Course - Lập trình AI cho Unreal Engine

AI của chúng ta có giác quan như thế nào?

Giải thích cách sử dụng các thành phần khác nhau có sẵn trong Unreal Engine 4 để cho phép AI của chúng ta cảm nhận được các AI khác và những con tốt mà chúng ta sẽ đặt trong thế giới.

Tags: AI của chúng ta có giác quan như thế nào?

Trong chương này, bạn sẽ học cách sử dụng các component khác nhau có sẵn trong Unreal Engine 4 để cho phép AI của chúng ta cảm nhận được AI khác và những con tốt mà chúng ta đặt trong thế giới. Chúng ta sẽ làm điều này bằng cách tận dụng một hệ thống trong Unreal Engine được gọi là các component AI Perception. Các thành phần này có thể được tùy chỉnh và thậm chí là viết kịch bản để giới thiệu hành vi mới bằng cách mở rộng giao diện cảm biến hiện tại.

Các chủ đề chúng ta sẽ đề cập trong chương này như sau:

- Các AI components
- Đăng ký một actor có thể nhận biết bằng AIPerceptionStimuliSource
- Nhận thức các đối tượng bằng AI Perception
- Khái niệm state machine

Bạn có thể download toàn bộ source dự án, tại đây.

Tổng quan

AI Perception là một hệ thống trong Unreal Engine 4 cho phép các nguồn đăng ký các giác quan của chúng để tạo ra các kích thích và sau đó những người nghe khác được cập nhật định kỳ khi các kích thích giác quan được tạo ra trong hệ thống. Điều này thật tuyệt vời khi tạo ra một hệ thống có thể tái sử dụng, có thể phản ứng với một loạt các cảm biến có thể tùy chỉnh. Vì vậy, chương này sẽ tập trung vào việc sử dụng component AI Perception để AI của kẻ thù đuổi theo chúng ta bất cứ khi nào chúng ta bị phát hiện. Những gì chúng ta sẽ làm khác đi là tất cả những thứ này sẽ được viết script dựa vào Blueprint. Vì vậy, không cần sử dụng Behavior Tree vào khoảng thời gian này!

Làm việc với AI Sense

Hãy bắt đầu bằng cách mở Unreal Engine 4 và mở cửa sổ **New Project** của chúng ta lên. Sau đó, thực hiện các bước sau:

1. Đầu tiên, chọn mẫu dự án là **Third Person,** đặt tên cho nó là *AISense* và nhấn nút **Create Project**. Sau khi tải xong, chúng ta sẽ bắt đầu bằng cách tạo một **AIController** mới sẽ chịu trách nhiệm gửi cho AI của chúng ta các hướng dẫn phù hợp.
2. Hãy đi đến thư mục *Blueprint* và tạo một **AIController** mới từ context-menu **Blueprint Class**, và đặt tên cho class mới của chúng ta là *EnemyPatrol*.
3. Làm thế nào để gán *EnemyPatrol*, chúng ta cần đặt một con tốt vào màn chơi sau đó gán AIController cho nó.
4. Sau khi đặt quân tốt vào màn chơi, hãy nhấp vào panel **Details** trong **Level Editor**. Tiếp theo, chúng ta sẽ tìm kiếm AIController. Theo mặc định, nó là class mẹ AIController,

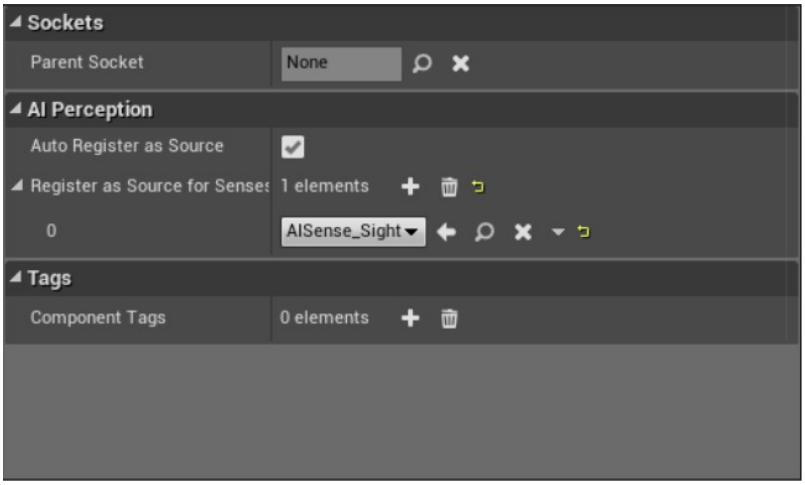
nhưng chúng ta muốn ở đây là *EnemyPatrol*:

5. Tiếp theo, chúng ta sẽ tạo một Blueprint Class khác cho **PlayerController** mới có tên *PlayerSense*.
6. Sau đó, chúng ta cần phải giới thiệu component AI Perception cho những người mà chúng ta muốn được nhìn thấy hoặc để xem. Trước tiên, hãy mở controller *PlayerSense* và sau đó thêm các thành phần cần thiết.

Các component AI Perception

Có hai component hiện đang có sẵn. Đầu tiên là thứ bạn đã quen thuộc: component AI Perception. Cái còn lại là thành phần **AIPerception Stimuli Source**. Cái sau được sử dụng để dễ dàng đăng ký quân tốt như một nguồn kích thích, cho phép nó được phát hiện bởi các component AI Perception khác. Điều này có ích, đặc biệt là trong trường hợp của chúng ta. Bây giờ, hãy làm theo các bước sau:

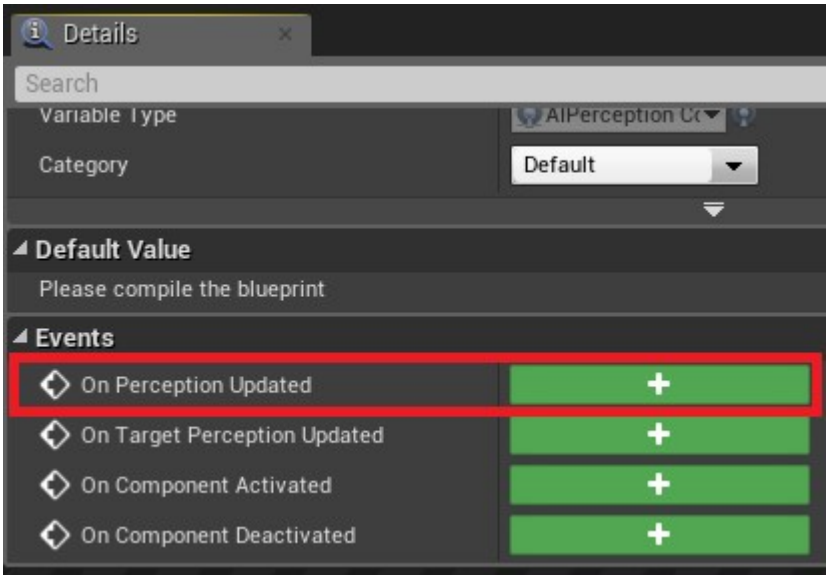
1. Khi *PlayerSense* được mở lên, hãy thêm một component mới gọi là **AIPerception Stimuli Source**. Sau đó, trong panel **Details**, hãy bật tùy chọn **Auto Register as Source** dưới phần AIPerception.
2. Tiếp theo, chúng ta sẽ thêm các giác quan mới để tạo nguồn cho nó. Vì vậy, hãy chú ý vào **Register as Source for Senses** ngay bên dưới tùy chọn mình vừa bật, nó có một mảng **AISense**.
3. Điền mảng này với blueprint **AISense_Sight** để các component AI Perception khác có thể phát hiện bằng mắt. Bạn sẽ lưu ý rằng cũng có các giác quan khác để chọn— ví dụ: **AISense_Hearing**, **AISense_Touch**, v.v.
   Các cài đặt hoàn chỉnh được hiển thị trong ảnh chụp màn hình sau:



Điều này khá đơn giản khi xem xét quy trình tiếp theo của chúng ta. Nó cho phép quân tốt của người chơi của chúng ta bị AI của kẻ thù phát hiện bất cứ khi nào chúng ta ở trong phạm vi được định cấu hình của giác quan của chúng.

Tiếp theo, hãy mở class *EnemyPatrol* của chúng ta và thêm các component AI Perception khác vào AI của chúng ta. Component này được gọi là **AIPerception** và chứa nhiều cấu hình khác, cho phép bạn tùy chỉnh và điều chỉnh AI cho các tình huống khác nhau:

1. Nhấp vào component **AI Perception**, bạn sẽ nhận thấy rằng trong phần AI, mọi thứ đều chuyển sang màu xám. Nó là do chúng ta có cấu hình cụ thể cho từng giác quan. Nó cũng xảy ra nếu bạn tạo các class AI Sense của riêng mình.
2. Hãy tập trung vào hai phần trong component này: phần đầu tiên là cài đặt **AI Perception** và phần còn lại là event được cung cấp cùng với component này:
    1. Phần **AI Perception** sẽ giống với phần tương tự trên **AIPerceptionStimuliSource**. Sự khác biệt là bạn phải đăng ký các giác quan của mình và bạn cũng có thể chỉ định một giác quan chủ đạo. **Dominant Sense** được ưu tiên hơn các giác quan khác được xác định ở cùng một vị trí.
    2. Hãy chú ý vào **Senses Config** và thêm một yếu tố mới. Thao tác này sẽ điền vào mảng một cấu hình cảm quan mới, sau đó bạn có thể sửa đổi cấu hình này.
3. Hiện tại, hãy chọn cấu hình **AI Sight**, sau đó chúng ta có thể giữ nguyên các giá trị mặc định. Trong trò chơi, chúng ta có thể trực quan hóa các cấu hình, cho phép chúng ta kiểm soát nhiều hơn các giác quan của mình.
4. Có một cấu hình khác cho phép bạn chỉ định liên kết, nhưng tại thời điểm viết bài này, các tùy chọn này không khả dụng.
5. Trên cấu hình sense bạn vừa tạo ra hãy mở mục **Sense** ra, bạn sẽ tìm thấy và nhấp vào **Detection By Affiliation**, bạn cần phải chọn **Detect Neutrals** để phát hiện bất kỳ quân tốt nào có **Sight Sense Source**.
6. Tiếp theo, chúng ta cần phải thông báo cho AI của mình về một mục tiêu mới. Chúng ta sẽ làm điều này bằng cách sử dụng Event mà chúng ta đã thấy qua như một thành phần của component **AI Perception**. Bằng cách đi đến mục **Events** trong **Details**, chúng ta có thể thấy một event có tên là **OnPerceptionUpdated**.



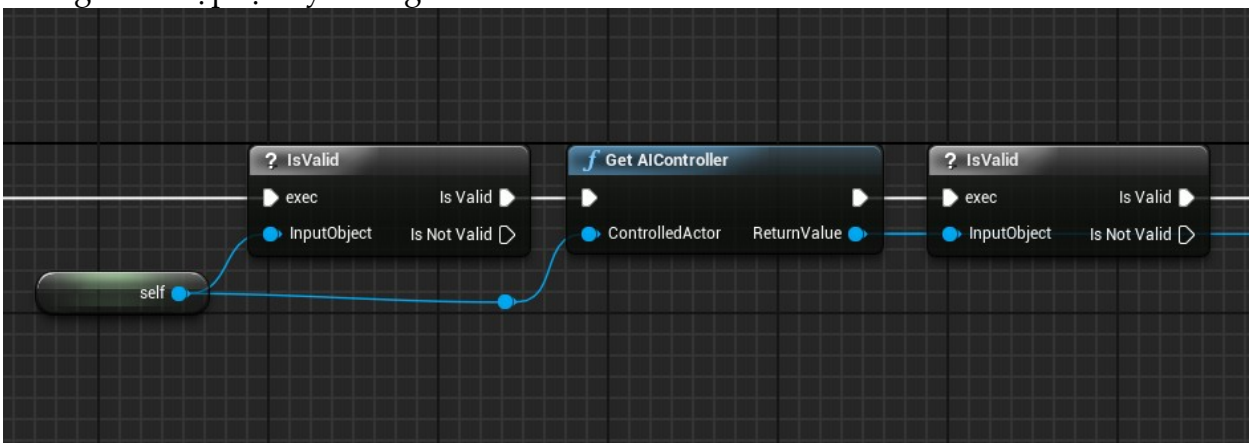
Nó sẽ được cập nhật khi có những thay đổi về trạng thái giác quan, giúp cho việc theo dõi các giác quan trở nên dễ dàng và đơn giản. Hãy chuyển sang event **OnPerceptionUpdated** và thực hiện như sau:

1. Nhấp vào **OnPerceptionUpdated** và tạo nó trong sơ đồ EventGraph. Bây giờ, trong EventGraph, bất cứ khi nào sự kiện này được gọi, các thay đổi sẽ được thực hiện đối với các giác quan và nó sẽ trả về các actor được cảm nhận, như thể hiện trong ảnh chụp màn hình sau:

   Bây giờ chúng ta đã hiểu cách chúng ta sẽ có được các actor được cảm nhận được tham chiếu của mình, chúng ta nên cho ra một cách thức để quân tốt của chúng ta duy trì các trạng thái khác nhau tương tự như những gì chúng ta sẽ làm trong Behavior Tree.
2. Trước tiên, hãy thiết lập một vị trí nhà để quân tốt của chúng ta chạy đến khi người chơi không còn bị AI phát hiện nữa. Trong cùng một thư mục *Blueprint*, chúng ta sẽ tạo một subclass của **Target Point**. Hãy đặt tên cho nó là *Waypoint* và đặt nó ở một vị trí thích hợp trong màn chơi.
3. Bây giờ, chúng ta cần mở subclass *Waypoint* này và tạo các biến bổ sung để bảo trì các route có thể đi qua. Chúng ta có thể làm điều này bằng cách xác định waypoint kế tiếp một waypoint, cho phép chúng ta tạo cái mà các lập trình viên gọi là danh sách được liên kết (Linked List). Điều này dẫn đến việc AI có thể liên tục di chuyển đến tuyến

đường có sẵn tiếp theo sau khi đến đích của tuyến đường hiện tại.

4. Khi editor của class *Waypoint* mở, hãy thêm một biến mới có tên *NextWaypoint* và đặt loại biến này theo kiểu class *Waypoint* mà chúng ta đã tạo.
5. Quay trở lại *Content Browser* của chúng ta.
6. Bây giờ, trong AIController *EnemyPatrol* của chúng ta, hãy tập trung vào **Event BeginPlay** trong sơ đồ. Chúng ta phải lấy tham chiếu đến waypoint mà chúng ta đã tạo trước đó và lưu trữ nó trong AIController của chúng ta.
7. Vì vậy, hãy tạo một biến loại **Waypoint** mới và đặt tên là *CurrentPoint*.
8. Bây giờ, trên **Event BeginPlay**, thứ đầu tiên chúng ta cần là AIController, nó sẽ tự tham chiếu vì chúng ta đang ở trong chính class **AIController**.
9. Vì vậy, hãy lấy bản tự tham chiếu của chúng ta, Hãy kiểm tra xem nó có hợp lệ không. An toàn là trên hết! Tiếp theo, chúng ta sẽ lấy AIController từ tài liệu tham khảo của mình. Sau đó, một lần nữa để đảm bảo an toàn, hãy kiểm tra xem AIController của chúng ta có hợp lệ hay không.



10. Tiếp theo, chúng ta sẽ tạo node **Get all Actors Of Class** và đặt class **Actor** thành *Waypoint*.
11. Bây giờ, chúng ta cần chuyển đổi một số hướng dẫn thành macro vì chúng tôi sẽ sử dụng các hướng dẫn trong suốt dự án. Vì vậy, hãy chọn các nút được hiển thị như trên và nhấn chuột phải chọn **Collapse to macro**. Cuối cùng, đổi tên macro này là *getAIController*. Bạn có thể thấy các nút cuối cùng trong ảnh chụp màn hình sau:

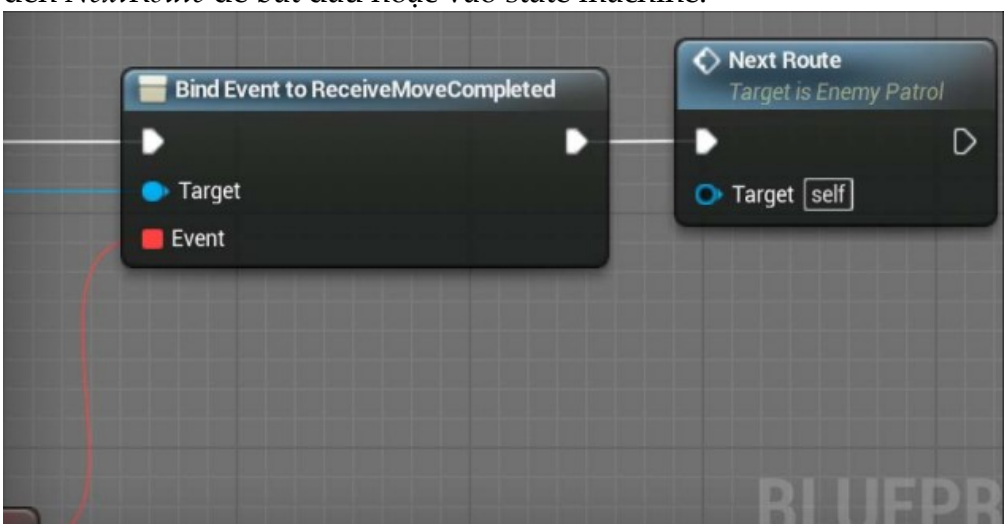
12. Tiếp theo, chúng ta muốn AI của mình lấy một tuyến đường mới ngẫu nhiên và đặt nó làm một biến mới. Vì vậy, trước tiên hãy lấy độ dài **Length** của mảng actor được trả về. Sau đó, chúng ta sẽ trừ **Subtract** 1 từ độ dài này và nó sẽ cho chúng ta chỉ số của phần tử cuối cùng của mảng của chúng ta.
13. Từ đó, chúng ta sẽ lấy từ **Subtract** cho vào node **Random Integer**. Sau đó, từ mảng của chúng ta, chúng ta sẽ lấy node **GET** và bơm node **Random Integer** của chúng ta vào chỉ mục để truy xuất mảng.
14. Tiếp theo, kéo biến có sẵn được trả về từ node **GET** và cắm vào biến *CurrentPoint* đã tạo từ trước.
15. Sau đó, từ macro *getAIController* của chúng ta, chúng ta sẽ chỉ định sự kiện **ReceiveMoveCompleted**. Nó được thực hiện để khi AI của chúng ta di chuyển thành công sang tuyến đường tiếp theo, chúng ta có thể cập nhật thông tin và báo cho AI của chúng ta chuyển sang tuyến đường tiếp theo.

State machines

Theo truyền thống, chúng ta sẽ mặc định sử dụng AI Behavior Tree, có sẵn cho bất kỳ ai sử dụng UE4. Tuy nhiên, trong kịch bản của chúng ta, chúng ta sẽ chia công việc của Behavior Tree thành các component được viết một cách trực tiếp trong blueprint. Vì vậy, điều tiếp theo mà chúng ta cần tạo là một cách thức duy trì trạng thái. Sau đó, về cơ bản, chúng ta có thể tạo một state machine bằng cách cập nhật biến, cho phép AI của chúng ta chuyển sang các trạng thái khác nhau bằng cách kiểm soát luồng thực thi.

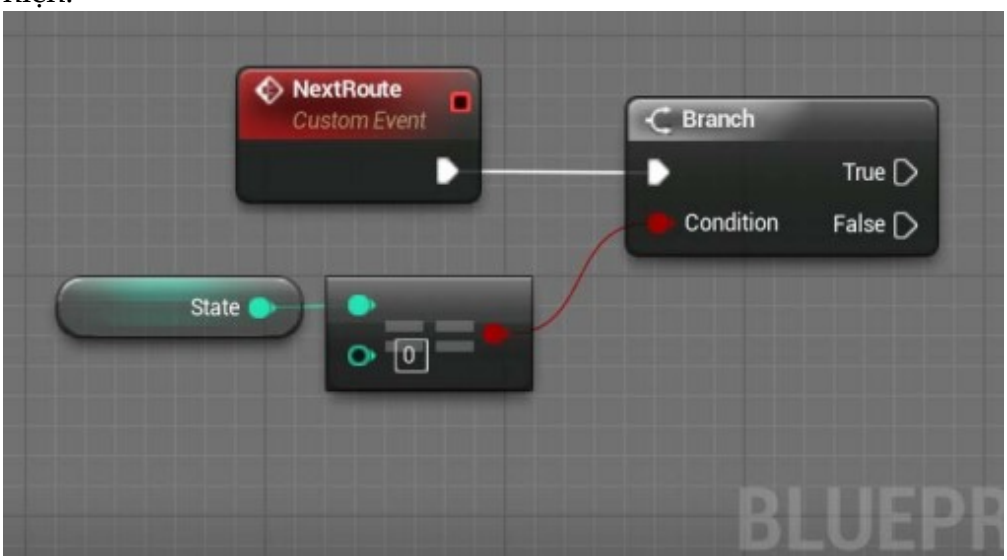
Chúng ta sẽ thiết lập điều này bằng cách sử dụng một event sẽ cập nhật trạng thái của chúng ta khi các điều kiện mà chúng ta chỉ định được đáp ứng. Để bắt đầu điều này, hãy tiếp tục bước tiếp theo!

1. Đầu tiên, hãy tạo một biến **Int** mới trong AIController của chúng ta có tên là **State**. Điều này sẽ duy trì trạng thái hiện tại của chúng ta trong state machine của chúng ta.
2. Bây giờ chúng ta cần một event mới và chúng ta sẽ gọi nó là *NextRoute*. Vì vậy, khi nhấp chuột phải vào sơ đồ EventGraph và vào phần **Add New**, chúng ta sẽ chú ý **Add New Event**. Hãy đặt tên cho event mới này là *NextRoute*.
3. Bây giờ, phía sau event **ReceiveMoveCompleted** đã được gán nhượng, chúng ta sẽ gọi đến *NextRoute* để bắt đầu hoặc vào state machine:



4. Tiếp theo, tập trung vào **ReceiveMoveCompleted**, hãy kéo chân **Result** để tạo node **Switch**. Từ node **Switch** vừa tạo, hãy kéo từ chân **Success** tạo ra node **Branch**. Điều này có nghĩa là chỉ khi AI của chúng ta hoàn thành việc di chuyển, thì chúng ta mới tiếp tục chuyển sang bước tiếp theo trong luồng thực thi.
5. Từ **Success**, hãy tạo một node **Branch** mới và thiết lập một điều kiện mới. Điều kiện này sẽ kiểm tra xem biến **State** của chúng ta hiện có bằng 0 hay không. Giá trị 0 đại diện cho trạng thái mặc định của chúng ta.
6. Nếu giá trị của biến **State** của chúng ta là 0, hãy tạo node **Delay Retriggerable** và đặt **Duration** tại **0,2** giây. Tiếp theo, chúng ta muốn kiểm tra xem biến **Current Point** của chúng ta có hợp lệ hay không; nếu vậy, thì chúng ta sẽ lấy biến mà chúng ta đã tạo trước đó để xác định tuyến đường tiếp theo trong danh sách được liên kết.
7. Từ đó, chúng tôi muốn thiết lập biến mới này thành biến **Current Point** của chúng ta. Điều này sẽ cho phép chúng ta điều hướng vô thời hạn giữa các waypoint.
8. Tiếp theo, chúng tôi muốn tạo hai biến mới sẽ chịu trách nhiệm trì hoãn một lượng thời gian cho đến khi chúng ta tiếp tục lộ trình tiếp theo.
9. Vì vậy, hãy tạo một biến **Float** mới có tên là **RoutePauseDelay**, biến này sẽ xác định thời gian chúng ta sẽ đợi. Tiếp theo, chúng ta muốn tạo độ lệch để thời gian chờ không phải lúc nào cũng giống nhau. Vì vậy, bây giờ hãy tạo một biến **Float** mới có tên là **RoutePauseDevia**.
10. Hãy bơm **RoutePauseDeviation** vào node **Random Float in Range** và thêm phần này vào **RoutePauseDelay**. Sau đó, nó sẽ được bơm vào một node **Delay**.
11. Từ node **Delay**, chúng ta sẽ gọi sự kiện *NextRoute* mà chúng tôi đã tạo trước đó.
12. Tập trung vào sự kiện *NextRoute* vừa tạo, chúng ta sẽ kiểm tra xem biến **State** của

chúng ta có bằng 0 hay không và tạo một **Branch** node để kiểm tra kết quả khi thực hiện:

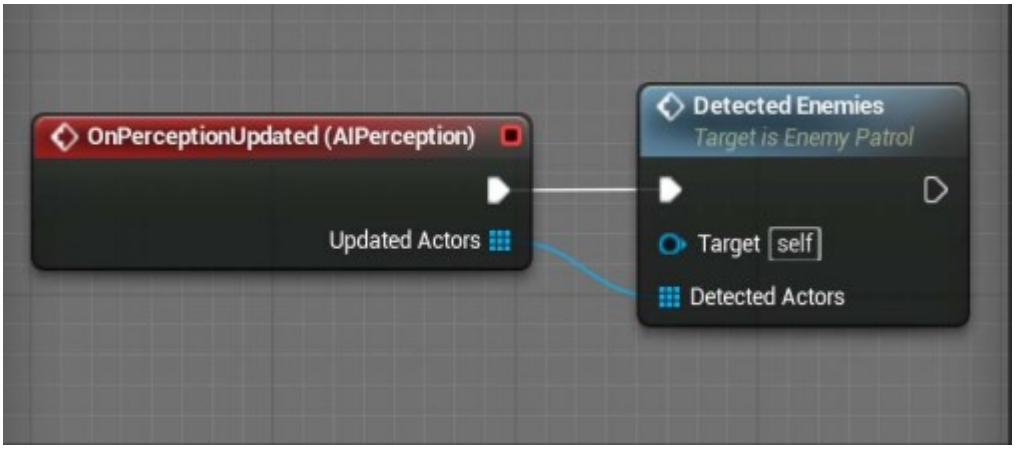


13. Tiếp theo, chúng ta sẽ lấy AIController từ Actor của chúng ta và kiểm tra xem biến *CurrentPoint* của bạn có hợp lệ hay không.
14. Tiếp theo, nếu biến *CurrentPoint* của chúng ta hợp lệ, thì chúng ta nên kéo từ Return Value của *getAIController* vào node **Move to Actor**.
15. Từ đó, kéo biến *CurrentPoint* mà chúng tôi đã kiểm tra và đặt biến này vào mục tiêu của node **Move to Actor**. Bây giờ, khi AIController chiếm hữu actor mà nó thuộc về, nó sẽ gọi sự kiện này. Sau khi di chuyển xong, nó sẽ cập nhật tuyến đường và tiếp tục đến tuyến đường tiếp theo.

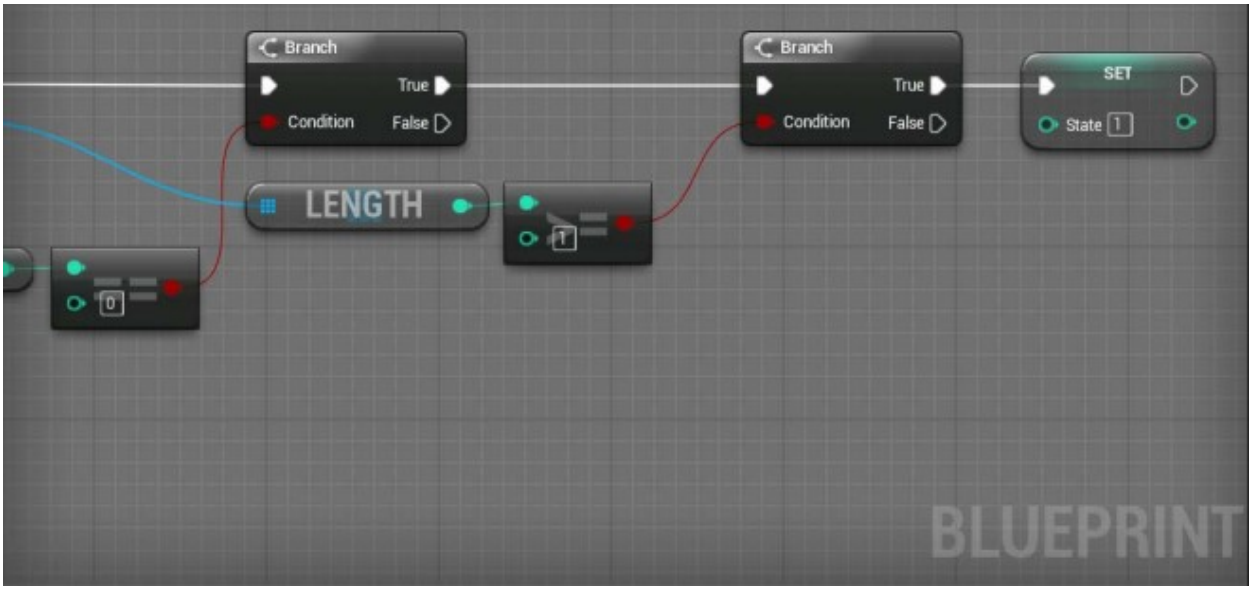
Pawn Detection

Để cung cấp cho con tốt của chúng ta thông tin cập nhật, chúng ta cần tạo một event sẽ nắm bắt và phản hồi các event cập nhật giác quan từ component AI Perception. Chúng ta sẽ thiết lập nó bằng cách tạo một sự kiện mới chỉ chịu trách nhiệm xử lý các con tốt được phát hiện. Dưới đây là các bước:

1. Hãy nhấp chuột phải vào bất kỳ đâu và vào đến mục **Add Event**. Từ đó, chúng ta sẽ thêm một event tự tạo bằng cách nhấp chuột vào mục **Add Custom Event** và chúng ta sẽ đặt tên cho sự kiện mới là *Detected Enemies*. Chúng ta cũng phải tạo một tham số mảng actor mới có tên là *Detected Actors* để giữ mảng **Updated Actors**.
2. Bây giờ, trong bước tiếp theo, chúng ta cần biên dịch lại một blueprint để gọi hàm AIController và *Detected Enemies* từ event node **OnPerceptionUpdated** của chúng ta:



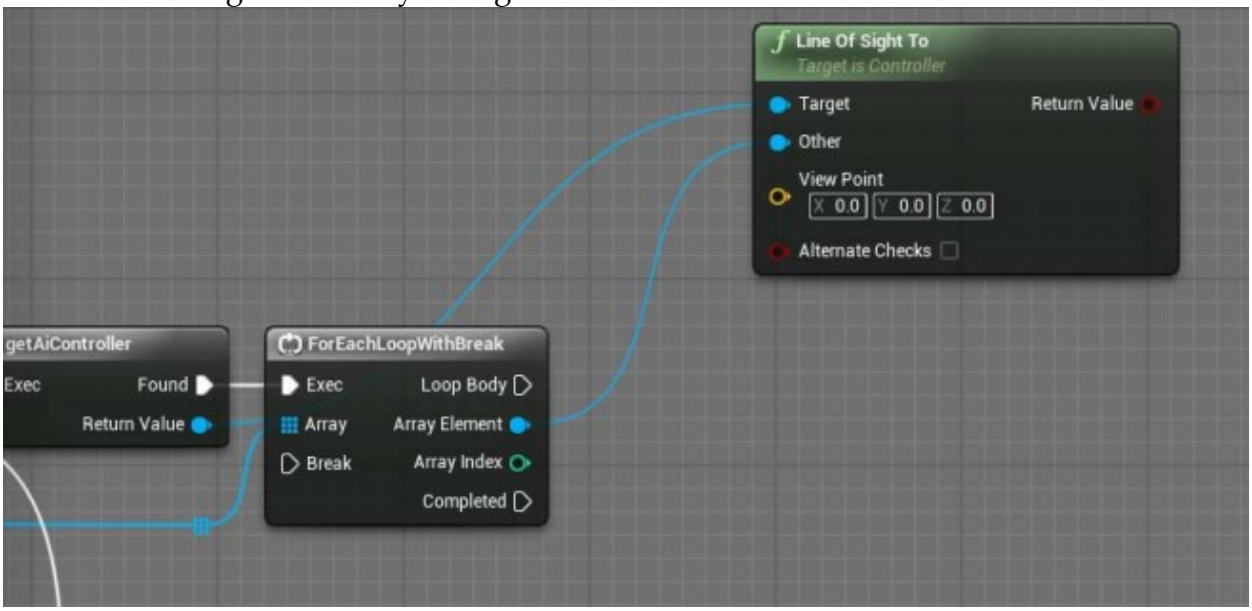
3. Trong các bước tiếp theo, chúng ta sẽ xử lý các chuyển đổi giữa các trạng thái khác nhau. Ví dụ, chúng ta có thể ở trạng thái mặc định và tìm một kẻ thù mới. Từ đó, chúng ta cần dừng chuyển động hiện tại và di chuyển về phía kẻ thù được phát hiện. Nếu chúng ta mất dấu kẻ thù mà chúng ta đã đuổi theo trong một khoảng thời gian cụ thể, chúng ta có thể hủy chuyển động và quay lại điều hướng đến các waypoint của mình.
4. Bây giờ, chúng ta sẽ tập trung lại vào event *Detected Enemies* mà chúng ta đã tạo trước đó. Giờ đây, event này được gọi bởi component AI Perception của chúng ta, chúng ta muốn sử dụng các actor này một cách thích hợp cho từng trạng thái.
5. Hãy lấy từ event này và tạo một node **Branch**. Tiếp theo, chúng ta muốn tạo điều kiện cho **Branch** này. Chúng ta cần kéo biến **State** của mình vào sơ đồ và kiểm tra xem nó có bằng **0** hay không. Nếu là **True**, chúng ta sẽ kiểm tra xem chúng ta có phát hiện ra kẻ thù nào không.
6. Vì vậy, kéo từ **True**; hãy kiểm tra xem độ dài của mảng *Detected Actors* của chúng ta có dài hơn hoặc bằng **1** hay không. Nếu đúng như vậy, chúng ta sẽ có kẻ thù và chúng ta có thể tiến lên trong quá trình này. Vì vậy, hãy kéo từ **True** của node **Branch** trước đó rồi tạo một **Branch** mới. Thao tác này sẽ kiểm tra độ dài của mảng **Detected Actor**.
7. Tiếp theo, nếu chúng ta tìm thấy địch thủ, chúng ta sẽ thay đổi trạng thái để thay đổi đường dẫn thực thi. Chúng ta có thể làm điều này bằng cách đặt biến **State** là **1**:



8. Tiếp theo, chúng tôi tạo một câu lệnh chuyển dịch mới cho biến **State** của chúng ta. Điều này sẽ giúp chúng ta có quy trình thực thi. Chúng ta sẽ bắt đầu với chỉ mục có khởi đầu từ **1** và không có chân mặc định.
9. Bây giờ, chúng ta có thể quay lại điều kiện **Branch** đầu tiên của mình, nơi chúng ta sẽ kiểm tra xem trạng thái có bằng **0** hay không. Sau đó, khi câu lệnh này là False — có nghĩa là chúng ta hiện đang ở trong trạng thái tìm kiếm địch thủ của mình — chúng ta cũng sẽ dẫn ra câu lệnh **Switch on Int** từ đây. Nó cho phép trạng thái **State** của chúng ta chuyển ra khỏi trạng thái nếu chúng ta mất dấu kẻ thù vào một thời điểm nào đó.
10. Tập trung vào node **Switch on Int**, hãy thêm một dấu ngắt khác cho chỉ mục **2**. Sau đó, kéo từ chân giá trí **1**, và gọi đến **macro** *getAiController*. Nếu **Found** được thực thi, thì chúng ta đã sẵn sàng để tiếp tục.
11. Hãy lấy tham số **Detected Actors** từ event của chúng ta và tạo một node **ForEachLoopWithBreak**. Từ node này, hãy kiểm tra xem kẻ thù được phát hiện có nằm trong tầm nhìn hay không. Nhấp chuột phải và tìm kiếm node **Light of Sight To**, giữ nguyên tham số **Target**. Sau đó, cho cắm vào chân **Other** với biến **ArrayElement**.
12. Kéo từ chân **Return Value**, chúng ta sẽ tạo một node **Branch** mới. Điều kiện **Branch** này sẽ cần kiểm tra xem chúng ta có địch thủ hiện tại hay không; nếu không, chúng ta sẽ thiết lập một địch thủ mới. Từ giá trị **True** của node **Branch**, hãy tạo biến kiểu **Actor** và đặt tên cho nó là *Enemy Actor*, sau đó đặt một địch thủ vào *Enemy Actor* từ **Array Element**.
13. Từ *getAiController*, chúng ta muốn tập trung vào địch thủ mà chúng ta vừa đặt trong *Enemy Actor*. Hãy kéo tiếp ra node **Set Focus**, đưa AIController vào tham số **Target**. Còn tham số **new Focus**, chúng ta sẽ cắm *Enemy Actor* đã phát hiện vào.
14. Tiếp theo, chúng ta sẽ lấy lại AIController của chúng ta và tạo node **Move To Actor**. Sau đó, chúng ta sẽ đặt **Enemy Actor** vào chân **Goal** của chúng ta cho node **Move to Actor**. Nó sẽ điều hướng AI của chúng ta đến kẻ thù mới được tìm thấy của chúng ta.
15. Tiếp theo, hãy bật **Return Value** từ node **Move To Actor**. Chúng ta sẽ nhận được một node sẽ chuyển dịch giữa **Failed**, **Already At Goal** và **Request Successful**. Hãy kéo từ

Request Successful và đặt biến **State** của chúng ta thành giá trị **2**.

16. Từ biến **Set State** của chúng ta, sau đó chúng ta muốn ngắt node **ForEachLoopWithBreak** của mình. Bằng cách này, chúng ta sẽ không tiếp tục tìm kiếm kẻ thù khi chúng tôi tìm thấy chúng:

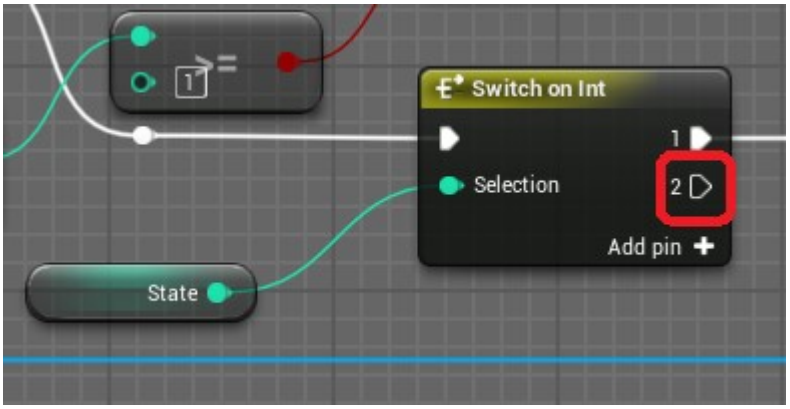


17. Hãy quay lại vòng lặp của chúng ta tại node **Branch** đầu tiên của chúng ta và từ chân **False**, chúng ta sẽ tạo một **Branch** khác. Nó sẽ kiểm tra xem chân **Array Element** đó chính là địch thủ mà chúng ta vừa chọn thất bại trong node **Line Of Sight To**, có phải là kẻ thù mà chúng ta hiện đang nhắm mục tiêu hay không. Nếu vậy, chúng ta sẽ xóa hết biến *Enemy Actor*, cũng như sự tập trung và đặt giá trị **State** của chúng ta về **0**.

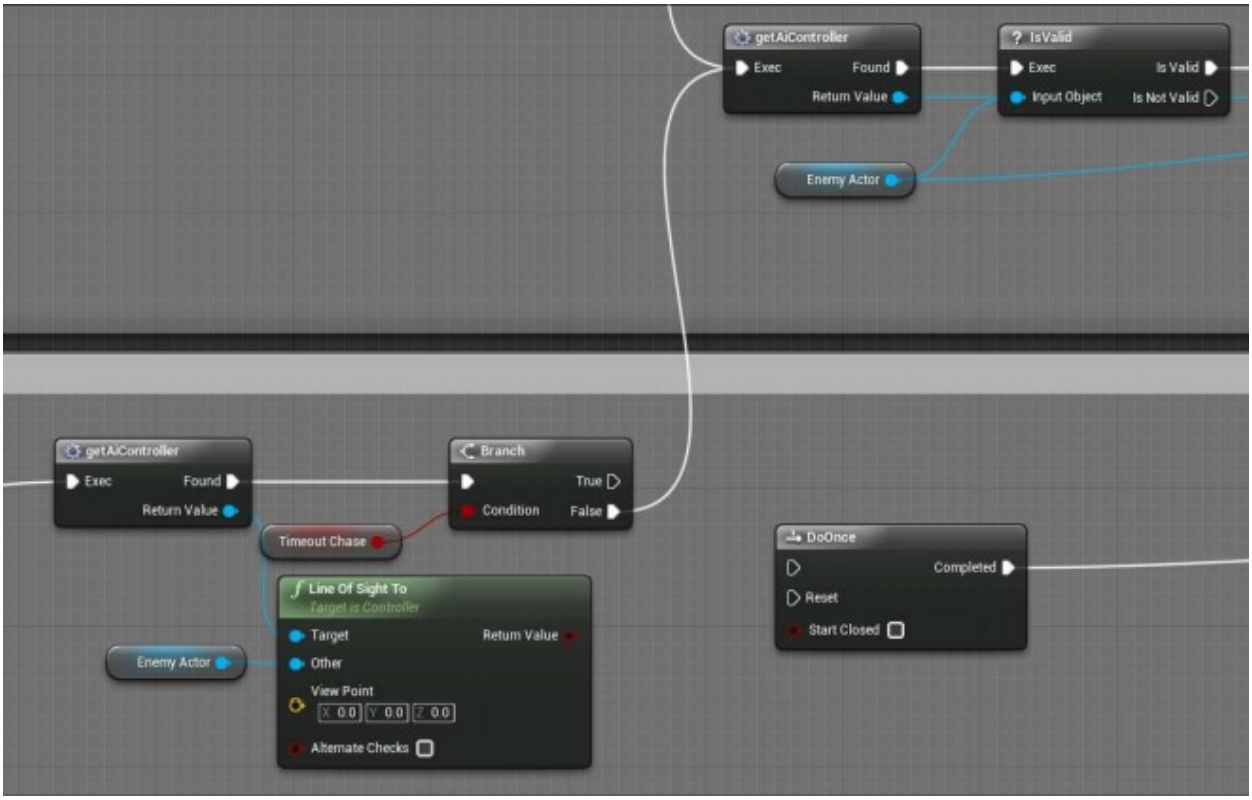
Chuyển đổi trạng thái

Nó kết thúc những gì cần thiết cho quá trình chuyển dịch của chúng ta ra khỏi trạng thái 1. Tiếp theo, chúng ta sẽ làm những gì cần thiết cho trạng thái 2 và điều này sẽ cho phép chúng ta phát hiện những thay đổi cần thiết để thoát khỏi trạng thái này trở lại trạng thái 0 sau khi chúng ta không còn nhìn thấy kẻ thù. Thực hiện các bước sau:

1. Quay trở lại node chuyển đổi trên **State** mà chúng ta đã tạo trước đó (ở Bước 55) và tạo một kết nối mới từ chân giá trị 2, hãy gọi đến *getAIController*.



2. Tiếp theo, chúng ta cần kiểm tra xem biến **Enemy Actor** của chúng ta có hợp lệ hay không. Chúng ta sẽ làm điều này bằng cách kéo biến **Enemy Actor** vào sơ đồ, và kéo từ chân đó, sau đó tìm và tạo node **IsValid**.
3. Nếu biến hợp lệ, chúng ta sẽ di chuyển AIController của mình về phía địch thủ của mình. Chúng ta có thể làm điều này bằng cách lấy từ **Return Value** trên *getAiController* và sau đó tạo node **Move To Actor**. Sau đó, chúng ta có thể kết nối biến **Enemy Actor** hợp lệ với tham số **Target** trên node **Move To Actor**.
4. Tiếp theo, chúng ta sẽ sử dụng **Tick Event** trong AIController để kiểm tra xem chúng ta có nhìn thấy địch thủ khi chúng ta ở trạng thái **2** hay không. Chúng ta có thể làm điều này bằng cách tập trung vào **Tick Event**, chọn nó và sau đó tạo node **Branch**.
5. Sau đó, gắp bỏ biến **State** vào sơ đồ và kiểm tra xem nó có bằng **2** hay không. Kết quả của node **Equal** này được bơm vào điều kiện của node **Branch**.
6. Nhìn vào node **Branch**, kéo từ **True** và tạo node *getAiController*. Sau đó, chúng ta sẽ lấy từ chân **Found** và tạo node **Branch**.
   Bây giờ, chúng ta cần tạo hai biến mới để duy trì việc đuổi theo kẻ thù của mình. Điều này sẽ cho phép chúng ta hết thời gian theo đuổi sau một khoảng thời gian nhất định hoặc kết thúc cuộc rượt đuổi ngay lập tức, buộc AI quay lại định tuyến giữa các waypoint mà chúng ta đã tạo.
7. Vì vậy, hãy tạo một biến mới gọi là *TimeoutChase*; đây sẽ là một biến kiểu **Boolean**. Tiếp theo, tạo một biến *ChaseTime* và đây sẽ là một **Float**. Nó sẽ yêu cầu dòng thực thi của chúng ta chuyển đổi giữa các luồng khác nhau nếu chúng ta không muốn hết thời gian theo đuổi.
8. Tập trung lại vào node **Branch** cuối cùng mà chúng ta đã tạo, chúng ta muốn cắm biến *TimeoutChase* mới tạo của chúng ta vào điều kiện.
9. Nếu node **Branch** này là **False**, chúng ta sẽ cắm nó vào node *getAiController* từ câu lệnh chuyển đổi trước đó của chúng ta. Nó sẽ trông giống như ảnh chụp màn hình sau:



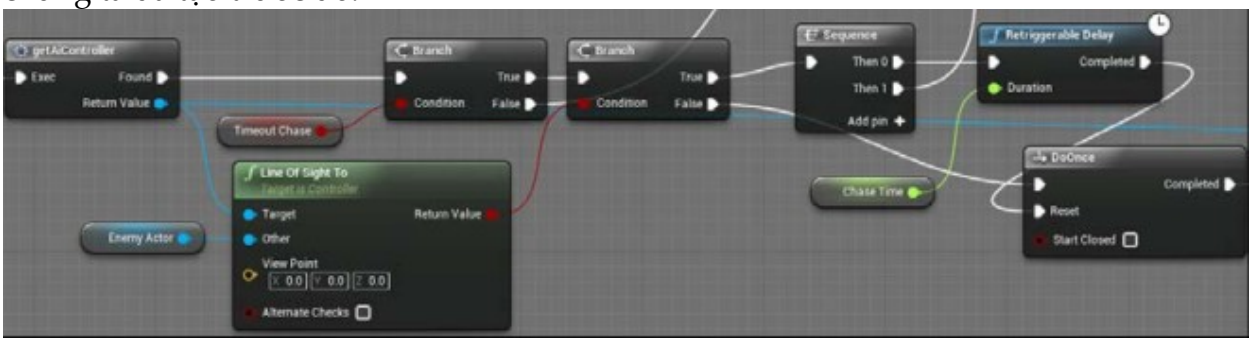
10. Bây giờ, nếu phát biểu điều kiện cắm vào **Branch** là **True**, chúng ta sẽ tạo node **Branch** mới để kiểm tra xem địch thủ của chúng ta có ở trong tầm nhìn hay không. Kéo từ **Return Value** của node *getAiController*, hãy tạo một node mới gọi là **Line of Sight To**.
11. Sau đó, lấy biến **Enemy Actor** của chúng ta vào trong sơ đồ, chúng ta sẽ cắm biến này vào đầu vào **Other** của node **Line of Sight To**. Sau đó, từ chân **Return Value** của nó, chúng ta có thể tạo một node Branch mới với điều kiện đã được đặt sẵn trước đó.

Khôi phục trạng thái

Bây giờ, phần tiếp theo này sẽ cho phép chúng ta sử dụng biến *ChaseTime* trong node **Retriggerable Delay**. Về cơ bản, khi chúng ở trong tầm nhìn, node **Retriggerable Delay** sẽ tiếp tục được khôi phục. Nhưng khi chúng ta mất tầm ngắm, thì đường dẫn thực thi khác sẽ thực thi **DoOnce**, cố gắng khôi phục biến **State** của AI của chúng ta. Khi node **Retriggerable Delay** kết thúc việc trì hoãn theo giá trị **Chase Time** được chỉ định, nó sẽ đặt lại **DoOnce**, cho phép bot của chúng ta khôi phục trạng thái của nó. Hãy thực hiện các bước sau:

1. Bây giờ, hãy xem node **Branch** mà chúng ta vừa tạo và kéo từ chân **False** để tìm đồng thời tạo một node **DoOnce** mới. Chúng ta có thể để **Start Closed** như mặc định.
2. Nếu chúng ta theo dõi trở lại **True** từ node **Branch** trước đó, chúng ta có thể kéo từ **True** sau đó tạo node **Sequence**.
3. Kéo từ chân thực thi **Then 0** đầu tiên và tạo node **Retriggerable Delay**. Tiếp theo, hãy cắm biến *ChaseTime* của chúng ta vào chân **Duration** của node **Retriggerable Delay**.

Cuối cùng, hãy cắm chân **Completed** vào chân tham số **Reset** của nút **DoOnce** mà chúng ta đã tạo trước đó:

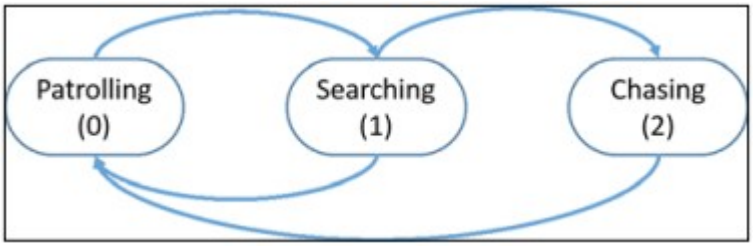


4. Quay lại node **Sequence**, chúng ta cần định tuyến **Then 1** đến node **Move To Actor** trước đó, như chúng ta đã làm trong phần *chuyển đổi trạng thái*.
   Vì vậy, trong khi chúng ta nhìn thấy địch thủ của mình, chúng ta sẽ tiếp tục tiến về phía chúng cho đến khi chúng ta mất tầm với địch thủ, trong trường hợp đó, nó sẽ chuyển sang luồng thực thi khác. Bây giờ, nếu bạn muốn AI di chuyển đến vị trí của kẻ thù, bạn sẽ cần lấy vị trí của actor. Điều này sẽ dẫn đến việc AI di chuyển về vị trí cuối cùng đã biết của địch thủ trước khi được đặt lại.
5. Bây giờ, hãy tập trung vào nút **DoOnce**, hãy kéo từ nút *getAiController* mà chúng ta đã tạo trước đó trong chuỗi. Sau đó, tạo một node mới sau node **DoOnce** có tên là **Stop Movement**. Điều này sẽ hủy bỏ mọi chuyển động về phía kẻ thù của chúng ta.
6. Tiếp theo, chúng ta cần đặt biến **Enemy Actor** thành **null**. Chúng ta sẽ đặt biến **State** của mình thành **0**. Cuối cùng, chúng ta sẽ gọi event **Next Route** của mình. Buộc AI trả lại tuyến đường giữa waypoint của chúng ta một lần nữa, chúng ta sẽ đợi địch thủ xuất hiện.

Giả lập và chơi game

Bây giờ nếu chúng ta nhấn **Play**, chúng ta sẽ thấy AI đuổi theo bạn cho đến khi bạn khuất tầm nhìn!

Vì vậy, bot sẽ định tuyến như trước đây cho đến khi kẻ thù ở trong tầm ngắm. Nếu bot ở trạng thái 0, nó sẽ chuyển sang trạng thái 1 sau khi kẻ thù được phát hiện bởi component AI Perception. Tiếp theo, nếu một trong những kẻ thù này ở trong tầm nhìn và chúng ta không có địch thủ hiện tại, chúng ta sẽ thay đổi trạng thái của mình thành 2 và thiết lập địch thủ của chúng ta. Cuối cùng, chúng ta sẽ tiếp tục di chuyển về phía địch thủ của mình cho đến khi chúng ta mất dấu chúng, dẫn đến việc chúng ta bị đưa về trạng thái **0**. Sơ đồ sau đây minh họa các chuyển đổi này:



Đây là cách chúng ta tạo AI mà không cần sự hỗ trợ của **Behavior Tree**. Cuối cùng, bạn vẫn cần tạo một state machine vì đây là cách tiếp cận cơ bản được sử dụng trong lập trình để tạo các hàm chuyển đổi lẫn nhau dựa trên các biến được chia sẻ bởi các hàm. Bằng cách thực hiện điều này trực tiếp trong blueprint, chúng ta có thể hiểu sâu hơn về cơ chế kiểm soát hành vi dựa trên quá trình chuyển đổi trạng thái và điều này tốt cho hành vi AI đơn giản.

Tóm tắt

Bây giờ chúng ta sẽ kết thúc chương về những gì bạn đã tìm hiểu về component AI Sense. Công cụ này thật tuyệt vời để dễ dàng tích hợp các cảm biến phản hồi mới vào đầu vào AI của bạn mà bạn không phải tự tạo từ những thứ như Ray Traces.

Tiếp theo, chúng ta sẽ tập trung vào chuyển động nâng cao hơn, chẳng hạn như giúp AI của chúng ta tránh chướng ngại vật bằng cách sử dụng một số công cụ do Unreal Engine cung cấp và cả cách chúng ta có thể để AI theo dõi nhau, tương tự như hành vi trong đội.