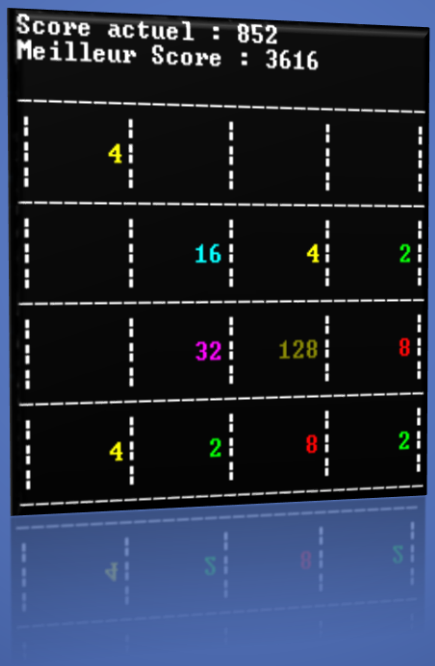
2015

Projet 2048



ARNAUD GERMAN – ADRIEN MAGNIN – LAURENT TOSON

LANGAGE C – S1A INFORMATIQUE – IUT SOPHIA ANTIPOLIS

Université Nice Sophia Antipolis

iut Nice Côte d'Azur

Table des matières

INTRODUCTION

Au cours du premier semestre à l'IUT Informatique en alternance, il nous a été demandé de réaliser le jeu « 2048 », initialement créé par Gabriele Cirulli, à l'aide du langage de programmation C.

Dans ce rapport, nous allons vous présenter dans un premier temps le fonctionnement du programme réalisé, puis nous allons détailler les fonctions principales qui permettent au jeu de fonctionner.

Ce travail a été réalisé au sein d'un trinôme (Arnaud GERMAN, Adrien MAGNIN et Laurent TOSON), où la programmation des différentes fonctions a été répartie de manière équitable. En effet, la conception et la réalisation des menus, du tutoriel, et de l'affichage graphique du jeu a été réalisée par Adrien, les fonctions de tests, de fin de partie (victoire/défaite), de capture de touches et de manipulation de fichiers (lecture/écriture dans un fichier texte) par Laurent et la gestion des mouvements, des scores et la génération aléatoire de nombres par Arnaud.

MODE D'EMPLOI DU PROGRAMME

Compilation du code source

Pour lancer le programme 2048, il faut d'abord extraire les données de 2048.zip. Le fichier exécutable se trouve dans le dossier Jeu_2048.

Les bibliothèques utilisées dans le programme sont **<stdio.h>**, **<stdlib.h>**, **<time.h>** et **<windows.h>**. Si vous souhaitez recompiler le programme, veillez à ce que ces bibliothèques soient bien installées sur votre ordinateur.

Ce programme est destiné à être utilisé sous Windows. Cependant, le code source peut toujours être modifié si on souhaite l'exécuter sous Linux ou OSX. Pour cela, il faut modifier les bibliothèques, la gestion des touches appuyées, la fonction **clearScreen** ainsi que la fonction **color**.

Utilisation du programme

Une fois le programme lancé, une console s'ouvrira, affichant le menu principal du programme. Pour effectuer une action, il suffit d'appuyer sur une touche correspondant au numéro de l'action désirée. Lors d'une partie, il faut appuyer sur une touche directionnelle pour effectuer un mouvement. Si vous désirez quitter la partie, la touche correspondante est la touche « Echap ». A chaque fois qu'une partie est quittée, elle est automatiquement enregistrée dans un fichier nommé save.txt.

Gestion des sauvegardes

A chaque déplacement dans une partie, la grille est enregistrée dans le fichier save.txt. Lorsqu'une partie est perdue, le fichier n'est pas supprimé mais la grille contenu dans le fichier est réinitialisée à zéro.

Lors de la première victoire ou défaite de la partie, le score est enregistré puis un fichier nommé scores.txt est créé. Dans ce fichier seront stockés les 10 meilleurs scores.

Ces deux fichiers save.txt et scores.txt sont créés dans le même répertoire que l'exécutable. Si les fichiers sont déplacés ou supprimés, le programme ne génèrera aucune erreur mais devra recréer ces fichiers, ainsi les données seront perdues. Ces fichiers peuvent alors être récupérés de façon à partager ses scores ou sa dernière partie enregistrée.

FONCTIONNEMENT GÉNÉRAL DU PROGRAMME

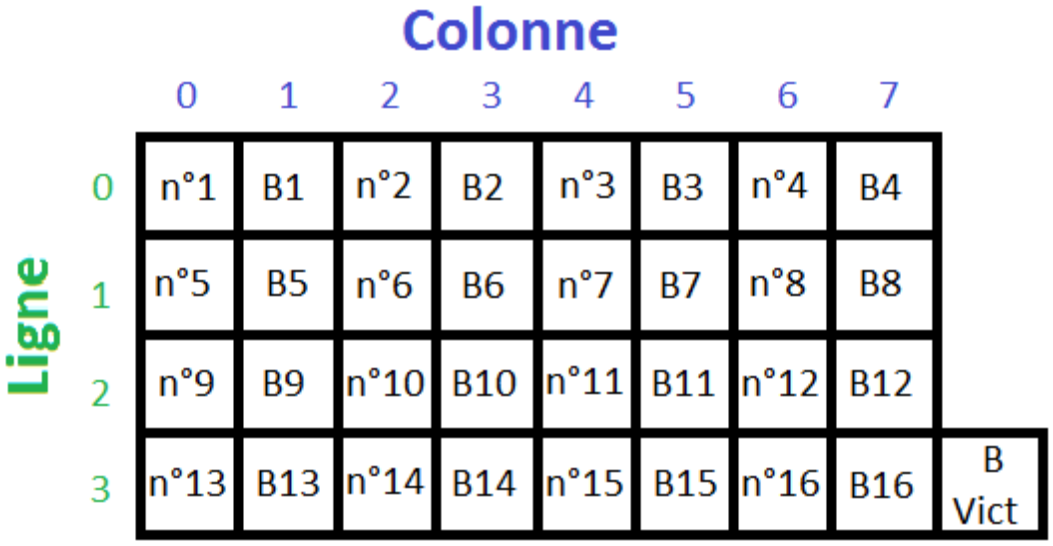
De base, un 2048 doit être constitué de 16 cases (4x4). Afin de faciliter la programmation du jeu, nous avons utilisé un tableau de 33 cases réparties comme ci-dessous. En effet, en plus des 16 cases affichées de base pour un 2048 (n°1 à n°16), nous avons ajouté 16 autres cases qui stockent un booléen (B1 à B16) ainsi qu'un booléen de victoire (B Vict).

Pour accéder à la case n°X de colonne i et de ligne j, nous procédons comme ci-dessous:

```
*(grille+i+j*8) = contenu de n°X
```

Pour le booléen, on utilise la même méthode en rajoutant « +1 » :

```
*(grille+i+j*8+1) = contenu de BX
```



La description des **booléens** est abordée de manière plus détaillée dans la suite du rapport.

Au lancement du programme, la grille est initialisée avec des « 0 » pour toutes les cases. On affiche ensuite le menu **(cf. Figure 3)** où l'utilisateur peut choisir de commencer une nouvelle partie, charger une partie existante, d'afficher les 10 meilleurs scores, de lancer le tutoriel ou de quitter le programme. Si l'utilisateur choisit de charger une partie, la procédure **lectureGrille** est appelée et va initialiser la grille avec les valeurs sauvegardées dans le fichier save.txt, puis la procédure **gaming** est appelée. Lors d'une nouvelle partie, la grille est réinitialisée (procédure **initZero**) avant d'appeler la procédure **gaming**.

L'affichage des scores est réalisé par la procédure **afficheScores** et le tutoriel **(cf. Figure 4) par** la procédure **tutoriel**.

ÉTUDE DE L'ALGORITHME GAMING

La procédure gaming est structurée par un **while** principal, permettant tous les appels de fonctions/procédures nécessaires au déroulement d'une partie jusqu'à la fin de partie. Premièrement, une seconde boucle **while** permet la capture des flèches directionnelles ou de la touche « Echap » pour quitter la partie. En fonction de la touche directionnelle appuyée, une fonction de mouvement est appelée, retournant la valeur 1 si le mouvement est réalisable (ou 0 dans le cas contraire). Lorsqu'un mouvement est effectué, on appelle alors la procédure **genereNombre** dont le but est de générer aléatoirement un « 2 » ou un « 4 » dans une des cases vides de la grille (valeur = 0). Ensuite, on réinitialise les **booléens** BX de la grille à zéro, on appelle la fonction **clearScreen** pour vider l'affichage de la console et par la suite, on fait appel à la procédure **afficheGrille** pour afficher le résultat du déplacement et le score incrémenté. Enfin, on teste alors si l'utilisateur a gagné ou perdu.

AFFICHAGE DE LA GRILLE

Le but de cette procédure est d'afficher la grille, notamment le score et le meilleur score. Notre grille est affichée sous forme de texte avec une largeur de case fixe. Le nombre est précédé d'un nombre d'espaces variable que l'on détermine grâce à la fonction **tailleNombre**. On utilise deux procédures pour afficher les lignes et les bordures de notre grille, **bordureH** et **afficheLigne**.

```
void afficheGrille(int *grille, int *score, int meilleurScore)
{
    // Affichage du score actuel
    printf("Score actuel : %d\n", *score);

    // Affichage du meilleur score
    if (*score>=meilleurScore)
    {
        printf("Meilleur Score : %d\n", *score);
    }
    else
    {
        printf("Meilleur Score : %d\n", meilleurScore);
    }

    printf("\n");
    // Affichage des lignes de la grille
    bordureH();
    afficheLigne(0,grille);
    bordureH();
    afficheLigne(1,grille);
    bordureH();
    afficheLigne(2,grille);
    bordureH();
    afficheLigne(3,grille);
    bordureH();
}
```

ÉTUDE DE L'ALGORITHME CAPTURE DE TOUCHE

Pour capturer une touche du clavier sans validation par la touche « Entrée » et sans interrompre le programme, nous avons utilisé la fonction **getch()**, disponible avec la bibliothèque **<stdio.h>**. La fonction nous permet d'également de ne pas afficher sur la console ce qui est saisi par l'utilisateur. Lorsque l'utilisateur appuie sur une touche du clavier, la fonction **getch()** retourne un entier correspondant au code unique de la touche. Les flèches du clavier sont un cas particulier. En effet, elles doivent être capturées par deux **getch()** consécutifs, le premier étant le code 224, le second étant spécifique à chacune des flèches (ex : 75 pour la flèche gauche).

```
while((toucheAppuyee=getch()) == 224)
{
    toucheFleches=getch();
    switch (toucheFleches)
    {
    case 75:
        // Touche gauche
        break;
    case 77:
        // Touche droite
        break;
    case 72:
        // Touche haut
        break;
    case 80:
        // Touche bas
        break;
    }
}
```

A l'aide d'une boucle **while**, tant qu'une touche directionnelle n'est pas appuyée, la boucle continue, le **switch** permet ensuite de distinguer quelle flèche est choisie.

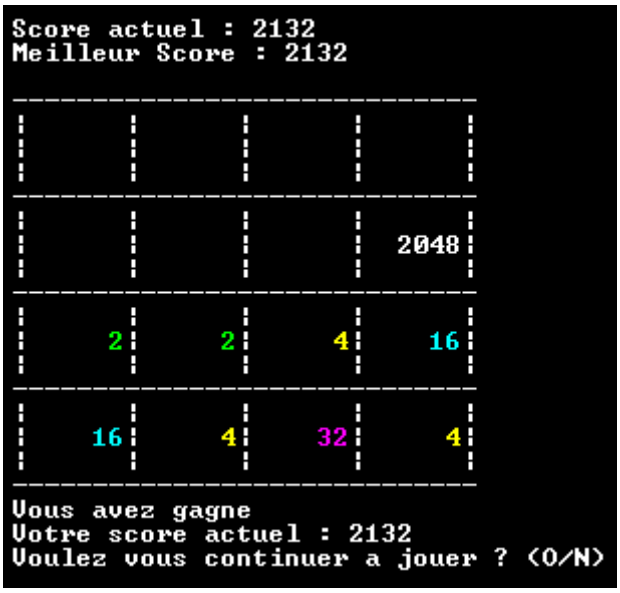
ÉTUDE DE L'ALGORITHME DES MOUVEMENTS

Nous allons maintenant étudier l'algorithme d'un mouvement étant l'un des plus complexes de notre programme. Lorsque l'utilisateur appuie par exemple sur la flèche gauche, les nombres doivent être déplacés vers la gauche, ne laissant aucun espace libre entre le dernier nombre de chaque ligne et le bord gauche de la ligne. De plus, si deux nombres adjacents sont égaux, ils sont alors fusionnés en un seul nombre étant la somme de ces derniers. Enfin, un nombre qui a été fusionné ne peut être fusionné une seconde fois au cours d'un même mouvement. Ainsi, l'algorithme procède ligne par ligne (ou colonne par colonne pour un mouvement vertical) où l'on cherche dans un premier temps à faire tous les déplacements possibles puis dans un second temps, procéder aux additions. Lorsqu'une addition est effectuée, un **booléen** « **aajoute** » est initialisé pour indiquer que des nouveaux déplacements sont possibles. Il en est de même lorsqu'on effectue des déplacements, le **booléen** « **adeplace** » indique la possibilité de nouvelles fusions. Si lors de la relecture d'une ligne, aucun déplacement ou aucune addition n'est effectué, on passe alors au traitement de la ligne suivante. Pour finir, si l'on a effectué un déplacement ou une addition sur toute la grille, le **booléen** « **mouvementvalable** » prendra la valeur « 1 », ainsi, la fonction retournera cette valeur.

ÉTUDE DE L'ALGORITHME DE FIN DE PARTIE

Condition de victoire

Pour gagner au 2048, il faut additionner deux nombres 1024 pour atteindre 2048. Après chaque mouvement, le programme réalise une lecture complète de la grille à la recherche d'une valeur égale à 2048. Une fois le 2048 trouvé, on affiche un écran de victoire, et on initialise un **booléen** (à la valeur de 1), situé dans la dernière valeur du tableau **grille**. Ce **booléen** sert de mémoire au programme afin d'éviter de re-tester la présence d'un 2048 si le joueur continue la partie.



Condition de défaite

Le joueur a perdu si aucun mouvement n'est possible **(cf. Figure 1)**. Le programme vérifie dans un premier temps la présence d'une case vide (égale à 0), car dans ce cas, il est inutile de faire d'autres opérations, un mouvement étant toujours possible. Cependant, si aucune case vide n'est détectée, on cherche alors ligne par ligne la présence de deux nombres adjacents égaux. Si aucune addition horizontale n'est possible, on cherche alors verticalement. En effet, si dans la grille, deux nombres adjacents sont égaux, cela signifie que l'on peut les additionner et donc qu'il reste un mouvement possible.

```c
int testGameOver(int* grille)
{
    int i,j;
    int casePrecedente, caseSuivante;


    // Renvoie 0 si une case est vide
    for (i=0; i<16; i++)
    {
        if (*(grille + i*2)==0)
        {
            return 0;
        }
    }


    // Renvoie 0 si deux cases adjacentes horizontalement sont égales
    for (i=0; i<4; i++)
    {
        for (j=0; j<3; j++)
        {
            casePrecedente = *(grille + j*2 + i*8);
            caseSuivante = *(grille + j*2 + i*8 + 2);
            if (casePrecedente == caseSuivante)
            {
                return 0;
            }
        }
    }


    // Renvoie 0 si deux cases adjacentes verticalement sont égales
    for (i=0; i<3; i++)
    {
        for (j=0; j<4; j++)
        {
            casePrecedente = *(grille + j*2 + i*8);
            caseSuivante = *(grille + j*2 + i*8 + 8);
            if (casePrecedente == caseSuivante)
            {
                return 0;
            }
        }
    }
    return 1;
}
```

LECTURE/ECRITURE D'UN FICHIER TEXTE

Afin de manipuler un fichier texte en C, on utilise les fonctions **fprintf()**, **fscanf()**, **fopen()** et **fclose()**. On charge en mémoire le fichier texte dans une structure de type **FILE** qui gère le **buffer** du fichier à l'aide de la commande **fopen()**. **fprintf()** et **fscanf()** permettent respectivement d'écrire et de lire une ligne dans le fichier et **fclose()** libère la mémoire du **buffer**.

Fichier save.txt

Afin de sauvegarder la partie de manière durable (sur la mémoire morte), la grille, le score et le **booléen** de victoire sont écrits après chaque mouvement dans le fichier save.txt qui est écrasé. On écrit une valeur par ligne en commençant par la première case de la grille jusqu'à la dernière, puis le score et enfin, le **booléen** de victoire décrit précédemment.

Fichier scores.txt

Lorsqu'une partie est perdue, l'utilisateur est invité à saisir son pseudonyme. Le programme teste la présence du fichier scores.txt, si celui-ci est trouvé, son contenu est chargé ligne par ligne dans les tableaux **TabScore** et **TabPseudos**. On teste à chaque score si celui-ci est bien supérieur à celui de la partie perdue, dans le cas contraire, on insère donc le score de cette partie ainsi que le pseudo dans les deux tableaux, puis on continue le chargement des autres valeurs du fichier scores.txt. Ainsi, les scores restent triés de manière décroissante dans les tableaux **TabScore** et **TabPseudos**, il nous reste alors à écraser le fichier scores.txt par le contenu de ces deux tableaux **(cf. Figure 2)**. L'avantage de cette méthode réside dans un algorithme simple et rapide. Cependant, si le fichier scores.txt est corrompu (score non trié ou ligne manquante), le programme n'effectuera pas de restauration du contenu du fichier.

```c
int triScore(int score, char *TabPseudo)
{
    int i, j;
    int flagMove=0;
    int TabScores[11];
    char TabPseudos[11][20];
    int SaveScore;

    FILE *fichierScore = NULL;
    fichierScore = fopen("scores.txt", "r");
    i=0;

    while (i<11)
    {
        fscanf(fichierScore, "%d", &SaveScore );

        if (score<SaveScore || flagMove==1)
        {
            TabScores[i]=SaveScore;
            fscanf(fichierScore, "%s", TabPseudos[i] );
        }
        else if (flagMove == 0)
        {
            flagMove=1;
            TabScores[i]=score;
            for (j=0; j<20; j++)
            {
                TabPseudos[i][j]=TabPseudo[j];
            }
            i++;
            TabScores[i]=SaveScore;
            fscanf(fichierScore, "%s", TabPseudos[i] );
        }
        i++;
    }
    fclose(fichierScore);

    fichierScore = fopen("scores.txt", "w");

    // On réécrit les 10 meilleurs scores avec les pseudos correspondant
    for (i=0; i<10; i++)
    {
        fprintf(fichierScore, "%d\n", TabScores[i]);
        fprintf(fichierScore, "%s\n", TabPseudos[i]);
    }
    fclose(fichierScore);
}
```

CONCLUSION

Lors de ce projet, nous avons pu parfaire nos connaissances dans l'art de la programmation et de l'algorithmique. Nous avons aussi découvert en quoi consiste la programmation collaborative d'un même programme, notamment lorsque nous avons dû rassembler nos codes sources respectifs dans un espace partagé. De nouvelles notions sont introduites telles que la gestion d'un fichier texte, de la capture des touches du clavier.

Le jeu 2048 paraît simple lorsqu'on y joue mais les algorithmes utilisés pour le réaliser sont beaucoup plus complexes, notamment pour réussir les mouvements et les fusions.

Grâce à ce projet, nous planifions de travailler ensemble sur d'autres projets informatiques.

ANNEXES

Figure 1 : Ecran défaite

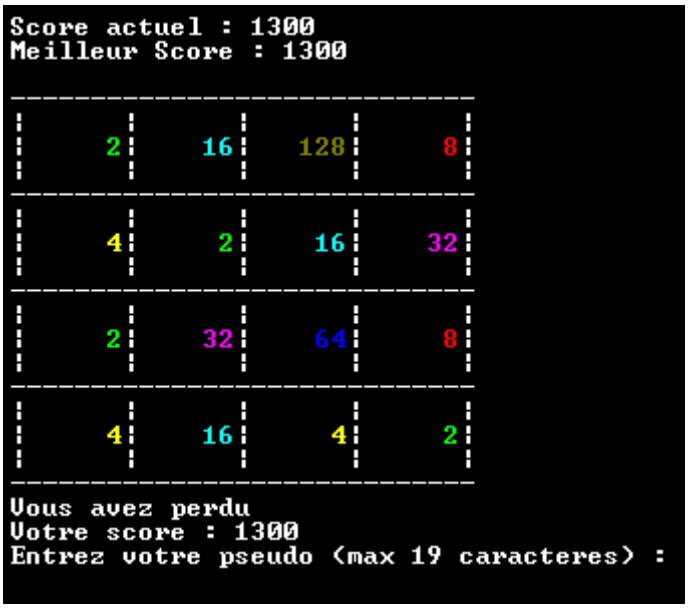


Figure 2 : Ecran d'affichage des meilleurs scores

Figure 3 : Ecran menu

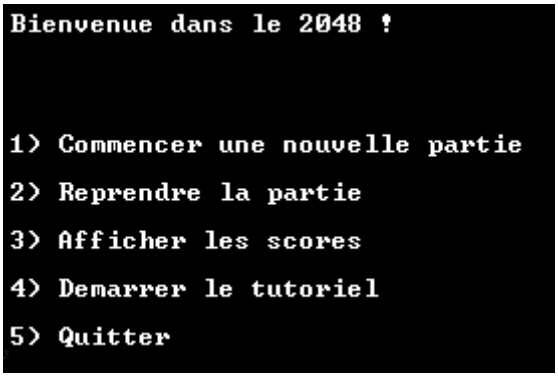
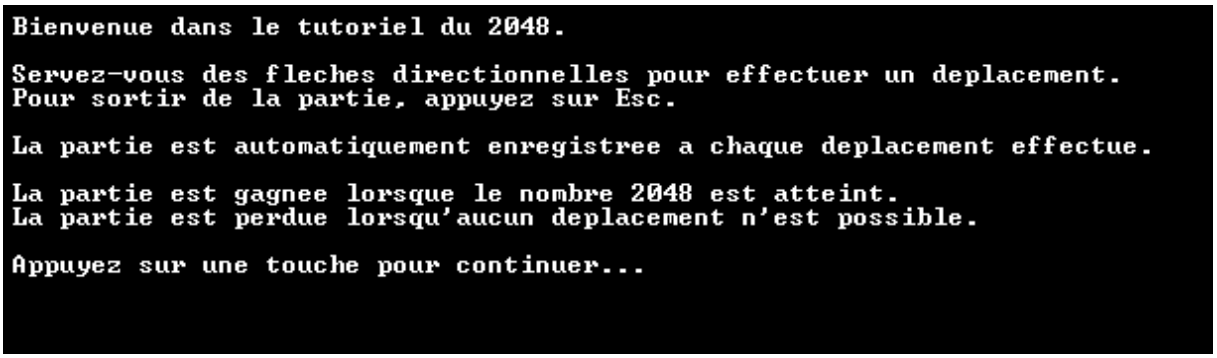


Figure 4 : Ecran tutoriel



DÉCOMPOSITION FONCTIONNELLE

Procédure : gaming
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Permet de jouer une partie de 2048 en appelant les fonctions de déplacement et de vérification de victoire ou de défaite.

Procédure : sauteLigne
Paramètres : (entier) nbLigne ;
Description : Permet de faire des sauts de lignes en rendant le code plus propre. On rentre en paramètres le nombre de lignes que l'on veut sauter.

Procédure : initZero
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Initialise la grille de la partie pour qu'elle soit entièrement vide et initialise le score à 0.

Procédure : afficheGrille
Paramètres : (pointeur-entier) grille, (pointeur-entier) score, (entier) meilleurScore ;
Description : Permet d'afficher la grille graphiquement, avec le score et le meilleur score. Elle fait appel à différentes procédures pour afficher les bordures.

Procédure : bordureH
Paramètres : aucun
Description : Permet de tracer les bordures horizontales de la grille.

Fonction : tailleNombre
Paramètres : (entier) nombre ;
Description : Permet de connaître la taille d'un nombre de la grille en digit.
Retour : Renvoie la taille du nombre en digit. (Exemple: 64 = 2 digits)

Procédure : afficheLigne
Paramètres : (entier) ligne, (pointeur-entier) grille ;
Description : Affiche les lignes du le grille contenant les valeurs ainsi que les deux lignes vides qui les entourent.

Procédure : clearScreen
Paramètres : aucun
Description : Permet d'effacer le contenu de la console.

Procédure : ecritureGrille
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Enregistre la grille et le score dans le fichier save.txt.

Procédure : lectureGrille
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Lis le fichier save.txt pour récupérer la grille et le score enregistrés.

Fonction : moveLeft
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Effectue tous les mouvements possibles vers la gauche.
Retour : Renvoie 1 si un mouvement a pu être effectué, sinon 0.

Fonction : moveRight
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Effectue tous les mouvements possibles vers la droite.
Retour : Renvoie 1 si un mouvement a pu être effectué, sinon 0.

Fonction : moveUp
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Effectue tous les mouvements possibles vers le haut.
Retour : Renvoie 1 si un mouvement a pu être effectué, sinon 0.

Fonction : moveDown
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Effectue tous les mouvements possibles vers le bas.
Retour : Renvoie 1 si un mouvement a pu être effectué, sinon 0.

Procédure : genereNombre
Paramètres : (pointeur-entier) grille ;
Description : Génère un nombre aléatoire puis le place dans une case libre toujours de manière aléatoire.

Procédure : resetFlags
Paramètres : (pointeur-entier) grille ;
Description : Initialise les booléens de la grille qui gèrent la mise en mémoire d'une addition à 0.

Fonction : testGameOver
Paramètres : (pointeur-entier) grille ;
Description : Test si la partie est perdue ou si elle peut être continuée.
Retour : Renvoie 1 si la partie ne peut plus être continuée, sinon 0.

Procédure : menu
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Affiche le menu principal et attend le choix de l'utilisateur.

Procédure : triScore
Paramètres : (entier) score, (pointeur-caractère) TabPseudo ;
Description : Ajoute le dernier score enregistré comme 11^{e} score du fichier scores.txt puis trie les 11 scores du fichier.

Procédure : gameOverScreen
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Ecran à afficher lorsqu'une partie est perdue.

Fonction : testVictoire
Paramètres : (pointeur-entier) grille ;
Description : Vérifie si une case de la grille contient la valeur 2048 afin de savoir si la partie est gagnée.
Retour : Renvoie 1 si la partie est gagnée.

Procédure : victoireScreen
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Ecran à afficher lorsqu'une partie est gagnée.

Procédure : changeColor
Paramètres : (entier) nombre ;
Description : Choisis la couleur à afficher en fonction du nombre à afficher.

Procédure : color
Paramètres : (entier) c ;
Description : Procédure permettant de modifier la couleur de la console.

Procédure : afficheScores
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Récupère les 10 meilleurs scores dans le fichier scores.txt puis affiche ces 10 scores ainsi que leurs pseudonymes correspondants dans la console.

Fonction : testSave
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Test si la grille du fichier save.txt ne contient pas que des 0.
Retour : Renvoie 0 si la grille est vide, sinon 1

Procédure : noSave
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Message d'erreur à afficher lorsque l'utilisateur souhaite continuer une partie alors qu'aucune partie est enregistrée.

Procédure : tutoriel
Paramètres : (pointeur-entier) grille, (pointeur-entier) score ;
Description : Procédure contenant le tutoriel.

Fonction : victoireTuto
Paramètres : (pointeur-entier) grille ;
Description : Test si une case de la grille contient la valeur 64 afin de savoir si le tutoriel est terminé.
Retour : Renvoie 1 si le grille contient un 64, sinon 0.

Fonction : getMeilleurScore
Paramètres : aucun
Description : Permet de récupérer le meilleur score dans le fichier scores.txt
Retour : Renvoie le meilleur score.

Procédure : initScores
Paramètres : aucun
Description : Test si le fichier scores.txt est présent. Sinon, elle le créée et initialise les 10 meilleurs scores à 0.