Spike 15: Unreal Engine Familiarisation
CORE SPIKE

Context: The Unreal Engine is a powerful and free tool that all game developers will benefit from knowing. Through the visual scripting of blueprints, many tasks will be able to be implemented in a shorter time period.

Knowledge/Skill Gap: A developer should know their options when it comes to programming a game. No engine is right for every situation. With game development being such a costly and lengthy process, it's important to use the right tools from the start.

Goals/Deliverables:
[CODE] + [SPIKE REPORT]
Write a report on the Unreal Engine. This report should include:
- A list of 3 games created in the Unreal engine and whether you think this was the correct choice for this game. Give pros, cons and any other reasoning behind your decision.
- An example of a blueprint for an actor that prints a string out to the console (top left hand side of the screen) on every tick.
- An example of the two commenting styles possible in UE4 blueprints.

Extension:
- Compare the Unreal engine to another game engine in a comparison table. Discuss the strengths or weaknesses of each engine when compared to each other and which engine should be used under different circumstances.

Recommendations:
- When searching for Unreal games to write about, include the word "popular" to get a more noteworthy list rather than every Unreal game ever.

Spike 16: Create Simple Scene

CORE SPIKE

Context: Using the Unreal Engine allows a developer to quickly create a scene that looks professionally rendered. However, this does not mean a development team can rely on this fact alone when it comes to making your game visually interesting. The quality of your assets, the placement of lighting and the stylistic cohesion of your models.

Knowledge/Skill Gap: A student needs to know how to quickly set up a scene in Unreal using their own models.

Goals/Deliverables:
[CODE] + [SPIKE REPORT]
Create a scene in Unreal Engine 4. This scene should contain 2 actors with imported 3D models and one dynamic light and one baked light. The scene should be built in order to resolve the lighting information for the scene.

Extension:
Make your scene do something. For example animate a rocket, make something explode, make a slowly rotating pelican. As long as you use the blueprinting system, make whatever you feel is apt.

Recommendations:

- If you don't have any 3D models on hand, search for sites that supply them for free such as http://www.turbosquid.com/
- Make sure to make the models their own actors.

Spike 17: Create Simple Blueprint in C++

CORE SPIKE

Context: While the blueprinting system may be powerful, it does have its limitations. While there is a large selection of nodes to support most features a developer may wish to implement, more specific functionality is not supported by default. To deal with these limitations, Unreal Engine supports the C++ language. A developer can either keep this functionality in C++ or expose the functionality to the blueprint system. Once code has been exposed to blueprints, it can be created as a node from any blueprint map.
Exposing C++ to blueprints allows a programmer to make their own APIs (Application Programming Interface). An API is a tool that software developers can use to help them with development. For example, a vector math api or a

Knowledge/Skill Gap: Students will need to know how to add functionality to the UE4 blueprint system through exposing C++ code to it. This will allow students to not be restricted to the base functionality of the Unreal engine and be able to create anything they wish to while still being able to utilise the Blueprints system.

Goals/Deliverables:
[CODE] + [SPIKE REPORT]
Students are required to implement a 3D vector math API in C++, then **expose it to the Blueprinting system**. In order to pass this task, your vector API must be able to have:

- Have an X, Y and Z float.
- X, Y and Z values should be able to be set and gotten separately.

Extension:
Add the following methods to your API:

- Vector addition
- Vector subtraction
- Multiplication
- Division
- Dot product OR cross product OR get unit vector

Recommendations:

- Make sure to look up a tutorial how to expose C++ to blueprints before starting to code.
- You will need to use “UPROPERTY” in your code in order to enable you to expose the code to blueprint.

Spike 18: Input Handling

CORE SPIKE

Context: During your time as a developer, you will likely be tasked with accepting input from various devices. Whether it be a controller for console development, a touch screen for mobile development, a VR device or something a bit more niche. A developer shouldn't restrict themselves to just a mouse and keyboard.

Knowledge/Skill Gap: Students should know how to map actions to specific keys. While Unreal automatically maps the more popular inputs such as WASD for movement and space for jump, there will be many times where you will wish to change this.
Students would benefit from knowing how to use any input method that they wish to use in the Unreal engine and how to properly map this input.

Goals/Deliverables:
[CODE] + [SPIKE REPORT]
Create a new project form the FPS template. Remap the movement keys from WASD to SDAW (change forward to S, left to D, etc.). Open the player actor and make it print a message to the screen when a button is pressed.

Extension:
Also use a different input method. This input method can be anything you want (phone touch screen, phone gyroscope, controller, VR headset, ect.). You don't have to do anything too special, just have an element that reacts to the input.

Recommendations:

- The extension for this task is highly valuable. In the games industry, you will be expected to know how to accept input from more than just a mouse and keyboard.
- If you're wanting to complete a distinction project this year but do not yet have an idea, this spike is a great place to start.

Spike 19: Sounds

CORE SPIKE

Context: While it may seem like a trivial aspect to the development of most games, proper sound implementation is incredibly important to making a polished product. High quality sound implementation is often overlooked, but poor quality sound implementation is one of the most noticeable errors a developer can make.

Knowledge/Skill Gap: Developers should know how to play both global and location based sounds. Global sounds are heard at the same volume no matter where the player is on the map. This style of sound is often used for things like announcers, first person dialog, weapon sounds in first person games, non-diegetic sounds, etc. Location based sounds have their volume and parallaxing based upon where the player is in reference to the origin of the sound. This approach will be used for most diegetic elements that are not emanating from the player.

Goals/Deliverables:
[CODE] + [SPIKE REPORT]
Load and play 2 different sounds. Play one sound globally so that it can be heard at the same volume no matter where the player is on the map. Have the second sound originate at a location on the map.

Extension:
Use the Unreal performance profiler to show how much RAM your sound is using. Also show the memory voices (also known as streams, waves, channels, and a stupendous amount of other synonyms). Memory voices are the number of channels you have to play sound on. You should try to breach the maximum number of memory voices and show what happens.

Recommendations:

- This is very quickly done through blueprinting.
- To avoid extreme annoyance, consider having the sounds play when a button is pressed rather than constantly.

Spike 20: Collision

CORE SPIKE

Context: Collision between two game objects is one of the most common interactions you will come across in games development. A badly optimised collision system can be a major performance bottleneck, causing noticeable framerate changes if not done correctly. It is because of this that knowing popular collision methods is imperative.

Knowledge/Skill Gap: Students should know how to implement a collision system in Unreal Engine 4. Ideally, students should be able to implement their own collision methods in order to better optimise any project they may work on in the future.

Goals/Deliverables:
[CODE] + [SPIKE REPORT]
Use Unreal's inbuilt collision system in order to detect collisions between two objects. When an actor is colliding with another actor, change the material of both. Any actors not colliding with another should all have a different material to that which is applied on collision.

Extension:
Implement your own collision system using C++ or blueprints. Any one type of system will be acceptable for this (sphere/sphere or rectangle/rectangle).

Recommendations:

- Any students wanting a challenge can try to implement separating axis theorem working. The collision method that can work with meshes of any shape. This is a large undertaking and likely worth a distinction alone.

Spike 21: Measuring Performance

CORE SPIKE

Context: Tools for measuring performance are almost mandatory in games development. When assigned tasks in a game development studio, you will likely be given a performance budget. Any developer that does not know how to use these tools correctly will be crippling themselves and the studio they work for.

Knowledge/Skill Gap: Students will need to learn how to record changes in performance when different changes are implemented. To do this, students need to know how to use the Unreal Engine's built in performance measuring toolkit.

Goals/Deliverables:
[CODE] + [SPIKE REPORT]
Write a short report explaining how to measure performance in the Unreal Engine. After this, record and report the performance difference between 100 dynamic lights as opposed to 100 baked lights. Include in this report why the performance difference exists. You should use your sample scene from spike 16 or a similar scene to demonstrate the lighting (otherwise we won't have anything to light).

Extension:
Make your collision system from spike 20 more efficient. The performance increase does not have to be major, but you will have to note the difference in efficiency.

Recommendations:

- If you are doing the extension and can't think of a way to make your individual collision checks more efficient, think of a way to flag when collisions do not need to be checked for.