TPC#13 : PixelPower

1 Structure du rendu

Voici la structure du rendu pour ce TP :

```
rendu-tpX-login_x.zip/
    login_x/
        AUTHORS           // '* login_x'
        README            // ne mettez un README que si vous avez quelque chose à préciser
        PixelPower/       // contient tout ce qu'il faut pour compiler
```

Vous ne devez pas rendre de code qui **ne compile pas** !
Vérifiez bien avant d'envoyer votre fichier .zip que vous pouvez **dézipper, ouvrir votre solution .sln sans erreurs et compiler.**

2 Introduction

Ce tp a pour but de vous faire faire un mini paint avec la possibilité de dessiner, effacer, changer la couleur du pinceau etc...

Mais ce n'est pas tout, vous allez aussi implémenter un mode pixelpower qui va animer votre image avec une pseudo gravité.

Vous serez libres d'implémenter votre code comme vous le voulez, les prototypes de fonctions ne sont là qu'à titre indicatif. Vous serez noté par étapes donc ne faites pas l'étape N+1 si vous n'avez pas fini l'étape N.

3 Le projet

Avant de commencer récupérez le dossier PixelPower qui contient la solution ainsi que tous les fichiers de base qui vous seront utiles.

3.1 Etape 1 : Piece of cake

Ouvrez le fichier main.cs, dans le constructeur chargez une nouvelle texture2D dans une variable qu'on appelera *baseimage*. Dans la fonction draw dessinez *baseimage*. Si tout c'est bien passé vous devriez voir une image apparaître dans votre fenêtre lorsque vous compilez. Pensez à écrire 'sb.Draw(' et regardez ensuite les surcharges qui vous sont proposés par notre cher Visual studio.

FIGURE 1 – a piece of cake

3.2 Etape 2 : Two pieces of cake

Créez une classe Map. A l'intérieur de la classe ajoutez deux variables statiques height et width. Grâce à cela vous allez pouvoir accéder à partir de n'importe quel classe de votre programme à ces deux variables. Essayez par vous même en écrivant dans le main "Map." (vous verrez les deux propriétés être proposées par visual studio).

Initialisez ces deux variables avec la taille de *baseimage* juste après le chargement de ce dernier.

Pour finir ouvrez le fichier *game1.cs* et écrivez les lignes suivantes après la création de Main.

```
graphics.PreferredBackBufferHeight = Map.height;
graphics.PreferredBackBufferWidth = Map.width;
graphics.ApplyChanges();
```

Ce bout de code va permettre de mettre la dimension de l'écran de rendu aux dimensions de baseimage. Essayez de compiler vous verrez normalement une image avec un petit camion, des traits noirs et un hibou.

FIGURE 2 – Two pieces of cake

3.3 Etape 3 : Help me !

Créez une classe *Help* qui sera instanciée dans le main. Cette classe devra avoir au moins une méthode *Draw* pour l'affichage. Lors du lancement de votre programme il faut que un petit texte 'help : H' s'affiche en haut à gauche de votre écran. Si l'utilisateur appuye avec grâce et volupté sur la touche 'H' une aide doit s'afficher. S'il réappuye (toujours avec grâce et volupté) sur H l'aide disparaît et le texte 'help : H' s'affiche de nouveau en haut à gauche de l'écran.

Vous êtes libre de l'implémentation mais nous vous conseillons de mettre un booléen 'activated' (avec le getter qui va avec) dans la classe Help qui permettra de savoir s'il faut afficher de l'aide ou non.

Pour vous aider sur les entrées clavier/souris vous pouvez utiliser la classe *Inputs.cs* (codé par nos soins), pour l'utiliser tapez simplement '*Inputs.*' et vous verrez plusieurs méthodes vous être proposés.

Exemple pour savoir si un utilisateur a tapé sur la touche 'X'.

```
if (Inputs.isKeyRelease(Keys.X))
```

Vous devez rajouter au fur et à mesure du tp du texte d'aide dans cette classe.

3.4 Etape 4 : Brush display

Ajoutez l'affichage de la brush.
Vous avez l'image dans votre conteneur qui s'appelle brush.png. Pour le positionnement de la souris n'oubliez pas la classe Inputs qui pourra vous aider grandement.

3.5 Etape 5 : Brush2 bigger and smaller

Changez la taille de la brosse avec les touches + et - de votre clavier. N'oubliez pas de mettre à jour la classe *Help*.

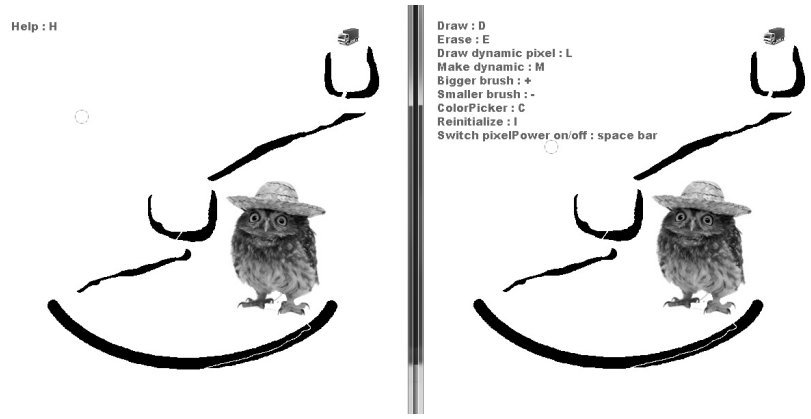

FIGURE 3 – Un exemple à quoi pourrait ressembler votre aide à la fin du tp

3.6 Etape 6 : Let's draw something

Pour pouvoir modifier notre image en temps réel on va convertir la texture en un tableau de *Color*, ce tableau sera utilisé pour la modification pixel par pixel de notre image.

Commencez par créer une propriété de la classe *Main* qui sera un tableau de *Color* de dimension height*width de votre image que vous appelerez *pixels*.

Ensuite, ajoutez cette fonction dans votre fichier *Main* qui vous permettra de récupérer l'image *Texture2D* sous la forme d'un tableau de *Color*.

```
private Color[] get_color_data(Texture2D texture)
{
      Color[] data = new Color[texture.Height * texture.Width];
      texture.GetData(data);

      return data;
}
```

Puis appelez le avec ce bout de code

```
pixels = get_color_data(baseimage);
```

pixels contient maintenant tous les pixels sous la forme d'un tableau à une dimension.

Pour finir mettez ce bout de code au début de votre fonction *Draw* du *Main*.

```
/* unset curr texture */
baseimage.GraphicsDevice.Textures[0] = null;
/* set new texture that has been computed */
baseimage.SetData(pixels);
```

Grâce à ces deux lignes votre tableau *pixels* est converti en *Texture2D* et stocké dans *baseimage* avant d'être dessiné.

Maintenant implémentez la fonction SetPixel dont le prototype pourrait être :

```
public void SetPixel(int X, int Y, Color c, Color[] pixels)
```

N'oubliez pas d'utiliser ce que vous avez appris lors de votre tp sur les boucles pour convertir les positions X Y d'un tableau en deux dimensions à un index dans un tabeau à une dimension et vice versa.

Lorsque l'utilisateur enfonce la touche 'D' vous devez utiliser la fonction *SetPixel* pour transformer les pixels à l'index X,Y de votre tableau *pixels*. Vous pouvez mettre la couleur que vous voulez pour l'instant.

Normalement vous devez être capable de dessiner un trait fin avec la touche 'D' lorsque vous lancez votre programme.

3.7 Etape 7 : Variable size drawing

Implémentez la fonction *SetPixels* qui va dessiner des pixels par rapport au rayon de notre brush. Un exemple de prototype pourrait être :

```
SetPixels(int X, int Y, Color c,  Color[] pixels, int brush_radius)
```

Si vous avez bien fait votre boulot vous êtes maintenant capable de dessiner de jolis traits de couleur de la même taille que votre brush.

3.8 Etape 8 : Erase

Implémentez l'effaçage de pixels lorsque l'utilisateur tape sur la touche 'E'. Pensez à utiliser ce que vous avez fait juste avant !!! Normalement cette étape est très rapide (<2min) si tout a été bien fait auparavant.

Un pixel vide est le pixel de RGBA = 0,0,0,0.

3.9 Etape 9 : Reinit

Implémentez la restauration de votre image de départ lorsque l'utilisateur tape sur la touche 'I'.

3.10 Etape 10 : ColorPicker display

Impléméntez l'affichage ou pas du *colorpicker* lorsque l'utilisateur tape sur la touche 'C'. Utilisez la classe *ColorPicker* déjà disponible dans votre projet pour gagner du temps. Pensez à décommenter le code ligne 21 dans le contructeur pour qu'il fonctionne.

3.11 Etape 11 : Pick a color

Ecrivez le code qui permet de récupérer la couleur lorsque l'utilisateur clique sur le bouton gauche de sa souris sur le *colorpicker* (stockez la par exemple dans une propriété de la classe *Main 'curr_ color')*. Essayez de dessiner avec différentes couleurs, si cela marche c'est que cette étape est validée.

N'oubliez pas d'utiliser la méthode *getColor()* de la classe *ColorPicker* !

3.12 Etape 12 : PixelPower

Créer une classe *PixelPower* avec une méthode *Update()* qui prend en paramètre votre tableau de *Color 'pixels'*. Nous allons considérer que tous les pixels dont la propriété alpha n'est pas maximale doivent être soumis à la gravité, nous les appelerons désormais les pixels dynamiques.

Parcourez le tableau de pixels en partant du bas du tableau pour mettre à jour les pixels de la façon suivante (le pixel sombre est la case du tableau pixels que vous êtes en train de traîter, les cases gris clair sont celles d'un pixel non vide et les case blanches représentent les pixels vides RGBA=0) :

FIGURE 4 – Si le pixel en dessous est vide on déplace le pixel d'une case vers le bas

FIGURE 5 – Si le pixel en dessous n'est pas vide et que les deux pixels adjacents sont vides alors on déplace aléatoirement le pixel sur l'une des deux cases

FIGURE 6 – Si le pixel en dessous et le pixel en bas à gauche ne sont pas vides mais le pixel en bas à droite est vide, alors on place le pixel dessus

FIGURE 7 – Si le pixel en dessous et le pixel en bas à droite ne sont pas vides mais le pixel en bas à gauche est vide, alors on place le pixel dessus

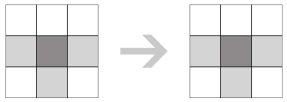

FIGURE 8 – Si le pixel en dessous et les pixels adjacents horizontalement ne sont pas vides, alors on ne fait rien

FIGURE 9 – Si le pixel en dessous et les pixels adjacents ne sont pas vides, alors on ne fait rien

FIGURE 10 – Le pixel ne peut pas se mettre en bas à gauche car il est bloqué par une case horizontale

FIGURE 11 – Idem mais dans l'autre sens

Appelez votre fonction *update()* dans la fonction *update* du fichier *Main.cs*. Normalement vous verrez le camion tomber vers le bas et s'écouler le long des traits noirs.

3.13 Etape 13 : PixelPower Activate !

Ajoutez la touche 'espace' pour activer désactiver le mode *pixelpower*.

3.14 Etape 14 : Make dynamic

Ajoutez la touche 'M' pour rendre les pixels sous la brush dynamiques.

3.15 Etape 15 : Draw dynamic pixels

Ajoutez la touche 'L' pour dessiner des pixels dynamiques.

4 Conclusion

N'oubliez pas de mettre à jour votre classe *Help*. Le correcteur se basera sur ce que vous avez marqué là dedans pour tester votre programme.