

Course - Lập trình AI cho Unreal Engine

Giới thiệu Di Chuyển

Giải thích cách giới thiệu di chuyển cho các nhân vật AI của chúng ta trong Unreal Engine 4. Tính năng Tìm đường sẽ được sử dụng để cho phép nhân vật của chúng ta điều hướng một cách thông minh theo một cấp độ.

Tags: Giới thiệu Di Chuyển

Trong chương này, bạn sẽ tìm hiểu cách đưa di chuyển vào các nhân vật AI của chúng ta trong Unreal Engine 4. Chúng ta cũng sẽ xem xét các thuật toán cơ bản cho phép chúng ta hướng dẫn các nhân vật của mình điều hướng đến một điểm trên mặt phẳng 2D. Chúng ta sẽ sử dụng các công cụ khác sẽ hỗ trợ AI của chúng ta điều hướng các đường dẫn được chỉ định.

Các chủ đề mà chúng ta sẽ đề cập trong chương này bao gồm:

- Dò theo đường dẫn (Path Finding)
- NavMesh
- Công cụ sửa đổi navigation và navigation
- Các node blueprint cho navigation

Bạn có thể download source full dự án, [tại đây](#).

Tổng quan

Trong chương này, di chuyển sẽ là mục tiêu chính. Làm thế nào chúng ta đạt được di chuyển hoặc sử dụng các công cụ khác để giới thiệu di chuyển là những câu hỏi chúng ta phải đặt ra. Bây giờ bạn đã cảm thấy thoải mái với một số công cụ chúng ta sử dụng trong AI, chúng ta phải làm cho AI có khả năng thực hiện hành động phổ biến nhất của nó: di chuyển. Hệ thống chúng tôi sử dụng để đưa di chuyển vào AI được gọi là Path Finding.

Cách hoạt động của Path Finding là lấy các vị trí trong không gian được chỉ định là có thể đi qua. Các vị trí bắt đầu và kết thúc này được đưa vào một hàm tính toán đường đi ngắn nhất giữa hai vị trí. Thuật toán sử dụng thông tin vị trí tương đối, chẳng hạn như liệu nó có bị chặn bởi một đối tượng hoặc tác nhân hay không và ngăn không cho vị trí này có thể di chuyển được. Điều này cực kỳ hữu ích khi cố gắng tạo một đường dẫn trong một thế giới có các đối tượng di chuyển linh hoạt.

Dò theo đường dẫn (Path Finding)

Chúng ta sẽ chia hoạt động dò đường này thành các component khác nhau sao cho tất cả đều hoạt động gắn kết với nhau. Component đầu tiên là **NavMesh**, đại diện cho đường dẫn có thể đi lại. Nó không chỉ là component tạo Navigable Mesh, mà nó còn là cách biểu diễn Navigable Mesh được sử dụng rộng rãi. Một component khác là **NavMeshModifiers**, có thể phục vụ các mục đích khác nhau trong một vài ví dụ như sau:

- **Influence Mapping:** Nó cho phép bạn cung cấp thông tin đầu vào cho AI dựa trên vị trí của chúng trên NavMesh.
- **Null Paths:** Những thứ này chỉ đơn giản cho phép bạn lược bỏ vài khu vực của

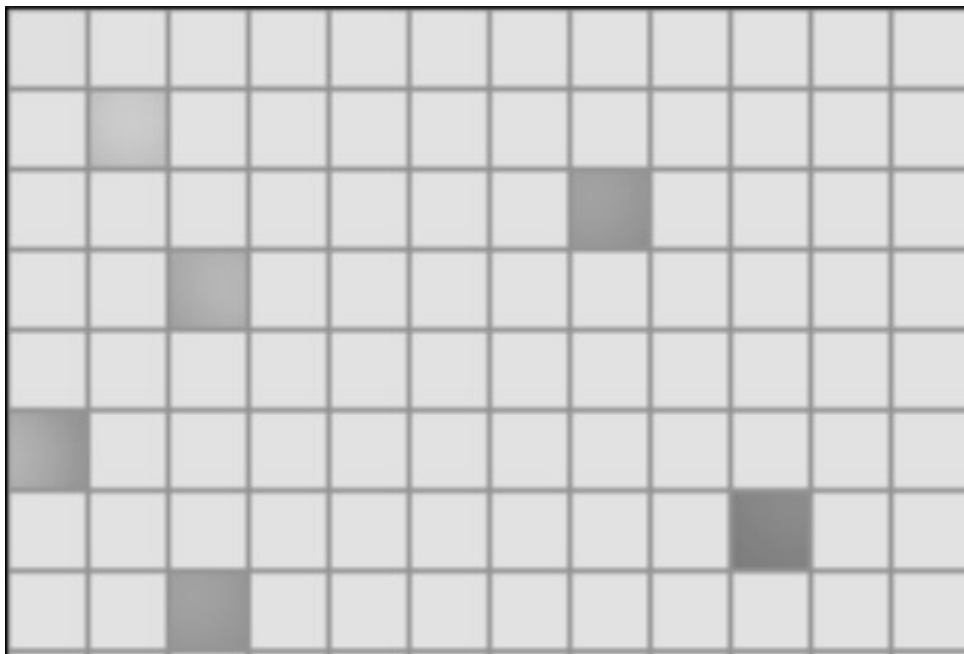
NavMesh.

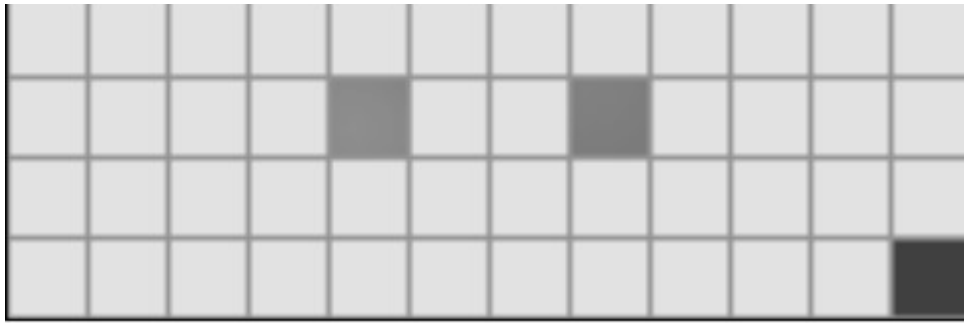
- **Allowed Paths:** Những cái này chỉ đơn giản là cho phép bạn chặn bất kỳ AI nào không được phép điều hướng đến khu vực này của NavMesh.

Thuật toán A*

Cốt lõi của hệ thống dò đường là một thuật toán tính toán đường đi ngắn nhất có thể điều hướng giữa hai điểm trên NavMesh. Thuật toán Dijkstra được đặt theo tên của nhà khoa học máy tính Edsger Dijkstra, người đầu tiên tạo ra nó vào năm 1956 và sau đó xuất bản nó vào năm 1959 để tìm đường đi ngắn nhất giữa hai điểm. Có một số thuật toán tìm đường đáng được đề cập, nhưng thuật toán được sử dụng phổ biến nhất trong trò chơi là một biến thể của thuật toán Dijkstra. Biến thể được gọi là thuật toán A*. A* hoạt động bằng cách có một danh sách các điểm có thể duyệt. Sau đó, từ vị trí bắt đầu, chúng ta sẽ tìm kiếm con đường ngắn nhất theo mỗi hướng có sẵn. Chúng ta có thể xác định điều này bằng các giá trị heuristic. Các giá trị này sẽ cho chúng ta biết lượng tính toán để đi đến điểm tiếp theo dựa trên một số quy tắc được xác định trước. Trong A*, chúng ta sẽ di chuyển từ điểm này sang điểm khác để giảm lượng tính toán thăm dò. Sự khác biệt lớn giữa hai thuật toán là ở A*, bạn có một giá trị heuristic, giá trị này ảnh hưởng đến các quyết định về đường dẫn của bạn. Các giá trị heuristic là những trở ngại đối với lượng tính toán dự đoán để di chuyển đến điểm có thể tiếp theo. Khi đường dẫn được tạo, chúng ta muốn giá trị heuristic thấp nhất. Vậy nó có nghĩa là bằng cách đặt các giá trị heuristic thành 0, thuật toán giống như của Dijkstra. Thuật toán của Dijkstra quét nhiều nút hơn, thường làm cho nó kém hiệu quả hơn vì nó không dự đoán lượng tính toán của các đường dẫn đi qua.

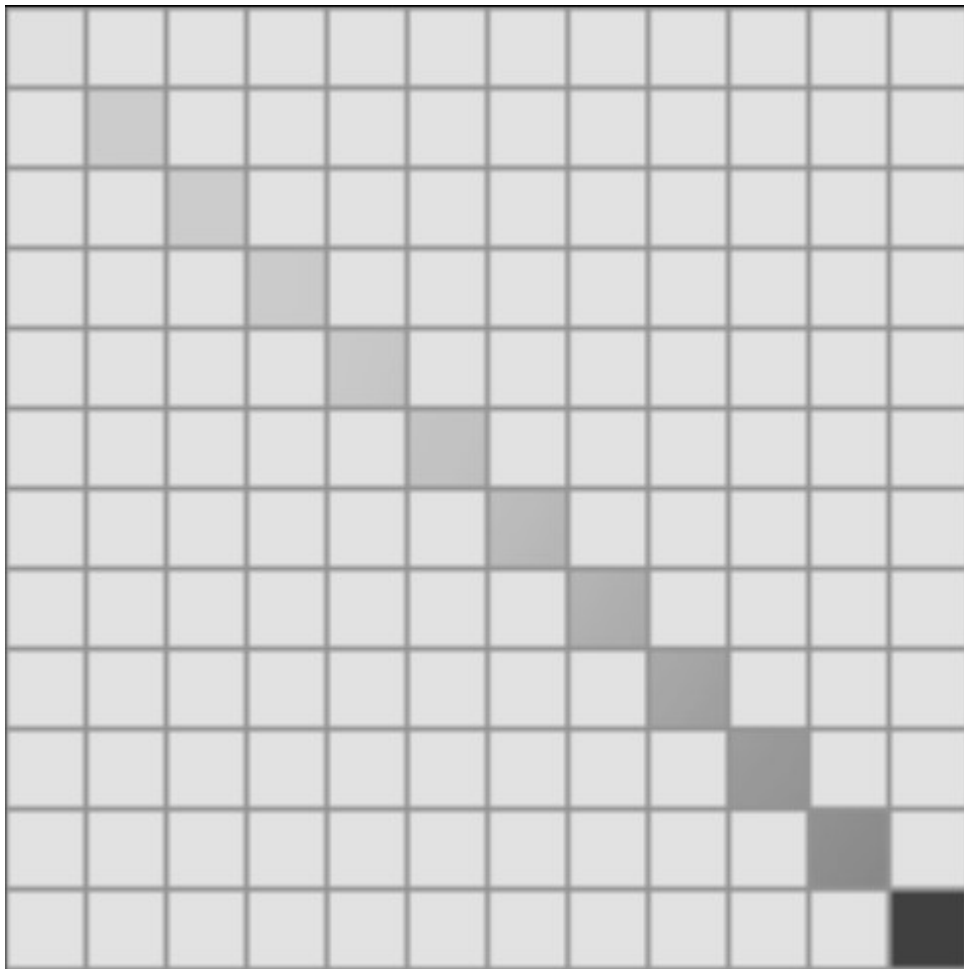
Lấy hình sau làm ví dụ,





Thuật toán Dijkstra tính đến mọi điểm khả dụng. Vì vậy, trong ví dụ này, chúng tôi sẽ bắt đầu từ góc trên bên trái và cố gắng đi đến góc dưới bên phải và tất cả các nút liền kề được coi là được kết nối và có thể đi ngang qua nhau. Thực hiện các bước sau:

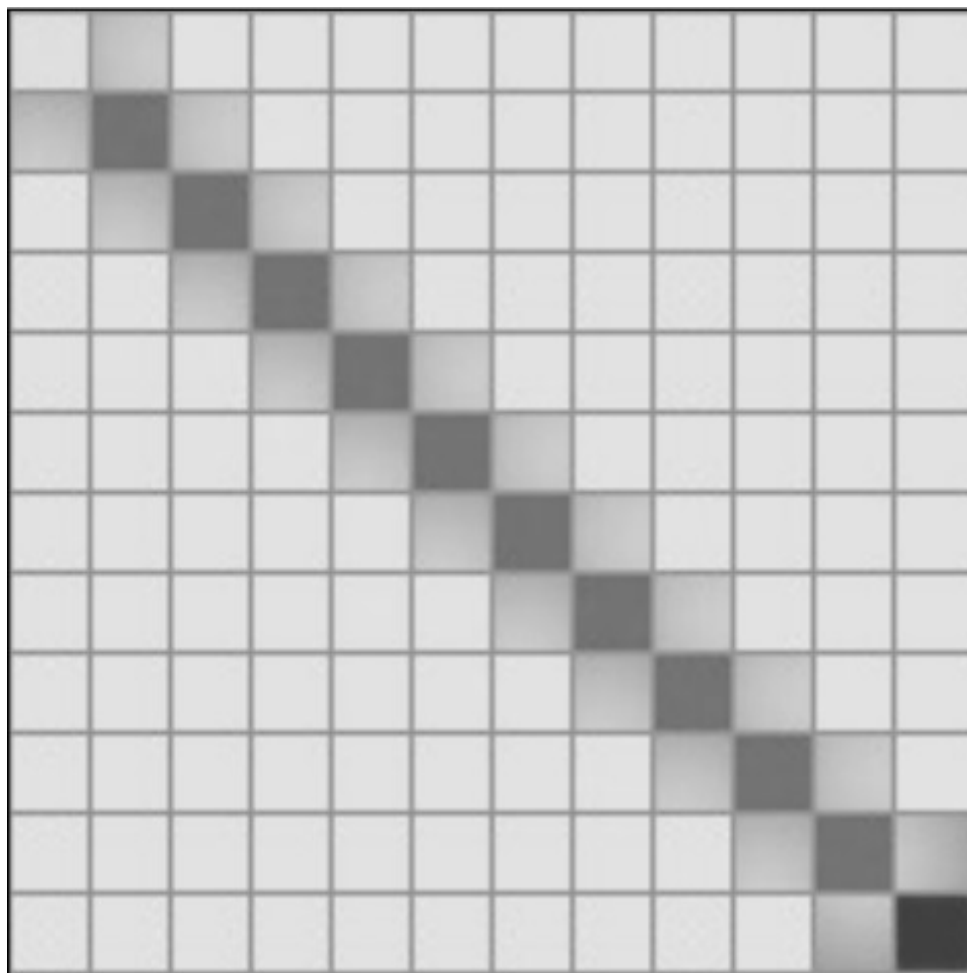
Thuật toán quét tất cả các nút có sẵn, lưu giữ bản ghi về đường đi ngắn nhất được tìm thấy. Các nút có thang độ xám khác nhau biểu thị rằng thuật toán phải khám phá các vị trí khác nhau để tìm đường dẫn:



Sau khi mục tiêu được phát hiện, một trình tự theo dõi được đưa vào hoạt động, do đó tạo ra đường dẫn.

Quá trình này có thể mất nhiều bước và đó là lý do chúng tôi điều chỉnh phỏng đoán trong A*. Thay vì tìm kiếm từng nút, chúng tôi sẽ tiếp tục từ con đường ngắn nhất hiện tại đến mục tiêu.

Một ví dụ với A* được trình bày trong lưới sau. Thuật toán sẽ bắt đầu từ nút đầu tiên (ở góc trên bên trái) và sau đó quét các nút lân cận để tính chi phí di chuyển. Nút có chi phí thấp nhất sẽ được chọn và các nút không được chọn sẽ được đưa vào danh sách các nút được quét. Điều này tránh cho chúng tôi phải đi đến các nút này một lần nữa cho truy vấn này:

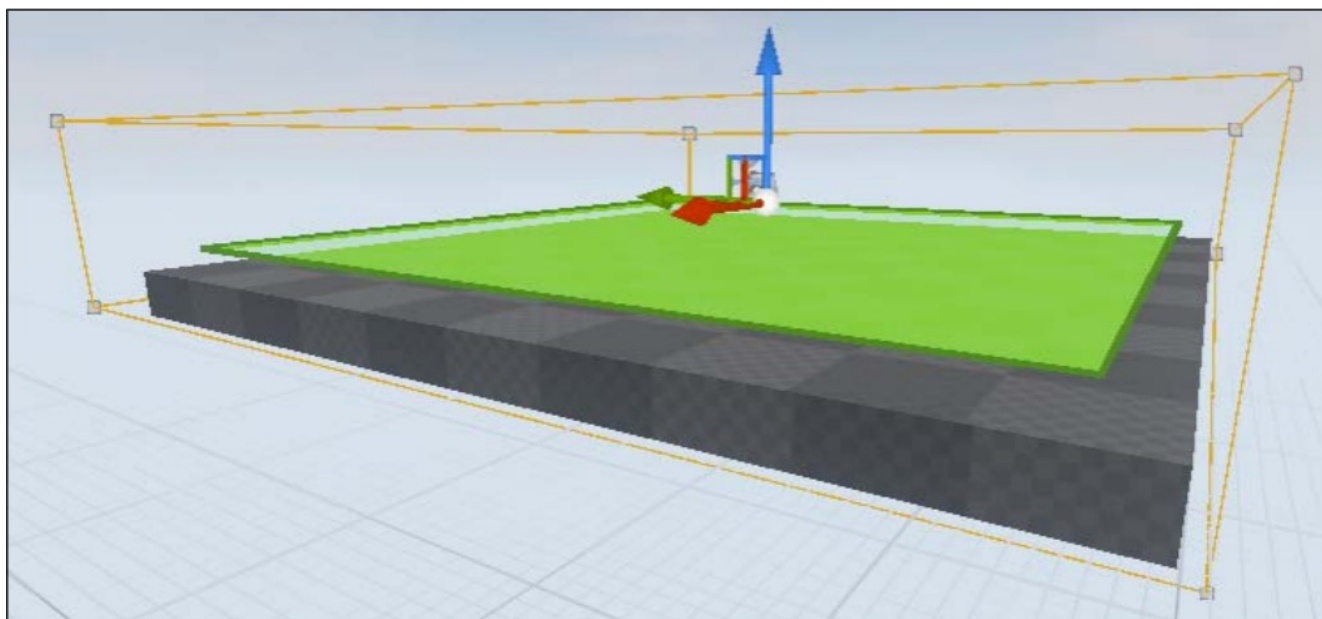


Từ bước 1, các nút lân cận được quét và kiểm tra các giá trị heuristic hoặc chi phí đường dẫn để xác định bước tối ưu tiếp theo. Kết quả là một đường dẫn trực tiếp (nếu không bị cản trở).

Có nhiều thuật toán khác nhau, nhưng A* dễ triển khai và không tốn nhiều tài nguyên, vì vậy thuật toán này thường được triển khai trong các công cụ trò chơi như một cách tạo đường dẫn giữa hai điểm trên lưới.

Navigation Mesh

Navigation Mesh trong Unreal Engine 4 cho phép chúng ta báo cho công cụ biết đường dẫn có thể di chuyển của chúng ta sẽ được tạo ở đâu và chúng ta có thể tối ưu hóa điều này để giúp tạo ra các đường dẫn chính xác hơn cho AI của chúng ta. Chúng ta sẽ gọi tắt là "NavMesh" cho Navigation Mesh vì hầu hết các nhà phát triển trò chơi đều quen thuộc với thuật ngữ này. Nếu bạn đang ở trong Unreal Engine và điều hướng đến panel **Modes** bên dưới **Volumes**, bạn sẽ thấy **NavMeshBoundsVolume**. Đây là một khối mà bạn có thể sử dụng để bao phủ hình học và tạo NavMesh trên đó. Tôi có một mẫu về những gì mô tả nó. Nếu bạn không thể nhìn thấy lưới màu lục do NavMesh tạo ra, hãy đảm bảo rằng bạn đã chọn tùy chọn Navigation (hoặc nhấn phím tắt **P**) trong menu **Show** trên màn hình:



RecastNavMesh

Trên mỗi màn chơi, bạn sẽ thấy một node trong World Outliner có tên là **RecastNavMeshDefault**. Nếu bạn nhấp vào nó, bạn có thể khám phá các tùy chọn mặc định ảnh hưởng đến **Navigation Mesh** trong màn chơi.

Nếu bạn đi xuống phần **Generation**, đây là thứ bạn sẽ bắt đầu điều chỉnh theo nhu cầu của mình. Nếu bạn đang cố gắng để có được hiệu suất cao hơn khi rebuild lúc chạy hoặc chính xác hơn với di chuyển của actor, thì hãy tối ưu hóa các giá trị sau:

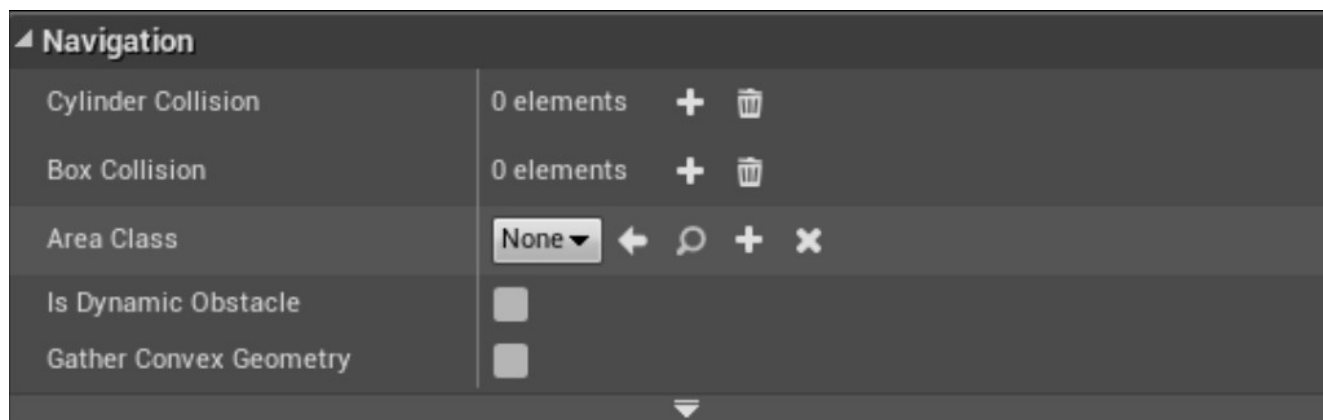


Tôi sẽ nói sơ qua một số thuộc tính để giúp bạn hiểu vai trò của chúng và cách chúng ảnh hưởng đến hiệu suất. Hãy xem xét một số thuộc tính quan trọng nhất trong **RecastNavMesh**:

- **Cell Size:** Đây là kích thước 2D của voxel, về cơ bản là một pixel 3D, đại diện cho một không gian để điều hướng. Đây là diện tích sàn của voxel. Điều này liên quan trực tiếp đến độ phân giải của NavMesh. Kích thước nhỏ hơn sẽ tạo ra nhiều điểm ảnh ba chiều hơn, dẫn đến độ chính xác của di chuyển tốt hơn. Thật không may, hiệu suất bị ảnh hưởng trong quá trình tạo NavMesh với nó.
- **Cell Height:** Đây là chiều cao của voxel. Giá trị này tương quan với **Agent Max Step Height** và nên giữ ở mức một nửa giá trị của nó. NavMesh sẽ không được tạo nếu giá trị **Cell Height** không nhỏ hơn ít nhất **0,1** so với **Agent Max Step Height**.
- **Tile Size UU:** Khi giá trị này ở giá trị thấp hơn, nó sẽ tăng hiệu suất tạo NavMesh. Hãy thử bật **Draw Poly Edges** để xem tính năng này hoạt động.
- **Agent settings (Radius, Height, Max Height, Max Slope, Max Step Height):** cụ thể cho các tác nhân của bạn và phải được chỉ định một cách thích hợp.
- **Min Region Area:** Nó loại bỏ những thứ có vẻ là tạo phẩm của các phần của thể hệ

NavMesh, những thứ quá tầm thường để điều hướng.

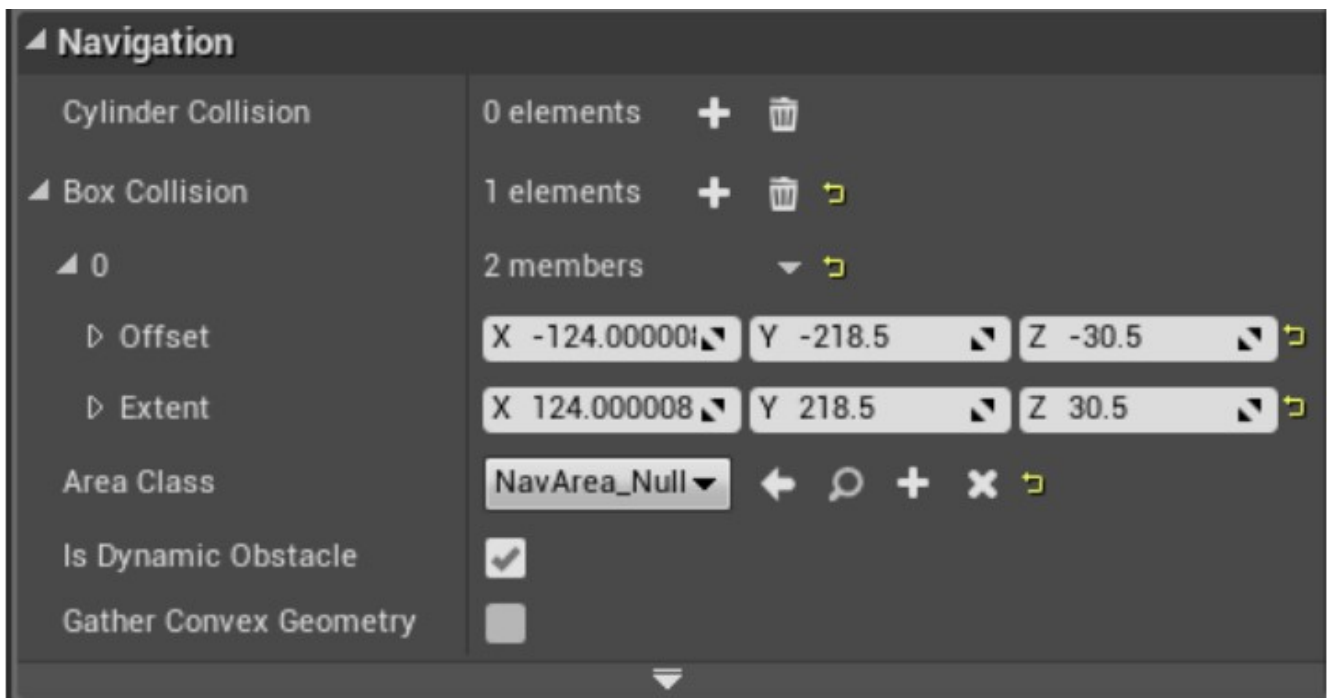
Nếu bạn có bất kỳ chương ngại vật nào trong NavMesh của mình và NavMesh đang được tạo trên các chương ngại vật, thì đối tượng này được coi là hình học và để có hiệu suất tốt hơn, bạn có thể làm cho lưới của mình được công nhận là Chương ngại vật động. Điều này ngăn NavMesh được tạo trên nó. Bây giờ, để ảnh hưởng đến khu vực này, bạn cũng phải đặt các giá trị **Offset** và **Extent** cho thuộc tính **Box Collision**. Hãy xem ảnh chụp màn hình sau:



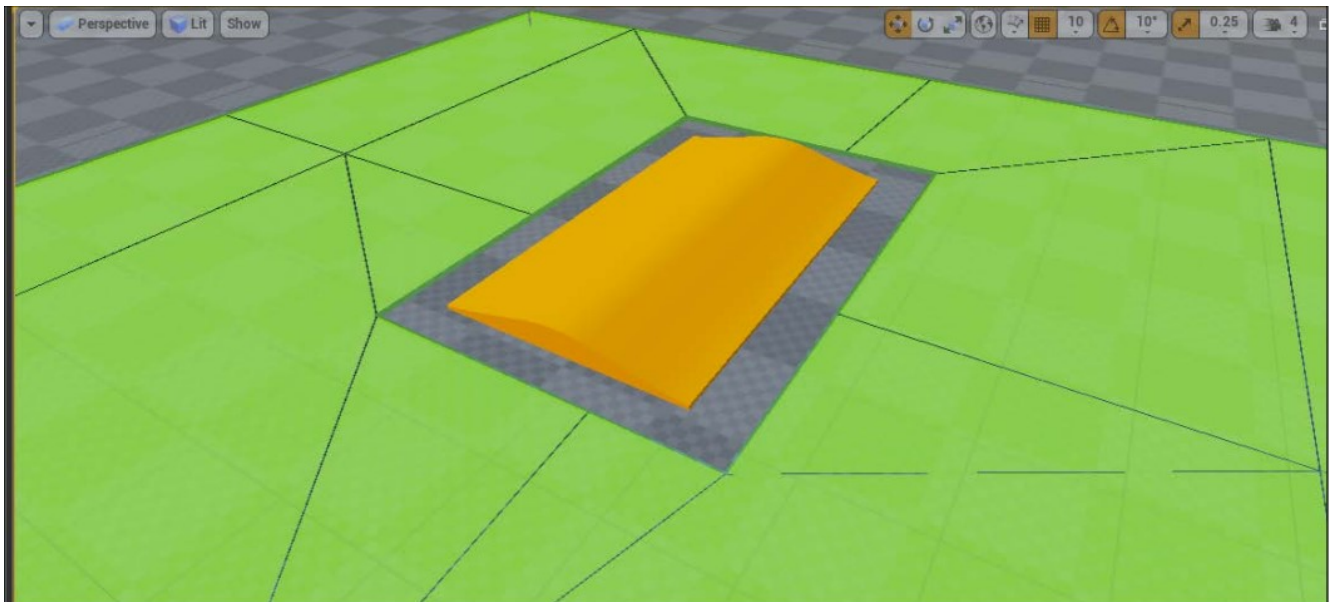
Hãy đi qua các bước sau:

1. Hãy tạo một phần tử **Box Collision** mới. Nó sẽ điền vào mảng một phần tử chứa hai giá trị.
2. Chúng ta cần đặt giá trị **Offset** thành giá trị âm của một nửa chiều rộng, chiều dài và chiều cao của NavMesh. Giá trị **Offset** là vị trí bắt đầu xung đột.
3. Tiếp theo, thiết lập giá trị **Extent** của chúng ta thành một nửa chiều rộng, chiều dài và chiều cao của NavMesh của chúng ta. Giá trị **Extent** là khả năng mở rộng của nó từ điểm trung tâm của NavMesh. Điều này có nghĩa là nó tương quan với một nửa kích thước volume.

Ví dụ, trong NavMesh này, tôi đã đặt các thuộc tính tương ứng; hãy xem:



Như bạn có thể thấy trong ảnh chụp màn hình sau trong trò chơi, vị trí chính xác bị ảnh hưởng bởi class *NavArea Null* mà chúng ta đã đặt trên tùy chọn **Is Dynamic Obstacle**:



Component di chuyển

Component di chuyển là động lực đằng sau hoạt hình. Tương tự như các loại component di chuyển khác, chẳng hạn như component di chuyển của xe, chúng ta sẽ cung cấp đầu vào, sau đó các thành phần này sẽ giao tiếp với các thành phần khác trong phần phụ trợ để mô phỏng hoạt ảnh hoặc thay đổi tỷ số truyền một cách chính xác. Điều này có nghĩa là các chức năng

khác cũng có thể ảnh hưởng đến quân tốt có thành phần di chuyển. Vì vậy, nếu **Behavior Tree** của chúng ta yêu cầu AIController tập trung vào một actor khác, thì AIController sẽ ảnh hưởng đến hướng mà actor do nó sở hữu đang đối mặt. Điều này có nghĩa là bạn không cần phải thông báo rõ ràng cho actor của mình thông tin này.

AIController


Trước đây chúng ta đã sử dụng AIController để di chuyển quân tốt trong *Chương 2, Tạo AI Cơ bản* và *Chương 3, Thêm Tính ngẫu nhiên và Xác suất*, nhưng chưa bao giờ dành thời gian để hiểu điều gì đang xảy ra trong Unreal Engine 4 để biến nó thành hiện thực. Việc hiểu rõ điều này cho phép bạn mở rộng các component di chuyển, chẳng hạn như khi bạn muốn tạo các phần mở rộng đặc biệt của component di chuyển để hỗ trợ việc nhận các hướng dẫn, chẳng hạn như **Move to Location** cho một phương tiện giao thông.

AIController là bộ điều khiển được xây dựng riêng cho AI. Tương tự như **PlayerController**, nó điều khiển actor mà nó hiện đang sở hữu. Sự khác biệt chính là AIController đi kèm với các chức năng cho phép bạn di chuyển AI bằng các công cụ như **Move to Location**. Với **Move to Location**, bạn có thể chỉ định giá trị **Acceptance Radius**, có sử dụng Path Finding hay không và các tùy chọn hữu ích khác. Ngoài ra còn có một phiên bản đơn giản của **Move to Location**, đôi khi cung cấp di chuyển mượt mà hơn nhưng không thể thay đổi các tùy chọn. Điều này có nghĩa là tất cả các tùy chọn đều là mặc định, như được hiển thị trong node **Move to Location**.

Nó được gọi là **Simple Move to Location** và được định vị như trong ảnh chụp màn hình sau:

Simple Move to Location





 Controller

 Goal


Move to Location *Target is AIController*





 Target


Return Value 


 Dest

 Acceptance Radius

 Stop on Overlap ☒

 Use Pathfinding ☒

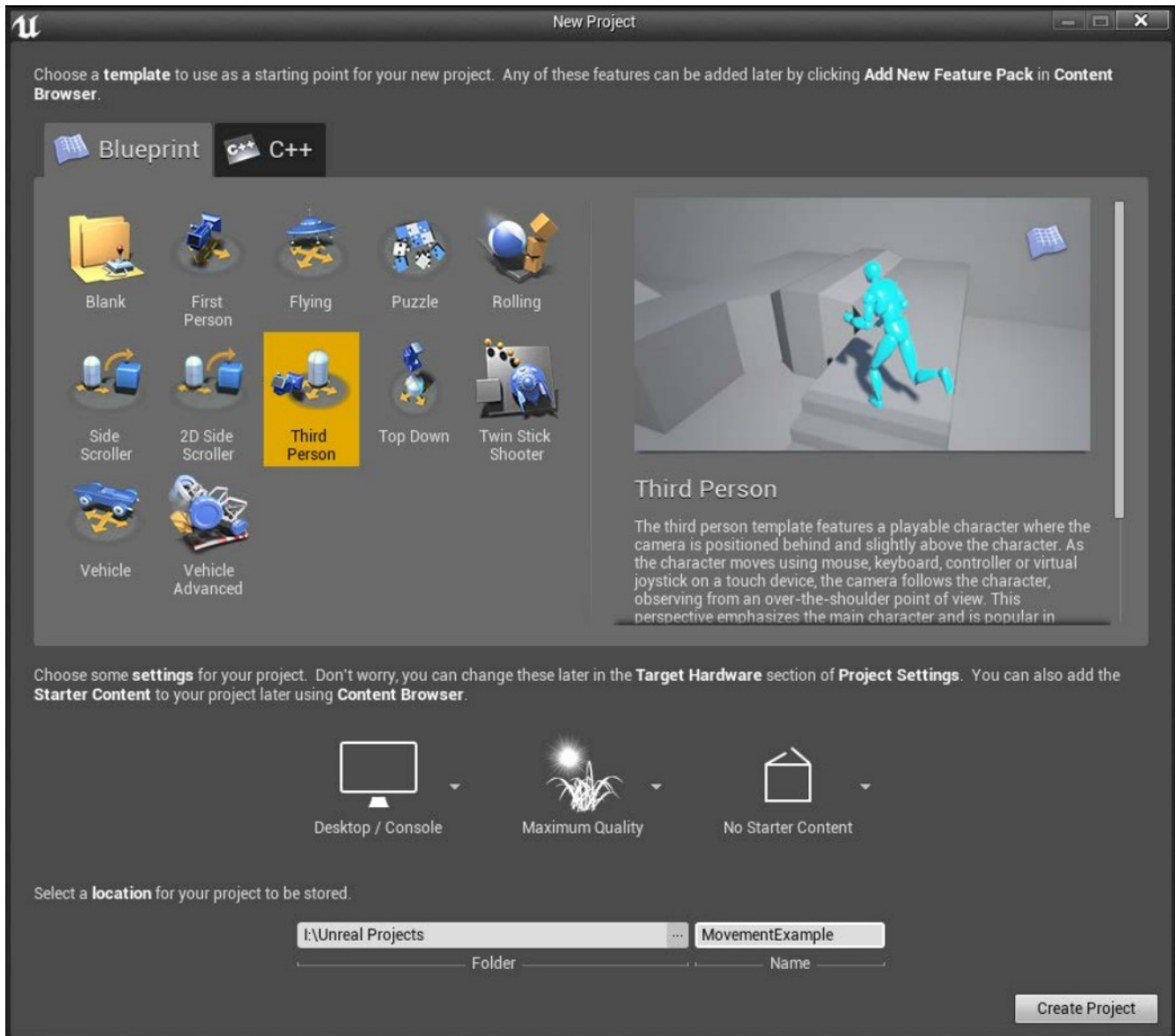
 Project Destination to Navigation
☐

 Can Strafe ☒

 Filter Class
  

Hãy bắt đầu

Mở Unreal Engine 4 và bắt đầu dự án mẫu **ThirdPerson** mới. Chúng ta cần mẫu này để chúng ta có thể sử dụng actor của Third Person tạo ra những con tốt Pawn do AI điều khiển cho chúng ta. Vì vậy, hãy chọn các cài đặt như trong ảnh chụp màn hình sau và nhấn **Create Project**:

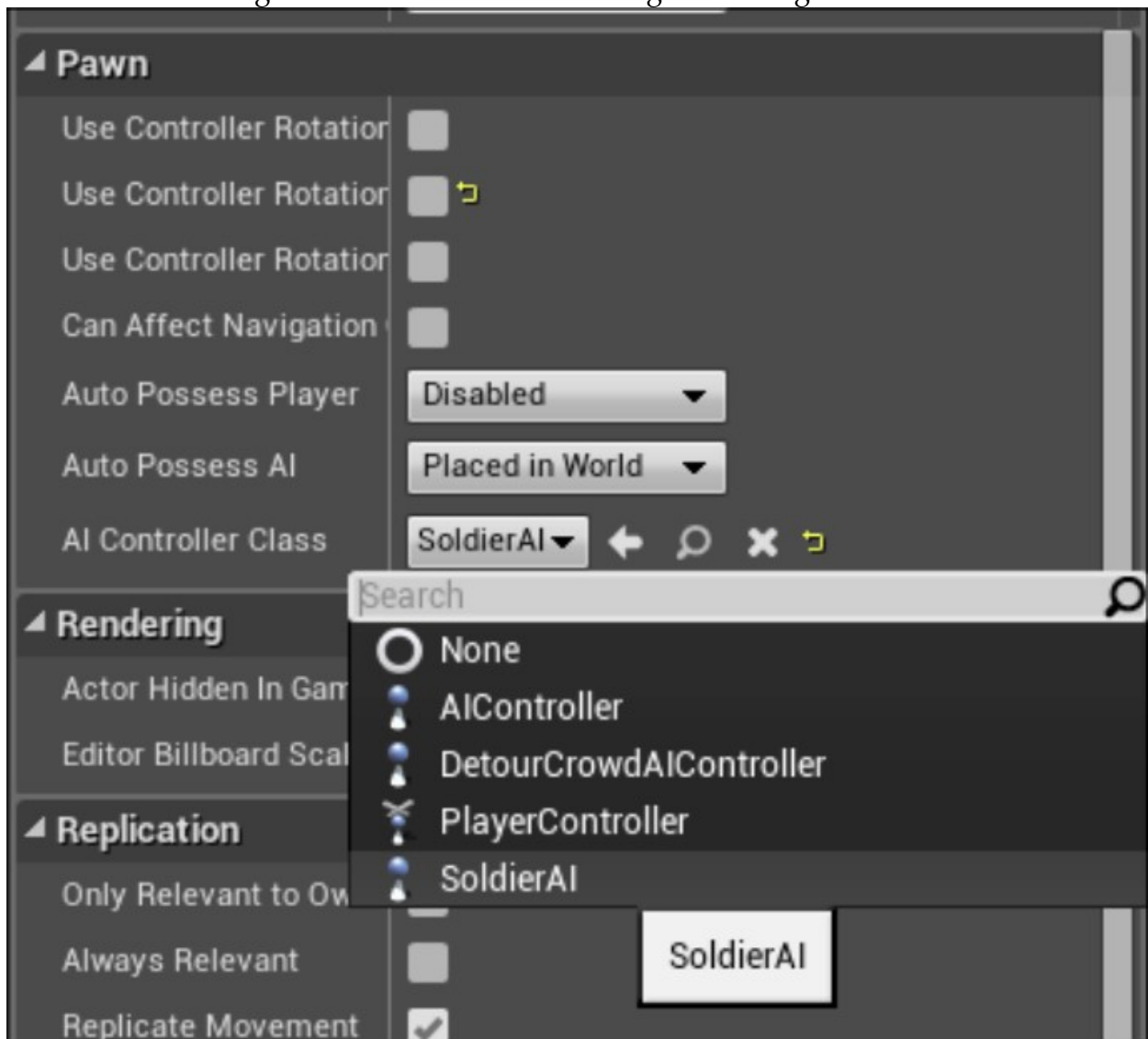


Tiếp theo, chúng ta sẽ tạo AIController mới của mình và nó sẽ chịu trách nhiệm yêu cầu con tốt của chúng ta di chuyển giữa hai điểm. Kế hoạch là đưa ra một số trở ngại để xem tác động mà chúng ta có thể có đối với đường đi của AI. Thực hiện các bước như sau:

1. Hãy bắt đầu bằng cách vào thư mục *Blueprint*. Và tạo một class AIController mới, đặt tên nó là **SoldierAI**. AIController này sẽ chịu trách nhiệm điều hướng con tốt mà nó

đang chiếm hữu.

2. Xóa mọi con tốt pawn khỏi màn chơi được thiết lập sẵn bởi dự án mẫu. Sau đó, chúng ta sẽ đặt thêm hai con tốt **ThirdPersonCharacter** mới vào trong màn chơi. Tiếp theo, chúng ta sẽ mở blueprint **ThirdPersonCharacter** này và truy cập các giá trị mặc định của nó.
3. Từ đây, hãy tìm AIController mặc định của con tốt này và thay đổi giá trị này thành class AIController mới của chúng ta có tên là **SoldierAI**. Cái mà nó sẽ làm là luôn sinh ra con tốt của chúng ta với AIController mà chúng ta đã dùng:



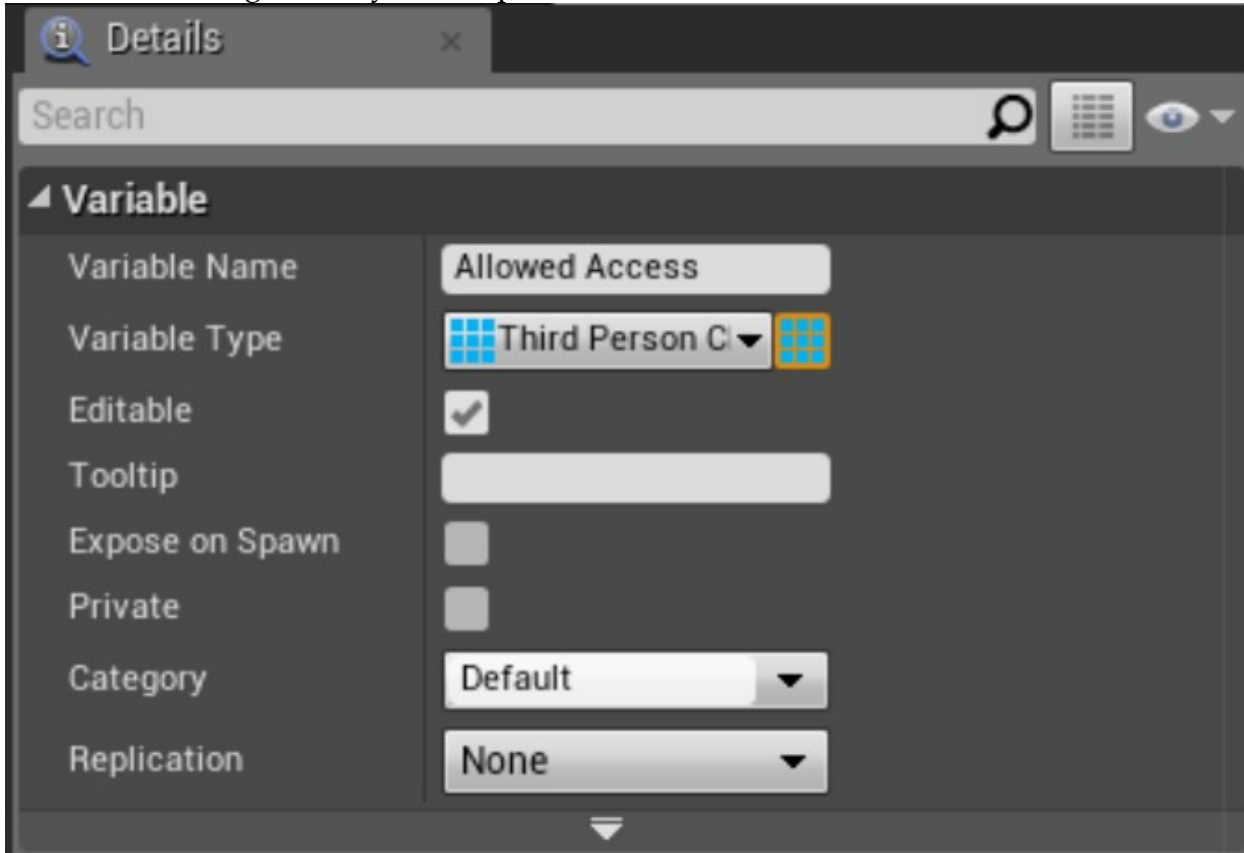
4. Lưu tất cả! Hãy tiến tới phần tiếp theo của chương này!

Waypoints

Trong chương này, các điểm waypoint đại diện cho các điểm mà bot sẽ hướng đến. Chúng ta sẽ thực hiện các thay đổi bổ sung cho các waypoint của mình để hạn chế những người có thể đi qua các điểm này. Nó sẽ được thiết lập bằng cách tạo một loạt các actor trên các waypoint

riêng lẻ. Khi những actor truy cập vào danh sách các waypoint có sẵn, chúng sẽ chỉ thêm các waypoint mà chúng được phép. Dưới đây là các bước để thực hiện việc này:

1. Bắt đầu trong cùng một thư mục *Blueprint* như trước đây. Nhấp chuột phải vào nó và tạo một blueprint mới. Chuyển đến Custom Class và chọn actor **Target Point**.
2. Đặt tên cho class con **Target Point** mới này là *Waypoint* và nhấn Okay.
3. Mở blueprint *Waypoint* mới của chúng ta và tạo một biến mảng public mới chứa các pawn **ThirdPersonCharacter** có tên là *Allowed Access*. Sau đó, chọn **Editable** để chúng ta có thể sửa đổi giá trị này trực tiếp từ Unreal Editor:



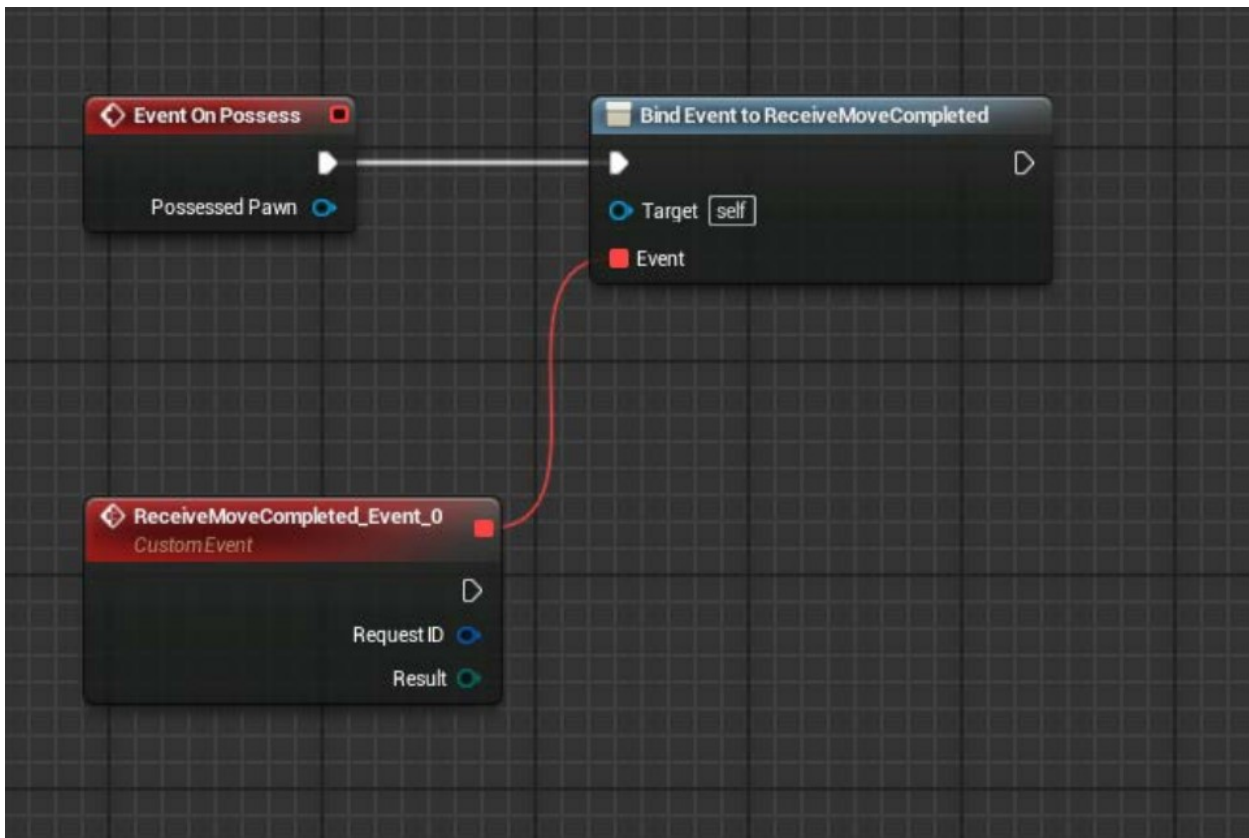
4. Bây giờ, hãy đặt một số waypoint vào trong màn chơi và chúng ta sẽ làm cho nó trở nên ngẫu nhiên. Nó sẽ cho phép đường đi của quân tốt của chúng ta được kiểm soát tốt hơn khi chúng ta xem xét một số chủ đề khác ở phần sau của chương này.
6. Tôi chọn đặt bảy waypoint trong màn chơi của mình và yêu cầu chúng chia sẻ khoảng ba điểm trong số đó để tạo ra sự độc đáo trong các đường dẫn. Vì vậy, bên trong một trong các node, hãy thêm cả hai con tốt mà chúng ta đã đặt vào màn chơi trước đây.

Navigation

Bây giờ chúng ta có những người lính của mình trong màn chơi, chúng tôi muốn điều hướng họ đến các waypoint tương ứng. Điều này sẽ được thực hiện thông qua AIController mà chúng tôi đã thiết lập cho người lính của mình. Nếu tập trung trở lại vào Unreal Engine,

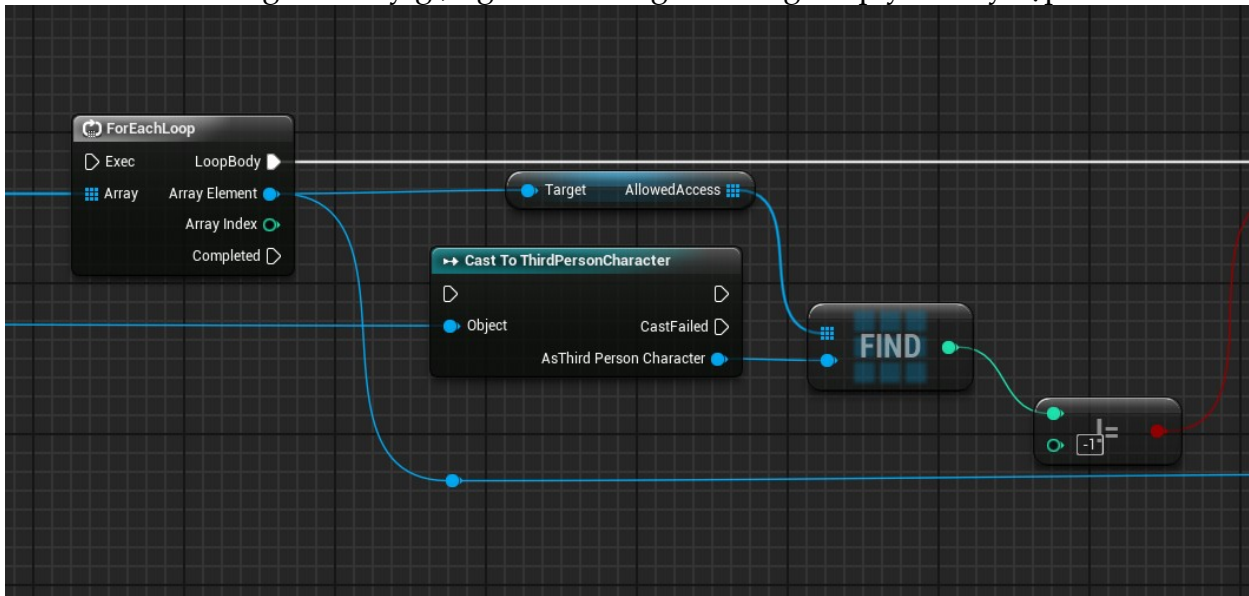
chúng ta có thể bắt đầu! Thực hiện theo các bước sau:

1. Mở class **AIController** mà chúng ta đã tạo trước đó và đã đặt tên là *SoliderAI*.
2. Sau đó, chúng ta sẽ chuyển qua tab sơ đồ **EventGraph**. Từ đây, chúng ta muốn **AIController** của mình bắt đầu điều hướng đến tuyến đường đầu tiên ngay khi nó sở hữu một con tốt xứng đáng. Vì vậy, hãy thêm về một event để thúc đẩy hành động này (tôi khuyến khích sử dụng các event). Event mà chúng ta đang tìm kiếm có tên là **Event On Possess**. Điều này cũng sẽ trả về con tốt mà nó sở hữu và chúng ta có thể sử dụng để so sánh với danh sách *AllowedAccess* được phép từ các waypoint.
3. Kéo từ chân thực thi và hãy chỉ định một event mới có tên là **RecieveMoveCompleted**. Event này sẽ tiếp tục đưa con tốt đến lộ trình tiếp theo ngay sau khi nó hoàn thành lộ trình trước đó:

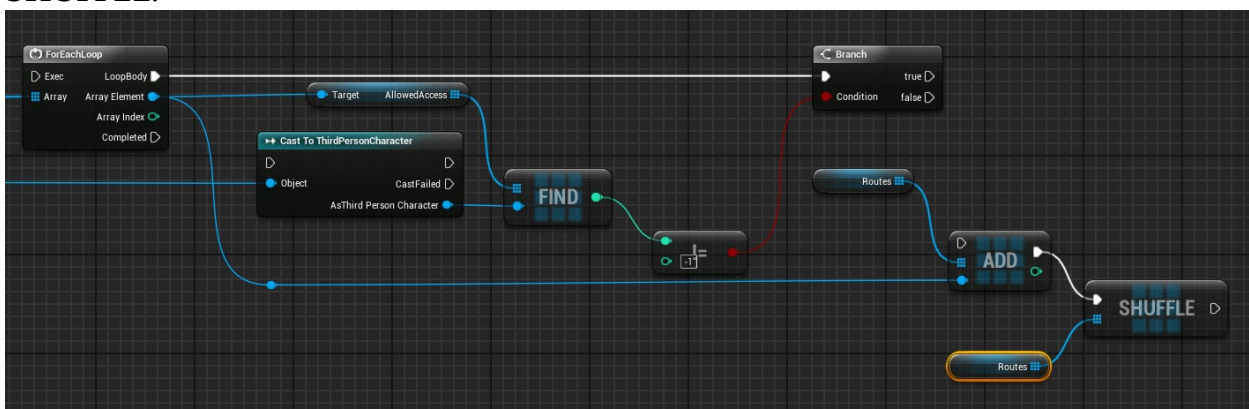


4. Bây giờ, từ chân thực thi **Event On Possess**, hãy kéo dây và tạo node **DoOnce** để ngăn điều này được gọi hai lần.
5. Tiếp theo, từ node **Bind Event ReceiveMoveComplete**, hãy tìm kiếm và tạo node **Get All Actors Of Class** và chọn blueprint tự tạo *Waypoint* mà chúng ta đã tạo ra lúc trước cho chân tham số **ActorClass** của node **Get All Actors Of Class**. Bây giờ trong các bước tiếp theo chúng ta sẽ tìm kiếm trong danh sách các waypoint có sẵn và kiểm tra xem chúng ta có được phép đi lại ở đó hay không.
6. Tiếp đến, hãy tạo một node **ForEachLoop** từ mảng **Out Actors** và kéo mảng *Allowed Access* từ chân **Array Element**. Sau đó, chúng ta sẽ lấy con tốt được trả về từ biến **Event On Possess** và chuyển nó sang con tốt của **Third Person Character**.

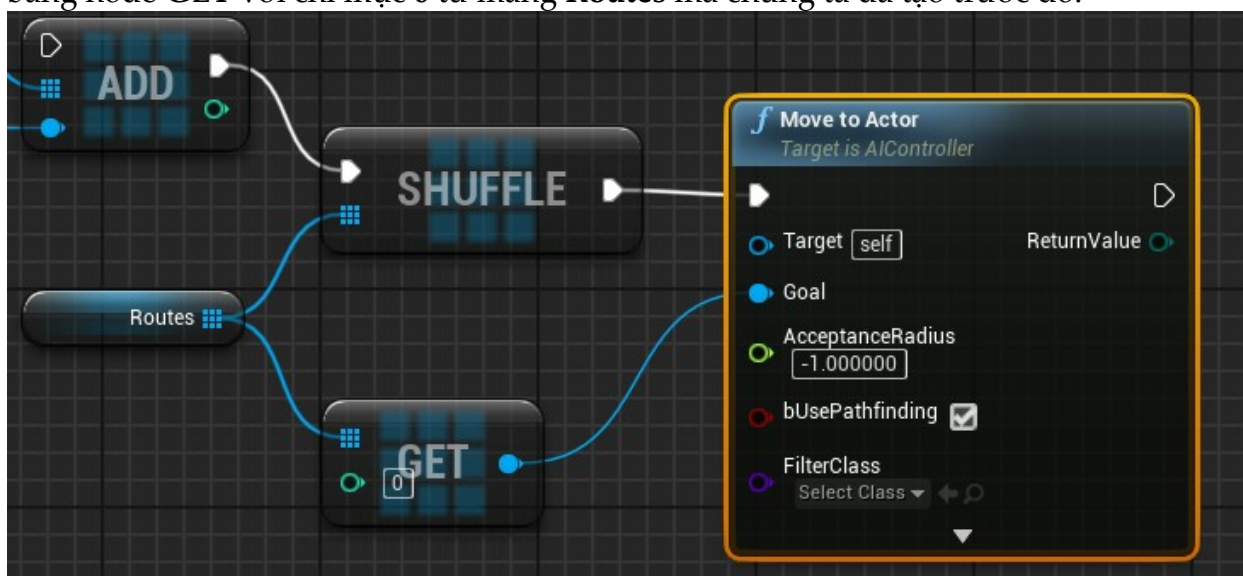
7. Từ đó, hãy cố gắng tìm con tốt trong mảng **Allowed Access** bằng node **FIND**. Mảng sẽ trả về **-1** nếu không tìm thấy gì, nghĩa là chúng ta không có quyền truy cập!



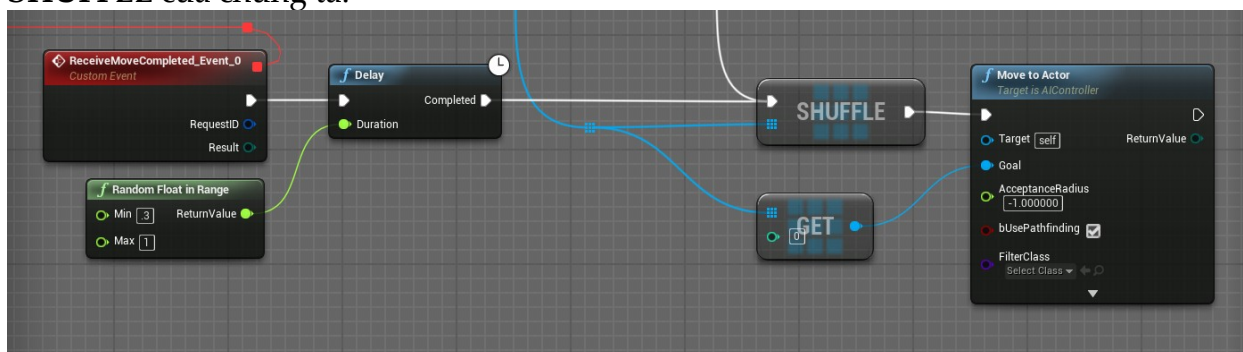
8. Chúng ta cần thực hiện lệnh *Compare Int != Int Boolean* và so sánh kết quả của chúng ta từ node **FIND** với **-1**. Sau đó chúng ta sẽ bom nó vào một node **Branch**.
9. Bây giờ, hãy cắm chân **LoopBody** của node **ForEachLoop** vào node **Branch**. Khi node **Branch** này trả về *True*, nó có nghĩa là chúng ta được phép điều hướng đến chân **Array Element** hiện tại và chúng ta nên thêm nó vào các điểm mà AI có thể điều hướng đến.
10. Bây giờ, hãy tạo một mảng **Routes** mới kiểu *Waypoint* để giữ các waypoint mà chúng ta có thể đi lại. Sau đó, chúng ta tạo node **ADD** từ node **Routes**, và cắm node **Branch** vào node **ADD** và chân **Array Element** của node **ForEachLoop** vào chân phần tử của node **ADD**. Điều này sẽ cho phép chúng ta truy cập nhanh vào các tuyến đường mà chúng đã chỉ định cho chúng ta.
11. Bây giờ, chúng ta cần yêu cầu quân tốt điều hướng đến điểm waypoint sau khi thao tác này hoàn thành. Điều này có thể được thực hiện bằng cách kéo ra sơ đồ một mảng **Routes** khác riêng biệt khỏi các node mà chúng ta đã đặt trước đó.
12. Chúng ta muốn xáo trộn mảng **Routes** của mình để điểm đến tiếp theo của chúng ta là ngẫu nhiên. Bạn có thể làm điều này bằng cách kéo từ node **Routes** và tìm kiếm **SHUFFLE**.



13. Bây giờ, chúng ta đã có vị trí mà con tốt cần đến và chúng ta cần ra lệnh cho con tốt di chuyển. Chúng ta có hai tùy chọn: thứ nhất là cung cấp cho node **Move to Location** một vị trí và thứ hai là cung cấp một actor và cho phép công cụ xử lý việc tìm đích. Tôi sẽ chọn cái sau cho ví dụ này.
14. Nhấp chuột phải và tìm kiếm nút **Move to Actor** và chúng ta sẽ bơm actor đầu tiên bằng node **GET** với chỉ mục **0** từ mảng **Routes** mà chúng ta đã tạo trước đó:



15. Từ đó, hãy dò ngược trở lại event **ReceiveMoveCompleted** của chúng ta và kéo từ sự kiện này và tạo ra node **Delay**, sau đó cắm vào chân **Duration** của node **Delay** một node **Random Float in Range** với **Min:3**, **Max:1**, cuối cùng là cắm node **Delay** vào node **SHUFFLE** của chúng ta.



16. Bây giờ xem như đã hoàn thành, và chúng ta sẽ quay trở lại **Level Editor**. Hãy cùng mô phỏng và xem AI làm được những gì! Không làm gì cả trong kết quả preview!!!

Bây giờ, hãy quay lại với công việc sau khi đã xem xong preview.

Công cụ Navigation Modifier

Bạn có nhớ khi chúng ta nói về heuristic không? Công cụ sửa đổi điều hướng cho phép bạn tác động trực tiếp đến chi phí điều hướng các khu vực mà công cụ sửa đổi chông chéo trực

tiếp. Bạn cũng có thể biểu thị khu vực bị ảnh hưởng này bằng một màu mới.

Bây giờ, trong ví dụ của chúng ta, chúng ta sẽ phân tích hành vi của AI khi nó đi qua các Công cụ Navigation Modifier khác nhau mà chúng ta thiết lập. Điều bạn muốn hiểu từ điều này là kinh nghiệm giúp tìm ra giải pháp tối ưu nhanh hơn thuật toán Dijkstra một mình trong khi sử dụng cùng một phương pháp khám phá.

Vì vậy, ví dụ: nếu bạn có cảnh quan thành phố và muốn AI của mình ở trên vỉa hè và ngoài đường phố càng nhiều càng tốt, bạn có thể đặt Công cụ Navigation Modifier trên đường phố, điều này khiến việc đi qua khu vực này rất tốn kém. Vì vậy, AI của bạn sẽ chỉ đi qua những khu vực này nếu con đường thích hợp bị chặn bởi các chi phí cao khác, chẳng hạn như chướng ngại vật.

Bây giờ, hãy chuyển sang Level Editor của Unreal Engine 4 và sẵn sàng tạo chướng ngại vật của riêng chúng ta để xem hành vi này hoạt động. Chúng ta cũng sẽ nói về NavModifierVolume và cách bạn có thể sử dụng nó trong Unreal Engine 4.

Quay lại Editor

Hãy điều hướng và nhìn vào panel **Modes**. Trong tab đầu tiên, chúng ta sẽ nhấp chọn vào **Volumes**. Bây giờ, trong **Volumes**, bạn sẽ thấy **NavModifierVolume** và nó sẽ cần được kéo vào màn chơi:

1. Kéo nhiều **NavModifierVolume** vào màn chơi. Chúng ta muốn các khối này chặn các đường dẫn trực tiếp giữa các điểm tham chiếu mà chúng tôi đã tạo trước đó.
2. Vì vậy, sau khi kéo các khối này ra, chúng ta sẽ có một sự tách biệt đẹp mắt giữa **NavModifierVolume** và *Waypoints*. Màn hình sẽ trông giống như ảnh chụp màn hình sau:



3. Tiếp theo, chúng ta cần tạo **NavArea**, nó sẽ ghi đè lên các thuộc tính của Navigation Mesh mà **NavModifierVolume** chồng lên.

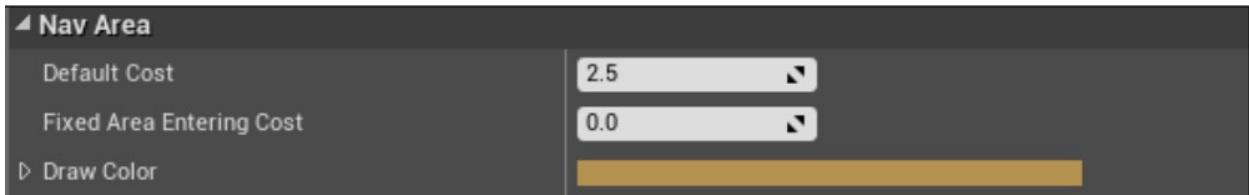
Class NavArea

Class này chịu trách nhiệm áp dụng chi phí đã điều chỉnh của **NavModifierVolume**. Nó cũng có nghĩa là **NavModifierVolume** được sử dụng để tác động đến Navigation Mesh. Các class **NavArea** được sử dụng một cách rõ ràng để ghi đè các thuộc tính hoặc chức năng trong Navigation Mesh bị ảnh hưởng. Hôm nay, chúng ta sẽ tạo hai class **NavArea** để thể hiện hai loại tình huống môi trường khác nhau. Một cái sẽ là đại diện của nước và cái kia là bùn. Lý do đằng sau điều này là nước sẽ có khả năng đi vào cao vì nó không nhanh bằng bước trong bùn, nhưng bùn sẽ có khả năng đi lại thấp vì không dễ đi vào.

Với tính toán này, hãy chuyển sang bước tiếp theo, đó là tạo ra các class **NavArea** khác nhau để áp dụng cho các **NavModifierVolume** actor của chúng ta:

1. Nhấp chuột phải vào thư mục */Blueprints* của chúng ta trong **Content Browser** và chuyển đến **Blueprint Class**.
2. Sau đó, chúng ta sẽ chuyển đến **Custom Class** ở cuối cửa sổ.
3. Tìm kiếm **NavArea** và hãy tạo một subclass mới từ class này.
4. Sau khi tạo thêm một **NavArea**, hãy đặt tên cho một cái là *AreaOfMud* và tên còn lại là *AreaOfWater*.
5. Nhấp đúp vào *AreaOfMud* và đi đến panel **Details**.

6. Trong đó, dưới phần **Nav Area** bạn sẽ thấy **Default Cost** và **Fixed Area Entering Cost**.
7. Chúng ta muốn giá trị **Default Cost** là **2.5** và điều này sẽ ảnh hưởng đến actor **NavModifierVolume** mà nó thuộc về.
8. Đối với **Fixed Area Entering Cost**, chúng ta sẽ để giá trị này ở mức **0.0** đối với bùn. Cuối cùng, thay đổi màu **Draw Color** thành màu nâu để chúng ta biết điều gì đang ảnh hưởng đến Navigation Mesh. Như tôi đã nói trước đó, đi vào bùn tương đối dễ dàng hơn so với nước:



9. Hãy đóng cái này và sau đó mở *AreaOfWater*.
10. Chúng ta sẽ thay đổi **Default Cost** thành **1.5** và hãy thay đổi **Fixed Area Entering Cost** thành **35**. Các giá trị này có thể được điều chỉnh theo trò chơi cụ thể mà bạn đang làm việc. Và tất nhiên, thay đổi **Draw Color** thành thứ gì đó tương tự như màu xanh nhạt để thể hiện nước.
11. Lưu và đóng blueprint class *AreaOfWater* rồi quay lại **Level Editor**.
12. Bây giờ, hãy bắt đầu và áp dụng các **NavArea** này cho **NavModifierVolume** mà chúng ta đã đặt ở cấp độ trước đó.
13. Sau khi bạn thực hiện điều này, bây giờ bạn sẽ thấy **Draw Color** được cập nhật. Hãy nhấn Play và quan sát AI của chúng ta.



Khả năng đi lại (Navigation Cost)

Như bạn có thể thấy, AI thực hiện chính xác như bạn dự đoán. Nó tránh bùn vì khả năng được áp dụng khi nó đi qua **NavModifierVolume** và đôi khi, **Fixed Area Entering Cost** của nước sẽ ngăn chặn AI, mặc dù AI thường xuyên được nhìn thấy khi đi qua nước vì một khi bạn ở trong đó, mọi thứ sẽ ổn. Đây thường là mẹo đối với tôi. Vì vậy, với điều này, tôi hy vọng bạn đã hiểu thuật toán A* và tìm kiếm của nó để tìm đường đi tối ưu nhất.

Tóm tắt

Trong chương này, tôi đã đề cập đến các công cụ khác nhau cho phép chúng tôi kiểm soát và tác động đến hành vi di chuyển của AI bằng cách sử dụng **NavModifierVolumes** và các thuộc tính reagent khác nhau. Chúng ta cũng đã học cách ra lệnh cho quân tốt do AI điều khiển của chúng ta điều hướng đến các điểm tham chiếu thuộc về chúng. Cuối cùng, chúng ta đã đề cập đến nhiều nguyên tắc cơ bản hơn, chẳng hạn như thuật toán đường dẫn ban đầu do Edsger Dijkstra tạo ra. Tiếp theo, chúng ta biết được rằng chúng ta có thể tối ưu hóa thuật toán ban đầu của anh ấy theo hướng có lợi cho hiệu suất và tài nguyên và đạt được điều này thông qua phỏng đoán. Trong chương tiếp theo, chúng ta sẽ tạo một cây hành vi và điều chỉnh những gì chúng ta đã học được từ các chương trước để tạo ra một số tương tác AI thú vị. Chúng ta cũng sẽ tận dụng hệ thống cảm biến được cung cấp trong Unreal Engine 4.