Projet : Jeu du Pendu en C

L'objectif de cet exercice est de programmer un jeu du pendu en langage C.

Spécifications :

1. Un mot mystère est défini.
2. L'utilisateur doit deviner les lettres du mot, avec un nombre limité d'erreurs.
3. À chaque tentative, afficher le mot avec les lettres correctement devinées et les emplacements des lettres manquantes sous forme de tirets.
4. Afficher les lettres incorrectes déjà essayées.
5. Indiquer à l'utilisateur s'il a gagné (mot entièrement deviné) ou perdu (toutes les tentatives épuisées).

Étape 1 : Jeu du pendu à deux joueurs.

L'objectif de cette étape est de programmer une première version à deux joueurs. Le joueur 1 choisit le mot mystère et le nombre maximal d'erreurs (**N**), le joueur 2 doit deviner le mot mystère avant d'atteindre le nombre maximal d'erreurs (**N**) pour gagner. Le mot mystère ainsi que du nombre maximal d'erreurs (**N**) sont communiqués lors de l'exécution du programme **en utilisant la ligne de commande**.

Pour mettre en place le programme, nous pouvons utiliser les fonctions suivantes :

1. **verifier_tentative** : cette fonction sert à vérifier si le caractère saisi par le joueur 2 appartient au mot mystère. Elle prend en entrée un caractère **c** (deviné par l'utilisateur), ainsi qu'une chaîne de caractères **mot** (mot mystère). Elle renvoie ***vrai*** si le caractère **c** appartient à **mot**, sinon elle renvoie ***faux***.
2. **maj_mot_devine** : cette fonction sert à mettre à jour le mot deviné par l'utilisateur, en remplaçant les tirets correspondants au caractère choisi par l'utilisateur **(dans le cas où il appartient au mot mystère).** Elle prend en entrée deux chaînes de caractères **mot_devine** et **mot_mystere** et renvoie une chaîne de caractère.
3. **afficher_mot_devine** : cette fonction sert à afficher le mot deviné par l'utilisateur, en remplaçant les caractères inconnus par des tirets '_'. Elle prend en entrée une chaîne de caractères et ne renvoie rien.

4. **afficher_progression** : cette fonction sert à afficher la progression du joueur 2, elle doit afficher le nombre d'essais restant, la liste des caractères incorrects déjà essayés ainsi que le mot deviné (appel à la fonction **afficher_mot_devine**).

Remarques :

- Pensez à transformer tous les caractères en minuscule en utilisant la fonction **tolower()**
- Pensez à tester vos fonctions individuellement et au fur et à mesure de leurs implémentations.
- Garder trace des caractères incorrects saisies par le joueurs 2, vous pouvez utiliser un tableau de taille **N** (équivalent au nombre maximal d'erreurs).
- Dans le cas où le joueur 2 choisit un caractère incorrect déjà essayé, **rien ne se passe (ceci ne compte pas comme une erreur)**.
- Vous pouvez utiliser la fonction **afficher_pendu** disponible ci-dessous pour afficher l'ASCII art en fonction du nombre d'erreurs.

```c
// Fonction pour afficher l'ASCII art en fonction du nombre d'erreurs
void afficher_pendu(int erreurs_actuelles, int erreurs_max) {
    // Calcul du taux d'erreur en pourcentage
    int taux_erreur = (erreurs_actuelles * 100) / erreurs_max;
    // Ajout de la barre supérieure (30%)
    if (taux_erreur >= 30) {
        printf("   _______\n");  // Barre supérieure (transversale)
    }
    // Ajout de la corde (40%)
    if (taux_erreur >= 40) {
        printf("  |/      |\n");  // Corde accrochée
    } else if (taux_erreur >= 30) {
        printf("  |/      \n");  // Barre sans corde
    }

    if (taux_erreur>=50){
         printf("  |      (_)\n");  // Tête du pendu
    }
    else if(taux_erreur>=20){
        printf("  |\n");  // Pilier complet (à 20%)
    }

    // Ajout des bras (60%)
    if (taux_erreur >= 60) {
        printf("  |      \\|/\n");  // Bras complets
    } else if (taux_erreur >= 20) {
        printf("  |\n");  // Pas de bras, mais espace sous le torse
```

```
    }

    // Ajout du torse (80%)
    if (taux_erreur >= 80) {
        printf("  |       |\n");  // Torse
    } else if (taux_erreur >= 20) {
        printf("  |\n");  // Pas de torse, mais espace sous la tête
    }

    // Ajout des jambes (100%)
    if (taux_erreur >= 100) {
        printf("  |      / \\\n");  // Jambes complètes
    } else if (taux_erreur >= 10) {
        printf("  |\n");  // Pas de jambes, mais espace sous les bras
    }


    if (taux_erreur>=10){// Affichage de la base du pilier
        printf("__|__\n");
    }
}
```

Étape 2 : Version à un seul joueur.

Dans cette étape, nous allons améliorer notre jeu en permettant au programme de choisir aléatoirement le mot mystère (depuis une liste de mots prédéfinie) ainsi que le nombre maximal d'erreurs (**N**).

Pour mettre en place cette nouvelle version, nous pouvons rajouter les fonctions suivantes :

1. **choisir_N_erreurs** : cette fonction sert à choisir aléatoirement le nombre d'erreurs autorisées. Elle ne prend aucun paramètre en entrée et renvoie un entier aléatoire compris entre **8** et **12**.
2. **choisir_mot_mystere**: cette fonction sert à choisir aléatoirement le mot mystère. Elle prend en entrée une liste de mots (**char** liste_mots**) ainsi que le nombre de mot (**int n**), et renvoie en sortie un mot choisi aléatoirement depuis **liste_mots.**

Remarque : Il est possible de définir une liste de mots de la manière suivante :

```
char* liste_mots[] = { "bonjour", "maison", "papillon", "artisan",
"ruelle", "ordinateur", "chocolat", "fromage", "jardin", "etoile",
"nuages", "cyclone", "robotique", "escargot", "chameau", "probleme",
"terrasse", "bonsoir", "biscuit", "catastrophe" };
int nb_mots = 20; //Nombre de mots dans liste_mots.
```

Étape 3 : Version sans joueur.

L'objectif de cette étape est de rendre le programme autonome. Dans un premier temps, il devra choisir aléatoirement le mot mystère ainsi que le nombre maximal d'erreurs (**N**) (voir étape 2). Par la suite, il devra essayer de gagner en devinant le mot mystère, pour cela, nous allons devoir rajouter une nouvelle fonction :

- **choisir_caractere**: cette fonction sert à choisir aléatoirement un caractère. Elle ne prend aucun paramètre en entrée et renvoie en sortie un caractère choisi aléatoirement**.**

Étape 4 : IA comme joueur.

L'objectif de cette étape est de remplacer le choix aléatoire de caractères par un modèle d'IA apte à faire des choix plus judicieux, en utilisant une chaîne de Makrov.

Une chaîne de Markov appliquée à des caractères modélise la probabilité de transition d'un caractère à un autre, permettant ainsi de générer des séquences de caractères en se basant sur des états (caractères) précédents.

L'idée est de calculer la probabilité d'avoir un caractère "X" qui vient juste après un caractère "C", autrement dit : la probabilité d'avoir X sachant C (*P(X|C)*).

Exemple 1

Dans premier temps, considérons le mot "banane". Il est possible de voir que:

- la lettre b est suivie de la lettre a une fois
- la lettre a est suivie de la lettre n 2 fois
- la lettre n est suivie de la lettre a une fois
- la lettre n est suivie de la lettre e une fois

A partir de ce constat, nous pouvons construire le tableau suivants:

		Caractère suivant				
		b	**a**	**n**	**e**	**Nombre d'apparitions**
Caractère actuel	**b**	0	1	0	0	1
	a	0	0	2	0	2
	n	0	1	0	1	2
	e	0	0	0	0	1

L'intersection entre une ligne *i* et une colonne *j* du tableau représente le nombre de fois où le caractère de la ligne *i* (Ci) est suivi par le caractère de la ligne *j* (Cj). En divisant ce nombre par le nombre d'apparition de Ci, il est possible d'obtenir la probabilité P(Cj/Ci).

Il est possible d'effectuer cette opération pour l'ensemble des cases du tableau, pour obtenir un tableau de probabilité :

	b	**a**	**n**	**e**
b	0	1	0	0
a	0	0	1	0
n	0	0,5	0	0,5
e	0	0	0	0

Grâce à ce tableau, il est possible de choisir le caractère qui suit un caractère connu, en prenant celui qui a la plus grande probabilité. Par exemple, si le caractère connu est "b", nous savons que seul le caractère "a" peut être son successeur.

Exemple 2

Dans l'exemple précédent, nous avons utilisé un seul mot pour construire notre modèle, nous allons voir que ça devient plus intéressant lorsque nous avons plus de mots.

- Considérons les mots suivants : "bonjour", "banane", "arbre", "nuit".
- A partir de cet ensemble de mots, il est possible de construire le tableau suivant :

	b	**o**	**n**	**j**	**u**	**r**	**a**	**e**	**i**	**t**	**Nombre d'apparitions**
b	0	1	0	0	0	1	1	0	0	0	3
o	0	0	1	0	1	0	0	0	0	0	2
n	0	0	0	1	1	0	1	1	0	0	4
j	0	1	0	0	0	0	0	0	0	0	1
u	0	0	0	0	0	1	0	0	1	0	2
r	1	0	0	0	0	0	0	1	0	0	3
a	0	0	2	0	0	1	0	0	0	0	3
e	0	0	0	0	0	0	0	0	0	0	2
i	0	0	0	0	0	0	0	0	0	1	1
t	0	0	0	0	0	0	0	0	0	0	1

En divisant le nombre de fois où un caractère Cj à suivit un caractère Ci par le nombre d'apparition de Ci dans les mots, nous obtenons le tableau de probabilité :

	b	**o**	**n**	**j**	**u**	**r**	**a**	**e**	**i**	**t**
b	0	0,33	0	0	0	0,33	0,33	0	0	0
o	0	0	0,50	0	0,50	0	0	0	0	0
n	0	0	0	0,25	0,25	0	0,25	0,25	0	0
j	0	1	0	0	0	0	0	0	0	0
u	0	0	0	0	0	0,50	0	0	0,50	0
r	0,33	0	0	0	0	0	0	0,33	0	0
a	0	0	0,67	0	0	0,33	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	1
t	0	0	0	0	0	0	0	0	0	0

Pour choisir le caractère suivant d'un caractère connu C, nous pouvons procéder ainsi :

- Etape 1 : Ordonner l'ensemble des successeur possible selon leur probabilité (la plus grande probabilité en premier)
- Etape 2 : Choisir le caractère X ayant la plus grande probabilité
- Etape 3 : Tester le caractère choisi
- Etape 4 : Si X est bien le successeur de C dans le mot mystère, repasser à l'étape 1 pour chercher le successeur de X. Sinon, sélectionner le caractère Y qui vient juste après X (selon les probabilités) et repasser à l'étape 3.

Exemple 3

Dans l'exemple précédent, nous avons vu qu'il était possible de générer le successeur d'un caractère connu en se basant sur les probabilités de transitions. Mais comment faire pour générer le premier caractère (nous n'avons aucun caractère connu) ? Il est possible de rajouter un "Faux caractère" au début des mots, par exemple "0" afin de nous permettre de générer le premier caractère.

Dans ce cas, la liste des mots de l'exemple 2 devient : "0bonjour", "0banane", "0arbre", "0nuit".

Et le tableau des probabilités de transition devient :

	0	**b**	**o**	**n**	**j**	**u**	**r**	**a**	**e**	**i**	**t**
0	0	0,50	0	0,25	0	0	0	0,25	0	0	0
b	0	0	0,33	0	0	0	0,33	0,33	0	0	0
o	0	0	0	0,50	0	0,50	0	0	0	0	0
n	0	0	0	0	0,25	0,25	0,00	0,25	0,25	0,00	0
j	0	0	1	0	0	0	0	0	0	0	0
u	0	0	0	0	0	0	0,50	0	0	0,50	0
r	0	0,33	0	0	0	0	0	0	0,33	0	0
a	0	0	0	0,67	0	0	0,33	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	1
t	0	0	0	0	0	0	0	0	0	0	0

Maintenant, il est possible de générer le premier caractère aussi, il suffit de chercher le successeur de “0”. Dans notre cas, nous avons 3 options :

- b avec une probabilité de 0.5 (b est le premier caractère des mots “banane” et “bonjour”).
- a avec une probabilité de 0.25 (a est le premier caractère du mot “arbre”)
- n avec une probabilité de 0.25 (n est le premier caractère du mot “nuit”).

Nous pouvons commencer par prendre le caractère ayant la plus grand probabilité (“b” dans notre cas) et tester s’il début bien le mot mystère, si ce n’est pas le cas, en passe au caractère suivant dans l’ordre des probabilités (nous avons une égalité dans notre cas, nous pouvons tester “a” puis “n”, ou “n” puis “a”), etc…

A faire

- **Essayez d’améliorer votre programme en utilisant un modèle de Markov.**