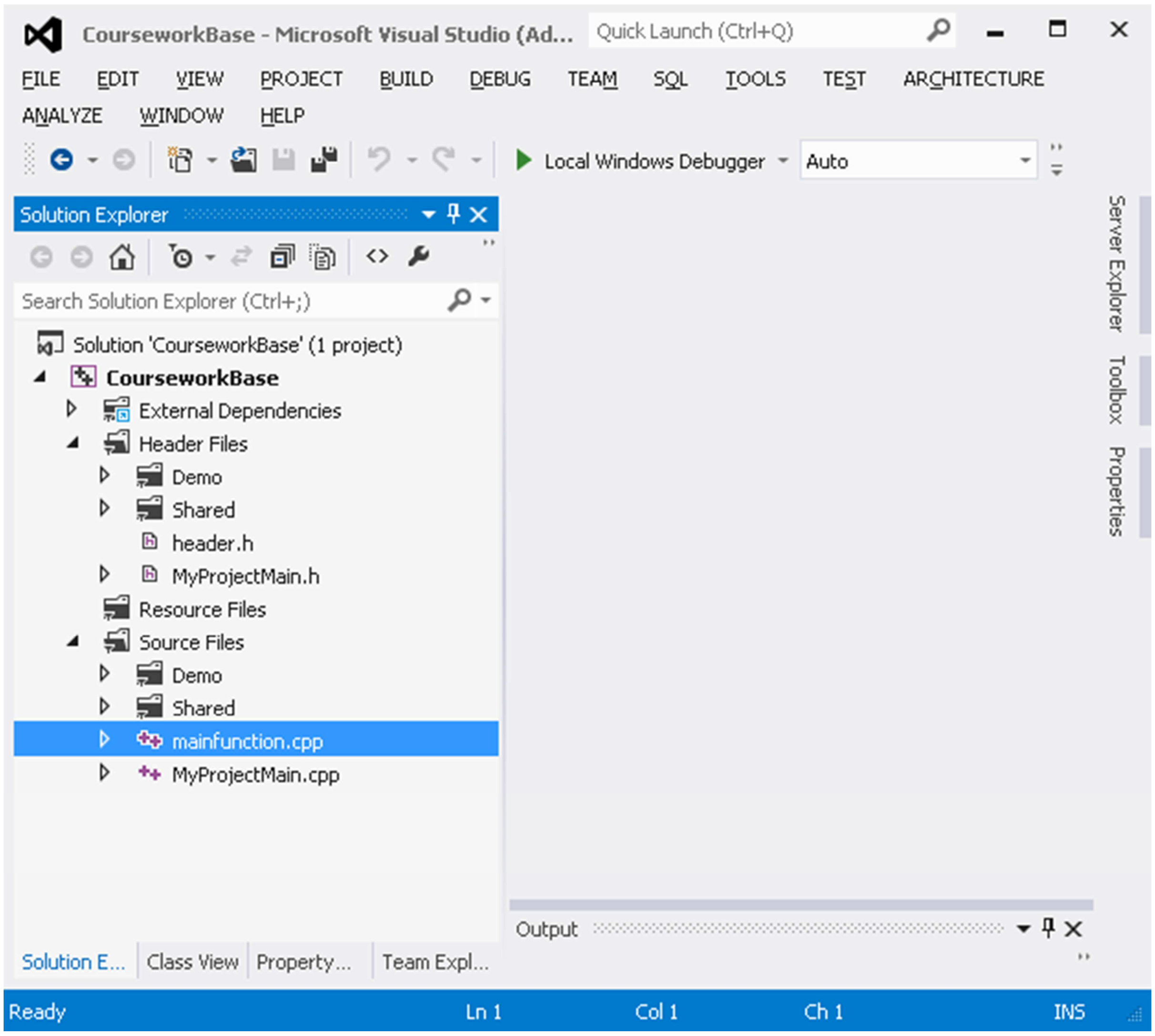
Coursework Lab A

The aim of this lab session is for you to learn the basics of how the coursework framework works. In this first of two lab sessions you will learn about drawing backgrounds and handling input. In the second you will learn about moving objects – drawing them and controlling them, then a couple of other things like drawing strings and drawing tiles on the background.

Open the coursework project



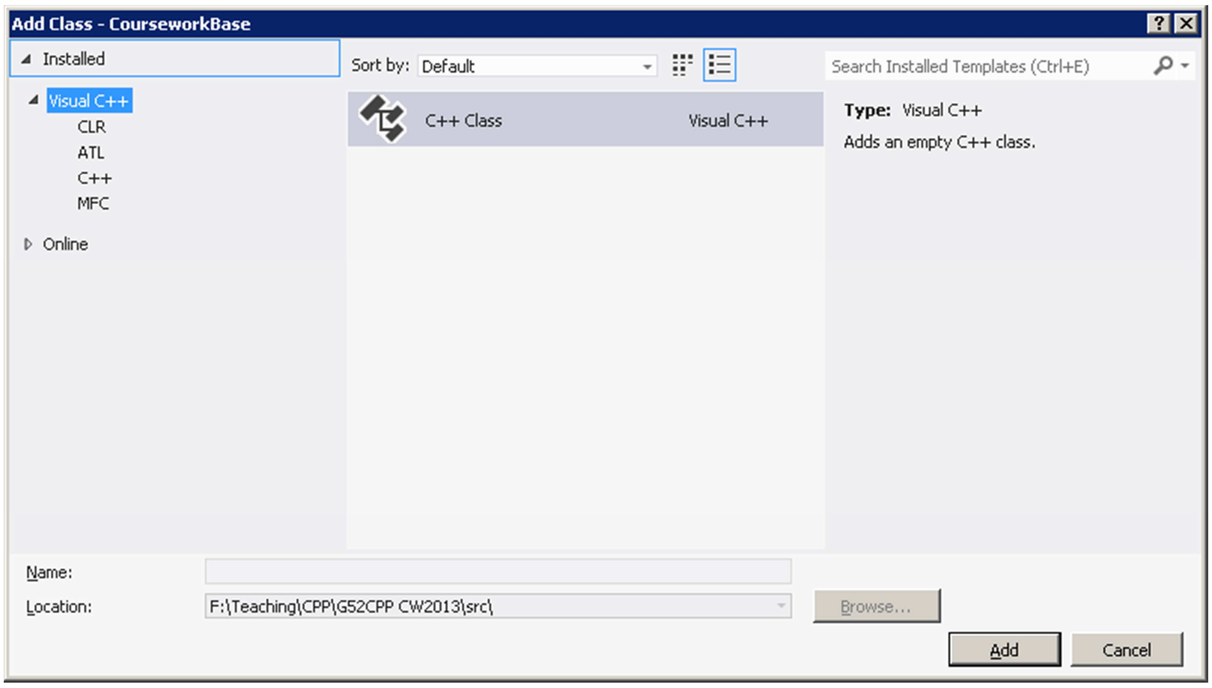
CourseworkBase - Microsoft Visual Studio (Ad...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Local Windows Debugger
Auto
Solution Explorer
Search Solution Explorer (Ctrl+;)
Server Explorer
Toolbox
Properties
Output
Solution E... Class View Property... Team Expl...
Ready Ln 1 Col 1 Ch 1 INS

```
Solution 'CourseworkBase' (1 project)
  CourseworkBase
    External Dependencies
    Header Files
      Demo
      Shared
      header.h
      MyProjectMain.h
    Resource Files
    Source Files
      Demo
      Shared
      mainfunction.cpp
      MyProjectMain.cpp
```

Add a new class for the main program:

Right click on Source Files and choose “Add Class”.

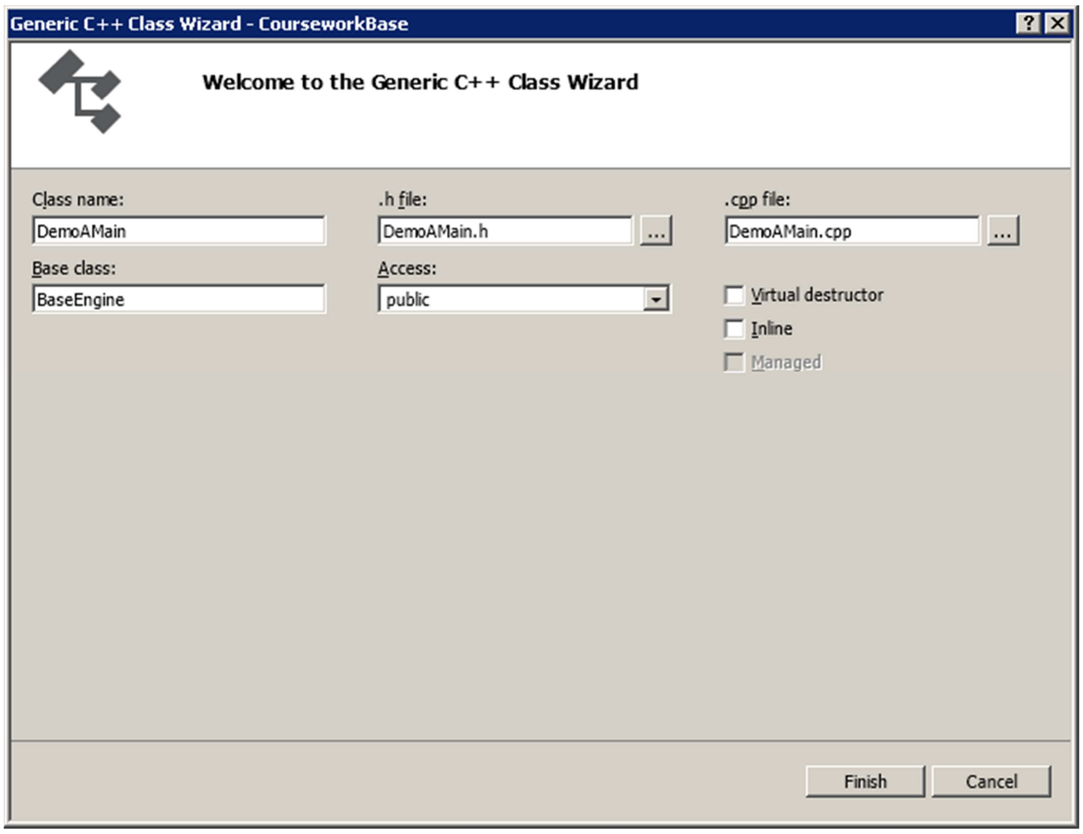
The dialog below will then appear. Choose “C++ Class” and press the Add button.



Type: Visual C++
Adds an empty C++ class.

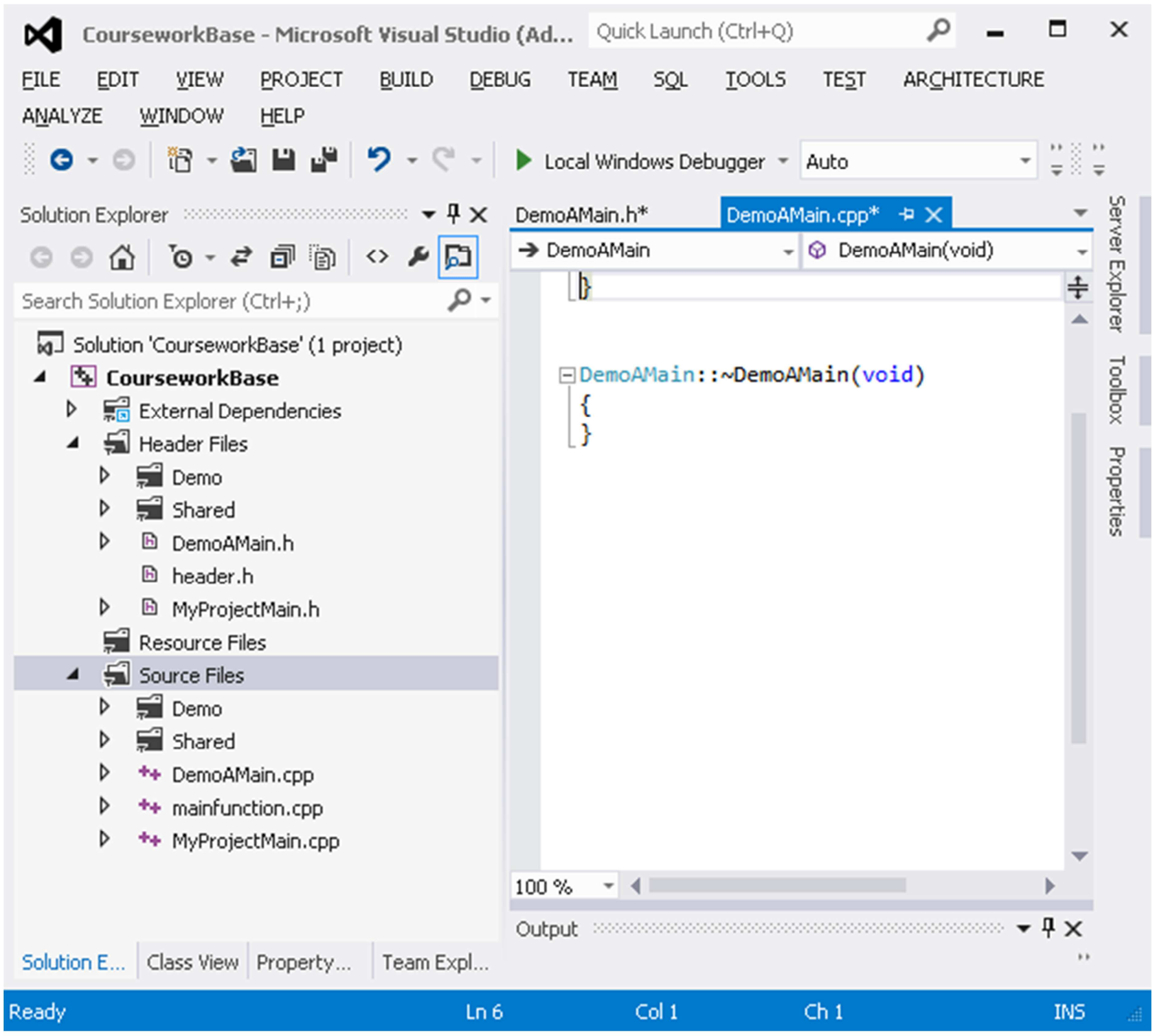
Name:
Location: F:\Teaching\CPP\G52CPP CW2013\src\

Choose “BaseEngine” as the base class and “DemoAMain” as the Class Name, then press Finish.


Generic C++ Class Wizard - CourseworkBase
Welcome to the Generic C++ Class Wizard
Finish
Cancel

Class name: DemoAMain
.h file: DemoAMain.h
.cpp file: DemoAMain.cpp
Base class: BaseEngine
Access: public
☐ Virtual destructor
☐ Inline
☐ Managed

A new class will have been created and both a header and a cpp file will have been added to your project (see next page):


CourseworkBase - Microsoft Visual Studio (Ad...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Local Windows Debugger
Auto
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'CourseworkBase' (1 project)
CourseworkBase
External Dependencies
Header Files
Demo
Shared
DemoAMain.h
header.h
MyProjectMain.h
Resource Files
Source Files
Demo
Shared
DemoAMain.cpp
mainfunction.cpp
MyProjectMain.cpp
DemoAMain.h*
DemoAMain.cpp*
DemoAMain
DemoAMain(void)
100 %
Output
Solution E... Class View Property... Team Expl...
Server Explorer Toolbox Properties
Ready Ln 6 Col 1 Ch 1 INS

```
}

DemoAMain::~DemoAMain(void)
{
}
```

Further information if you want it:

Your header files will be organised under the "Header files" group and your source files under the "Source Files" group. Within each group I have divided the files into three categories: the uncategorised files (e.g. header.h) the Demo files (open these to see the various demos) and the shared files, which are basically the coursework framework. These shared files include the base classes that you will inherit your new classes from. Please feel free to take some time to investigate these files. You may find the framework documentation useful for this: http://www.cs.nott.ac.uk/~jaa/cpp/cw/SDL_framework_docs.pdf

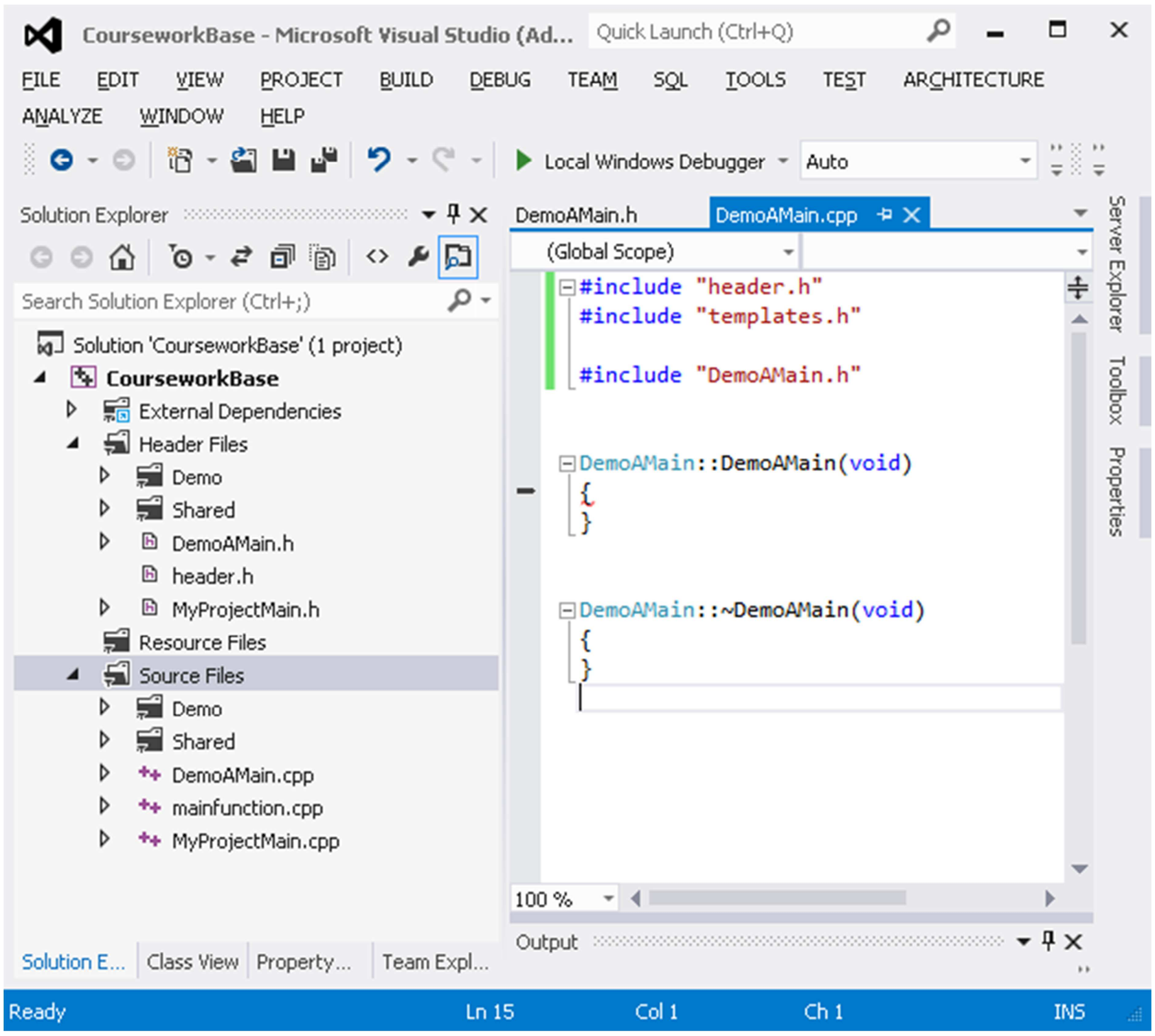
When you use the wizard to create a new class, it will usually put the .h file under Header Files and the .cpp file under Source Files (which is probably where you want them). Where the files are categorised is irrelevant: these 'folders' have no effect upon the program, they just help you to organise you files within the development tool.

Include the standard header files for the framework

Before doing anything else, go to the top of the file and add the following lines:

```
#include "header.h"
#include "templates.h"
```

As in the picture below:



```
#include "header.h"
#include "templates.h"

#include "DemoAMain.h"


DemoAMain::DemoAMain(void)
{
}


DemoAMain::~DemoAMain(void)
{
}
```

Important Note: You need to include the file "header.h" at the top of ALL of your sourcecode files. If you do not, they may not compile. SDL changes some things and to make it work the SDL headers have to be added to all of the files. If you look at header.h (in Header Files) you will see that it includes SDL.h and SDL_ttf.h, to ensure that these are added to all source files.

Alter the constructor

Now you have a basic implementation of the main class, and inherit the behaviour of the base class, but you need to pass the correct values to the base class constructor. You will see this in lectures later, but for the moment just make the following changes:

Go to the DemoAMain.cpp file. File the constructor:

```
DemoAMain::DemoAMain(void)
{
}
```

Add the line

```
       : BaseEngine( 6 )
so that it reads:
```

```
DemoAMain::DemoAMain(void)
       : BaseEngine( 6 )
{
}
```

This is how we pass values to a superclass constructor in C++. The BaseEngine(6) says pass 6 to the constructor of the base class called BaseEngine. The : means that this is an initialisation list. We will see both of these concepts in more detail in lectures later and for the moment they are not too important. The only important thing to know here is that the 6 that is passed is the maximum number of moving objects that you will be able to redraw. For these examples 6 will be sufficient. If you later want to have a lot of these you may want to increase this number for your own program.

You code should now compile.

Go to mainfunction.cpp

Near the top, below: `#include "Demo3Main.h"`
Add #include “DemoAMain.h” to include your new header file, so that it reads:

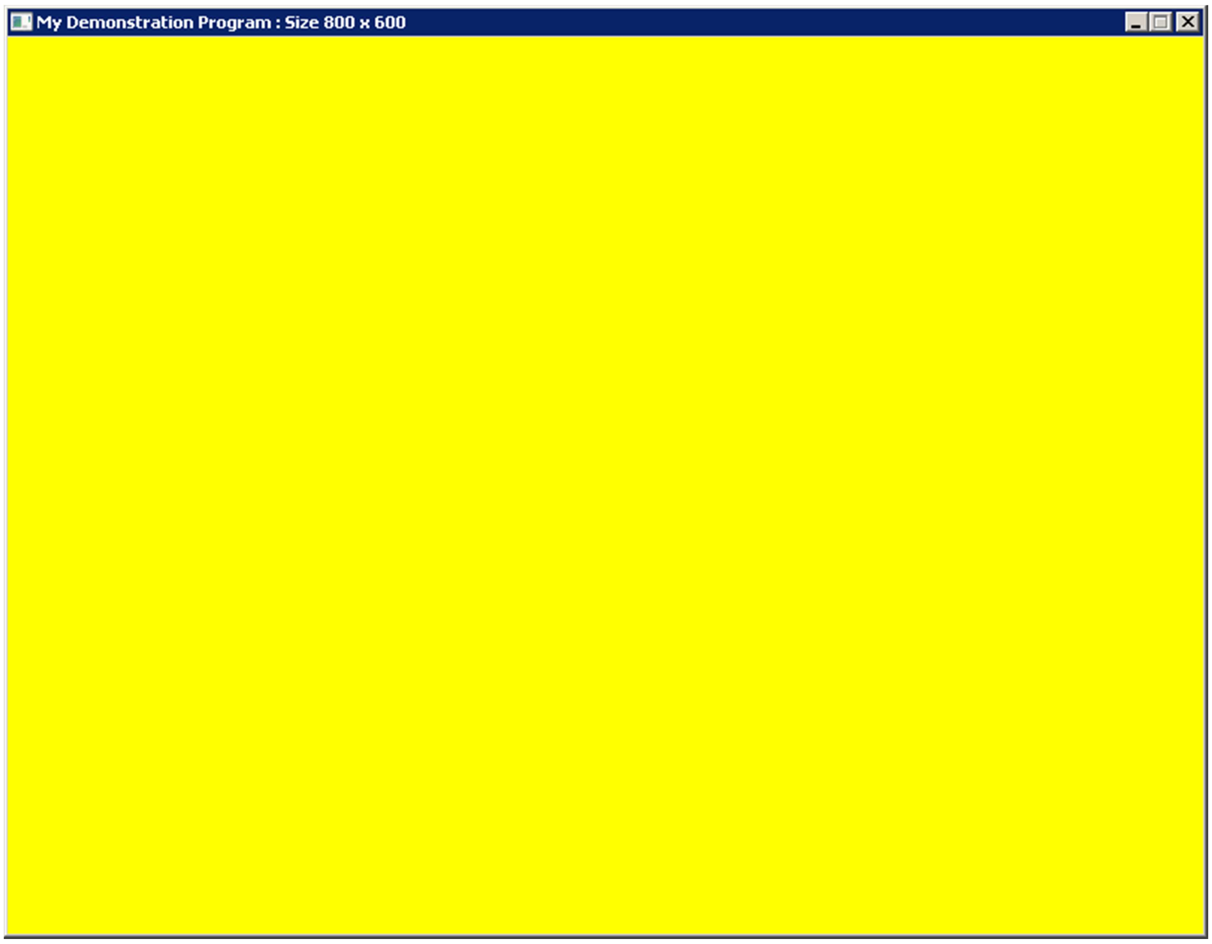
```
#include "Demo2aMain.h"
#include "Demo3Main.h"
#include "DemoAMain.h"
```

Next create an object of your new class (which you compiler will be able to identify because you included the header file) in the main function, below the line `//Demo3Main oMain;` , and comment out the line `MyProjectMain oMain;` so that it reads:

```
       // Needs just one of the two following lines:
       //BouncingBallMain oMain;
       //MyProjectMain oMain;
       //Demo2Main oMain;
       //Demo2aMain oMain;
       //Demo3Main oMain;
       DemoAMain oMain;
```

Now build (build menu, choose “build solution” or “rebuild solution”) and execute (Debug menu, choose “start debugging” or “start without debugging”) your program.

Your program should look like this:



The line:

```
        iResult = oMain.Initialise( buf, BASE_SCREEN_WIDTH, BASE_SCREEN_HEIGHT,
"Cornerstone Regular.ttf", 24 );
```

in mainfunction.cpp is important. It specifies the window caption (see the previous sprintf which set up the value of buf, which is why I encouraged you to learn how strings work in C), how large the window is (the two #defines for the screen width and height) and the default font to use for text. "`Cornerstone Regular.ttf`" is a font file in the directory where the exe is built.

The window is using a default draw function. We will now experiment to change the way in which drawing is performed.

You can try out the other demos at this point if you wish, to see what is available. To do so, just change which line is not commented out.

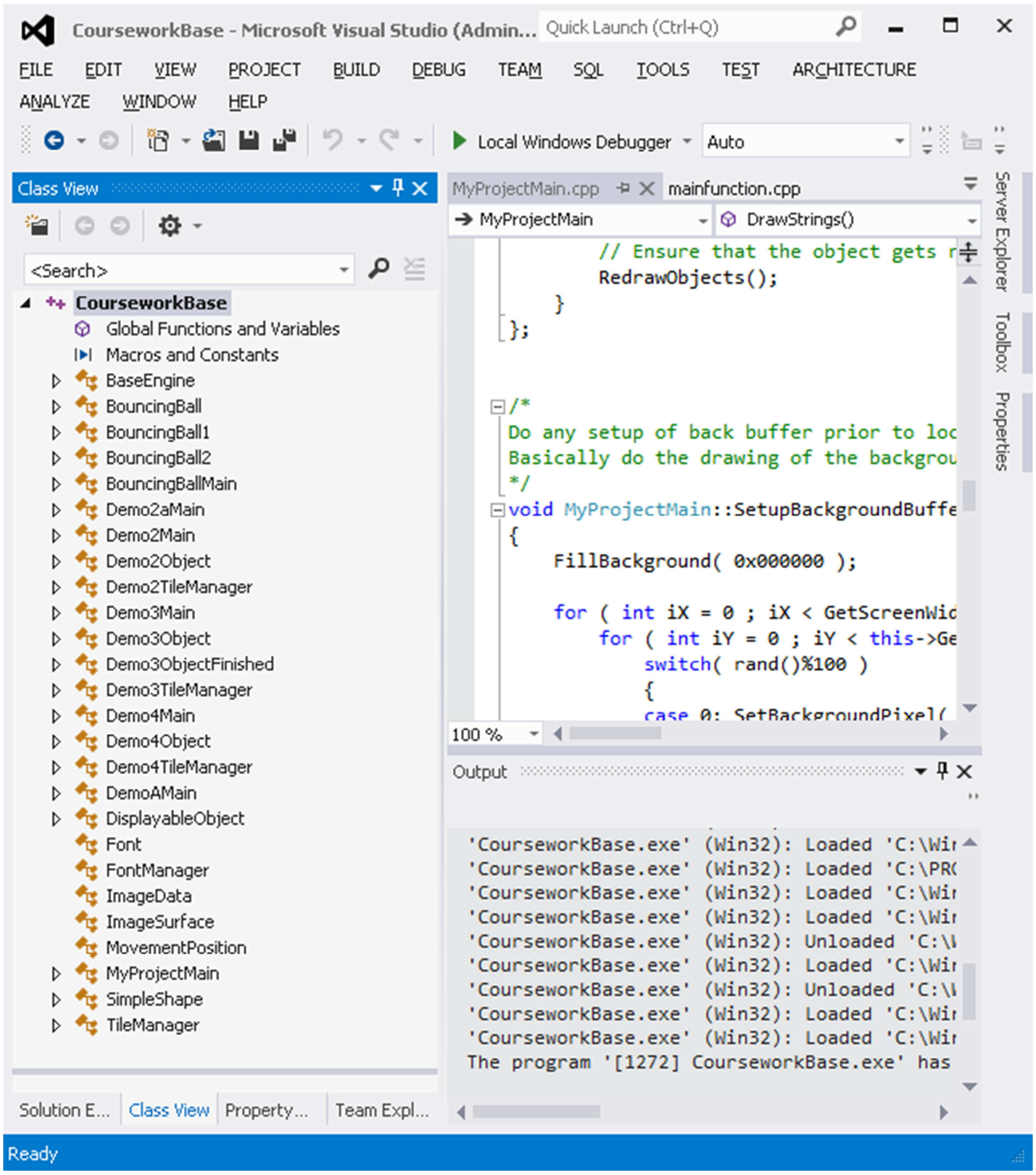
e.g. to run the BouncingBall demo uncomment out that line and ensure the others are commented out:

```
        // Needs just one of the two following lines:
        BouncingBallMain oMain;
        //MyProjectMain oMain;
        //Demo2Main oMain;
        //Demo2aMain oMain;
        //Demo3Main oMain;
        //DemoAMain oMain;
```

Drawing your own background

The framework keeps a background buffer for you. This is something that you can draw to and then copy onto the screen to overwrite anything drawn there already.

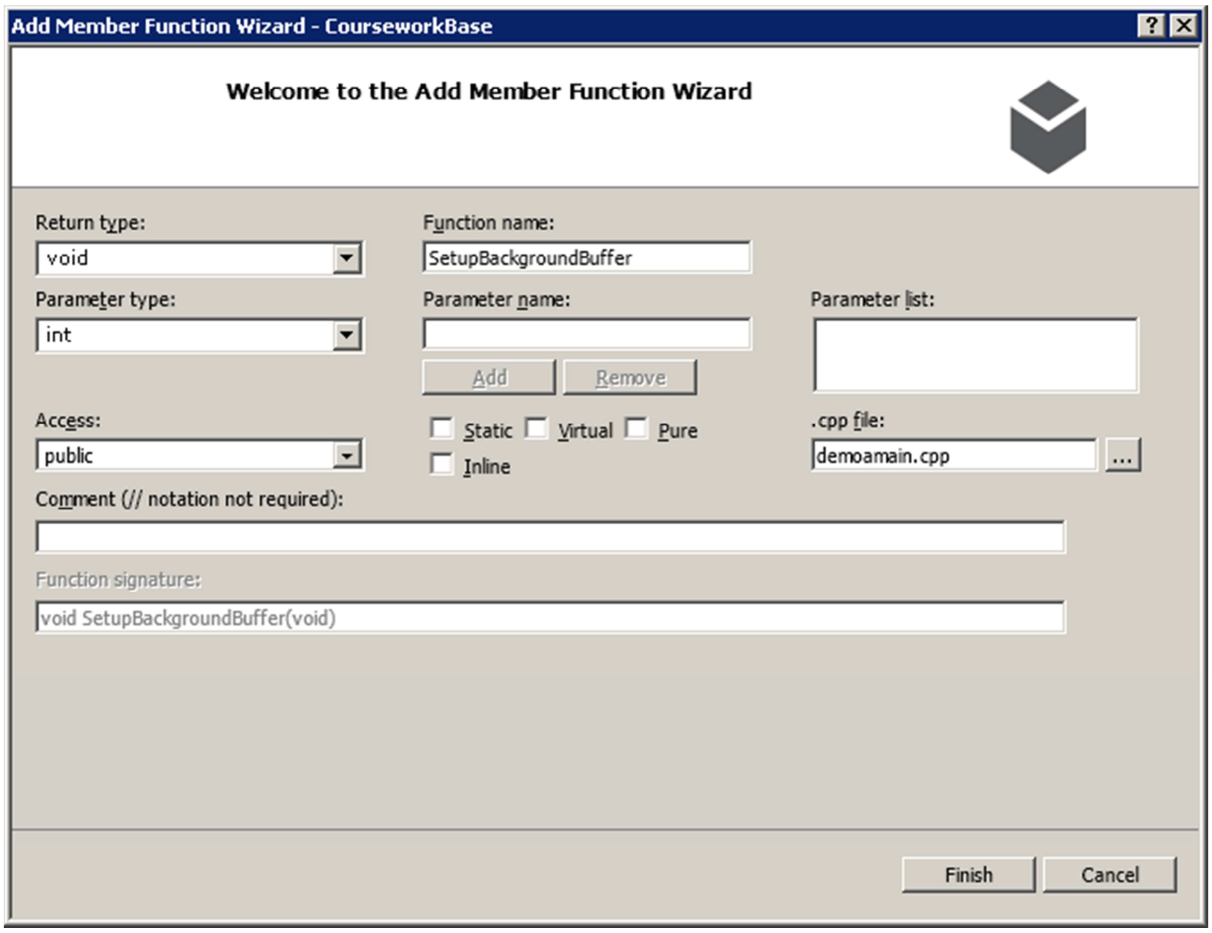
Switch to the class view (it's an option at the bottom of the solution explorer). This will show classes instead of files. Open up the coursework base project and you will see something like this:



```
            // Ensure that the object gets r
            RedrawObjects();
        }
};


/*
Do any setup of back buffer prior to loc
Basically do the drawing of the backgrou
*/
void MyProjectMain::SetupBackgroundBuffe
{
    FillBackground( 0x000000 );

    for ( int iX = 0 ; iX < GetScreenWic
        for ( int iY = 0 ; iY < this->Ge
            switch( rand()%100 )
            {
            case 0: SetBackgroundPixel(
```

```
'CourseworkBase.exe' (Win32): Loaded 'C:\Wir
'CourseworkBase.exe' (Win32): Loaded 'C:\PR(
'CourseworkBase.exe' (Win32): Loaded 'C:\Wir
'CourseworkBase.exe' (Win32): Loaded 'C:\Wir
'CourseworkBase.exe' (Win32): Unloaded 'C:\W
'CourseworkBase.exe' (Win32): Loaded 'C:\Wir
'CourseworkBase.exe' (Win32): Unloaded 'C:\W
'CourseworkBase.exe' (Win32): Loaded 'C:\Wir
'CourseworkBase.exe' (Win32): Loaded 'C:\Wir
The program '[1272] CourseworkBase.exe' has
```

Right click on DemoAMain and choose "Add Function". You will then get the following dialog shown:


Add Member Function Wizard - CourseworkBase

Welcome to the Add Member Function Wizard

Return type: void

Function name: SetupBackgroundBuffer

Parameter type: int

Parameter name:

Parameter list:

Access: public

Static [ ] Virtual [ ] Pure [ ] Inline [ ]

.cpp file: demoamain.cpp

Comment (// notation not required):

Function signature: void SetupBackgroundBuffer(void)

Set the return type to void and the function name to "SetupBackgroundBuffer", as above, then press finish.

This will insert code into your project as follows:
In DemoAMain.h:

```
void SetupBackgroundBuffer(void);
```

In DemoAMain.cpp:

```
void DemoAMain::SetupBackgroundBuffer(void)
{
}
```

Try running the program now and it will no longer give a yellow window. The yellow window was in the base class implementation of SetupBackgroundBuffer. We have just given our own implementation and made it do nothing.

Go to the `SetupBackgroundBuffer` function in the .cpp file and add the following code:

```
void DemoAMain::SetupBackgroundBuffer(void)
{
        FillBackground( 0xff0000 );
}
```

Build and run the program and it will have a red background. The 0xff0000 is a hexadecimal colour code. The first two digits are the amount of red (00 to ff), the next two the amount of green and the last two the amount of blue. Try some other colours before you continue.

Now try the following code instead:

```
void MyProjectMain::SetupBackgroundBuffer()
{
	FillBackground( 0x000000 );

	for ( int iX = 0 ; iX < GetScreenWidth() ; iX++ )
		for ( int iY = 0 ; iY < this->GetScreenHeight() ; iY++ )
			switch( rand()%100 )
			{
			case 0: SetBackgroundPixel( iX, iY, 0xFF0000 ); break;
			case 1: SetBackgroundPixel( iX, iY, 0x00FF00 ); break;
			case 2: SetBackgroundPixel( iX, iY, 0x0000FF ); break;
			case 3: SetBackgroundPixel( iX, iY, 0xFFFF00 ); break;
			case 4: SetBackgroundPixel( iX, iY, 0x00FFFF ); break;
			case 5: SetBackgroundPixel( iX, iY, 0xFF00FF ); break;
			}
}
```

Try to work out what this does before running it or continuing. Then run it to see what happens. Try to understand what it is doing before reading the explanation below.

Behind the scenes:

This method of adding functions and seeing features appear works because the functions such as SetupBackgroundBuffer(), MouseDown(), KeyDown() etc are all implemented in the base class called BaseEngine, of which your class is a subclass.

These functions are all virtual functions (see later lecture) and by implementing a version in your class, your version will be called instead of the base class version.

Answer:

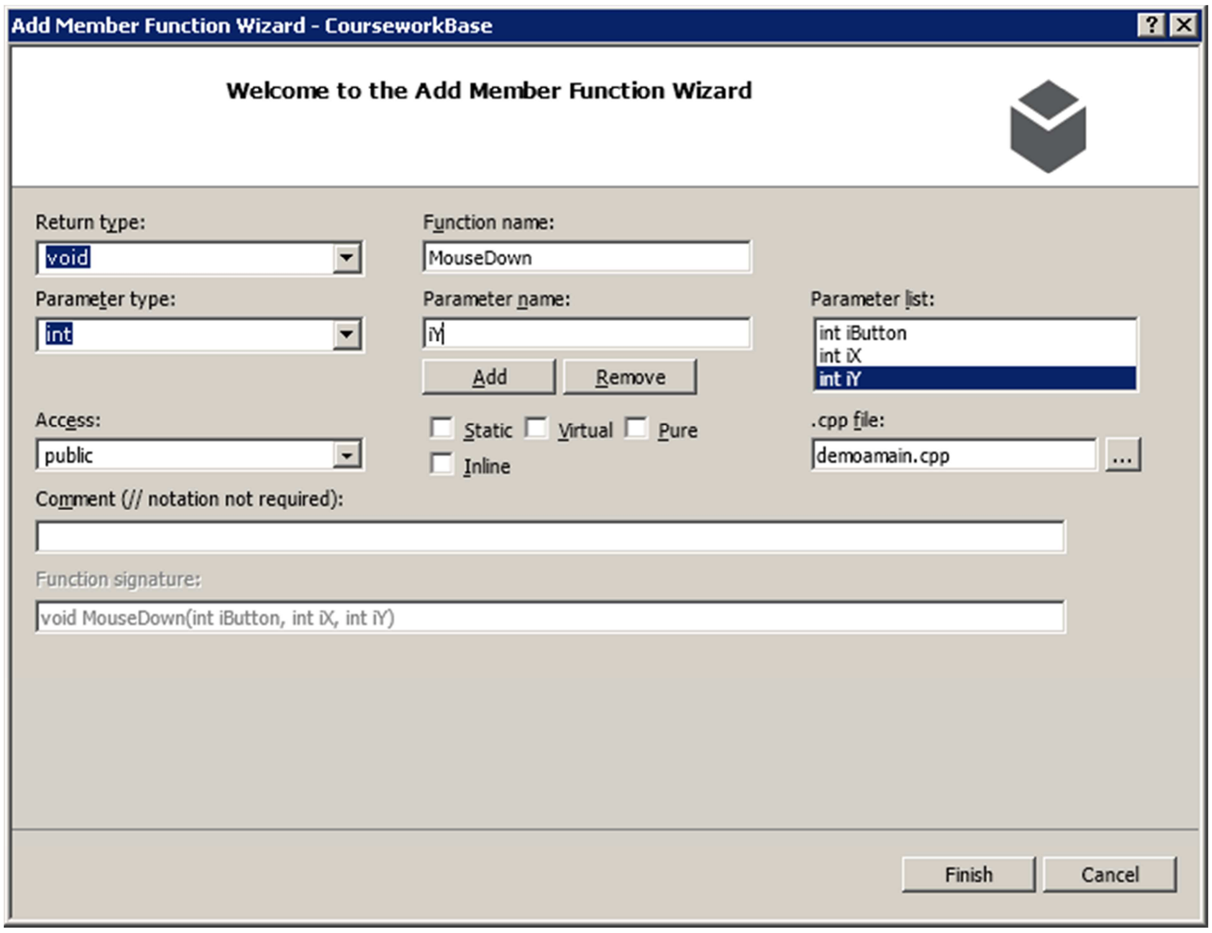
This code has two loops, forcing it to consider every pixel in the background in turn. i.e. it iterates every column of the screen, and for each column it iterates through each pixel in that column.

For each pixel it chooses a random number. It then takes the remainder when that number is divided by 100, giving it a random number from 0 to 99, with each value equally likely.

If the number is a 0 it will set the pixel to Red (0xff0000). If it is 1 then it will be green. If 2 then it is blue. If 3 then it is yellow (red+green). If 4 then it is cyan (green+blue). If 5 then it is magenta (red+blue). Any other value will leave the pixel unchanged. So the majority will be unchanged, but some will be coloured.

Now we will add some interaction:

Add a new function called MouseDown as follows, this will handle mouse presses.


Add Member Function Wizard - CourseworkBase

Welcome to the Add Member Function Wizard

Return type: void

Function name: MouseDown

Parameter type: int

Parameter name: iY

Parameter list:
int iButton
int iX
int iY

Access: public

Static ☐ Virtual ☐ Pure ☐ Inline ☐

.cpp file: demoamain.cpp

Comment (// notation not required):

Function signature: void MouseDown(int iButton, int iX, int iY)

Note: Return type void, name “Mouse Down” and add three parameters (set the name then press Add to add each in turn) called iButton, iX and iY. All are of integer type.

When you choose Finish the following function will be created:

```
void DemoAMain::MouseDown(int iButton, int iX, int iY)
{
}
```

Add the following code to draw a circle on the FOREGROUND when the left button is pressed:

```
void DemoAMain::MouseDown(int iButton, int iX, int iY)
{
	printf( "%d %d\n", iX, iY );

	if ( iButton == SDL_BUTTON_LEFT )
	{
		DrawRectangle( iX-10, iY-10, iX+10, iY+10, 0xffff00 );
		SetSDLUpdateRectImmediately( iX-10, iY-10, 21, 21 );
	}
}
```

The first line will cause some debug info to go to the output, so you can see what the x and y coordinates were of the click.

Then the if statement verifies that the left mouse button was clicked.

The DrawRectangle function will draw a rectangle on the foreground, from X-10 to X+10 and Y-10 to Y+10 (i.e. height and width of 21).

SDL needs to know when you draw something. The next line tells SDL that you have changed something on the screen, telling it that the rectangle around the change starts at (x-10,y-10) and has width 21 and height 21.

Behind the scenes:

Functions like DrawRectangle(), DrawOval() and SetSDLUpdateRectImmediately() are implemented in the base class. Your class inherits these functions so you can call them to 'draw a rectangle' or the tell SDL that the screen has changed.

The MouseDown() function is another of these virtual functions in the base class. You can look for it in BaseEngine.cpp if you wish. You will see that the base class version does nothing.

The BaseEngine code will check for events constantly, and every time it gets a mouse down event it will call the MouseDown() function. In the base class this will do nothing, but as soon as you add your own implementation, it will start to do something.

You can now add some code for the right mouse button as well:

```
void DemoAMain::MouseDown(int iButton, int iX, int iY)
{
	printf( "%d %d\n", iX, iY );

	if ( iButton == SDL_BUTTON_LEFT )
	{
		DrawRectangle( iX-10, iY-10, iX+10, iY+10, 0xffff00 );
		SetSDLUpdateRectImmediately( iX-10, iY-10, 21, 21 );
	}
	else if ( iButton == SDL_BUTTON_RIGHT )
	{
		DrawOval( iX-10, iY-10, iX+10, iY+10, 0xff0000 );
		SetSDLUpdateRectImmediately( iX-10, iY-10, 21, 21 );
	}
}
```

Compile and execute this to try it.

Try the following change (comment out a line and add Redraw(true), and see what happens when you right-click the mouse button:

```
void DemoAMain::MouseDown(int iButton, int iX, int iY)
{
	printf( "%d %d\n", iX, iY );

	if ( iButton == SDL_BUTTON_LEFT )
	{
		DrawRectangle( iX-10, iY-10, iX+10, iY+10, 0xffff00 );
		SetSDLUpdateRectImmediately( iX-10, iY-10, 21, 21 );
	}
	else if ( iButton == SDL_BUTTON_RIGHT )
	{
		DrawOval( iX-10, iY-10, iX+10, iY+10, 0xff0000 );
		//SetSDLUpdateRectImmediately( iX-10, iY-10, 21, 21 );
		Redraw(true);
	}
}
```

When you right click, everything that you drew to the foreground will vanish. Basically, a redraw will redraw the background (the original thing which we drew in the SetupBackgroundBuffer function), and will overwrite everything that we draw onto the foreground. This can be really useful, but also really confusing until you understand it.

Behind the scenes:

The framework will sit in a loop, constantly repeating the following steps: (look for MainLoop in BaseEngine.cpp to see the code)

1) Handle any events, e.g. key presses, mouse presses etc, and call the relevant functions.
2) Call GameAction(). This will eventually call the DoUpdate() functions on the moving objects which you will learn about in the next lab. This function is responsible for changing anything, e.g. moving any objects around or handling some user input.
3) Call GameRender(). This function is responsible for re-drawing the screen if necessary.

GameRender() uses two variables to determine whether anything has to be redrawn. The first says that something changed (see `m_bNeedsRedraw` in BaseEngine) and the second says that the whole screen needs to be redrawn (`m_bWholeScreenUpdate).` The screen will only be redrawn if the first flag is set. If the second flag is set then the whole screen will be redrawn, if not then only the moving objects (see Coursework Lab B) will be redrawn. Redraw(true) says to redraw the whole screen.

The way this works is:

- If nothing changes on the screen, GameRender() will never need to redraw anything.
- If something moves around the screen, then it is responsible for recording which part of the screen it changed and calling Redraw(false). GameRender() will then effectively look up which parts of the screen to draw and draw them (the details are a little more complicated, since it asks objects to draw themselves, but this is the basic principle).
- If something changes a lot of the screen, or does not want to record which parts it changed, then it is worth just calling Redraw(true). This will force a redraw of the whole screen.

The background and foreground

Think of there being two different copies of the screen – a background and a foreground. When you draw moving objects (see next lab) you will draw to the foreground. This means that an object can be moved by redrawing the background (removing the old position of the object) then drawing it again in the new position. You can only do this because you have a copy of the background as well, to draw over the objects to remove them.

Rules:

1) If you are drawing a static item them draw it to the background, otherwise it will be eliminated when you redraw the screen.
2) If you are drawing a moving object then draw it to the foreground – you WANT it to be removed when you redraw the screen, to remove it from the old position on the screen.

Since you want the things you draw on a mouse click the stay after a redraw, instead draw them to the background as shown below:

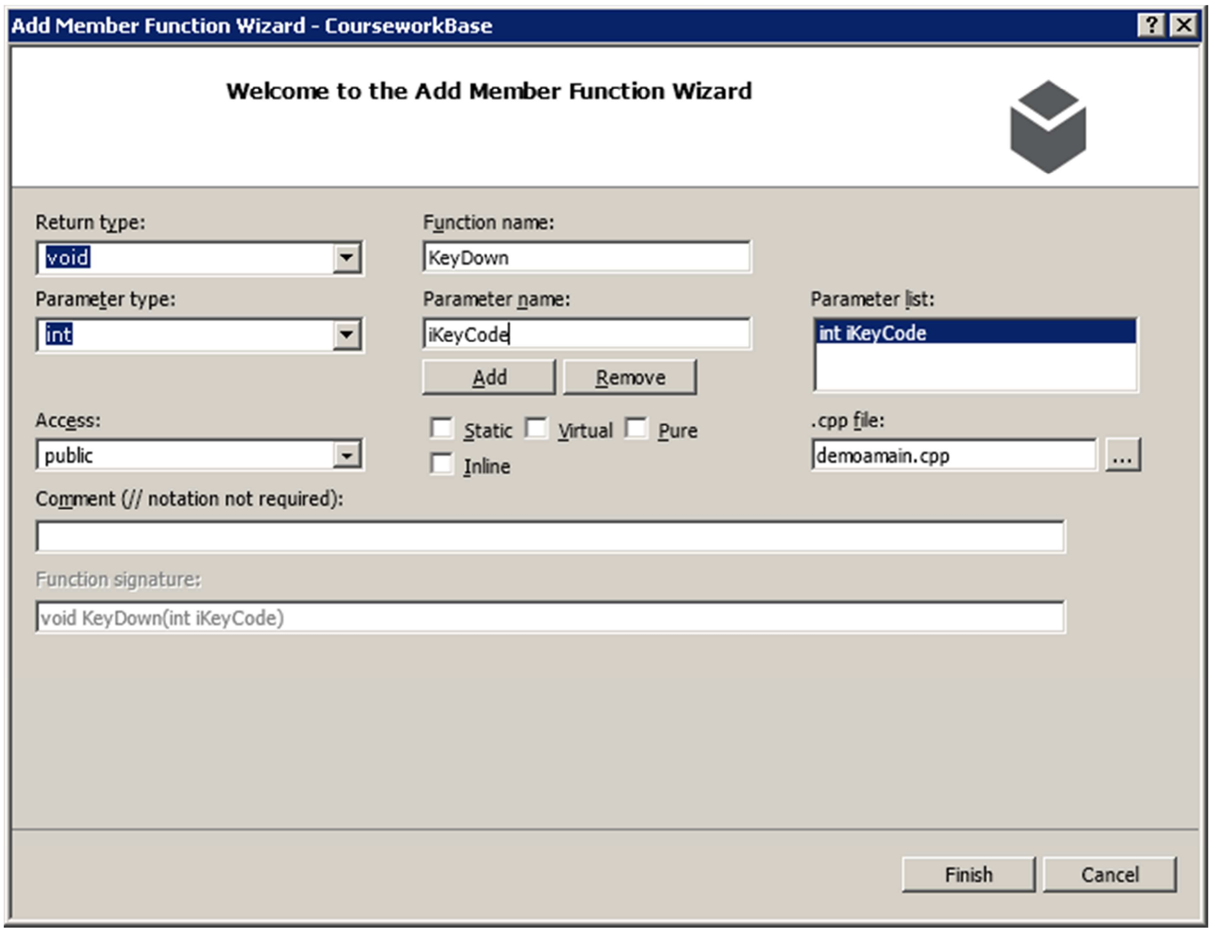
```
void DemoAMain::MouseDown(int iButton, int iX, int iY)
{
	printf( "%d %d\n", iX, iY );

	if ( iButton == SDL_BUTTON_LEFT )
	{
		DrawBackgroundRectangle( iX-10, iY-10, iX+10, iY+10, 0xffff00 );
		//SetSDLUpdateRectImmediately( iX-10, iY-10, 21, 21 );
		Redraw(true);
	}
	else if ( iButton == SDL_BUTTON_RIGHT )
	{
		DrawBackgroundOval( iX-10, iY-10, iX+10, iY+10, 0xff0000 );
		//SetSDLUpdateRectImmediately( iX-10, iY-10, 21, 21 );
		Redraw(true);
	}
}
```

You will now see that the shapes stay when you redraw the screen. This will become more important when you see moving objects, because when the moving objects go over these they will not eliminate them.

Handling key presses

You can handle key presses as the key is pressed down using the following method.

First add the KeyDown function as specified below (note the single integer parameter):



Add Member Function Wizard - CourseworkBase

Welcome to the Add Member Function Wizard

Return type: void

Function name: KeyDown

Parameter type: int

Parameter name: iKeyCode

Parameter list: int iKeyCode

Access: public

Static Virtual Pure Inline

.cpp file: demoamain.cpp

Comment (// notation not required):

Function signature: void KeyDown(int iKeyCode)

This will create you the following function:

```
void DemoAMain::KeyDown(int iKeyCode)
{
}
```

Then provide the following implementation:

```
void DemoAMain::KeyDown(int iKeyCode)
{
        switch( iKeyCode )
        {
        case ' ':
                SetupBackgroundBuffer();
                Redraw(true);
                break;
        }
}
```

This will redraw the entire background again, calling the function which we specified earlier, drawing over any existing images/shapes, when SPACE is pressed. You can handle other keys in a similar way.

If you need to handle other characters, you can use the SDL keycode, usually named SDLK_ then the name of the key. See http://wiki.libsdl.org/moin.cgi/SDLKeycodeLookup for a list, or look in SDL_keysym.h. e.g.:

```
void DemoAMain::KeyDown(int iKeyCode)
{
	switch( iKeyCode )
	{
	case SDLK_SPACE:
		SetupBackgroundBuffer();
		Redraw(true);
		break;
	}
}
```

If just happens that most are set to the same as the printable ASCII character, so you can just use ‘a’ for example, rather than SDLK_a. (Well it’s deliberate, but very useful.)

What now?

Next time we will consider how to handle moving items, but this should be enough to get you started with knowing how to draw a background and how to handle both mouse presses and key presses. Please take some time to experiment, and to look at what functions are available in the BaseEngine.h file. You will find that there are quite a few useful drawing functions. Hopefully you have also seen that it is important to draw things that you want to keep onto the background not the foreground.