

FIT2096: Games Programming 1

Week 1: Introduction to Games Programming in C++



Staff

- Chris Nelson
 - Research in Game Design
 - Games for Change
 - PhD with Melbourne Uni

BIT Game Programming

- FIT2096 Game Programming 1
 - Unreal Engine 4 using Blueprints and C++
 - Behind the scenes
 - Basics and fundamentals
- FIT2097 Game Programming 2
 - Advanced Unreal Engine 4 with Blueprints and C++

- What courses are you doing?
- Who is doing the Games Major?

- What games have you been playing over summer?

Today

- Unit introduction
- Some Game Fundamentals
- Introduction to Unreal Engine 4
- Source and Asset Management

Unit introduction

Staff

- Chris Nelson – Lecturer/Chief Examiner
 - chris.nelson@monash.edu
- Jason Haasz – Tutor
 - jason.haasz@monash.edu
- Josh Olsen – Tutor
 - josh.olsen@monash.edu
- Stephen Mcnamara– Tutor
 - stephen.mcnamara@monash.edu

Schedule

Week 1	Introduction to Unreal Engine 4	
Week 2	Coordinate Spaces and Vectors	
Week 3	Matrices and Quaternions	
Week 4	The Game Loop. Lerp. Timelines, Spawning C++	
Week 5	Vertices, Triangles and Polygons, Textures	Assignment 1 - Extended Lab Exercises (15%)
Week 6	Cameras and Rendering Pipelines, Cameras in UE4, Culling, Introduction to Blueprints	
Week 7	Physics Engines & Collisions	
Week 8	UE4 Gameplay Framework & Classes	Assignment 2 (15%)
Week 9	UI - HUD, Menus, Model-View-controller, UX	
Week 10	Game Audio, Particles & Lighting	
Week 11	Introduction to FIT2097 topics	
Week 12	Review	Assignment 3 (20%)

Assessment

- The assessment in this unit is 50/50
 - 50% of your total mark comes from the in-semester assignments.
 - 50% of your total mark comes from the exam.
- In accordance with the faculty hurdle policy you must achieve the following to pass this unit:
 - A mark of at least 45% for the in-semester assessment.
 - A mark of at least 45% for the exam.
 - An overall mark of at least 50% for the unit as a whole.
- Any student who achieves 50% or greater for the unit as a whole but misses either of the 45% hurdles will receive a maximum mark of 49N.

Assignments

- Assignment 1 (15%):
 - Will combine the knowledge of weeks 1 to 4 (due week 5)
- Assignment 2 (15%):
 - Will combine the knowledge of weeks 1 to 7 (due week 8)
- Assignment 3 (20%):
 - Will combine the knowledge of weeks 1 to 11 (due week 12)

Late assignments

- Late assignments will be penalised 5% of the total available mark per day late (including weekends).
- This means that if an assignment is assessed out of 100 marks, then each day late will result in a 5 mark penalty (down to a minimum of 0).
- If you are seeking an extension please try your hardest to inform us before the due date.
- All extension requests require a special consideration form to be submitted. This form requires some documentation as to why the extension is required.
- <http://www.monash.edu.au/connect/assets/docs/forms/in-semester.pdf>

Examination

- The exam is worth 50% of your final mark.
- It is a 2 hour, closed book exam.
- Content from weeks 1 to 12 will be covered.

Lecture	Wed	10:00	CL_16Rnf/S9
Laboratory	Wed	12:00	CL_14Rnf/G11B
Laboratory	Thu	14:00	CL_14Rnf/147
Laboratory	Thu	16:00	CL_14Rnf/147
Laboratory	Fri	10:00	CL_14Rnf/147
Laboratory	Fri	12:00	CL_14Rnf/147

- If you are off campus:
- Monday at 12 - 2 AEDST (9 am in China?)
- Jason Haasz on Zoom
- jason.haasz@monash.edu

Lab classes & Assignment 1

- They start this week with a C++ open book test, then getting Unreal Engine 4 and Visual Studio to run together.
- Labs will cover the material covered in the lectures from the week before.

Independent Learners

- Technology in Game development is constantly changing and evolving
- To keep up with developments in this area you will need to do your own research and investigation

Independent Learners

- Unreal Engine 4 is massive and changes with an update every 2 months or so
- Even the developers of the engine don't know it all (and neither do we!)
- We will be giving you a good grounding in Unreal Engine 4
- To fill out your knowledge and do well in the assignments you will need to utilise the online documentation and forums

A quick note on coding style

- As this is a programming unit it is expected that your code is easy to read and neatly formatted.
- There is no prescribed style guide but...
- Your code should look something like this:

A quick note on coding style

```
#include <iostream>

using namespace std;

int findMin(int array[], int first, int size)
{
    int minIndex = first;
    for(int i = first; i <= size; i++)
    {
        if(array[i] < array[minIndex])
        {
            minIndex = i;
        }
    }
    return minIndex;
}
```

```
int main()
{
    const int ARRAY_SIZE = 4;
    int numbers[ARRAY_SIZE] = {7, 3, 2, 8};
    int minIndex = -1;
    for (int i = 0; i < ARRAY_SIZE; i++)
    {
        minIndex = findMin(numbers, i, ARRAY_SIZE-1);
        int temp = numbers[i];
        numbers[i] = numbers[minIndex];
        numbers[minIndex] = temp;
    }
    for(int i = 0; i < ARRAY_SIZE; i++)
    {
        cout << numbers[i] << endl;
    }
    return 0;
}
```

A quick note on coding style

- And not like this (brace yourselves)...

Yuk! My eyes burn...

```
#include <iostream>
using namespace std;

int findMin(int array[],
            int first, int size)
{
    int minIndex = first;
    for(int i = first; i <= size; i++)
    {
        if(array[i] < array[minIndex])
        {
            minIndex = i;
        }
    }
    return minIndex; }


```

```
int main()
{
    const int ARRAY_SIZE =4;
    int numbers[ARRAY_SIZE] = {7, 3 , 2, 8};
    int minIndex= -1;
    for(int i = 0;i < ARRAY_SIZE; i++)
    {
        minIndex = findMin(numbers, i, ARRAY_SIZE-1);
        int temp = numbers[i];
        numbers[i] = numbers[minIndex];
        numbers[minIndex] = temp;
    }
    for(int i = 0; i < ARRAY_SIZE; i++) {
        cout << numbers[i] << endl;
    }return 0;
}
```

A quick note on coding style

- Indent every block of code
 - A block of code is the code contained within a pair of { }
 - Brace styles (same line or next line) are up to you, but be consistent.
- Give your variables meaningful names
 - int myInt or int refCounter?
- Should you use Hungarian notation?
 - Microsoft does for its C code...
 - But for us it's not a huge deal.
 - I think it helps with global and member variables, you can use m_ and g_
- Comment your code to explain logic, not syntax. Think why and how, not what.
 - What not to do:
 - `int score = 0; // Creates an integer called "score" and sets its value to 0`

Some Game Fundamentals

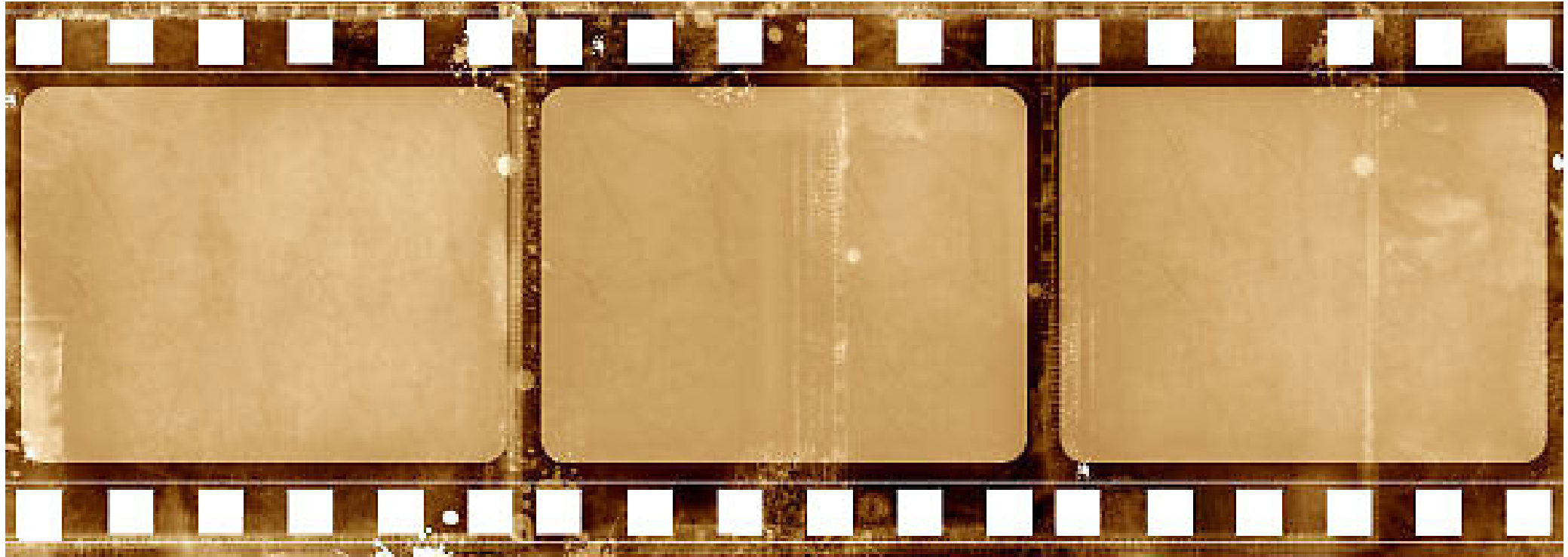
Animation

- Let's look at some animation...
- <https://www.youtube.com/watch?v=7M5So1WFNzo>

Animation

- How were these early animations achieved?
- What is the difference between an animation and what you see in a digital game?

Frames – what is this?



Frames

- Just so we all start the unit on the same page, let's think about how games work at their most basic level.
- Just like in film, games are a **sequence of frames** displayed continuously which gives the illusion of movement.
- In film, the frames are already prepared, hence your computer never sounds like a jet engine when watching a movie.

Frames

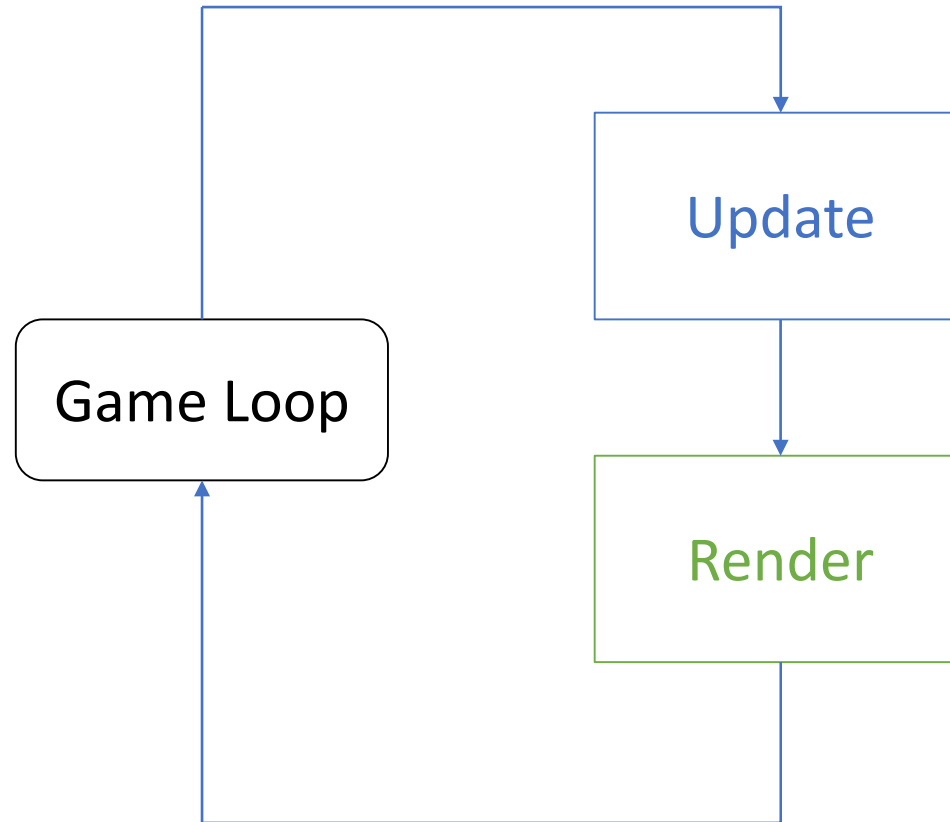
- In games, we're **creating each frame in real-time** just before it's displayed on the monitor.
- The process of building one of these frames is broadly referred to as rendering. Games use a subset of this called real-time rendering.
- **Remember the monitor has no idea whether it's displaying a movie or a game – both are just a sequence of still images.**
 - That seems blatantly obvious, but it's a refreshing thought that we sometimes lose sight of.

Frames

- Every frame, the entire world is rendered from scratch.
 - There's nothing fancy happening here where only objects that move are re-rendered – everything is rendered from scratch each frame of the game.
 - Eventually we can optimise the game so we only render objects which are visible, but that's a thought for later in the year.
- Let's dissect a single frame of a game and see how it's constructed.

Real-time Game Loop

Frames Per Second
(FPS) =
Number of iterations
per second

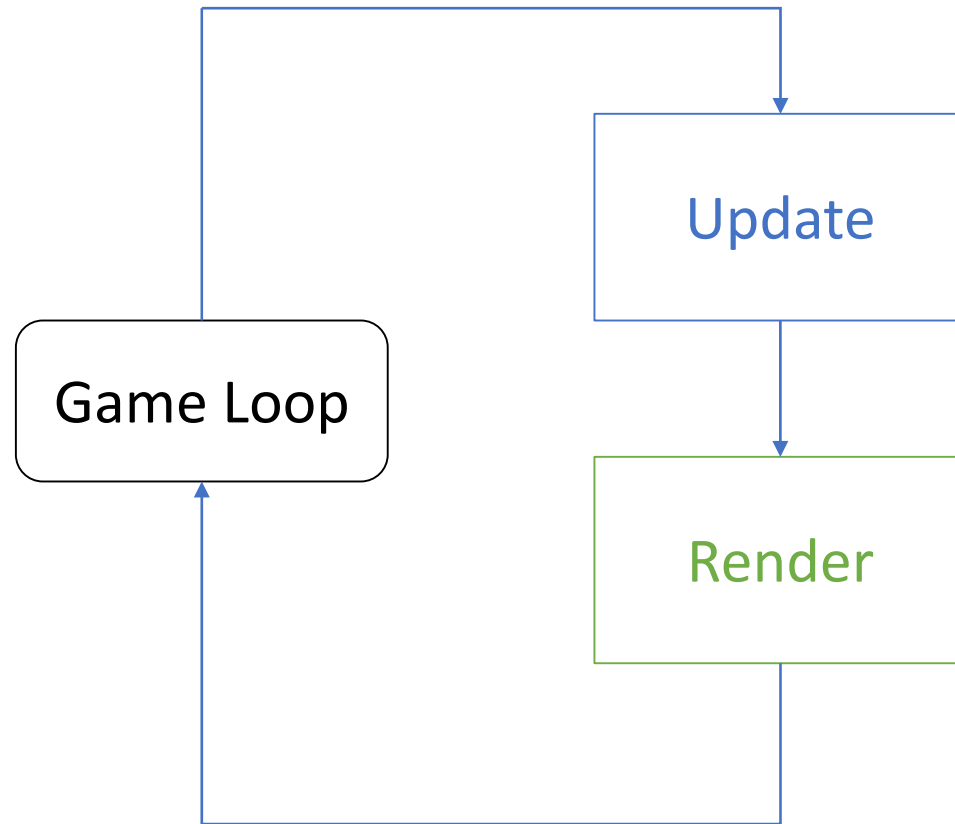


- Takes the state of the simulation
- And advances it
- Shows the simulation on the screen

The game loop

- We know we need to render frames over and over again, so that suggests that **at heart games are just a loop**. We call this **The Game Loop**.
- Every tick of the loop results in one completed frame of the game.
- The entire logic of the game takes place somewhere within this loop.
- The more work we do inside this loop, the slower our game runs.
 - Developers strive for 60 iterations of the Game Loop every second, although they don't always hit that target.
- Frames per second (FPS) is a common measure of game performance. We can now think of this more precisely as iterations of our game loop per second.

Real-time Game Loop



- Transforming Game Objects
 - Applying physics
 - Running AI routines
 - Processing network data
 - Etc.
-
- Determines the game objects that can be seen by the camera
 - And renders them to the screen using the Render Pipeline

Updating and Rendering

- **Update**

- The update phase takes the state of the simulation, and advances it.
- This could involve moving the player, running AI (decisions and movement), resolving physics, sending and receiving network data, etc.

- **Render**

- The render phase takes the state of the simulation, and shows it on the screen.
- After the Update phase, every object in our game knows where it should be and what it should be doing.
- Here we just render the world to visually reflect these changes.

Updating and Rendering

- Psuedo code example:

```
while( user doesn't exit )  
{  
    // Update  
    check for user input  
    run AI  
    move enemies  
    resolve collisions  
  
    // Render  
    draw graphics  
}
```

Updating and Rendering

- Even with only two phases, a structure like this has advantages:
 - If you were to stop calling the Update phase, your game would essentially pause. Handy.
 - All your graphics calls are compartmentalised into their own phase, decoupling them from the state of the simulation.

Recap

- Animations, films, and games all render frames to the display
- The difference is that a game has to
 - Update the scene
 - And then render the scene each time a frame is displayed

- You will be learning to use a real-time game loop throughout this unit
- We will go into this in more detail in future lectures

Introduction to Unreal Engine 4

Unreal Engine 4

- Unreal Editor.

- Quick tour:

https://www.youtube.com/watch?v=WC6Xx_jLXmg&index=5&list=PLZlv_N0_O1gYfw89JtRX0bTb_YbxHOXwz#t=188.218244

Using the editor

- Creating a new project from a template

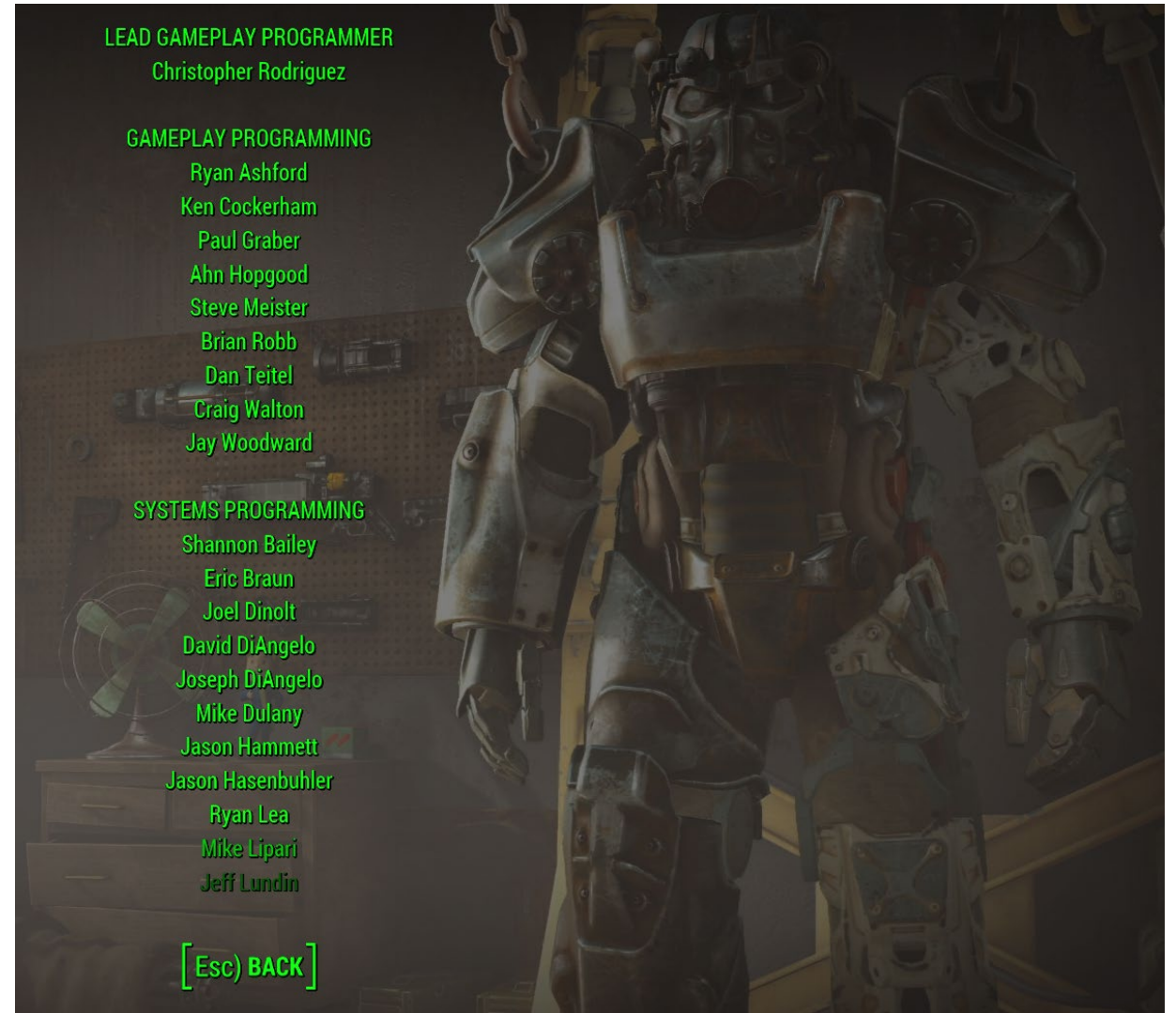
Unreal Engine Versions

- Engine version 4.24.1 is installed on lab PCs
- If you use other versions, we might not be able to open your assignments to mark them.

Source and Asset Management

Games as a Software Project

- Look at how many developers there are in the credits of a game



Games as a Software Project

- All of their work needs to be merged into the final game
- AAA games are a medium sized software engineering project

Tools

- There are tools to manage assets and help with collaboration
- Some you might see mentioned
 - Perforce for source code (Expensive)
 - Alien Brain for art and sound (Very expensive)
 - Git (Open Source)
- We'll use Git in this unit

Git

- Designed for source code, but can manage art assets
- Allows you to see the history of your project, and to roll back to an earlier, working version if you break things badly
- Designed for collaboration on a project. (But you'll be doing your assignments yourself.)

Git Concepts

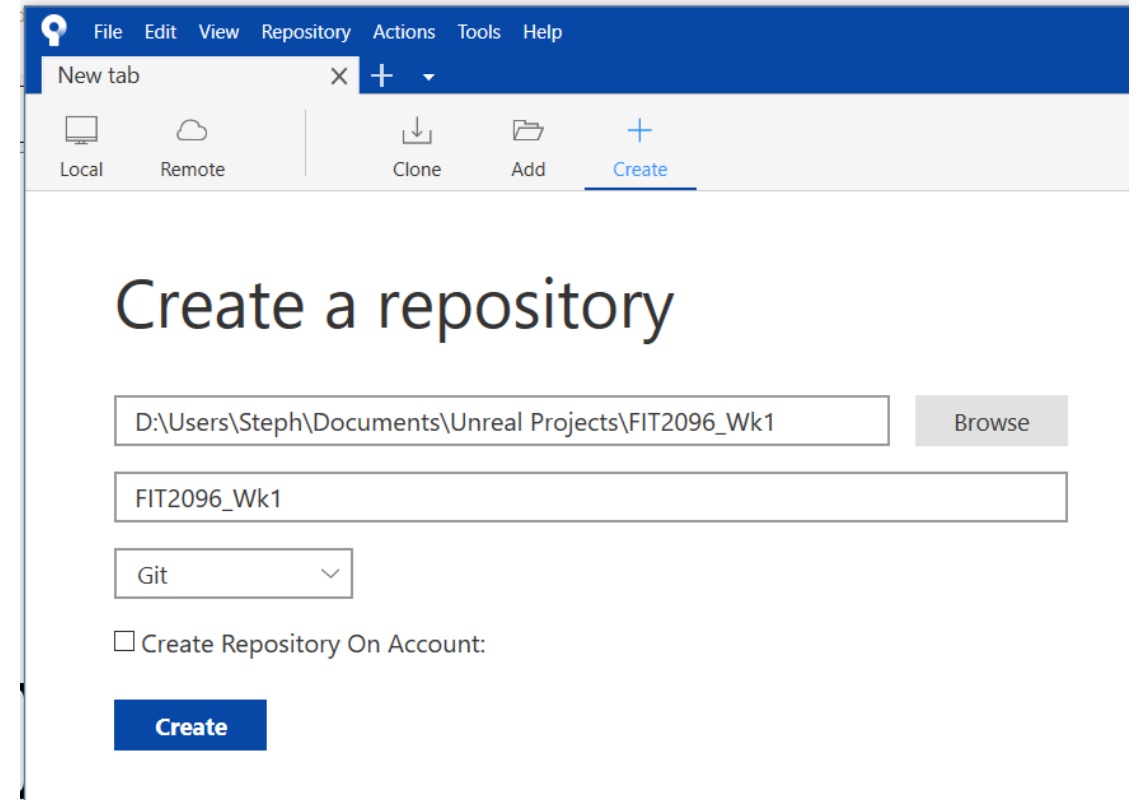
- Git keeps a repository of your source code and it's history
- You commit your changes to a repo, and add a comment explaining what you did, and why
- Git is a distributed source code control system. You can have multiple repos for a project
 - For example, hosting a repo on Gitlab.com would allow you to share a portfolio project with prospective employers

Using Git

- There are many GUI and command line tools for using a git repository. If you already know one, feel free to use it.
- **Atlassian SourceTree** is a straightforward GUI app, with free registration required
- <https://www.sourcetreeapp.com/>

Using SourceTree to create a local repository

- Select **Local**, **Create**, and choose a directory for the repository
- A **.git** directory will be created under the directory you choose



.gitignore

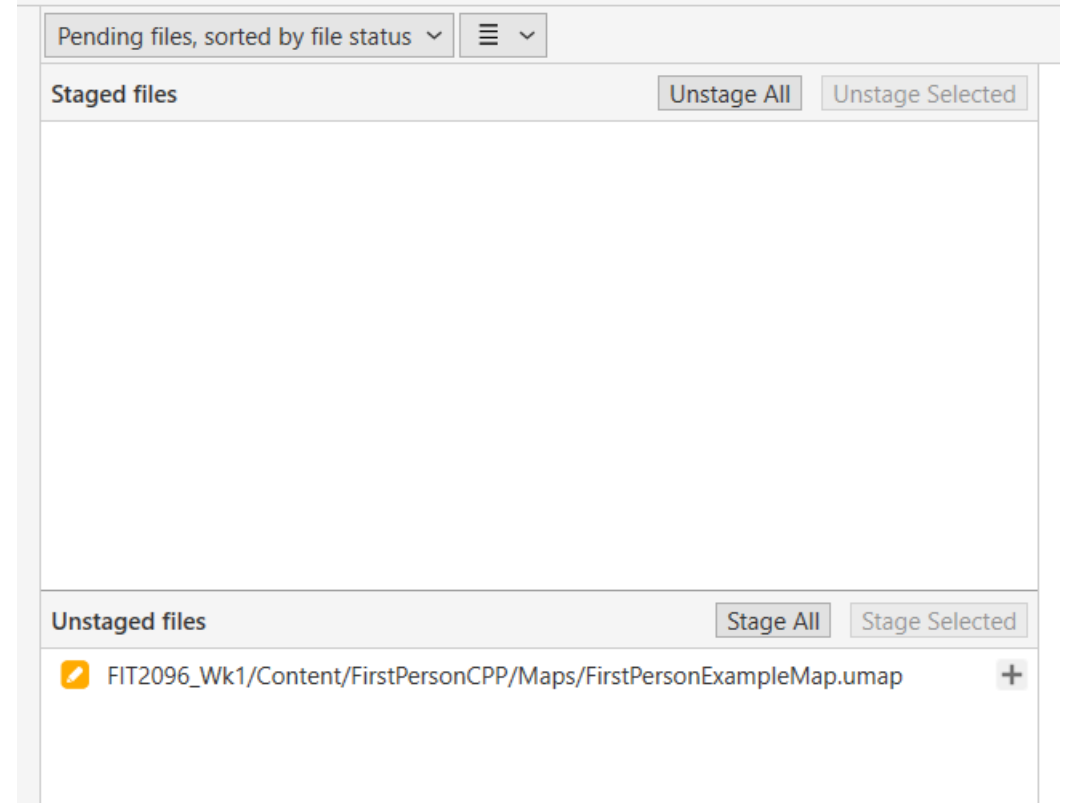
- You don't want to add everything to your repo, just the files needed to build your project.
- You can tell Git not to include files with a .gitignore file in your project root directory
- <https://github.com/github/gitignore/blob/master/UnrealEngine.gitignore>

Adding your project

- With the .gitignore in place, SourceTree will show you a list of unstaged files. Click **Stage All** to confirm you want to add these to the repository.
- Write a commit comment that explains why you are adding or changing the files, then click **Commit** to add the staged files.

Making changes

- When you edit your project, SourceTree will find the files that have changed.
- **Stage** and **Commit** these files with a commit comment.



Further resources

- <https://confluence.atlassian.com/get-started-with-sourcetree>

Next Week

- Maths! To find your way around a 3D world, you need 3D maths, so we'll start by looking at Vectors.