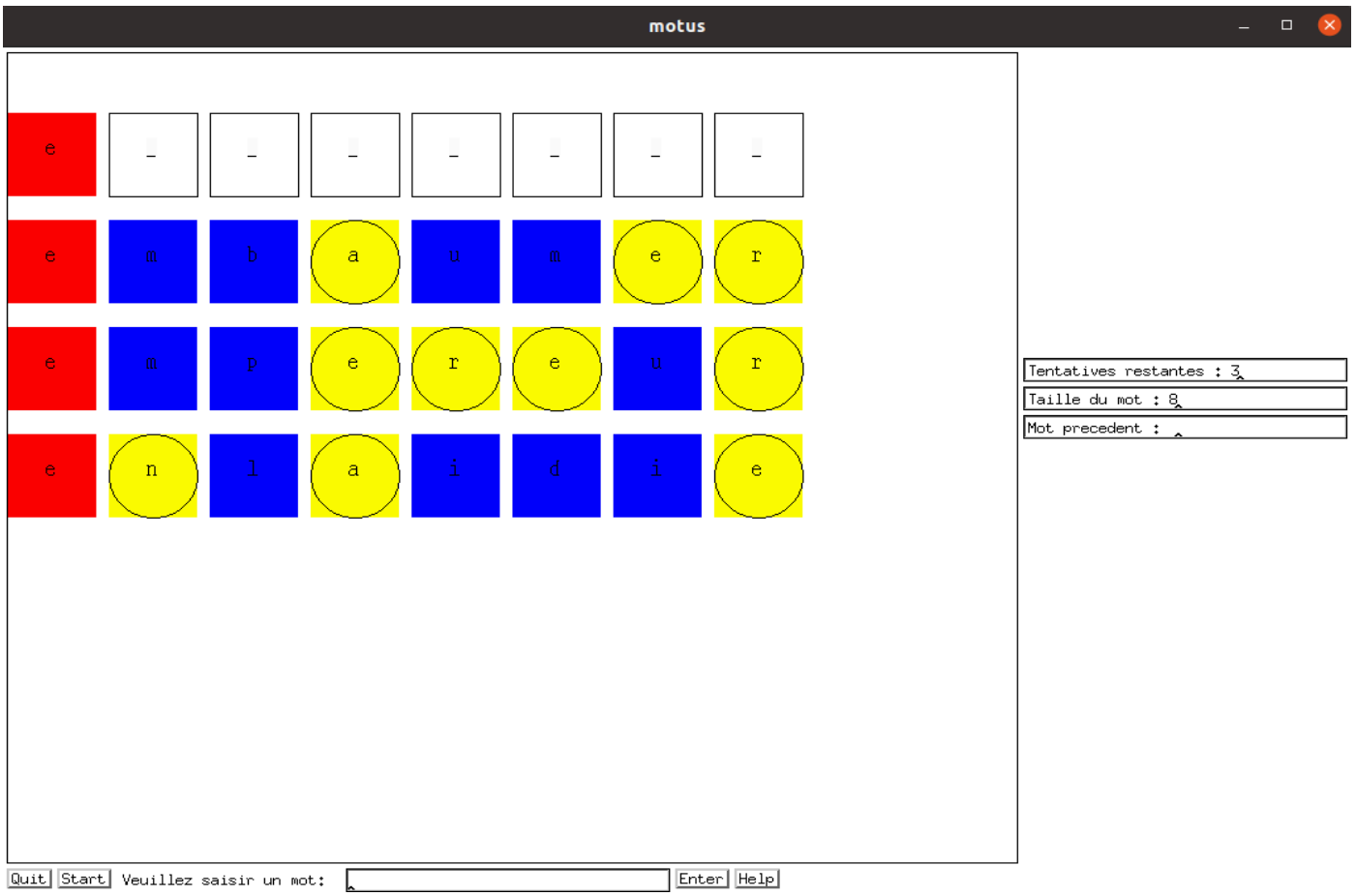
Motus

Projet final de programmation en C : Jeu Motus



Auteurs: Mohamed Redha ABDERRAHMANE & Fouad BOULOUIZ

Durée du projet: Fin avril au 9 juin 2024

Sommaire :

Introduction:

L'objectif de ce projet est de se familiariser avec le développement des programmes à plusieurs fichiers dépendant les uns des autres et l'implémentation d'une interface graphique en utilisant la bibliothèque libsx.
Pour cela nous allons développer notre propre version du jeu Motus en utilisant le langage de programmation C.

Motus est un célèbre jeu télévisé où les joueurs doivent deviner un mot caché en se basant seulement sur la première lettre du mot et sa longueur.
Les joueurs ont un nombre limité de tentatives pour deviner le mot.

Étapes de développement:

Dans un premier temps nous avons créé la logique du jeu pour tester d'une manière textuelle le programme en interagissant avec l'utilisateur à l'aide de la sortie standard pour vérifier le fonctionnement du code.

Dans un deuxième temps, nous avons implémenté l'interface graphique à l'aide de la bibliothèque libsx qui permet l'interaction de l'utilisateur avec le programme grâce à une fenêtre dédiée. Pour cela, nous avons adapté les fonctions du jeu pour pouvoir laisser le joueur déclencher les fonctionnalités du programme.

L'interface graphique nous accorde une certaine liberté pour afficher des dessins qui représente l'état d'avancement du joueur, du texte pour indiquer au joueur certaines informations et la mise en place des boutons qui déclenchent le fonctionnement du jeu.

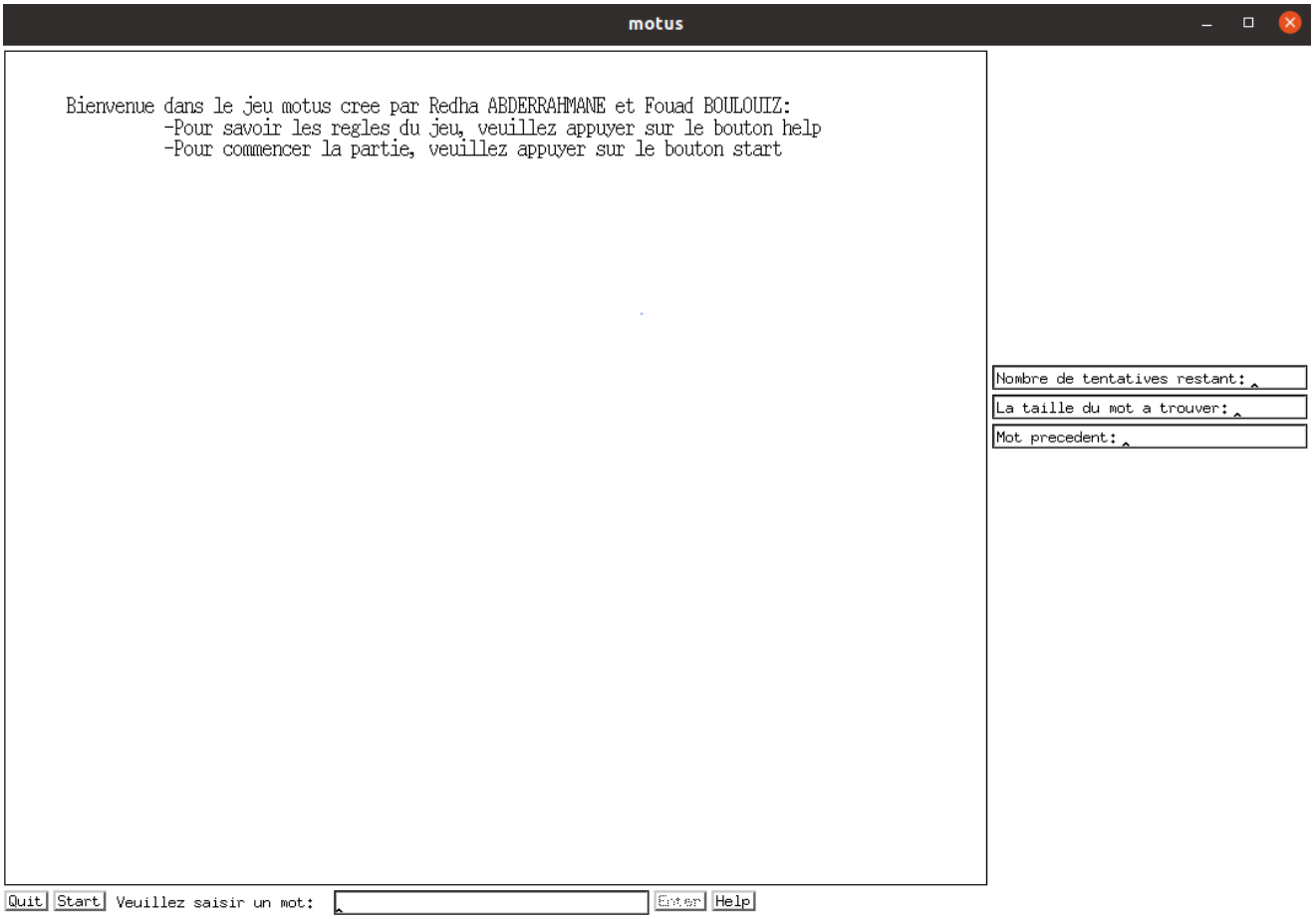
Comment lancer le jeu:

Pour lancer le jeu, veuillez ouvrir un terminal dans le dossier où se trouve le fichier "motus" et le fichier textuel "DicoMotus.txt".

Dans le terminal veuillez entrer la commande suivante: ./motus

Attention: la présence du fichier textuel "DicoMotus.txt" est **cruciale** pour le fonctionnement du jeu.
Sans le fichier **"DicoMotus.txt"** le programme **ne fonctionnera pas**.

Fonctionnalités du programme :

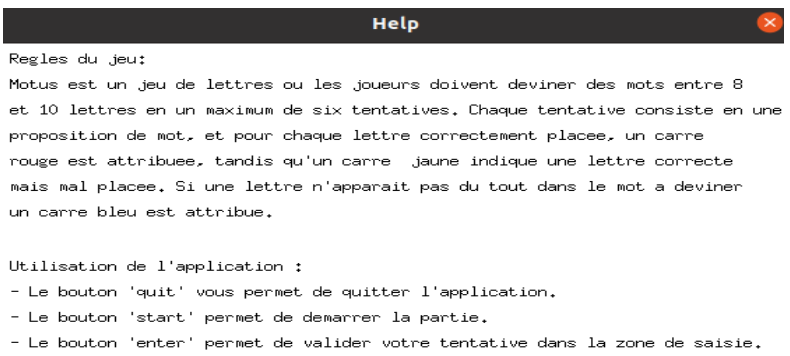


Le jeu dispose d'une interface graphique avec plusieurs boutons (Start,Help…) permettant de communiquer avec l'utilisateur.

Lorsqu'on lance le programme une fenêtre s'affiche avec un message d'accueil qui explique comment démarrer le jeu et obtenir plus d'informations si nécessaire.

<u>Boutons:</u>

-Bouton "Help": Lance une fenêtre de texte expliquant les règles du jeu et le fonctionnement des boutons.



-Bouton "Start": Permet de lancer une partie du jeu motus et afficher sur l'interface le mot caché.

-Bouton "Quit": Permet de quitter le programme.

Différentes options du jeu:

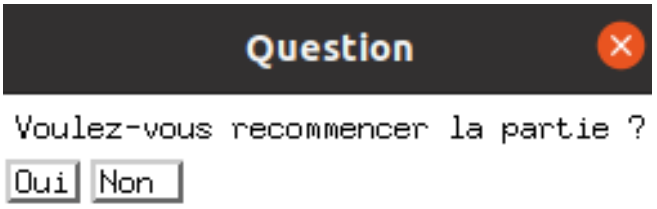
Il dispose également d'un champ de saisie du mot à trouver et de "sous fenêtres" permettant d'afficher les informations suivantes : la taille du mot à trouver, le nombre de tentatives restantes et le mot précédemment entré par l'utilisateur (le mot affiché sera toujours de la bonne taille, les mots invalides sont représentés par le mot "INVALIDE").

Indications au cours du jeu:

Au cours des tentatives, les mots valides tapés précédemment sont affichés sur l'interface. Plusieurs couleurs sur les lettres donnent des indices pour trouver le mot. La couleur rouge indique que la lettre est dans le mot et à la bonne place. La couleur jaune indique que la lettre est aussi dans le mot, mais pas à la bonne place. La couleur bleue indique que la lettre n'est pas dans le mot.

Messages du jeu en fonction de l'issue:

Après être arrivé au bout des tentatives, le jeu affiche un message de fin de partie, et propose à l'utilisateur de refaire une partie. L'utilisateur peut choisir de recommencer la partie, ou quitter le jeu.



Choix de la structure de données:

Dans le programme, nous utilisons la structure "Donnees" qui contient plusieurs composantes qui vont être accédées par les procédures de l'interface graphique notamment les chaînes de caractères qui représentent:

- La chaîne de caractères "motOriginal" qui représente le mot choisi aléatoirement dans le dictionnaire.
- La chaîne de caractères "motFloute" qui représente le mot caché qui indique la première lettre du mot original et sa taille.
- La chaîne de caractères "motProp" qui représente le mot proposé par l'utilisateur.
- La chaîne de caractères "motPrec" qui représente le mot précédent saisi par l'utilisateur.

Dans la structure nous trouverons aussi:
-L'entier "tentative" qui représente le nombre de tentatives de la partie qui va être incrémenté après chaque saisie.
-L'entier "taille" qui représente la taille du mot original.

Nous avons décidé d'implémenter ces 6 composantes dans la structure car elles sont indispensables au fonctionnement du jeu.
La structure "Donnees" facilite l'accès à ses composantes pour l'interface graphique grâce aux différentes fonctions qui renvoient les chaînes de caractères demandés et les entiers demandés
Exemple:les fonctions ReturnMotOriginal et ReturnTaille qui renvoient respectivement le mot original et la taille du mot original.

Il faut noter que le nombre de tentatives et la taille maximale du mot sont définis dans les variables globales TENTATIVES = 6 et MAXLENGTH = 11 (taille maximale d'un mot + 1) respectivement ce qui permet de les modifier facilement.
Cependant l'interface est conçue pour afficher six mots dans la fenêtre d'affichage ce qui veut dire que le nombre de tentatives ne doit pas dépasser 6.

Fonctions et procédures importantes:

-**La fonction "Occurence"** prend un tableau d'entiers short et un caractère en paramètres.
Dans un premier temps, elle remplit chaque case du tableau par -1 puis elle boucle sur le mot original et identifie la position du caractère dans le mot. La fonction enregistre donc les positions de la lettre dans le tableau et le renvoie.

-La procédure "remplacer" permet de remplacer un caractère dans le mot caché par 'x', 'O' ou '-' en fonction de la présence du caractère dans le mot original.
le caractère à mettre dans le mot caché et sa position est passé par les paramètres.

-**La procédure "VerifierOccurence"** permet de comparer les caractères du mot saisi, et du mot original, un à un.
Si le caractère est au bon endroit, elle appelle la procédure "remplacer" avec le caractère 'x' et l'indice du caractère dans les paramètres.

Si un des caractères ne correspond pas, elle appelle la fonction "Occurence" pour obtenir le tableau des indices afin de savoir si le caractère existe dans le mot original.

Cas 1:
La première valeur dans le tableau d'indices est égale à -1, alors le caractère n'existe pas dans le mot original. Donc la procédure "remplacer" est appelée avec le caractère '-' et l'indice du caractère dans les paramètres.

Cas 2:
La première valeur dans le tableau d'indices est différente de -1, alors le caractère existe dans le mot original. Donc la procédure "remplacer" est appelée avec le caractère 'O' et l'indice du caractère dans les paramètres.

-**La fonction "RecevoirMot"** permet de recevoir le mot proposé par l'utilisateur, l'affecter à la composante qui représente le mot proposé dans la structure puis vérifie la présence du mot dans le dictionnaire. Elle renvoie un booléen pour indiquer la validité du mot.

-**La procédure "initialise_Mots"** permet l'initialisation des composantes de la structure "Données" au début de la partie.

Dans un premier temps, un mot choisi aléatoirement dans le dictionnaire sera affecté à la chaîne de caractères "motOriginal" de la structure. L'entier "taille" prend la valeur de la longueur du mot choisi.

Dans un deuxième temps, la chaîne de caractères "motOriginal" sera copiée à la chaîne de caractères "motFloute" et sera cachée à l'aide de la procédure "Flouter" qui remplace les caractères par "_" sauf pour le premier caractère.
On affecte également une copie de "motFloute" à "motPrec" pour dessiner le mot caché sur l'interface graphique en cas d'une première saisie fausse par l'utilisateur.

Enfin, le nombre de tentatives sera initialisé à la valeur affectée à la variable global "TENTATIVES".

-**La procédure "enter"** permet de recevoir le mot saisi de l'utilisateur dans la zone de texte lorsque l'utilisateur appuie sur le bouton "Enter" et d'effectuer le traitement grâce aux fonctions et procédures expliquées précédemment et appelle les procédures de dessin pour mettre à jour l'interface en fonction du mot caché.
Si le mot est valide, une procédure d'affectation est appelée pour affecter à "motPrec" une copie du mot proposé par l'utilisateur.

Algorithme du jeu:

Lorsque le joueur commence une partie en appuyant sur le bouton "Start", les composantes de la structure "Donnees" seront initialisées grâce à la procédure "initialise_Mots". L'interface graphique affichera donc le mot caché grâce aux procédures de dessin non détaillées dans le rapport.

Lorsque l'utilisateur envoie le mot saisi avec le bouton "Enter" , le mot saisi sera vérifié à l'aide des fonctions et procédures expliquées dans la partie précédente et l'interface graphique affichera le résultat.
En cas de victoire ou de défaite, le choix du joueur détermine si le programme commence une nouvelle partie ou quitte le jeu immédiatement.

Remarques sur l'interface graphique:

Pendant le développement du projet, nous avons remarqué plusieurs erreurs :

-Erreur de type segmentation fault (core dumped) lors de l'exécution du jeu avec l'interface graphique. En utilisant le débugueur gdb lors de l'exécution du jeu, nous avons trouvé que la source de ce problème est les fonctions "GetRGBColor" et "GetNamedColor".

Ces deux fonctions sont fournies par la bibliothèque libsx pour renvoyer un entier qui représente la couleur voulue.
Cette erreur est irréparable et peut se produire de manière aléatoire lors de l'exécution du programme.
Pour contourner cette erreur, nous avons cherché à comprendre comment une couleur RGB(RGB8 précisément) est codée :
Une couleur RGB est composée de 3 composantes R : rouge, G : vert, B : bleu.
Chaque composante est codée sur 8 bits. Pour représenter une couleur RGB, on doit assurer que chaque composante est codée sur 8 bits, soit un maximum de 255.

Les 8 bits de poids fort de la couleur représentent la composante rouge, donc nous devons faire un décalage à gauche de 16 bits.
Les 8 bits du milieu de la couleur représentent la composante verte, donc nous devons faire un décalage à gauche de 8 bits.
Enfin, les 8 bits de poids faible de la couleur représentent la composante bleue, donc aucun décalage n'est nécessaire.

Pour assurer la stabilité de notre programme, nous avons décidé de créer une redéfinition de "GetRGBColor" que nous avons définie par "Get_RGBColor" pour assurer la stabilité du système.

-Lorsque l'utilisateur essaie de mettre la fenêtre en plein écran, le programme se comporte de manière inattendue et non prévue dans le code. Par exemple, cela peut provoquer le redémarrage du jeu vers l'écran d'accueil.

-Le texte saisi par l'utilisateur devient invisible dans la zone de saisie pour certains utilisateurs ayant des petits écrans.

Conclusion:

Ce projet nous a permis de nous familiariser avec le développement de programmes multi-fichiers interdépendants et l'implémentation d'une interface graphique en utilisant la bibliothèque libsx. Notre programme est passé par plusieurs étapes pour aboutir à la version finale que nous avons rendue. Dans un premier temps, nous avons commencé par la version textuelle pour valider la logique du jeu. La transition vers l'interface graphique a requis une grande modification du programme de base pour intégrer la logique du jeu avec l'interface graphique. Une grande partie du travail consistait à éviter les erreurs et à assurer la stabilité de l'interface graphique lors du fonctionnement du jeu. La redéfinition de la fonction GetRGBColor a assuré la stabilité de notre programme.

Cependant, il existe plusieurs pistes d'amélioration dans notre programme :

-Ajouter plusieurs modes de difficulté dans le jeu.

-Redimensionner la taille des widgets lorsque l'utilisateur modifie la taille de la fenêtre d'affichage.
Nous avons aussi appris que les bibliothèques dans les langages de programmation sont des outils qui peuvent faciliter l'implémentation de plusieurs concepts (comme les interfaces graphiques dans notre cas). Cependant, certaines fonctionnalités peuvent ne pas être optimisées pour toutes les applications, donc en tant que programmeurs, nous devons nous adapter pour assurer le bon fonctionnement du programme codé.