GAM340 – Programming	Podcast 1
SDL Programming	

Introduction
This podcast will look at using SDL to create the beginnings of your own game engine, rather than using Unreal or Unity to create games.

Working with engines vs. making engines
Within the Academy, we're all very familiar with using tools like Unity and Unreal to create games. As creative developers, these engines give us a great deal of functionality allowing to fairly easily take our creative ideas and turn them into proofs of concept, prototypes, demos and full products.

However, there is often the feeling as developers that we are just filling content into slots in these engines and there's a great deal going on under the hood that we are not privy to. As programmers this can be particularly frustrating as there tends to be an inbuilt desire in programmers to understand how things work, to make your own implementations of 'things' and to have some kind of agency and control over the systems you create. Building our own engines from scratch allows us to scratch those itches.

What is SDL and how does it work
SDL is an open-source cross-platform development library, rather than engine, that enables developers to create games and other interactive applications. The name is short for Simple DirectMedia Layer and provides a wrapper for Windows DirectMedia functionality (audio, keyboard, mouse, controller, graphics and so on) that is both (fairly) simple to use and standardised across platforms (Windows, OSX, Linux, Mobile etc). SDL is provided as both libraries (making it easy to import functionality into your projects) and source code (allowing you to port and extend functionality across to new platforms).
As Windows programmers, SDL is used as a library that is part of a C++ project.

Using SDL in Windows
To make it easier to use SDL, I have set up an initial SDL project at https://github.com/Falmouth-Games-Academy/GAM340-SDL.git. This is designed to work with Visual Studio. If you want to build an application from scratch go to https://www.libsdl.org/download-2.0.php and download the development libraries for Windows (Visual C++ 32/64-bit). In addition, go to https://www.libsdl.org/projects/SDL_image/ and again download the development libraries for Windows (Visual C++ 32/64-bit). The SDL_Image project contains functionality for loading different image file formats, in particular PNG as it supports an alpha channel allowing for transparency in images.

To experiment with the initial projects, clone the git repo and load each one into visual studio. They should be built using the Debug X86 configuration and will generate build errors if other configurations are used. For more information:

https://docs.microsoft.com/en-us/visualstudio/debugger/how-to-set-debug-and-release-configurations?view=vs-2019

You may run into an issue in Visual Studio with the 'Windows SDK Version', this can be addressed through setting the SDK version in project's properties and the general configuration properties:
https://docs.microsoft.com/en-us/visualstudio/ide/managing-project-and-solution-properties?view=vs-2019

These three projects are designed as a starting point for SDL. To create your own SDL projects, copy the 01.sdl_SimpleShell folder in explorer, rename it and get creative.

Project 1 – A simple SDL shell (01.sdl_SimpleShell)
This project is a bare bones SDL shell that will create an SDL window and run it until it ends. The demo deals with startup, the application loop and shutdown. In addition, the application will 'sleep' and run at 60 frames per second, rather than as fast as possible.

This application does very little in terms of output, but the code contained in it is the core functionality for making SDL applications.

Project 2 – An SDL application that loads an image and displays it (02.sdl_LoadImage)
This project loads a PNG file (with transparency) and displays it as a texture. The code uses the IMG library to load a bitmap as a SDL_Surface and then converts the surface to an SDL_Texture for drawing with SDL_RenderCopy, which allows the texture to be drawn anywhere on the screen and at any size with the dest SDL_Rect. In addition, another SDL_Rect can be used to draw a region within the texture using pixels to determine the (x,y) offset within the texture and the width and height of the region to be drawn.

Project 3 – An SDL application that moves an image using the arrow keys (03.sdl_MoveImage)
This project builds on project 2 and includes user controls. This is done through SDL's event system and input is detected using SDL_KEYDOWN and SDL_KEYUP events. Unlike some input systems, SDL only creates events on state change, so a key is down until it generates an SDL_KEYUP event. The code to move the ball around the screen is implemented to use this approach.

Develop your own applications
These three applications, and the third one in particular, create an ideal base to start developing demos and games. However, one of the challenges of working in limited engines is that there is not a great deal of functionality to support your development activities, so you often end up spending time writing 'core functionality' that you would have in Unity and Unreal for 'free'. However, the benefit of this approach is that you can gain a far deeper understanding of 'how things work'.

Also, it's worth developing your google-fu skills to search out SDL related topics and help on the internet. Typing any SDL_ function should result in the manual page for the function and typing 'sdl <whatever>' should take you to some useful sites, though some sites may be more useful than others.

Here are some ideas of programs to develop:

1. **Bouncing Balls**
The SDL demos in the repo allow a user to move a ball. What happens if we remove the user from the equation and create lots of balls that can bounce around the screen using floating point maths to create smooth trajectories and speeds. One challenge with this demo is to make sure that the balls are drawn so that they are not hanging off the screen. Typically, sprites are drawn from the top left-hand corner which means that collisions with the right or bottom edges of the screen will result in balls going largely off the screen. However, this can be addressed by considering the dimensions of the ball.

2. **Working with texture pages**
In order to create homages to existing games, there are lot of game texture sets on the internet. Typically, these consist of images that are combined into large textures. To draw the individual textures meta data is required that describes the location and size of a texture.
Within the demo applications, there is an 8x8 bitmap font texture page that contains all the printable characters order by ASCII code. You can use this as the basis of your own font renderer.

3. **Pong Game**
Pong was the first successful video game and 40-odd years on it should present a fairly straightforward programming challenge. SDL has functionality for dealing with rectangle intersections, but you may wish to develop your own.

4. **Breakout Game**
Breakout takes the sensibilities of pong and turns them on their side to create a highly addictive and destructive game.

Good luck and have fun.