# Software Architecture for Games

## Week 4

The plan for this week is:

* To get you used to compiling the Shell Engine.
* To use the engine to draw a picture on the screen.
* To learn how to move the picture around.
* To learn how to get user input from the keyboard.
* To learn how to use the engine to play a sound effect.

We are going to start using the Shell Engine today. To use the engine you will need:

1. Visual Studio 2015. This is already in the lab. You can easily get the Community edition to use at home.
2. Microsoft DirectX SDK June 2010. This is already in the lab. If you plan to work at home, you will need to install it there. You can download it for free from Microsoft’s DirectX site.
3. Shell2020.zip. This is the current version of the engine and is on Blackboard.

**To compile the engine**

1. Run Visual Studio 2015.
2. Create a Windows project. It needs to be an empty project.
3. Extract the files in Shell2020.zip to the correct folder. This will be a subdirectory of your project directory, with the same name. For example:

MyDocuments\VisualStudio 2015\Projects\MyFirstGame\MyFirstGame

1. Use Project ... Add ... Existing files. This will open a file dialogue. If the files are not in the right folder, move the files. Do not select a different folder.
2. Go to Project .... Properties... VC++ Directories ... Include Directories and add a new folder. This will be the location of the DirectX SDK header files and is probably “C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Include;”. If you run into problems here, you should check:
   1. There is a semicolon on the end. It may need it.
   2. Try putting this folder first in the list. If that doesn’t work, try last.
   3. Sometimes you need to do it for both the Debug and Release configuration. I suspect a bug in VS 2015.
3. Go to Project .... Properties... VC++ Directories ... Library Directories and add a new folder. This will be the location of the DirectX SDK library files and is probably “C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\lib\x86;”.

Note the “\86” on the end. You can try “\64” if you want, but I have tested “\86” more carefully.

The program should now compile and give you a basic menu when it runs.

**Drawing a picture on the screen**

Download the file assets.zip from Blackboard. This contains a few game assets that you might want to use. Some of the ship images come from <http://millionthvector.blogspot.com/>. Acknowledge the source if you use them for anything.

For now, you want “ship.bmp”. Extract this to the same folder as your code.

You will be mainly programming in three functions: Game::StartOfGame(), Game::Update() and Game::EndOfGame(). You can find these in gamecode.cpp.

We need to create a PictureIndex to keep track of an image when we load it. A “PictureIndex” is really just a number. Add a new member variable to the Game class:

**PictureIndex image;**

Next, we need to load the ship.bmp image. You should do this inside Game::StartOfGame()

**MyDrawEngine\* pDE = MyDrawEngine::GetInstance();**

**image = pDE->LoadPicture(L"ship.bmp");**

StartOfGame runs once whenever a player starts the game. We use this to load assets into memory, to avoid doing it while the game is running.

Next, in Game::Update() we need to tell the engine to draw the picture, in a particular location.

Game::Update() runs every frame – probably 50-100 times every second.

We need a Vector2D to specify the location of the image.

We need to get a pointer to the Draw Engine.

And, we need to tell it to draw:

**Vector2D pos(300, 300);**

**MyDrawEngine\* pDE = MyDrawEngine::GetInstance();**

**pDE->DrawAt(pos, image);**

You could, actually, do this all in one line:

**MyDrawEngine::GetInstance()->pDE->DrawAt(Vector2D (300, 300), image);**

But it is better to separate code when you can. It makes debugging easier, for one thing.

Play around with different values for the position, so you get used to where things are on the screen. You can also experiment with adding a few numbers to the DrawAt call:

**pDE->DrawAt(pos, image, 2.0f, 0.5f, 0.8f);**

**Moving a picture around**

To move a picture around, we can change the “pos” vector. However, this won’t work:

**Vector2D pos(300, 300);**

**Vector2D move(2.0f, 2.0f);**

**pos = pos + move;**

Why doesn’t this work? Because every frame, you are setting “pos” back to (300, 300). You will need to make “pos” a member variable (in gamecode.h). You can initialise it in StartOfGame( ) by setting it to (300, 300). Then in Update(), you make it change.

Play around for a while – make the ship move in different directions and at different speeds.

Also try to get it to spin, or shrink, or fade from view.

**Getting user input from the keyboard**

To check the keyboard, you first need to “sample” it.

**MyInputs\* pInputs = MyInputs::GetInstance();**

**pInputs->SampleKeyboard();**

You can then ask if a particular key is being held down.

**if (pInputs->KeyPressed(DIK\_W))**

**{**

**}**

“DIK\_W” means “Direct Input Key W”.

You can use this to make the object move upwards:

**if (pInputs->KeyPressed(DIK\_W))**

**{**

**pos = pos + Vector2D(0, 4.0f);**

**}**

Get the spaceship to move in four different directions. You only need to sample the keyboard once each frame.

What happens if you press two keys?

Try to make the spaceship rotate left and right, using the arrow keys (DIK\_LEFT and DIK\_RIGHT).

Can you make the spaceship move in the direction it is pointing? This is a bit tricky. First, you need to create a vector in the right direction, using the “setBearing” method. Then you need to add it to the position.

**Vector2D velocity;**

**velocity.setBearing(angle, 4.0f);**

**pos = pos + velocity;**

**Playing sound effects**

Playing sound effects is a lot like drawing a picture. You load the sound at the start of the game, and play it during the game.

Move the sound effect “shoot.wav” into the same folder as your code.

First you need a SoundIndex. You can make this a member variable – somewhere near your PictureIndex.

**SoundIndex shootSound;**

Second, you can load a sound. You should do this inside Game::StartOfGame()

**MySoundEngine\* pSE = MySoundEngine::GetInstance();**

**shootSound = pSE->LoadWav(L"shoot.wav");**

Finally, you can try to play it. You should do this inside Game::Update()

**MySoundEngine\* pSE = MySoundEngine::GetInstance();**

**pSE->Play(shootSound);**

Doesn’t work. Why not? Because you are telling it to “Play” every frame. Each time you do that, it re-starts. So you never hear anything. Instead, get it to only play when you press the space bar. (DIK\_SPACE).

This still is not quite right, but you are final year students and you can probably use the tools at your disposal to get it working. (Hint: It plays when you let go of the space bar. Why? Look at other functions in MyInputs to see if any of them look useful.)

One final trick. You can make a sound effect play on a continuous loop. Make a new sound effect, and load the file “thrustloop2.wav”. To play on a loop, you can add:

**pSE->Play(shoodSound, true);**

This makes the sound effect loop over and over. See if you can get the sound to play when you move the spaceship and stop when you let go of the key.

**Wrap it up**

OK – you have an object that moves around and makes sound effects. It is tempting to dive in and start making a game. You can if you want. However, it will be easier to start splitting it up into pieces rather than doing everything in Update( ). That’s pretty much what we will look at for the rest of the module.

In fact, we wanted to call the module “KF6017 How to Split Things Up Rather Than Putting It All In Update()” but somebody objected.

For now, play around. As final year students you probably don’t need to be given exact tasks. Get used to using the Shell. Look at functions that I have not told you about. Try using the mouse. Find bugs and report them to me. Criticise my code standards. Read about “Hungarian Notation”.