PART 3: “Variables”

Introduction

The previous tutorial introduced you to the template. You can now draw some lines and print some text in all kinds of colors, but that’s clearly far from the goal of making actual games, where bullets wizz past in glorious 3D and enemies flank you in the smartest ways. To get a bit closer to that, we’ll take the static coordinates from last time and make them a bit more *flexible.* We’ll need that for the next part, which is all about loops, sprites and other bouncy things.

Getting the stuff you need

As promised, we’ll use the template again. Extract the package to a fresh directory (say, `c:\my_projects\variables`) and load up the .sln file. Remove the ‘hello world’ stuff in the `Tick` function.

Now make your Tick function like this:

```
void Game::Tick( float deltaTime )
{
    screen->Clear( 80 );
    screen->Line( 100, 50, 200, 50, 0xffffff );
    screen->Line( 150, 50, 150, 300, 0xffffff );
    screen->Line( 100, 300, 200, 300, 0xffffff );
}
```

This paints the ‘I’ of ‘IMPRESSIVE’ in glorious white on a vibrant blue backdrop. Now imagine we would like thicker lines. We take the three line commands, copy them, and draw them one pixel to the right:

```
void Game::Tick( float deltaTime )
{
    screen->Clear( 80 );
    screen->Line( 100, 50, 200, 50, 0xffffff );
    screen->Line( 150, 50, 150, 300, 0xffffff );
    screen->Line( 100, 300, 200, 300, 0xffffff );
    screen->Line( 101, 50, 201, 50, 0xffffff );
    screen->Line( 151, 50, 151, 300, 0xffffff );
    screen->Line( 101, 300, 201, 300, 0xffffff );
}
```

Now, that’s a bit disappointing: obviously, only the vertical line got thick now. Let’s draw all six lines one more time, this time one pixel lower:

```
void Game::Tick( float deltaTime )
{
    screen->Clear( 80 );
    screen->Line( 100, 50, 200, 50, 0xffffff );
    screen->Line( 150, 50, 150, 300, 0xffffff );
    screen->Line( 100, 300, 200, 300, 0xffffff );
    screen->Line( 101, 50, 201, 50, 0xffffff );
    screen->Line( 151, 50, 151, 300, 0xffffff );
```

```
    screen->Line( 101, 300, 201, 300, 0xffffff );
    screen->Line( 100, 51, 200, 51, 0xffffff );
    screen->Line( 150, 51, 150, 301, 0xffffff );
    screen->Line( 100, 301, 200, 301, 0xffffff );
    screen->Line( 101, 51, 201, 51, 0xffffff );
    screen->Line( 151, 51, 151, 301, 0xffffff );
    screen->Line( 101, 301, 201, 301, 0xffffff );
}
```

This time the result is satisfying. However, this is not something you want to spend a career on: we need to find a smarter way to do this. In C/C++ you would use a function for this. But before we get to that, let's try something else.

```
void Game::Tick( float deltaTime )
{
    screen->Clear( 80 );
    int x = 100;
    int y = 0;
    screen->Line( 100 + x, 50 + y, 200 + x, 50 + y, 0xffffff );
    screen->Line( 150 + x, 50 + y, 150 + x, 300 + y, 0xffffff );
    screen->Line( 100 + x, 300 + y, 200 + x, 300 + y, 0xffffff );
}
```

Here we have the original three lines again, but this time, each coordinate has either '`+ x`' or '`+ y`' attached to it. X and Y are variables here, and you may give them any name you like. The number they represent is simply added to the coordinate. This is useful; we can now make the 'I' move. Try this:

```
int x = 100;
int y = 0;
void Game::Tick( float deltaTime )
{
    screen->Clear( 80 );
    screen->Line( 100 + x, 50 + y, 200 + x, 50 + y, 0xffffff );
    screen->Line( 150 + x, 50 + y, 150 + x, 300 + y, 0xffffff );
    screen->Line( 100 + x, 300 + y, 200 + x, 300 + y, 0xffffff );
    x = x + 1;
}
```

Now, if we could draw those three lines multiple times with different variable values, we would be all set: we could draw super-fat characters; heck, we could even move them across the screen! It's not a bullet yet, but it's a start.

By the way, something odd happened to that '`int x = 100` etc.' line: It was moved *outside* the `Tick` function. When it was inside, x and y where reset for every image, because the line '`int x = 100, y = 0;`' is executed for every frame. Since they are supposed to be set to their initial values only in the first frame, you would expect that they should go to the `Init` function. However, one function cannot access variables in another function in C/C++, and even if two variables in different functions have identical names, they still will be two different things. Variables created *outside* a function however are available in *all* functions. And, they are initialized only once. *In C/C++ terminology, the variables x and y used to be 'declared within the scope of the Tick method', and now they are 'declared in the global scope'.*

Functions

A function is a piece of code. You have seen three of them so far: `Game::Init()`, `Game::Tick()`, and `Game::Shutdown()`. We can create our own too. Go to game.h and add the following line:

```
void DrawI( int x, int y );
```

You can put it in many correct locations, but between `Init` and `Tick` is probably a good place. Then, return to `game.cpp`, and add the following function:

```
void Game::DrawI( int x, int y )
{
    screen->Line( 100 + x, 50 + y, 200 + x, 50 + y, 0xffffff );
    screen->Line( 150 + x, 50 + y, 150 + x, 300 + y, 0xffffff );
    screen->Line( 100 + x, 300 + y, 200 + x, 300 + y, 0xffffff );
}
```

And, change the `Tick` function to:

```
void Game::Tick( float deltaTime )
{
    screen->Clear( 80 );
    DrawI( 0, 0 );
    x++;
}
```

In other words: When we type `DrawI( 0, 0 )`, we actually execute the code specified in `Game::DrawI()`. And, whatever we type after `DrawI` (in this case: 0, 0) gets passed to the function: Inside the function, x and y will now be 0 and 0. So, our fat character code becomes:

```
void Game::Tick( float deltaTime )
{
    screen->Clear( 80 );
    DrawI( 0, 0 );
    DrawI( 1, 0 );
    DrawI( 0, 1 );
    DrawI( 1, 1 );
    x++;
}
```

And that's quite a bit more reasonable.

By the way, the 'Clear' and 'Line' that you have been using all along are functions too! Have a look at surface.cpp, line 140:

```
void Surface::Line( float x1, float y1, float x2, float y2, Pixel c )
{
    ...
    float b = x2 - x1;
    float h = y2 - y1;
    float l = fabsf( b );
    if (fabsf ( h ) > l) l = fabsf( h );
    int il = (int)l;
    float dx = b / (float)l;
    float dy = h / (float)l;
    for ( int i = 0; i <= il; i++ )
    {
        *(m_Buffer + (int)x1 + (int)y1 * m_Pitch) = c;
```

```
        x1 += dx, y1 += dy;
    }
}
```

Not everything in this function will make sense right now, but rest assured: this is how you draw a proper line in C/C++.

Assignment

Here's your task for today:

1. Create a function that draws a fat I. Call it '`DrawFatI`'. This function will use `DrawI` in turn, of course.
2. Modify `Game::Tick()` in such a way that the fat I moves across the screen.

Once you have completed this assignment you may continue with the next part.

END OF PART 3

Next part: "Loops"