Documentation technique

I. Introduction

A. Présentation du jeu

1. Le jeu **2048** est un jeu de puzzle basé sur une grille, où le joueur doit combiner des blocs de valeurs identiques pour atteindre la valeur cible de 2048 (ou plus dans les modes infini).
2. Chaque mouvement fait glisser toutes les tuiles de la grille dans la direction choisie, fusionnant les tuiles de même valeur et générant une nouvelle tuile (2 ou 4) de manière aléatoire.
3. Le jeu se termine lorsque le joueur atteint la valeur cible 2048 (victoire en mode classique) ou lorsqu'il n'y a plus de mouvements possibles (défaite).

B. Objectifs de la documentation technique

1. L’objectif principal de cette documentation technique est de fournir une vue détaillée du fonctionnement interne du jeu, des choix de conception, et des méthodologies utilisées. Cela inclut :
    a. Décrire les composants et leur interaction dans le système.
    b. Documenter les algorithmes et les mécanismes clés (fusion, déplacement, génération de tuiles).
    c. Fournir des indications pour la maintenance et les futures évolutions.
    d. Offrir une base de référence pour les développeurs et contributeurs du projet.

II. Architecture du jeu

A. Moteur du jeu

1. **Gestion des mouvements** : Déplacement des tuiles en réponse aux entrées utilisateur (flèches, gestes).

```
function MoveCells(event : KeyboardEvent) {
    const gridCopy = JSON.stringify(grid.value);

    switch (event.key) {
        case 'ArrowUp':
            MoveCellsUp();
            break;
        case 'ArrowDown':
            MoveCellsDown();
            break;
        case 'ArrowLeft':
            MoveCellsLeft();
            break;
        case 'ArrowRight':
            MoveCellsRight();
            break;
    }

    ResetEmptyTiles();

    // Spawn a new cell only if the grid changed
    if (gridCopy !== JSON.stringify(grid.value)) {
        SpawnRandomCell();
    }
}
```

```
// Moves tiles up
function MoveCellsUp() {
    for (let col = 0; col < gridSize; col++) {
        for (let row = 1; row < gridSize; row++) {
            if (grid.value[row][col] !== -1) {
                let target = row;
                while (target > 0 && grid.value[target - 1][col] === -1) {
                    grid.value[target - 1][col] = grid.value[target][col];
                    grid.value[target][col] = -1;
                    target--;
                }

                if (
                    target > 0 &&
                    grid.value[target - 1][col] === grid.value[target][col]
                ) {
                    score.value += grid.value[target - 1][col] * 2;
                    grid.value[target - 1][col] *= 2;
                    grid.value[target][col] = -1;
                }
            }
        }
    }
}
```

La première fonction MoveCells permet de reconnaître la touche cliqué et appelle la fonction correspondante tell que la fonction MoveCellUp

2. **Fusion des tuiles** : Détection des collisions entre tuiles et combinaison de leurs valeurs.
3. **Génération aléatoire** : Création de nouvelles tuiles (valeurs 2 ou 4) dans des emplacements disponibles.

Cette fonction s'occupe de la création de nouvelle tuile avec 90% de chance que une tuile avec le nombre 2 apparaissent.

```
// Spawns a random tile
function SpawnRandomCell() {
    if (isGridFull()) return;
    let emptyCells: any[] = [];

    grid.value.forEach((row, rowIndex) => {
        row.forEach((cell, colIndex) => {
            if (cell === -1) {
                emptyCells.push({ rowIndex, colIndex });
            }
        });
    });

    const randomCell =
        emptyCells[Math.floor(Math.random() * emptyCells.length)];
    grid.value[randomCell.rowIndex][randomCell.colIndex] =
        Math.random() < 0.9 ? 2 : 4;
}
```

4. **État de la partie** : Détection des conditions de victoire (atteinte de 2048) ou de défaite (aucun mouvement possible).

B. Système de rendu

1. **Affichage des tuiles** : Placement dynamique sur la grille en fonction des positions de la matrice.

```
<div class="grid-container u-flex u-flex-direction-column u-p15 u-gap15">
    <div class="u-flex u-gap15" v-for="rows in grid">
        <div class="grid-cell" v-for="cell in rows"></div>
    </div>
    <div class="tile-container">
        <div v-for="(tile, index) in grid.flat()" :key="index">
            <div :class="`tile-position-${Math.floor(index / gridSize )}-${index % gridSize}`">
                <div v-if="tile > -1" :class="`tile tile-color-${tile}`">{{ tile }}</div>
            </div>
        </div>
    </div>
</div>
```

Code en html qui affiche les tuiles sur le plateau en faisant attention aux bordures.

2. **Animations** : Mouvements fluides des tuiles et transitions lors de la fusion.
3. **Mises à jour en temps réel** : Synchro entre les changements d'état dans le moteur de jeu et leur affichage.

C. Gestion des ressources (graphisme, sons, etc...)

1. Graphisme
    a. **Tuiles dynamiques** : Générées avec des couleurs et des tailles en fonction des valeurs.
    b. **Thèmes** : Prise en charge de modes visuels (ex. : clair/sombre).
2. Sons
    a. Sons pour les mouvements et la fusion.
    b. Musique de fond ou effets sonores personnalisables.

III. Conception du gameplay

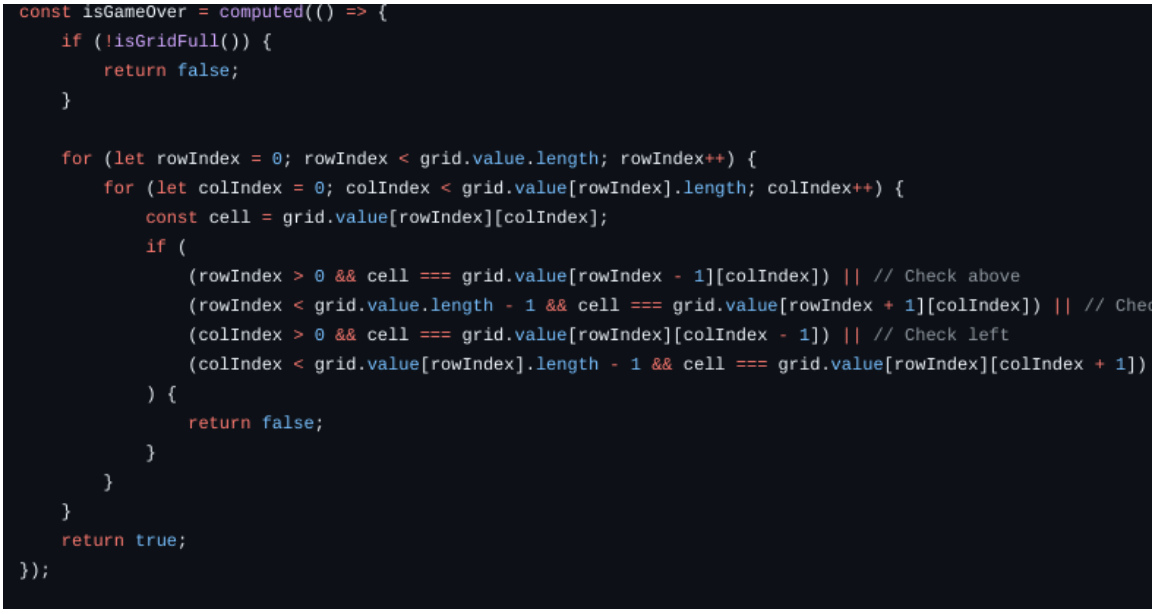
A. Mécanique du jeu

1. Déplacement des blocs
    a. Les blocs se déplacent dans une direction (haut, bas, gauche, droite) lorsque l'utilisateur effectue un geste ou appuie sur une touche.
    b. Les blocs glissent jusqu'à rencontrer une autre tuile ou le bord de la grille.
2. Fusion des blocs
    a. Deux blocs avec la même valeur se fusionnent lorsqu’ils se rencontrent après un déplacement.
    b. La fusion crée un nouveau bloc dont la valeur est la somme des deux blocs originaux.
    c. Un seul mouvement peut provoquer plusieurs fusions successives, mais une tuile ne peut fusionner qu'une seule fois par mouvement.
3. Génération aléatoire de blocs
    a. À chaque tour, un nouveau bloc apparaît dans une case vide. Probabilité de génération :
        i. 90% pour un bloc de valeur 2.
        ii. 10% pour un bloc de valeur 4.
4. Conditions de victoire et défaite

a. Victoire : un joueur atteint la tuile de valeur **2048** (dans le mode classique).

b. Défaite : aucun mouvement possible, toutes les cases de la grille sont remplies et aucune fusion n'est réalisable.

c. Cette fonction vérifie si la grille est remplie, si oui et que aucun coup n’est possible alors on affiche la page défaite sinon on continue le jeux.



```
const isGameOver = computed(() => {
    if (!isGridFull()) {
        return false;
    }

    for (let rowIndex = 0; rowIndex < grid.value.length; rowIndex++) {
        for (let colIndex = 0; colIndex < grid.value[rowIndex].length; colIndex++) {
            const cell = grid.value[rowIndex][colIndex];
            if (
                (rowIndex > 0 && cell === grid.value[rowIndex - 1][colIndex]) || // Check above
                (rowIndex < grid.value.length - 1 && cell === grid.value[rowIndex + 1][colIndex]) || // Chec
                (colIndex > 0 && cell === grid.value[rowIndex][colIndex - 1]) || // Check left
                (colIndex < grid.value[rowIndex].length - 1 && cell === grid.value[rowIndex][colIndex + 1])
            ) {
                return false;
            }
        }
    }
    return true;
});
```

B. Systèmes de contrôle

1. Contrôles sur navigateur web
    a. Flèches directionnelles ou touches WASD pour déplacer les blocs.
2. Tactile (mobile)
    a. Glissement du doigt (swipe) dans la direction souhaitée.

C. Interaction joueur-environnement

1. Grille de jeu
    a. Représentation visuelle intuitive avec des lignes délimitant les cases.
2. Blocs de jeu
    a. Chaque bloc a une couleur distincte en fonction de sa valeur.
    b. Les couleurs augmentent en intensité pour les valeurs plus élevées, facilitant la lecture visuelle.

3. Menu et barre de scores
    a. Le menu inclut les options de personnalisation, la possibilité de recommencer une partie ou de revenir à l'écran principal.
    b. La barre de scores affiche
        i. Le score actuel.
        ii. Le meilleur score de la session ou sauvegardé dans le stockage local du navigateur.

IV. Graphiques et Animation

A. Modélisation 3D

1. **Grille en 3D** : La grille de jeu peut être représentée avec une légère perspective 3D, donnant l'impression que les tuiles s'élèvent et se déplacent dans un espace tridimensionnel.

B. Textures et shaders

1. **Texturation des tuiles** :
    a. Chaque tuile aura une texture de fond distincte basée sur sa valeur, afin de permettre aux joueurs d'identifier rapidement la valeur d'une tuile.
    b. Les tuiles contenant des puissances de 2 (2, 4, 8, 16, etc.) pourraient avoir des dégradés de couleur allant du beige clair au rouge vif, avec des chiffres au centre de la tuile.

C. Animation des personnages et objets

1. **Shaders de fusion** : Lorsqu'une tuile fusionne avec une autre, un effet visuel dynamique pourrait être appliqué comme un éclair lumineux
2. Lorsque l'utilisateur déplace une tuile, elle devrait glisser dans la direction choisie (haut, bas, gauche, droite), avec une animation fluide.

V. Son et Musique

A. Effets sonores

1. **Mouvements des tuiles** : Un léger son de bruit de vent lorsque les tuiles se déplacent d'une position à une autre sur la grille.
2. **Victoire et défaite** : Des sons distincts pour signaler la fin de la partie, comme un son joyeux lors de la victoire (pour 2048) ou un son dramatique pour la défaite (lorsqu'il n'y a plus de mouvement possible).

B. Musique d'ambiance

1. **Musique calme et répétitive**

C. Intégration sonore dans le gameplay

1. Synchronisation des effets sonores et visuels
2. Volume et contrôles sonores
    a. Le volume des effets sonores et de la musique devrait être ajustable par l'utilisateur, avec une option dans les paramètres pour activer ou désactiver la musique et les effets.

VI. Optimisation et Performance

A. Gestion des ressources système

1. Utilisation de la mémoire
    a. **Grille dynamique** : La taille de la grille doit être adaptable, mais il est important d'éviter la surcharge mémoire.
    b. **Libération de la mémoire** : Après chaque action, il est important de libérer les ressources inutilisées afin de réduire la consommation mémoire.
    c. **Réduire les calculs répétitifs**

B. Optimisation du code

1. Gestion des événements
    a. Lors des déplacements du joueur, il est important de limiter la fréquence des événements, surtout en cas de déplacements rapides.
    b. Utiliser des événements simples et efficaces

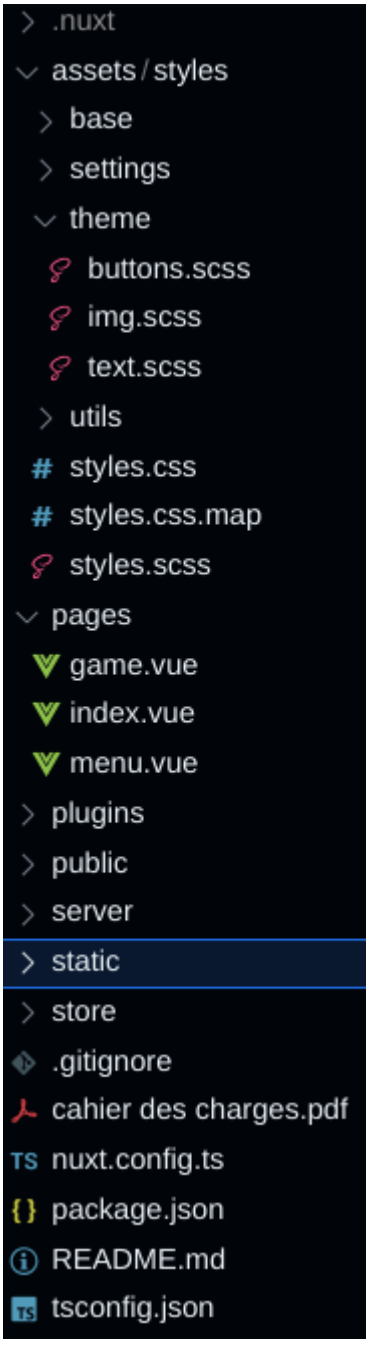
c. **Tri des tuiles** : Au lieu de vérifier chaque tuile indépendamment, il est possible de trier les tuiles de chaque ligne ou colonne avant de les déplacer ou de les fusionner, ce qui simplifie le code et améliore la performance.

C. Tests de performance

1. Tests de charge
    a. Effectuer des tests de charge pour vérifier comment le jeu réagit lorsqu'une grande quantité de tuiles est présente sur la grille
    b. Tester les déplacements et les fusions avec une grande quantité de blocs pour s'assurer que le jeu reste fluide même avec des grilles plus grandes.
2. Tests de fluidité
    a. Tester les animations (comme les déplacements des tuiles) pour s'assurer qu'elles sont fluides, sans saccades ni retard.

VII. Documentation du Code

A. Structure du code source



B. Commentaires et documentation interne

1. **Style des commentaires**
    a. Utilisation de commentaires **uniquement là où nécessaire** pour éviter de surcharger le code.
    b. Séparer les commentaires explicatifs en haut des fonctions/classes et les commentaires détaillés dans les sections complexes du code.
    c. Nous utiliserons des outils tels que **JSDoc** pour générer automatiquement une documentation à partir des commentaires que

nous avons mis.

C. Bonnes pratiques de codage

1. Normes générales
    a. Nom des variables et fonctions
        i. Utiliser des noms explicites et descriptifs
    b. Découpage du code
        i. Éviter les fonctions trop longues (>20 lignes). Chaque fonction doit se concentrer sur **une seule responsabilité**.
    c. Gestion des erreurs
        i. Gérer les erreurs et les exceptions pour éviter les crashs imprévus

VIII. Déploiement et Maintenance

A. Configuration requise

1. Côté utilisateur
    a. Un navigateur web avec une prise en charge de javascript.
    b. Mémoire vive minimale : 2 Go.
2. Coté serveur
    a. Compatibilité avec node.js
    b. Prise en charge du HTTPS pour plus de sécurité.

B. Procédure d’installation

1. Installer node.js
2. Cloner le dépôt github disponible en open source
3. Installer toutes les dépendances.
4. Lancer le serveur de déploiement

C. Gestion des mises à jour et correctifs

1. Diviser les mises à jour en trois catégories : **correctifs**, **améliorations**, et **nouvelles fonctionnalités**.
2. Effectuer des tests unitaires et d'intégration sur toutes les nouvelles modifications.
3. Utiliser un environnement de test avant tout déploiement sur le serveur de production.
4. Veiller à ce que les nouvelles versions soient compatibles avec les anciennes données ou fonctionnalités.