Course - Lập trình AI cho Unreal Engine

Thêm tính ngẫu nhiên và xác suất

Hướng dẫn bạn cách tạo các kỹ thuật xác suất và ngẫu nhiên có thể được sử dụng để thêm tính ngẫu nhiên, cơ hội và tính chất cho AI, giúp trò chơi trở nên khó đoán và thú vị hơn. Chúng tôi sẽ đề cập đến cách chúng được sử dụng trong Unreal Engine 4.

Tags: Thêm tính ngẫu nhiên và xác suất

Trong chương này, chúng tôi sẽ giới thiệu các kỹ thuật ngẫu nhiên và xác suất có thể được sử dụng để thêm tính ngẫu nhiên, cơ hội và đặc điểm cho AI. Chúng ta sẽ bắt đầu nhanh với "Xác suất 101" để giải thích một số khái niệm cơ bản; sau đó, chúng ta sẽ trình bày cách sử dụng Stream để kiểm soát phân phối số không đồng đều và sử dụng các kết quả này để chứng minh xác suất. Chúng ta sẽ đề cập đến cách chúng được sử dụng trong Unreal Engine 4. Cuối cùng, chúng ta sẽ xây dựng dựa trên AIController của kẻ thù để nó thực hiện một hành động ngẫu nhiên.

Chương này sẽ đề cập đến:

- Xác suất, phân phối xác suất và phân phối không đều.
- Sử dụng RandomStream trong UE4 để thêm tính ngẫu nhiên vào AI của chúng ta.
- Thêm hành vi ngẫu nhiên vào trạng thái dựa trên AI của kẻ thù.

Bạn có thể download dự án full source, tại đây.

Giới thiệu về tính xác suất

Chúng ta biết một sự kiện sẽ xảy ra, nhưng nó sẽ xảy ra thường xuyên như thế nào? Đây là cách chúng ta có thể định lượng xác suất và đây là thứ chúng ta sẽ sử dụng để kiểm soát tần suất của một kết quả. Vì vậy, giả sử chúng ta lật một phần tư. Sự kiện mà chúng ta biết sẽ xảy ra là mặt ngửa (H), nhưng nó vẫn có thể rơi xuống mặt sấp (T). Vì vậy, cách chúng ta viết xác suất rơi ngửa là *P (H) = ?*

Tại thời điểm này, chúng ta biết rằng mặt ngửa sẽ xảy ra, mặc dù chúng ta vẫn không biết nó sẽ xảy ra thường xuyên như thế nào. Để hiểu điều này, trước tiên chúng ta phải lấy số lượng các kết quả có thể xảy ra đáp ứng các điều kiện của chúng ta, là 1 cho mặt ngửa. Sau đó, chúng ta phải tính số sự kiện có khả năng xảy ra như nhau, là 2. Vì vậy, bây giờ chúng ta cần đưa điều này vào phương trình xác suất:

Nếu chúng ta làm một số phép toán cơ bản và chia nhỏ hơn nữa, chúng ta sẽ có 50%:

P (H) = 1/2 = 50%

Điều này có nghĩa là, nếu bạn tung một đồng xu một triệu hoặc thậm chí một tỷ lần, số lần bạn tung đồng xu càng nhiều thì sẽ càng gần đến 50% các sự kiện xuất hiện mặt ngửa và mặt sấp như nhau. Điều này rất hữu ích khi kiểm soát khả năng một chức năng sẽ thực thi.

Trong ví dụ của chúng tôi, chúng ta sẽ chứng minh xác suất kiểu xổ số. Chúng ta xác định rằng chúng ta có 20% cơ hội nhận được vàng, tương đương với số thập phân là 0,2. Chúng ta sẽ tạo một số ngẫu nhiên từ 0 đến 1. Chúng ta nhận được 0,19 và chúng ta đang có tiền! Chúng ta quay lại và chúng ta nhận được 0,57, chúng ta hết tiền!

Đây là một số cách để kiểm soát luồng thực thi AI của bạn. Bạn cũng có thể áp dụng điều này cho các hành vi khác có liên quan trực tiếp đến các đặc điểm của AI cụ thể. Vì vậy, nếu nhân vật này thuộc một class nào đó, một số quái vật nhất định có thể sợ hãi hơn trước class nhân vật này.

Phân phối xác suất

Phân phối xác suất của các con số có nghĩa là chúng ta sẽ biết số lượng hữu hạn trong các kết quả bằng nhau. Vì vậy, chúng ta sẽ sử dụng điều này, ví dụ, nếu chúng ta có một con quái vật với rất nhiều vũ khí. Chúng ta không muốn tất cả vũ khí rơi ra nếu quái vật chết. Mặc dù, nếu chúng ta chỉ áp dụng một quy tắc đơn giản để hạn chế số lượng vũ khí rơi ra, thì cũng vẫn có cơ hội ngang nhau để họ có được thanh kiếm hiếm so với thanh kiếm dở tệ mà không ai muốn.

Giả sử chúng ta đã áp dụng phân phối xác suất để lựa chọn vũ khí. Đây là kịch bản. Chúng ta có 50% vật phẩm được xếp hạng dở đến ổn và 35% vật phẩm được xếp hạng tốt đến tốt nhất, kết thúc với vật phẩm hiếm ở mức 15%. Bây giờ, chúng ta sẽ tạo một số lượng vũ khí ngẫu nhiên mà họ có thể nhận được và các mục riêng lẻ được chọn sẽ được tạo với phân phối xác suất.

Phân phối không đồng đều

Phân phối số không đồng đều có nghĩa là chúng ta sẽ không biết số lượng kết quả hữu hạn. Chúng ta biết rằng 0 đến 1 sẽ được tạo nhưng không biết liệu 0 hay 1 có nhiều khả năng được tạo hơn. Như đã nói, điều này là hoàn hảo để tạo ra nhiều kịch bản không thể đoán trước. Trong UE4, chúng ta có công cụ Random và RandomStream điển hình. Sự khác biệt rõ ràng giữa hai loại này là chúng ta có thể kiểm soát hạt giống mà sẽ sinh ra đầu ra ngẫu nhiên của chúng ta.

Vì vậy, nói một cách khác, phân phối không đồng đều là một sự sinh ta điều ngẫu nhiên của các số không có khả năng xảy ra như nhau. Hành vi thực sự của tính ngẫu nhiên có thể được nhân rộng nhưng phải trả giá bằng thời gian và hiệu suất. Vì vậy, chúng tôi đã đưa ra các thuật toán để tạo ra các số ngẫu nhiên. Một trong những công cụ mà UE4 cung cấp để tạo số ngẫu nhiên là RandomStream.

RandomStream trong Unreal Engine 4

Giả sử bạn muốn đóng băng các cây được tạo ngẫu nhiên mà bạn đã tạo. Thông thường, khi sử dụng một node ngẫu nhiên từ pháo UE4, thông tin này sẽ bị mất khi bạn thoát khỏi trò chơi. Với RandomStream, bạn thực sự có thể lưu lại hạt giống đã sinh ra đầu ra ngẫu nhiên

và tải cùng một đầu ra ngẫu nhiên bằng cách gọi hạt giống này. Vì vậy, trong UE4, bạn sẽ tìm thấy một node blueprint có tên **Make RandomStream**. Node này có một chân đầu vào được gọi là Initial Seed.

Lý do chúng tôi sử dụng hạt giống là vì nó là chương trình tạo số ngẫu nhiên giả. Có nghĩa là các con số được tính toán và mang tính xác thực. Giả có nghĩa là chúng ta sẽ sử dụng các thuật toán để tính toán tính ngẫu nhiên; tuy nhiên, bằng cách tính toán tính ngẫu nhiên, về mặt kỹ thuật, chúng ta biết trước nó sẽ là gì, do đó làm cho nó trở nên xác thực. Sự tương phản là tính ngẫu nhiên thực sự, ghi lại tiếng ồn từ một khu rừng, thác nước, bầu không khí hoặc thậm chí là một kênh truyền hình. Lý do điều này sẽ vô cùng ngẫu nhiên hơn là do tiếng ồn thu được không có khả năng lặp lại, cũng như nhiều thứ trong cuộc sống. Với các tính toán, luôn có khả năng vì nhiệm vụ là tái tạo hiện tượng có tiêu tốn một phần hiệu suất. Ban đầu, khi chúng tôi tạo ra máy tính, bộ nhớ rất quan trọng — nó vẫn vậy. Nhưng tại thời điểm này, chúng ta đã tạo ra các thuật toán để thay thế nhu cầu lưu trữ tiếng ồn, nhưng chúng ta muốn có một khóa hoặc hạt giống vì chúng tôi quen thuộc hơn với chúng. Lý do mà nó có thể được coi là một chìa khóa là khi có nó, hai người trên các máy tính riêng biệt có thể tạo ra các số ngẫu nhiên giống hệt nhau.

Bây giờ về mặt kỹ thuật, nếu chúng ta tăng chiều dài của hạt lên một cái gì đó cao theo cấp số nhân, nó sẽ thực sự là vô cùng ngẫu nhiên. Bạn phải thêm điều này bởi vì chúng tôi biết rằng có một kết thúc nhất định, trong khi với cuộc sống thì không có kết thúc hữu hạn. Vì vậy, trong UE4, việc ngẫu nhiên hóa hạt giống sẽ ảnh hưởng trực tiếp đến tính ngẫu nhiên của các số được tính toán. Đây là những gì tôi làm để tạo ra sự ngẫu nhiên giả lập tính không thể đoán trước được.

Kế hoạch như sau

Chúng ta sẽ tiếp tục từ nơi đã dừng ở *Chương 2, Tạo AI cơ bản* và tiếp tục thực hiện các thay đổi đối với hai AIController của chúng ta. Trong chương trước, chúng ta đã thiết lập *Enemy* của mình để đuổi theo *Hero* của chúng ta vô thời hạn. Trong chương này, chúng ta sẽ giới thiệu một trạng thái khác cho AI của *Enemy*, trạng thái này sẽ cho phép nó chạy khỏi chúng ta hoặc chạy về phía chúng ta. Nó thể hiện giả lập cho việc Enemy lại gần Hero hoặc tránh xa Hero, tùy thuộc vào xác suất sốc được khi *Hero* gặp *Enemy*.

Bắt đầu nào!

Thêm trạng thái Đi lang thang (Wander)

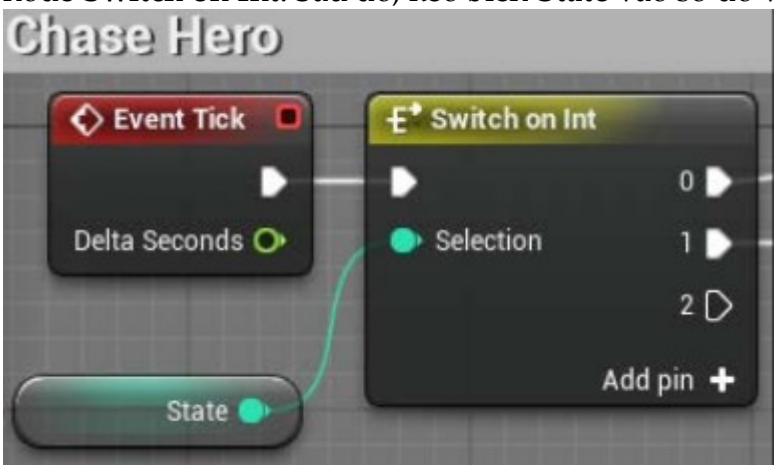
Ở đây, chúng ta sẽ xác định ba trạng thái của AI. Điều này sẽ giúp dễ dàng hơn khi chỉ định các hành động ở các trạng thái khác nhau. Bây giờ, trạng thái đầu tiên chúng ta sẽ thêm là

Wander. Nó sẽ làm cho AI di chuyển ngẫu nhiên và vô tận. Đây sẽ là trạng thái ban đầu của AI và nó sẽ có khả năng chuyển đổi sang các trạng thái khác khi chúng ta tiếp cận AI. Hai trạng thái còn lại là hai phản ứng mà chúng ta sẽ cung cấp cho AI. Ý tưởng là khi chúng ta tiếp cận AI, có khả năng nó sẽ chạy trốn hoặc lại gần. Cơ hội là xác suất mà chúng tôi xác định và tạo ra trong Blueprint.

Thiết lập dự án

Hãy mở Unreal Engine 4! Thực hiện các bước sau:

1. Đầu tiên, chúng ta phải thay đổi *Enemy* của mình để hành động như một con quái vật sợ hãi lang thang. Vì vậy, hãy nhấp đúp vào *Enemy* AIController. Bây giờ, điều hướng đến sơ đồ **Graph Event** và phóng to.
2. Chúng tôi muốn tạo một biến mới cho **State** và cho nó kiểu giá trị **Integer**.
3. Tiếp theo, chúng ta phải thay đổi script lệnh **Find Hero**. Chúng ta sẽ điều chỉnh điều này để phát hiện ra *Hero*. Những gì chúng ta muốn giữ lại là blueprint của **Chase Hero**. Nó sẽ được sử dụng đến gần hoặc chạy khỏi chúng ta, Bây giờ, hãy tập trung vào kịch bản.
4. Xóa **Event Begin Play** từ đầu script lệnh. Tiếp theo, kéo xuống **Event Tick** từ script lệnh **Chase Hero**.
5. Thay thế node **Delay** nằm sau node **Event Tick** trong script lệnh **Chase Hero** bằng node **Switch on Int**. Sau đó, kéo biến **State** vào sơ đồ và kết nối nó với chân **Selection**:

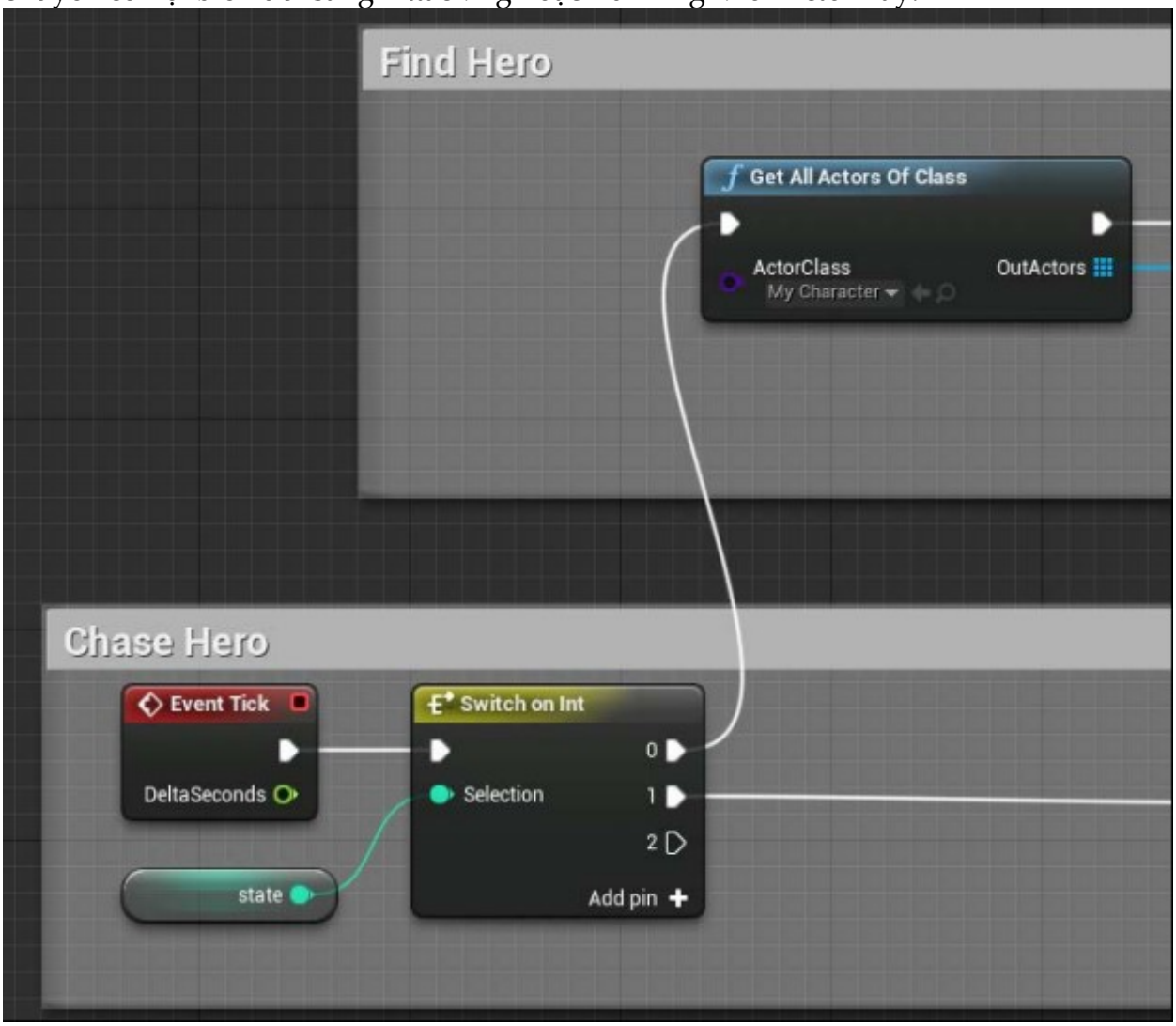


> **Chú ý: Switch on Int** là một chức năng rất hữu ích. Tôi đã sử dụng nó vô số lần để thay thế công việc dư thừa mà bạn thường có thể thực hiện theo lượt, chẳng hạn như lặp lại một nhiệm vụ nhiều lần trước khi chuyển sang chỉ mục tiếp theo.

6. Giờ đây, blueprint AI sẽ liên tục nhảy dựa trên **State**. Các script lệnh khác nhau sẽ được thực thi, và nó sẽ được chuyển đổi giữa các trạng thái và đây là cơ sở cho Finite State Machine.
7. Tiếp theo, hãy kết hợp hai script lệnh của chúng ta với node **Switch on Int**. Chúng ta

muốn script lệnh **Find Hero** được kết nối với chỉ mục **0** của node. Sau đó, kết nối chỉ mục **1** của node với script lệnh **Chase Hero**. Vì vậy, bạn có thể nhấn *Ctrl* và nhấp vào chỉ số **0**, di chuyển **Move to Location** đến chỉ mục **1** sau khi kết nối chỉ số **0** với **Get All Actors Of Class**.

8. Vì vậy, chúng tôi phải kiểm tra khoảng cách của phần tử Actors được tìm thấy từ **Get All Actors Of Class**, và sau đó, nếu một Actors nằm trong phạm vi, chúng tôi sẽ chuyển cơ hội biến đổi sang Attacking hoặc Running khỏi Actor này:

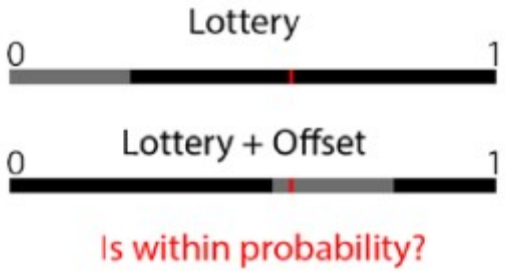


9. Bây giờ, hãy tập trung vào **ForEachLoopWithBreak** và hãy kiểm tra phần tử Actor đầu ra hiện tại để biết khoảng cách với Actor của blueprint hiện tại bằng cách sử dụng node **Get Horizontal Distance To**. Bây giờ tạo hai node **Get Horizontal Distance To** và **Get Controlled Pawn** và kết nối node **Get Controlled Pawn** với chân **Target** của node **Get Horizontal Distance To**.
10. Từ chân **Array Element**, kết nối với **Other Actor** của **Get Horizontal Distance To**. Khoảng cách mà chúng ta muốn cho Actor là 512 đơn vị. Vì vậy, hãy kéo từ chân **Return Value**, nhập ký hiệu nhỏ hơn (<) và so sánh nó với 512. Một lần nữa, điều chúng ta muốn làm là đảm bảo rằng chúng ta không bao giờ chạy khỏi chính mình.

11. So sánh node **Array Element** với **Get Controlled Pawn** bằng cách sử dụng node **Not Equal**. Sau đó, tạo một node cổng **AND** và nối cả hai kết quả Boolean. Tạo node **Branch** từ kết quả từ nút cổng **AND**. Node **Branch** này phải ở sau node **Get Horizontal Distance To**.

Tạo ra xác suất

Script lệnh Blueprint mà chúng ta sắp tạo thể hiện một xác suất đơn giản nhưng hiệu quả. Hệ thống hoạt động dựa trên ý tưởng về xác suất xổ số đơn giản là 0-1 với một biến thể. Vì vậy, trước tiên bạn xác định khả năng TRUE sẽ trả về từ hàm. Tiếp theo, bạn đã xác định phạm vi xổ số sẽ mở rộng, theo mặc định là 1. Phần còn lại để lại cho script xử lý. Bây giờ nó sẽ lấy **Change Weight** và **Number Span** sau đó tính toán phần bù. Từ đó, chúng ta sẽ tính toán một loạt các xác suất từ **Chance Weight**. Sau đó, chúng ta sẽ tạo một số ngẫu nhiên trong phạm vi số đếm của chúng ta và kiểm tra xem nó có nằm trong phạm vi xác suất mà chúng ta đã xác định trước đó hay không. Biến thể nhỏ này làm cho xác suất ít dự đoán được hơn. Hình ảnh sau đây cho thấy kịch bản xác suất xổ số đơn giản đang kiểm tra một giá trị ngẫu nhiên trong phạm vi xác suất đã xác định:



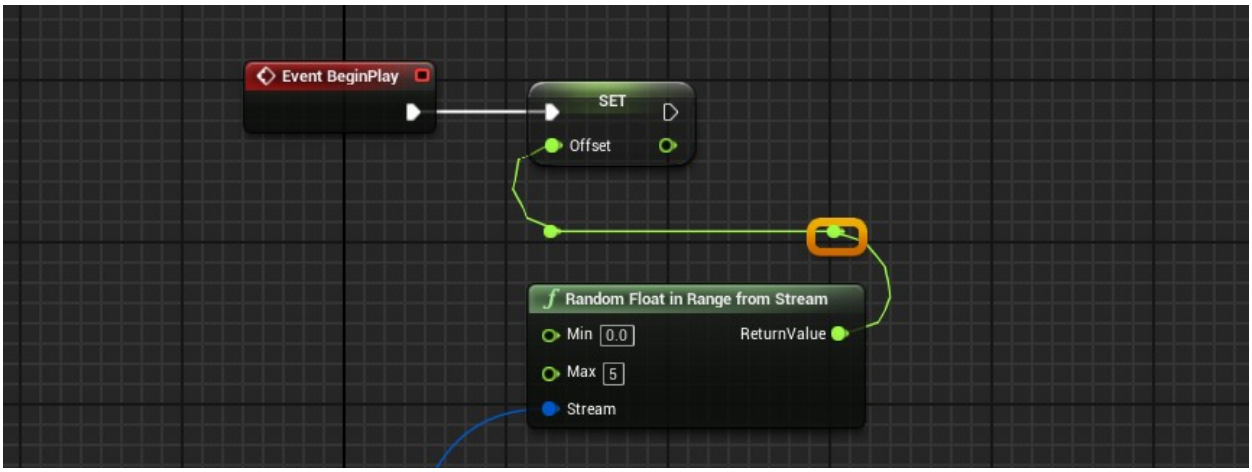
Lưu ý **Offset** từ Lottery trong phần thứ hai, kết quả là đúng. Lần tới, phạm vi xác suất sẽ không còn ở đó và một giá trị ngẫu nhiên mới sẽ được tính toán.

Phân phối không đồng đều với RandomStream

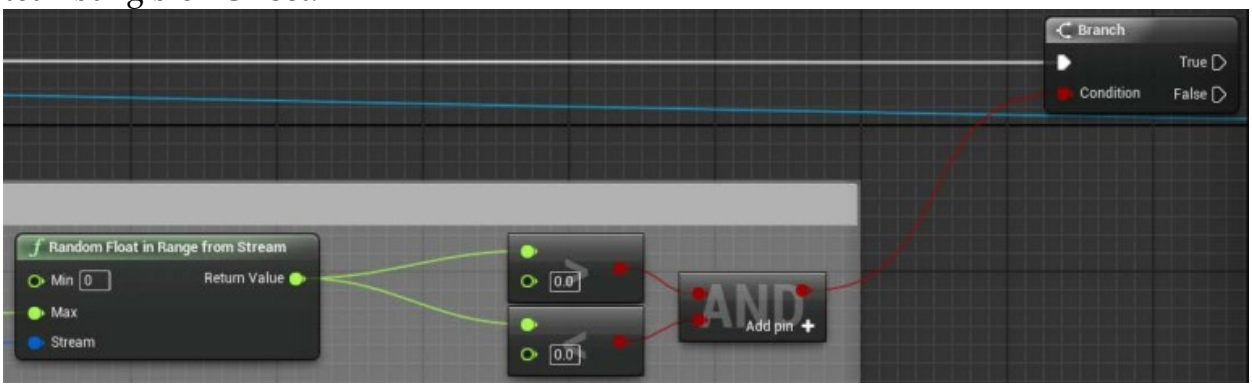
Điều đầu tiên chúng ta muốn thực hiện là khởi tạo **RandomStream**. Chúng ta có thể làm điều này bằng cách tạo một node **Make RandomStream** và điền mã pin hạt giống ban đầu bằng một số nguyên ngẫu nhiên trong phạm vi số đếm mà mình muốn. **RandomStream** cho phép chúng ta kiểm soát việc tạo đầu ra ngẫu nhiên. Nếu chúng ta không thay đổi hạt giống trong **RandomStream**, nó sẽ liên tục tạo ra kết quả giống nhau. Vì lí do đó, chúng ta phải liên tục đặt lại hạt giống trong luồng để đầu ra ngẫu nhiên thay đổi. Thực hiện các bước sau:

1. Tập trung trở lại phầm *Enemy* AIController, tạo node **Random Integer in Range** và đặt giá trị **Max** thành **65535**. Sau đó, kéo từ **Return Value** và tạo node **Make Random Stream**.

2. Bây giờ, chúng ta cần tạo các biến để cập nhật RandomStream. Tạo một số float **Offset** với giá trị mặc định là **0**. Sau đó, tạo một số float **ChanceWeight** với giá trị mặc định là **0,25**. Tạo một số float **Max N** với giá trị mặc định là **1**.
3. Sau đó, tạo node **Event Begin Play**. Kéo biến **Offset** vào sơ đồ và tạo node **SET** (dùng **Alt+Drag**). Kéo từ chân thực thi của **Event Begin Play** đến node **SET** bạn vừa tạo. Tiếp theo, tạo **Random Float in Range from Stream** và kéo từ **Return Value** sang số float **Offset**:

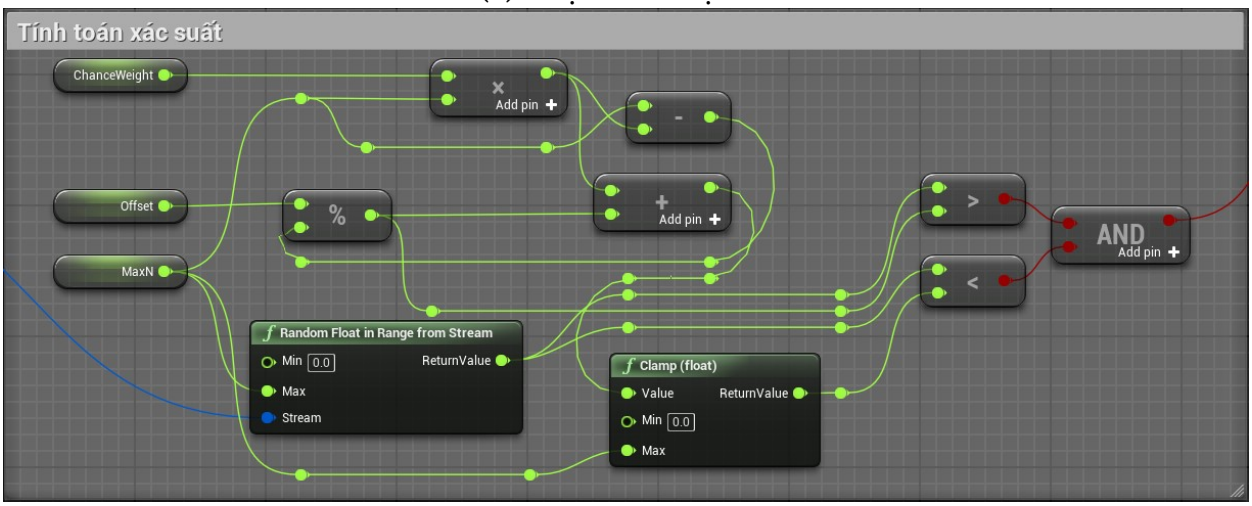


4. Bây giờ chúng ta cần cắm RandomStream của mình vào node này và kết nối biến **Max N** với số float **Max** của **Random Float in Range from Stream**. Nó bây giờ sẽ tạo ra một offset ngẫu nhiên khi bắt đầu trò chơi.
5. Bây giờ, hãy bắt đầu script tính xác suất. Hãy lấy biến của chúng ta và tạo node **Random Float in Range from Stream**. Sau đó, chúng ta sẽ tạo hai node từ node này để so sánh **Return Value**. Chúng ta muốn kiểm tra xem giá trị lớn hơn hay nhỏ hơn. Sau đó, kéo chân và tạo cổng logic **AND**. Cuối cùng, kéo so sánh Boolean khác vào.
6. Tiếp theo, kéo kết quả của **AND** sang một node **Branch** mới. Điều chúng ta cần làm tiếp theo là so sánh kết quả ngẫu nhiên với **Lottery Probability Range** mà chúng ta đã tính toán bằng biến **Offset**:



7. Vì vậy, lấy **Offset** từ các biến vào trong sơ đồ. Chúng ta muốn đặt cái này theo modulo (%) và sau đó kết nối nó với nút so sánh lớn hơn (>).
8. Tiếp theo, chúng ta sẽ lấy **ChanceWeight** từ các biến vào sơ đồ. Sau đó, chúng ta sẽ nhân giá trị này với giá trị **Max N** để tính phạm vi xác suất. Chúng ta sẽ làm hai việc với kết quả của phép tính này, như trong các bước tiếp theo.

9. Đầu tiên, chúng ta muốn lấy sự khác biệt từ giá trị **Max N** với kết quả. Sau đó, chúng ta sẽ kết nối nó với modulo (%).
10. Tiếp theo, chúng ta muốn thêm phạm vi xác suất vào kết quả của modulo (%). Bằng cách là chúng ta sẽ luồn modulo này vào một cái node **Clamp** để đảm bảo an toàn. Chúng ta sẽ đặt giá trị **Min** thành **0** và **Max** thành **Max N**. Kết quả của Clamp này sẽ cắm vào node so sánh **Less Than (<)** được hiển thị như sau:



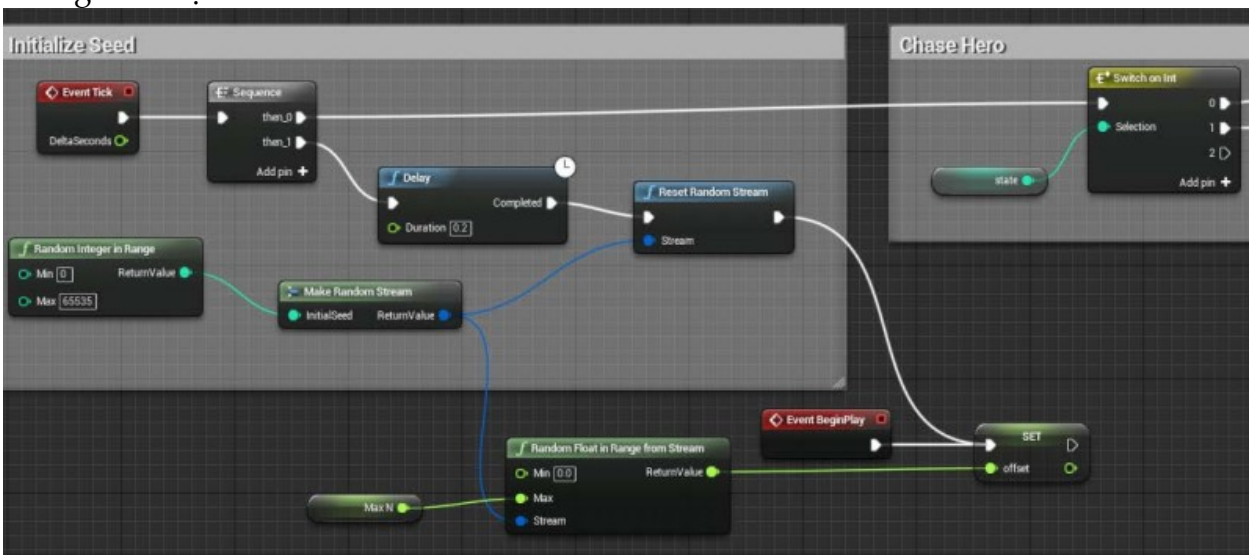
Tạo sự chuyển tiếp

Chuyển tiếp giúp áp dụng các hành động thích hợp cho những gì đang được cảm nhận trong môi trường của AI. Chúng sẽ tiếp tục luồng thực thi và đây là điều làm cho Finite State Machine trở nên đặc biệt. Chúng ta sẽ áp dụng điều này trong dự án của mình bằng cách chuyển sang trạng thái Running hoặc Attacking, và sau một thời gian, nó sẽ tính toán xem nên chạy trốn hay tiếp tục tấn công. Khi không tìm thấy kẻ thù, AI sẽ chuyển trở lại trạng thái đi lang thang. Thực hiện theo các bước sau:

1. Bây giờ, từ node **Branch** mà chúng ta đã tạo lần cuối trong script trên, hãy tạo node **SET** biến **State** của chúng ta trên chân **True** cho node **Branch**. Sau đó, chúng ta sẽ tạo node **SET** khác cho biến **State** trên chân **False** của node **Branch**. Từ đó, hãy thiết lập giá trị **True Set State** thành **1**, sau đó thiết lập giá trị **False Set State** thành **2**.
2. Tiếp theo, hãy đổi tên biến **Hero** thành **Target** để dễ thực hiện hơn. Tiếp theo, chúng ta sẽ đặt **Target** sau hai node **Set State** vừa tạo của chúng ta.
3. Sau đó, chúng ta sẽ kết nối cả hai node **Set State** vào node biến **Set Target**. Node **Set Target** cần cắm vào trong chân **Break** của node **ForEachLoopWithBreak**.
4. Bây giờ để thiết lập lại RandomStream. Hãy thu nhỏ và tìm node **Event Tick**. Từ đó, hãy kéo chân và tạo **Sequence** giữa **Switch on Int**.
5. Từ **then_1** của nút **Sequence**, hãy tạo node **Delay** với **Duration** là **0.2** giây. Sau đó, kéo từ chân **Return Value** của **Make Random Stream** và tìm **Reset Random Stream**. Thao tác này sẽ thiết lập lại luồng và nếu bạn có **Random Integer in Range** được kết nối như

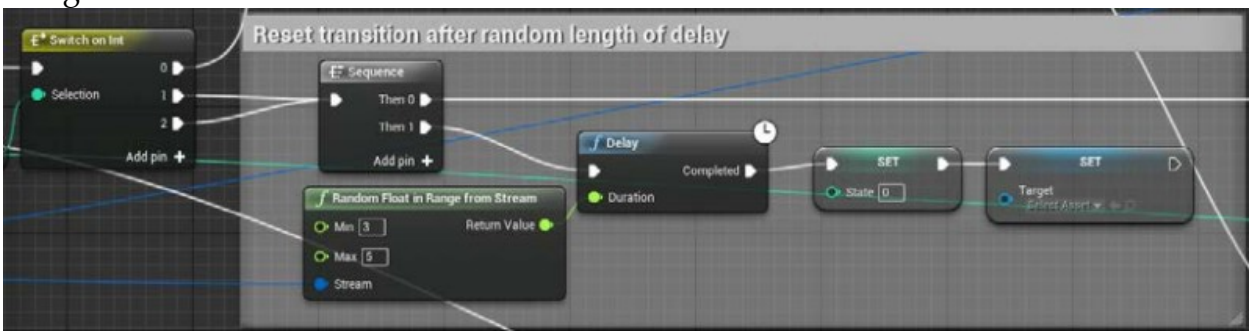
chúng ta thực hiện bên dưới, nó sẽ tạo một số nguyên **Int** ngẫu nhiên mới mỗi khi nó được reset lại.

6. Cắm cái này vào sau node **Delay** để quá trình thiết lập lại diễn ra mỗi giây. Sau đó, chúng ta sẽ kéo từ node **Reset Random Stream** sang biến **offset** của node **SET** mà chúng ta đã tạo từ **Event**:



Set offset thành một giá trị mặc định

7. Bây giờ, hãy tạo một node **Sequence** mới bằng cách kéo từ chỉ mục **1** của node **Switch on Int**. Sau đó, kết nối chỉ mục **2** của node **Switch on Int** với node **Sequence** trước đó.
8. Bây giờ, hãy kéo từ **Then 1** của node **Sequence** và tạo node **Delay**. Node này sẽ chuyển AI của chúng ta trở lại trạng thái lang thang từ đó nó có thể phát hiện lại *Enemy*.
9. Vì vậy, hãy tạo node **Random Float in Range from Stream** và kết nối chân **Stream** với **Random Stream** mà chúng ta đã tạo trước đó. Sau đó, đặt giá trị **Min** thành **3** và giá trị **Max** thành **5**.
10. Sau node **Delay**, thiết lập biến **State** với giá trị **0**. Sau đó, thiết lập biến **Target** thành trống:

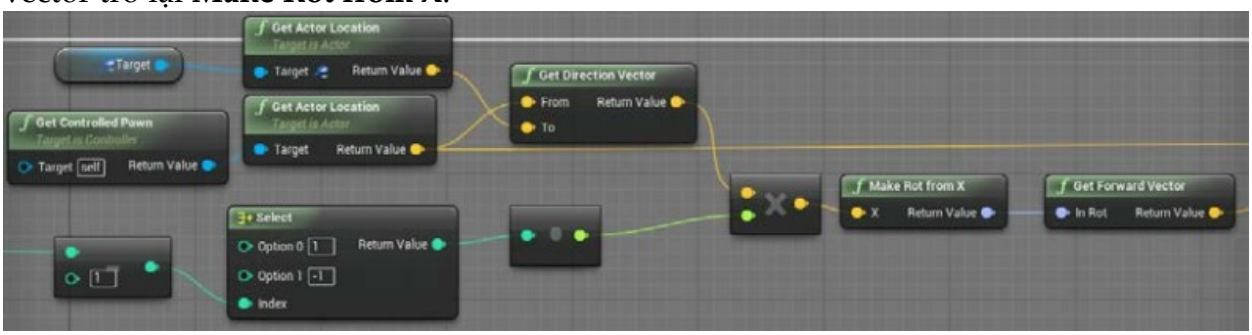


Chạy trốn và tấn công

Bây giờ, chúng ta sẽ lấy một phần đã có của mình và thêm một vài thay đổi nhỏ, nó sẽ khiến nhân vật chạy khỏi **Target** thay vì hướng về phía mục tiêu đó. Điều này về cơ bản sẽ chứng

minh khả năng bạn có thể đưa ra các phản ứng AI của mình trước những tình hình khác nhau. Bây giờ chúng ta thực hiện như sau:

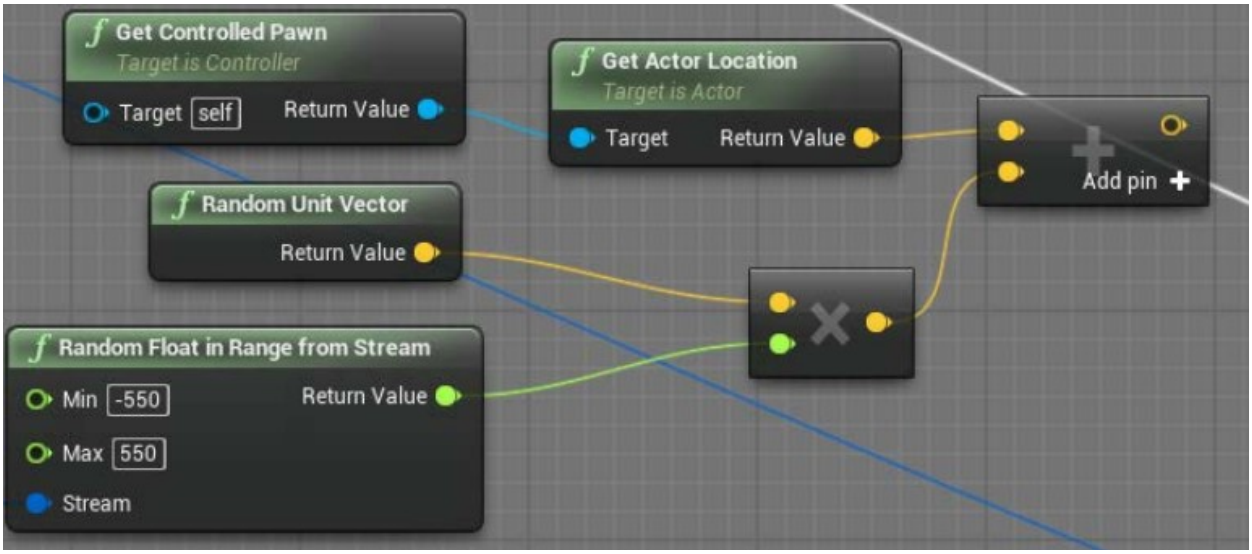
1. Chúng ta sẽ nhận biến **State** của chúng ta. Sau đó, chúng ta sẽ trừ đi **1** từ số này, và kết quả sẽ nhận được là số **0** hoặc **1**. Tiếp theo, chúng ta sẽ tạo một node **Select** và cắm kết quả của phép trừ vào node.
2. Từ đó, chúng ta sẽ thiết lập **Option 0** là **1** và **Option 1** là **-1**. Điều này sẽ khiến nhân vật Run hoặc Flee khỏi Target. Bây giờ, hãy phóng to node **Get Direction Vector**.
3. Hãy nhân **Return Value** với kết quả của node **Get Direction Vector**. Trước tiên, bạn phải chuyển đổi khiểu **Int** thành kiểu **Float**. Sau đó, nhân số này với vector. Sau đó, kéo vector trở lại **Make Rot from X**:



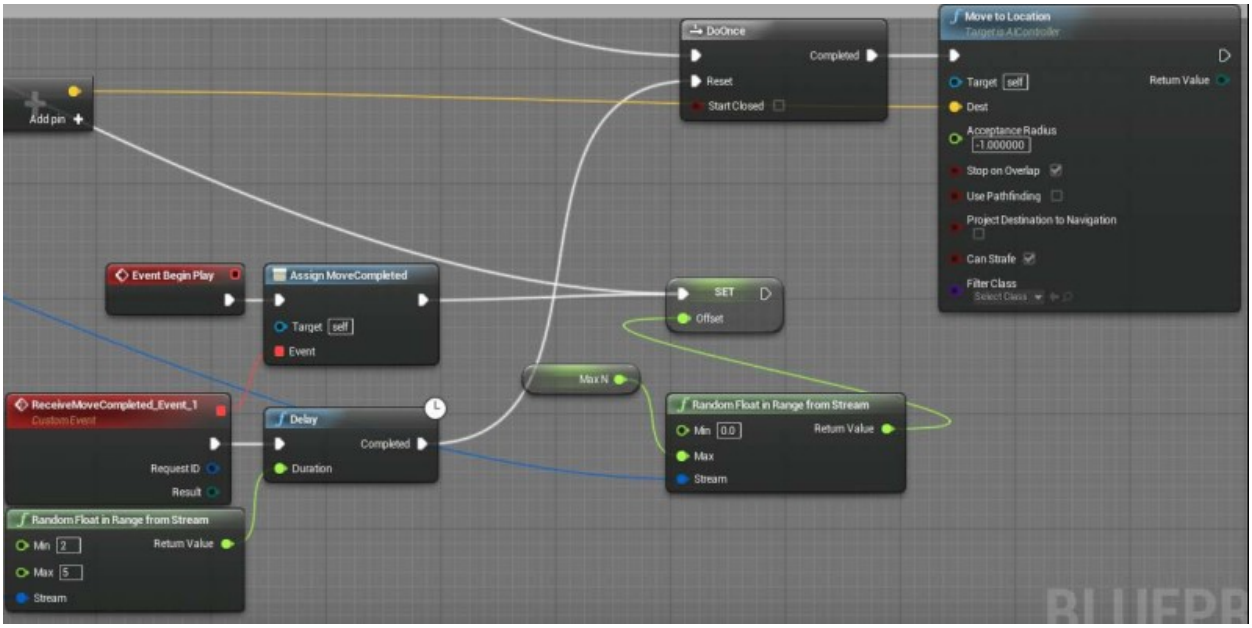
Trở lại hành động

Hãy tập trung trở lại vào *Enemy* EventGraph. Chúng ta sẽ thực hiện scripting để AI của chúng ta đi lang thang. Hãy thực hiện các bước sau:

1. Bắt đầu bằng cách tìm kiếm **Get Controlled Pawn**.
2. Tiếp theo, chúng ta muốn thực hiện **Get Actor Location** trên Actor vừa mới nhận được. Sau đó, vector mà nó trả về sẽ được ghim vào node **Vector + Vector**.
3. Từ đó, chúng ta sẽ tạo node **Random Unit Vector** để tạo một vector ngẫu nhiên từ 0-1 trên mỗi trục riêng lẻ.
4. Chúng ta phải kéo từ biến **Stream** và tạo **Random Float in Range from Stream**. Sau đó, đặt giá trị **Min** thành **-550** và **Max** thành **550**. Sau đó, chúng ta sẽ nhân những gì nó trả về với **Random Unit Vector**.
5. Cuối cùng, từ node nhân vector, chúng ta sẽ cắm vào node **Vector + Vector** mà chúng ta đã tạo ở Bước 2:



6. Bây giờ, chúng ta cần lấy kết quả của node **Vector + Vector** và tạo node **Move to Location** để hướng dẫn Actor hiện tại di chuyển đến node **Move to Location** của vector.
7. Chúng ta sẽ xem lại trạng thái đầu tiên của mình, trạng thái này sẽ kiểm tra các Actor trong bán kính. Sau khi hoàn thành **ForEachLoopWithBreak**, chúng ta sẽ muốn nó thực thi Move to Location. Vì vậy, hãy kéo từ đó và tạo node **DoOnce** trước tiên.
8. Chúng ta sẽ làm điều này vì chúng ta không muốn bot liên tục được yêu cầu di chuyển đến một vị trí; chúng ta muốn nó làm điều này sau một sự chậm trễ ngẫu nhiên. Vì vậy, bây giờ hãy kết nối chân **Completed** với **Move to Location**.
9. Từ đó, chúng ta muốn chỉ định một sự kiện mới. Sự kiện này sẽ thiết lập lại **DoOnce** sau thời gian trễ ngẫu nhiên. Vì vậy, đằng sau **Event Begin Play**, chúng ta sẽ tạo **Assign Move Completed**.
10. Từ **RecieveMoveCompeted_Event**, tạo node **Delay**. Kéo từ **Duration**, chúng ta tìm **Random Float in Range from Stream**. Chúng tôi muốn đặt giá trị **Min** thành **2** và giá trị **Max** thành **5**.
11. Sau khi node **Delay** được tạo, chúng ta sẽ cắm nó chân **Reset** của node **DoOnce** mà chúng ta đã tạo ở Bước 7:



Kết quả

Bây giờ chúng ta có thể thiết lập cơ hội để AI của chúng ta chạy trốn. Script lệnh xác suất này đã được thực hiện ít dự đoán hơn với phần bù biến thể ngẫu nhiên cho phạm vi xác suất. Chúng ta đã hoàn thành điều này bằng cách tạo ra một Finite State Machine, chuyển đổi giữa các trạng thái cần thiết tại bất kỳ thời điểm nào. Vì vậy, script chúng ta đang tiếp cận AI lang thang và khi chúng ta ở trong phạm vi, AI sẽ tấn công hoặc bỏ chạy. Chúng tôi đã mặc định cho nó 25% cơ hội chạy trốn. Trí tuệ nhân tạo chạy trốn và timer thiếp lập lại trạng thái của bot bắt đầu. Vài giây sau, AI trở lại trạng thái lang thang.

Tóm tắt

Trong chương này, bạn đã học về tính ngẫu nhiên và xác suất cũng như cách chúng được thực hiện. Chúng ta đã sử dụng các công cụ trong Unreal Engine để áp dụng những lý thuyết này. Chúng ta hiểu điều này bằng cách sử dụng một trình tạo số giả và chúng tôi đã tính toán về mặt kỹ thuật tính ngẫu nhiên trên một phổ giới hạn được xác định bởi hạt giống. Chúng ta có thể sử dụng những con số không đồng nhất này để tạo các hàm xác suất, cho phép AI tấn công hoặc bỏ chạy. Bạn cũng đã học cách tạo một FSM trong quá trình thiết lập để tạo các trạng thái của AI.

Trong chương tiếp theo, chúng ta sẽ khám phá các kỹ thuật khác nhau để giới thiệu chuyển động cho quân tốt trong trò chơi của chúng ta. Chúng ta sẽ đề cập đến Path Finding, Nav Mesh, EQS và các thành phần liên quan khác ảnh hưởng trực tiếp đến chuyển động.