**Guards distraction.**

Add code to **OnNoiseHear** method in **AIGuard.cpp** to look at where the noise came from **FRotationMatrix::MakeFromX(X).Rotator();** This creates a rotation vector from direction vector.

X should be a normalised direction vector. To get direction vector

FVector Direction = Location - GetActorLocation();

To normalise

Direction.Normalize();

We have to create a rotator vector from the earlier code. For that add the following code.

FRotator NewLookAt = FRotationMatrix::MakeFromX(Direction).Rotator();

Now we use the above variable to use in the **SetActorRotation** method.

SetActorRotation(NewLookAt);

Final set of codes will look like this

FVector Direction = Location - GetActorLocation();

Direction.Normalize();

FRotator NewLookAt = FRotationMatrix::MakeFromX(Direction).Rotator();

SetActorRotation(NewLookAt);

The AIGuard should look at the direction of the noise. One problem is the guard looks up if the bullet hits a wall up.

Exercise: fix it.

Ans

NewLookAt.Roll = 0.0f;

NewLookAt.Pitch = 0.0f;

Now we need to turn the guard back to its initial rotation after a certain amount of time.

So first go to AIGuard.h and create a variable as follows.

FRotator OriginalRotation;

Now we initialise the variable in the **begin play** method as follows.

OriginalRotation = GetActorRotation();

This will set the variable with the initial rotation of the guard.

Now we must set timer. So create a timer handle in the header file method

FTimerHandle TimerHandle\_ResetOrientation;

Now include the timemanager header file

#include "TimerManager.h"

Now go to the **OnNoiseHear** method and reset timer first

GetWorldTimerManager().ClearTimer(TimerHandle\_ResetOrientation);

GetWorldTimerManager().SetTimer(TimerHandle\_ResetOrientation, this, &AFPSAIGuard::ResetOrientation,3.0f);

We need to create a function called **ResetOrientaion** that is passed into the method **SetTimer**.

Declare the function as follows in the header file.

UFUNCTION()

void ResetOrientation();

Create its implementation as follows in the C++ file.

void AFPSAIGuard::ResetOrientation()

{

SetActorRotation(OriginalRotation);

}

Now we need to add game over when the guard spots the player.

We are adding a rule to the game. This is done in **GameMode**.

So in gamemode, we need to add a Boolean called **MissionComplete** to method **CompleteMission**

AFPSObjectiveGameMode();

void MissionComplete(APawn\* InstigatorPawn, **bool MissionComplete**);

We need to add the same variable in declaration and implementation.

We need to add this variable to the BP version of the function **OnMissionCompleted** in the gamemode.

void OnMissionCompleted(APawn\* InstigatorPawn, **bool MissionComplete**);

In the gamemode.cpp file we need to add the variable in the **OnMissionControl** BP function call at the end inside the CompleteMission function as follows.

OnMissionCompleted(InstigatorPawn, MissionComplete);

We are calling the MissionComplete function in **ExtractionZone** as well.

So we need to pass the boolean while calling the function in the **ExtractionZone.cpp**  in the **HandleOverlap** function.

GM->MissionComplete(MyPawn, **true**);

Now we need to specify the failure state when the guard sees the player.

So head into the **AIGuard.cpp** file and add the following in **OnPawnSeen** function.

AFPSObjectiveGameMode\* GM = Cast<AFPSObjectiveGameMode>(GetWorld()->GetAuthGameMode());

if (GM)

{

GM->MissionComplete(SeenPawn, false);

}

Need to include 2 header files in the AIGuard.cpp

#include "FPSObjectiveGameMode.h"

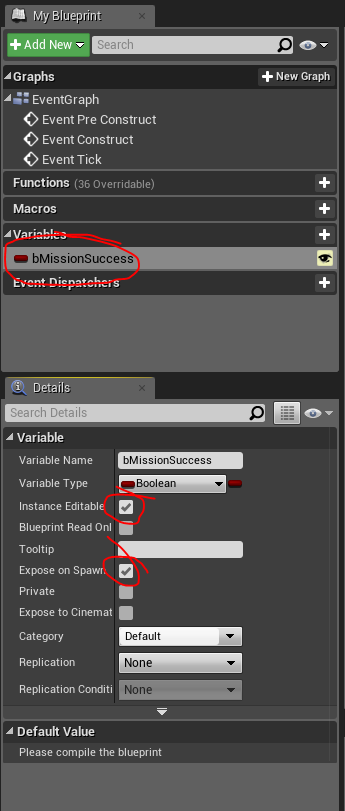
#include "GameFramework/Actor.h"

Now when the AIGuard sees the player the game will be over.

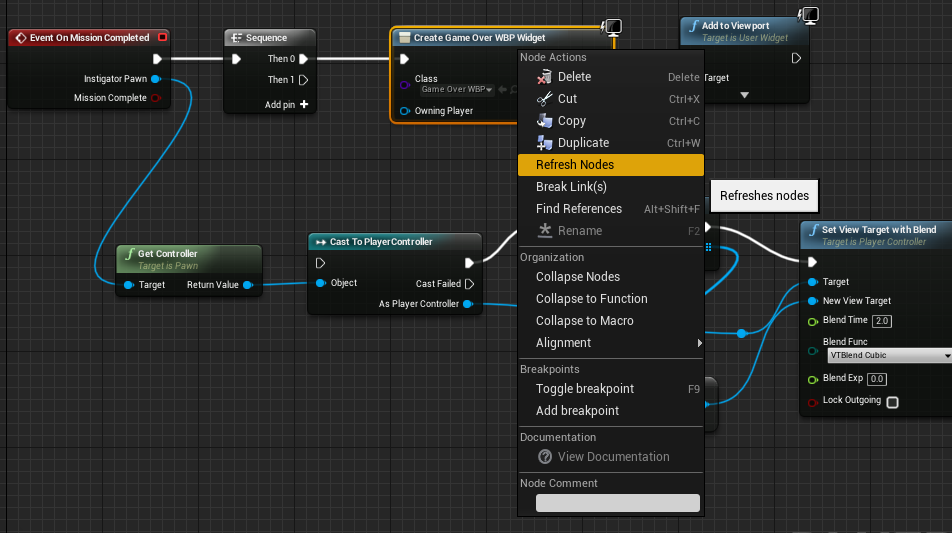
**Game Over Message**

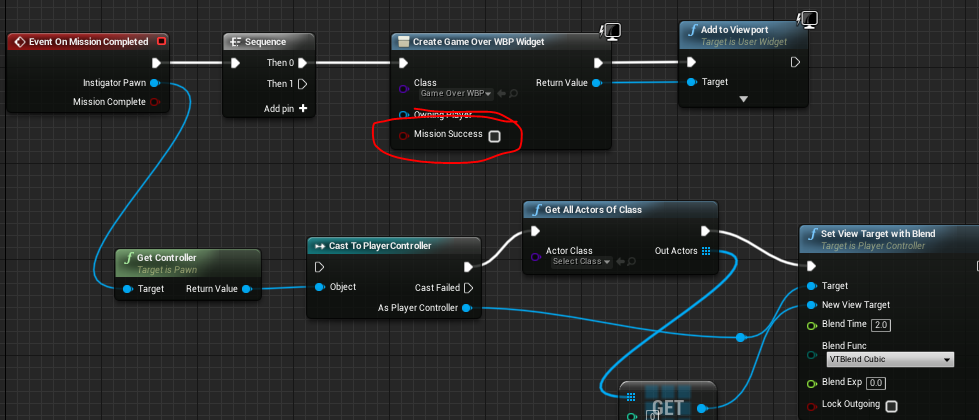
We need to change the text when the game is over. It should tell the player whether the game is over or mission completed.

Go to **GameOverWidgetBP** and create a Boolean variable called **bMissionSuccess** and expose on spawn and instance editable.

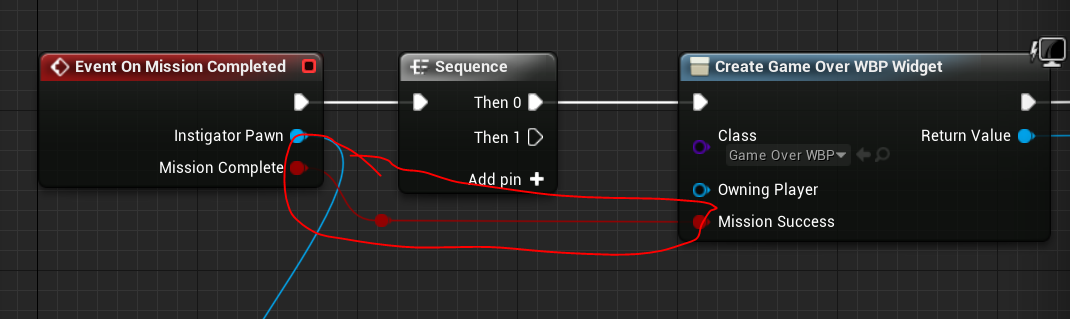


Now go to **BP\_GameMode** and rightclick on the create widget node an refreshnode and we will get the new variable create in the widget visible.

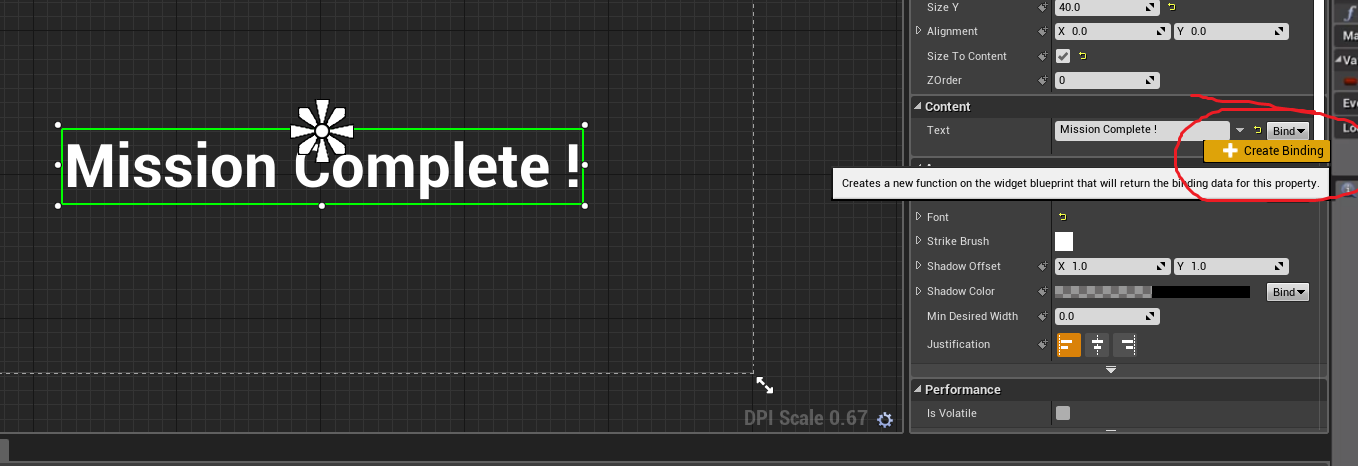




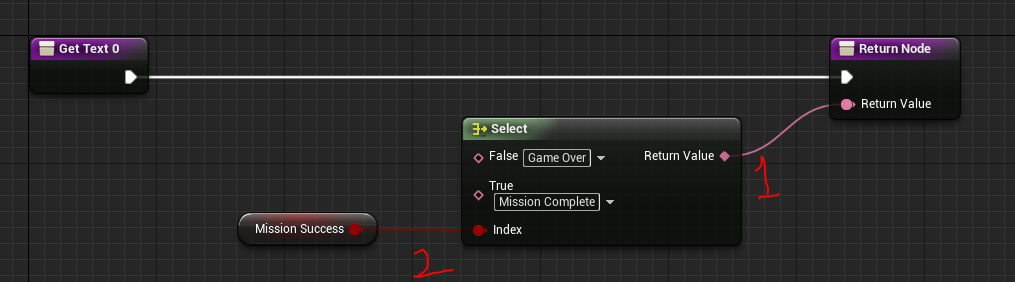
And connect the Boolean



Now go to the **GameOverWBP** widget and create a binding for the text “Mission Complete”.



In the binding graph add the following BP.



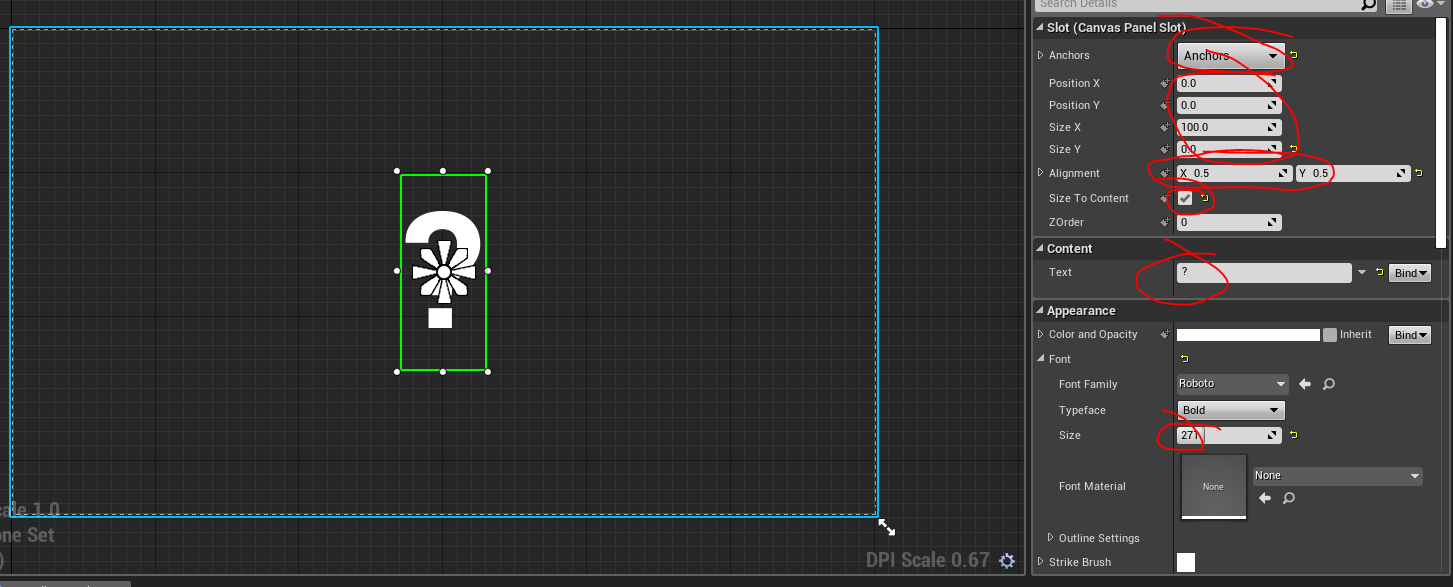
When we connect the return value of the select node to the return node the options of the select node will change to text.

Now game should be working!

**AI Guard Status**

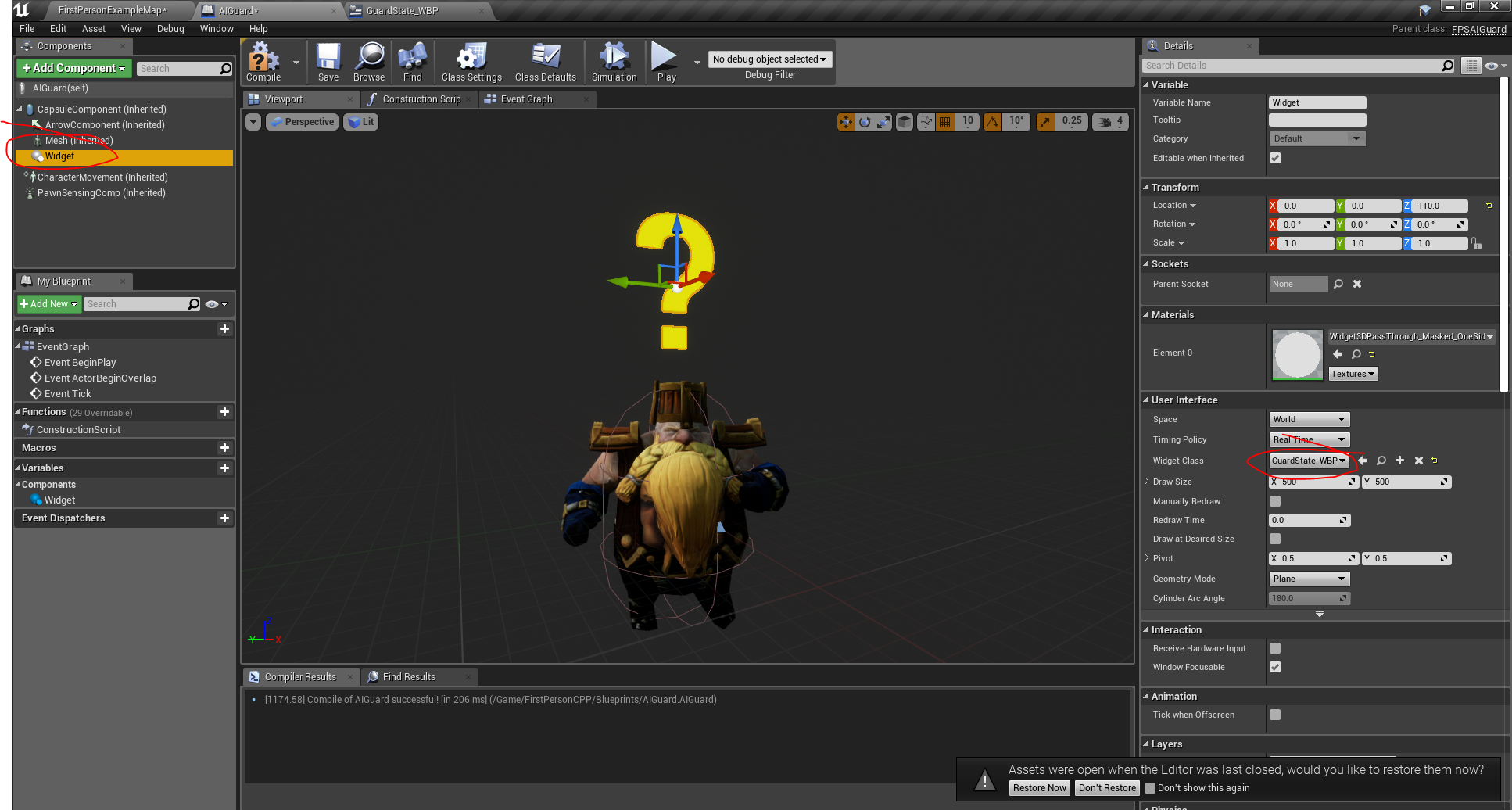
So create a UMG Widget called “Status”

Add a question mark. Set the anchor to centre.

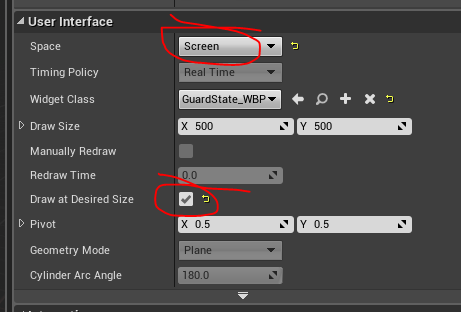


Now we have to add this to the guard. This will be done partially in BP and Partially in C++.

Go to GuardBP and add a component ‘Widget’.



Select the new widget from the details panel and move the widget up and about the character. You may have to change the size of the text in the Widget BP



Select screen from the details so that it will be screen aligned when playing the game.

Now we need to create an enum to store the state of the guard.

So go to the header file of the **FPSAIGuard** add the declaration at the beginning outside the class .

UENUM(BlueprintType)

enum class EAIState : uint8

{

Idle,

Suspicious,

Alerted

};

We have to specifically state uint8 to get it in Blueprints

Now create a variable of state and 2 functions to change the value of the state in the **protected** section as follows.

EAIState GuardState;

void SetGuardState(EAIState NewState);

UFUNCTION(BlueprintImplementableEvent, Category = "AI")

void OnStateChanged(EAIState NewState);

We will not have implementation of the **OnStateChange** function. It will be done in BP.

Now add the implementation of SetGuardState as follows.

void AFPSAIGuard::SetGuardState(EAIState NewState)

{

if (GuardState == NewState)

{

return;

}

GuardState = NewState;

OnStateChanged(GuardState);

}

We are only calling the **OnstateChange** function. The implementation is done in BP.

Now we need to initialise the variable in the **constructor** as follows.

**GuardState = EAIState::Idle;**

Now in the **OnPawnScene** function we need to change the enum to alerted. So add the following to **OnPawnSeen** function.

SetGuardState(EAIState::Alerted);

Now in the **OnNoiseHear** function we need to change the enum to Suspicious. So add the following to the beginning of the function.

**if (GuardState == EAIState::Alerted)**

**{**

**return;**

**}**

This will ensure that when the guard is in alert state it will not go to any other state.

Add the following to the end of the function to change the state when guard hears.

SetGuardState(EAIState::Suspicious);

Now we have to reset the state when **ResetOrientation** function is called. Add the following to ResetOrientation function.

if (GuardState == EAIState::Alerted)

{

return;

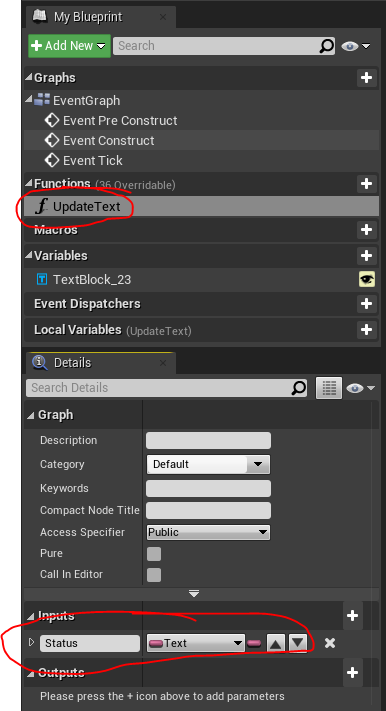
}

SetActorRotation(OriginalRotation); // statement was already in the function

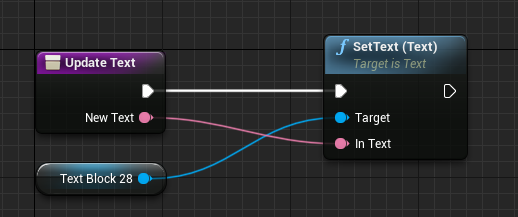
SetGuardState(EAIState::Idle);

Now go to the **GuardStatus** widget make the text “is Variable”

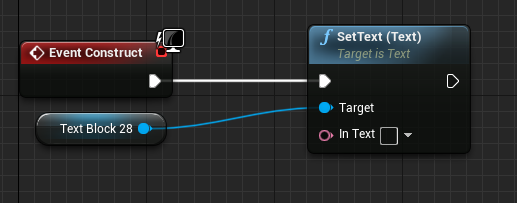
Create a function called UpdateText, add an input to the function



Add the following BP.

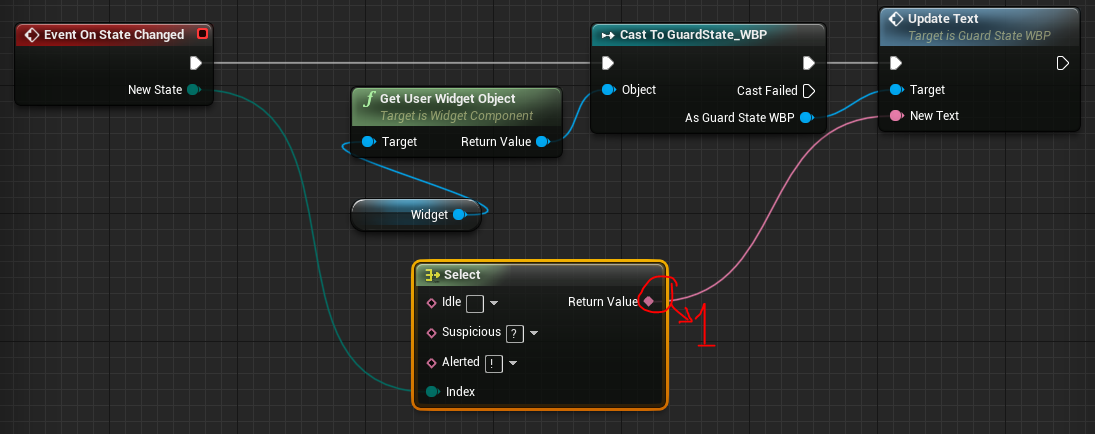


Add the following BP to the **Event Construct** of the **GuardStatus**.



Now go to the **AIGuard** blueprint event graph and add the following BP.

(need to compile CPP to get the exposed function)



Connect the return value of the select first so that the idle suspicious text will be exposed to edit.

Finished!