Example Game

Today we will look at an example game built with the programming framework we use over the first 12 weeks of Fundamentals of Games and Graphical Systems Development module.

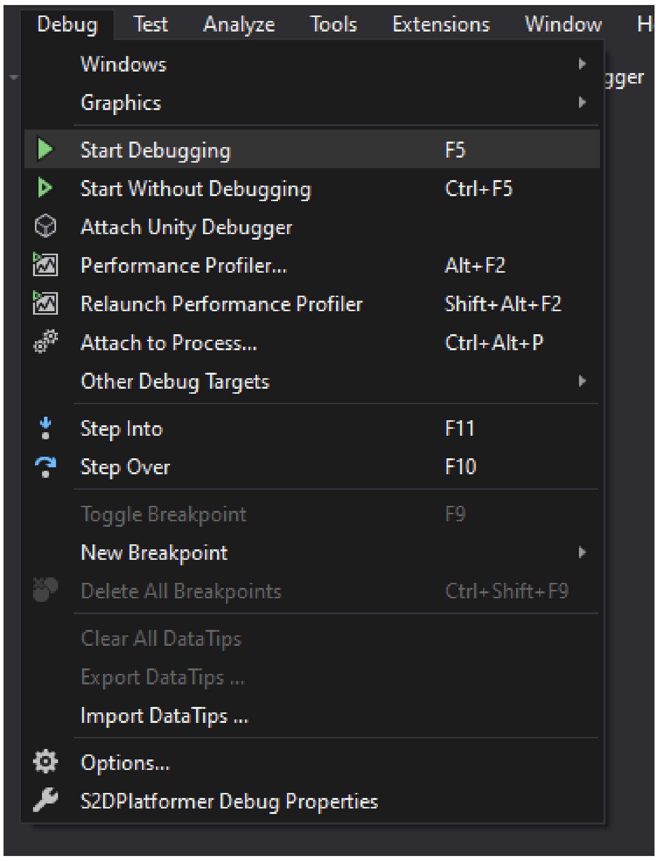
The game we will be looking at today has been built using Visual Studio 2017 and uses a variety of technologies which have been wrapped up into a helpful programming framework called S2D written in the industry standard language, C++.

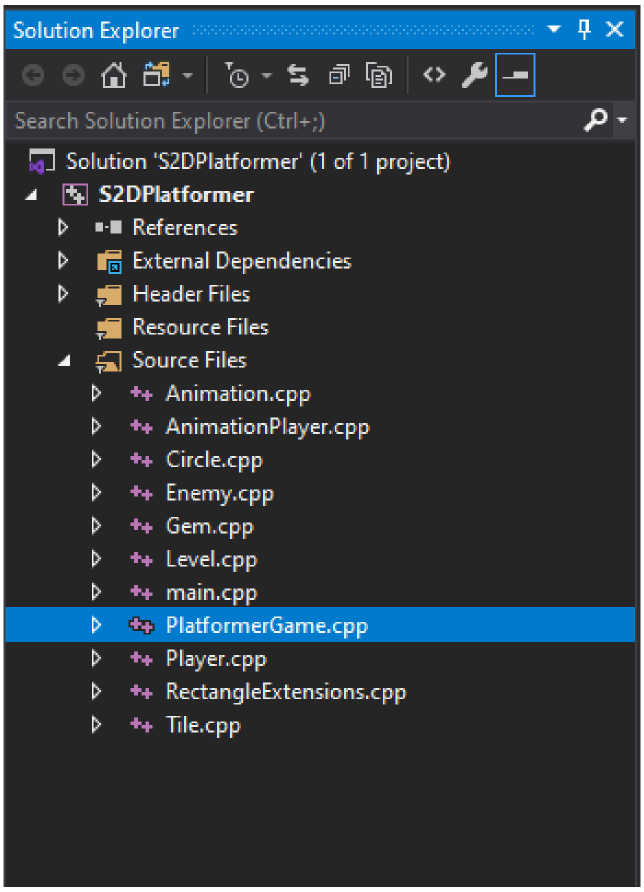
The underlying technologies at work in this framework are OpenGL, OpenAL and freeglut.
You find out more about these throughout the modules.
Some of you may find this game demo rather familiar, it's a direct port of a Microsoft XNA demo into S2D.

What you will need:
Microsoft Visual Studio 2017
Platformer Demo

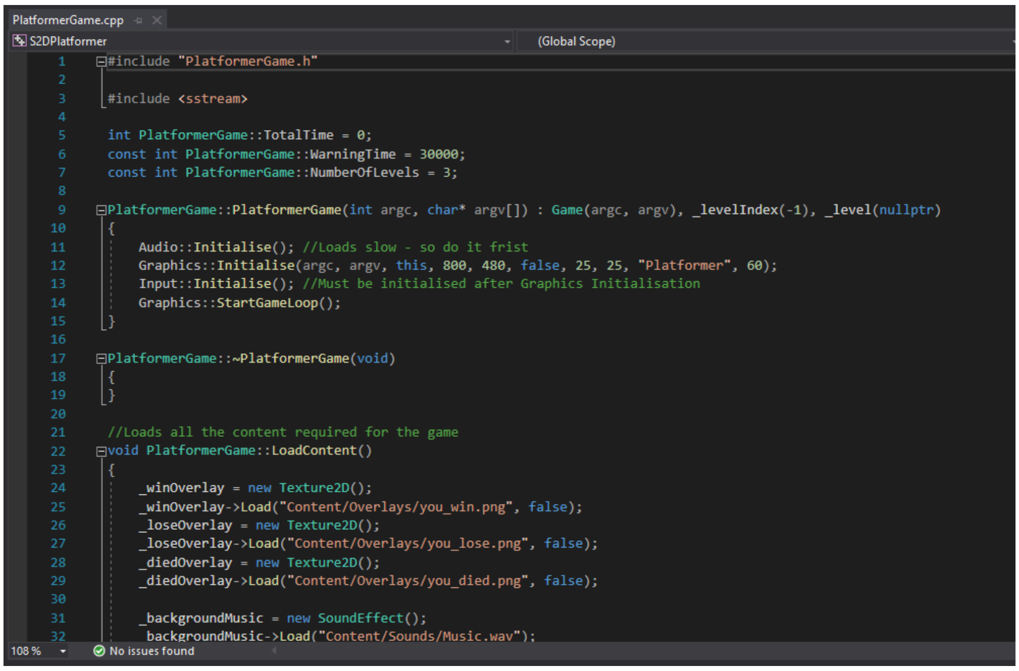
Getting Started

Visual Studio is a very complex piece of software with many features. You will learn about many of these throughout modules, for now the key areas we need to focus on are the Solution Explorer, Debug Menus and The Code Window.





Green **Start Debugging** Button Runs the Application
Solution Explorer List all the files in the Solution.
Double click a file to open in **Code Window**.



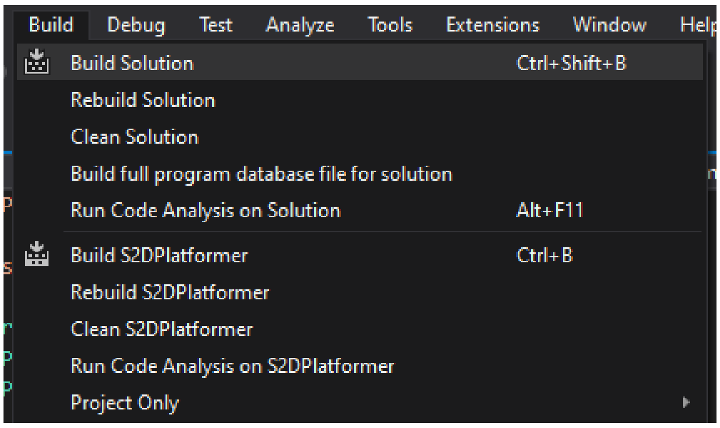
```cpp
#include "PlatformerGame.h"

#include <sstream>

int PlatformerGame::TotalTime = 0;
const int PlatformerGame::WarningTime = 30000;
const int PlatformerGame::NumberOfLevels = 3;

PlatformerGame::PlatformerGame(int argc, char* argv[]) : Game(argc, argv), _levelIndex(-1), _level(nullptr)
{
    Audio::Initialise(); //Loads slow - so do it frist
    Graphics::Initialise(argc, argv, this, 800, 480, false, 25, 25, "Platformer", 60);
    Input::Initialise(); //Must be initialised after Graphics Initialisation
    Graphics::StartGameLoop();
}

PlatformerGame::~PlatformerGame(void)
{
}

//Loads all the content required for the game
void PlatformerGame::LoadContent()
{
    _winOverlay = new Texture2D();
    _winOverlay->Load("Content/Overlays/you_win.png", false);
    _loseOverlay = new Texture2D();
    _loseOverlay->Load("Content/Overlays/you_lose.png", false);
    _diedOverlay = new Texture2D();
    _diedOverlay->Load("Content/Overlays/you_died.png", false);

    _backgroundMusic = new SoundEffect();
    _backgroundMusic->Load("Content/Sounds/Music.wav");
```

Try 'PlatformerGame.cpp'.
The solution explorer contains all the files that comprise the project. Familiarise yourself with the files in the project. Try to see if you can work out what each file does.

All the files that have the .h extension are called header files.
These describe the functionality that will be containing in the matching .cpp file. The .cpp files (C++ source files) implement what the header files say will be included.
For example, the Circle.h file lists Center() and Radius() methods among others.
If you look in Cicle.cpp, you should see code within matching Center() and Radius() methods. You will learn a lot more about header and source files later within modules. Be careful not to delete any files from the solution as they are all necessary to compile and run the game!

Building and Debugging the Game

To compile the source code files into an executable binary file, you need to click 'Build Solution' from the 'Build' menu. This will take a few moment and you should see each file being compiled in the Output Window at the bottom of Visual Studio.
Once the build is complete, choose 'Start Debugging' from the 'Debug' menu. You can also press the Green Arrow in the toolbar, or just press F5. In fact, just pressing 'Local Windows Debugger' will start a Build first if one is required.

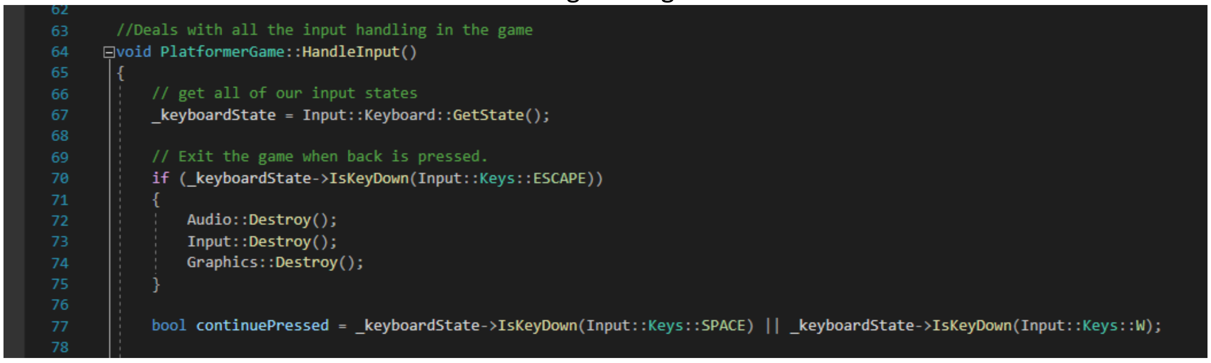


Build
Debug
Test
Analyze
Tools
Extensions
Window
Help
Build Solution Ctrl+Shift+B
Rebuild Solution
Clean Solution
Build full program database file for solution
Run Code Analysis on Solution Alt+F11
Build S2DPlatformer Ctrl+B
Rebuild S2DPlatformer
Clean S2DPlatformer
Run Code Analysis on S2DPlatformer
Project Only

However, it's handy to know what is in the Build menu in case something goes wrong.

Code Window

This is the area you write your code. You're currently looking at 'PlatformerGame.cpp' an important file of the Platformer Demo which controls the game logic.



```
//Deals with all the input handling in the game
void PlatformerGame::HandleInput()
{
    // get all of our input states
    _keyboardState = Input::Keyboard::GetState();

    // Exit the game when back is pressed.
    if (_keyboardState->IsKeyDown(Input::Keys::ESCAPE))
    {
        Audio::Destroy();
        Input::Destroy();
        Graphics::Destroy();
    }

    bool continuePressed = _keyboardState->IsKeyDown(Input::Keys::SPACE) || _keyboardState->IsKeyDown(Input::Keys::W);
```

Playing the Game

You should then be presented with a working platform game!

Controls To play

The controls are as follows: W, A, S, and D move the player Up, Left, Down and Right.
You can also jump with the Space Bar.
Collect all the crystals and move the player to the Exit sign.
Avoid any wandering monsters!

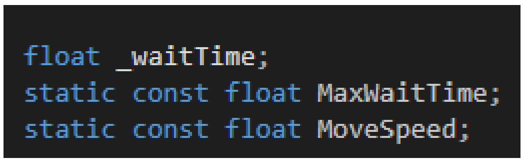
When you're finished, you can close the game by clicking the 'X' in the top right corner or by going back to Visual Studio and pressing the Red Square 'Stop Debugging' button.

Making Modifications

There are many aspects of this game which can be modified, in fact everything can be modified since you have access to the source code.
First, make sure you are not running the application by closing any running instances or by pressing the Red Square 'Stop Debugging' button in Visual Studio 2017.

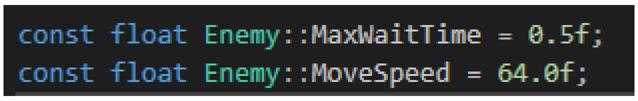
Increasing the difficulty

Open the 'Enemy.h' file. You should notice towards the bottom, some Constant values have been defined called 'MaxWaitTime' and 'MoveSpeed'.



```
float _waitTime;
static const float MaxWaitTime;
static const float MoveSpeed;
```

These variables control how the Enemies in the game behave. Although they are defined in the header file, the actual values are set in the source file, so now navigate the 'Enemy.cpp'.
At the top, you should see they have been defined with values 0.5f and 64.0f.
The **f** states that these are floating point variables (numbers with a decimal point). Change these values and see what effect this has on the game (Specifically Level 2 and 3).

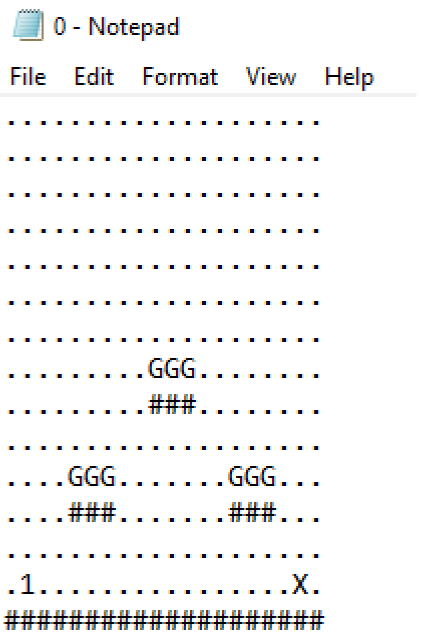


```
const float Enemy::MaxWaitTime = 0.5f;
const float Enemy::MoveSpeed = 64.0f;
```

Adding a Level

The platformer demo provides code that can automatically load new levels from the text files. The levels for this game are stored as text files within the ‘Content’ folder under a directory named ‘levels’, using File Explorer to navigate to this folder.

The text files are named in numerical sequential order starting from zero E.g., 0.txt, 1.txt, 2.txt. You shall create the fantastic new level 4!
Look at level 0 by double clicking the text file. Examine the contents of this file that comprise the first level. It looks like the following:



```
....................
....................
....................
....................
....................
....................
....................
.........GGG........
.........###........
....................
....GGG.......GGG...
....###.......###...
....................
.1................X.
####################
```

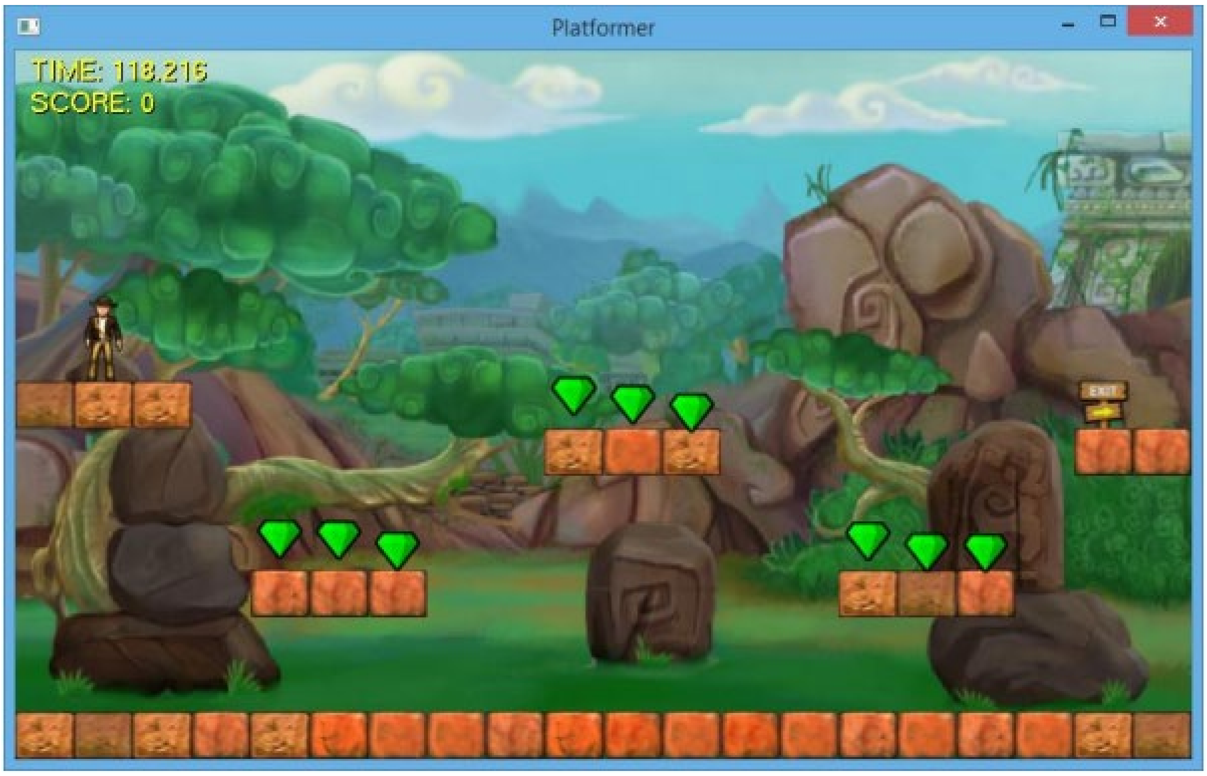
Each level is a 21-character by 21-character grid. Code is already in place to load each tile of the game world from the text file. Each time the player reaches an exit, the next level is loaded from the text file and the game continues.
The supported characters that represent game tiles are:

Period (.) – represents blank space
X – Exit
G – collectable gem
Hyphen (-) – floating platform
A, B, C, D – represent different types of monsters
Tilde (~) – platform block
Colon (:) – Variety platform tile
1 – Player start position
Hash (#) – impassable block (floor)

Enhance a level

It is much easier to work on a copy of an existing file. Right click ‘0.txt’ and select ‘Copy’ from the context menu. Right click the Level folder and select ‘Paste’ from the context menu. This should create a file named ‘0 – Copy.txt’, rename this file to ‘3.txt’. The text files are not compiled like code, so they are not directly included in our Solution. That’s why we are using File Explorer for this. We could include them in our solution if we wished to, however.

Edit '3.txt' so it resembles the screenshot below. You will also have to tell the game that there is now a new level.



Open 'PlatformerGame.cpp' in Visual Studio from the Solution Explorer and locate the constant variable called NumberOfLevels near the top of the file.
Change this value to 4 and you should now have a new 4th level.
You may notice your level doesn't look quite right...can you work out what is wrong?

Challenge

Make a brand-new 5th level using 'never before seen' monsters and passable platforms (Remember there's 4 types of monsters and platform)!
Run the game and test your level.

Summary

Congratulations! You have completed an introduction to game programming using S2D, built on OpenGL and OpenAL. Having completed the exercises, you will have experienced first-hand games programming using C++. This tutorial has only begun to touch on the potential of this framework. Feel free to experiment further with this Demo.

You will get the opportunity to build new games with this framework within the module.
To experience more of the programming aspect of games programming, you may wish to familiarise yourself with C++ in general.

Have fun, happy coding!