TPC#13 : PixelPower

1 Handout structure

This is the structure you have to follow for your handout :

```
rendu-tpX-login_x.zip/
    login_x/
        AUTHORS          // '* login_x'
        README           // only put a README if you have some precisions to give
        PixelPower/      // contains your project
```

Do NOT send us code that **does not compile**!
Double-check that you can : **unzip your archive, open your solution and compile it without errors**.

2 Introduction

In this session, your goal is to implement a paint-like program that lets you draw, erase and change the colors of your pixels.
But that's not all. You will also create the PixelPower mode, which will add pseudo-gravity to your project. This will animate the pixels in your image.
The function prototypes in this project are only there to give you the general idea, you are free to implement everything the way you like. You will be graded stepwise. Do not start step N+1 if you haven't finished step N.

3 The project

Before you can start the project, you have to retrieve the PixelPower given sources. They will be useful for this project.

3.1 Step 1 : A piece of cake

Open *main.cs* and load a new Texture2D in a variable we will call *baseimage*. In the draw function, draw *baseimage*. If everything is done correctly, you will see a picture appear in the window when you launch your program.
You will have to use the *Draw()* method of the *sb* SpriteBatch object.

FIGURE 1 – A piece of cake

3.2 Step 2 : Two pieces of cake

Create a *Map* class. It has to contain two static variables : *height* and *width*. This will allow you to access these two parameters in the whole project. Test it yourself by writing *Map.* in the main class. You should see the two variables appear.
Initialize the two variables with the size of *baseimage* after you loaded it.

Finally, open the *game1.cs* file and write the following lines after the initialization of the *Main* class :

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

This code snippet will adapt the size of your window to the dimensions of *baseimage*. You should see an owl, a truck and some black lines.

FIGURE 2 – Two pieces of cake

3.3 Step 3 : Help me !

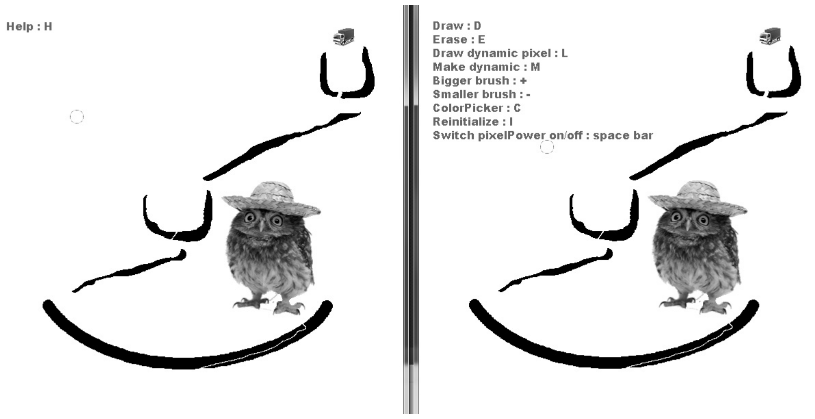
Create a *Help* class that will be instantiated in your *Main* class. It will have a *Draw()* method that will allow it to be displayed. When your program is launched, there must be a small note with 'help : H' displayed in the top left corner. When the user hits the 'H' key, a help message must be displayed. Hitting the 'H' key again returns to the default state.

You can implement this as you like, but we advise you to use a boolean 'activated' (with appropriate accessors) in the Help class which will inform you if you need to display the message or not.

To help you with mouse and keyboard input, we provide the *Input.cs* file. Simply type *Inputs.* and you will see several methods proposed to you.
For example, to know if a user has hit the 'X' key.

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

You need to add text to the *Help* class throughout the whole project, to document expected user input.



```
Help : H
```

```
Draw : D
Erase : E
Draw dynamic pixel : L
Make dynamic : M
Bigger brush : +
Smaller brush : -
ColorPicker : C
Reinitialize : I
Switch pixelPower on/off : space bar
```

FIGURE 3 – Your help message may look like this at the end of this project

3.4 Step 4 : Brush display

Add the brush display.
The brush.png image can be used to display the brush. You can use the *Inputs* class for mouse positioning.

3.5 Step 5 : Brush2 bigger and smaller

Change the size of the brush with the + and – keys of your keyboard. Do not forget to update the *Help* class.

3.6 Step 6 : Let's draw something

To update our image in real-time we will convert the texture in a Color array, which we will use to modify the image pixel by pixel.

Start by creating a field in the *Main* class which will be a Color array of size *height*width* of your image that we will call *pixels*
After that, you can add the following function to your main file, which will allow you to translate a 2D texture in a Color array.

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

      return data;
}
```

Then call it with this code :

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

pixels now contais all the pixels in a Color array.

Finally, add this code block at the start of the *Draw* function of your main file.

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

These two lines will convert the Color array to the Texture2D and store it in *baseimage*.

You can now implement the *SetPixel* function, that can have the following prototype :

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

Do not forget to use what you learned on the '%' operator to map 2D coordinates to a single dimension array, and other methods for the other way around.
When the user hits the 'D' key, you have to use the *SetPixel* function to change the pixels at the X,Y index of your array. For now, you can use the color you want.

Normally, you must be able to draw a thin line in your program by pressing the 'D' key.

3.7 Step 7 : Variable size drawing

Implement the *SetPixels* function which will draw pixels according to the radius of our brush. The prototype of this function could be :

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

If everything has gone well, you can draw lines and dots that have the same size as your brush.

3.8 Step 8 : Erase

Implement the 'erase' function of your program, bound to the 'E' key. Do not forget to use what you have implemented already ! ! ! This should only take a minute or so.

An empty pixel has the following color code : RGBA = 0,0,0,0

3.9 Step 9 : Reinit

Implement an image reset feature. When the user hits the 'I' key, the image should be set back to what it was before any modifications.

3.10 Step 10 : ColorPicker display

Display the colorpicker when the user presses the 'C' key. You can use the *ColorPicker* class already available in the project to gain time.

Uncomment the code at line 21 in the constructor to make it work.

3.11 Step 11 : Pick a color

Write the code that allows the user to change the color he is drawing with. You can detect this when the user left-clicks on the colorpicker. You could store this in a variable of the main class. Try to draw with several colors to test this step.
getColor() in the ColorPicker class will be useful !

3.12 Step 12 : PixelPower

Create the PixelPower class with an *Update()* method that takes the Color array *pixels* as argument. We define that pixels with an alpha value lower than 255 are affected by gravity. We will call these pixels "dynamic pixels".

Browse the pixel array, starting from the bottom of the image, and update the pixels according to the rules below. Note : the dark pixel is the pixel you are on currently, the grey pixels are non-empty pixels and the white pixels are empty.

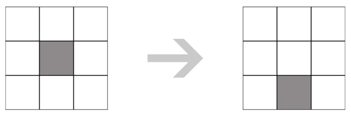


FIGURE 4 – If the bottom pixel is empty we move the pixel down one block

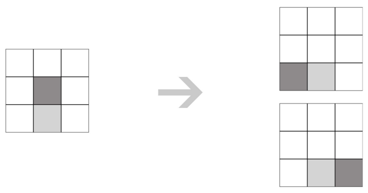


FIGURE 5 – If the bottom pixel is not empty and the left and right pixels are empty, we move the pixel randomly to the left or to the right

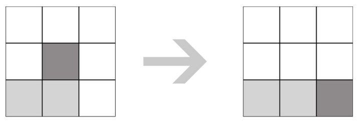


FIGURE 6 – If the bottom and bottom-left pixels are not empty but the bottom-right one is, then move the pixel there

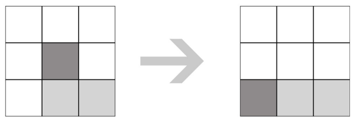


FIGURE 7 – If the bottom and bottom-right pixels are not empty but the bottom-left one is, then move the pixel there

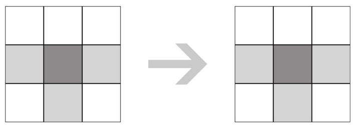


FIGURE 8 – If the bottom pixel and the adjacent pixels are not empty, then nothing happens

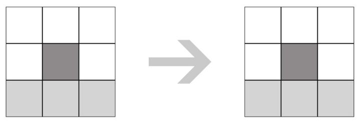


FIGURE 9 – If the bottom pixels are not empty, then nothing happens

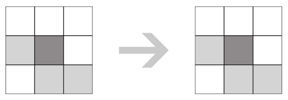


FIGURE 10 – The pixel cannot go to the bottom-left because it is blocked by the left pixel

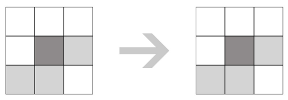


FIGURE 11 – Same as above but the other way around

Call the *Update()* function in the *main.cs* file. You should see the truck fall and melt on the black line.

3.13 Step 13 : PixelPower Activate !

Add the spacebar to activate the pixelpower mode.

3.14 Step 14 : Make dynamic

Add the 'M' key to make the pixels under the brush dynamic.

3.15 Step 15 : Draw dynamic pixels

Add the 'L' key to draw dynamic pixels.

4 Conclusion

Do not forget to update your *Help* class. When we review your work, we will use that information to test your program.