Spike: 16
Title: Create Simple Blueprint in C++

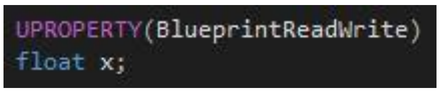
Author: Ford Killeen, 9731822

Goals / deliverables:
Goals this spike aims to achieve:
- Create an Unreal Engine 4 blueprint in C++
- Must have an X, Y and Z float value with separate get/set methods
- Vector addition, subtraction, multiplication, division and dot product

Deliverables required:
- Proof of UE4 C++ blueprint class
- Proof of working functions and set/get methods
- Spike report

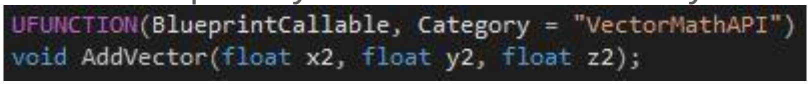
Technologies, Tools, and Resources used:
The following is required to complete this spike:
- Unreal Engine 4 (ver. 4.13.2)
- Visual Studio 2015 Community Edition
- Online UE4 material and guides

Tasks undertaken:
The list below details the steps taken to complete this spike.
- The first thing I did was look up and research how to create a C++ class in UE4 that could then be used to make a blueprint class. I found some good documentation on the official Unreal Engine website along with a few other handy guides online.
- In UE4, I selected *File -> New C++ Class*, followed the steps to create my `VectorMathAPI` class and let Unreal handle the rest.
- Upon seeing the newly created `.cpp` and `.h` files open up in Visual Studio I proceeded to add to my class. I first added the three float values using the `UPROPERTY()` macro so that they could be used in the blueprint system.



```
UPROPERTY(BlueprintReadWrite)
float x;
```

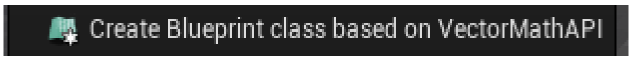
- I then added all the functions I wanted using the `UFUNCTION()` macro to again allow the blueprint system to see and use my code.



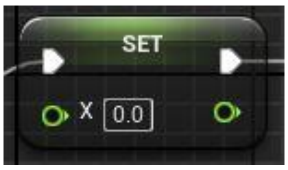
```
UFUNCTION(BlueprintCallable, Category = "VectorMathAPI")
void AddVector(float x2, float y2, float z2);
```

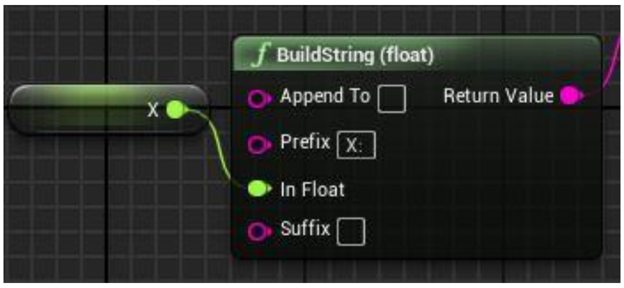
- Once the `.h` header file was complete I got onto implementing the functions in the `.cpp` file. This was quite easy to do and didn't take long at all.

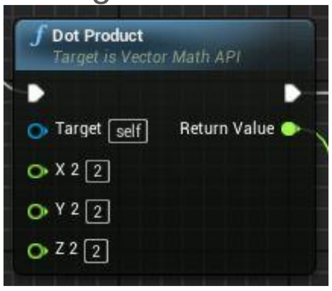
- I then saved and built my C++ code and jumped back into UE4. In UE4 I hit *Compile* to ensure my C++ code had compiled successfully and been recognised by the Unreal Engine.
- In UE4, under the *Content Browser -> C++ Classes -> Scene -> Public*, I right clicked my `VectorMathAPI` class and selected *Create Blueprint Class based on VectorMathAPI*.

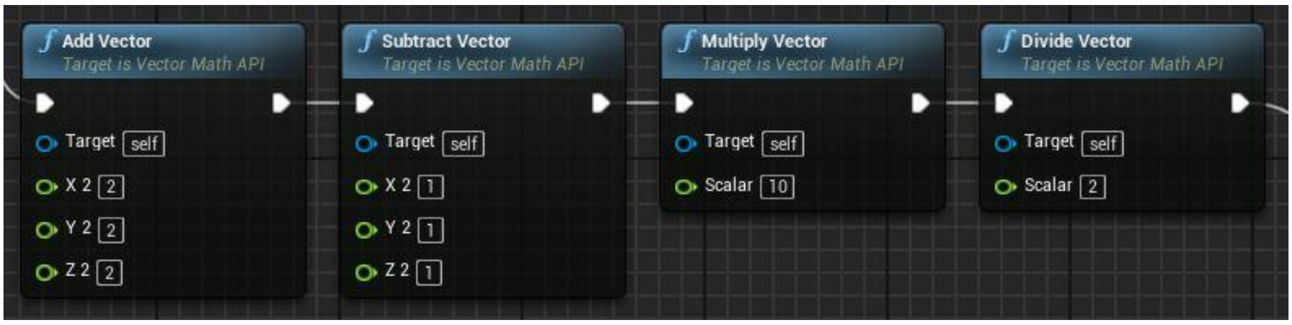


- I then added all the nodes I wanted to use to show off my API as well as some `Print String` nodes to see that it was all working.



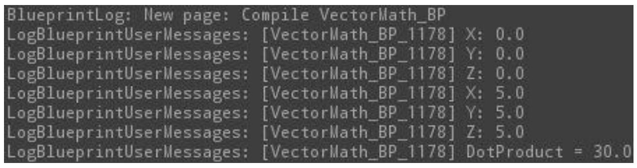






What we found out:

By completing this spike we found out how to create our own C++ class APIs which can be used to create blueprint classes. These custom C++ classes allow programmers to set up the core functionality and custom features, exposing them to the blueprinting system, for designers to then come along and use. Allowing more advanced nodes than the default blueprint system.



```
BlueprintLog: New page: Compile VectorMath_BP
LogBlueprintUserMessages: [VectorMath_BP_1178] X: 0.0
LogBlueprintUserMessages: [VectorMath_BP_1178] Y: 0.0
LogBlueprintUserMessages: [VectorMath_BP_1178] Z: 0.0
LogBlueprintUserMessages: [VectorMath_BP_1178] X: 5.0
LogBlueprintUserMessages: [VectorMath_BP_1178] Y: 5.0
LogBlueprintUserMessages: [VectorMath_BP_1178] Z: 5.0
LogBlueprintUserMessages: [VectorMath_BP_1178] DotProduct = 30.0
```

Issues

It seems silly but one of the main issues I had was getting my functions to appear in the blueprint system. After compiling and rebuilding my project over and over again I finally found that it was because I had initially used `UFUNCTION(BlueprintNative)` instead of `UFUNCTION(BlueprintCallable)` in my C++ header file. It was easy to fix once I found out what the issue was, but for someone who had never created an API for the blueprint system it was a little frustrating and confusing at first.