RAPPORT DU JEUX



GROUP 1

PARTICIPANTS:

BENALLAL AMINE 232331666205 [benallalamine2209@gmail.com](mailto:benallalamine2209@gmail.com)

Noureddine Hadil Narimane 232331419815 [noureddinehadil4@gmail.com](mailto:noureddinehadil4@gmail.com)

bassaidouelhadj Rafik 232331530912 [rafikbassaidouelhadj@gmail.com](mailto:rafikbassaidouelhadj@gmail.com)

SENNOUN Mohamed 232331690602 [mohamed.sennoun20@gmail.com](mailto:mohamed.sennoun20@gmail.com)

HADIBI Roumaissa 232331574110 [roumaissa.hadibi.dev@gmail.com](mailto:roumaissa.hadibi.dev@gmail.com)

Rahim Abderrahmane 232331624613 [abderrahman.rahim18@gmail.com](mailto:abderrahman.rahim18@gmail.com)

ISMAIL Ziad 212132055823 [iziad0040@gmail.com](mailto:iziad0040@gmail.com)

SAADI Nassim 222233016014 honassim518@gmail.com

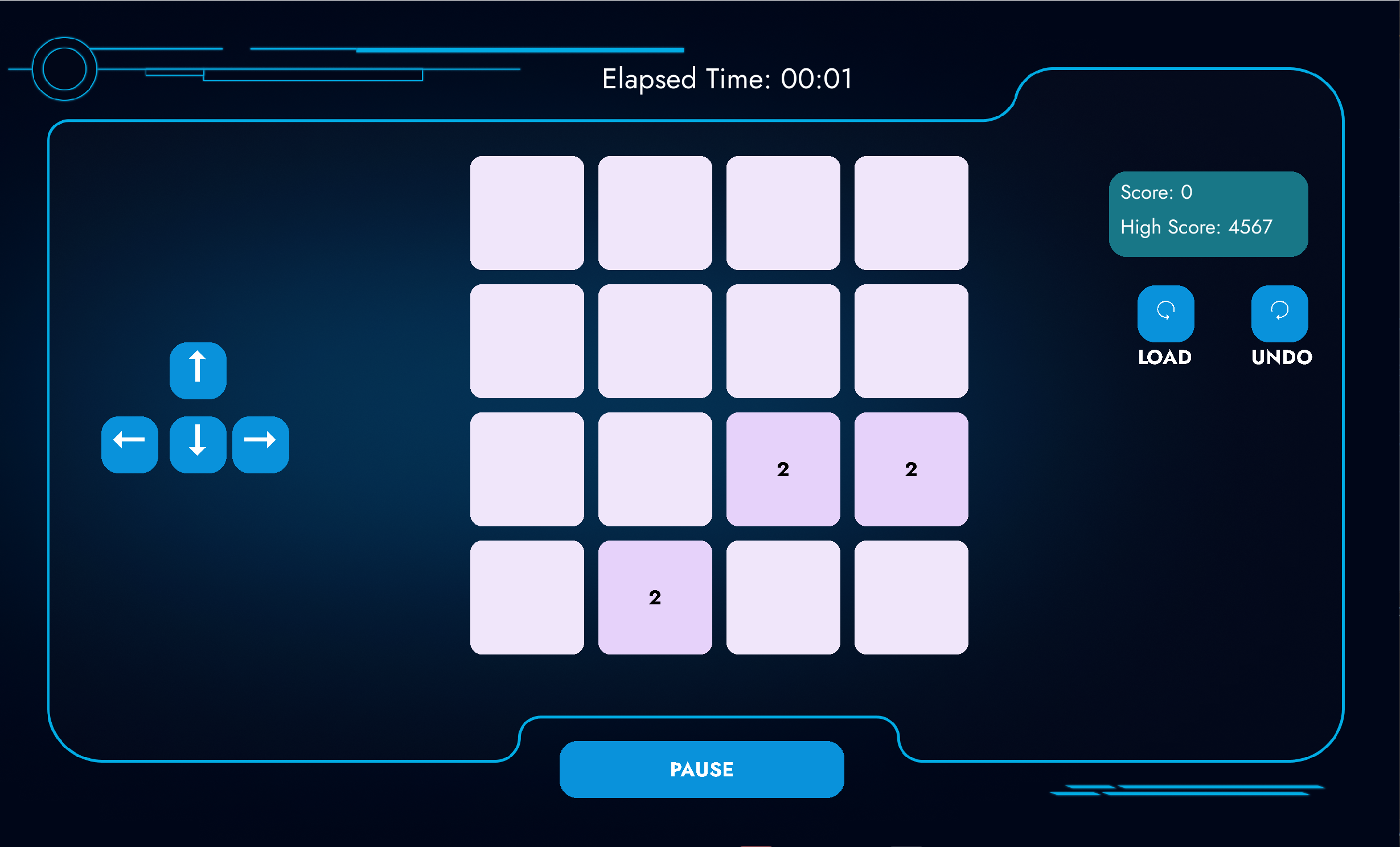
**Introduction**

Ce rapport présente une vue d'ensemble du développement et des fonctionnalités d'un jeu créé à l'aide de SDL2 (Simple DirectMedia Layer 2) en langage C. SDL2 est une bibliothèque puissante utilisée pour les applications multimédias, offrant un support pour le rendu graphique, l'audio et la gestion des entrées utilisateur sur plusieurs plateformes. Ce jeu est un projet unique qui illustre ma compréhension des concepts de programmation, notamment le rendu graphique, la gestion des entrées utilisateur et la logique du jeu.

Le jeu est conçu pour fonctionner efficacement, offrant une expérience utilisateur attrayante avec des éléments interactifs, des visuels dynamiques et des effets sonores. Il intègre plusieurs fonctions personnalisées, chacune traitant un aspect particulier des mécanismes ou de l'interface du jeu. Dans ce rapport, chaque fonction sera expliquée pour illustrer comment elle contribue au gameplay global et aux performances du système. Les explications visent à trouver un équilibre entre la profondeur technique et la clarté afin de garantir leur compréhension.

Les sections suivantes examineront en détail le fonctionnement des principales fonctionnalités du jeu, en commençant par la phase d'initialisation et en progressant à travers la logique du jeu, le rendu et des fonctionnalités supplémentaires telles que la gestion des scores et la sauvegarde des états du jeu. Cette approche structurée permettra de fournir une compréhension complète du fonctionnement du jeu et mettra en avant les stratégies de résolution de problèmes adoptées lors de son développement.

1. **Mode Player (Joueur Seul)**



**Fonctionnement Global**

Le mode Player permet à un joueur de contrôler entièrement la grille 4x4 en utilisant les touches directionnelles. Le but est de fusionner les tuiles pour atteindre la tuile 2048 ou obtenir le score le plus élevé possible avant que la grille ne soit remplie. Ce mode repose sur plusieurs fonctions clés pour gérer les mouvements, l'affichage, et la logique du jeu.

**Fonctions Utilisées**

1. **updateMatrix(int direction)**  
   Cette fonction est au cœur du mode Player. Elle met à jour la matrice en fonction de la direction choisie par le joueur. Les directions possibles sont : haut (0), bas (1), droite (2), et gauche (3). Lorsqu'une direction est donnée, la fonction appelle les sous-fonctions appropriées pour déplacer les tuiles dans cette direction. Par exemple, si le joueur choisit de déplacer les tuiles vers le haut, la fonction en\_haut est appelée. Après chaque mouvement, la fonction vérifie si la matrice a été modifiée en utilisant matrixchange. Si un changement est détecté, un nouveau nombre aléatoire est ajouté à la matrice via randomrvennumber. Cette fonction garantit que le jeu reste dynamique et réactif aux actions du joueur.
2. **printMatrix()**  
   Cette fonction est principalement utilisée pour le débogage. Elle affiche la matrice actuelle dans la console, ce qui permet aux développeurs de vérifier l'état du jeu à tout moment. Chaque élément de la matrice est imprimé avec un espacement de 4 caractères pour une meilleure lisibilité. Par exemple, une matrice avec quelques tuiles pourrait être affichée comme suit :

2 0 0 0

0 4 0 0

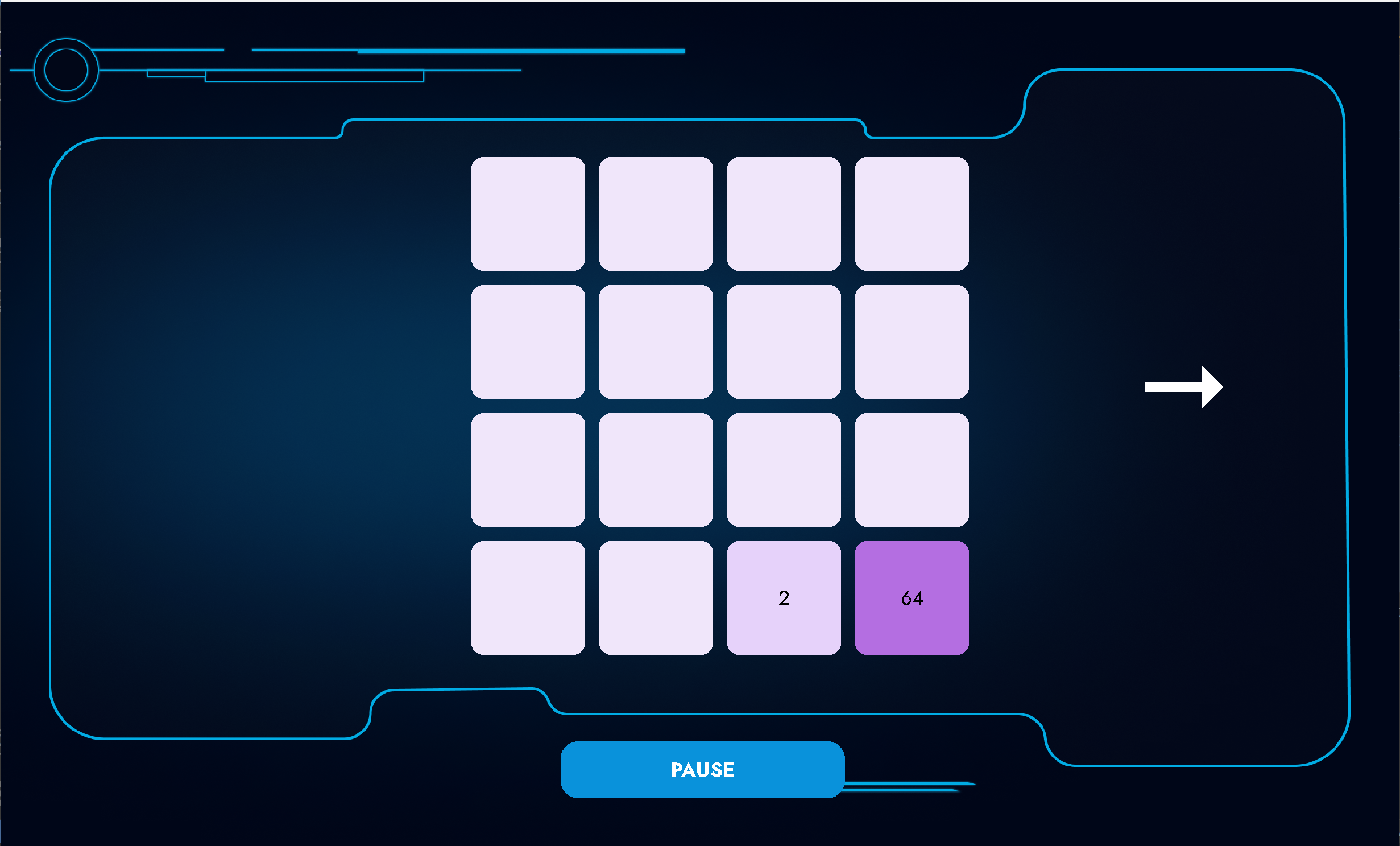
0 0 8 0

0 0 0 16

Cette fonction est utile pour suivre l'évolution de la matrice pendant le développement ou pour diagnostiquer des problèmes.

1. *renderMatrix(SDL\_Renderer, int matrix[4][4], TTF\_Font*, int windowWidth, int windowHeight, gc theme)\*\*  
   Cette fonction est responsable de l'affichage graphique de la matrice à l'écran. Elle utilise SDL2 pour dessiner chaque tuile avec des couleurs et des formes adaptées au thème choisi (par exemple, défaut, cyan, ou sombre). Les tuiles sont dessinées avec des coins arrondis grâce à la fonction fillRoundedRect, ce qui donne un aspect moderne et esthétique à l'interface. La fonction calcule également la position de chaque tuile pour centrer la grille à l'écran, en tenant compte des dimensions de la fenêtre. Cela garantit que l'affichage reste cohérent sur différents écrans et résolutions.
2. **matrixchange(int matrixcopy[4][4], int matrix[4][4])**  
   Cette fonction compare deux matrices pour déterminer si un mouvement a entraîné un changement. Elle est utilisée après chaque déplacement pour vérifier si la matrice actuelle diffère de la copie précédente. Si les matrices sont identiques, cela signifie que le mouvement n'a pas eu d'effet (par exemple, si le joueur essaie de déplacer les tuiles dans une direction où aucun mouvement n'est possible). Dans ce cas, aucun nouveau nombre n'est ajouté à la matrice. Si les matrices sont différentes, la fonction retourne true, indiquant qu'un changement a eu lieu, et un nouveau nombre est ajouté via randomrvennumber.
3. **initmatrix(int matrix[4][4])**  
   Cette fonction initialise la matrice au début du jeu. Elle remplit d'abord toutes les cases avec la valeur 0, puis place deux nombres aléatoires (généralement des 2) dans des cases vides. Si aucune case vide n'est disponible, la fonction affiche un message "Game Over", indiquant que le jeu est terminé avant même de commencer. Cette fonction garantit que le jeu démarre toujours dans un état valide et prêt à être joué.
4. *randomrvennumber(int matrix[4][4], int gameOverFlag)*\*  
   Cette fonction ajoute un nombre aléatoire (2 ou 4) dans une case vide de la matrice après chaque mouvement valide. Elle commence par identifier toutes les cases vides (valeurs égales à 0) et en choisit une au hasard pour y placer le nouveau nombre. Si aucune case vide n'est trouvée, la fonction met à jour le drapeau gameOverFlag pour indiquer que le jeu est terminé. Cette fonction est essentielle pour maintenir la progression du jeu et ajouter un élément de surprise à chaque tour.
5. **isGameOver(int matrix[4][4])**  
   Cette fonction vérifie si le jeu est terminé en examinant deux conditions :
   * **Cases vides** : Si une case de la matrice contient la valeur 0, le jeu n'est pas terminé.
   * **Fusions possibles** : Si deux tuiles adjacentes (horizontalement ou verticalement) ont la même valeur, une fusion est possible, et le jeu continue.  
     Si aucune case vide n'est trouvée et qu'aucune fusion n'est possible, la fonction retourne true, indiquant que le jeu est terminé.
6. **isWin(int matrix[4][4])**  
   Cette fonction vérifie si le joueur a atteint l'objectif du jeu en cherchant une tuile avec la valeur 2048 ou supérieure. Si une telle tuile est trouvée, la fonction retourne true, et le joueur est déclaré vainqueur. Sinon, elle retourne false, et le jeu continue.

**2. Mode Machine (IA Seule)**

****

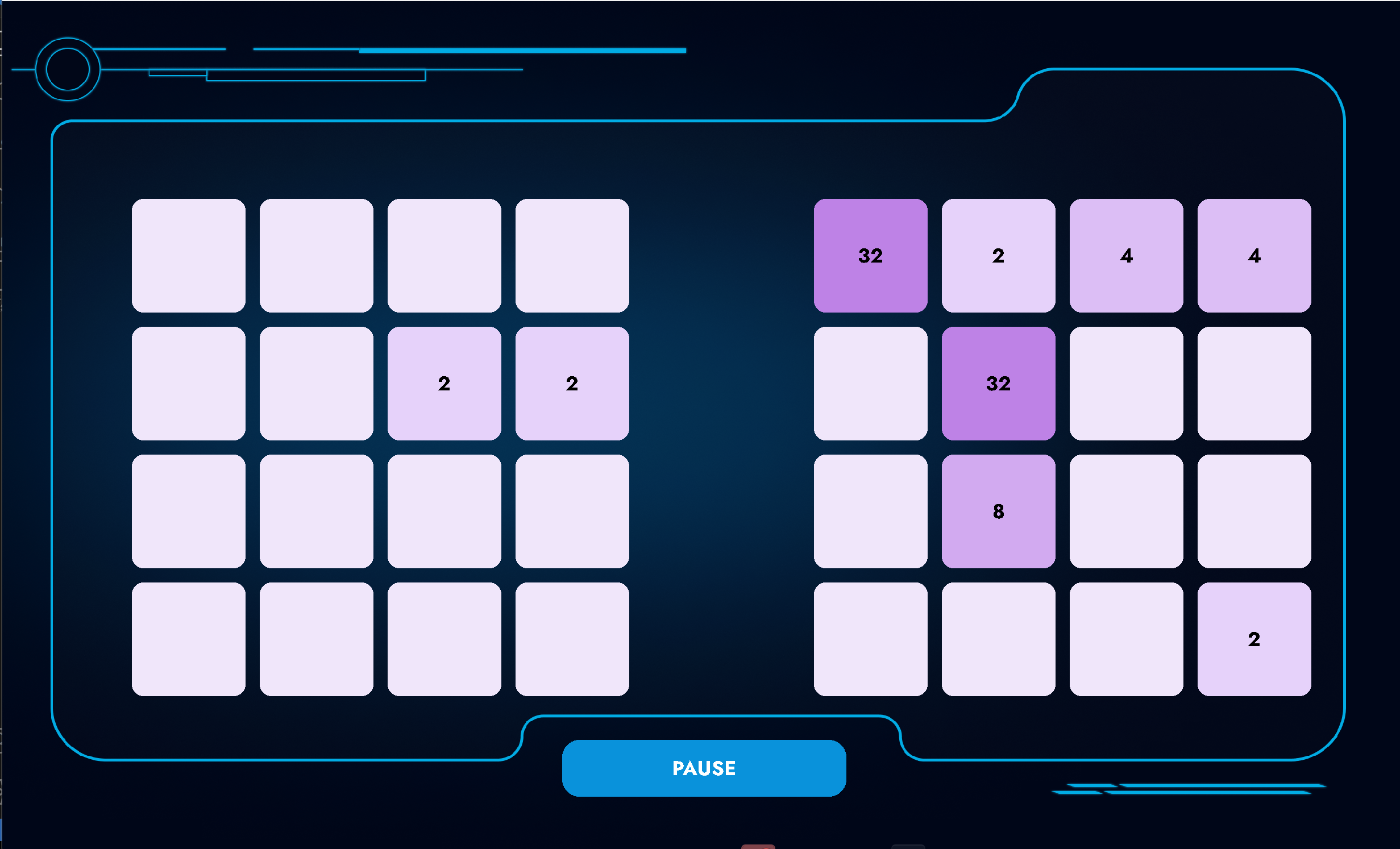
**Fonctionnement Global**

Le mode Machine permet à l'ordinateur de jouer automatiquement en utilisant un algorithme pour déterminer les meilleurs mouvements. L'IA essaie de maximiser les fusions et le score tout en évitant de remplir la grille trop rapidement.

**Fonctions Utilisées**

1. **copy\_matrix(int src[4][4], int dest[4][4])**  
   Cette fonction copie les valeurs d'une matrice source vers une matrice de destination. Elle est utilisée pour créer des copies de la matrice actuelle afin que l'IA puisse simuler les effets de différents mouvements sans modifier la matrice principale.
2. **check\_merge\_down(int board[4][4])**, **check\_merge\_right(int board[4][4])**, **check\_merge\_up(int board[4][4])**, **check\_merge\_left(int board[4][4])**  
   Ces fonctions vérifient les fusions possibles dans chaque direction. Par exemple, check\_merge\_down examine la matrice pour trouver des paires de tuiles adjacentes dans la direction bas qui peuvent être fusionnées. Chaque fonction retourne le nombre de fusions possibles, ce qui aide l'IA à évaluer les meilleurs mouvements.
3. **find\_best\_move(int board[4][4])**  
   Cette fonction évalue les quatre directions possibles (haut, bas, gauche, droite) en utilisant les fonctions check\_merge\_\*. Elle choisit la direction qui offre le plus grand nombre de fusions possibles. Si plusieurs directions offrent le même nombre de fusions, l'IA simule les résultats de chaque mouvement pour choisir celui qui maximise le score.
4. *machine(int board[4][4], bool active, ma move)*\*  
   Cette fonction exécute le mouvement choisi par l'IA. Si l'IA est active, elle copie la matrice actuelle, trouve le meilleur mouvement avec find\_best\_move, et met à jour la matrice en conséquence. Après chaque mouvement, un nouveau nombre est ajouté à la matrice via randomrvennumber.

**3. Mode Machine vs Player (Multijoueur)**

****

**Fonctionnement Global**

Dans ce mode, deux grilles sont affichées côte à côte : une contrôlée par le joueur et une autre par l'IA. Les deux grilles évoluent simultanément, et le but est de voir qui peut atteindre 2048 en premier ou obtenir le score le plus élevé.

**Fonctions Utilisées**

1. *renderTwoMatrices(SDL\_Renderer, int matrix1[4][4], int matrix2[4][4], TTF\_Font*, int windowWidth, int windowHeight, gc gametheme)\*\*  
   Cette fonction affiche deux matrices côte à côte à l'écran. Elle utilise renderMatrix pour dessiner chaque grille et les positionne de manière symétrique par rapport au centre de la fenêtre. Les dimensions et l'espacement sont ajustés en fonction de la résolution de l'écran.
2. **updateMatrix(int direction)**  
   Cette fonction est utilisée pour gérer les mouvements du joueur sur sa grille. Elle fonctionne de la même manière que dans le mode Player.
3. *machine(int board[4][4], bool active, ma move)*\*  
   Cette fonction gère les mouvements de l'IA sur sa grille. Elle fonctionne de la même manière que dans le mode Machine.

**4. Mode Score**

**Fonctionnement Global**

Le mode Score permet aux joueurs de consulter les meilleurs scores enregistrés. Les scores sont sauvegardés dans un fichier texte et peuvent être triés et affichés dans un tableau des scores (leaderboard).

**Fonctions Utilisées**

1. *SaveMatrixAndScoreToFile(int matrix[4][4], int score, const char filename)*\*  
   Cette fonction sauvegarde la matrice et le score dans un fichier texte. Elle est utilisée pour enregistrer les performances des joueurs.
2. *LoadMatrixAndScoreFromFile(int matrix[4][4], int score, const char* filename)\*\*  
   Cette fonction charge la matrice et le score depuis un fichier texte. Elle est utilisée pour reprendre une partie sauvegardée.
3. *ajouterEtTrierScore(const char nomFichier, int nouveauScore)*\*  
   Cette fonction ajoute un nouveau score au fichier et trie les scores en ordre décroissant. Seuls les meilleurs scores sont conservés.
4. *RenderLeaderboardFromFile(SDL\_Renderer, TTF\_Font*, const char\* filename, int screenWidth, int screenHeight)\*\*  
   Cette fonction affiche les meilleurs scores depuis un fichier. Les scores sont affichés dans un format visuellement attrayant, avec des tuiles arrondies et des couleurs adaptées au thème.

**5. Thème PERSO**

**Fonctionnement Global**

**Le mode Thème permet aux joueurs de personnaliser l'apparence du jeu en choisissant parmi plusieurs thèmes visuels. Chaque thème modifie les couleurs des tuiles, l'arrière-plan, et parfois les polices, offrant une expérience visuelle unique. Ce mode ajoute une touche de personnalisation et de variété au jeu.**

**Fonctions Clés**

1. ***renderMatrix(SDL\_Renderer, int matrix[4][4], TTF\_Font*, int windowWidth, int windowHeight, gc theme)\*\***
   * **Cette fonction affiche la matrice en utilisant les couleurs et styles définis par le thème choisi.**
   * **Exemples de thèmes :**
     + **Défaut (def) : Nuances de violet.**
     + **Cyan : Tons de bleu et cyan.**
     + **Sombre (dark) : Couleurs profondes et contrastées.**
2. ***fillRoundedRect(SDL\_Renderer, SDL\_Rect rect, int radius)*\***
   * **Dessine des tuiles avec des coins arrondis, adaptés au thème choisi.**
   * **Les coins arrondis ajoutent un aspect moderne et esthétique à l'interface.**
3. **Gestion des Thèmes dans le Menu**
   * **Les joueurs peuvent choisir un thème dans le menu principal.**
   * **Les boutons et l'arrière-plan du menu s'adaptent au thème sélectionné pour une expérience cohérente.**

**Exemples de Thèmes**

1. **Thème Défaut (def)**
   * **Couleurs : Nuances de violet.**
   * **Style : Classique et épuré.**
2. **Thème Cyan**
   * **Couleurs : Tons de bleu et cyan.**
   * **Style : Moderne et rafraîchissant.**
3. **Thème Sombre (dark)**
   * **Couleurs : Noir, gris foncé, accents blancs ou jaunes.**
   * **Style : Élégant et minimaliste.**

**Personnalisation Avancée**

**Le mode Thème est conçu pour être extensible, permettant d'ajouter facilement de nouveaux thèmes. Par exemple, un thème "Rétro" pourrait utiliser des couleurs vives et des polices pixelisées, tandis qu'un thème "Nature" pourrait utiliser des tons de vert et de brun.**

**Impact sur l'Expérience Utilisateur**

**Le mode Thème améliore l'expérience utilisateur en offrant une personnalisation visuelle. Les joueurs peuvent choisir un thème qui correspond à leurs préférences, rendant le jeu plus engageant et agréable. La variété des thèmes maintient également l'intérêt des joueurs sur le long terme.**

**Conclusion**

Ces modes de jeu offrent une expérience variée et engageante, en utilisant les fonctions fournies pour gérer la logique du jeu, l'affichage graphique, et la gestion des scores. Chaque mode est conçu pour être intuitif et amusant, tout en réutilisant les mêmes fonctions de base pour assurer la cohérence et la modularité du code.