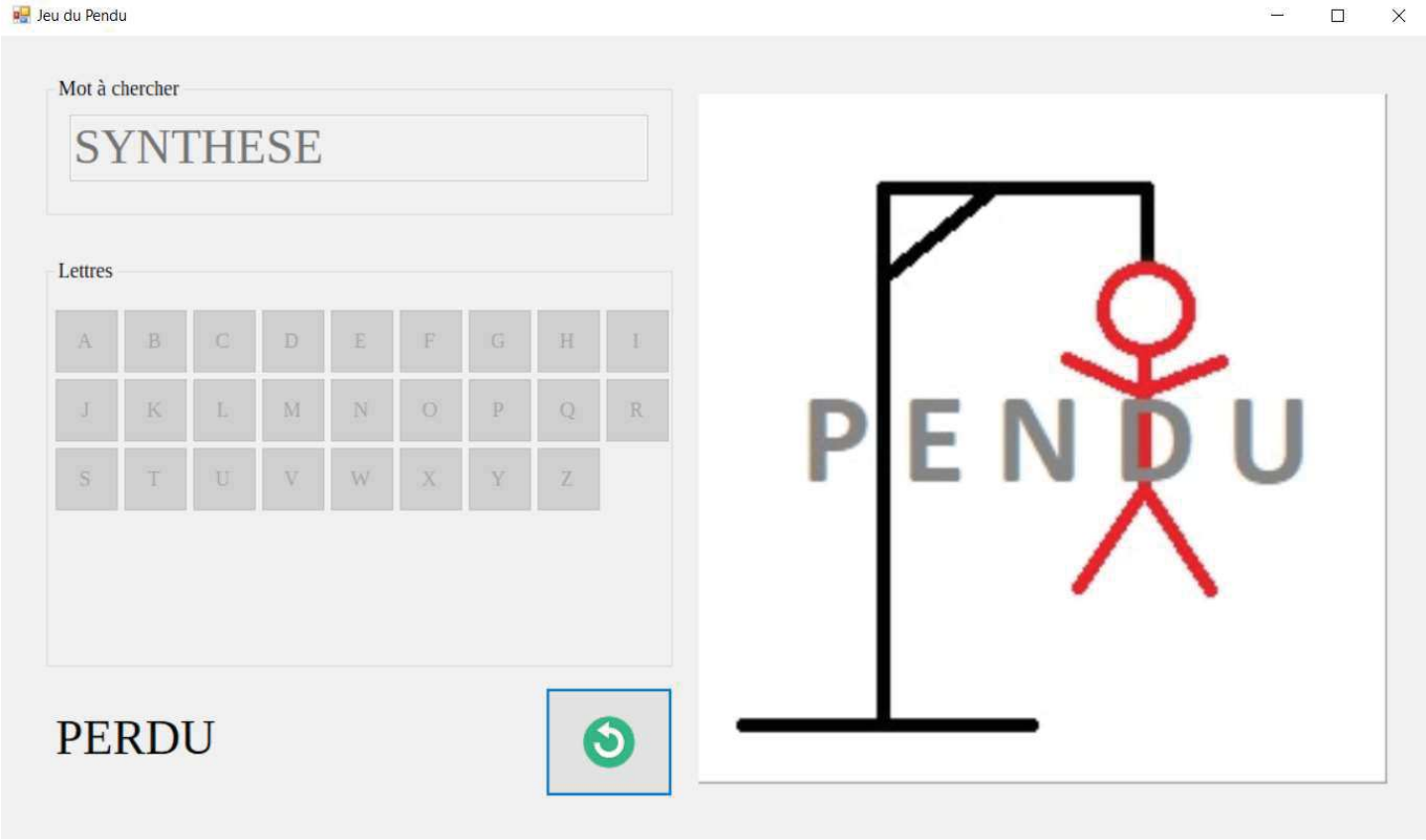
***Fiche de Synthèse : Pendu*** *Novembre 2019*

Développement :
Il s'agit d'une adaptation du celèbre jeu de Pendu, développé en mode graphique.
Déroulement du jeu : un 1er joueur saisi un mot, puis le valide. Le mot disparaît alors et le 2ème joueur doit le trouver avant d'être pendu.




Cadre :
Il s'agit d'un TP proposé dans le cadre du cours "Bases de la Programmation ", par Elisabeth Martins Da Silva.

Support :
Développé en C# avec Visual Studio Community 2019.

Contraintes :
C'est le 1er TP qui utilise les possibilités graphiques de Visual Studio, lors duquel j'ai pu re-découvrir la programmation évènementielle.

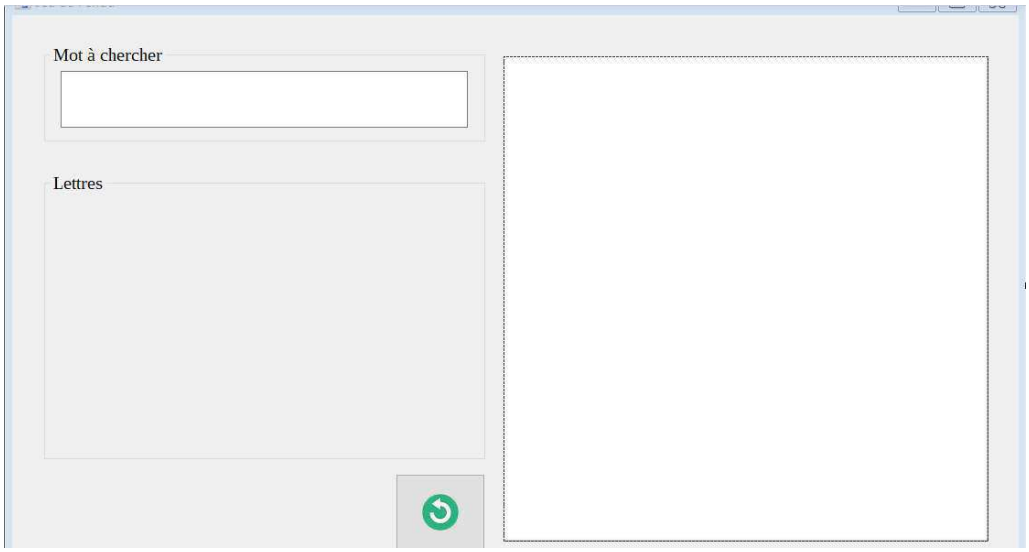
Difficultés rencontrées :
Pas de difficultés.

## Description détaillée du développement

J'ai suivi les étapes suivantes pour le développement:
- construction de l'interface
- saisie du mot à chercher
- gestion des boutons des lettres
- fin du jeu
- relance du jeu

1. La construction de l’interface

J’ai utilisé l’outil de Visual Studio, qui permet de “ dessiner ” tout ce dont on a besoin...sauf les boutons.




Pour les 26 boutons des lettres, il était plus simple de les coder manuellement, afin d’obtenir des boutons de mêmes dimensions, de pouvoir maîtriser l’espacement etc.…..ce n’était pas impossible de les dessiner directement avec la souris, comme le reste, mais ça aurait certainement bien plus long.

Au chargement du jeu donc, je créée 26 boutons de taille 50 x 50, j’y ajoute une écoute d’évènement, et enfin j’affiche l’image 1 du pendu (les dessins nous ont été fournis), c’est à dire un écran blanc.

```
1 référence
private void frmPendu_Load(object sender, EventArgs e)
{
    // création du tableau de lettres :
    Char[] tabAlpha = new char[26];
    tabAlpha = tAlpha();

    // déclarations variables : coordonnées boutons lettres :
    int x = 6, y = 38;

    // création des boutons + lettres :
    for (int i = 0; i < 26; i++)
    {
        Button btn = new Button();
        grpBoutons.Controls.Add(btn);
        btn.Location = new Point(x, y);
        btn.Size = new Size(50, 50);
        btn.Text = tabAlpha[i].ToString();
        btn.Enabled = false;

        x += 53;

        if (i == 8 || i == 17)
        {
            y += 53;
            x = 6;
        }

        // ajout d'une écoute d'un évènement clic sur le bouton :
        btn.Click += new System.EventHandler(btnAlpha_Click);

        // image de départ
        imgPendu.ImageLocation = Application.StartupPath + "/../../Resources/pendu" + "0.png";
    }
}
```

2. La saisie du mot à chercher

Il est donc temps que le 1er joueur saisisse le mot que l'autre joueur devra trouver.
Dans la zone de saisie, il lui suffit de taper un mot, en minuscule ou majuscules indifféremment, mais sans accent ni espaces ni caractères spéciaux. Ensuite, il tape entrée pour valider.

J'ai donc par exemple une fonction qui vérifie que le mot saisi est bien valide :

```
private bool estValide(string chaine)
{
    bool valide = true;
    chaine = chaine.ToUpper();

    if (chaine.Length < 1)
    {
        valide = false;
    }
    else
    {
        for (int k = 0; k < chaine.Length; k++)
        {
            if (chaine[k] < 65 || chaine[k] > 91)
            {
                valide = false;
            }
        }
    }

    return valide;
}
```

et, si oui, il est enregistré dans un tableau de " char " dans une variable motATrouver, et il est remplacé par des tirets (un tiret pour chaque lettre, rangé dans un tableau motRecherche).
S'il n'est pas valide, un message s'affiche et la zone de texte se vide.


Si le mot estValide, les boutons des lettres s'activent, la zone de texte elle se désactive (il n'est plus possible de saisir dedans).

3. La gestion des boutons des lettres

Lorsque le joueur qui recherche le mot clique sur une lettre : le bouton de la lettre est désactivé sur le 'clavier', la lettre est recherchée dans le motATrouver ; dans ce cas, soit elle y est et dans ce cas elle apparaît à la place du/des tiret(s) correspondants, soit elle n'y est pas et le dessin du pendu avance d'un cran.

Pour illustrer, les 2 lignes de codes qui récupèrent la lettre cliquée et qui la désactive :

```
// clic sur une lettre : on récupère le texte dans une variable :
Char lettre = Convert.ToChar(((Button)sender).Text);
// désactive le bouton cliqué
((Button)sender).Enabled = false;
```

et la fonction booléenne qui contrôle si la lettre figure dans le mot à rechercher, et qui remplace le tiret le cas échéant :

```
1 référence
private bool inMot(Char lettre, Char[] tableau1, Char[] tableau2)
{
    bool presente = false;

    for (int i = 0; i < tableau1.Length; i++)
    {
        if (tableau1[i] == lettre)
        {
            presente = true;
            tableau2[i] = lettre;
        }
    }
    return presente;
}
```

4. La fin du jeu

Le jeu est terminé :
- quand le joueur a trouvé le mot : dans ce cas le mot “ gagné ” s’affiche
- quand le joueur est pendu avant d’avoir trouvé : dans ce cas le mot “ perdu ” s’affiche et le mot qui était à trouver s’affiche lui aussi

Ensuite dans les 2 cas les boutons des lettres sont désactivés, et le seul bouton sur lequel on peut cliquer est le bouton “ relance ”.

```
/* Si c'est la fin du jeu, c'est à dire :
 * - soit le mot est trouvé
 * - soit on est "pendu"
 * alors, procédure de fin de jeu pour désactiver les lettres
 * et mettre le focus sur le bouton relance
 */
if (motEstTrouve(motATrouver, motRecherche))
{
    lblGagnant.Text = "GAGNE";
    finJeu();
}
else if (numeroImage == 11)
{
    lblGagnant.Text = "PERDU";
    afficheLeMot(motATrouver);
    finJeu();
}
```

5. La relance du jeu

Enfin, lors du clic sur le bouton pour relancer le jeu :

```
/* Relance du jeu :
 * - le dessin s'efface
 * - la zone de texte se vide
 * - et elle redevient active pour la saisie d'un nouveau mot
 */
1 référence
private void btnRelance_Click(object sender, EventArgs e)
{
    imgPendu.ImageLocation = Application.StartupPath + "/../../Resources/pendu" + "0.png";
    txtMot.Text = "";
    txtMot.Enabled = true;
    txtMot.Focus();
    lblGagnant.Text = "";
    numeroImage = 1;
    resetMots(motATrouver, motRecherche);
}
```