**WARNING:** It is strongly recommended that you DO NOT copy/paste all of the code at once. If your project or class have different naming in any way, your code will not work. You may, however, copy/paste the highlighted areas.

|  |
| --- |
| // Fill out your copyright notice in the Description page of Project Settings.  #pragma once  #include "GameFramework/Character.h"  #include "BaseCharacter.generated.h"  //Step 1 : Make class Blueprintable //Must derive from an A\* or U\* Class  UCLASS(Blueprintable)  class TS\_TUTFORLUISANDY\_API ABaseCharacter : public ACharacter  {  GENERATED\_BODY()  public:  //Step 2: Expose a float property  UPROPERTY(BlueprintReadWrite, EditAnywhere, Category = "BaseCharacter")  float Health = 100;  //Step 3: Expose a boolean property  UPROPERTY(BlueprintReadOnly, VisibleAnywhere, Category = "BaseCharacter")  bool isDead = false;  //Step 4: Make a helper function, just because we are lazy  virtual void CalculateDead();  //Step 5: Expose a method  UFUNCTION(BlueprintCallable, Category = "BaseCharacter")  virtual void CalculateHealth(float delta);  //Step 6: Editor code to make updating values in the editor cleaner  #if WITH\_EDITOR  virtual void PostEditChangeProperty(FPropertyChangedEvent& PropertyChangedEvent) override;  #endif  public:  // Sets default values for this character's properties  ABaseCharacter();  // Called when the game starts or when spawned  virtual void BeginPlay() override;    // Called every frame  virtual void Tick( float DeltaSeconds ) override;  // Called to bind functionality to input  virtual void SetupPlayerInputComponent(class UInputComponent\* InputComponent) override;      }; |

* + 1. **Step 1:** All we did was add *Blueprintable* to the class declaration.
    2. **Step 2:** We make a new Health property, of type *float*. It has the following keywords:
       1. BlueprintReadWrite: this property is readable and writable from within Blueprint
       2. EditAnywhere: it can be edited from any class
       3. Category = “BaseCharacter”: this property will appear in the category “BaseCharacter” in the Blueprint Editor
    3. **Step 3:** We create a new boolean property named isDead. It has the following keywords:
       1. BlueprintReadOnly: The property is only readable from within Blueprint, so Blueprints cannot change it.
       2. VisibleAnywhere: Other classes can see this property, but will not be able to write to it. Only this class can edit it.
       3. Category = “BaseCharacter”: this property will appear in the category “BaseCharacter” in the Blueprint Editor
    4. **Step 4:** We’re setting up a simple helper function that will do some simple math. This is just a way to separate out the actual health adjustment math from a check to see if that adjustment just killed the character. This will become clearer when we edit the C++ file.
    5. **Step 5:** We set up the CalculateHealth function. This will do most of the heavy lifting for us in managing our character’s Health value. The function has the following keywords:
       1. BlueprintCallable: This keyword means that Blueprints can call this function. Put another way, this will turn our function into our own custom Blueprint node!
       2. Category = “BaseCharacter”: this property will appear in the category “BaseCharacter” in the Blueprint Editor
    6. **Step 6:** This is just some “helper code” required by the engine so that it knows what to do with this function if we start changing values in the editor.
  1. Now let’s take a look at the \*.cpp file, which contains the implementation of all the functionality we set up in the header file. Once again, the section below contains all the code for the class, with the parts we added highlighted for convenience:

**WARNING:** It is strongly recommended that you DO NOT copy/paste all of the code at once. If your project or class have different naming in any way, your code will not work. You may, however, copy/paste the highlighted areas.

|  |
| --- |
| // Fill out your copyright notice in the Description page of Project Settings.  #include "TS\_TutForLuisAndy.h"  #include "BaseCharacter.h"  // Sets default values  ABaseCharacter::ABaseCharacter()  {  // Set this character to call Tick() every frame. You can turn this off to improve performance if you don't need it.  PrimaryActorTick.bCanEverTick = true;  }  // Called when the game starts or when spawned  void ABaseCharacter::BeginPlay()  {  Super::BeginPlay();    }  // Called every frame  void ABaseCharacter::Tick( float DeltaTime )  {  Super::Tick( DeltaTime );  }  // Called to bind functionality to input  void ABaseCharacter::SetupPlayerInputComponent(class UInputComponent\* InputComponent)  {  Super::SetupPlayerInputComponent(InputComponent);  }  //Step 1: Implement the CalculateHealth function.  void ABaseCharacter::CalculateHealth(float Delta)  {  Health += Delta;  CalculateDead();  }  //Step 2: Implement the CalculateDead function.  void ABaseCharacter::CalculateDead()  {  if (Health <= 0)  isDead = true;  else  isDead = false;  }  #if WITH\_EDITOR  //Step 3: Implement the remainder of our helper code, used by the editor when we change values.  void ABaseCharacter::PostEditChangeProperty(FPropertyChangedEvent& PropertyChangedEvent)  {  isDead = false;  Health = 100;  Super::PostEditChangeProperty(PropertyChangedEvent);  CalculateDead();  }  #endif |

* 1. Step 1: This is the base functionality for CalculateHealth. It simply takes in a float value (named Delta) and adds it to the existing value for Health. It then runs the CalculateDead function.
  2. Step 2: CalculateDead functionality is just a quick check to see if health is less than or equal to zero. If it is, we set the *isDead* property to True. If it’s not, we set *isDead* to false.
  3. Step 3: we finish up our little bit of helper code, which handles what should happen if we adjust values in the Editor.

1. Press Ctrl + Shift + B to build your solution. If you get errors, you have probably misspelled something.
2. Relaunch the Editor! Since you’ve added a new class to your project and added new properties to that class, you need to reload your entire project. You don’t have to close, you can just open your current project from File > Recent Projects.

**Fun Fact:** The step comments in your code are going to translate into the tooltips for your class. Obviously, that’s less than ideal for production, but for academic purposes we’re going to let it slide.

# 