Petit Jade, Thomasson Lucie
Ecole Nationale Supérieure de Cognitique (Bordeaux) - 2019/2020

Projet informatique Motus

Justifications techniques

Table des matières

Contexte

Fonctionnement du jeu Motus

Le jeu Motus consiste à deviner un mot avec pour seul indice sa première lettre (pas de mot composé ou de nom propre). Le joueur dispose d'un nombre limité de tentatives afin de trouver le mot.
Le joueur commence par proposer un mot puis, pour chaque essai, doit deviner les lettres qui le composent grâce au code couleur suivant :

- ➢ Une lettre colorée en ROUGE signifie que celle-ci est bien placée
- ➢ Une lettre colorée en JAUNE signifie qu'elle est présente dans le mot, mais qu'elle n'est pas bien placée
- ➢ Une lettre colorée en BLEU signifie qu'elle ne fait pas partie du mot

Si le joueur parvient à trouver le mot sans épuiser toutes ses tentatives, il gagne.

Contraintes liées au programme

La programmation est faite en C#, en mode console, sous Windows, sans utilisation de bibliothèques de fonctions externes autre que les bibliothèques standards du framework .NET. La programmation orientée objet ainsi que l'utilisation de listes ne sont pas autorisés.
Le code doit être commenté et respecter la convention de nommage camelCase.

Fonctionnement du programme

Programmes annexes pour la création du dictionnaire

Le programme doit s'appuyer sur un dictionnaire valide pour fonctionner. Un dictionnaire est considéré comme valide s'il ne contient pas de noms propres, de mots n'existant pas dans la langue française ou de mots composés.

Afin de pouvoir utiliser le dictionnaire dans notre programme, nous avons créé deux petits programmes : *Decoupage_dico* et *Filtre_mots_composes_dico*.
Filtre_mots_composes_dico s'occupe d'enlever les mots composés, c'est à dire les mots avec des "-", car ils ne sont pas considérés comme valide dans le jeu Motus.
Decoupage_dico nous a permis de découper le dictionnaire principal en plusieurs sous-dictionnaires qui contiennent chacun des mots d'une certaine longueur, allant de 6 à 10. Le programme principal fait appel à l'un de ces petits dictionnaires en fonction de la difficulté choisie par l'utilisateur.

Pour créer ces deux programmes, nous nous sommes basées sur les instances *StreamReader* et *StreamWriter*. Nous sommes parties du dictionnaire donné pour le projet et nous avons réécrit les sous-dictionnaires dont nous avions besoin.

Ces deux programmes ont été lancés au préalable afin que les dictionnaires finaux soient prêts à être utilisés. Il n'est donc pas nécessaire de les lancer pour faire fonctionner Motus.

Fonctions et procédures

Fonction/Procédure	Type en entrée	Type en sortie	Variables locales	Rôle
InitialiserNombreLettres	/	Un entier	nbLettres (entier)	Demande à l'utilisateur un nombre de lettres pour le mot à deviner et le renvoie en sortie
InitialiserNombreTentatives	/	Un entier	nbTentatives (entier)	Demande à l'utilisateur un nombre de tentatives pour deviner le mot et le renvoie en sortie
RemplirLigne	Un tableau à deux dimensions contenant des caractères, un entier correspondant au numéro de la ligne à remplir, une chaîne de caractère	/	nbColonnes (entier)	Remplit la ligne demandée du tableau avec le mot tapé par l'utilisateur
CreationTableau	Deux entiers correspondant au nombre de lignes et de colonnes	Un tableau à deux dimensions	/	Créé un tableau vide pouvant contenir des caractères
GenererMot	un entier correspondant au nombre de lettres du mot à deviner	Une chaîne de caractère	rangDansLeDico (entier) mot (chaîne de caractères) nombresMotDico (entier)	Génère un mot aléatoire à partir du dictionnaire réduit
AfficherTableau	Un tableau à deux dimensions contenant des caractères, une chaîne de caractères correspondant au mot à deviner	/	nbLignes (entier) nbColonnes (entier)	Affiche un tableau à 2 dimensions contenant des caractère et gère la couleur. La fonction change le fond d'affichage d'une lettre en fonction du jeu.
VerifierMot	Un tableau à deux dimensions contenant des caractères, une chaîne de caractère contenant le mot à vérifier	Un booléen	nbLignes (entier) nbColonnes (entier) cpt (entier) check (booléen)	Renvoie true si le bon mot a été trouvé
AfficherHistorique	/	/	score (chaîne de caractères)	Fonction qui affiche l'historique des scores depuis un fichier texte, en le lisant ligne par ligne

La fonction Main

Variables

Nom de la variable	Type	Rôle de la variable
trouvé	booléen	Variable qui prend la valeur true si l'utilisateur a trouvé le bon mot et false sinon
menu	entier	Variable qui prend la valeur 1 ou 2 en fonction du choix de l'utilisateur de commencer le jeu ou de consulter l'historique
rejouer	booléen	Variable qui permet à l'utilisateur de pouvoir rejouer ou non
parcoursScore	entier	Compteur qui va parcourir le tableau des scores
nbLignes nbColonnes nbLettres nbTentatives	entiers	Nombres de lignes et de colonnes du tableau de jeu. Le nombre de lettres du mot est égale au nombre de colonnes du tableau et le nombre de tentatives pour deviner le mot au nombre de lignes.
motAdeviner	chaîne de caractères	Variable qui contient le mot à deviner
ligneAremplir	entier	Numéro de la ligne en cours de remplissage dans le tableau
motDonné	chaîne de caractères	Variable qui contient le mot entré par l'utilisateur lorsqu'il essaie de deviner le mot
nom	chaîne de caractères	Variable qui contient un nom entré par l'utilisateur lorsque celui-ci veut inscrire son score dans l'historique

Structure

Une première variable *menu* permet au joueur de choisir s'il lance le jeu ou s'il regarde l'historique des scores.

```
Console.WriteLine("HISOTIQUE DES SCORES : 1");
Console.WriteLine("COMMENCER LE JEU : 2");

//Variable qui prend la valeur 1 si l'utilisateur veut consulter l'historique et 2 s'il veut jouer
int menu = int.Parse(Console.ReadLine());

if(menu == 1)
{
    voirHistorique = true;
}
else
{
    if(menu == 2)
    {
        rejouer = true;
    }
    else
    {
        Console.WriteLine("Merci de taper 1 ou 2");
    }
}
```

La structure du *Main* est basée sur deux grandes boucles *while* qui gèrent l'avancement du jeu. L'une relance le jeu tant que l'utilisateur décide de rejouer, à l'aide d'une variable booléenne *rejouer*, tandis que l'autre fait tourner la création des tableaux contenant les mots entrés par l'utilisateur tant que la partie en cours n'est pas terminée. Un chronomètre se lance lorsque l'utilisateur a la possibilité d'écrire son premier mot.

Les boucles *while* contiennent essentiellement des appels aux fonctions et aux procédures décrites précédemment. On vérifie juste que les mots entrés par l'utilisateur ont la bonne longueur.

L'enregistrement des scores se fait dans le *Main*.

```
using (System.IO.StreamWriter monStreamWriter = new System.IO.StreamWriter("Historique_score.txt", true))
{
    monStreamWriter.WriteLine(score);
    monStreamWriter.Close();
}
```

Le fichier texte est créé à la première lecture du programme. Ensuite, à chaque itération, *StreamWriter* complète le fichier avec le même nom s'il existe déjà.
Après chaque entrée d'un nouveau score, le fichier texte est refermée.

On aurait aussi pu créer une fonction qui s'occupe de l'enregistrement. Dans le cas où le joueur gagne, on lui propose d'inscrire son nom dans un tableau de scores, où sera également consigné son temps de jeu. L'inscription du score se fait grâce à une instance *StreamWriter*.

Qu'il gagne ou qu'il perde, le joueur se voit proposé de rejouer ou de quitter le jeu grâce à une variable booléenne.

Si l'utilisateur décide de voir l'historique des scores, on a un appel à la fonction *AfficherHistorique* qui lit les scores depuis un fichier .txt.

Fonctionnalités supplémentaires

Chronomètre

Nous avons ajouté au programme un chronomètre qui mesure le temps écoulé lors d'une partie. Pour cela, nous avons utilisé la classe *Stopwatch* avec les instructions suivantes :

```
//Initialisation du chronomètre
Stopwatch stopWatch = new Stopwatch();
stopWatch.Start();
```

```
//Fin du chrnomètre
stopWatch.Stop();
```

```
//Intervalle de temps du chronomètre
TimeSpan ts = stopWatch.Elapsed;
```

Les instructions *.Start* et *.Stop* permettent de lancer puis d'arrêter le chronomètre. *StopWatch.Elapsed* permet d'obtenir le temps écoulé entre ces deux instructions. Ensuite, on convertit ce temps en heures, minutes et secondes pour afficher une durée totale compréhensible :

```
string elapsedTime = String.Format("{0:00} heure(s) {1:00} minute(s) {2:00} seconde(s)", ts.Hours, ts.Minutes, ts.Seconds);
Console.WriteLine("La durée de la partie est de " + elapsedTime);
```

Historique des parties

L'historique des parties est contenu dans un fichier texte. La lecture de ce fichier se fait grâce à la procédure *AfficherHistorique*. Une instance *StreamReader* lit chaque ligne du fichier texte et l'affiche simplement. Nous nous sommes aidées des programmes qui s'occupent du dictionnaire pour créer cette procédure.

```
public static void AfficherHistorique()
{
    System.Text.Encoding encoding = System.Text.Encoding.GetEncoding("iso-8859-1");
    StreamReader monStreamReader = new StreamReader("Historique_score.txt", encoding);

    string score = monStreamReader.ReadLine();

    //Lecture de tous les mots du fichier (un par ligne)
    while (score != null)
    {
        Console.WriteLine(score); //On affiche le score contenu dans une ligne
        score = monStreamReader.ReadLine(); //On passe à la ligne suivante
    }
```

Problèmes rencontrés

Code couleur

Notre principal problème vient du code couleur des lettres durant la partie.
En effet, une lettre surlignée en rouge veut dire qu'elle est correctement placée dans le mot, tandis qu'une lettre surlignée en jaune veut dire que celle-ci apparaît bien dans le mot, mais qu'elle n'est pas à la bonne place.

Prenons l'exemple du mot à deviner **ECHAFAUD**. La première lettre de ce mot n'est présente qu'une fois. Elle sera donc surlignée en rouge dès le début. Cependant, si l'utilisateur entre un mot contenant deux fois la lettre **E** (**ECHARDES**), le programme ne déduit pas que le deuxième **E** n'existe pas dans le mot. Il le surlignera donc en jaune, suggérant au joueur que le mot contient une deuxième fois cette lettre, alors que ce n'est pas le cas.
Nous n'avons, à ce jour, pas réussi à corriger cela.

Pistes d'amélioration

Temps de réponse

Dans le jeu original, les candidats ont 8 secondes pour proposer un mot. S'ils ne vont pas assez vite, ils perdent la main et c'est à l'équipe suivante de répondre. Nous aurions pu implémenter un temps limite de réponse pour le candidat. Cela aurait été plus intéressant qu'un simple chronomètre. Mais cela suppose que le jeu ait un mode deux joueurs.

Mode deux joueurs

Il aurait été intéressant d'avoir la possibilité d'un mode deux joueurs, puisque le vrai jeu se joue avec deux équipes qui s'affrontent. Dans ce cas, les joueurs proposeraient chacun à leur tour :

- ➢ Soit un mot valide jusqu'à victoire d'un des deux utilisateurs
- ➢ Soit un mot valide jusqu'à ce que leur nombre de tentatives soit épuisé, comme c'est le cas dans le jeu original.

Vérification de la validité des mots entrés par l'utilisateur

Dans le jeu original, les candidats doivent épeler le mot qu'ils proposent. Si jamais le mot épelé contient une faute d'orthographe, ou que le nombre de lettres ne correspond pas au nombre de lettres du mot recherché, l'équipe perd la main.

Dans notre version, les joueurs peuvent écrire un mot non valide. Pour les obliger à écrire un mot valide, il faudrait que le mot écrit corresponde bien à un mot du dictionnaire. Mais cela supposerait un dictionnaire de vérification des mots très conséquent.
De plus, cela pourrait prendre du temps au programme de comparer le mot écrit par l'utilisateur avec tous les mots du dictionnaire pour voir si le celui-ci existe.
Nous avons donc décidé de ne pas implémenter cette vérification.