Justificatif technique

Projet Xmas Treats

Kami GEERLINGS
Alexia ILIE
Groupe 4

15 décembre 2023

Sommaire

Introduction

Dans le cadre de la matière "Introduction à la programmation", Madame Tetelin nous a demandé de programmer un jeu de plateau sur le thème de Noël en binôme. Les règles du jeu sont simples. À partir d'une grille de jeu de taille 4x4, on introduit deux bonbons dans une position aléatoire. A chaque tour de jeu, le joueur a la possibilité de déplacer les bonbons vers le haut, le bas, la gauche ou la droite. Une fois que le coup est joué, un nouveau bonbon est introduit de manière aléatoire dans la grille. Le jeu s'arrête lorsque le nombre maximal de coups est joué ou lorsque la grille est remplie, et donc bloquée. Ce jeu est similaire à un 2048 et nous permet de mieux comprendre la logique attendue.
Ce document de justifications techniques a pour objectif de clarifier le code proposé pour le projet de création du jeu de XMAS Treats. Dans un premier temps, nous vous expliquons le programme de notre jeu en fonction des contraintes du jeu données. Une deuxième partie explique nos choix techniques et explique de manière détaillée les différentes étapes de notre code. Enfin, dans une dernière partie sur l'organisation des projets, nous avons récapitulé l'organisation de notre travail de groupe, tant sur notre participation personnelle que sur le planning que nous avons suivis. Pour créer ce jeu, nous avons programmé sur Visual Studio en mode Console en utilisant le langage C#, avec une approche procédurale. Ce travail d'équipe à été rendu possible par l'utilisation de GitHub pour coopérer à distance.

<u>Les contraintes techniques :</u>

La programmation a été faite en C#, en mode console, dans les conditions habituelles des TP sans utiliser de bibliothèques de fonctions externes. La programmation orientée objet et l'utilisation de listes des collections est elle aussi exclue. Le code source respecte la norme camelCase et des commentaires sont présents pour expliquer des parties de code complexes ou importantes du code. Chaque sous-programme est également nommé à l'aide d'un verbe à l'infinitif et en commençant par une majuscule.

I. Explication du programme

Structure générale du jeu et du code

L'objectif du jeu pour le joueur est de déplacer des bonbons sur le plateau de jeu afin de les combiner en bonbons plus gros et obtenir le score le plus élevé possible. Il existe pour cela 4 types de bonbons qui rapportent plus ou moins de points. Le plateau de jeu, initialement de taille 4x4, comporte initialement 2 bonbons placés aléatoirement dans le plateau de manière à faciliter le démarrage du jeu. Le joueur peut alors déplacer les bonbons dans une des 4 directions possibles de manière à rencontrer 2 treats identiques pour les transformer dans le treat de score supérieur. A chaque tour, un nouveau bonbon est introduit aléatoirement dans une des cases vides restantes. Dans le cas échéant, le jeu s'arrête à cause du blocage du plateau de jeu. Le jeu s'arrête également lorsque le joueur à épuisé tous les coups qu'il avait initialement choisis. Le but de jeu est alors de faire le score le plus élevé en fonction du nombre de coups choisis.

Le programme de ce jeu permet donc de créer les différentes fonctionnalités nécessaires à sa mise au point. Dans un premier temps, le joueur est interrogé sur le nombre de coups qu'il souhaite autoriser pour la partie. Ensuite, le programme construit le plateau de jeu dans lequel se déplacent les bonbons, et dans lequel se déroule la partie. Le joueur doit pouvoir déplacer les bonbons au gré de ses envies, grâce aux touches indiquées plus bas. Enfin, notre programme affiche le plateau de jeu au fur et à mesure des tours de manière à ce que le joueur suive le déroulement de la partie et qu'il fasse les bons choix de déplacements pour augmenter au maximum son score. Pour créer ce jeu, nous avons créé un programme principal qui regroupe diverses sous-fonctions permettant chacune des fonctionnalités particulières du jeu.

La première partie du code principal pose des questions à l'utilisateur afin de s'assurer qu'il connaît bien les règles du jeu. Dans le cas échéant, les règles du jeu lui sont expliquées. Une deuxième question permet de demander au joueur le nombre de coups autorisé pour la partie, de manière à faire durer la partie plus ou moins longtemps.

```
Console.WriteLine();
Console.WriteLine("Bienvenue sur 🔍🔍🔍CANDYMIX🔍🔍🔍");
Console.WriteLine();
Console.WriteLine("Connaissez-vous les règles du jeu ? Repondez par oui ou non.");
string reponse = Console.ReadLine()!;

if (reponse == "non" || reponse == "Non") //si le joueur connait pas les regles alors on les affiche
{
    Console.WriteLine("Le but du jeu est de déplacer les bonbons dans la grille du jeu afin qu'ils se rencontrent et se transforment dans le treat supér
}

else           //si il les connait alors on commence le jeu
{
    Console.WriteLine("✨Parfait, c'est parti !✨");
}
```

Partie du code affichant les règles du jeu

```
Console.WriteLine("Choissisez le nombre de coups autorisés:");      //choix du nombre de coups
int nbCoups = Convert.ToInt32(Console.ReadLine()!);                 //on enregistre la reponse du joueur et on le converti en entier
Console.WriteLine($"✨Vous avez choisi de jouer en {nbCoups} coups, c'est parti !✨");  //on lui affiche le nombre de coups qu'il a choisi
```

Partie du code pour que le joueur choisisse le nombre de coups qu'il veut jouer

La seconde partie du code principal permet le fonctionnement du jeu en lui-même. On affiche la matrice de jeu de départ qui contient deux bonbons à l'origine, qu'on a introduit grâce à la sous- fonction 'SymboleMatrice2' de manière aléatoire, et on demande à l'utilisateur d'utiliser les touches 8, 6, 4 et 2 pour déplacer tous les bonbons simultanément dans la direction de son choix.

```
Random aleatoire = new Random();                                    //on crée une variable qui va tirer au hasard des entiers
int[,] matriceDeJeuEntiers = SymboleMatrice2(MatriceEntiers(4));    //on place le premier bonbon dans une case aléatoire
matriceDeJeuEntiers = SymboleMatrice2(matriceDeJeuEntiers);         //on place le deuxième bonbon dans une case aléatoire
Console.WriteLine();
Console.WriteLine("Voici votre plateau de jeu de départ");
AfficherMatrice(ConversionMatrice(matriceDeJeuEntiers));            //affichage de la matrice de départ
```

Partie du code qui introduit les deux bonbons dans la grille de jeu puis l'affiche

Grâce à une boucle 'do while', si l'utilisateur rentre dans le terminal d'autres caractères que 2, 4, 6 ou 8, on continue d'afficher la consigne indiquant les commandes à utiliser pour permettre le déplacement des bonbons dans la grille.

```
int deplacement;
do                                                  //boucle qui permet de réafficher la consigne si l'utilisation des commandes n'a pas été respectée
{
    Console.WriteLine();
    Console.WriteLine("Déplacez les bonbons grâce aux touches 8(↑), 4(←), 2(↓) et 6(→):");
    deplacement = Convert.ToInt32(Console.ReadLine()!);        //on convertit les données rentrées par l'utilisateur en entier
} while (deplacement != 8 && deplacement != 4 && deplacement != 6 && deplacement != 2); //la boucle while s'arrête quand l'utilisateur rentre 2, 4, 6 ou 8 da
```

Partie du code qui vérifie si l'utilisateur rentre 2, 4, 6 ou 8 dans le terminal

Pour permettre des comparaisons et faciliter le codage du jeu, nous avons codé une matrice contenant des 0, 2, 4, 8 et 16 qui est ensuite transformée en une matrice de caractères qui contient les bonbons grâce à une sous-fonction ('ConvertirMatrice'). Chaque entier correspond finalement à un symbole que l'on visualise dans le jeu:

Entiers	0	2	4	8	16
Symboles	espace	*	@	o	J

Le jeu contient finalement deux conditions d'arrêt. La partie s'arrête la plupart du temps si le joueur épuise la totalité des coups permis. Cela est programmé grâce à une boucle 'for' qui est conditionnée par le nombre de coups initialement choisis par l'utilisateur. Pourtant, si le joueur choisit un nombre important de coups possible (au delà de 150), un blocage du plateau de jeu peut être possible que ce soit parce-que le joueur n'est pas efficace pour faire se rencontrer les treats identiques, ou au contraire, s'il est très efficace et entraîne donc le remplissage du plateau de jeu avec des 'J', le treat maximal. Ainsi, une sous-fonction ('VerificationMatrice'), permet de vérifier après chaque déplacement des bonbons à chaque tour, s'il y a un blocage dans la matrice, c'est-à-dire si la matrice est remplie de bonbons. Dans ce cas, le jeu s'arrête. Un message signale au joueur la raison de cette fin de partie et l'invite à rejouer une partie.

```
if (VerificationMatrice(matriceDeJeuEntiers) == true)                    //on verifie si il y a un blocage dans la grille
{
    Console.WriteLine();
    Console.WriteLine("Fin de la partie. Le plateau de jeu est remplie de bonbons et il y a un blocage... Rejouez et faites mieux !");
    break;
}
```

Partie du code qui vérifie si il y a un blocage dans la grille de jeu

Le fonctionnement général du jeu est donc possible grâce à plusieurs sous-programmes qui permettent l'enchaînement des diverses fonctionnalités du jeu. Chaque sous-programme est créé de manière presque indépendante puis appelé au fur et à mesure par le programme principal en fonction du besoin de la fonctionnalité qu'il édifie. En effet, ce fonctionnement évite des répétitions dans notre code et permet une organisation plus spécifique de notre code, assurant un meilleur contrôle. L'utilisation des sous-programmes nous a permis une structure organisationnelle de notre projet tout en simplifiant chaque action ou fonctionnalité du jeu.

Bonus

En plus des fonctionnalités exigées pour créer ce jeu, nous avons décidé d'implémenter une fonctionnalité en plus pour rendre le jeu plus complet et agréable à jouer. Cette fonctionnalité en plus permet, en répondant correctement à une énigme, de gagner 5 coups supplémentaires alors que le nombre de coups choisis par le joueur est épuisé. Ainsi, lorsque le joueur a épuisé le nombre de coups auquel il avait droit, il choisit s'il souhaite l'opportunité ou non de gagner des coups supplémentaires. S'il décide de tenter d'y répondre, alors il y a deux possibilités selon sa réponse. S'il répond correctement, le joueur obtient cinq coups supplémentaires, dans le cas contraire, le jeu se termine. Ce bonus a été programmé à partir d'une boucle 'if/else'. Si la réponse est correcte, alors le programme principal du jeu se répète, et ce seulement pour un nombre de coups limité à cinq. Dans le

cas d'une mauvaise réponse, un message annonce au joueur son échec et la partie se termine véritablement. Pour rejouer et tenter de faire un meilleur score, le joueur devra donc relancer une nouvelle partie.
Par ailleurs, nous avons décidé de donner un titre à notre jeu pour le rendre plus interactif et agréable à jouer; notre jeu s'appelle donc 'CANDYMIX'.

II. Choix techniques

Explication détaillée des sous-fonctions

Le programme principal, celui qui permet d'exécuter le jeu, fait appel à l'ensemble de sous-fonctions dont le code est composé. Chacune des sous-fonctions a une fonctionnalité propre qui est nécessaire au déroulement du jeu. Voici le détail de ces différentes sous-fonctions:

La sous-fonction ('*MatriceEntiers*') permet de créer un tableau d'entiers, rempli de 0. Elle renvoie donc un tableau entier.

→ Cette fonction parcourt le tableau créé par la fonction et place en chaque position un 0. Ici, le zéro est représenté par un vide ou espace dans la matrice.

La sous-fonction ('*SymboleMatrice2*') permet de placer aléatoirement des '2' dans une matrice d'entiers. La fonction renvoie un tableau entier.

→ Elle vérifie tout d'abord si la position choisie aléatoirement au début est disponible. Cela veut dire qu'elle regarde s' il y a un 0 à cet endroit. Si non, alors on choisit à nouveau une position aléatoire pour placer le bonbon. On effectue cette opération n fois jusqu'à qu'on trouve une case qui est vide (une case contenant un 0). Cela est programmé à partir d'une boucle 'while' car on ne sait pas au préalable le nombre de fois qu'on doit tirer au hasard de nouveaux indices jusqu'à trouver une case vide. Cette sous-fonction permet de lancer le jeu en plaçant initialement les deux premiers bonbons dans la grille.

La sous- fonction ('*ConversionMatrice*') permet de convertir une matrice d'entiers en une matrice de caractères ainsi que de mettre à jour le score du joueur. La fonction renvoie un tableau de caractères.

→ La fonction parcourt tout le tableau et en fonction de la valeur (l'entier) que comprend chaque case, il remplace l'entier par un caractère (voir tableau ci-dessus). Cette sous-fonction est programmée à partir des fonctions switch et case qui aident à contrôler les opérations conditionnelles et de créations de branches complexes. L'instruction break permet de mettre fin au traitement d'une instruction étiquetée particulière à l'intérieur de l'instruction switch, et se branche à la fin de l'instruction switch. L'instruction default est vide dans cette sous-fonction car aucune action ne doit être exécutée si le joueur n'utilise pas les bonnes commandes. Dans ce cas, la description des commandes réapparaît.

La sous-fonction ('*AfficherMatrice*') permet d'afficher une matrice de caractères (elle permet d'afficher la grille de jeu en réalité). La fonction renvoie un tableau de caractères.

→ C'est dans cette sous-fonction que nous avons déterminé le design de notre matrice, ou plutôt de notre plateau de jeu, que nous avons souhaité clair et précis. Via une double 'boucle for', la sous-fonction affiche progressivement les séparations horizontales entre chaque case de la grille. La boucle 'for' principale, elle, regroupe l'affichage des lignes de séparation entre chaque ligne de 4 cases. Finalement, cette sous-fonction affiche la grille de jeu dans son ensemble d'une taille 4x4.

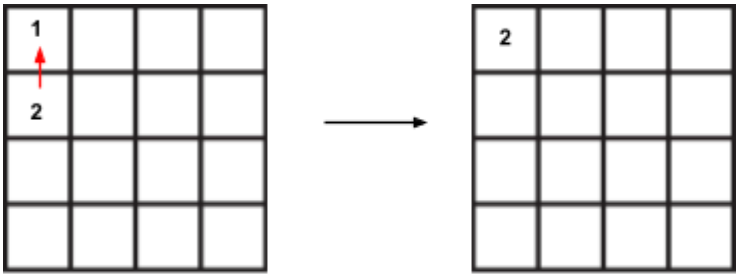
Quatre sous- fonctions permettent le déplacement des bonbons dans la matrice de jeu et permettent de faire: le déplacement *vers le haut* ('*MoveUp*'), le déplacement *vers le bas* ('*MoveDown*'), le déplacement *vers la gauche* ('*MoveLeft*') et le déplacement *vers la droite* ('*MoveRight*').

→ Les quatre fonctions suivent le même principe. Si l'utilisateur décide qu'il veut déplacer le bonbon dans un sens donné, par exemple ici en haut, on va parcourir le tableau (la grille de jeu) dans le sens opposé. Pour la

1			
2			
3			
4			

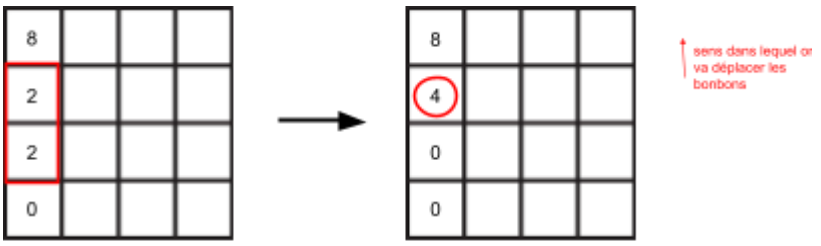
fonction ('*MoveUp*'), on fait donc deux boucles 'for' imbriqués pour parcourir chaque élément de la matrice. Si on imagine la matrice comme étant une grille, on commence le trajet dans la case 1 et ensuite on passe à la case 2, etc. Puis on passe à la colonne suivante et on répète le même processus.

Les comparaisons se font deux à deux. D'abord on vérifie si la case 1 contient un 0. Si cela est le cas, alors on déplace l'entier que contient la case 2 dans la case 1. La case 2 est mise à 0, pour qu'il ne contienne pas de bonbon.



étape 1 *étape 2*

Si la case contient un 2, 4, 8 ou 16 alors on va regarder si la case au-dessus contient la même valeur. Si c'est le cas, alors on le multiplie par 2 et on met cette valeur dans la case placée le plus haut. La case initiale est mise a 0.



exemple de cas de figure

Ces comparaisons sont répétées sur chaque colonne de la grille de jeu.
Elles fonctionnent toutes selon la même logique et le même principe: la seule différence est le sens de parcours de la grille de jeu.

```
// programme qui permet de faire le deplacement en haut
void MoveUp(int[,] tab)
{
    for (int c = 0; c < tab.GetLength(1); c++)
    {
        for (int l = 1; l < tab.GetLength(0); l++)
        {
            if (tab[l, c] != 0)
            {
                int ligne = l;
                while (ligne > 0 && tab[ligne - 1, c] == 0)
                {
                    tab[ligne - 1, c] = tab[ligne, c];          //on déplace le bonbon d'une ligne vers le haut
                    tab[ligne, c] = 0;                          //on met la case initiale du bonbon à 0
                    ligne--;                                    //on met l'indice de la ligne de ou se situe le bonbon actuellement dans
                }

                if (ligne > 0 && tab[ligne - 1, c] == tab[ligne, c] && tab[ligne - 1, c] != 16)
                {
                    tab[ligne - 1, c] *= 2;          //si les deux entiers cote a cote sont egaux et sont différent de 'J', alors on les
                    tab[ligne, c] = 0;              //on met la case précendente à 0
                }
            }
        }
    }
}
```

Exemple de la fonction 'MoveUp' qui permet de déplacer les bonbons vers le haut. Les trois autres fonctions fonctionnent sous le même principe.

La sous-fonction ('*VerificationMatrice*') permet de vérifier si la grille de jeu est remplie et s' il y a un blocage. C'est une fonction qui renvoie un booléen.

→ Pour commencer, une copie est faite de la grille de jeu qu'on va vérifier. La fonction regarde tout d'abord si il y a un 0 dans la grille de jeu, si c'est le cas, alors il n'y a pas de blocage: la fonction renvoie directement *"false"*.
Si il n'y a pas de 0, alors il va essayer les 4 déplacements possibles sur la copie de la grille de jeu et il va le comparer avec la 'vraie' grille. Si ces deux grilles sont égales alors on considère qu'il y a un blocage et la fonction renvoie *"true"*.
Si elles sont différentes, alors la fonction renverra '*false*'.

```
//programme permettant de verifier si la grille est bloquée
bool VerificationMatrice(int[,] tab)
{
    int[,] tab2 = new int[tab.GetLength(0), tab.GetLength(1)];
    tab2 = tab;
    for (int i = 0; i < tab.GetLength(0); i++)  //on verifie si la matrice contient des 0
    {
        for (int j = 0; j < tab.GetLength(1); j++)
        {
            if (tab[i, j] == 0)
            {
                return false;       //si elle contient des 0 alors il n'y a pas de blocage
            }
        }
    }
    MoveDown(tab2);         //sinon on effectue chaque deplacement possible sur une copie de la matrice d'origine
    MoveLeft(tab2);
    MoveRight(tab2);
    MoveUp(tab2);
    if (tab == tab2)        //et on fait une comparaison entre les deux tableaux
    {
        return true;        //si elles sont pareilles alors il y a un blocage
    }
    else
    {
        return false;       //si elles sont différentes alors non
    }
}
```

Choix de modélisation des données

Nous avons décidé de coder le jeu avec plusieurs fonctions implémentées dans un code principal de manière à débugger beaucoup plus rapidement notre programme.
De plus, le fait d'avoir décomposé le travail en plusieurs petites fonctions rend la compréhension du fonctionnement du jeu beaucoup plus claire et permet une lecture plus agréable. Nous comprenons parfaitement et clairement la tâche attribuée à chaque fonction et cela nous permet de comprendre notre code principal dans les détails, et donc de travailler dessus de manière plus efficace. En cas de problèmes dans une partie du jeu, nous pouvons déterminer quelle fonction est concernée par le problème en fonction du problème

d'exécution. Par exemple, dans le cas d'un problème de déplacement vers la droite, nous saurons qu'il y a un problème à régler dans la fonction 'MoveRight'.

III. Organisation du projet

Partie codage

En ce qui concerne le codage de notre programme, nous avons beaucoup travaillé en binôme, notamment pour créer chaque sous-fonction du programme. Ce travail a été permis grâce aux 3 séances de 2h40 prévues à cet effet, et au cours desquelles nous avons mis au point l'organisation de notre projet puis sa conception. Chaque membre y a contribué à hauteur de ses connaissances et compétences particulières, pour arriver à un ensemble harmonieux et efficace. Dès le fonctionnement global de notre programme, chacun des membres du duo a pu travailler à améliorer le code via des branches différentes et personnelles créées sur GitHub. Cette plateforme nous a permis de poursuivre ce travail de groupe à distance tout en nous focalisant personnellement sur des points précis du programme. La répartition des tâches s'est faite très naturellement, chacune s'attachant à travailler les parties du programme qu'elle jugeait imprécises ou défaillantes.

Matrice d'implication

Matrice d'implication: Projet Informatique XMAS Treats - 15/12/2023			TOTAUX
Tâches	Kami	Alexia	
PROGRAMME PRINCIPAL			
Programme principal	**70**	**30**	**100**
SOUS-FONCTIONS			
MatriceEntiers	**50**	**50**	**100**
SymboleMatrice2	**50**	**50**	**100**
ConversionMatrice	**60**	**40**	**100**
AfficherMatrice	**50**	**50**	**100**
MoveUp	**70**	**30**	**100**
MoveDown	**70**	**30**	**100**
MoveLeft	**70**	**30**	**100**
MoveRight	**70**	**30**	**100**
VerificationMatrice	**75**	**25**	**100**
TÂCHES ANNEXES			
Commentaires	**50**	**50**	**100**
Consignes utilisateurs	**25**	**75**	**100**
Design affichage du code	**25**	**75**	**100**

JUSTIFICATIF TECHNIQUE			
Justificatif technique	**25**	**75**	**100**

Planning de réalisation

Nous avons eu un mois pour réaliser ce projet, au cours duquel nous avons pu travailler de manière encadrée par nos professeurs à trois reprises. Ces créneaux nous ont permis un travail de groupe efficace notamment pour lancer le projet. Le reste du temps, nous avons travaillé ensemble via la plateforme GitHub et ce de manière régulière jusqu'à la remise du projet. En terminant le travail principal de mise au point de notre code assez rapidement, nous avons pu prendre le temps à le perfectionner et à régler les petites défaillances de l'ensemble.

Planning de réalisation Projet Xmas Treats	
Semaine 1	-Mise au point du projet et mise en route: création de nos comptes GitHub et familiarisation avec le sujet.
Semaine 2	-Vendredi 24 novembre: cours encadré, début de la réflexion sur le code et début de la programmation -Puis travail de groupe à la réalisation du code
Semaine 3	-Mercredi 29 novembre: cours encadré, amélioration du code déjà bien avancé -Travail individuel de perfectionnement du programme
Semaine 4	-Mercredi 6 décembre: cours encadré, mise à jour de notre code -Travail individuel de finalisation du code -Réalisation du Justificatif technique de manière individuelle mais collaborative

15-12-2023	-Finalisation du code et du justificatif technique et envoie des livrables attendus