FULL SAIL UNIVERSITY

Tile Editor Mechanics

The Tile Set

This tool is a level editor for a 2-D scrolling game. It must load in a tile set (a bitmap divided into tiles) and display it in a window. In another window it will display the map (the individual tiles rearranged into a larger grid). The user must be able to select individual tiles in the tile set and then place them at the desired location in the map. To do this in code will require a system for referencing individual tiles. The following is just a suggestion. If you have other ideas on how to handle the mathematics and the data structures your welcome to follow them.

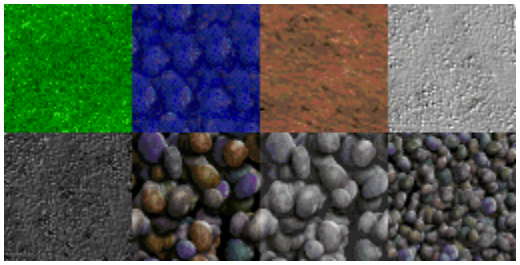


Figure 1: The Default Bitmap

When I refer to a tile set I will use the following terminology:

- *TileSetSize* – This refers to the size of a tile set in tiles (not pixels). Since a tile set is a 2-D array of tiles this value will have two components: width and height. The default tile set has a *TileSetSize* of (4, 2).
- *TileSize* – When we are ready to render tiles into a window we will also need to know the size of an individual tile in pixels. This I will refer to as *TileSize* and it too has a width and height component only this time measured in pixels, not tiles. The *TileSize* for the default tile set is (64, 64).

Consider the following grid. This will represent the default tile set shown above. I will use this tile set for most of the class demonstrations. Pay close attention to the two numbering systems that are used to identify an individual tile. To manipulate tiles in code you will need to reduce them to numbers.

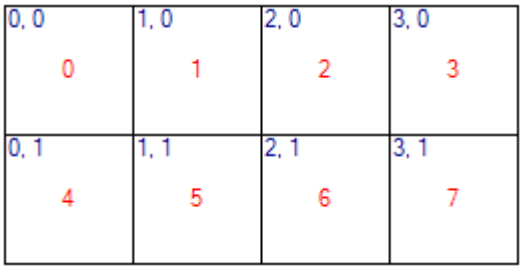


Figure 2: The Default Tile Set Grid

An individual tile can be identified by either a coordinate pair: (2, 1) or a single index value: (6). Please make note of the fact that tile (2, 1) and tile (6) are the same tile. In fact there is a solid mathematical relationship between the two number and it can be expressed with 3 formulas that you probably have seen before in another form:

$$index = (y * tileSetSize.Width) + x$$

$$y = index / tileSetSize.Width$$

$$x = index \% tileSetSize.Width$$

These formulas can be used to translate between a coordinate and an index identifier:

$$6 = (1 * 4) + 2$$

$$1 = 6 / 4$$

$$2 = 6 \% 4$$

Another bit of the old mathematics that might come in handy is being able to calculate the size and position of an individual tile in pixels. It will be easier for us if we can refer to tile (3, 1) or tile (7) but when you get ready to render the tile into a window using the method *System.Drawing.Graphics.DrawImage* it's not going to understand (3, 1) or (7). The graphics API needs pixels! Converting between pixels and grid units is really just a matter of scaling by the size of an individual tile. To scale from coordinate pairs to pixels is two formulas:

$$pixelX = x * tileSize.Width$$

$$pixelY = y * tileSize.Height$$

With these formulas we can calculate the pixel location of tile (3, 1):

$$192 = 3 * 64$$

$$64 = 1 * 64$$

Sometimes you will need to scale back in the other direction. If the user clicks the mouse in your tile set you want to know which tile they intended to select. The mouse click just gives you pixels so you will need to convert back to a coordinate pair:

$$x = mouseClick.X \ / \ tileSize.Width$$

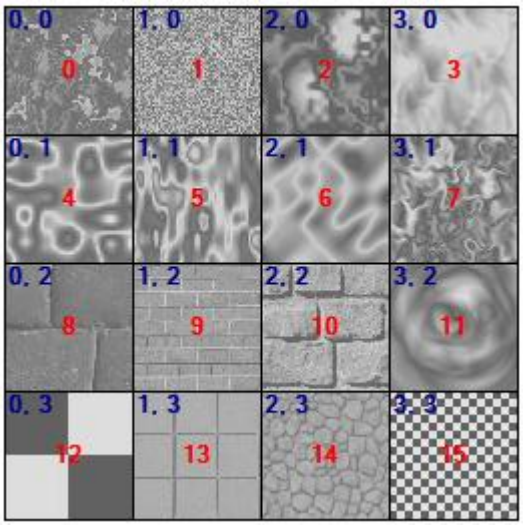
$$y = mouseClick.Y \ / \ tileSize.Height$$

$$3 = 200 \ / \ 64$$

$$1 = 74 \ / \ 64$$

Keep in mind that our tile editor must be able to work with different sizes of tiles and tile sets. As you build the program consider how you are going to deal with variants that might be similar or different to the ones below:

Figure 3: TileSetSize = (4, 4), TileSize = (64, 64)



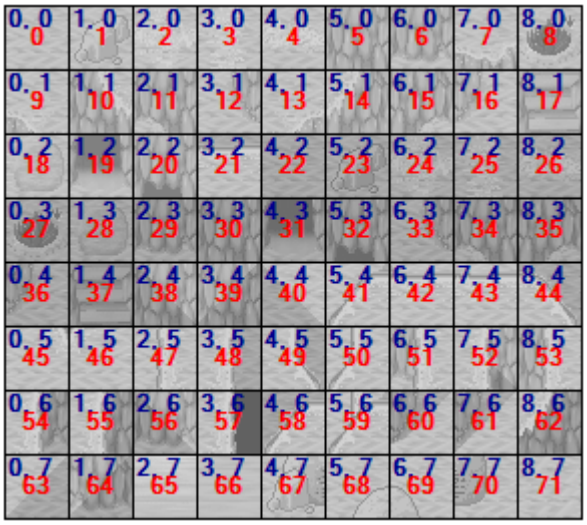


Figure 4: TileSetSize = (9, 8), TileSize = (32, 32)

The Map

Our application contains two distinct window components: the tile set and the map. There are many similarities between the two windows:

How the Map and Tile Set are Similar

- They both have a width and height value in tiles. To keep things straight I will use *TileSetSize* to refer to the width and height in tiles of the tile set. I will use *MapSize* to refer to the width and height in tiles of the map. The following figure is a map whose *MapSize* = (15, 10) and whose *TileSize* = (32, 32).

0, 0	1, 0	2, 0	3, 0	4, 0	5, 0	6, 0	7, 0	8, 0	9, 0	10, 0	11, 0	12, 0	13, 0	14, 0
0, 1	1, 1	2, 1	3, 1	4, 1	5, 1	6, 1	7, 1	8, 1	9, 1	10, 1	11, 1	12, 1	13, 1	14, 1
0, 2	1, 2	2, 2	3, 2	4, 2	5, 2	6, 2	7, 2	8, 2	9, 2	10, 2	11, 2	12, 2	13, 2	14, 2
0, 3	1, 3	2, 3	3, 3	4, 3	5, 3	6, 3	7, 3	8, 3	9, 3	10, 3	11, 3	12, 3	13, 3	14, 3
0, 4	1, 4	2, 4	3, 4	4, 4	5, 4	6, 4	7, 4	8, 4	9, 4	10, 4	11, 4	12, 4	13, 4	14, 4
0, 5	1, 5	2, 5	3, 5	4, 5	5, 5	6, 5	7, 5	8, 5	9, 5	10, 5	11, 5	12, 5	13, 5	14, 5
0, 6	1, 6	2, 6	3, 6	4, 6	5, 6	6, 6	7, 6	8, 6	9, 6	10, 6	11, 6	12, 6	13, 6	14, 6
0, 7	1, 7	2, 7	3, 7	4, 7	5, 7	6, 7	7, 7	8, 7	9, 7	10, 7	11, 7	12, 7	13, 7	14, 7
0, 8	1, 8	2, 8	3, 8	4, 8	5, 8	6, 8	7, 8	8, 8	9, 8	10, 8	11, 8	12, 8	13, 8	14, 8
0, 9	1, 9	2, 9	3, 9	4, 9	5, 9	6, 9	7, 9	8, 9	9, 9	10, 9	11, 9	12, 9	13, 9	14, 9

Figure 5: MapSize = (15, 10), TileSize = (32, 32)

- They both need to know the size of a tile in pixels. In my program they both share the same variable. I have already introduced *TileSize* as referring to the width and height of an individual tile in pixels. As far as I am concerned this value applies to both the map and the tile set.
- For the purposes of scrolling each will need to keep track of its document size. Document size is the absolute width and height in pixels of both the tile set and the map individually. To keep them straight I will use *TileSetDocSize* and *MapDocSize* for each individual window. These values are easy to calculate from the previously defined terms:

```
Size tileSetDocSize = Size.Empty;
tileSetDocSize.Width = tileSetSize.Width * tileSize.Width;
tileSetDocSize.Height = tileSetSize.Height * tileSize.Height;
```

or

```
Size mapDocSize = Size.Empty;
mapDocSize.Width = mapSize.Width * tileSize.Width;
mapDocSize.Height = mapSize.Height * tileSize.Height;

The document size for some of the previous figures would be as follows:
```

- *Figure 3: (256, 256) = TileSetSize (4, 4) * TileSize (64, 64)*
- *Figure 4: (288, 256) = TileSetSize (9, 8) * TileSize (32, 32)*
- *Figure 5: (480, 320) = MapSize (15, 10) * TileSize (32, 32)*

How the Map and Tile Set are Different

The biggest difference between the map and the tile set is that the map is a virtual construction. The tile set window just needs to draw the selected bitmap and render a grid on top of it that matches the *TileSetSize* and *TileSize* values.

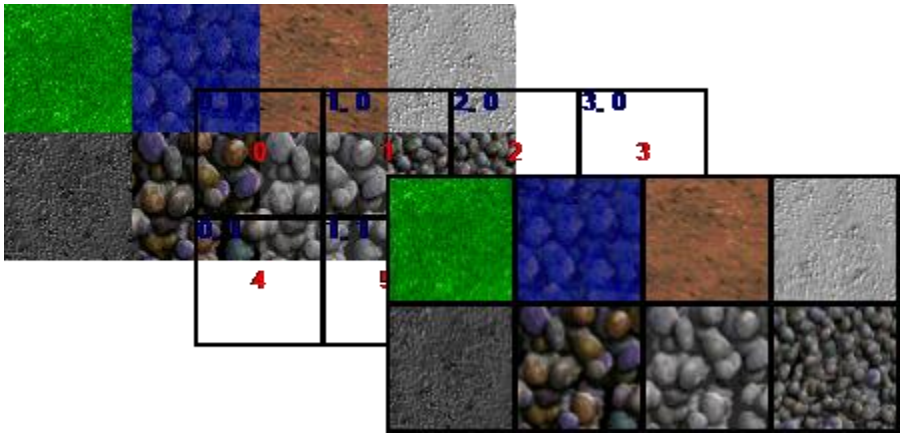


Figure 6: The tile set is just a bitmap and a grid.

The map however is not a single bitmap but instead a series of operations your code performs in the window's *Paint* event. You need to specify which tile from the tile set is drawn in which grid of the map. To do this you need some kind of a data structure to hold this information. Since the grid is a 2-D array that is the most obvious approach. If you intend to store each tile's location as a coordinate pair (x, y) then the array should be composed of coordinates. You could create a special class for this purpose or use the *System.Drawing.Point* structure already in the NET Framework.

In order to render a tile in the map you will need two pieces of information: the source rectangle (where the tile is coming from) in the tile set, and the destination rectangle (where the tile will be displayed in the map window), both of these rectangles need to be in pixels.

The source rectangle can be calculated from the coordinate pairs stored in the 2-S array:

```
Rectangle srcRect = Rectangle.Empty;
srcRect.X = map[x, y].X * tileSize.Width;
srcRect.Y = map[x, y].Y * tileSize.Height;
srcRect.Size = tileSize;
```

The destination rectangle can be calculated from the tile's current position in the array, here represented by *x* and *y*:

```
Rectangle destRect = Rectangle.Empty;
destRect.X = x * tileSize.Width;
destRect.Y = y * tileSize.Height;
destRect.Size = tileSize;
```

The following figure should help illustrate this relationship:

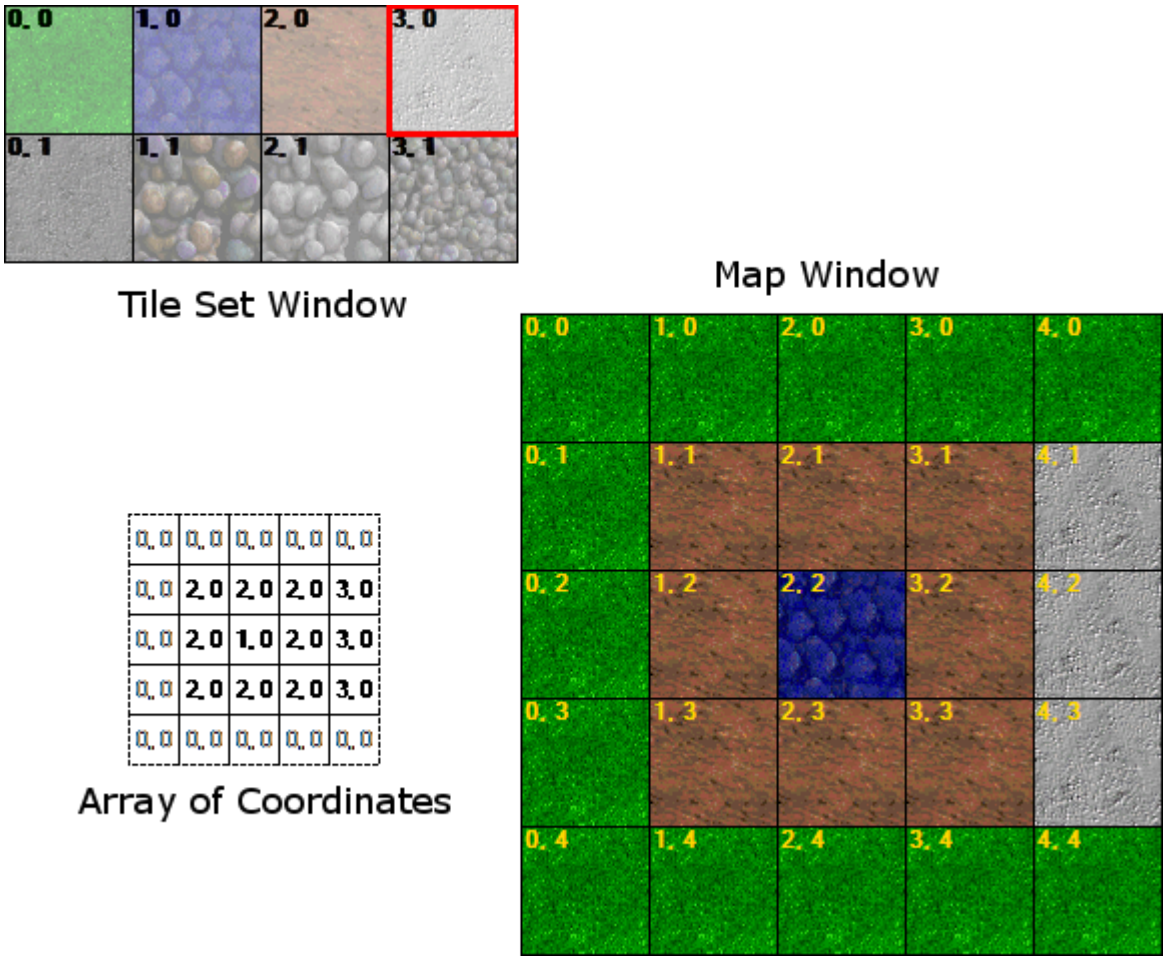


Figure 7: Map, Tile Set, and Internal Array

The *x* and *y* values can easily be calculated from a couple of nested for loops in the Paint event of the map window:

```
for (int x = 0; x < MapSize.Width; x++)
{
    for (int y = 0; y < MapSize.Height; y++)
    {
        Rectangle destRect = Rectangle.Empty;
        destRect.X = x * tileSize.Width;
        destRect.Y = y * tileSize.Height;
        destRect.Size = tileSize;


        Rectangle srcRect = Rectangle.Empty;
        srcRect.X = map[x, y].X * tileSize.Width;
        srcRect.Y = map[x, y].Y * tileSize.Height;
        srcRect.Size = tileSize;

        // Render the tile in the map using an overload of the Graphics.DrawImage method.
        g.DrawImage(tileSet, destRect, srcRect, GraphicsUnit.Pixel);
    }
}
```