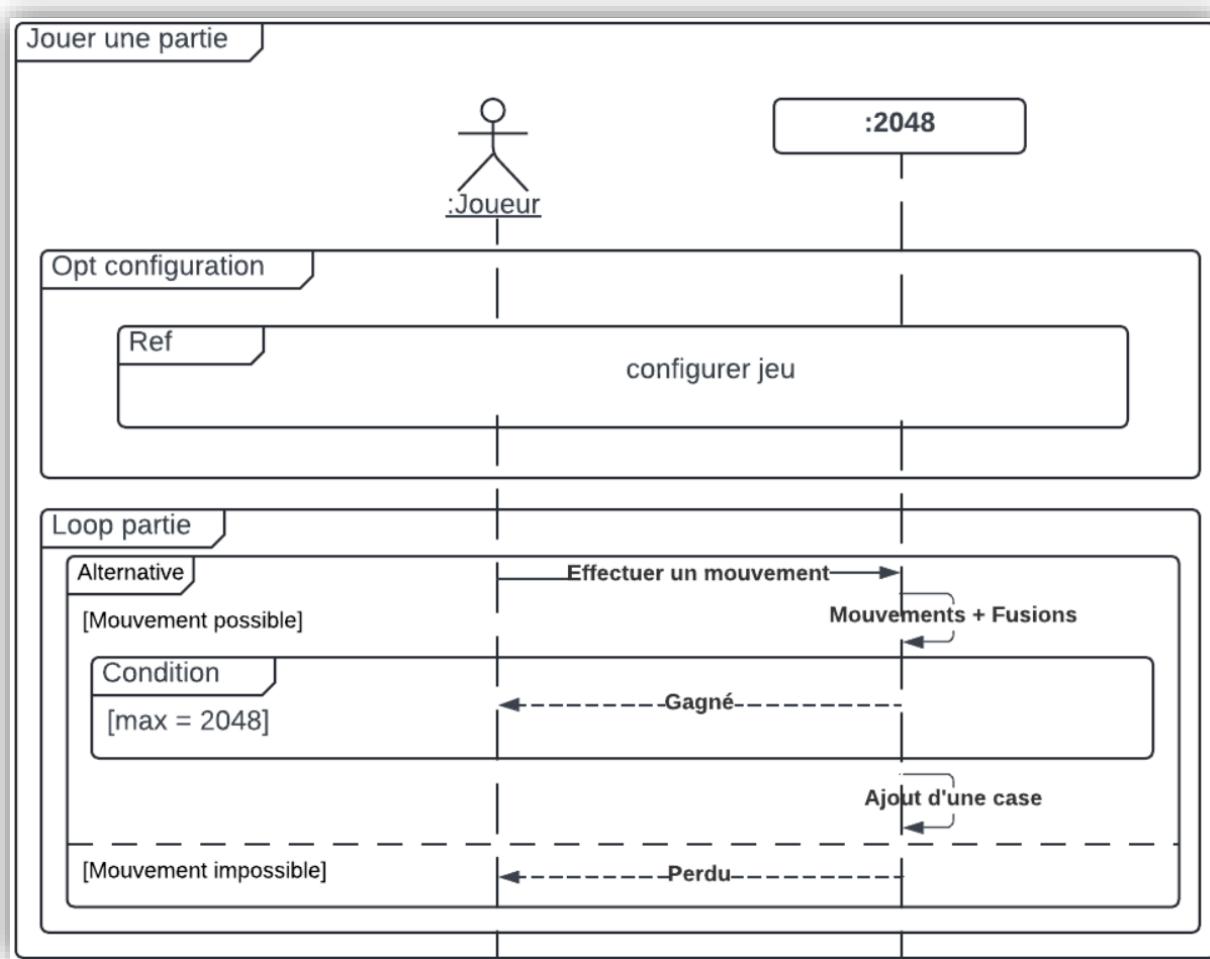


FIGURE 2 - DIAGRAMME DE SEQUENCE (HAUT NIVEAU)

Le diagramme de séquence du jeu 2048 (**cf. figure 2**) met en lumière les différentes interactions entre le joueur et le système, en détaillant le déroulement logique d'une partie. L'acteur principal de ce diagramme est le joueur, qui interagit avec l'objet 2048, représentant le système du jeu.

Avant de débuter une partie, une étape optionnelle permet au joueur de **configurer le jeu**. Cette action est représentée par un bloc optionnel (Opt), indiquant que la configuration des paramètres (taille de la grille, thème graphique, niveau de difficulté) peut être réalisée avant de commencer.

Une fois la partie lancée, une **boucle principale** (Loop) prend en charge la logique du jeu. Chaque tour de jeu, le joueur effectue un mouvement en appuyant sur une touche directionnelle (ou via un geste tactile), déclenchant ainsi le déplacement des tuiles. Si deux tuiles identiques se rencontrent, elles fusionnent et leur valeur double. Une nouvelle tuile apparaît après chaque mouvement, augmentant ainsi le défi.

Le diagramme illustre également la **gestion des conditions de victoire et de défaite** à travers une alternative (Alternative). Si un mouvement est encore possible, le système vérifie si une tuile 2048 a été atteinte. Si oui, le joueur est déclaré vainqueur. Dans le cas contraire, une nouvelle tuile est ajoutée et le jeu continue. En revanche, si aucun mouvement valide ne peut être effectué, la partie est perdue et un message de fin de jeu est affiché.

Ce diagramme de séquence met en évidence l'enchaînement structuré des interactions entre le joueur et le système, illustrant de manière claire le fonctionnement interne du jeu 2048.

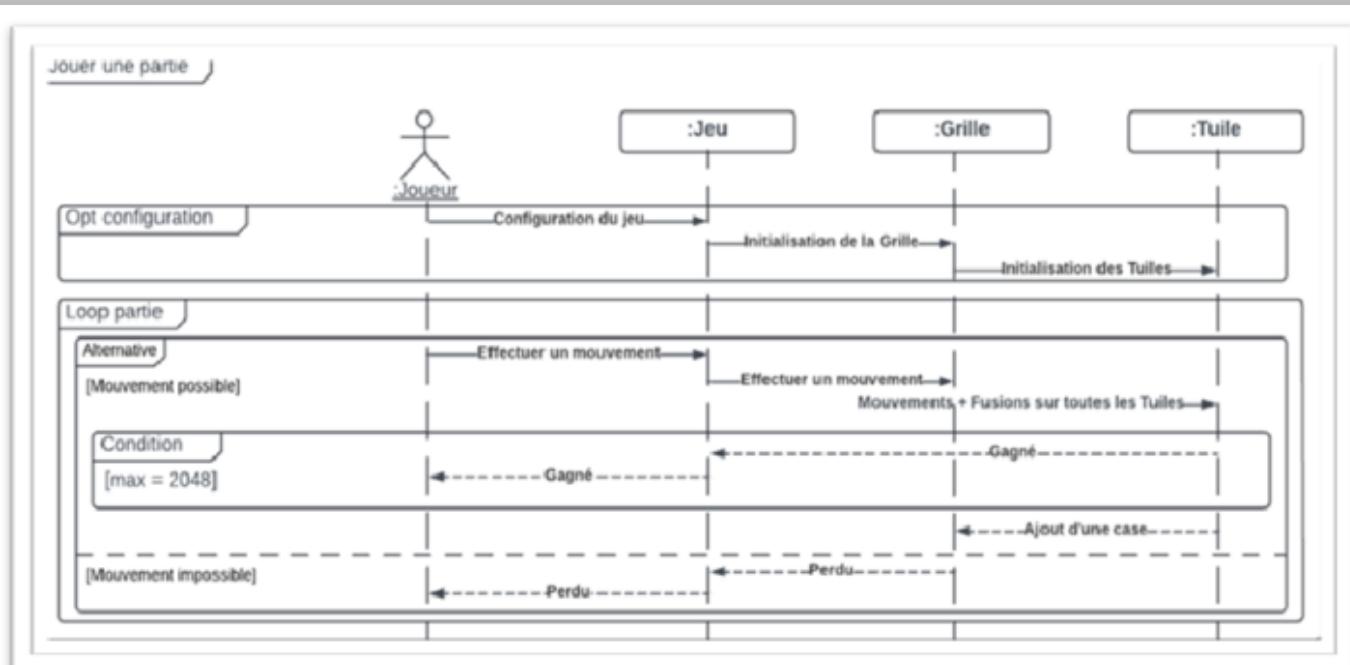


FIGURE 3 - DIAGRAMME DE SÉQUENCE (BAS NIVEAU)

Le diagramme de séquence de bas niveau (cf. figure 3) du jeu 2048 représente avec plus de précision les échanges entre les différentes composantes internes du système lors du déroulement d'une partie. Contrairement au diagramme de haut niveau, qui se concentre principalement sur les interactions générales entre le joueur et le système, ce diagramme détaille la communication entre les objets internes du jeu, notamment **le joueur, le jeu, la grille et les tuiles**.

Au début de la partie, le joueur envoie une commande pour effectuer un mouvement en utilisant les touches directionnelles **z**, **q**, **s**, **d** ou les flèches. Cette action est transmise à l'objet **Jeu**, qui sert d'orchestre et transmet la requête à la **Grille**. La grille, élément clé du système, traite alors la demande en identifiant les tuiles concernées et en appliquant le déplacement. Si deux tuiles adjacentes ont la même valeur, elles fusionnent pour former une nouvelle tuile dont la valeur est doublée. Ensuite, une nouvelle tuile est générée aléatoirement à un emplacement libre, rendant chaque mouvement imprévisible et ajoutant un aspect stratégique à la gestion de l'espace disponible.

Le diagramme illustre également la vérification des conditions de jeu après chaque action. Une séquence conditionnelle est déclenchée pour analyser si une tuile **2048** a été créée, auquel cas la partie est gagnée. À l'inverse, si aucun mouvement valide n'est possible, une autre séquence est activée pour signaler la fin de la partie. Cette vérification passe par une analyse approfondie de la grille, où le système évalue la possibilité d'effectuer un déplacement ou une fusion.

Ce diagramme de séquence de bas niveau permet ainsi de visualiser précisément comment les différentes entités du jeu interagissent entre elles et comment le système traite chaque action du joueur. Il met en évidence le rôle central de la grille dans la gestion du jeu et l'importance des conditions de victoire et de défaite dans l'expérience utilisateur. En mettant en lumière ces interactions détaillées, ce diagramme constitue un support essentiel pour la conception et l'amélioration du jeu, notamment en optimisant l'algorithme de déplacement et de fusion des tuiles.

2.3. Diagramme de Classes

Le modèle du jeu 2048 repose sur une structure en plusieurs classes (cf. figure 4), où la classe **Partie** joue un rôle central en orchestrant l'ensemble du système. Elle est responsable de la gestion du **Plateau**, qui représente la grille de jeu et contrôle l'évolution des **Tuiles**. La relation entre **Partie** et **Plateau** possède une relation de **composition**, signifiant que le Plateau est une partie essentielle de chaque instance de Partie et ne peut exister indépendamment.

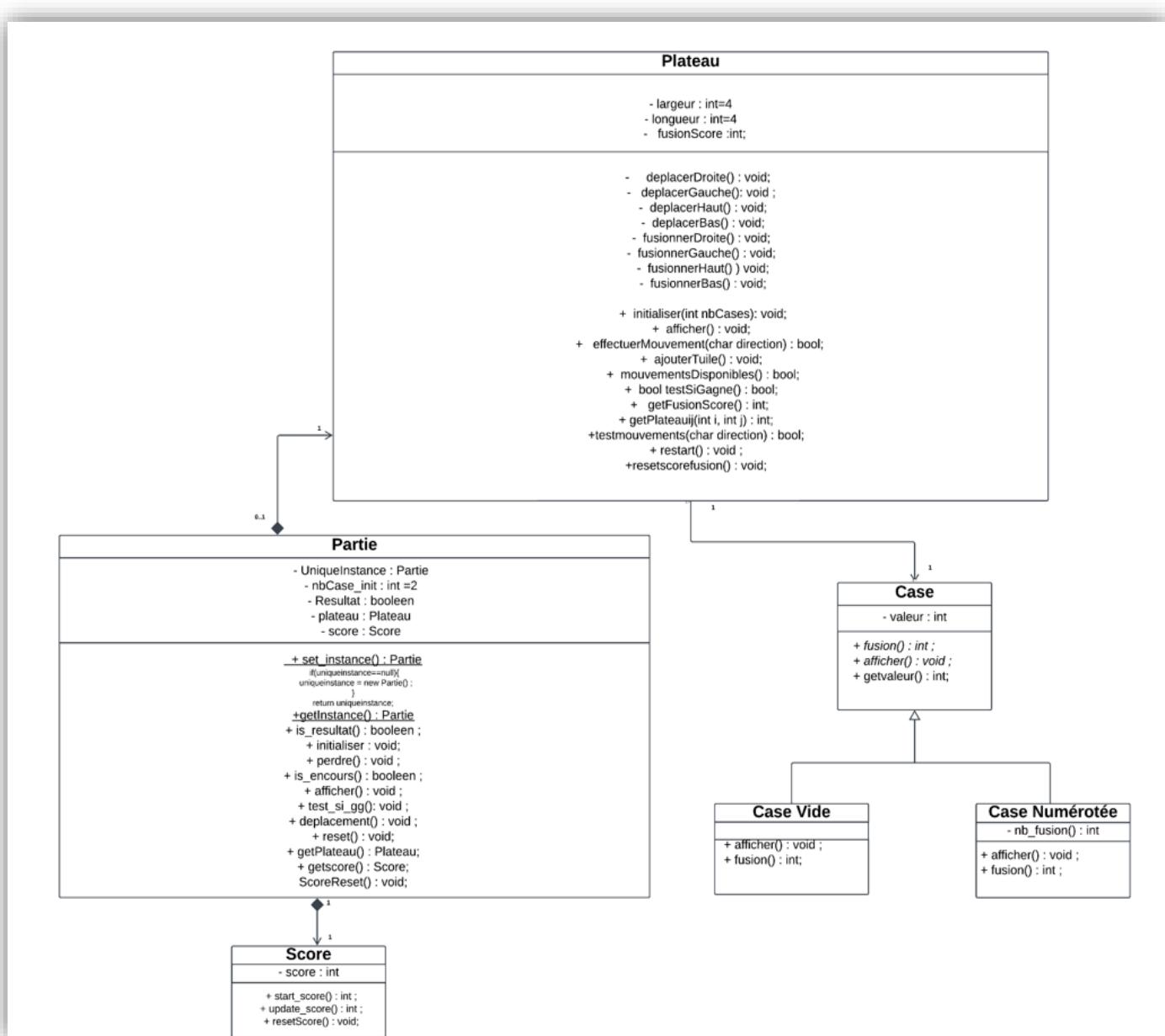


FIGURE 4 - DIAGRAMME DE CLASSES TECHNIQUE

Le **Plateau**, quant à lui, agit comme un gestionnaire des tuiles, régissant leurs déplacements et leurs fusions selon les actions du joueur. Il assure également la gestion des nouvelles tuiles qui apparaissent après chaque mouvement et vérifie si des actions restent possibles. Les **Tuiles**, qui peuvent être numérotées ou vides, sont intégrées au Plateau via une relation d'**agrégation**, ce qui signifie que les tuiles appartiennent au Plateau, mais peuvent être manipulées et remplacées sans que la suppression du Plateau entraîne leur destruction immédiate. De plus, elles suivent un modèle d'**héritage**, où les **Tuiles Numérotées** héritent des propriétés générales des **Tuiles**, leur permettant de se fusionner et d'afficher leur valeur.

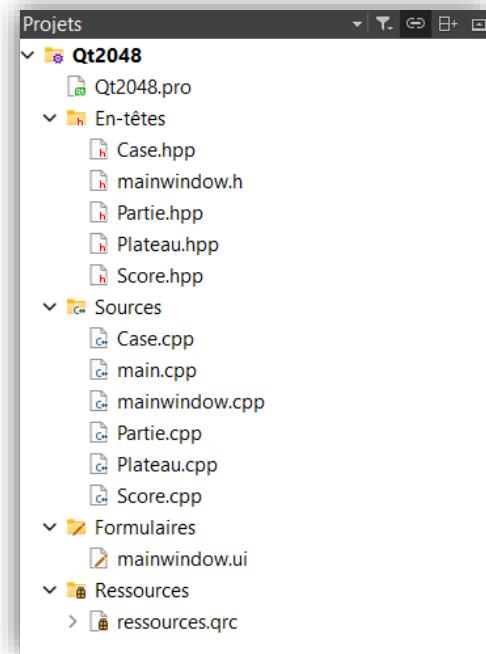
En parallèle, la **classe Score** est directement reliée à la Partie, formant une autre composition, car chaque partie possède son propre score, mis à jour à chaque fusion de tuiles. Cette mise à jour s'effectue de manière indirecte via le Plateau, qui transmet les points gagnés à la Partie.

L'ensemble de ces relations assure une architecture modulaire et cohérente, où chaque classe joue un rôle bien défini. La Partie centralise la gestion du jeu, le Plateau contrôle l'évolution des tuiles via une relation d'agrégation, et le Score suit la performance du joueur. Cette organisation permet une séparation claire des responsabilités, facilitant ainsi l'extension et la maintenance du jeu.

3. Programmation du logiciel

3.1. L'architecture du logiciel

L'arborescence des fichiers (cf. figure 5) du projet Qt2048 est structurée comme suit :



```

Qt2048/
├── Qt2048.pro
├── En-têtes/
│   ├── Case.hpp
│   ├──mainwindow.h
│   ├── Partie.hpp
│   ├── Plateau.hpp
│   └── Score.hpp
└── Sources/
    ├── Case.cpp
    ├── main.cpp
    ├──mainwindow.cpp
    ├── Partie.cpp
    ├── Plateau.cpp
    └── Score.cpp
└── Formulaires/
    └── mainwindow.ui
└── Ressources/
    └── ressources.qrc

```

FIGURE 5 - ARBORESCENCE DES FICHIERS

Le projet Qt2048 est une implémentation du célèbre jeu 2048 en utilisant le **framework Qt**. Le projet est structuré de manière modulaire, facilitant ainsi la séparation de l'interface utilisateur, de la logique du jeu et de la gestion des ressources. Cette organisation améliore la lisibilité du code et la maintenabilité du projet. Les fichiers sont organisés en plusieurs catégories : fichiers de configuration, en-têtes, sources, formulaires d'interface utilisateur et ressources.

Le fichier **Qt2048.pro** est un fichier de configuration essentiel au projet. Il définit les bibliothèques nécessaires, les fichiers sources et en-têtes à inclure, ainsi que les ressources à utiliser pour la compilation du projet. Ce fichier gère aussi les options de compilation et les dépendances.

Le dossier **En-têtes/** contient les fichiers d'en-têtes qui définissent les structures de données et les interfaces des classes utilisées dans le jeu sans en fournir l'implémentation. Par exemple, **Case.hpp** déclare la classe Case, qui représente une case dans la grille du jeu. Elle contient les

attributs nécessaires pour gérer la valeur de chaque case ainsi que des méthodes pour gérer les fusions de cases. **mainwindow.h** déclare la classe **MainWindow**, qui gère l'interface graphique du jeu, en incluant des éléments comme les labels pour afficher le score et la grille de jeu. La classe Partie est déclarée dans **Partie.hpp** et gère le déroulement du jeu, en définissant les règles, l'ajout de nouvelles cases, et la vérification des conditions de victoire ou de défaite. **Plateau.hpp** définit la classe Plateau, qui représente la grille 4x4 du jeu et contient les méthodes permettant de déplacer et de fusionner les cases. Enfin, **Score.hpp** est responsable de la gestion du score, en déclarant des méthodes pour calculer et afficher le score actuel du joueur ainsi que le meilleur score.

Le dossier **Sources/** contient les fichiers sources qui implémentent les classes et les fonctionnalités déclarées dans les fichiers d'en-têtes. Par exemple, **Case.cpp** contient l'implémentation de la classe Case, où sont définies des méthodes pour initialiser une case, changer sa valeur, et effectuer des fusions entre deux cases. **main.cpp** est le point d'entrée du programme, il initialise l'application Qt et lance la fenêtre principale du jeu. **mainwindow.cpp** implémente la classe **MainWindow**, qui gère l'affichage de l'interface utilisateur et les interactions de l'utilisateur avec le jeu, comme la capture des entrées clavier pour déplacer les cases. La classe Partie est implémentée dans **Partie.cpp**, où sont gérées les règles du jeu, l'ajout de nouvelles cases après chaque tour, et la détection des conditions de victoire ou de défaite. **Plateau.cpp** contient l'implémentation de la grille du jeu, avec des méthodes pour effectuer des mouvements et des fusions de cases, en fonction des actions de l'utilisateur. Enfin, **Score.cpp** implémente la gestion du score du joueur, en mettant à jour et en affichant le score et le meilleur score après chaque mouvement.

Le dossier **Formulaires/** contient les fichiers d'interface utilisateur générés par Qt Designer. **mainwindow.ui** est un fichier XML qui définit la structure de l'interface graphique de la fenêtre principale. Il inclut des composants comme la grille 4x4 pour afficher les cases, un label pour afficher le score et le meilleur score, ainsi qu'un bouton permettant de réinitialiser la partie.

Enfin, le dossier **Ressources/** contient les fichiers nécessaires aux ressources graphiques utilisées dans le projet. En effet, **ressources.qrc** est un fichier qui regroupe et référence les ressources comme les images et icônes utilisées dans l'interface graphique du jeu en particulier ici dans le projet la une image pour le bouton reset.

3.2. Points importants du code

- Fusion

```
// Fusionne les cases identiques vers le bas
void Plateau::fusionnerBas() {
    for (int j = 0; j < 4; ++j) {
        for (int i = 3; i > 0; --i) { // Parcourt de bas en haut
            if (grille[i][j] == grille[i - 1][j] && grille[i][j] != 0) {
                grille[i][j] *= 2; // Fusionne les valeurs
                fusionScore = grille[i][j]; // Ajoute au score de fusion
                grille[i - 1][j] = 0; // Vide la case fusionnée
            }
        }
    }
}

// Fusionne les cases identiques vers la droite
void Plateau::fusionnerDroite() {
    for (int i = 0; i < 4; ++i) {
        for (int j = 3; j > 0; --j) {
            if (grille[i][j] == grille[i][j - 1] && grille[i][j] != 0) {
                grille[i][j] *= 2;
                fusionScore = grille[i][j];
                grille[i][j - 1] = 0;
            }
        }
    }
}
```

FIGURE 6 - FUSION

Les quatre fonctions décrites dans le code, `fusionnerBas()`, `fusionnerDroite()`, `fusionnerGauche()`, et `fusionnerHaut()` (**cf. figure 6**), sont responsables de la fusion des cases identiques dans le jeu, chacune pour une direction spécifique (bas, droite, gauche, haut). Elles sont utilisées pour réaliser les mouvements et les fusions des tuiles dans le jeu 2048.

La fonction `fusionnerBas()` parcourt la grille de bas en haut (en commençant par la dernière ligne de chaque colonne) et vérifie si deux cases consécutives dans une colonne ont la même valeur. Si c'est le cas, la valeur de la case inférieure est doublée, et la case supérieure est mise à zéro. Ce processus fusionne les valeurs des cases et met à jour le score de fusion (`fusionScore`) avec la nouvelle valeur de la case fusionnée. Cette logique est implémentée avec une boucle imbriquée, où la première boucle parcourt chaque colonne et la deuxième boucle parcourt les lignes de bas en haut.

De manière similaire, la fonction `fusionnerDroite()` effectue la fusion des cases vers la droite (**cf. figure 7**). Elle parcourt chaque ligne de droite à gauche et vérifie si deux cases adjacentes dans la même ligne ont la même valeur. Si c'est le cas, les valeurs des cases sont fusionnées (la valeur de la case de droite est doublée) et la case à gauche est mise à zéro. Comme dans la fonction précédente, le score de fusion est mis à jour.

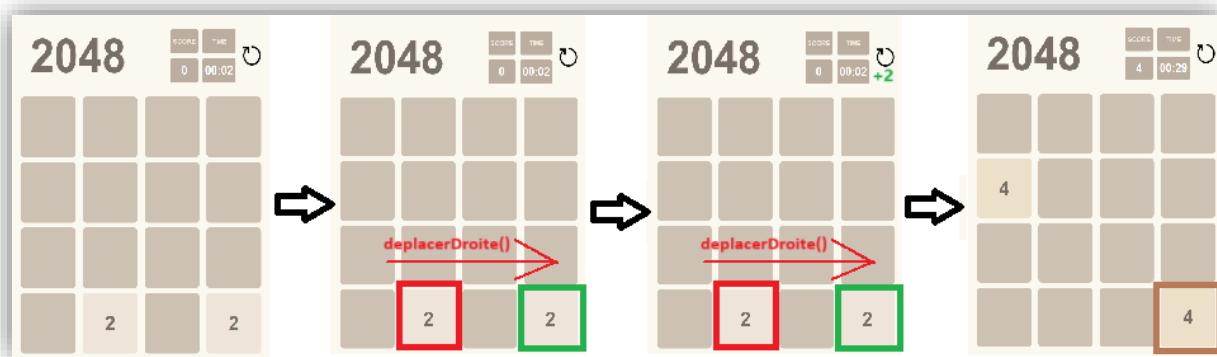


FIGURE 7 - ILLUSTRATION FONCTION FUSIONNERDROITE ()

Ces fonctions sont cruciales pour le bien être du jeu, car elles permettent d'effectuer les fusions nécessaires pour doubler les valeurs des tuiles et de faire progresser le jeu. Chacune de ces fonctions utilise une logique similaire mais adaptée à la direction du mouvement, ce qui permet de maintenir la cohérence du jeu tout en offrant une grande flexibilité dans les interactions avec la grille.

- **Structure de donnée**

Le projet repose sur une interface graphique construite avec Qt, où la fenêtre principale (MainWindow) gère l'affichage et les interactions du jeu 2048. L'élément central de cette interface est une **grille de 4x4 cases**, représentée par un **tableau de QLabel * labels [4][4]**. Chaque case correspond à une tuile du jeu et peut contenir un nombre ou être vide. Ces QLabel (cf. **figure 8**) sont disposés à l'écran à l'aide d'un QGridLayout, qui facilite l'organisation en tableau et permet une mise à jour dynamique lors des déplacements du joueur.

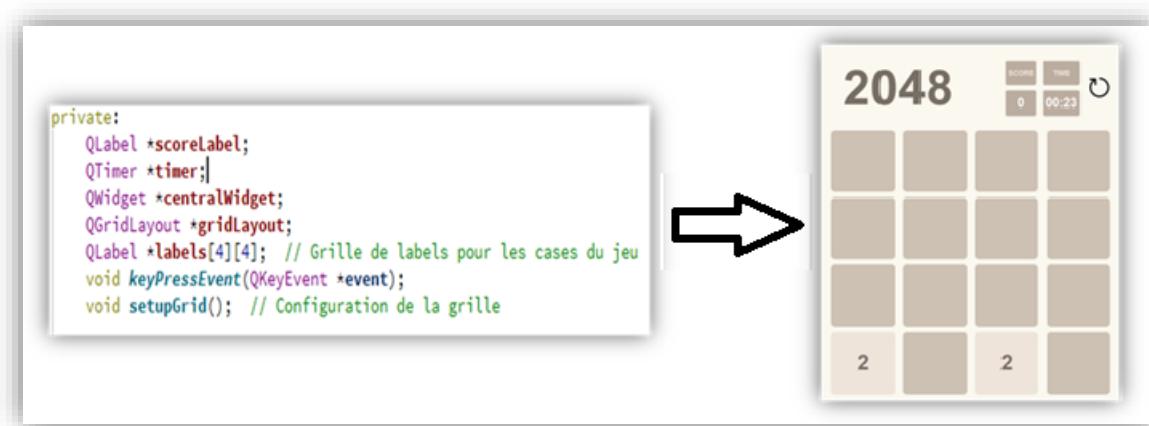


FIGURE 8 - STRUCTURE DE DONNEE

- **Interactions Utilisateur**

L'utilisateur interagit avec le jeu à l'aide des touches directionnelles du clavier, ce qui est capturé par la méthode keyPressEvent (QKeyEvent *event) (**cf. figure 9**). Selon la touche pressée, un mouvement est déclenché, déplaçant les cases dans la direction correspondante. Une fois le mouvement effectué, l'affichage est mis à jour pour refléter les nouvelles positions des cases.

```
void MainWindow::keyPressEvent(QKeyEvent *event) {
    // Gérer les touches du clavier
    modifAffichage();
    char direction;
    switch (event->key()) {
    case Qt::Key_Z :
        direction='z';
        break;
    case Qt::Key_S :
        direction='s';
        break;
    case Qt::Key_Q :
        direction='q';
        break;
    case Qt::Key_D :
        direction='d';
        break;
    default:
        QMainWindow::keyPressEvent(event); // Gestion des touches par défaut
        return;
    }
}
```

FIGURE 9 - INTERACTIONS UTILISATEUR

- **Timer**

Le **temps** dans l'interface est également affiché à l'aide d'un QLabel, appelé timeLabel, qui indique le temps écoulé depuis le début de la partie. Ce label est placé dans un conteneur QVBoxLayout (timeBoxLayout) (**cf. figure 10**), tout comme le score, et est ajouté à la barre d'en-tête de l'interface avec headerLayout->addLayout(timeBoxLayout). Le style du timeLabel est similaire à celui du score, avec une couleur de fond, une police en gras et un texte centré pour rendre le temps bien visible.

```
// Time box
QVBoxLayout *timeBoxLayout = new QVBoxLayout();
QLabel *timeTitle = new QLabel("TIME", this);
timeTitle->setStyleSheet(
    "font: bold 14px Arial; "
    "color: #EEE4DA; "
    "background-color: #BBADA0; "
    "border-radius: 4px; "
    "padding: 5px; "
    "text-align: center; "
);
timeTitle->setAlignment(Qt::AlignCenter);

QLabel *timeLabel = new QLabel("00:00", this);
```

FIGURE 10 – TIMER BOX

La gestion du temps repose sur un QTimer (cf. figure 11), qui déclenche un événement toutes les secondes. À chaque "timeout" du timer, une fonction de mise à jour formate et affiche le temps écoulé sous la forme "mm:ss". Ce mécanisme permet de suivre en temps réel le temps passé pendant la partie. Si l'utilisateur réinitialise le jeu, le timer est arrêté et redémarré à zéro, assurant ainsi une mise à jour continue du temps pendant chaque session de jeu.



FIGURE 11 - QTIMER

- **Reset Button**

Le bouton de réinitialisation (cf. figure 12) permet de recommencer une partie à tout moment. Lorsqu'il est pressé, la grille est vidée, le score est remis à zéro, et le chronomètre repart du début. Cette fonctionnalité est directement liée à la fonction rejouer (), qui s'assure que l'affichage est bien mis à jour pour refléter le nouvel état du jeu.

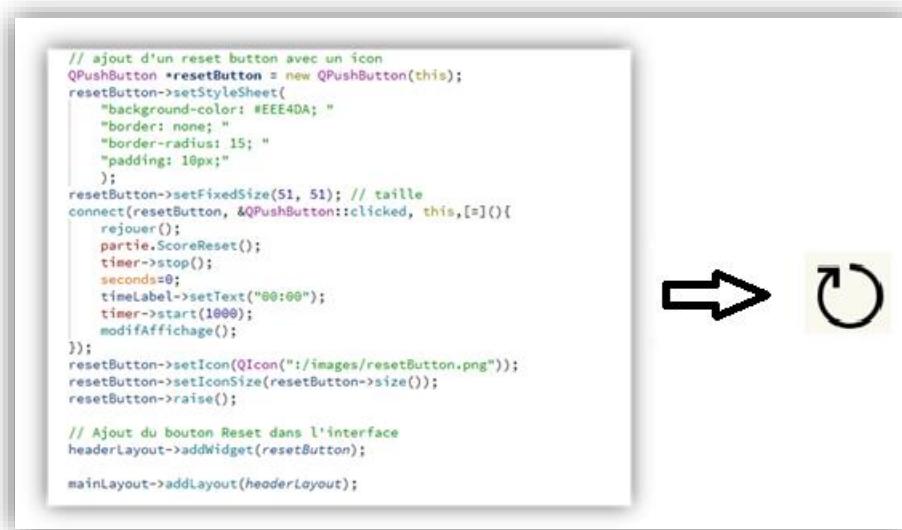


Figure 12 - BOUTON RESET

Le bouton de réinitialisation (**resetButton**) est un élément essentiel de l'interface permettant au joueur de recommencer une partie. Il est créé à l'aide de la classe QPushButton, avec un style personnalisé incluant une couleur de fond, des coins arrondis et une icône représentant une action de redémarrage.

Ce bouton est placé dans la barre supérieure du jeu et est ajouté à la mise en page avec headerLayout->addWidget(resetButton). Lorsqu'un utilisateur clique dessus, un gestionnaire d'événements associé via connect() exécute plusieurs actions : il relance une nouvelle partie avec rejouer(), remet le score à zéro, arrête et redémarre le chronomètre, met à jour l'affichage du temps et rafraîchit la grille de jeu. Grâce à cette structure, le bouton offre une expérience fluide, permettant de recommencer instantanément sans avoir à relancer l'application.

- **Score**

La gestion du **score** dans l'interface se fait à travers un **QVBoxLayout** appelé **scoreBoxLayout** (cf. figure 13), qui contient deux éléments principaux : un **QLabel** pour afficher le titre "SCORE" et un autre **QLabel** pour afficher la valeur du score. Ces deux éléments sont ajoutés à **scoreBoxLayout**, qui est ensuite intégré à l'en-tête de. Cette structure permet une présentation claire et attrayante du score en haut de la fenêtre du jeu.



FIGURE 13 - SCORE

En résumé, la structure du projet est pensée pour séparer clairement l'affichage des cases, la gestion des interactions et la mise à jour dynamique de l'interface, permettant ainsi une expérience fluide et intuitive pour l'utilisateur.