Course - Lập trình AI cho Unreal Engine

Di Chuyển nâng cao

Tập trung vào các hành vi di chuyển bầy đàn và đi theo con đường nâng cao hơn. Đổ xô cho phép chúng tôi tạo các hành vi nhóm cho một số nhân vật AI.

Tags: Di Chuyển nâng cao

Trong chương này, chúng ta sẽ tập trung vào bầy đàn và các hành vi dò đường nâng cao hơn. Những gì chúng tôi sẽ cố gắng đạt được là thực hiện các hành vi tụ tập để tạo ra di chuyển thực tế cho AI của chúng ta, chẳng hạn như khi bạn cần các tác nhân tránh nhau trong khi tất cả đang di chuyển theo cùng một hướng. Đôi khi, các đại lý của bạn cần phải tìm kiếm một leader, để bạn có thể tạo ra các nhóm đại lý khác nhau. Đầu tiên, chúng ta sẽ thiết lập mọi thứ chúng ta cần để khiến một số con tốt di chuyển trong một cấp độ. Tiếp theo, chúng ta sẽ thêm một số blueprint cho những con tốt này để cung cấp cho chúng khả năng khám phá những nhà lãnh đạo mới. Cuối cùng, chúng ta sẽ giới thiệu hành vi bầy đàn để chúng ta có thể thấy cách AI của chúng ta di chuyển theo nhóm.

Các chủ đề được đề cập trong chương này như sau:

- Thiết lập một blueprint actor cho di chuyển
- Thực hiện hành vi sau
- Thực hiện hành vi tụ tập với các tính năng như tách biệt, gắn kết và xếp hàng.
- Thêm kiểm soát hành vi thông qua UMG

Bạn có thể download toàn bộ fullsource, tại đây.

Thiết lập các agent

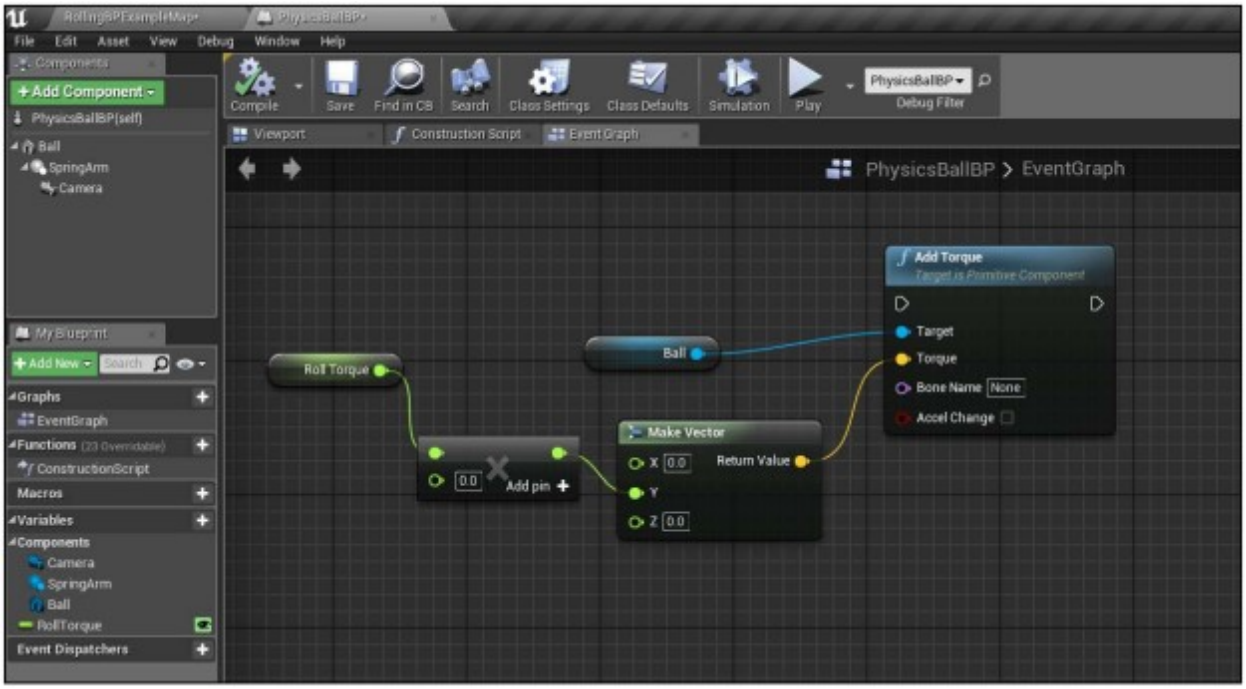
Hãy tạo một dự án mới có tên là *AdvancedMovement* bằng mẫu dự án **Rolling**. Điều đầu tiên chúng tôi muốn làm là tìm class **PhysicsBallBP** của chúng ta và mở sơ đồ EventGraph. Chúng ta có thể làm điều này bằng cách thực hiện các điểm được đưa ra như sau:

1. Sau khi dự án của chúng ta được tải, hãy vào thư mục *Blueprints*.
2. Ở đây, bạn sẽ tìm thấy **PhysicsBallBP**, và nó sẽ đóng vai trò là agent của chúng ta trong chương này. Hãy mở sơ đồ EventGraph của actor này lên editor.

   > **Chú ý:** Chúng ta sẽ giới thiệu hai biến vector mới để giữ hướng hiện tại của agent. Cái còn lại sẽ giữ vị trí mà agent được sinh ra.

3. Bây giờ, hãy loại bỏ mọi logic không cần thiết khỏi ví dụ. Tôi cũng đã loại bỏ các biến sau:
   - The *JumpImpulse* variable
   - The *CanJump* variable

   Tất cả các node Blueprint nằm trong EventGraph không có gì ngoài như ảnh chụp màn hình sau:



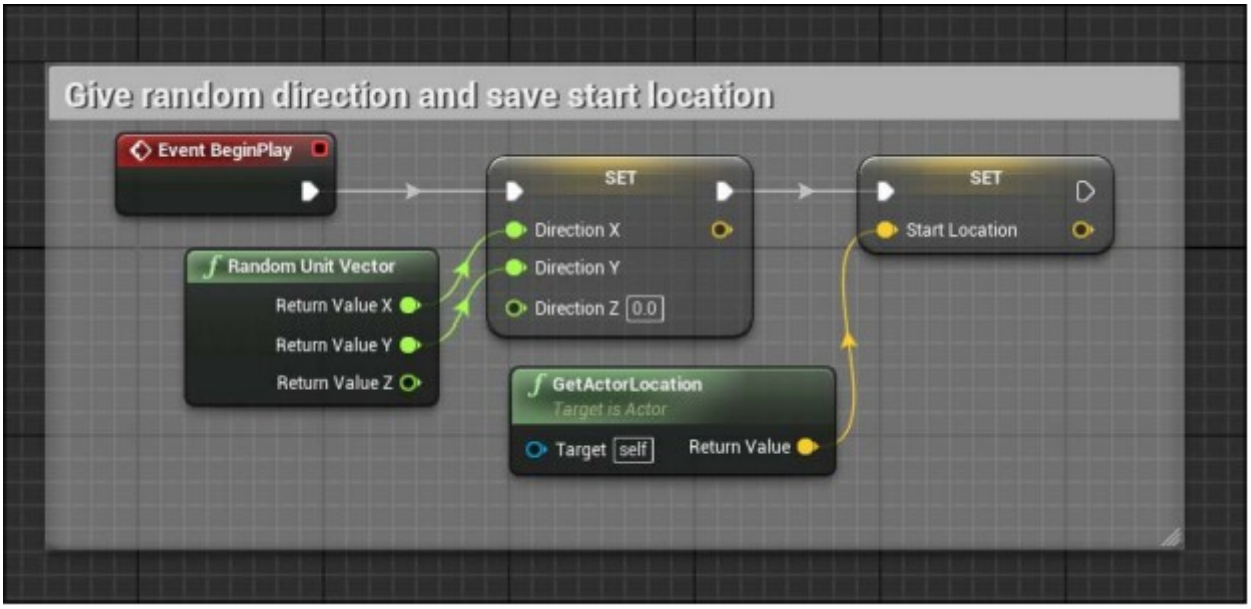
4. Được rồi, bây giờ chúng ta đã có nền tảng bắt đầu, hãy tạo một biến có tên là *Direction* và đặt cho nó kiểu biến **vector**.
5. Tiếp theo, tạo thêm một biến có tên *StartLocation* và đặt nó làm loại biến **vector**.

> **Chú ý:** *StartLocation* sẽ được sử dụng sau này trong khóa học cho nút nhấn **Reset** mà chúng ta phải triển khai.

6. Bây giờ, hãy tìm node **Event Begin Play** và hãy khởi tạo các biến **vector** mới của chúng ta. Đầu tiên, nhấp chuột phải vào khu vực gần sự kiện và tìm **Random Unit Vector**. Tách vector (bằng cách nhấp chuột phải vào chân, nhấp chọn **Split Struct Pin**) khỏi node **Random Unit Vector** vì chúng ta chỉ muốn các giá trị **X** và **Y**. Tiếp theo chúng ta kéo node **SET** *Direction* ra sơ đồ của chúng ta.

> **Chú ý:** Chúng ta chỉ có thể tiến hoặc lùi và sang trái hoặc phải. Các khoảng cách bằng phẳng này được xử lý bởi **X** và **Y**. Nếu **Z** được giới thiệu, nó có khả năng xoay camera về phía trước, điều này có thể không như mong muốn khi quan sát.

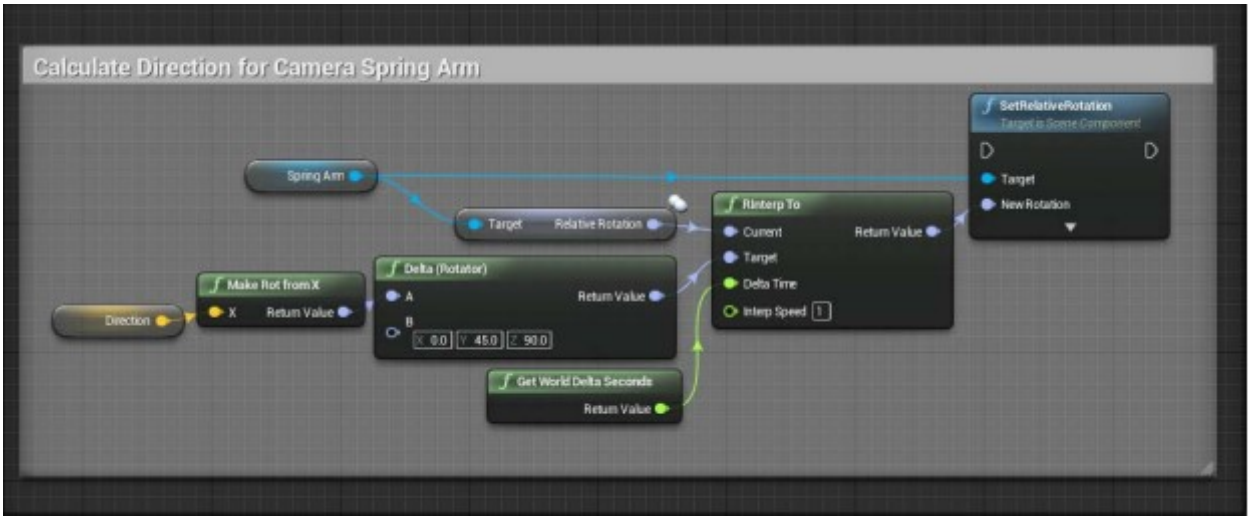
7. Tiếp theo, hãy nhấp chuột phải và tạo một node mới có tên là **Get Actor Location**. Chúng ta phải kéo biến SET vector *StartLocation*. Sau đó, từ **Return Value**, hãy đặt biến **Start Location** của chúng ta. Đánh một bình luận cho logic vừa tạo "**Tạo một hướng mặc định và lưu vị trí bắt đầu**":



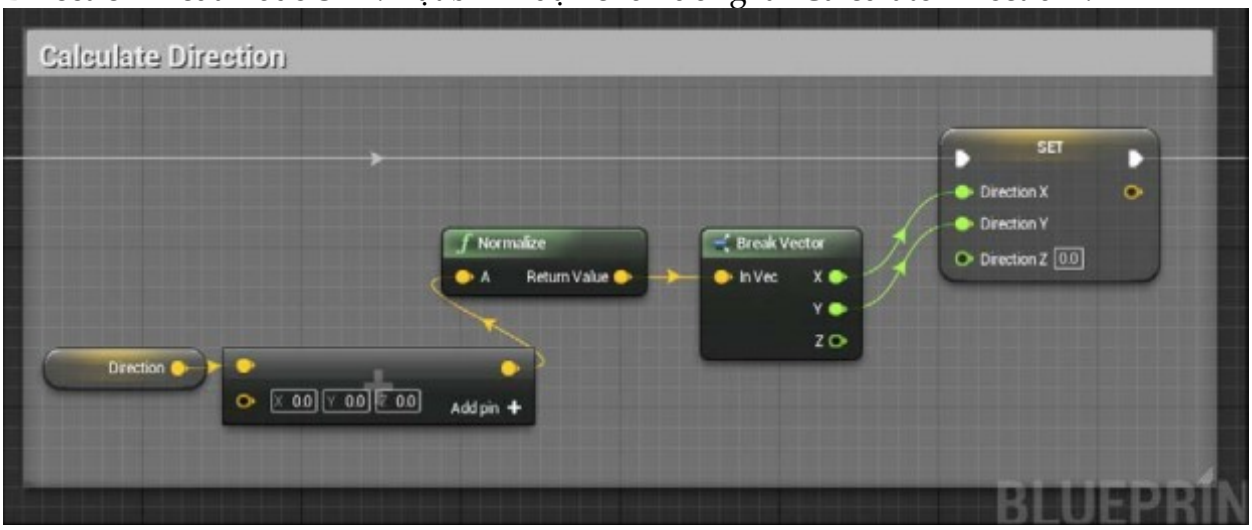
8. Bây giờ, chúng ta cần đặt biến *Direction* thành hướng của biến *SpringArm*. Lý do cho điều này là để máy ảnh luôn hướng về hướng mà chúng ta đang di chuyển trên thế giới. Vì vậy, khi bạn sửa đổi các biến khác nhau, bạn có thể thấy nó có hiệu lực.
9. Kéo biến *SpringArm* của chúng ta vào sơ đồ EventGraph. Sau đó, kéo ra node **SetRelativeRotation**.
10. Sau đó, chúng ta sẽ lấy giá trị **RelativeRotation** hiện tại của biến *SpringArm* để nội suy nó giữa các hướng mà chúng ta sẽ đối mặt. Nó tạo ra sự chuyển tiếp suôn sẻ khi cập nhật góc quay của máy ảnh.
11. Bằng cách lấy từ chân **RelativeRotation** hiện tại của biến *SpringArm*, chúng ta sẽ tạo node **RInterp To**.
12. Tiếp theo, chúng ta sẽ lấy biến *Direction* của mình và chuyển đổi hướng này thành hướng xoay bằng cách sử dụng **Make Rot from X**.
13. Chúng ta sẽ thay đổi công cụ quay vòng này bằng một offset. Tôi đã tính toán đây là offset xoay cục bộ của lò xo. Với hướng theo **A**, chúng ta nên đặt **B** có **0,0** Roll, **45,0** Pitch và **90,0** Yaw.
14. Bây giờ, hãy kết nối chân **Return Value** với chân **Target** của node **RInterp To** của chúng ta.
15. Bây giờ, nhấp chuột phải, tìm kiếm **Get World Delta Seconds** và cắm cái này vào chân **Delta Time** trên node **RInterp To** của chúng ta.
16. Chân **Return Value** sẽ được cắm vào **New Rotation** của node **SetRelativeRotation** mà chúng ta đã tạo trước đó.

> **Chú ý:** Từ đây, chúng ta sẽ cập nhật hướng của chúng ta. Lý do cho điều này là chúng ta sẽ tiếp tục thêm vào hoạt động này để chuẩn hóa hướng của chúng ta và nhận một giá trị trong phạm vi mà chúng ta mong đợi.

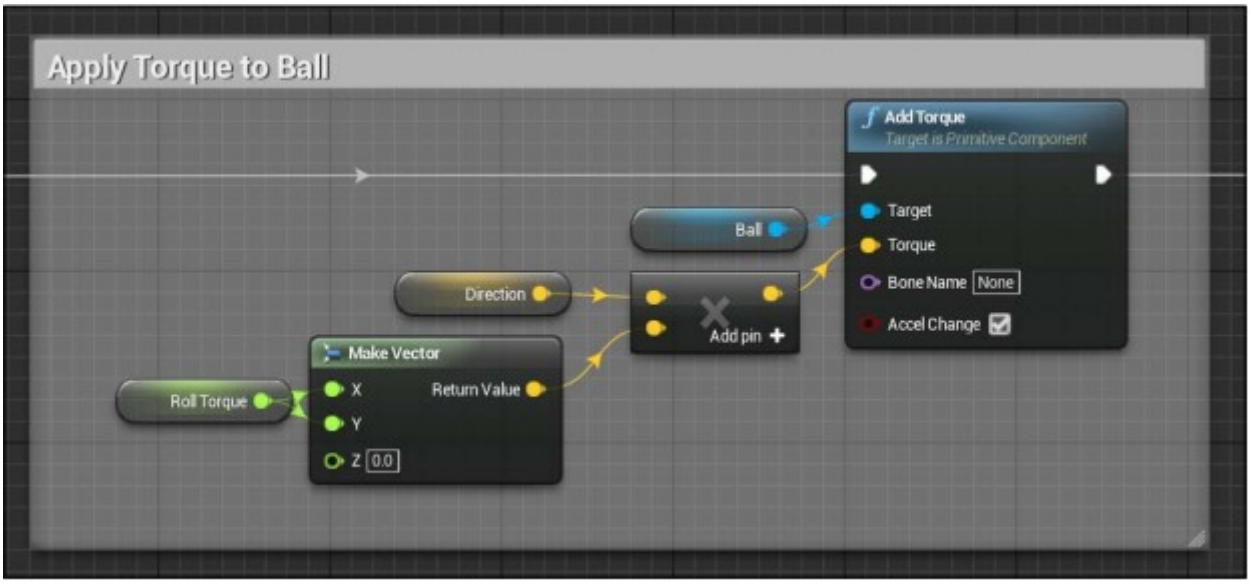
Đặt bình luận cho logic này là "**Calculate Direction for Camera Spring Arm**":



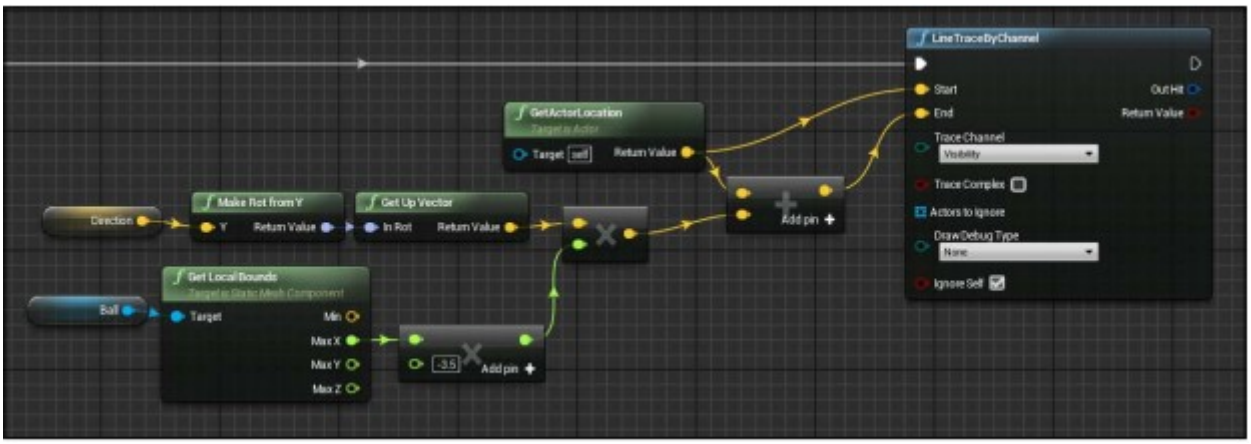
17. Bây giờ, để tiếp tục blueprint tiếp theo, hãy kéo biến **Direction** của chúng ta vào sơ đồ. Kéo từ biến và tìm **Vector + Vector**. Từ đây, chúng ta muốn chuẩn hóa vector. Cuối cùng, hãy bung vector với **Break Vector**.
18. Tiếp theo, hãy kéo biến **Direction** và tạo biến **SET**. Hãy tách biến thiết lập **Direction** bằng **Break Vector**. Sau đó, cắm cả hai biến float **X** và **Y** vào các biến **Direction X** và **Direction Y** của node **SET**. Đặt bình luận cho hướng là "Calculate Direction":



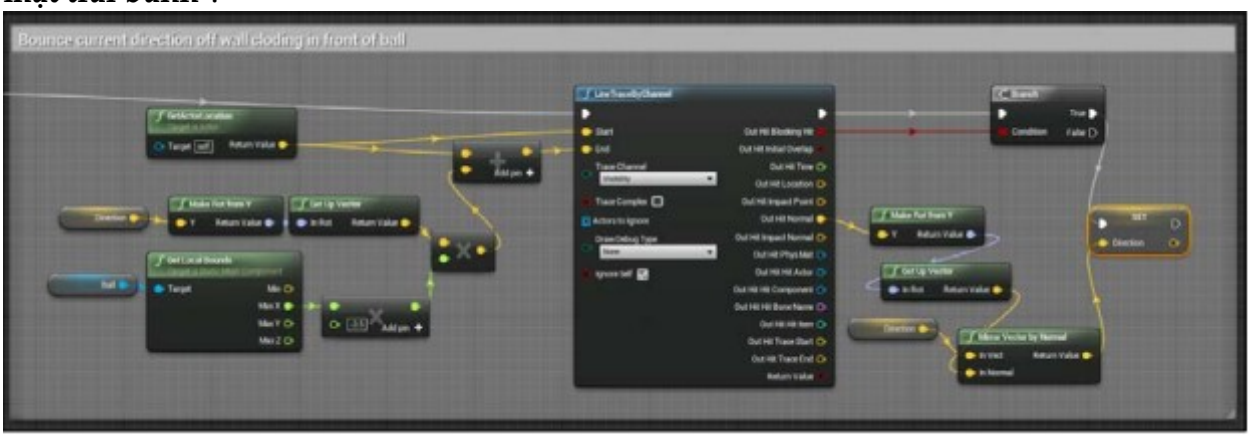
19. Chúng ta cần kéo một biến **Ball** và gọi một node có tên **Add Torque**.
20. Để tính giá trị **Torque**, chúng ta muốn lấy biến *Direction*. Lấy biến **Roll Torque** và nhân nó với vector của *Direction*. Kết quả của việc này sẽ được cắm vào chân **Torque** của node **Add Torque**. Đặt bình luận **Apply Torque to Ball**:



21. Bước tiếp theo là tạo một hàm sẽ quét đằng trước biến *Ball*. Nếu một dấu vết bị đụng trúng, nó có nghĩa là chúng ta đã va phải một bức tường. Từ đây, chúng ta sẽ sử dụng **Hit Result** để tạo phản xạ bật khỏi bức tường dựa trên **Hit Normal**. Hãy xem điều này trong thực tế.
22. Từ node cuối cùng của chúng ta, hãy tạo một node mới gọi là **LineTraceByChannel**. Từ đây, chúng ta sẽ tạo một node mới bằng cách nhấp chuột phải và tìm kiếm **Get Actor Location**. Cắm **Return Value** vào chân **Start** của node mới được tạo của chúng ta.
23. Lấy một bản sao khác của biến *Direction* của chúng ta và đặt nó vào sơ đồ. Kéo từ *Direction* của chúng ta và nhấp chuột phải vào node mới có tên **Make Rot from Y**. Từ chân **Return Value** của node **Make Rot from Y**, kéo dây để tạo node mới có tên **Get Up Vector**.
24. Đưa sơ đồ nhận một biến cho vector **Ball** của chúng ta và sau đó kéo ra một node mới có tên là **Get LocalBounds**. Bây giờ, tách vector **Max** và từ **Max X**, kéo dây vào một nút mới, **Float * Float**.
25. Bây giờ, biến này sẽ lấy kích thước của chúng ta và sử dụng nó để xác định khoảng cách về phía trước bên ngoài hình cầu. Trong ví dụ của chúng ta, chúng ta sẽ chia tỷ lệ theo **-3.5**. Chúng ta sẽ đảo ngược tỷ lệ của chúng ta vì vector chúng ta sẽ nhân lên có liên quan đến quả bóng chứ không phải thế giới.
26. Trong node **Float * Float**, hãy đặt giá trị **-3.5** vào chân không được nối dây.
27. Từ node **Get Up Vector** của chúng ta, hãy nhân nó với kết quả **Float * Float** từ bước trước đó. Bây giờ, chúng ta cần nhấp chuột phải và tìm node **Vector + Vector**. Thêm vector này vào vector **GetActorLocation** của chúng ta. Sau đó, cắm kết quả vào chân **End** của node **LineTraceByChannel** của chúng ta:



28. Từ node **LineTraceByChannel** của chúng ta, hãy chia nhỏ **Out Hit**(Click phải chuột, nhấp **Split Struct Pin**). Sau đó, tìm chân **Out Hit Blocking Hit** và tạo một nhánh **Branch** từ kết quả **Boolean**. Hãy kéo vector *Direction* của chúng ta vào sơ đồ và đặt biến **SET**. Cái này phải được kết nối với chân đầu ra **True** của node **Branch** đã tạo trước đó.
29. Từ chân **Out Hit Normal** của chúng ta, nhấp chuột phải và tạo node **Make Rot from Y**. Bây giờ, chúng tôi muốn lấy từ phần trả về này, nhấp chuột phải và tạo node **Get Up Vector**. Điều này đưa normal từ thế giới đến không gian hướng tương đối. Cuối cùng, chúng ta muốn lấy vector *Direction* để lấy biến.
30. Kéo từ biến *Direction* của chúng ta và thả ra để tìm **Mirror Vector by Normal**. Hãy cắm node **Get Up Vector** từ bước trước đó vào chân **InNormal** của **Mirror Vector by Normal**. Kết quả của node này được cắm vào chân **Direction** của biến **SET** được thực hiện trên câu lệnh **True**.
    Đặt bình luận cho logic này là "**Bật ngược hướng hiện tại ra khỏi bức tường đập vào mặt trái banh**":

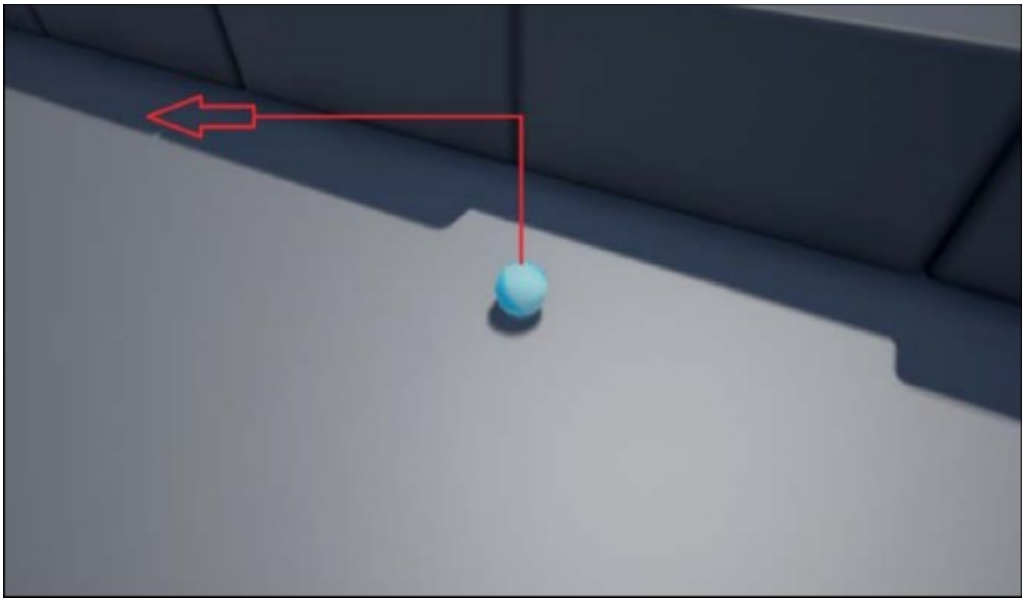


Complete blueprint setup for Bounce current direction off wall

Bây giờ, class hoặc agent PhysicsBallBP của chúng ta đã được cập nhật để xử lý vector chỉ hướng, chúng ta sẽ khởi tạo nó với các giá trị ngẫu nhiên. Nó sẽ liên tục di chuyển về phía trước và bật ra khỏi các bức tường khi actor đến đủ gần điểm va chạm.

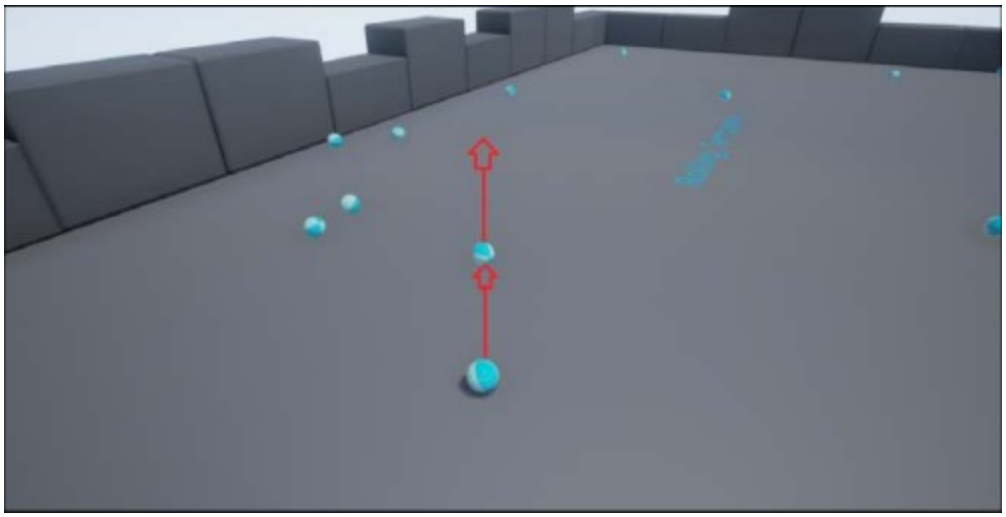
Quan sát agent

Chúng ta muốn nhấn **Play** và đảm bảo rằng agent của chúng tôi thoát khỏi các bức tường. Sau khi bạn nhấn **Play**, nó sẽ chơi và quân tốt mà bạn sinh ra sẽ tiến lên vĩnh viễn. Sau đó, chúng tôi sẽ phản chiếu hướng bình thường sau khi xảy ra va chạm:



Theo sau agent

Chúng ta muốn Agent của mình đi theo agent lãnh đạo của mình theo bất kỳ hướng nào. Trong khi vẫn tuân theo logic khác để bật ra khỏi tường và hành vi di chuyển mà chúng tôi dự định giới thiệu, chúng ta sẽ tạo một ví dụ cho thấy điều này sẽ ảnh hưởng đến di chuyển của agent như sau:



Bước này yêu cầu chúng ta tạo một vài biến và một hàm. Chúng ta muốn giữ các actor Follower và Leader của mình khi hoạt động trong một thiết lập miễn phí. Chúng ta cũng cần *LeaderDirection* để giữ hướng mà agent sẽ di chuyển. Cuối cùng, chúng ta có thể đánh dấu các

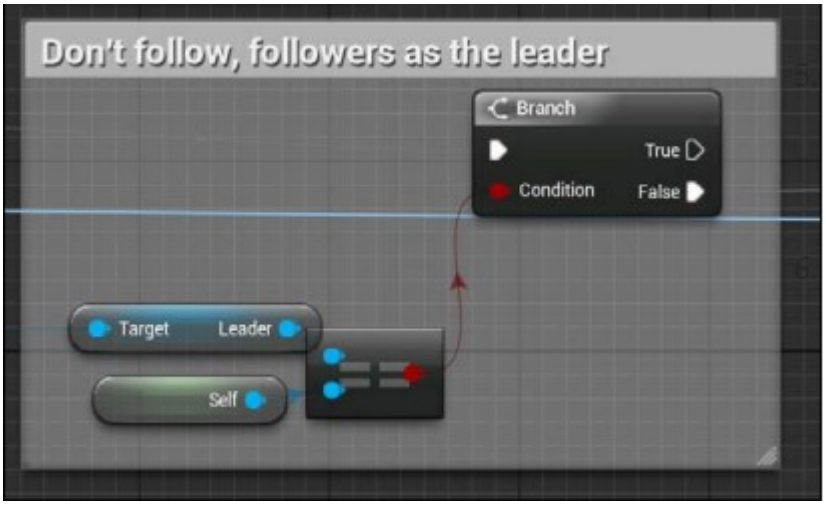
agent là follower hoặc leader bằng cách bật hoặc tắt Boolean được gọi là *isLeader*.

Với những biến số này, chúng ta có thể theo dõi và ngăn chặn các nhà lãnh đạo đi theo những người follower, điều này có thể tạo ra hành vi di chuyển thú vị nhưng không phải là điều chúng tôi dự định thực hiện trong cuộc biểu tình này.

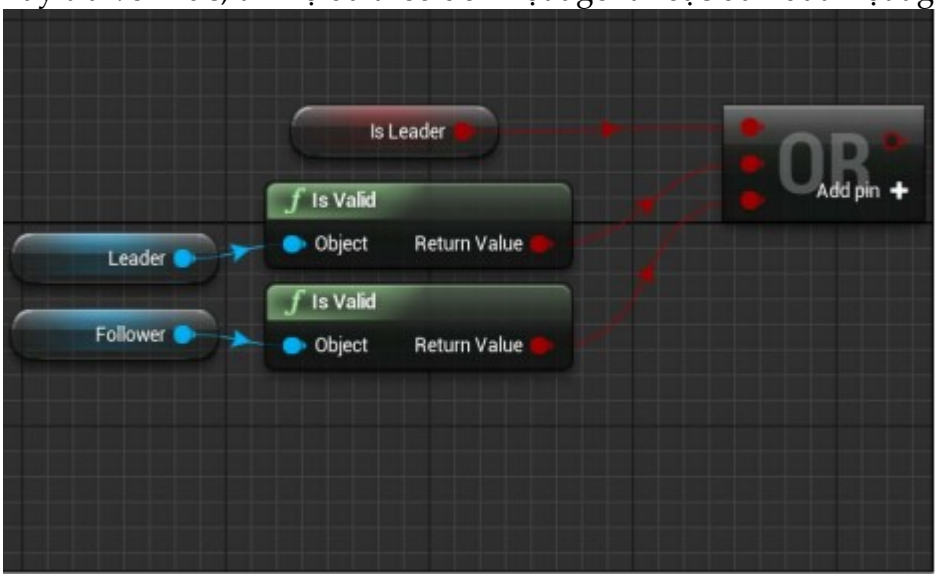
Theo sau hoặt dẫn đầu

Hãy mở class *PhysicsBallBP* của chúng ta và xem EventGraph. Tìm nơi chúng ta sẽ cập nhật chân *RelativeRotation* cho biến *Spring Arm* của chúng ta. Giữa cái này và **Calculate Direction**, chúng ta sẽ thêm bộ hướng dẫn mới này:

1. Trước tiên, chúng ta cần tham khảo actor nào gần agent của chúng ta. Chúng ta có thể làm điều này bằng node **SphereOverlapActors**. Nhấp chuột phải vào sơ đồ và tìm kiếm cái này.
2. Bây giờ, hãy lấy node **Get Actor Location**. Nhấp chuột phải, tìm cái này và cắm nó vào **Sphere Pos** của node mà chúng ta đã tạo ở bước trước. Tiếp theo, đặt giá trị **Sphere Radius** thành **300**. Bây giờ, hãy kéo từ **Object Types** và tạo node **Make Array**. Từ đó, chúng ta sẽ gán các **Object Types** với **Physics Body**. Tiếp theo, chúng ta muốn thiết lập chân **Actor Class Filter** của node **SphereOverlapActors** thành **PhysicsBall** để ngăn các kết quả không mong muốn. Cuối cùng, tạo một node **Make Array** khác, nhấp chuột phải vào sơ đồ và tìm kiếm nó. Sau đó, điền phần tử đầu tiên có tham chiếu đến **Self**. Sau đó, cắm mảng này vào chân **Actors To Ignore**. Đặt bình luận vào khu vực này câu "**Tìm kiếm gần xung quanh PhysicsBall**".
3. Bây giờ, hãy tập trung vào mảng **Out Actors** của node **SphereOverlapActors**. Nhấp chuột phải vào sơ đồ và tìm kiếm **For Each Loop**. Hãy kéo từ **Array Element** và chuyển cái này sang *PhysicsBall* bằng node **Cast to PhysicsBall**.
   Chúng ta phải biết liệu agent mà chúng ta cố gắng theo dõi có đang theo dõi chúng ta hay không. Nếu không, chúng ta có thể trải nghiệm hành vi di chuyển không mong muốn.
4. Hãy tạo ra biến *Leader* trong **My Blueprint**. Từ node **Cast to PhysicsBall**, hãy kéo chân **As Physics Ball BP** tìm biến **Leader** và so sánh nó với **Self** bằng (==). Lấy chân **Condition** và tạo một node **Branch**. Nếu là **False**, thì ngay bây giờ chúng ta cần kiểm tra xem mình có phải là **Leader** hay không và liệu chúng ta có **Leader** hay **Follower** hay không.
   Trước khi chúng ta tiếp tục, hãy đặt bình luận nhánh này với "**Không theo dõi, những người theo dõi với tư cách là người dẫn đầu**":

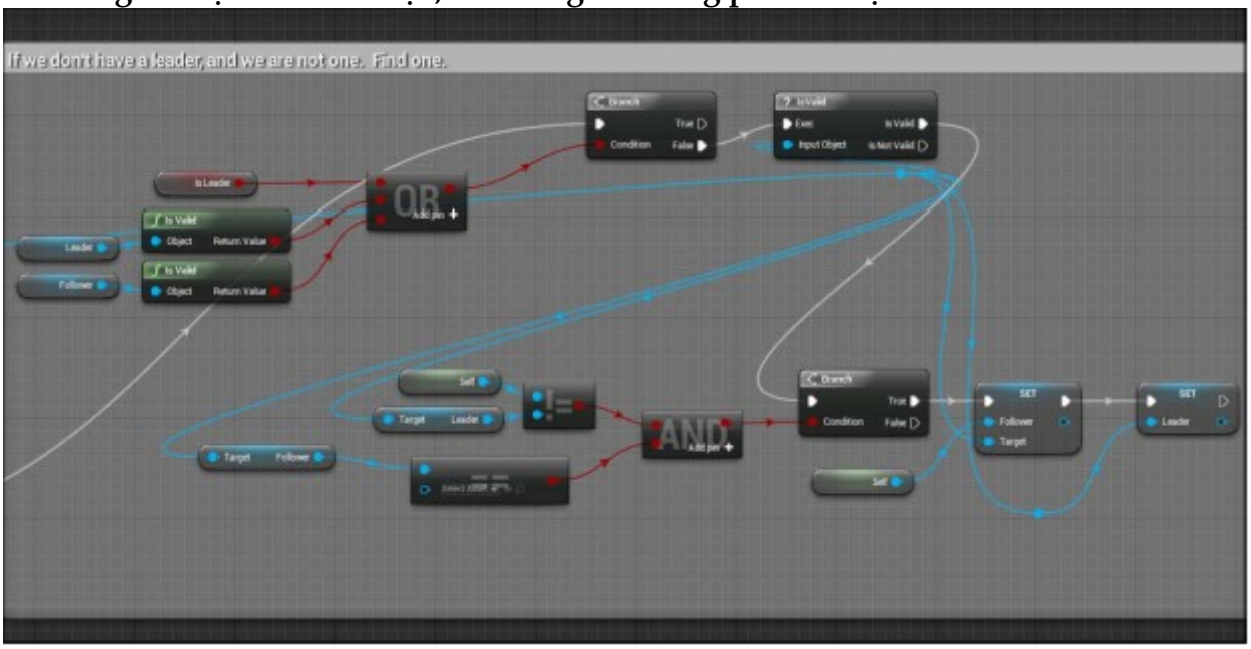


5. Vì vậy, hãy tìm biến *isLeader* và nếu không thì tạo biến này trong **My Blueprint**, sau đó kéo vào trong sơ đồ. Tiếp theo, tìm kiếm **Leader** và kiểm tra xem nó có hợp lệ hay không bằng cách tạo node **IsValid** (là một function). Chúng ta phải làm điều tương tự cho một biến khác gọi là **Follower**. Bây giờ tạo ra biến **Follower** trong **My Blueprint**.
6. Bây giờ, hãy kết nối cả ba điều kiện **Boolean** vào một node **OR**. Điều này sẽ trả về **True** nếu bất kỳ điều kiện nào của chúng ta trả về **True**. Nếu bất kỳ điều kiện nào trong số này trả về **True**, thì họ đã theo dõi một agent hoặc dẫn dắt một agent khác:

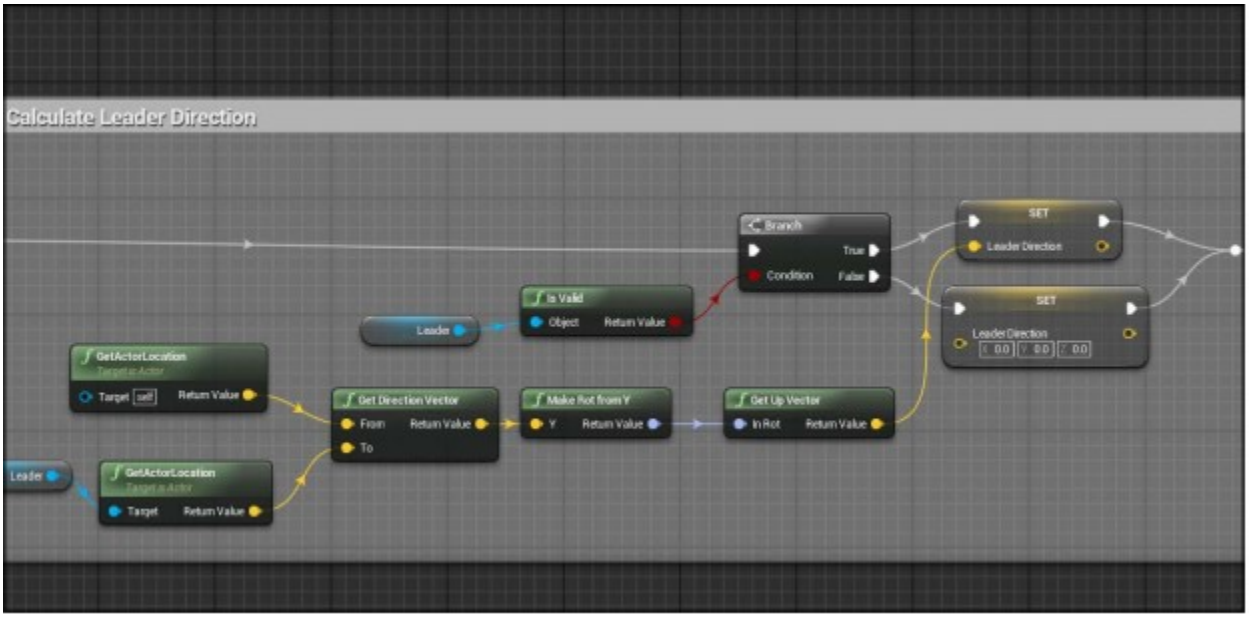


7. Từ node **OR** này, hãy tìm kiếm một node **Branch** mới và kết nối nó với chân **Condition**. Bây giờ, nếu giá trị node **Branch** này là **False**, hãy tạo một node có tên là **IsValid** (macro). Đối tượng mà chúng tôi muốn kiểm tra là quá trình truyền cast mà chúng tôi đã thực hiện ở bước 3. Kéo node này và tạo một node định tuyến lại gần node **IsValid** (macro) để các node khác có thể dễ dàng sử dụng node này.
8. Nếu **IsValid** (macro) trả về **Is Valid**, chúng ta sẽ tạo một node **Branch**. Chúng ta muốn kiểm tra hai điều kiện là **True** với cổng logic **AND** trong câu lệnh này.
9. Trước tiên, hãy lấy biến **Leader** từ trạm tiếp tuyến. Sau đó, chúng ta sẽ so sánh nó với **Self** với **not equal** (!=). Tạo cổng logic **AND** và kết nối chân đầu tiên với kết quả của điều kiện không bằng (!=).

10. Tiếp theo, hãy lấy biến **Follower** từ trạm tiếp tuyến. Sau đó, so sánh node này bằng cách sử dụng bằng (==). Tiếp theo, kết nối cổng này với cổng logic **AND** ở bước trước.
11. Cuối cùng, kết nối cổng logic **AND** này với node **Branch**. Nếu node này trả về **True**, thì hãy đặt chân **Follower** từ trạm tiếp tuyến. Cắm tham chiếu đến **Self** vào chân **Follower**. Sau đó, đặt biến **Leader** cục bộ từ định tuyến lại. Đặt bình luận khu vực này **Nếu chúng ta không có một nhà lãnh đạo, và chúng ta không phải là một**:



12. Bây giờ, hãy chuyển đến chân **Completed** của node **For Each Loop**. Hãy kéo một biến **Leader** vào sơ đồ và kiểm tra xem nó có hợp lệ hay không bằng cách sử dụng **Is Valid** (là một hàm). Bây giờ, hãy tạo một node **Branch** và cắm chân **Condition** từ node trước đó vào.
13. Nếu node **Branch** trả về **False**, chúng ta sẽ đặt giá trị **Leader Direction** (Nếu chưa có thì tạo ra) của biến **SET** thành **0.0**, **0.0** và **0.0**. Nếu node **Branch** của chúng ta trả về **True**, chúng ta sẽ tính toán hướng tới biến **Leader** của chúng ta.
14. Tạo một node mới có tên **GetActorLocation** và từ phần trả lại, chúng ta sẽ tạo **Get Direction Vector**. Bây giờ, hãy nhấp chuột phải vào sơ đồ và tìm kiếm biến cục bộ **Leader**. Sau đó, tạo node **GetActorLocation** từ biến **Leader** và cắm node này vào chân **To** của **Get Direction Vector**.
15. Từ node **Get Direction Vector**, kéo **Return Value** và tạo **Make Rot from Y**. Tiếp theo, chúng ta sẽ tạo **Get Up Vector** từ lần xoay trước. Sau đó, từ **Return Value**, nó sẽ đi vào chân **Leader Direction** của biến **SET**.
16. Cả hai biến cục bộ **Leader Direction** sẽ dẫn đến khu vực được nhận xét là **Calculating Direction**. Chúng ta cũng phải đi đến **Calculate Direction** và thêm **Leader Direction** vào phép tính hướng cuối cùng.
17. Đặt bình luận cho khu vực này là "**Tính Leader Direction**":

18. 

> **Chú ý:** Bây giờ, nếu nhấn **Play**, chúng ta có thể xem điều gì sẽ xảy ra. Các nhóm nhỏ bắt đầu hình thành khi những người theo dõi tìm thấy những nhà lãnh đạo tiềm năng bằng cách lướt qua nhau.

Hành vi rẽ hướng: đổ xô

Đổ xô là một hành vi rẽ hướng kết hợp Sự tách rẽ, Sự gắn kết và Sự xếp hàng. Hành vi tách biệt sẽ tránh các agent lân cận khác. Hành vi gắn kết giữ các agent theo một nhóm. Hành vi căn hàng sẽ tính toán trung bình hướng về phía trước bằng cách căn chỉnh với các agent lân cận.

Những gì chúng tôi sẽ làm ở đây là sao chép hành vi rẽ hướng, đổ xô, trong blueprint. Chúng ta cũng sẽ sử dụng UMG để hỗ trợ điều chỉnh trọng số cho từng hành vi. Hãy bắt đầu ngay bây giờ và tạo các biến mà chúng ta sẽ cần trong phần này của chương.

Xô đẩy các agent

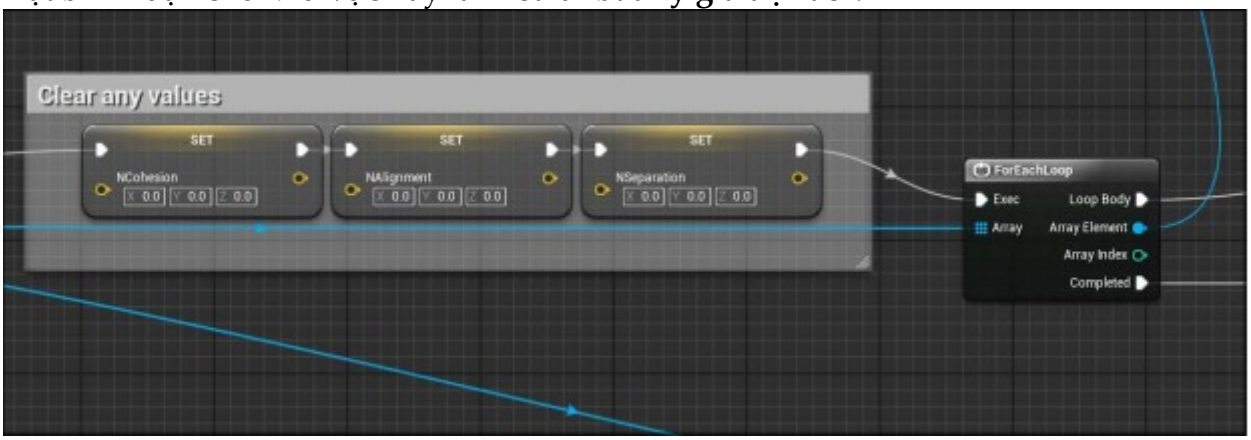
Đầu tiên chúng ta phải bắt đầu bằng cách tạo ra các biến số cần thiết để tính toán các hành vi cá nhân. Sau đó, chúng ta phải thêm kết quả để chuẩn hóa hướng chuyển tiếp cuối cùng cho agent của chúng ta.

Hãy tập trung vào chế độ trò chơi **RollingGameMode** của chúng ta và thêm ba biến global mới mà chúng ta sẽ phải sử dụng ở phần sau của chương này. Tương quan với ba hành vi, chúng ta cần tạo *GlobalAlignment*, *GlobalCohesion* và *GlobalSeparation* bằng cách thực hiện các bước sau:

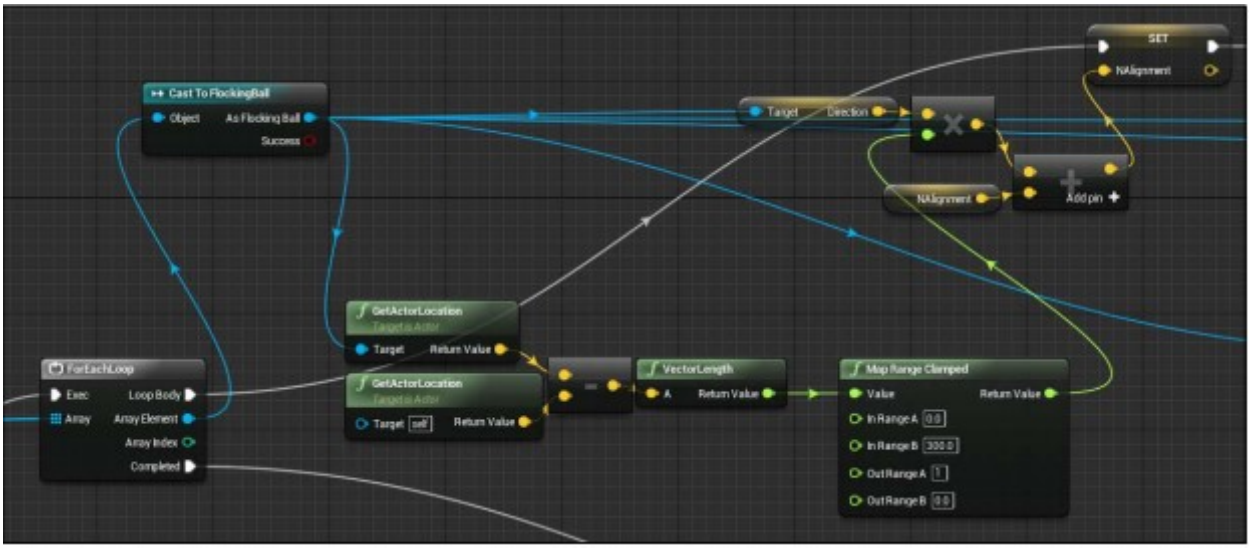
1. Mở **RollingGameMode** và tập trung vào EventGraph. Sau đó, từ đó, tạo ba biến dưới dạng kiểu **Float** với giá trị mặc định là **0.0**.

   Chú ý: Hãy tập trung trở lại sơ đồ sự kiện **FlockingBall** của chúng ta.

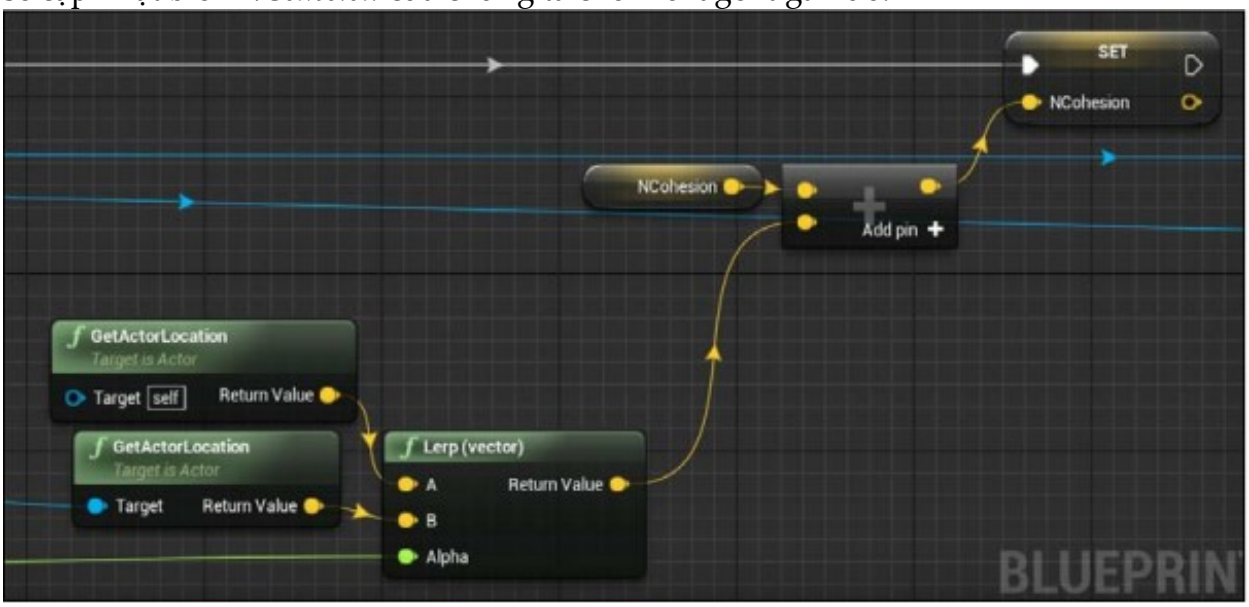
2. Chúng ta cần tạo ba biến vector: **NCohesion**, **NAlignment**, và **NSeperation**.
3. Tập trung vào node **SphereOverlapActors** trước **ForEachLoop**. Tại đây, chúng ta sẽ xóa mọi giá trị trước đó của ba vector mà chúng ta vừa tạo. Chúng ta sẽ làm điều này cho từng hành vi đổ xô của chúng ta, đặt nó thành **0,0,0**.
   Đặt bình luận cho khu vực này là "**Xóa đi bất kỳ giá trị nào**":



4. Tiếp theo, trước phần đã nhận xét trước đó có tên **Không theo dõi, những người theo dõi với tư cách là người dẫn đầu**, chúng ta muốn thêm khoảng cách giữa phần này và chân **Loop Body** từ node **ForEachLoop**.
5. Đầu tiên, chúng ta cần tính toán *NAalignment*, đây là hành vi chịu trách nhiệm điều khiển các agent gần đó theo cùng một hướng. Ta sẽ có được vector chỉ phương trung bình của các agent lân cận.
6. Bây giờ, chúng ta sẽ đặt một biến cục bộ, *NAalignment* và chúng ta sẽ thực hiện việc này bằng cách trước tiên lấy biến *Direction* của chúng ta từ node **Cast to PhysicsBallBP**. Bây giờ, chúng ta sẽ cần nhân giá trị *Direction* với cường độ dựa trên khoảng cách của agent này với các agent của chúng ta.
7. Trước tiên chúng ta cần tính độ dài vector giữa chúng ta và agent. Chúng ta sẽ làm điều này bằng cách trừ vị trí của chúng ta khỏi vị trí của agent khác. Sau đó, chúng ta sẽ nhận được giá trị **VectorLength** từ kết quả. Tiếp theo, chúng ta sẽ tạo **Map Range Clamped** và cắm kết quả từ nút **VectorLength** vào chân **Value**.
8. Bây giờ, để định cấu hình node **Map Range Clamped** này, hãy đặt **In Range B** thành **300.0** và sau đó đặt **Out Range A** thành **1.0**. Điều này sẽ dẫn đến cường độ đầy đủ khi nó ở mức 0 đơn vị và không có cường độ nào vượt quá 300 đơn vị. Giá trị trả về này bây giờ cần được nhân với giá trị *Direction*.
9. Sau đó, chúng ta sẽ thêm (+) kết quả của điều này vào **NAalignment** và đặt kết quả này trở lại biến **NAlignment** cục bộ:



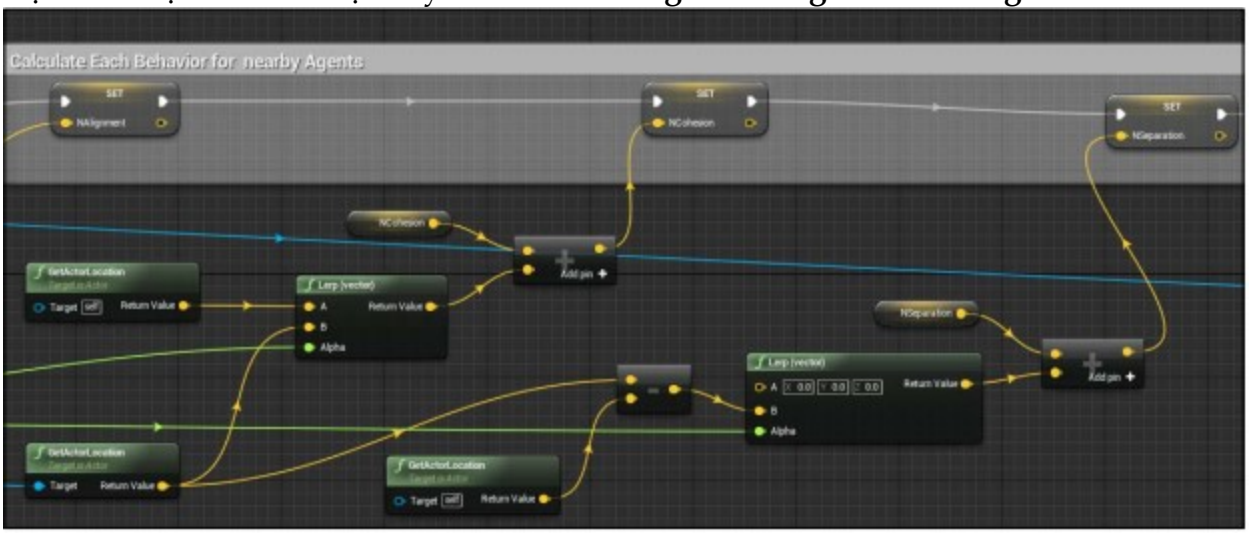
10. Bây giờ, chúng ta phải tính toán Cohesion (sự gắn kết), là hành vi chịu trách nhiệm điều khiển agent hướng tới trung tâm của các agent gần đó. Đây là hướng đến trung tâm đám đông tác nhân.
11. Đầu tiên, hãy tạo một node **GetActorLocation** (A) và một node **Lerp** (vector) kết nối chân **A** và node **GetActorLocation** (B) từ node reroute. Giá trị **Alpha** phải là **Return Value** của **Map Range Clamped** từ bước 6.
12. Sau đó, chúng ta muốn thêm **Return Value** của node **Lerp** (vector) vào *NCohesion*. Tiếp theo, chúng ta sẽ đặt biến *NCohesion* cục bộ của mình với kết quả cuối cùng. Bây giờ nó sẽ cập nhật biến *NCohesion* của chúng ta cho mỗi agent gần đó:



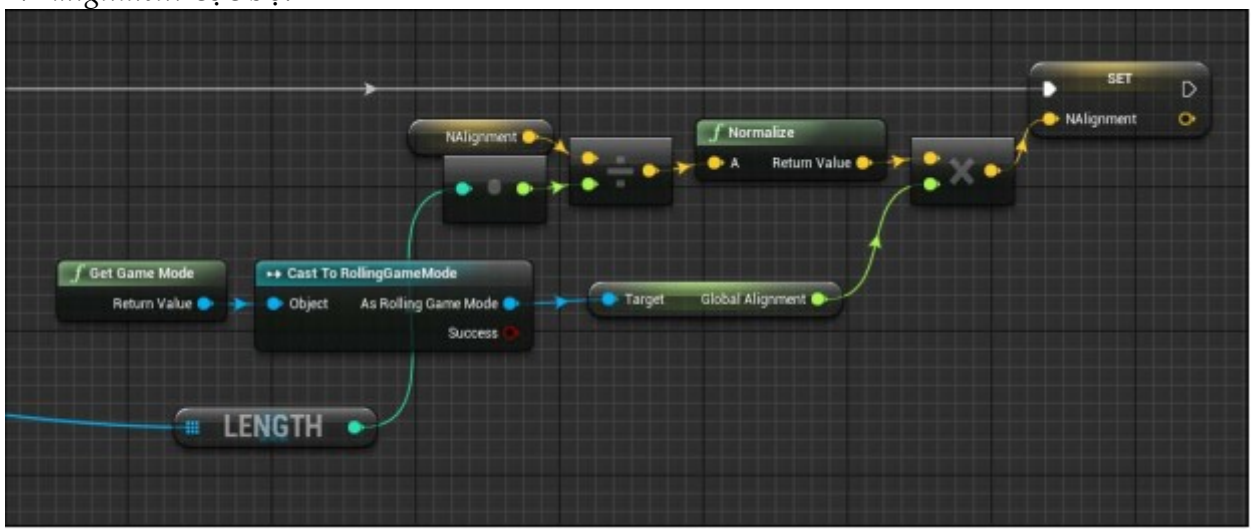
13. Hành vi cuối cùng để tính toán là Separation (Tách nhóm); nó sẽ chịu trách nhiệm buộc agent tránh xa các agent gần đó. Đây là vector *Direction* từ agent của chúng ta đến agent khác.
14. Trước tiên, chúng ta sẽ trừ node GetActorLocation với node GetActorLocation của agent của chúng ta. Sau đó, nó sẽ là chân **B** trên node **Lerp** (vector). Chúng ta sẽ quay lại **Return Value** của node **Map Range Clamped** và cắm node này vào chân Alpha của

node **Lerp** (vector). Chúng tôi sẽ để trống **A**.

15. Sau đó, chúng ta phải thêm **Return Value** của **Lerp** (vector) vào *NSeparation*. Cuối cùng, chúng ta sẽ đặt biến *NSeparation* cục bộ của mình với kết quả tính toán được.
    Đặt bình luận cho khu vực này "**Tính toán từng hành vi gần bởi các Agent**":



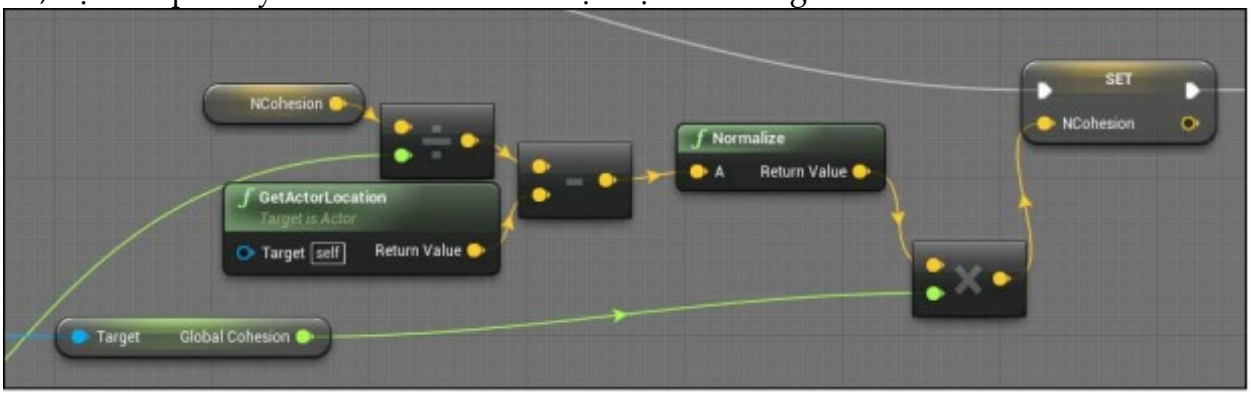
16. Tập trung vào chân **Completed** từ biến **Loop Body**. Bây giờ, chúng ta phải hoàn thành việc tính toán ba hành vi.
17. Hãy lấy độ dài của mảng được trả về bởi **SphereOverlapActors** để chúng ta biết cần chia bao nhiêu để tạo ra giá trị trung bình. Sau đó, chúng ta sẽ chuyển đổi **Int** này thành **Float**. Từ đó, chúng ta có thể chia biến *NAalignment* và sau đó normalize (chuẩn hóa) kết quả bằng cách sử dụng node **Normalize**. Sau đó, chúng ta phải lấy các biến toàn cục mà chúng ta đã xác định trong **RollingGameMode**.
18. Tạo node **Get Game Mode** và truyền nó tới RollingGameMode bằng cách nối dây đến node **Cast To RollingGameMode** để có quyền truy cập vào các biến của chúng ta. Tiếp theo, chúng ta sẽ lấy biến **GlobalAlignment** từ **RollingGameMode** và nhân nó với kết quả **Normalize** từ bước trước. Sau đó, kết quả của cái này phải được đặt vào biến *NAalignment* cục bộ:



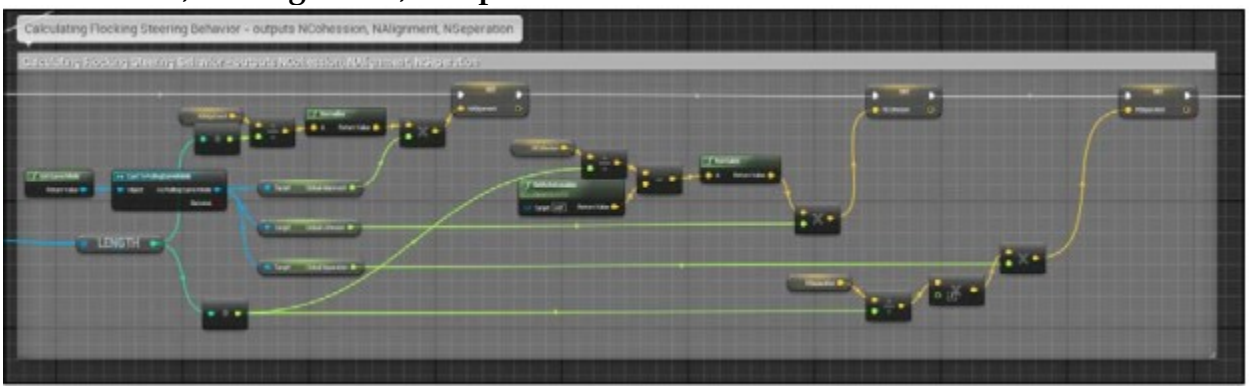
19. Tiếp theo, chúng ta phải tính toán Cohesion (Sự gắn kết). Từ đây, chúng ta phải chia

biến *NCohesion* cục bộ cho độ dài mảng và sau đó trừ nó khỏi node **GetActorLocation** của chúng ta. Sau đó, chúng ta sẽ áp dụng function **Normalize** để cung cấp cho chúng ta kết quả chúng ta cần.

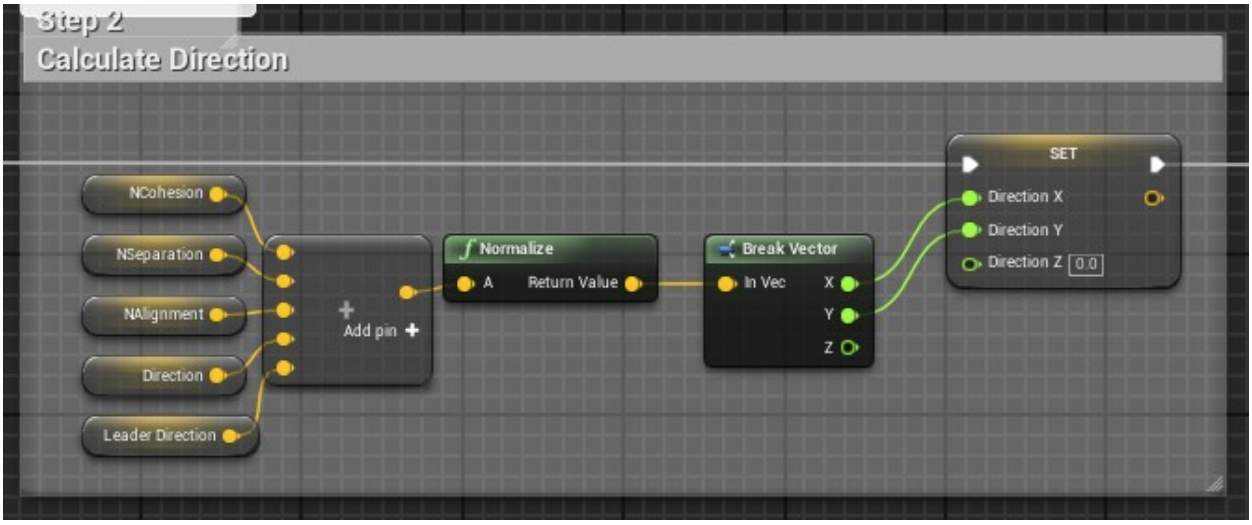
20. Tiếp theo, chúng ta sẽ nhân kết quả với biến *GlobalCohesion* từ **RollingGameMode**. Sau đó, đặt kết quả này vào biến *NCohesion* cục bộ của chúng ta:



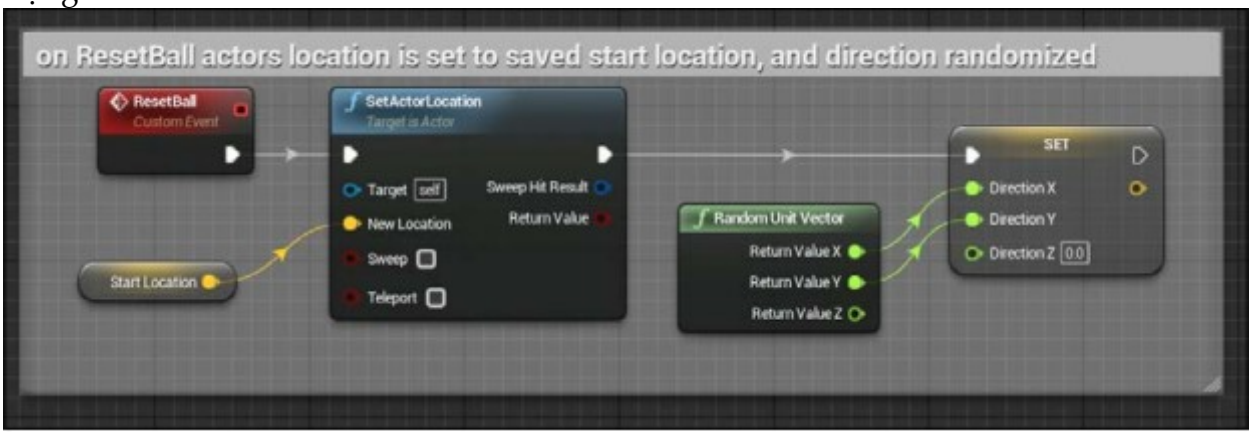
21. Tương tự như các quy trình trước đó, chúng tôi muốn lấy biến *NSeparation* cục bộ của mình và chia nó cho độ dài mảng từ trước đó. Nhân các kết quả với **-1**. Sau đó, nhân kết quả với biến *GlobalSeparation* chung.
22. Cuối cùng, chúng tôi sẽ nhận được kết quả và đặt biến *NSeparation* cục bộ của chúng ta. Đặt bình luận khu vực này là "**Tính toán hành vi điều khiển đổ xô - kết quả đầu ra NCohession, NAalignment, NSeperation**":



23. Sau đó, chúng ta phải tìm khu vực **Tính toán Direction** và thêm từng biến của chúng ta vào phép tính cuối cùng của biến *Direction* cho agent này:



24. Điều cuối cùng chúng ta cần làm là chuẩn bị cho phần tiếp theo trong khóa học này. Vì vậy, hãy tìm một chỗ trống phía trên Blueprint ở bên trái và thêm một số mã mới.
25. Thêm một event tự tạo mới gọi là **ResetBall**.
26. Tiếp theo, chúng ta sẽ tạo node **SetActorLocation** cho agent hiện tại và đặt **New Location** cho biến **Start Location** mà chúng ta đã thiết lấp lúc đầu.
27. Bây giờ, chúng ta muốn đặt một hướng mới và chúng ta sẽ thực hiện việc này bằng cách đặt một biến **Direction** trong node **SET**. Sau đó, chúng ta sẽ tách cấu trúc và chỉ áp dụng cho **Return Value X** và **Return Value Y** từ node **Random Unit Vector**:



Kiểm soát hành vi thông qua UMG

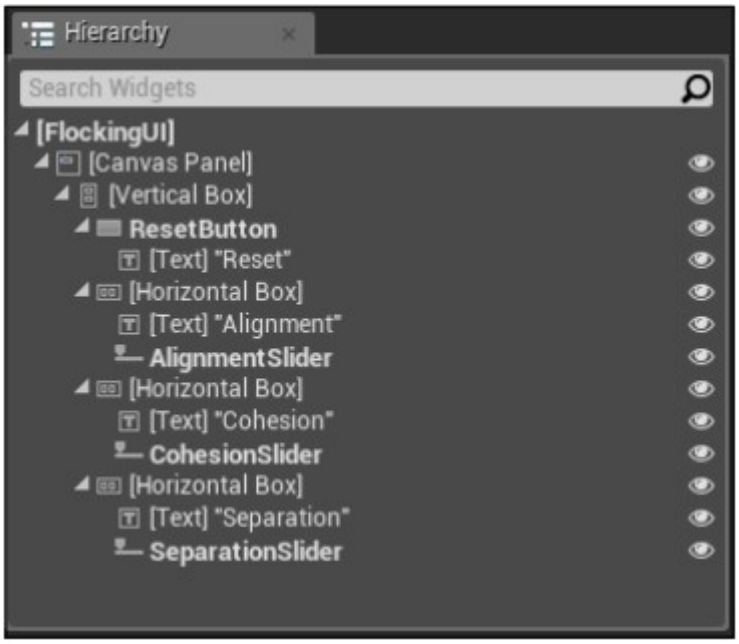
Trong phần này của chương, chúng ta sẽ đề cập đến việc sử dụng UMG để kiểm soát các hành vi ảnh hưởng đến các agent của chúng ta. Trước tiên, chúng ta sẽ thực hiện việc này bằng cách tạo một widget UMG với các điều khiển phù hợp để thao tác với ba biến float của chúng ta. Sau đó, chúng ta phải gán widget người dùng này cho PlayerController sở hữu. Sau đó, chúng ta sẽ kết thúc bằng cách thêm một chức năng sẽ đặt lại các Agent về vị trí ban đầu của chúng, bắt đầu mô phỏng mới.

Một giao diện đơn giản

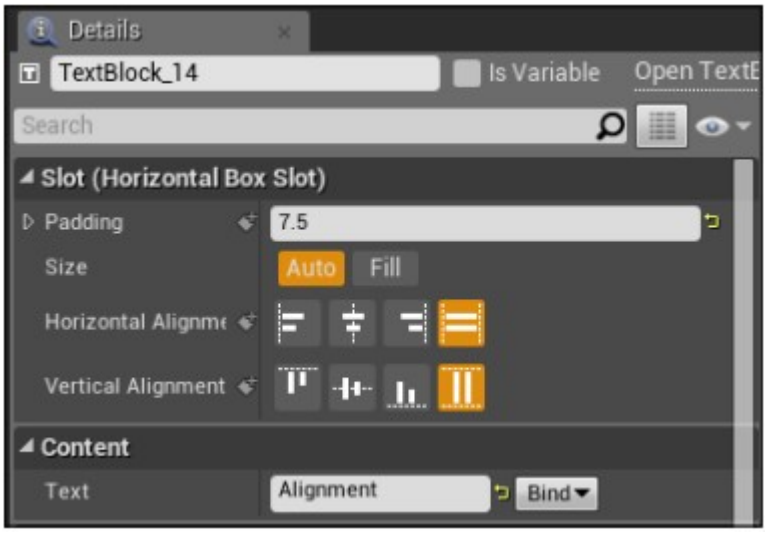
Hãy đi đến Content Browser của chúng ta, tạo một blueprint Widget mới và đặt tên là *FlockingUI*. Hãy mở cái này lên và chuyển đến tab **Designer** để bắt đầu, như sau:

1. Hãy kéo **Vertical Box** vào panel **Hierarchy** của chúng ta.
2. Tiếp theo, đặt các thuộc tính của vị trí (**Canvas Panel**): giá trị **Size X** thành **350.0** và giá trị **Size Y** thành **600.0**.
3. Sau đó, kéo một nút nhấn vào **Vertical Box** trong panel **Hierarchy** của chúng ta. Đổi tên nút nhấn thành **ResetButton**. Tiếp theo, bên trong panel **Details** của nút nhấn vừa tạo, đặt thuộc tính **Padding** của ô **Vertical Box** là **75.0** và **25.0**.
4. Cuối cùng, kéo **Text** vào panel **Hierarchy** của chúng ta và đặt bên dưới **ResetButton**. Tiếp theo, đặt thuộc tính văn bản **Text** thành "**Reset**":

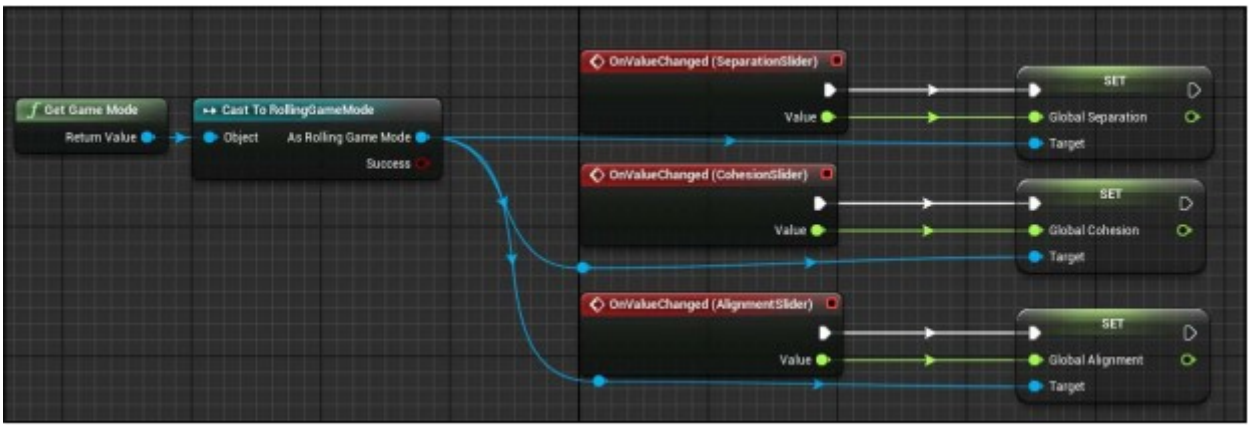
5. Bây giờ, hãy kéo **Horizontal Box** vào panel **Hierarchy** của chúng ta trong **Vertical Box**. Sau đó, thêm hai widget khác trong **Horizontal Box**, được gọi là **Text** và **Slider**.
6. Bây giờ, chúng ta sẽ sao chép **Horizontal Box** và các phần tử con của nó thêm hai lần nữa trong panel **Hierarchy**.
7. Nhấp chuột phải và sao chép **Horizontal Box** rồi dán vào **Vertical Box** hai lần:



8. Sau này, chúng ta sẽ đổi tên các vật dụng mà chúng ta sẽ sử dụng cho mục đích tổ chức. Theo trình tự, chúng ta muốn đặt tên cho thanh trượt đầu tiên là *AlignmentSlider*, thanh trượt thứ hai là *CohesionSlider* và thanh trượt cuối cùng là *SeparationSlider*.
9. Bây giờ, theo thứ tự, hãy đặt thuộc tính văn bản của chúng ta là **Text** thành ba tên sau: *Alignment*, *Cohesion* và *Separation*. Sau đó, chúng ta sẽ đặt thuộc tính **Padding** cho text là **7.5**.
   Bây giờ nó đã hoàn thành, bạn sẽ có một cái gì đó tương tự như ảnh chụp màn hình sau:



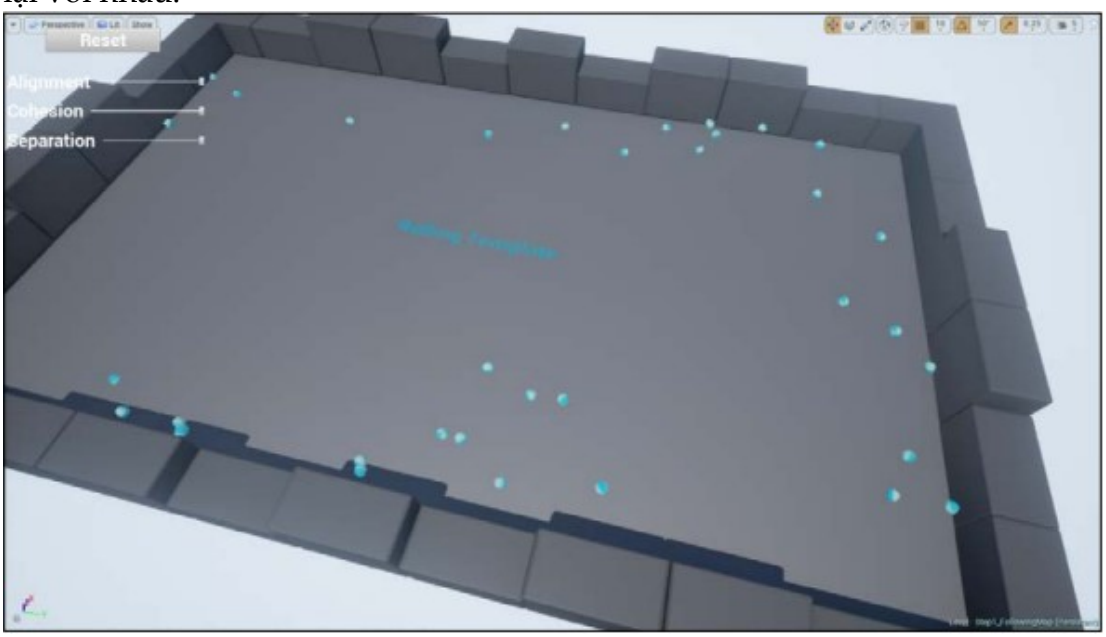
10. Để cập nhật các biến global, chúng ta phải tạo một event từ ba slider. Chúng ta có thể bắt đầu với *AlignmentSlider*, sau đó vào phần **Event**, rồi nhấp vào nút **[+]** trên tùy chọn **OnValueChanged**. Sau đó, khi event này được gọi, chúng ta sẽ cập nhật biến *GlobalAlignment* trong **RollingGameMode**.
11. Bây giờ, chúng ta sẽ làm tương tự cho hai thanh trượt khác—*CohesionSlider* và *SeparationSlider*—với giá trị global tương ứng:



12. Hãy tập trung trở lại chế độ **Designer** của widget người dùng này. Sau đó, nhấp vào event **ResetButton** mà chúng tôi đã thực hiện ở các bước trước.
13. Chúng ta muốn đi xuống phần **Event** và nhấp vào nút **[+]** trên tùy chọn **OnPressed**. Điều này sẽ cung cấp cho chúng ta event chúng ta vừa tạo.
14. Ở đây, chúng ta sẽ kéo từ chân và gọi **Get All Actors Of Class**. Tiếp theo, chúng ta muốn giá trị **Actor Class** là **PhysicsBallBP**. Từ đó, **Out Actors** sẽ trả lại tất cả các đối tượng của **PhysicsBallBP** trong màn chơi thành một mảng.
15. Điều cuối cùng cần làm là kéo từ **Out Actors** và gọi **Reset Ball**. Điều này sẽ thông báo cho tất cả các agent của chúng ta thiết lập lại mô phỏng:

16. Bây giờ, chúng ta phải quay lại **PhysicsBallBP** trong EventGraph để thêm vài blueprint bổ sung cho nó. Tìm một chỗ trống phía trên Event **Reset Ball**.
17. Nhấp chuột phải và tìm kiếm **Event Possessed**. Chiếu qua một controller mới tới *PlayerController*, sau đó kéo từ **As Player Controller** và tìm kiếm **Create Widget**. Thiết lập chân **Class** cho node này thành **FlockingUI**.
18. Tiếp theo, chúng ta phải kéo từ **Return Value** và gọi **Add To Viewport**. Cuối cùng, chúng ta cần cho người chơi thấy chuột để họ có thể tương tác với widget.
19. Nhấp chuột phải và tìm kiếm **Get Player Controller**. Kéo từ chân **Return Value** và tìm kiếm **Show Mouse Cursor**.
    Đặt bình luận cho khu vực này "**Hiện Flocking UI & Con trỏ chuột**":

Hãy biên dịch mọi thứ. Nhấp vào nút **Save** để lưu tất cả và quay lại Map của chúng ta. Nếu bạn nhấn Play, bạn sẽ thấy các tác nhân bật ra khỏi bức tường mà chúng va chạm. Sau đó, nếu chúng ta thực hiện các hành vi khác nhau, chúng ta sẽ thấy rằng chúng bắt đầu ảnh hưởng đến hướng của các tác nhân của chúng ta.
Ảnh chụp màn hình sau đây cho thấy cấp độ trông như thế nào sau khi kết hợp mọi thứ lại với nhau:



Tóm tắt

Hãy dành chút thời gian để nói ngắn gọn về những gì chúng ta đã làm trong chương này. Đầu tiên, chúng tôi thiết lập các tác nhân của mình để bật ra khỏi tường. Điều này cho phép chúng ta xem các tác nhân mô phỏng di chuyển và bốn hành vi sẽ ảnh hưởng đến chúng như thế nào. Tiếp theo, chúng ta sẽ triển khai hành vi Follower và Leader và điều này tạo ra các nhóm trong các agent trong quá trình mô phỏng.

Điều cuối cùng chúng ta làm là triển khai phong trào flocking, được chia thành ba hành vi khác nhau. Hành vi đầu tiên là Alignment và nó chịu trách nhiệm sắp xếp các tác nhân với các agent lân cận. Hành vi thứ hai là Cohesion và nó chịu trách nhiệm hướng các agent đến trung tâm của các tác nhân lân cận.

Hành vi thứ ba là Separation và nó chịu trách nhiệm hướng tác nhân ra khỏi các tác nhân gần đó. Sau đó, chúng ta chỉ cần tạo một giao diện người dùng để kiểm soát các trọng số mà chúng tôi đã tạo trong khóa học và đặt lại các agent bất kỳ lúc nào trong quá trình mô phỏng.

Bây giờ chúng ta đã hoàn thành việc đề cập đến các hành vi di chuyển khác nhau, đã đến lúc chúng ta kết hợp tất cả những gì chúng ta đã trình bày về AI trong cuốn sách này. Trong chương tiếp theo, chúng ta sẽ tạo ra một nhân vật AI, nhân vật này sẽ tuần tra, tìm kiếm và tiêu diệt bất kỳ kẻ thù nào. Vì vậy, chúng ta hãy chuyển sang chương tiếp theo để bắt đầu!