UNREAL FAST TRACK
WORKSHOP FOUR

Workshop Four: Starting the Twin Stick Shooter and Implementing AI

Learning Points

Test Drive (2 hours)

- Reviewing project setup
- Reviewing Unreal Engine class framework
- Using C++ in Unreal Engine
- Building hero characters
- Building simple AI enemies

Grand Prix (1 hour)

- Making enumerations
- Creating more complex AI with Blueprints
- Pathfinding for AI enemies

Off-Roading and Discussion (2 hours)

- Introduction to Behavior Trees
- Exploring splines

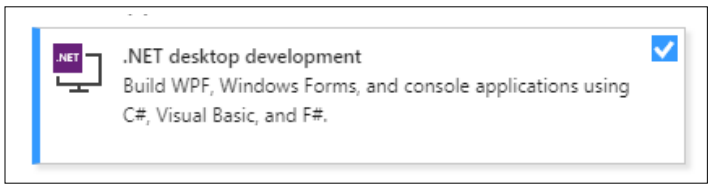
Test Drive

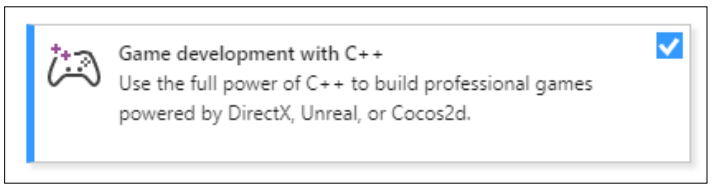
The fourth and fifth workshops use the same Unreal Online Learning course. In the fourth workshop, we will be doing the first half of the "Twin Stick Shooter with Blueprints" course. The course can be found here: https://www.unrealengine.com/en-US/onlinelearning-courses/twin-stick-shooter-with-blueprints.

The previous workshops focused on one specific aspect of Unreal Engine, but the goal of this course is to have you interact with many of the different systems within Unreal. Some of these systems will be familiar to you, which will allow you to review them and expand upon your knowledge; other systems will be brand-new to you, and so you will be interacting with them for the first time. The "Twin Stick Shooter with Blueprints" course will lead you through making a whole prototype, a different one from the one you have been building in the Grand Prix. By the end of the Fast Track, you will have two different prototypes built!

For some of the videos, we've included occasional mentions of tips and changes since the course came out here in the companion document. If needed, the tips and changes are time-stamped so you know when they are relevant. If there is no time stamp, you can wait until the end to read them.

For this course, you will need Visual Studio 2017 downloaded. IMPORTANT NOTE: When installing VS2017, you will need to download the ".NET desktop development" and "Game development with C++" modules for Visual Studio before beginning the Unreal Online Learning course.





4: Building the Level
0:31: The **BSP** tab has been renamed to "**Geometry**".
6:24: If you look closely at the windows on the left side of the photo, you'll see a diagonal line. That is the slide at Epic Games headquarters.
8:06: The **Post Process Volume Settings** section has changed since this video was made. You have to scroll down farther to find it, and the setting is now labeled "**Infinite Extent (Unbound)**".
8:14: The **Min Brightness** and **Max Brightness** settings have also moved. They are now in the **Lens** section, under "**Exposure**". You can use the **Search** bar in the **Details** panel to quickly find them.
8:48: "Save" has been renamed to "**Save Current**".

10: Building the Enemy Character
8:39: This node has been renamed to "**Set Timer by Function Name**".

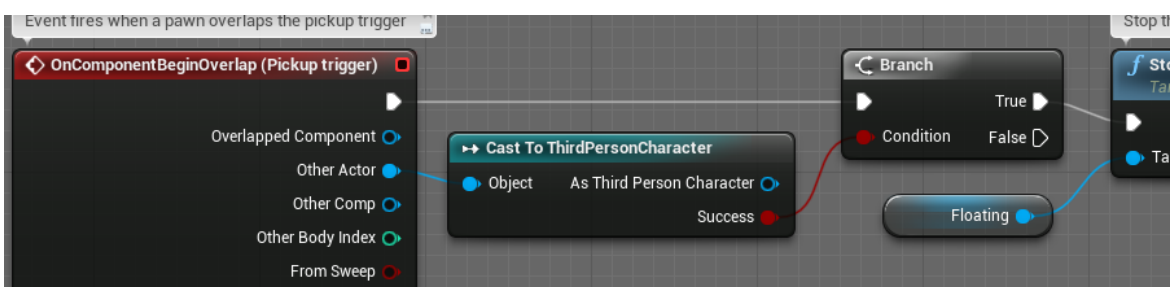
12: Twin Stick Shooter with Blueprints Quiz 2
After completing the quiz, pause the course and head on over to the Grand Prix section! You will finish the rest of the "Twin Stick Shooter with Blueprints" course in the next workshop.

Grand Prix

Let's add enemies to our game!

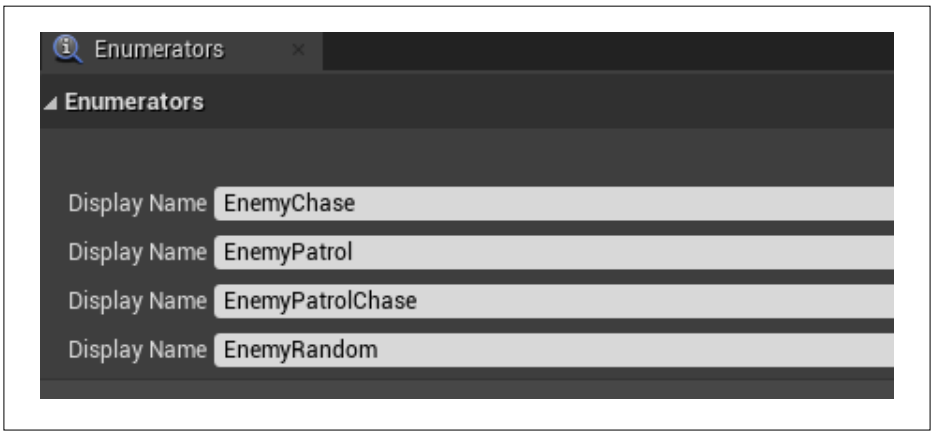
However, before we do that, we need to quickly add and change a few things.
First, currently, if we add enemies to our game, they would be able to pick up the powerups! We need to change that.

1. Find the **BP_Pickup_Parent** Blueprint in the **Context Browser** and open it.
2. Look for the **On Component Begin Overlap** event in the Event Graph. Disconnect it from the Stop node.
3. Drag a wire from the event's **Other Actor** pin and add a **Cast To ThirdPersonCharacter** node. Right-click on the node and convert it to a pure cast.
4. Off the **Cast To ThirdPersonCharacter** node's **Success** pin, add a **Branch** node.
5. Connect the **Branch** node's **True** pin to the **Stop** node's input execution pin.
6. Connect the **Branch** node's input execution pin to the **On Component Begin Overlap** event's output execution pin.



Now the AI won't be able to pick up powerups. Phew! Next, we will create a list of enemies.

1. In the **Content Browser**, create an enumeration by right-clicking and going to **Blueprints → Enumeration**. Name the enumeration "**E_EnemyType**". Double-click on it to open it.
2. On the right side, click the **New** button four times. Under "**Display Name**", enter the following names in order:
    a. **EnemyChase**
    b. **EnemyPatrol**
    c. **EnemyPatrolChase**
    d. **EnemyRandom**
3. Save the enumeration.

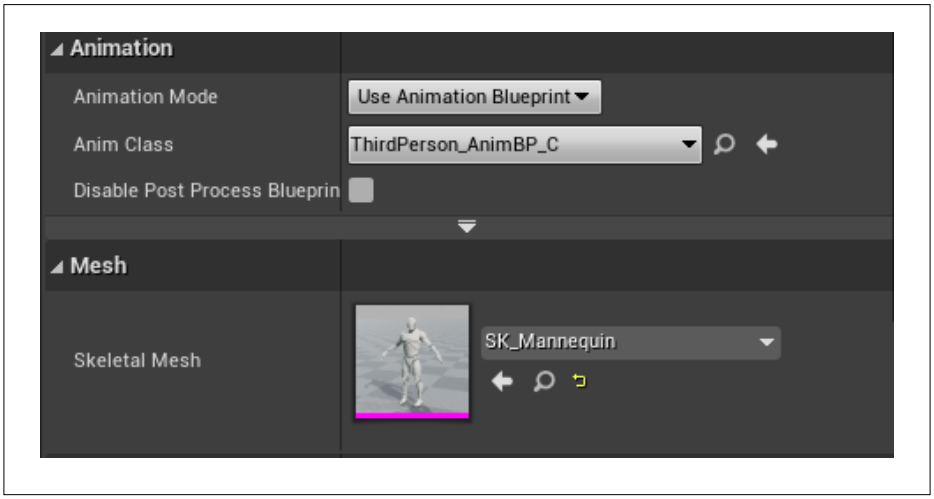


Let's make the AI Controller so we can reference it in other Blueprints. We'll come back to it later on.

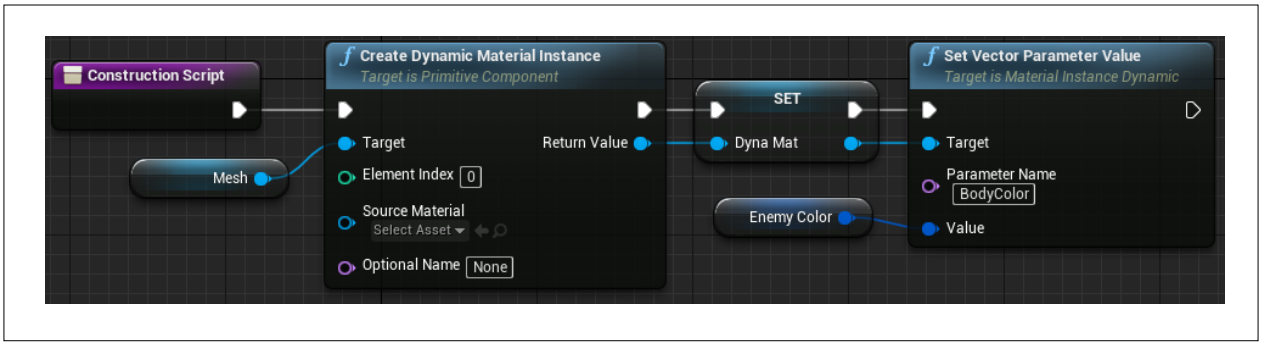
4. Back in the **Content Browser**, right-click and create a new Blueprint.
5. Just like you did in the course, name the Blueprint "**EnemyAI**" with the AI Controller class as the parent class.

Now we need a parent Blueprint enemy.

1. Create a Blueprint whose parent class is **Character**. Name the Blueprint "**EnemyCharacter**" and open it.
2. In the **Details** panel for the **Mesh** component, add "**SK_Mannequin**" to **Skeletal Mesh** and change **Anim Class** to "**ThirdPerson_AnimBP_C**".

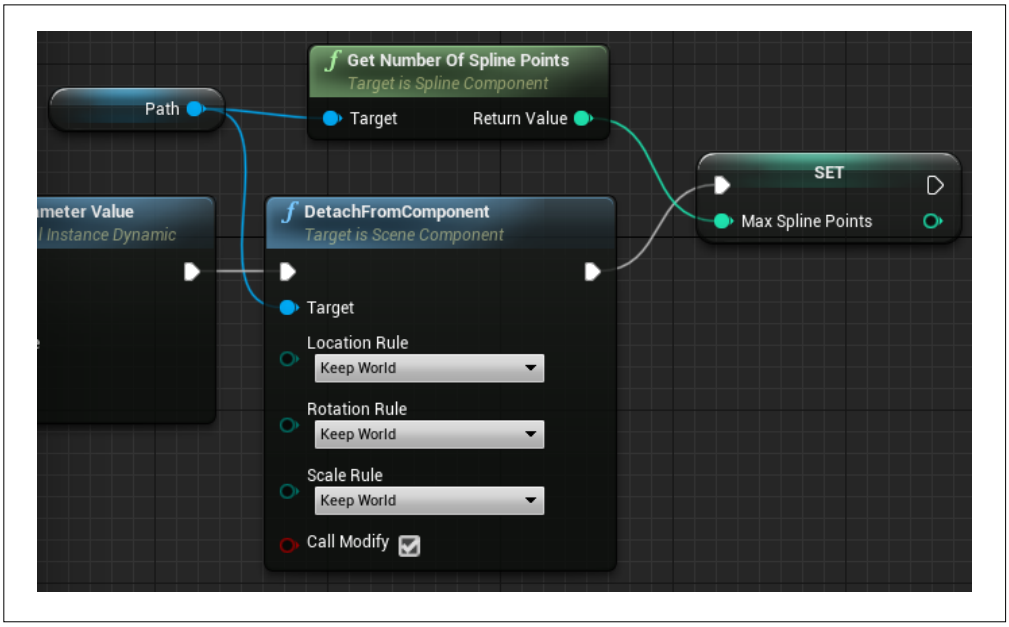


3. Re-create the Construction Script Blueprint nodes that you made in the "Twin Stick Shooter with Blueprints" **EnemyCharacter** Blueprint.
4. Drag a wire from the **Set Vector Parameter Value** node's **Value** pin and select "**Promote to Variable**". Name the variable "**EnemyColor**". Find the variable in the variables list and click the closed-eye icon to the right of the name. The icon should now be an open eye, making it a public variable.



5. Create a new variable. Name it "**EnemyType**". Under "Variable Type", click "**Enum**" and then find and select "**EEnemyType**". Make sure this variable is public, too.

6. Create an **Integer** variable and name it "**MaxSplinePoints**".
7. Create another **Integer** variable and call it "**SplineIndex**".
8. Add a **Spline** component and rename it to "**Path**".
9. In the **Details** panel for the **Spline** component, check the **Closed Loop** box.
10. Then, back in the Construction Script graph, drag and drop a reference to **Path** onto the graph.
11. From the **Path** reference pin, add a **DetachFromComponent** node and connect its input execution pin to the **Set Vector Parameter Value** node's output execution pin. Set the **DetachFromComponent** node's **Location Rule**, **Rotation Rule**, and **Scale Rule** properties to "**Keep World**".
12. From the **Path** reference pin again, add a **Get Number Of Spline Points** node. From that node's **Return Value** pin, add a **Set MaxSplinePoints** node.



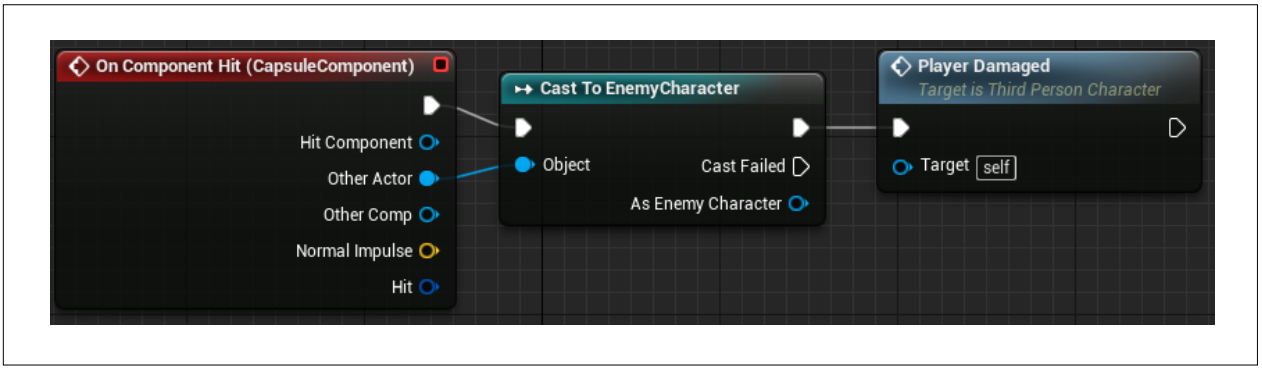
13. **VERY IMPORTANT**: Select the root component. Under the **Pawn** section in the **Details** panel, set **AI Controller Class** to "**EnemyAI**".
14. Compile the Blueprint.

Another very important step: Remember to add a Nav Mesh Bounds Volume to your Level!

1. Add a **Nav Mesh Bounds Volume** to your Level just like you did in the Unreal Online Learning course.
2. Scale it to your Level in the Viewport.

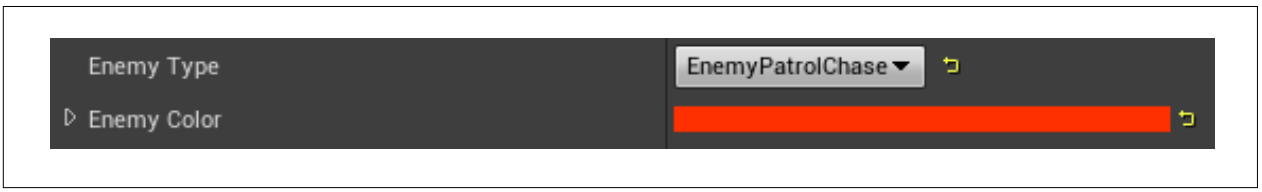
Next, so we don't forget, let's make the player lose health whenever they get hit by enemies.

1. Find and open the **ThirdPersonCharacter** Blueprint.
2. Select the **Capsule** component in the **Components** panel and add an **On Component Hit (CapsuleComponent)** event to the Event Graph.
3. From the event's **Other Actor** pin, add a **Cast To EnemyCharacter** node and connect the execution pins.
4. Finally, off the **Cast To EnemyCharacter** node's output execution pin, add a **PlayerDamaged** node to call the **PlayerDamaged** event we made in the last workshop.
5. Compile the Blueprint.



We are almost ready to start implementing AI logic. The last thing we need to do before doing that is to create child Blueprints of our parent **EnemyCharacte**r class.

1. In the **Content Browser**, right-click on the **EnemyCharacter** Blueprint and select "**Create Child Blueprint Class**".
2. Do this three more times. Name the four child Blueprints as follows:
    a. **EnemyChase**
    b. **EnemyPatrol**
    c. **EnemyPatrolChase**
    d. **EnemyRandom**
3. Open each of the child Blueprints, click on the root component, and look at the **Details** panel. For each Blueprint, set the **Enemy Type** property to match the name of the Blueprint. Then set the **Enemy Color** property to a different color for each.



4. For **EnemyChase** and **EnemyPatrolChase**, change **Max Walk Speed** in the **Character Movement** component to "**300**".
5. Compile all of the Blueprints.

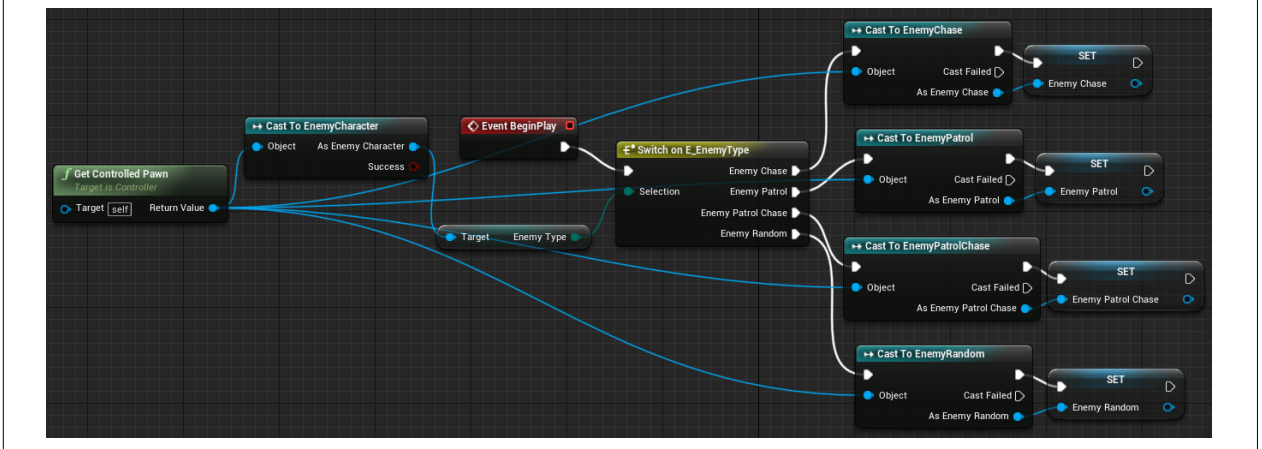
Now it is time to set the stage for building the AI behaviors. There will be a lot of nodes involved in building the behaviors, but if you follow along closely and use the pictures to check your work, you will do just fine.Compile all of the Blueprints.

1. Open the **EnemyAI** Blueprint.
2. First, we need to create references to each of the child Blueprints you created. Start by adding a **Get Controlled Pawn** node to the graph.
3. From the node's **Return Value** pin, add a **Cast To EnemyCharacter** node. Right-click on the node and select "**Convert to Pure Cast**".
4. Off the **Cast** node's **As EnemyCharacter** pin, place a **Get EnemyType** node.
5. From the **Get EnemyType** node's output pin, place a **Switch on E_EnemyType** node. If you search for "switch" it should be the only node that appears. The text on the node's output execution pins should look familiar; it is the same text you entered earlier when setting the Enemy Type property of the child Blueprints.
6. Place an **Event BeginPlay** node off the **Switch** node's input execution pin.

In the following, you'll be doing the same thing off each of the **Switch** node's output execution pins.

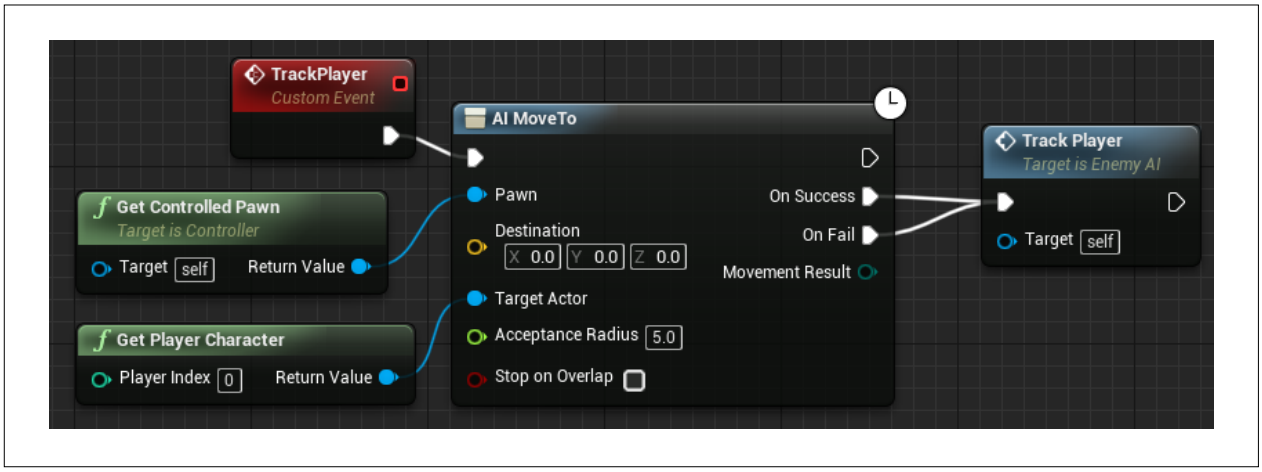
7. Off the **EnemyChase** pin, add a **Cast To EnemyChase** node. From the **Cast** node's **As EnemyChase** pin, drag a wire and select "**Promote to Variable**". Name the variable "**EnemyChase**".
8. Off the **EnemyPatrol** pin, add a **Cast To EnemyPatrol** node. From the **Cast** node's **As EnemyPatrol** pin, drag a wire and select "**Promote to Variable**". Name the variable "**EnemyPatrol**".
9. Off the **EnemyPatrolChase** pin, add a **Cast To EnemyPatrolChase** node. From the **Cast** node's **As EnemyPatrolChase** pin, drag a wire and select "**Promote to Variable**". Name the variable "**EnemyPatrolChase**".
10. Off the **EnemyRandom** pin, add a **Cast To EnemyRandom** node. From the **Cast** node's **As EnemyRandom** pin, drag a wire and select "**Promote to Variable**". Name the variable "**EnemyRandom**".

Next, drag a wire from the **Return Value** pin on the **Get Controlled Pawn** node you added in step 2 of this section and connect it to the **Object** pin on each of the four **Cast** nodes you created.

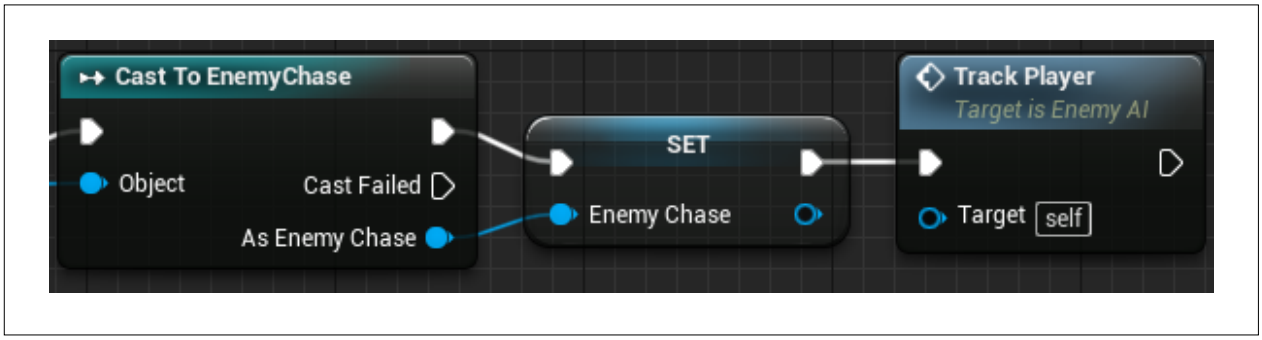


Now that you've created references to all of the enemies, let's start building their behaviors! Let's begin with **EnemyChase**, which, just like in the "Twin Stick Shooter with Blueprints" course, will chase the player.

1. At another spot in your Event Graph, add a custom event and name it "**TrackPlayer**".
2. From the **TrackPlayer** event, add an **AI MoveTo** node.
3. Just like in the course, add a **Get Controlled Pawn** node and connect its **Return Value** pin to the **AI MoveTo** node's **Pawn** pin.
4. Then, similarly, add a **Get Player Character** node and connect its **Return Value** pin to the **AI MoveTo** node's **Target Actor** pin.
5. From the **AI MoveTo** node's **On Success** pin, ***not its output execution pin***, add a **TrackPlayer** node. Then connect the **AI MoveTo** node's **On Fail** pin to the **TrackPlayer** node's input execution pin, too.



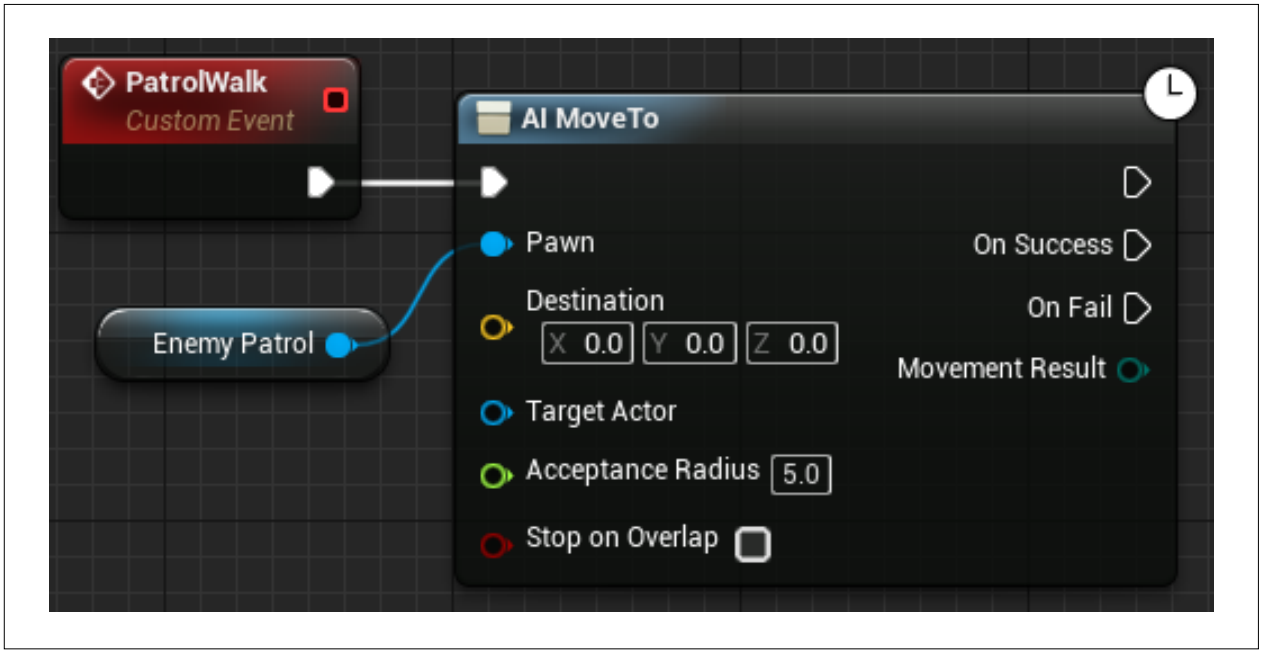
6. Finally, return to the node where you promoted **EnemyChase** to a variable. Off that node's output execution pin, add a **TrackPlayer** node.



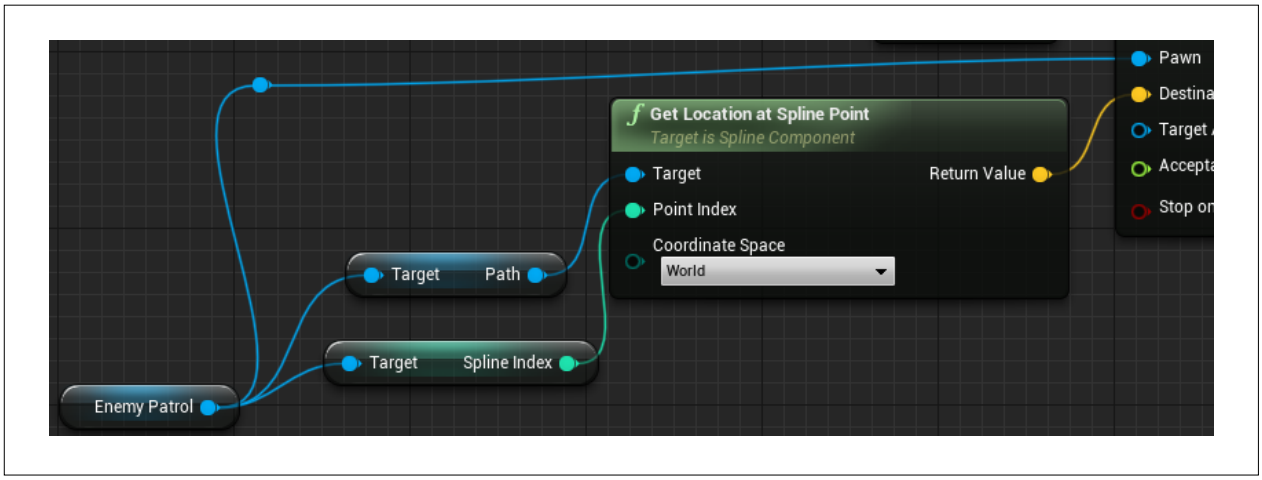
7. Compile the Blueprint.
8. Add an **EnemyChase** Blueprint to your Level and test it to see if it works! You may want to adjust this enemy's walk speed depending on the Level of challenge you want.

On to the second enemy! You will now be creating behavior for EnemyPatrol. This enemy will follow a specific path and never deviate from it. Its behavior will be a bit more complicated. We are going to be giving this enemy a path to walk with a number of points along the path. Every time the enemy walks to one point, the AI Controller will tell the enemy to walk to the next point. When the patrolling enemy walks to the last point, the AI Controller will tell it to walk back to the first point, finishing the loop. Let's do it!

1. Create a custom event called "**PatrolWalk**" and add it to the graph.
2. From the **PatrolWalk** event, add an **AI MoveTo** node.
3. Drag and drop an **EnemyPatrol** reference onto the graph. You'll be using this specific node for many different pins. Connect the **EnemyPatrol** node to the **AI MoveTo** node's **Pawn** pin.

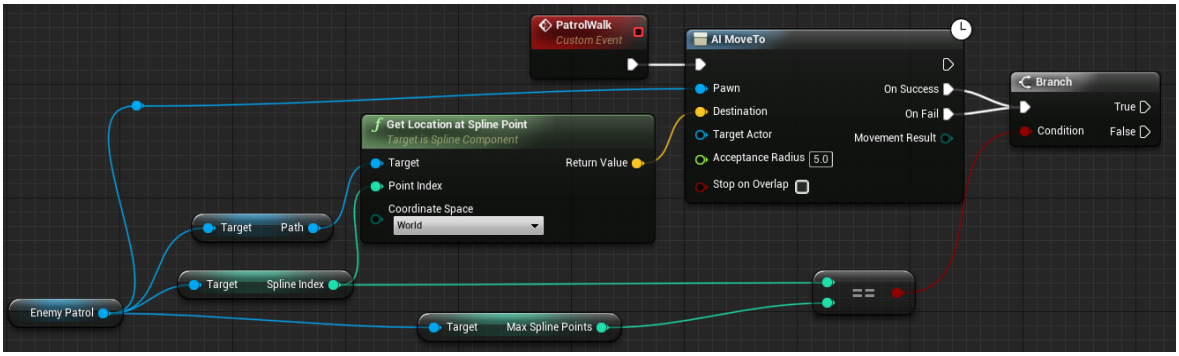


4. Drag another wire from the **EnemyPatrol** node and add a **Get Path** reference. The **Get Path** node will reference the **Spline** component you created in the Blueprint.
5. From the **Get Path** node's output pin, add a **Get Location at Spline Point** node. Connect that node's **Return Value** pin to the **AI MoveTo** node's **Destination** pin.
6. Once more, drag a wire from the **EnemyPatrol** node, and add a **Get Spline Index** node. Connect the **Get Spline Index** node's output pin to the **Get Location at Spline Point** node's **Point Index** pin.

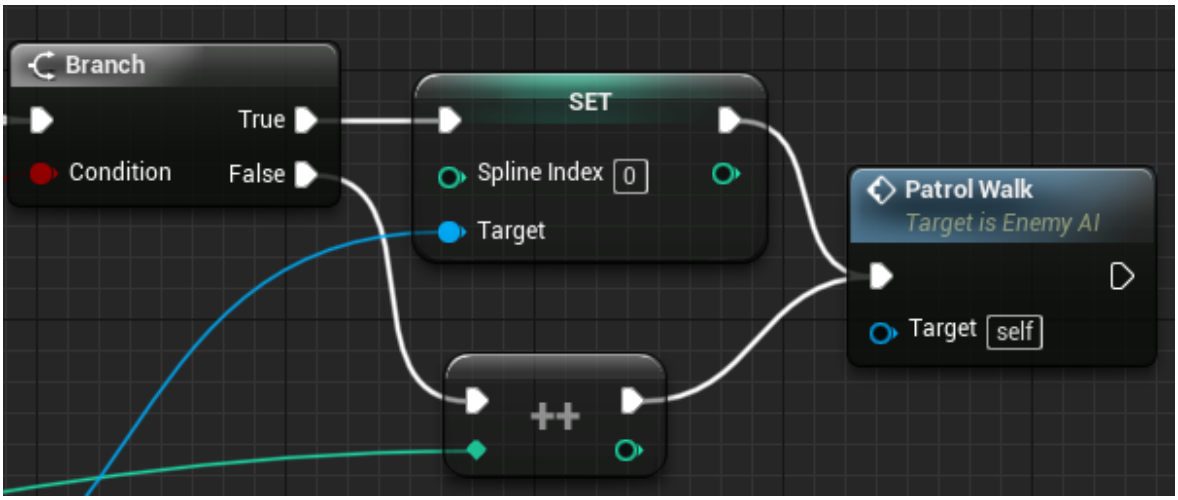


7. From the **AI MoveTo** node's **On Success** pin, add a **Branch** node. Connect the **AI MoveTo** node's **On Fail** pin to the **Branch** node's output execution pin as well.
8. From the **Branch** node's **Condition** pin, add an **Equal** (integer) node.
9. Connect the **Equal** node's top green input pin to the **Get Spline Index** node's output pin.

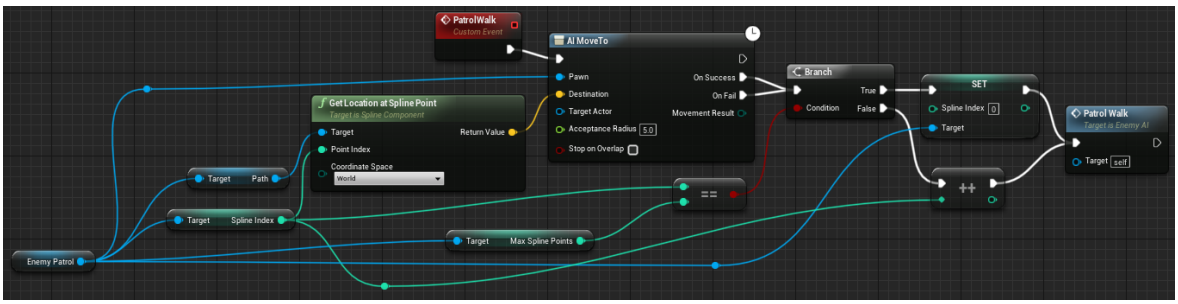
10. For the fourth time, drag a wire from the **EnemyPatrol** node, and now add a **Get Max Spline Points** node. Connect the **Get Max Spline Points** node's output pin to the **Equal** node's bottom green input pin.



11. For the last time, drag a wire from the **EnemyPatrol** node. Take it all the way to the other side of the **Branch** node and add a **Set Spline Index** node. Keep the **Spline Index** property set to "0". Connect the **Set Spline Index** node's input execution pin to the **Branch** node's **True** pin.
12. From the **Branch** node's **False** pin, add an **Increment Integer** node.
13. Drag a wire from the **Get Spline Index** node's output pin and connect it to the diamond-shaped pin on the **Increment Integer** node.
14. From the **Increment Integer** node's output execution pin, add a **PatrolWalk** node. Connect the **Set Spline Index** node's output execution pin to the **PatrolWalk node's** input execution pin, too.



Here is what the finished event should look like:



15. Finally, return to the node where you promoted **EnemyPatrol** to a variable. Off that node's output execution pin, add a **PatrolWalk** node.

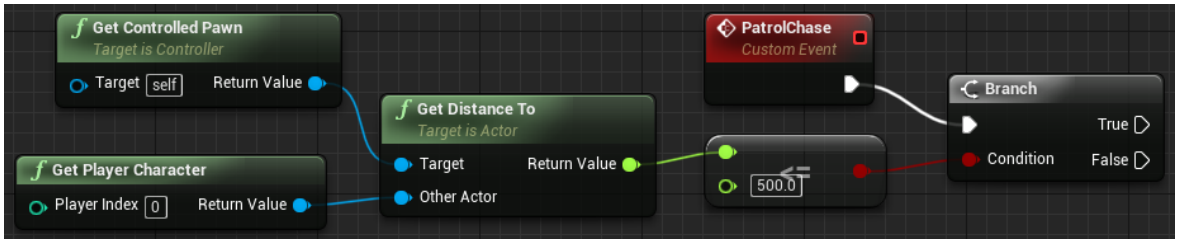
16. Compile the Blueprint.

Now that we've added the behavior, we just need to create the pathway.

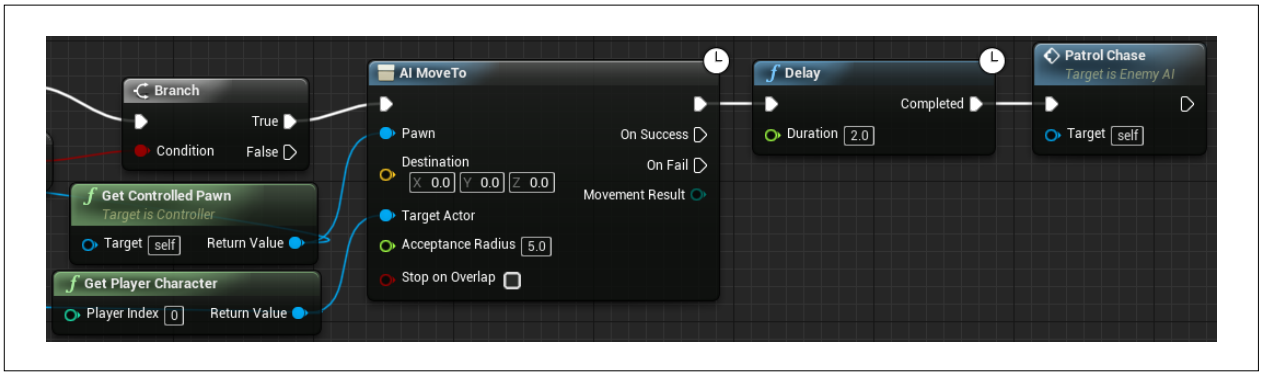
1. Add an **EnemyPatrol** Blueprint to your Level. If you look closely, you'll see a white line sticking out of the enemy; this is the spline. With **EnemyPatrol** selected, click on the white box at the front of the line. This is the spline point. You can place the spline point wherever you want to.
2. With the spline point selected, use the **Move** tool to begin creating a path.
3. To add an additional spline point, hold down the **Alt** key while using the **Move** tool.
4. You should notice that your spline loops back to the beginning. If it doesn't, you missed the step above about checking the **Closed Loop** box.
5. Finish creating a path you are happy with, and then test your Level to watch the enemy run the path! If the enemy is not following the path, try moving the enemy or the spline path to ensure that the enemy isn't being blocked by anything.

Now on to the third enemy, **EnemyPatrolChase**! This enemy acts just like **EnemyPatrol**, but if the player gets too close, **EnemyPatrolChase** will start chasing the player and run back if it is too far from the player. If you have not figured it out, the AI logic involved will be a combination of the logic implemented for the first two enemies.

1. Create a custom event called "**PatrolChase**" and add it to the graph.
2. From the **PatrolChase** event, add a **Branch** node.
3. First, we'll handle the **Branch** node's condition. Add a **Get Controlled Pawn** node and a **Get Player Character** node.
4. Now add a **Get Distance To** node. Connect the **Get Controlled Pawn** node's **Return Value** pin to the **Get Distance To** node's **Target** pin.
5. Next, connect the **Get Player Character** node's **Return Value** pin to the **Get Distance To** node's **Other Actor** pin.
6. Off the **Get Distance To** node's **Return Value** pin, add a **float <= float** (less than or equal to) node.
7. Set the value of the **float <= float** node's bottom input pin to "**500**". You can playtest and make this number larger or smaller depending on the experience you want.
8. Connect the **float <= float** node's red output pin to the **Branch** node's **Condition** pin.

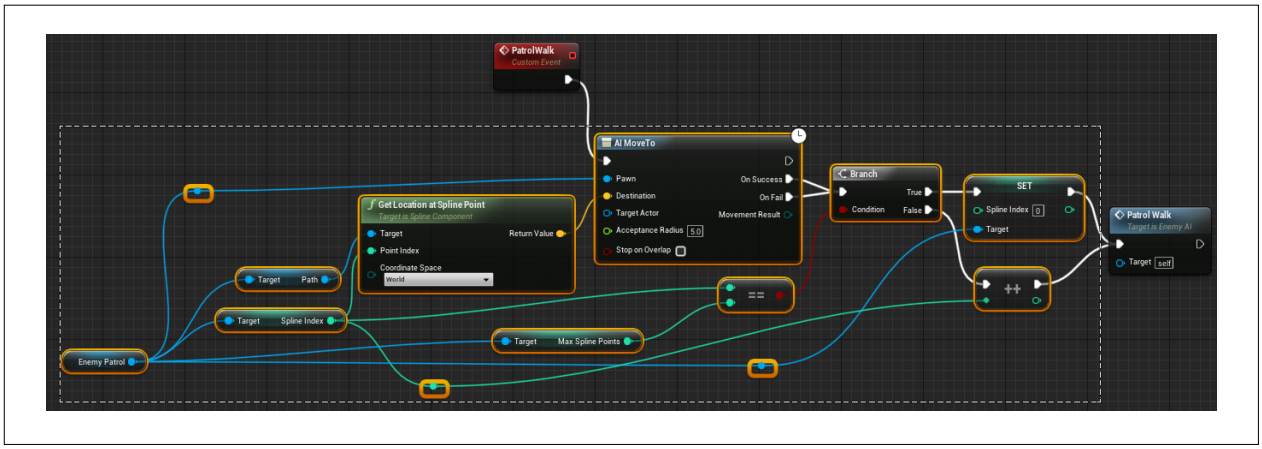


9. Off the **Branch** node's **True** pin, add an **AI MoveTo** node.
10. Connect the **Get Controlled Pawn** node's **Return Value** pin to the **AI MoveTo** node's **Pawn** pin and connect the **Get Player Character** node's **Return Value** pin to the **AI MoveTo** node's **Target Actor** pin.
11. From the **AI MoveTo** node's output execution pin, add a **Delay** node. Set the **Delay** node's **Duration** property to "**2.0**".
12. Off the Delay node's Completed pin, add a PatrolChase node.

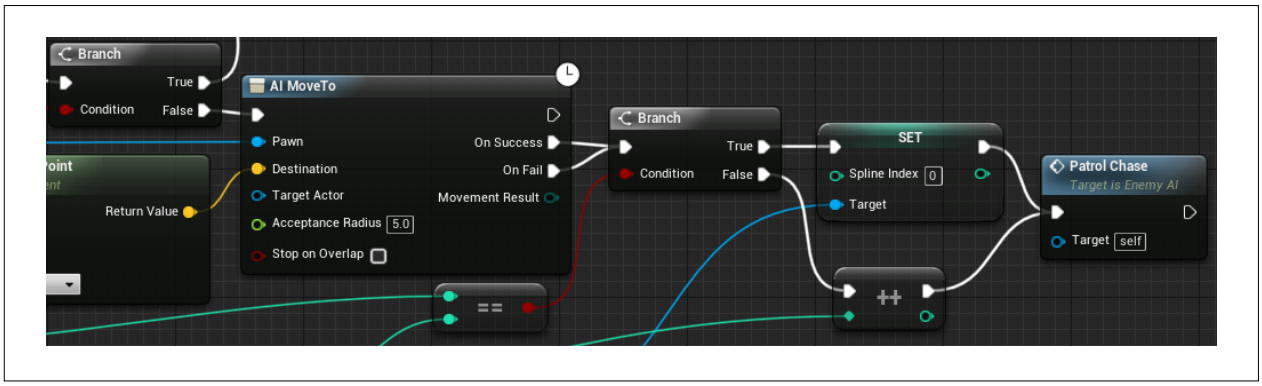


The patrolling part of the AI behavior is almost identical to what we've already created. Rather than rewrite all of it, we should utilize copy and paste.

13. Find the **PatrolWalk** event. Left-click and drag the mouse to select every single node in the **PatrolWalk** event ***except for*** the **PatrolWalk** event node at the beginning and the **PatrolWalk** function node at the end. Then right-click and select "**Copy**" or use the keyboard shortcut **(Ctrl + C)**.

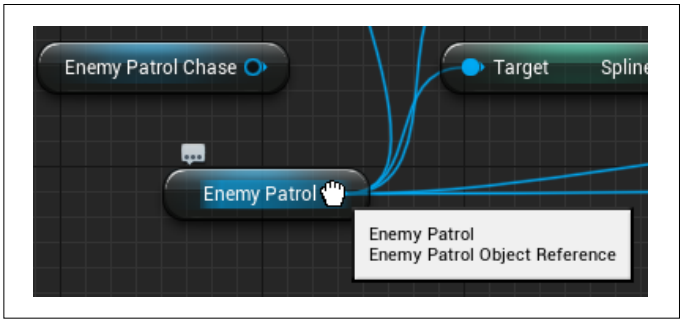


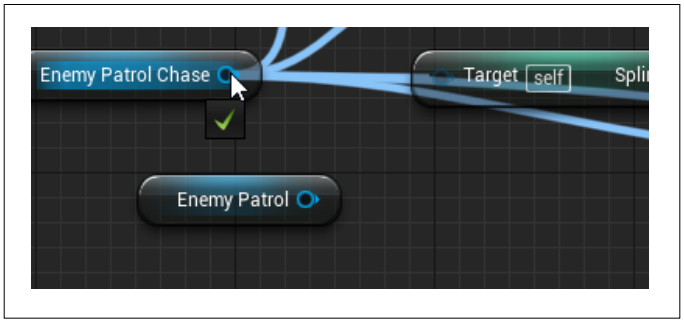
14. Place the copied nodes near the **Branch** node connected to the **PatrolChase** event node. Paste the nodes by right-clicking and selecting "**Paste**" or use the keyboard shortcut **(Ctrl + V)**.
15. Connect the input execution pin on the copied and pasted **AI MoveTo** node to the **Branch** node's **False** pin. At the end of the event, add a **PatrolChase** node and make sure it is connected to the **Increment Integer** node and the **Set Spline Index** node.



There's one last issue, though. All of the references are still to **EnemyPatrol**, not to **EnemyPatrolChase**! This is incredibly frustrating, because, since the **EnemyPatrol** reference is connected to so many other nodes, if we delete it the odds are high that we will forget to reconnect something. Thankfully there is an easy way to switch the reference.

16. Drag and drop a reference to **EnemyPatrolChase** near the **EnemyPatrol** reference.
17. Hold down the **Ctrl** key, and then click and hold down the mouse button on the **EnemyPatrol** reference pin. Drag the wires over to the **EnemyPatrolChase** reference pin and drop them there. Everything should easily move over.





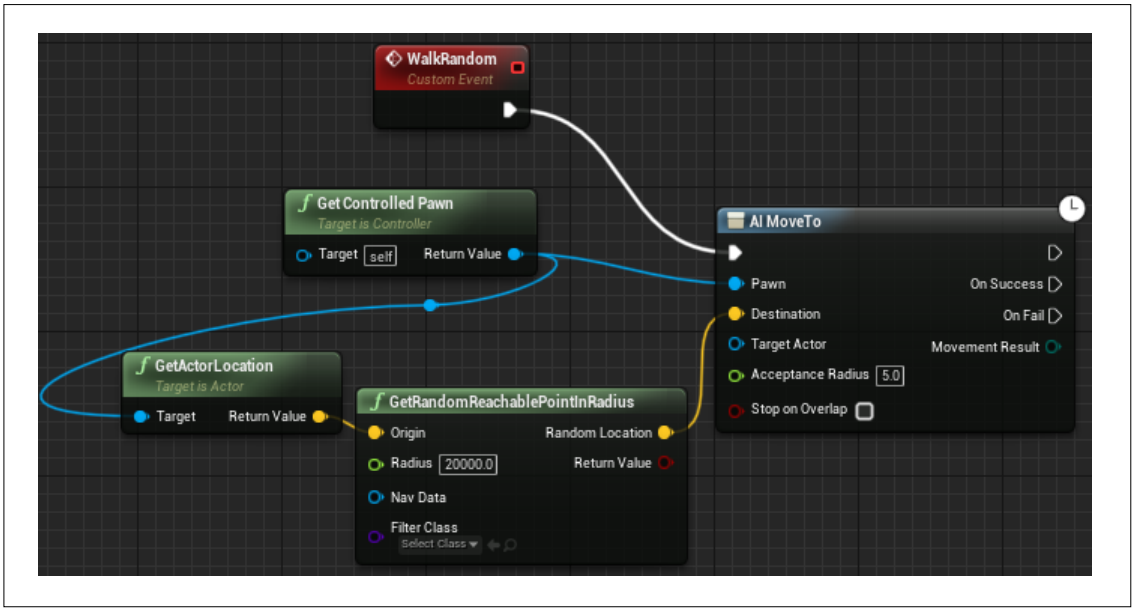
18. Delete the reference to **EnemyPatrol**.
19. Finally, return to the node where you promoted **EnemyPatrolChase** to a variable. Off that node's output execution pin, add a **PatrolChase** node.
20. Compile the Blueprint.

21. Head over to your Viewport, add an **EnemyPatrolChase** Blueprint to your Level, and give this enemy a path to walk. Run up to it and test its ability to chase you and return back to the path when you run too far away.

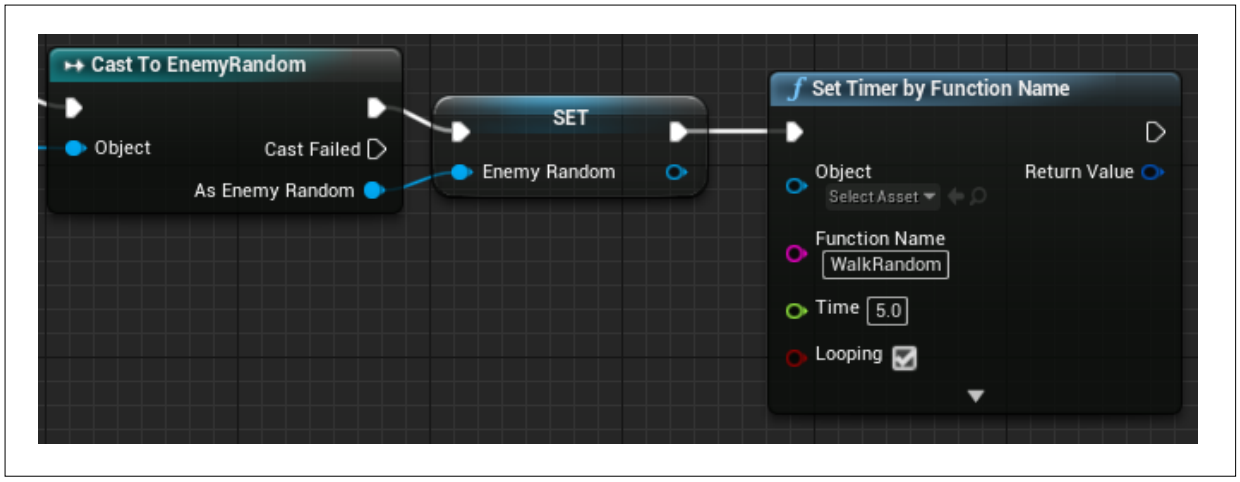
We're up to the last enemy! This enemy will just pick a spot it can get to and randomly run there.

1. Create a custom event called "**WalkRandom**" and add it to the graph.
2. From the **WalkRandom** node, as you might expect, add an **AI MoveTo** node.
3. Add a **Get Controlled Pawn** node and connect its **Return Value** pin to the **AI MoveTo** node's **Pawn** pin.
4. Off the **Get Controlled Pawn** node's **Return Value** pin, now add a **GetActorLocation** node.
5. From the **GetActorLocation** node's Return Value pin, add a **GetRandomReachablePointInRadius** node.

6. Set the **GetRandomReachablePointInRadius** node's **Radius** property to "**20000.00**". Connect its **Random Location** pin to the **AI MoveTo** node's **Destination** pin.



7. Finally, return to the node where you promoted **EnemyRandom** to a variable. Off that node's execution pin, we'll do something a little different. Add a **Set Timer by Function Name** node. Set the node's Function Name property to "**WalkRandom**", set **Time** to "**5.0**", and check the **Looping** box.
8. Compile the Blueprint.



9. Add an **EnemyRandom Blueprint** to your game and watch this enemy pick a random place, run there, and then keep doing it! If you want the enemy to run shorter amounts, lower the **GetRandomReachablePointInRadius** node's **Radius** value.

You now have four enemy types to use for your game! Place them in your Level to add to the challenge. However, you should be careful. AI is a resource-intensive process. This means that if you add too many enemies at once to your Level, the game may suffer from a lower frame rate, or just outright crash. As part of the next workshop, you will build an enemy spawner so you can save resources by not having all enemies in the game at once!

Off-Roading and Discussion

For this workshop's Off-Roading and Discussion section, you have two options to choose from: Behavior Trees and Splines. Both options are complex tools within Unreal Engine, and it will be difficult for you to successfully create something. If you can't, that's okay! This section is meant to push your understanding of Unreal, try new things, and break things.

Behavior Trees
Behavior Trees provide a way of building more complex AI in Unreal Engine. A Behavior Tree ranks actions that the AI character should do, and the AI constantly moves through the tree determining what it should do next. While Behavior Trees might be a hard topic to understand, learning how to use them will enable you to make much more intelligent AI in your game. Your group should all go through the "Behavior Tree Quick Start Guide" together. Everyone can follow along on their own project and talk through it together: https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreeQuickStart/index.html. Because the topic is complex, working on the quick start guide together will allow teammates to ask questions and get help more easily.
After your team has gone through the guide, discuss with your fellow teammates what kinds of AI you want to create! Talk about AI that would make your Grand Prix project better and talk through how you might build it. While you are discussing Behavior Trees and AI for your project, use the 3 **D**s to guide the discussion:

- **Difficulties**: What was the most difficult or confusing part of this week's workshop?
- **Discoveries**: What were some interesting, fascinating, or exciting things you discovered while building the AI in Blueprints and in the Behavior Tree?
- **Dreams**: What kind of AI do you want to build? How could you do it?

Splines
Splines are an incredibly versatile tool in Unreal Engine. Using splines to create paths for AI to walk is just one use. As a team, everyone should download the **Content Examples** project. You can find the project in the Epic Games Launcher by going to **Unreal Engine → Learn → Engine Feature Samples**. Note: The project is approximately 4GB and will take a while to open the first time.

Once the project opens, find and open the **Blueprint Splines** Level in the **Maps** folder. As a team, explore the Level and look at the various Blueprints both in-game and in-Editor. Pick one of the exhibits and try creating a version of it as a team. Look at the Event Graph of the Blueprint and talk through what each part does and why it is there.
After you've created a Blueprint Spline as a team, have a greater discussion about the splines, the AI, and where you can use what you learned. Use the 3 **D**s to guide your discussion:

- **Difficulties**: What was the most difficult or confusing part of this week's workshop?
- **Discoveries**: What were some interesting, fascinating, or exciting things you discovered while building the splines in the Grand Prix and Off-Roading and Discussion sections?
- **Dreams**: What kind of features could you make using Blueprint Splines that would work well in the game you have been building? How could you make them?

CONGRATULATIONS!

You've finished Workshop Four of the Fast Track!

MechWarrior 5: Mercenaries | Piranha Games