

## Nhà thiết kế hành vi

## Mục lục

Nhà thiết kế hành vi .....	2 Tổng quan .....	3
Cây Hành vi là gì? .....	7	
Cây hành vi hoặc máy trạng thái hữu hạn .....	9	
Cài đặt .....	10	
Thành phần cây hành vi .....	11 Tạo cây hành vi từ tập lệnh .....	13 Giám đốc hành vi .....
vụ .....	14	13 Nhiệm vụ .....
Nhiệm vụ của phụ huynh .....	16	
Viết một công việc có điều kiện mới .....	17 Viết một	
Nhiệm vụ Hành động Mới .....	20 Gõ	
lỗi .....	22 Biến số .....	25
Biến động .....	28 Biến toàn cục .....	28
Tạo các biến được chia sẻ .....	29 Truy cập Các biến từ các đối tượng không phải nhiệm vụ .....	30 Bãi bỏ có Điều kiện .....
kiện .....	30	
Sự kiện .....	33	
Cây hành vi bên ngoài .....	35	
Kết nối mạng .....		
37 Tham khảo các nhiệm vụ .....		
38 Người vẽ đối tượng .....		
39 Bộ đồng bộ hóa biến số .....	41 Đồng bộ hóa hoạt ảnh .....	43 Đối tượng cảnh tham chiếu .....
vụ .....	46	44 Thuộc tính nhiệm
Tích hợp .....	48	
Hệ thống đối thoại .....	48	
Bộ điều khiển ký tự cỡ rộng độ cao .....	51 Người chơi .....	62
chơi .....		
uScript .....		
66 Video .....	69	

# Nhà thiết kế hành vi

## Tổng quát

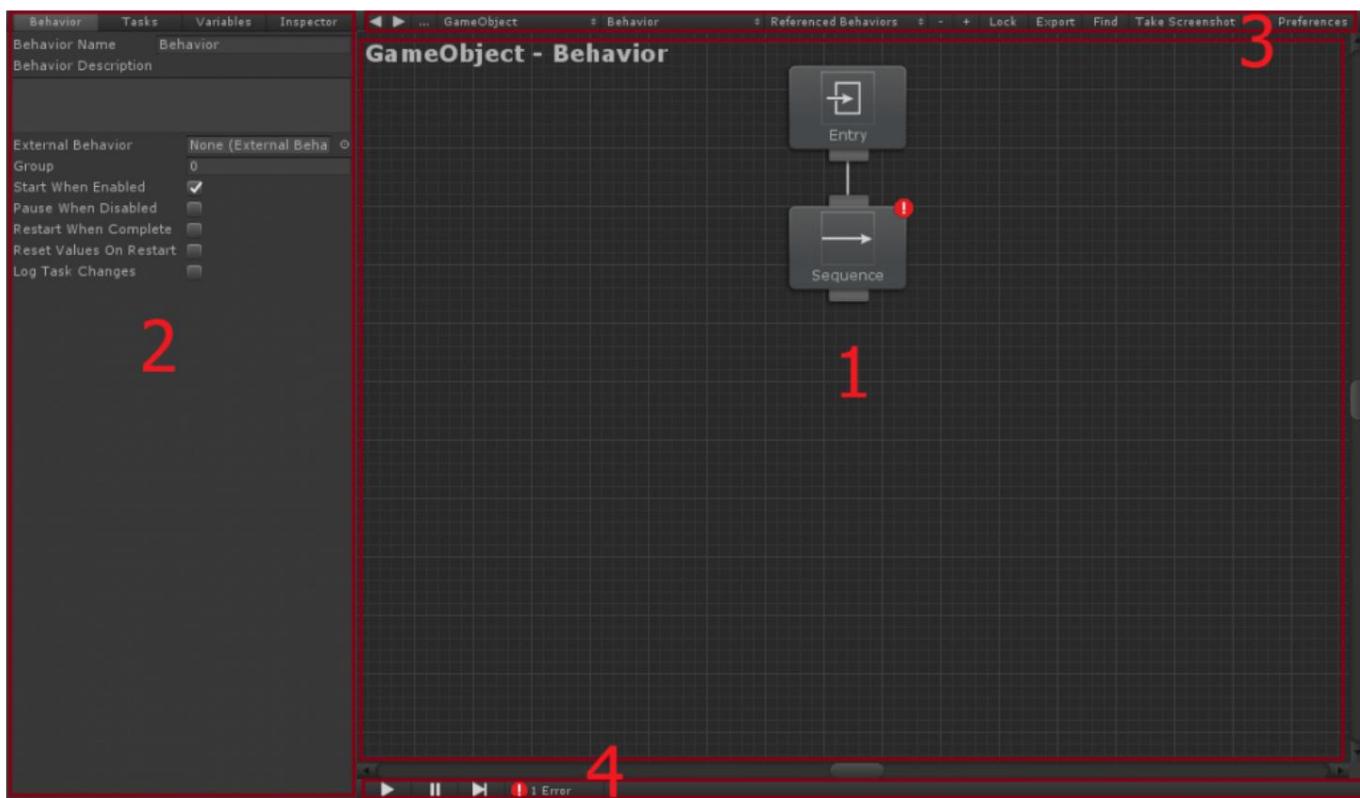
Behavior Designer là một triển khai cây hành vi được thiết kế cho tất cả mọi người - lập trình viên, nghệ sĩ, nhà thiết kế. Behavior Designer cung cấp một API mạnh mẽ cho phép bạn dễ dàng tạo các tác vụ mới. nó cung cấp một trình chỉnh sửa trực quan trực quan với sự tích hợp rộng rãi của bên thứ ba, giúp bạn có thể tạo ra những AI phức tạp mà không cần phải viết một dòng mã.

Hướng dẫn này sẽ cung cấp một cái nhìn tổng thể chung về tất cả các khía cạnh của Behavior Designer. Nếu bạn mới bắt đầu với cây hành vi, chúng tôi có một [loạt video "Kiến thức cơ bản về cây hành vi"](#). Trang này cũng có một [cái nhìn tổng quan](#) của cây hành vi. Với Behavior Designer, bạn không cần phải biết việc triển khai cây hành vi cơ bản như bạn nên biết một số khái niệm chính như các loại tác vụ (hành động, tổng hợp, điều kiện và trình trang trí).

Khi bạn mở Behavior Designer lần đầu tiên, bạn sẽ thấy cửa sổ sau:

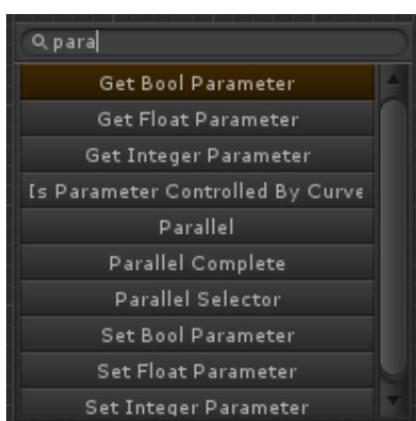


Có bốn phần trong Behavior Designer. Từ ảnh chụp màn hình bên dưới, phần 1 là vùng biểu đồ. Đó là nơi bạn sẽ tạo các cây hành vi. Phần 2 là một bảng thuộc tính. Bảng thuộc tính là nơi bạn sẽ chỉnh sửa các thuộc tính cụ thể của cây hành vi, thêm các tác vụ mới, tạo các biến mới hoặc chỉnh sửa các thông số của một tác vụ. Phần 3 là thanh công cụ hoạt động của cây hành vi. Bạn có thể sử dụng các hộp thả xuống để chọn cây hành vi hiện có hoặc thêm / xóa cây hành vi. Phần cuối cùng, phần 4, là thanh công cụ gỡ lỗi. Bạn có thể bắt đầu / dừng, bù ớc và tạm dừng Unity trong bảng điều khiển này. Ngoài ra, bạn sẽ thấy số lỗi mà cây của bạn mắc phải ngay cả trước khi bạn bắt đầu thực hiện cây của mình.



Phần 1 là phần chính của Behavior Designer mà bạn sẽ làm việc. Trong phần này, bạn có thể tạo các tác vụ mới và sắp xếp các tác vụ đó thành một cây hành vi. Để bắt đầu mọi thứ, trước tiên bạn cần thêm thành phần Cây hành vi. Thành phần Cây hành vi sẽ đóng vai trò là người quản lý cây hành vi mà bạn mới bắt đầu tạo. Bạn có thể tạo thành phần Cây hành vi mới bằng cách nhấp chuột phải vào vùng biểu đồ và nhấp vào "Thêm cây hành vi" hoặc nhấp vào nút dấu cộng bên cạnh "Khóa" trong vùng hoạt động của phần 3.

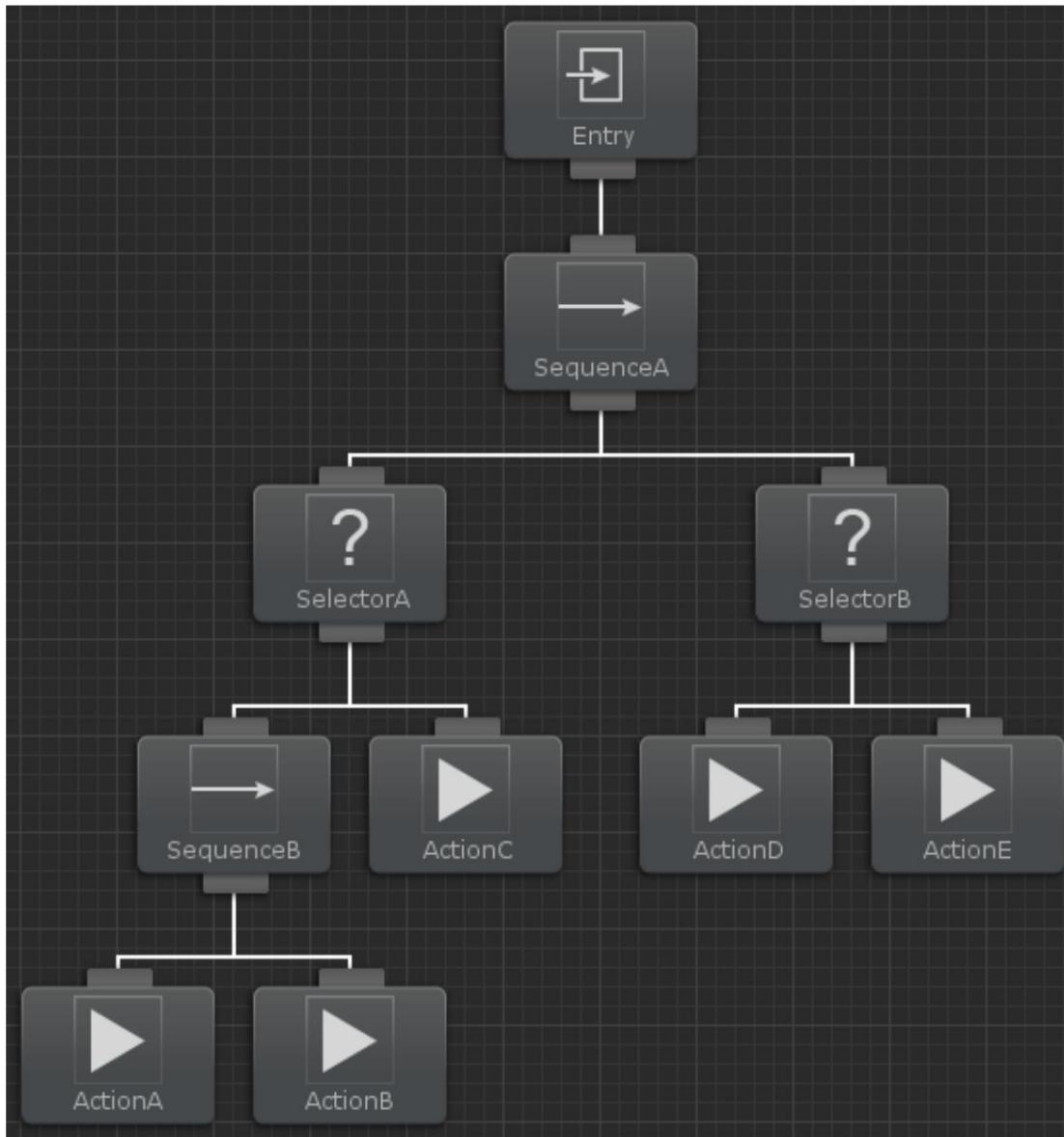
Khi Cây hành vi đã được thêm, bạn có thể bắt đầu thêm nhiệm vụ. Thêm nhiệm vụ bằng cách nhấp chuột phải vào vùng biểu đồ hoặc nhấp vào tab "Công việc" trong phần 2, bảng thuộc tính. Các tác vụ mới cũng có thể được thêm bằng cách nhấn phím cách và mở cửa sổ tìm kiếm tác vụ nhanh:



Sau khi một nhiệm vụ đã được thêm vào, bạn sẽ thấy như sau:



Ngoài nhiệm vụ bạn đã thêm, nhiệm vụ nhập cảnh cũng được thêm vào. Nhiệm vụ nhập đóng vai trò là gốc của cây. Đó là mục đích duy nhất của nhiệm vụ nhập cảnh. Nhiệm vụ trình tự có lỗi vì nó không có phần tử con. Ngay sau khi bạn thêm một đứa trẻ, lỗi sẽ biến mất. Bây giờ chúng tôi đã thêm nhiệm vụ đầu tiên của mình, hãy thêm một số tác vụ khác:

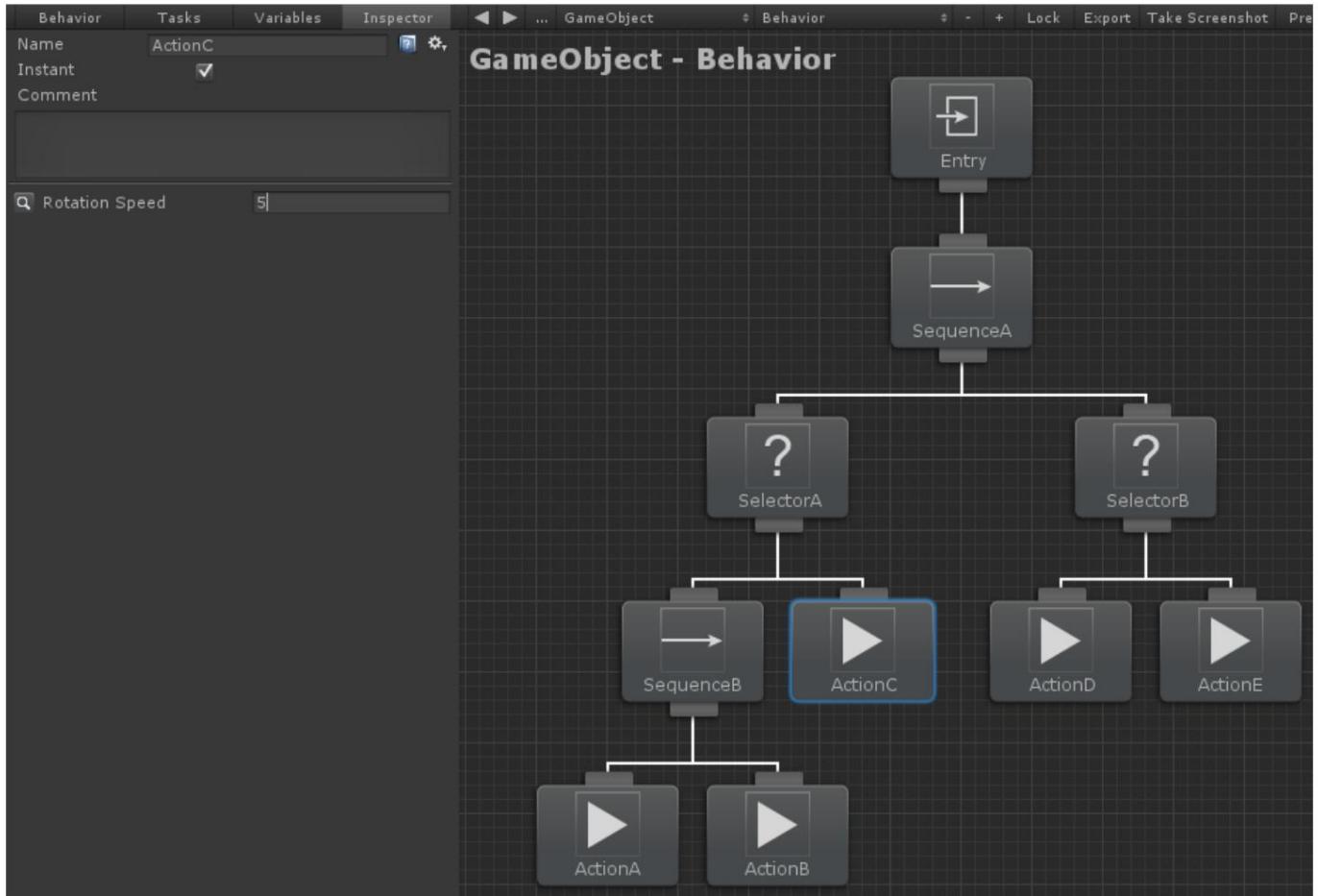


Bạn có thể kết nối tác vụ trình tự và bộ chọn bằng cách kéo từ cuối tác vụ trình tự lên đầu tác vụ bộ chọn. Lặp lại quá trình này cho các tác vụ còn lại. Nếu bạn mắc lỗi, bạn có thể chọn một kết nối và xóa nó bằng phím xóa. Bạn

cũng có thể sắp xếp lại các nhiệm vụ bằng cách nhấp vào một nhiệm vụ và kéo nó xung quanh.

Behavior Designer sẽ thực hiện các tác vụ theo thứ tự đầu tiên chuyên sâu. Bạn có thể thay đổi thứ tự thực thi của các tác vụ bằng cách kéo chúng sang trái / phải của anh chị em của chúng. Từ ảnh chụp màn hình ở trên, các tác vụ sẽ được thực hiện theo thứ tự sau:

SequenceA, SelectorA, SequenceB, ActionA, ActionB, ActionC, SelectorB, ActionD, ActionE



Bây giờ chúng ta đã tạo một cây hành vi cơ bản, hãy sửa đổi các tham số trong các tác vụ. Chọn nút ActionC để hiển thị Trình kiểm tra trong bảng thuộc tính. Bạn có thể thấy ở đây rằng chúng ta có thể đổi tên nhiệm vụ, đặt nhiệm vụ thành tức thi hoặc nhập nhận xét nhiệm vụ. Ngoài ra, chúng ta có thể sửa đổi tất cả các biến công khai mà lớp tác vụ chưa. Điều này bao gồm việc gán các biến được tạo trong Behavior Designer. Trong trường hợp của chúng tôi, biến công khai duy nhất là Tốc độ quay. Giá trị mà chúng tôi đặt tham số sẽ được sử dụng trong cây hành vi.

Có ba tab khác trong bảng thuộc tính: Biến, Nhiệm vụ và Hành vi. Bảng điều khiển biến cho phép bạn tạo các biến được chia sẻ giữa các tác vụ. Để biết thêm thông tin, hãy xem [chủ đề biến](#). Bảng nhiệm vụ liệt kê tất cả các tác vụ khả thi mà bạn có thể sử dụng. Đây là danh sách giống như những gì được tìm thấy khi bạn nhấp chuột phải và thêm một nhiệm vụ.

Danh sách này được tạo bằng cách tìm kiếm bất kỳ lớp nào có nguồn gốc từ loại tác vụ action, composite, có điều kiện hoặc decorator. Bảng cuối cùng, bảng hành vi, hiển thị trình kiểm tra cho thành phần Cây hành vi mà bạn đã thêm khi lần đầu tiên tạo cây hành vi.

Thông tin chi tiết về chức năng của từng tùy chọn có trên [trang Tổng quan về Thành phần Hành vi](#).

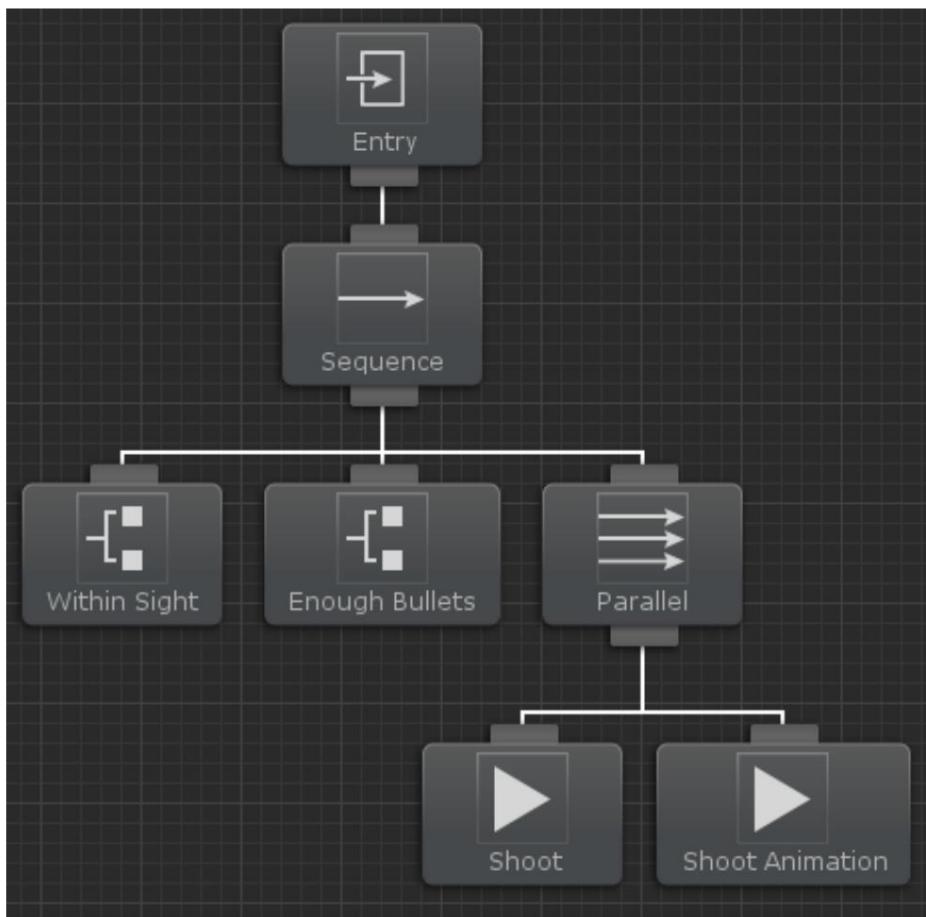


Phần cuối cùng cửa sổ Behavior Designer là thanh công cụ hoạt động. Thanh công cụ hoạt động chủ yếu đư ợc sử dụng để chọn cây hành vi cũng như thêm / xóa cây hành vi. Các thao tác sau đư ợc gán nhãn:

- Nhãn 1: Điều hướng tới / lui giữa các cây hành vi mà bạn đã mở.
- Nhãn 2: Liệt kê tất cả các cây hành vi có trong khung cảnh hoặc dự án (bao gồm cả prefabs).
- Nhãn 3: Liệt kê bất kỳ GameObject nào trong cảnh có thành phần cây hành vi đư ợc thêm vào nó.
- Nhãn 4: Liệt kê bất kỳ cây hành vi nào đư ợc gắn vào GameObject đư ợc chọn từ nhãn 3.
  
- Nhãn 5: Liệt kê bất kỳ cây hành vi bên ngoài nào mà cây hành vi hiện tại tham chiếu đến.
- Nhãn 6: Loại bỏ cây hành vi hiện đư ợc chọn.
- Nhãn 7: Thêm cây hành vi mới vào GameObject.
- Nhãn 8: Giữ cây hành vi hiện tại hoạt động ngay cả khi bạn đã chọn một GameObject khác trong cửa sổ phân cấp hoặc dự án. Nhãn 9: Xuất cây hành vi sang nội dung cây hành vi bên ngoài.
- Nhãn 10: Mở hộp thoại tìm kiếm có thể tìm kiếm cây hành vi của bạn.
- Nhãn 11: Chụp ảnh màn hình của cây hành vi hiện tại.
- Nhãn 12: Hiển thị các tùy chọn của Nhà thiết kế Hành vi.

## Cây Hành vi là gì?

Cây hành vi là một kỹ thuật AI phổ biến đư ợc sử dụng trong nhiều trò chơi. Halo 2 là trò chơi chính thống đầu tiên sử dụng cây hành vi và chúng bắt đầu trở nên phổ biến hơn sau khi [mô tả chi tiết](#) về cách chúng đư ợc sử dụng trong Halo 2 đã đư ợc phát hành. Cây hành vi là sự kết hợp của nhiều kỹ thuật AI khác nhau: máy trạng thái phân cấp, lập lịch, lập kế hoạch và thực thi hành động. Một trong những ưu điểm chính của chúng là dễ hiểu và có thể đư ợc tạo bằng trình chỉnh sửa trực quan.



Ở cấp độ đơn giản nhất, cây hành vi là một tập hợp các nhiệm vụ. Có bốn loại nhiệm vụ khác nhau: hành động, có điều kiện, kết hợp và trang trí. Các nhiệm vụ hành động có lẽ dễ hiểu nhất ở chỗ chúng thay đổi trạng thái của trò chơi theo một cách nào đó. Nhiệm vụ có điều kiện kiểm tra một số thuộc tính của trò chơi. Ví dụ, trong cây ở trên, tác nhân AI có hai nhiệm vụ điều kiện và hai tác vụ hành động. Hai nhiệm vụ có điều kiện đầu tiên kiểm tra xem có kẻ thù trong tầm nhìn của đặc vụ hay không và sau đó đảm bảo đặc vụ có đủ đạn để bắn vũ khí của mình.

Nếu cả hai điều kiện này đều đúng thì hai tác vụ hành động sẽ chạy. Một trong các nhiệm vụ hành động bắn vũ khí và nhiệm vụ còn lại đóng hoạt cảnh bắn súng. Sức mạnh thực sự của cây hành vi phát huy tác dụng khi bạn tạo thành các cây con khác nhau. Hai hành động chụp có thể tạo thành một cây con. Nếu một trong những nhiệm vụ có điều kiện trước đó không thành công thì một cây con khác có thể được tạo ra để đóng một tập hợp các nhiệm vụ hành động khác, chẳng hạn như chạy trốn khỏi kẻ thù. Bạn có thể nhóm các cây con chồng lên nhau để tạo thành một hành vi cấp cao.

Nhiệm vụ tổng hợp là một nhiệm vụ mẹ chứa một danh sách các nhiệm vụ con. Từ ví dụ trên, các nhiệm vụ tổng hợp được gán nhãn là trình tự và song song. Một tác vụ tuần tự chạy mỗi tác vụ một lần cho đến khi tất cả các tác vụ đã được chạy. Đầu tiên, nó chạy nhiệm vụ có điều kiện để kiểm tra xem có kẻ thù trong tầm nhìn hay không. Nếu kẻ thù trong tầm nhìn thì nó sẽ chạy nhiệm vụ có điều kiện để kiểm tra xem đặc vụ có còn viên đạn nào không. Nếu đặc vụ có đủ đạn thì nhiệm vụ song song sẽ chạy đó là bắn vũ khí và phát hoạt ảnh bắn súng. Trong đó một tác vụ tuần tự thực hiện một tác vụ con tại một thời điểm, một tác vụ song song thực hiện tất cả các tác vụ con của nó cùng một lúc.

Loại nhiệm vụ cuối cùng là nhiệm vụ trang trí. Tác vụ decorator là tác vụ mẹ chỉ có thể có một con. Chức năng của nó là sửa đổi hành vi của nhiệm vụ con theo một cách nào đó. Trong ví dụ trên, chúng tôi không sử dụng tác vụ decorator như ng bạn có thể muốn sử dụng tác vụ này nếu bạn muốn dừng một tác vụ chạy sớm (được gọi là tác vụ ngắn). Ví dụ, một đại lý có thể đang thực hiện một nhiệm vụ chẳng hạn như thu thập tài nguyên. Sau đó nó có thể có một tác vụ ngắn | 9

điều đó sẽ ngăn chặn việc thu thập tài nguyên nếu có kẻ thù ở gần. Một ví dụ khác về nhiệm vụ trang trí là nhiệm vụ chạy lại nhiệm vụ con của nó x số lần hoặc nhiệm vụ trang trí tiếp tục chạy nhiệm vụ con cho đến khi nó hoàn thành thành công.

Một trong những chủ đề chính của cây hành vi mà chúng tôi đã bỏ qua cho đến nay là trạng thái trả về của một tác vụ. Bạn có thể có một nhiệm vụ cần nhiều hơn một khung để hoàn thành. Ví dụ: hầu hết các hoạt động sẽ không bắt đầu và kết thúc chỉ trong một khung hình. Ngoài ra, các tác vụ có điều kiện cần một cách để cho tác vụ cha của chúng biết liệu điều kiện có đúng hay không để tác vụ cha có thể quyết định xem nó có nên tiếp tục chạy các tác vụ con của nó hay không. Cả hai vấn đề này đều có thể được giải quyết bằng cách sử dụng trạng thái nhiệm vụ. Một nhiệm vụ ở một trong ba trạng thái khác nhau: đang chạy, thành công hoặc thất bại. Trong ví dụ đầu tiên, nhiệm vụ hoạt hình bắn súng có trạng thái tác vụ chạy trong thời gian hoạt động bắn súng đang phát. Nhiệm vụ có điều kiện là xác định xem kẻ thù có trong tầm nhìn hay không sẽ trả về thành công hoặc thất bại trong một khung hình.

Behavior Designer lấy tất cả các khái niệm này và đóng gói nó trong một giao diện dễ sử dụng với một API tương tự như API MonoBehaviour của Unity. Behavior Designer bao gồm nhiều lớp tổng hợp và lớp trang trí trong cài đặt tiêu chuẩn. Các nhiệm vụ hành động và có điều kiện là trò chơi cụ thể hơn nên không có nhiều nhiệm vụ đó được bao gồm như ng có rất nhiều ví dụ trong các [dự án mẫu](#). Các nhiệm vụ mới có thể được tạo bằng cách [mở rộng từ một trong các loại nhiệm vụ](#). Ngoài ra, nhiều [video](#) đã được tạo ra để giúp việc học Behavior Designer trở nên dễ dàng nhất [có thể](#).

## Cây hành vi hoặc Máy trạng thái hữu hạn

Trong những tình huống nào bạn sử dụng cây hành vi trên một máy trạng thái hữu hạn (chẳng hạn như Playmaker)? Ở cấp cao nhất, cây hành vi được sử dụng cho AI trong máy trạng thái hữu hạn (FSM) để lập trình trực quan tổng quát hơn. Mặc dù bạn có thể sử dụng cây hành vi để lập trình trực quan chung và máy trạng thái hữu hạn cho AI, nhưng đây không phải là những gì mỗi công cụ được thiết kế để làm. Theo một số người, thời đại của những [cỗ máy trạng thái hữu hạn đã qua](#). Chúng ta sẽ không đi xa đến vậy, nhưng cây hành vi chắc chắn có lợi thế hơn các máy trạng thái hữu hạn khi nói đến AI.

Cây hành vi có một vài ưu điểm so với FSM: chúng cung cấp nhiều tính linh hoạt, rất mạnh mẽ và chúng thực sự dễ dàng thực hiện các thay đổi.

Đầu tiên chúng ta hãy nhìn vào ưu điểm đầu tiên: tính linh hoạt. Với FSM, làm cách nào để bạn chạy hai trạng thái khác nhau cùng một lúc? Cách duy nhất là tạo hai FSM riêng biệt. Với cây hành vi, tất cả những gì bạn cần làm là thêm tác vụ song song và bạn đã hoàn thành - tất cả các tác vụ con sẽ chạy song song. Với Behavior Designer, những nhiệm vụ con đó có thể là một FSM của PlayMaker và những FSM đó sẽ chạy song song.

Thêm một ví dụ về tính linh hoạt là nhiệm vụ bảo vệ tác vụ. Trong ví dụ này, bạn có hai tác vụ khác nhau phát hiệu ứng âm thanh. Hai nhiệm vụ khác nhau nằm ở hai nhánh khác nhau của cây hành vi nên chúng không biết về nhau và có thể phát hiệu ứng âm thanh cùng một lúc. Bạn không muốn điều này xảy ra vì nghe có vẻ không ổn.

Trong trường hợp này, bạn có thể thêm một nhiệm vụ semaphore (được gọi là Task Guard trong Behavior Designer) và nó sẽ chỉ cho phép một hiệu ứng âm thanh phát tại một thời điểm. Khi âm thanh đầu tiên kết thúc, âm thanh thứ hai sẽ bắt đầu phát.

Một ưu điểm khác của cây hành vi là chúng rất mạnh mẽ. Điều đó không có nghĩa là FSM không mạnh, chỉ là chúng mạnh theo những cách khác nhau. Theo quan điểm của chúng tôi, cây hành vi cho phép AI của bạn phản ứng với trạng thái trò chơi hiện tại dễ dàng hơn so với các máy trạng thái hữu hạn. Dễ dàng hơn để tạo một cây hành vi phản ứng với tất cả các loại tình huống trong khi sẽ mất rất nhiều trạng thái và chuyển đổi với một máy trạng thái hữu hạn để có AI tương tự. Để đạt được kết quả tương tự với FSM, bạn sẽ kết thúc với một [máy làm mì spaghetti](#).

---

Một ưu điểm cuối cùng của cây hành vi là chúng thực sự dễ dàng thực hiện các thay đổi. Một trong những lý do khiến cây hành vi trở nên phổ biến là vì chúng dễ tạo bằng trình chỉnh sửa trực quan. Nếu bạn muốn thay đổi thứ tự thực thi trạng thái với FSM, bạn phải thay đổi chuyển đổi giữa các trạng thái. Với cây hành vi, tất cả những gì bạn phải làm là kéo tác vụ. Bạn không phải lo lắng về quá trình chuyển đổi. Ngoài ra, thực sự dễ dàng thay đổi hoàn toàn cách AI phản ứng với các tình huống khác nhau chỉ bằng cách thay đổi các nhiệm vụ xung quanh hoặc thêm một nhiệm vụ mới vào một nhánh nhiệm vụ.

Như đã nói, cây hành vi và FSM không cần phải loại trừ lẫn nhau. Cây hành vi có thể mô tả luồng AI trong khi FSM mô tả chức năng. Sự kết hợp này cung cấp cho bạn sức mạnh của cây hành vi trong khi vẫn có chức năng của FSM.

## Cài đặt

Sau khi thiết kế hành vi được nhập, bạn có thể truy cập nó từ thanh công cụ Tools. Bạn có thể truy cập mã nguồn thời gian chạy bằng cách nhập gói Mã nguồn thời gian chạy [tại đây](#) trên trang Tải xuống. Trước khi giải nén gói này, hãy đảm bảo rằng bạn đã xóa thư mục Behavior Designer, nếu không, bạn sẽ gặp lỗi biên dịch.

Để biên dịch Behavior Designer cho Universal Windows Platform (UWP), bạn phải sử dụng mã nguồn thời gian chạy thay vì DLL đã biên dịch. Không cần thay đổi cài đặt biên dịch - Behavior Designer có thể biên dịch với .Net Core được bật.

Khi bạn cố gắng chạy ứng dụng UWP của mình, bạn có thể gặp lỗi cho biết rằng không thể tìm thấy các tác vụ. Để khắc phục điều này, dòng sau trong TaskUtility.GetTypeWithinAssembly (trong TaskUtility.cs) nên được thay đổi từ:

```
loadAssemblies = GetStorageFileAssemblies (typeName) .Result;
```

đến:

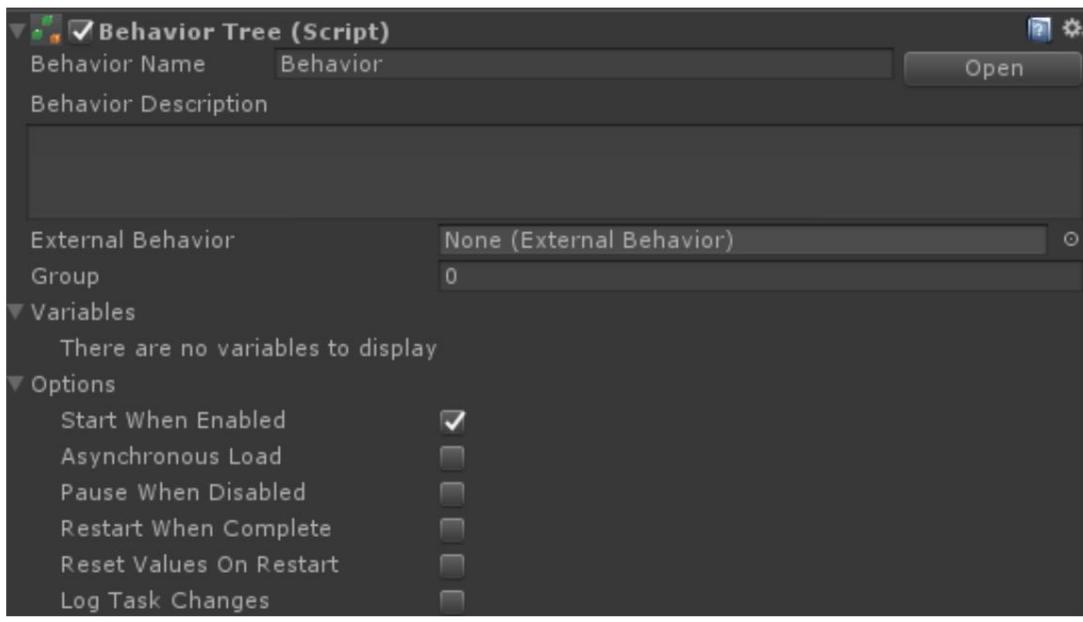
```
loadAssemblies = new List ();
loadAssemblies.Add ("Assembly-CSharp");
```

Điều này sẽ cho phép tập hợp C # của Unity được tìm thấy trong ứng dụng.

Nếu sử dụng IL2CPP để xây dựng dự án của bạn, bạn có thể nhận được một MissingMethodException liên quan đến hàm tạo mặc định không được tìm thấy. Điều này có thể được khắc phục bằng cách thực hiện một trong các thao tác sau:

- Nhập gói nguồn thời gian chạy.
- Thay đổi mức ưu tiên thành Thấp.
- Thêm lớp bị thiếu vào [link.xml tập tin](#).

# Thành phần cây hành vi



Thành phần cây hành vi lưu trữ cây hành vi của bạn và hoạt động như giao diện giữa Nhà thiết kế hành vi và các tác vụ. API sau được sử dụng để bắt đầu và dừng cây hành vi của bạn:

```
public void EnableBehavior (); //  
pause: tạm thời dừng cây hành vi tại điểm thực thi hiện tại của nó. Nó có thể được tiếp tục  
với EnableBehavior. public void DisableBehavior (bool pause = false);
```

Bạn có thể tìm nhiệm vụ bằng một trong các phương pháp sau:

```
TaskType FindTask <TaskType> ();  
List <TaskType> FindTasks <TaskType> ();  
Tác vụ FindTaskWithName (string taskName);  
List <Task> FindTasksWithName (string taskName);
```

Trạng thái thực thi hiện tại của cây có thể được lấy bằng cách gọi:

```
behaviorTree.ExecutionStatus
```

Trạng thái Đang chạy sẽ được trả lại khi cây đang chạy. Khi cây kết thúc trạng thái thực thi sẽ là Thành công hoặc Thất bại tùy thuộc vào kết quả tác vụ.

Các sự kiện sau cũng có thể được đăng ký:

```
OnBehaviorStart  
OnBehaviorRestart  
OnBehaviorEnd
```

Thành phần cây hành vi có các thuộc tính sau:

Tên hành vi

Tên của cây hành vi.

Mô tả hành vi

Mô tả chức năng của cây hành vi.

Hành vi bên ngoài

Một tru ờng để chỉ định cây hành vi bên ngoài sẽ được chạy khi cây hành vi này bắt đầu.

Tập đoàn

Một nhóm số cây hành vi. Có thể được sử dụng để dễ dàng tìm thấy cây hành vi. Dự án mẫu CTF cho thấy một ví dụ về điều này.

Bắt đầu khi được kích hoạt

Nếu đúng, cây hành vi sẽ bắt đầu chạy khi thành phần được kích hoạt.

Tải không đồng bộ

Chỉ định xem cây hành vi có nên tải trong một luồng riêng biệt hay không. Bởi vì Unity không cho phép thực hiện lệnh gọi API trên các luồng công nhân, tùy chọn này sẽ bị vô hiệu hóa nếu bạn đang sử dụng ánh xạ thuộc tính cho các [biến được chia sẻ](#).

Tạm dừng khi bị tắt

Nếu đúng, cây hành vi sẽ tạm dừng khi thành phần bị vô hiệu hóa. Nếu sai, cây hành vi sẽ kết thúc.

Khởi động lại khi hoàn tất

Nếu đúng, cây hành vi sẽ khởi động lại từ đầu khi nó hoàn thành việc thực thi. Nếu sai, cây hành vi sẽ kết thúc.

Đặt lại giá trị khi khởi động lại

Nếu đúng, các biến và biến công khai của nhiệm vụ sẽ được đặt lại về giá trị ban đầu của chúng khi cây khởi động lại.

Ghi nhật ký thay đổi tác vụ

Được sử dụng để gỡ lỗi. Nếu được bật, cây hành vi sẽ xuất ra bất kỳ lúc nào trạng thái tác vụ thay đổi, chẳng hạn như nó bắt đầu hoặc dừng.

# Tạo cây hành vi từ Script

Trong một số trường hợp, bạn có thể muốn tạo cây hành vi từ tập lệnh thay vì trực tiếp dựa vào nhà lắp ghép để chứa cây hành vi cho bạn. Ví dụ: bạn có thể đã lưu [cây hành vi bên ngoài](#) và muốn [tải cây đó vào từ cây hành vi mới](#) được tạo. Điều này có thể thực hiện được bằng cách đặt biến `externalBehavior` trên thành phần cây hành vi:

```
sử dụng UnityEngine; sử
dụng BehaviorDesigner.Runtime;

public class CreateTree: MonoBehaviour {

    công khai Bên ngoài Hành viTree Hành viTree;

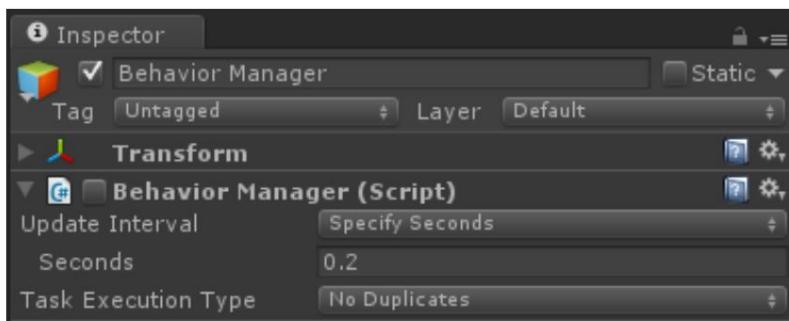
    private void Start () {

        var bt = gameObject.AddComponent <BehaviorTree> ();
        bt.StartWhenEnabled = false; bt.ExternalBehavior = behaviorTree;

    }
}
```

Trong ví dụ này, biến công khai `behaviorTree` chứa một tham chiếu đến cây hành vi bên ngoài của bạn. Khi tải cây mới tạo, nó sẽ tải cây hành vi bên ngoài cho tất cả các tác vụ của nó. Để ngăn cây chạy ngay lập tức, chúng tôi đặt `StartWhenEnabled` thành `false`. Sau đó, cây có thể được khởi động theo cách thủ công với `bt.EnableBehavior ()`.

## Quản lý hành vi



Khi cây hành vi chạy nó sẽ tạo một `GameObject` mới với thành phần Trình quản lý hành vi nếu nó chưa được tạo. Thành phần này quản lý việc thực thi tất cả các cây hành vi trong cảnh của bạn.

Bạn có thể kiểm soát tần suất đánh dấu của cây hành vi bằng cách thay đổi thuộc tính khoảng thời gian cập nhật. "Every Frame" sẽ đánh dấu vào cây hành vi mỗi khung trong vòng lặp Cập nhật. "Chỉ định số giây" cho phép bạn đánh dấu vào cây hành vi một số giây nhất định. Tùy chọn cuối cùng là "Thủ công" sẽ cho phép bạn kiểm soát thời điểm đánh dấu vào các cây hành vi. Bạn có thể đánh dấu vào cây hành vi bằng cách gọi  `đánh dấu:`

```
BehaviorManager.instance.Tick();
```

Ngoài ra, nếu bạn muốn mỗi cây hành vi có tỷ lệ đánh dấu riêng, bạn có thể đánh dấu vào từng cây hành vi theo cách thủ công bằng:

```
BehaviorManager.instance.Tick (BehaviorTree)
```

Loại thực thi tác vụ cho phép bạn chỉ định xem cây hành vi có nên tiếp tục thực thi các tác vụ cho đến khi nó chạm vào một tác vụ đã được thực thi trong lần đánh dấu đó hay không hoặc liệu nó có nên tiếp tục thực hiện các tác vụ cho đến khi số lượng tác vụ tối đa đã được thực hiện trong lần đánh dấu đó hay không. Ví dụ, hãy xem xét cây hành vi sau:



Tác vụ Bộ lặp được đặt để lặp lại 5 lần. Nếu Loại thực thi tác vụ được đặt thành Không trùng lặp, tác vụ Phát âm thanh sẽ chỉ thực hiện một lần trong một lần đánh dấu. Nếu Loại thực thi tác vụ được đặt thành Đếm, thì số lần thực thi tác vụ tối đa có thể được chỉ định. Nếu giá trị 5 được chỉ định thì tác vụ Phát âm thanh sẽ thực hiện tất cả 5 lần trong một lần đánh dấu.

## Nhiệm vụ

Ở cấp độ cao nhất, cây hành vi là một tập hợp các nhiệm vụ. Task có một API tương tự như MonoBehaviour của Unity, vì vậy bạn sẽ thực sự dễ dàng bắt đầu [viết các công việc của riêng mình](#). Lớp tác vụ có API sau:

```
// OnAwake được gọi một lần khi cây hành vi được kích hoạt. Hãy coi nó như một phuơng thức khởi tạo. void OnAwake ();
```

```
// OnStart được gọi ngay trước khi thực thi. Nó được sử dụng để thiết lập bất kỳ biến nào cần được đặt lại từ lần chạy trước. void OnStart ();
```

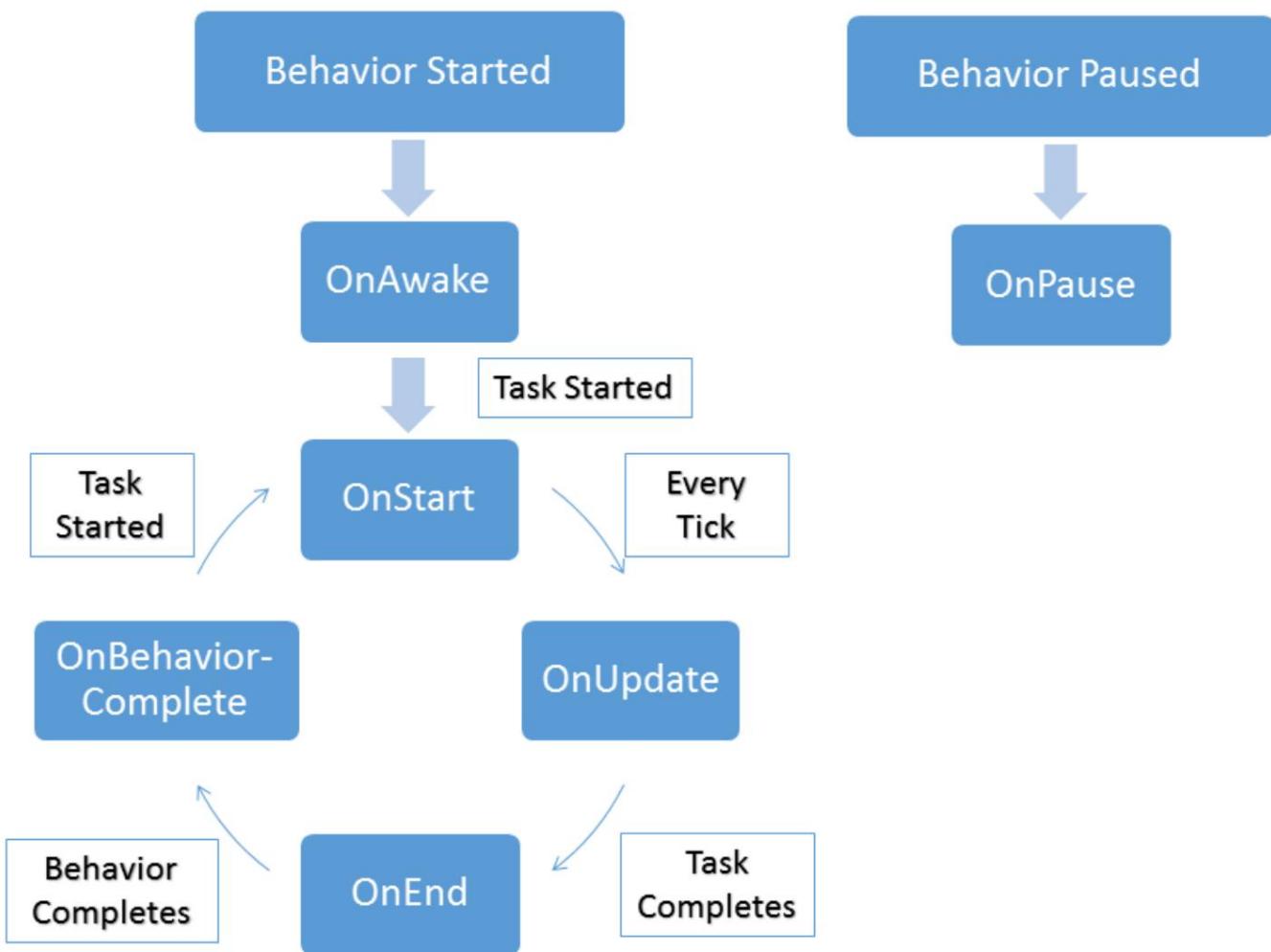
```
// OnUpdate chạy tác vụ thực tế.  
TaskStatus OnUpdate ();
```

```
// OnFixedUpdate thực thi trong vòng lặp FixedUpdate. TaskStatus phải được trả lại trong  
OnUpdate. void OnFixedUpdate ();
```

```
// OnEnd đư ợc gọi sau khi thực thi thành công hay thất bại. void OnEnd ();  
  
// OnPause đư ợc gọi khi hành vi bị tạm dừng hoặc tiếp tục. void OnPause (tạm dừng  
bool);  
  
// Trả về mức độ ưu tiên của tác vụ, đư ợc Bộ chọn Ưu tiên sử dụng. float GetPosystem ();  
  
// Trả về tiện ích của tác vụ, đư ợc sử dụng bởi Bộ chọn tiện ích cho Lý thuyết tiện ích.  
float GetUtility ();  
  
// OnBehaviorComplete đư ợc gọi sau khi cây hành vi kết thúc việc thực thi. void  
OnBehaviorComplete ();  
  
// OnReset đư ợc gọi bởi người kiểm tra để thiết lập lại các thuộc tính công cộng void OnReset  
();  
  
// Cho phép gọi OnDrawGizmos từ các tác vụ. void OnDrawGizmos ();  
  
// Giữ một tham chiếu đến hành vi sở hữu tác vụ này.  
Chủ sở hữu hành vi;
```

Tác vụ có ba thuộc tính hiển thị: tên, nhận xét và tức thì. Instant là tài sản duy nhất không rõ ràng về những gì nó làm. Khi một nhiệm vụ trả về thành công hoặc thất bại, nó sẽ ngay lập tức chuyển sang tác vụ tiếp theo trong cùng một lần đánh dấu cập nhật. Nếu bạn bỏ chọn tác vụ tức thì, nó sẽ đợi một tích tắc cập nhật trước khi tác vụ tiếp theo đư ợc thực thi. Đây là một cách dễ dàng để điều chỉnh cây hành vi.

Lưu ý sau đư ợc sử dụng khi thực hiện tác vụ:



Nhiệm vụ dành cho cha mẹ

Nhiệm vụ cha là các nhiệm vụ tổng hợp và trang trí trong cây hành vi. Mặc dù API ParentTask không có API tương ứng đư ơng với lớp MonoBehaviour của Unity, nhưng vẫn khá dễ dàng để xác định từng phu ơng thức đư ợc sử dụng để làm gì.

```
// Số lư ợng con tối đa mà một tác vụ cha có thể có. Thư ờng sẽ là 1 hoặc int.MaxValue public virtual int MaxChildren();
```

```
// Giá trị boolean để xác định xem tác vụ hiện tại có phải là tác vụ song song hay không. công khai bool ảo CanRunParallelChildren();
```

```
// Chỉ mục của con hiện đang hoạt động. public virtual int CurrentChildIndex();
```

```
// Giá trị boolean để xác định xem tác vụ hiện tại có thể thực thi hay không. public virtual bool CanExecute();
```

```
// Áp dụng trình trang trí cho trạng thái đư ợc thực thi. công khai trang trí TaskStatus ảo (trạng thái TaskStatus);
```

```
// Thông báo tác vụ cha rằng con đã đư ợc thực thi và có trạng thái là childStatus. | 17
```

```

public virtual void OnChildExecuted (TaskStatus childStatus);

// Thông báo tác vụ cha rằng con tại index childIndex đã được thực thi và có trạng thái là childStatus. public virtual void OnChildExecuted (int childIndex, TaskStatus childStatus);

// Thông báo tác vụ con đã bắt đầu chạy. public virtual void OnChildStarted ();

// Thông báo tác vụ song song mà con tại index childIndex đã bắt đầu chạy. public virtual void OnChildStarted (int childIndex);

// Một số tác vụ mẹ cần có khả năng ghi đè trạng thái, chẳng hạn như các tác vụ song song. public virtual TaskStatus OverrideStatus (trạng thái TaskStatus);

// Nút ngắt sẽ ghi đè trạng thái nếu nó đã bị ngắt. public virtual TaskStatus OverrideStatus ();

// Thông báo cho tác vụ tổng hợp rằng việc hủy bỏ có điều kiện đã được kích hoạt và chỉ mục con sẽ được đặt lại. public virtual void OnConditionalAbort (int childIndex);

```

## Viết một nhiệm vụ có điều kiện mới

Chủ đề này được chia thành hai phần. Phần đầu tiên mô tả việc viết một nhiệm vụ có điều kiện mới và phần thứ hai ([có sẵn tại đây](#)) mô tả việc viết một nhiệm vụ hành động mới. Tác vụ có điều kiện sẽ xác định xem có đối tượng nào nằm trong tầm nhìn hay không và lớp hành động sẽ hướng tới đối tượng nằm trong tầm nhìn. Chúng tôi cũng sẽ sử dụng [các biến](#) cho cả hai nhiệm vụ này. Chúng tôi cũng đã quay một video về chủ đề này và nó có sẵn trên [YouTube](#).

Nhiệm vụ đầu tiên mà chúng ta sẽ viết là nhiệm vụ Trong tầm nhìn. Vì nhiệm vụ này sẽ không thay đổi trạng thái trò chơi và chỉ là kiểm tra trạng thái của trò chơi nên nhiệm vụ này sẽ bắt nguồn từ nhiệm vụ Điều kiện. Đảm bảo bạn có bao gồm không gian tên BehaviorDesigner.Runtime.Tasks:

```

sử dụng UnityEngine;
sử dụng BehaviorDesigner.Runtime.Tasks;

lớp công khai WithinSight: Có điều kiện {}

```

Bây giờ chúng ta cần tạo ba biến công khai và một biến riêng:

```
sử dụng UnityEngine;
```

```
sử dụng BehaviorDesigner.Runtime; sử
dụng BehaviorDesigner.Runtime.Tasks;

lớp công khai WithinSight: Có điều kiện {

    public float fieldOfViewAngle; public
    string targetTag; mục tiêu
    SharedTransform công khai;

    riêng tư Transform [] có thểTargets;
}
```

Trường góc nhìn là trường nhìn mà đối tượng có thể nhìn thấy. Thủ mục tiêu là thủ của các mục tiêu mà đối tượng có thể hướng tới. Mục tiêu là một biến được chia sẻ sẽ được sử dụng bởi cả nhiệm vụ Trong tầm nhìn và Hướng tới. Nếu bạn đang sử dụng Biến được chia sẻ, hãy đảm bảo bạn bao gồm không gian tên BehaviorDesigner.Runtime. Biến cuối cùng, các mục tiêu có thể, là bộ nhớ đệm của tất cả các Biến đổi có thể đích. Nếu bạn xem qua [API nhiệm vụ](#), bạn có thể thấy rằng chúng tôi có thể tạo bộ nhớ đệm sẵn đó trong phương pháp OnAwake hoặc OnStart. Vì danh sách các chuyển đổi có thể sẽ không thay đổi khi tác vụ Within Sight được bật / tắt, chúng tôi sẽ thực hiện bộ nhớ đệm trong OnAwake:

```
ghi đè công khai void OnAwake () {

    var target = GameObject.FindGameObjectsWithTag (targetTag); có thểTargets =
    new Transform [target.Length]; for (int i = 0; i <target.Length; ++ i) {

        có thểTargets [i] = target [i] .transform;
    }
}
```

Phương thức OnAwake này sẽ tìm tất cả các GameObject có thủ mục tiêu, sau đó lặp qua chúng trong bộ nhớ đệm chuyển đổi của chúng trong mảng mục tiêu có thể. Sau đó, mảng mục tiêu có thể được sử dụng bởi phương thức OnUpdate bị ghi đè:

```
ghi đè công khai TaskStatus OnUpdate () {

    for (int i = 0; i <couldTargets.Length; ++ i) {
        if (WithinSight (couldTargets [i], fieldOfViewAngle)) {target.Value =
            couldTargets [i]; trả về TaskStatus.Success;

    }

    } trả về TaskStatus.Failure;
}
```

Mỗi khi nhiệm vụ được cập nhật, nó sẽ kiểm tra xem có mục tiêu nào trong tầm ngắm hay không. Nếu một mục tiêu nằm trong tầm nhìn, nó sẽ đặt giá trị mục tiêu và trả lại thành công. Đặt giá trị mục tiêu này là khóa vì điều này cho phép nhiệm vụ Hướng tới biết phải di chuyển theo hướng nào. Nếu không có mục tiêu nào trong tầm nhìn thì nhiệm vụ sẽ trả về thất bại. Phần cuối cùng của nhiệm vụ này là phương thức WithinSight: | 19

```

public bool WithinSight (Transform targetTransform, float fieldOfViewAngle)
{
    Vector3 hư ống = targetTransform.position -
biến đổi.position; trả
    về Vector3.Angle (hư ống, biến đổi. chuyển đổi) <
fieldOfViewAngle; }

```

Phương pháp này đầu tiên nhận được một vectơ hư ống giữa biến đổi hiện tại và biến đổi mục tiêu. Sau đó, nó sẽ tính toán góc giữa vectơ hư ống và vectơ chuyển tiếp hiện tại để xác định góc. Nếu góc đó nhỏ hơn góc nhìn của trường thì biến đổi mục tiêu nằm trong tầm nhìn của Biến đổi hiện tại.

Đó là nó cho nhiệm vụ Trong tầm nhìn. Đây là những gì nhiệm vụ đầy đủ trông như thế nào:

```

sử dụng UnityEngine;
sử dụng BehaviorDesigner.Runtime; sử
dụng BehaviorDesigner.Runtime.Tasks;

```

lớp công khai WithinSight: Có điều kiện {

```

// Độ rộng của một góc mà đối tượng có thể nhìn thấy
public float fieldOfViewAngle;
// Thủ của mục tiêu public string
targetTag;
// Đặt biến mục tiêu khi mục tiêu đã được tìm thấy để
các nhiệm vụ tiếp theo biết đối tượng nào là mục tiêu
mục tiêu SharedTransform công khai;

// Bộ nhớ đệm của tất cả các mục tiêu có thể private
Transform [] couldTargets;

```

ghi đè công khai void OnAwake () {

```

// Lưu vào bộ nhớ đệm tất cả các biến đổi có thẻ targetTag var target =
GameObject.FindGameObjectsWithTag (targetTag); có thẻTargets = new Transform
[target.Length]; for (int i = 0; i <target.Length; ++ i) {

    có thẻTargets [i] = target [i] .transform;
}
}
```

ghi đè công khai TaskStatus OnUpdate () {

```

// Trả về thành công nếu mục tiêu nằm trong tầm nhìn đối
với (int i = 0; i <couldTargets.Length; ++ i) {
    if (WithinSight (couldTargets [i], fieldOfViewAngle)) {
        // Đặt mục tiêu để các tác vụ khác biết được biến đổi nào | 20
    }
}
}
```

```

nằm trong tầm ngắm
    target.Value = couldTargets [i]; trả về
    TaskStatus.Success;
}

} trả về TaskStatus.Failure;
}

// Trả về true nếu targetTransform nằm trong tầm nhìn của biến đổi hiện tại

public bool WithinSight (Transform targetTransform, float fieldOfViewAngle)
{
    Vector3 hư ờng = targetTransform.position -
biến đổi.position;
    // Một đối tượng nằm trong tầm nhìn nếu góc nhỏ hơn trự ờng của
Quang cảnh
    trả về Vector3.Angle (hư ờng, biến đổi. chuyển đổi) <
fieldOfViewAngle; }

}

```

Tiếp tục đến phần thứ hai của chủ đề này, [viết nhiệm vụ Move Towards.](#)

## Viết một Nhiệm vụ Hành động Mới

Chủ đề này là sự tiếp nối của chủ đề trự ớc. Trự ớc tiên, bạn nên xem qua cách [viết một nhiệm vụ có điều kiện mới chủ đề đầu tiên.](#)

Nhiệm vụ tiếp theo mà chúng ta sẽ viết là nhiệm vụ Move Towards. Vì nhiệm vụ này sẽ thay đổi trạng thái trò chơi (di chuyển một đối tượng từ vị trí này sang vị trí khác), chúng tôi sẽ lấy nhiệm vụ từ lớp Hành động:

```

sử dụng UnityEngine; sử
dụng BehaviorDesigner.Runtime.Tasks;

lớp công khai MoveTowards: Hành động {}

```

Lớp này sẽ chỉ cần hai biến: cách đặt tốc độ và chuyển đổi của đối tượng mà chúng ta đang nhắm mục tiêu:

```

sử dụng UnityEngine; sử
dụng BehaviorDesigner.Runtime; sử dụng
BehaviorDesigner.Runtime.Tasks;

```

```
lớp công khai MoveTowards: Hành động {
```

```
    public float speed = 0; mục
    tiêu SharedTransform công khai;
}
```

Biến mục tiêu là một SharedTransform và nó sẽ được đặt từ tác vụ Within Sight sẽ chạy ngay trước tác vụ Move Towards. Để thực hiện chuyển động thực tế, chúng ta sẽ cần ghi đè phu ứng thức OnUpdate:

```
ghi đè công khai TaskStatus OnUpdate () {

    if (Vector3.SqrMagosystem (biến đổi.position -
target.Value.position) <0.1f) {return
    TaskStatus.Success;

    } biến đổi.position = Vector3.MoveTowards (biến đổi vị trí, mục
tiêu.Value.position, tốc độ * Time.deltaTime); trả về TaskStatus.Running;

}
```

Khi phu ứng thức OnUpdate được chạy, nó sẽ kiểm tra xem đối tượng đã đạt được mục tiêu hay chưa. Nếu đối tượng đã đạt được mục tiêu thì nhiệm vụ sẽ thành công. Nếu chưa đạt được mục tiêu, đối tượng sẽ di chuyển về phía mục tiêu với tốc độ được xác định bởi biến tốc độ. Vì đối tượng chưa đạt được mục tiêu nên tác vụ sẽ chạy trở lại.

Đó là toàn bộ nhiệm vụ của Move Towards. Toàn bộ nhiệm vụ trông giống như :

```
sử dụng UnityEngine; sử
dụng BehaviorDesigner.Runtime; sử dụng
BehaviorDesigner.Runtime.Tasks;
```

```
lớp công khai MoveTowards: Hành động {
```

```
// Tốc độ của đối tượng public
float speed = 0;
// Biến đổi mà đối tượng đang di chuyển tới mục tiêu SharedTransform
công khai;
```

```
ghi đè công khai TaskStatus OnUpdate () {
```

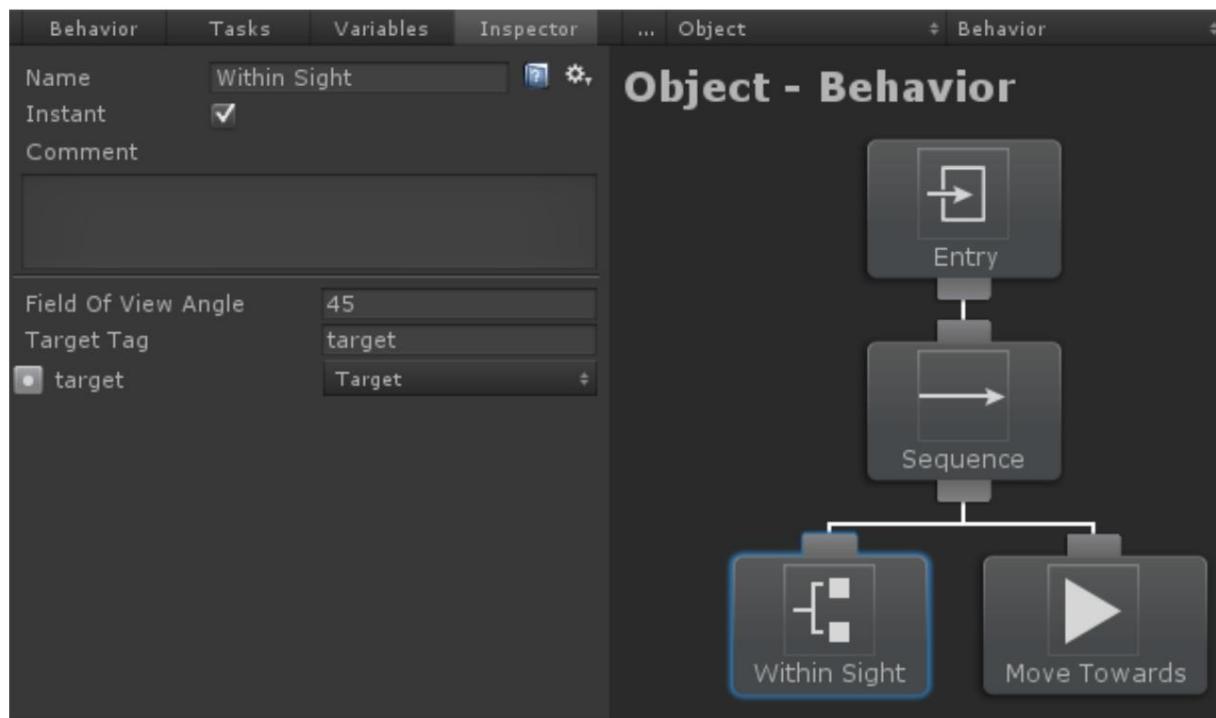
```
// Trả về trạng thái thành công của nhiệm vụ sau khi chúng ta đã đạt được mục tiêu
if (Vector3.SqrMagosystem (converter.position -
target.Value.position) <0.1f) {return
    TaskStatus.Success;
}
// Chúng ta vẫn chưa đạt được mục tiêu, vì vậy hãy tiếp tục tiến về phía nó biến
đổi.position = Vector3.MoveTowards (biến đổi vị trí, mục tiêu.Value.position, tốc
độ * Time.deltaTime); trả về TaskStatus.Running;
```

```

    }
}

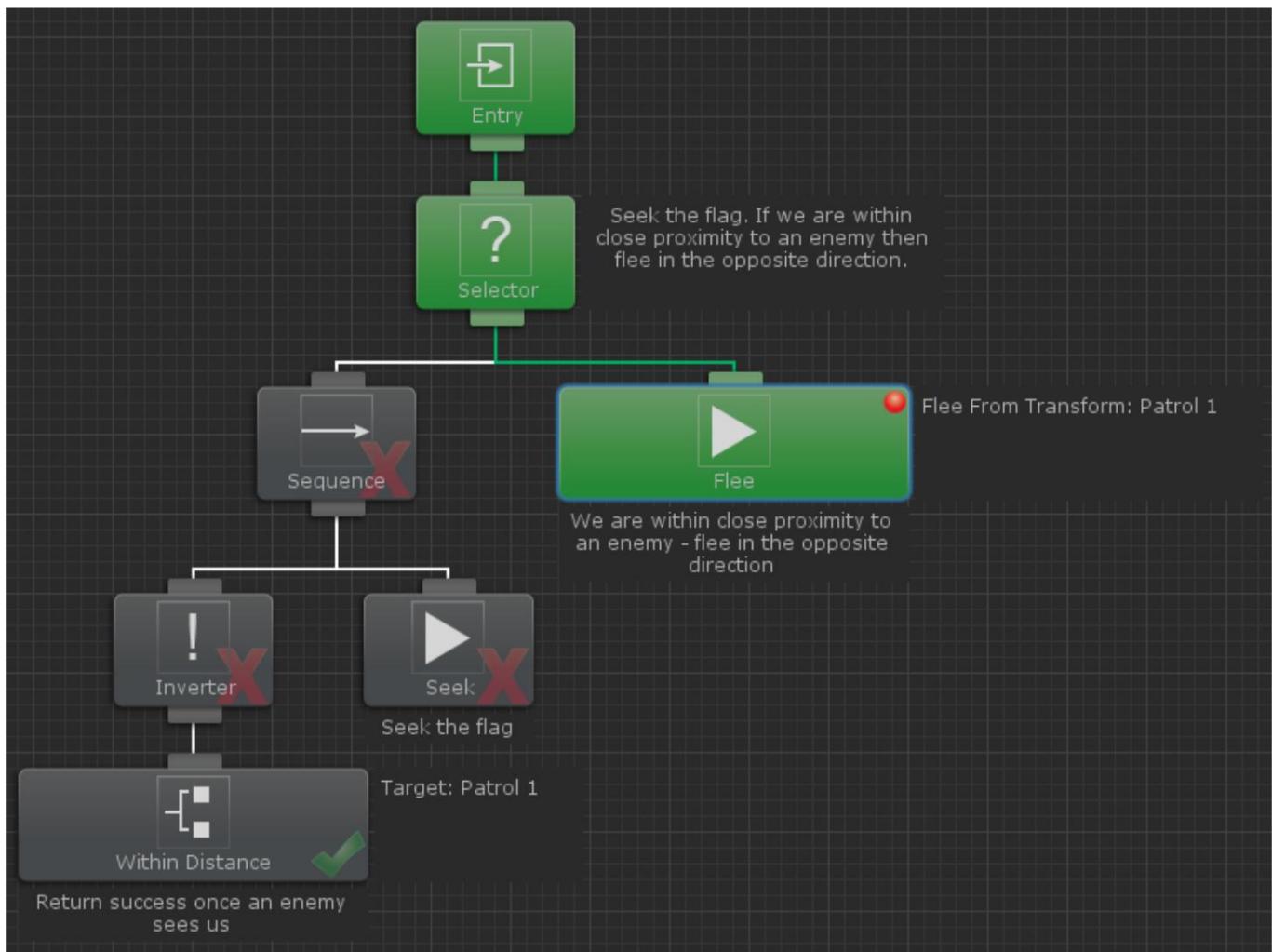
```

Bây giờ hai tác vụ này đã được viết xong, cấp cha các tác vụ bằng một tác vụ tuần tự và đặt các biến trong trình kiểm tra tác vụ. Đảm bảo rằng bạn cũng đã tạo một biến mới trong Behavior Designer:

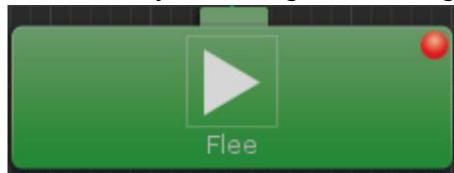


Đó là nó! Tạo một vài GameObject chuyển động trong cảnh để định với cùng một thẻ như targetTag. Khi trò chơi bắt đầu, đối tượng có gắn thẻ hành vi và di chuyển về phía bất kỳ đối tượng nào xuất hiện đầu tiên trong trường nhìn của nó. Đây là một ví dụ khá cơ bản và các nhiệm vụ có thể phức tạp hơn rất nhiều tùy thuộc vào những gì bạn muốn chúng làm. Tất cả các nhiệm vụ trong các dự án mẫu đều được nhận xét tốt vì vậy bạn có thể tiếp thu nó từ đó. Ngoài ra, chúng tôi đã viết thêm một số tài liệu về các chủ đề tiếp tục như [biến](#), [tham khảo nhiệm vụ](#) và [các thuộc tính nhiệm vụ](#).

## Gõ lối



Khi cây hành vi đang chạy, bạn sẽ thấy các tác vụ khác nhau thay đổi màu sắc giữa màu xám và xanh lá cây. Khi nhiệm vụ có màu xanh lục có nghĩa là nó hiện đang được thực thi. Khi nhiệm vụ có màu xám, nó không thực thi. Sau khi tác vụ được thực hiện, nó sẽ có dấu kiểm hoặc dấu x ở góc dưới cùng bên phải. Nếu nhiệm vụ trả về thành công thì một dấu kiểm sẽ được hiển thị. Nếu nó trả về lỗi thì một dấu x sẽ được hiển thị. Trong khi các tác vụ đang thực thi, bạn vẫn có thể thay đổi các giá trị trong

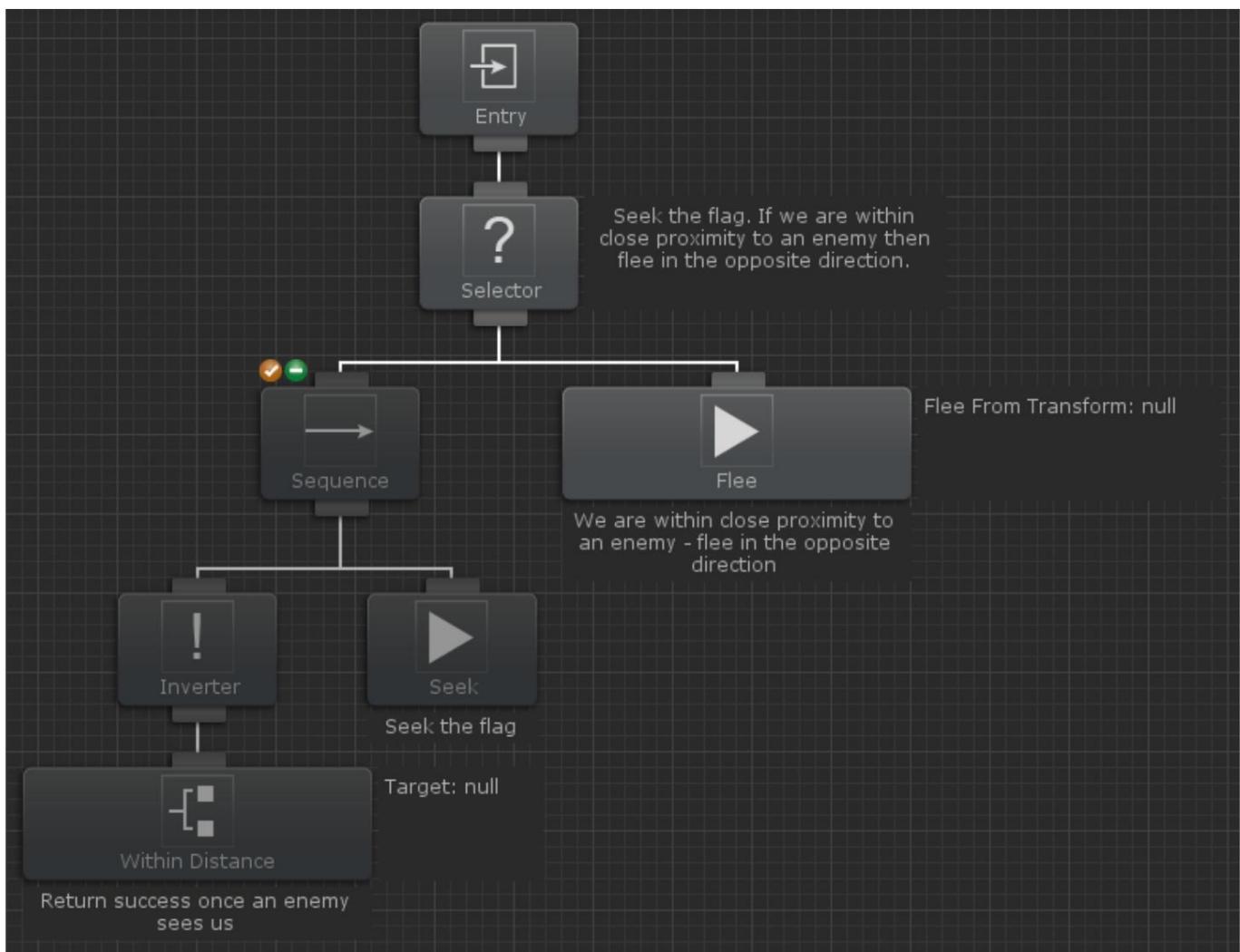


thanh tra và thay đổi đó sẽ được phản ánh trong trò chơi.

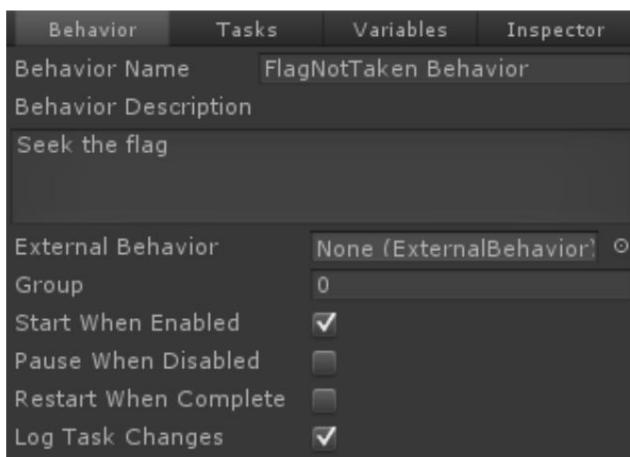
Nhấp chuột phải vào một nhiệm vụ sẽ xuất hiện menu cho phép bạn thiết lập điểm ngắt. Nếu một điểm ngắt được đặt trên một tác vụ cụ thể thì Behavior Designer sẽ tạm dừng Unity bắt cứ khi nào tác vụ đó được kích hoạt. Điều này rất hữu ích nếu bạn muốn xem khi nào một tác vụ cụ thể được thực thi.



Khi một nhiệm vụ được chọn, bạn có tùy chọn xem một biến trong biểu đồ bằng cách nhấp vào kính lúp ở bên trái của tên biến. Các biến đã xem là một cách tốt để xem giá trị của một biến cụ thể mà không cần phải mở trình kiểm tra tác vụ. Trong ví dụ trên, các biến "Khoảng cách chạy trốn" và "Chạy trốn khỏi sự biến đổi" đang được theo dõi và xuất hiện ở bên phải của nhiệm vụ Chạy trốn.



Đôi khi bạn chỉ muốn tập trung vào một nhóm nhiệm vụ nhất định và ngăn không cho phần còn lại chạy. Điều này có thể thực hiện được bằng cách tắt một tập hợp các nhiệm vụ. Có thể tắt tác vụ bằng cách di chuột qua tác vụ và chọn dấu X màu cam ở trên cùng bên trái của nhiệm vụ. Các tác vụ bị vô hiệu hóa sẽ không chạy và trả về thành công ngay lập tức. Các tác vụ đã tắt xuất hiện với màu đậm hơn các tác vụ đã bật trong biểu đồ.



Một tùy chọn gõ lỗi nữa là xuất ra bảng điều khiển bất kỳ lúc nào tác vụ thay đổi trạng thái. Nếu thay đổi tác vụ nhật ký được bật thì bạn sẽ thấy đâu ra cho nhật ký tương tự như sau:

GameObject - Behavior: Đẩy chuỗi nhiệm vụ (chỉ số 0) ở chỉ số ngăn xếp 0

GameObject - Behavior: Đẩy nhiệm vụ Chờ (chỉ số 1) ở chỉ số ngăn xếp 0

GameObject - Behavior: Tác vụ bật Chờ (chỉ số 1) ở ngăn xếp chỉ số 0 với trạng thái Thành công

GameObject - Behavior: Đẩy nhiệm vụ Chờ (chỉ số 2) ở chỉ số ngăn xếp 0

GameObject - Behavior: Tác vụ Pop Chờ (chỉ số 2) ở ngăn xếp chỉ số 0 với trạng thái Thành công

GameObject - Behavior: Pop task Sequence (index 0) at stack index 0 with status Success

### Tắt GameObject - Hành vi

Các thông báo này có thể được chia thành các phần sau:

{tên đối tượng trò chơi} - {tên hành vi}: {thay đổi nhiệm vụ} {loại tác vụ} (chỉ mục {chỉ mục tác vụ}) tại chỉ mục ngăn xếp {chỉ mục ngăn xếp} {trạng thái tùy chọn}

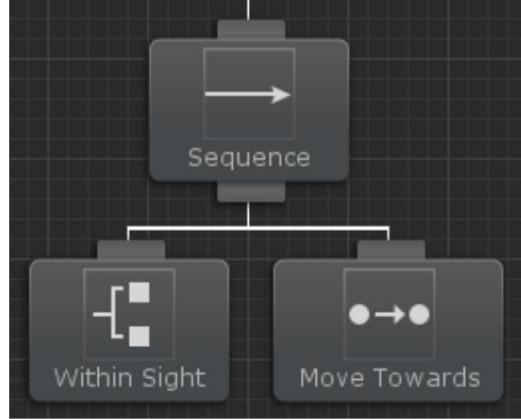
{tên đối tượng trò chơi} là tên của đối tượng trò chơi mà cây hành vi được gắn vào. {tên hành vi} là tên của cây hành vi. {thay đổi nhiệm vụ} cho biết trạng thái mới của nhiệm vụ. Ví dụ: một tác vụ sẽ được đẩy lên ngăn xếp khi nó bắt đầu thực thi và nó sẽ được bật lên khi nó được thực thi xong. {task type} là loại lớp của nhiệm vụ. {chỉ mục nhiệm vụ} là chỉ mục của nhiệm vụ trong tìm kiếm chuyên sâu đầu tiên. {chỉ số ngăn xếp} là chỉ số của ngăn xếp mà tác vụ đang được đẩy đến. Nếu bạn có một nút song song thì bạn sẽ sử dụng nhiều ngăn xếp. {tùy chọn trạng thái} là bất kỳ trạng thái bổ sung nào cho thay đổi cụ thể đó. Tác vụ pop sẽ xuất trạng thái tác vụ.

## Biến

Một trong những ưu điểm của cây hành vi là chúng rất linh hoạt ở chỗ tất cả các tác vụ đều được kết hợp với nhau một cách lỏng lẻo - có nghĩa là một tác vụ không phụ thuộc vào một tác vụ khác để vận hành. Hạn chế của việc này là đôi khi bạn cần các nhiệm vụ để chia sẻ thông tin với nhau. Ví dụ: bạn có thể có một nhiệm vụ là xác định xem mục tiêu có nằm trong Tầm nhìn hay không. Nếu mục tiêu là | 26

trong tầm nhìn, bạn có thể có một nhiệm vụ khác Di chuyển tới mục tiêu. Trong trường hợp này, hai nhiệm vụ cần giao tiếp với nhau để nhiệm vụ Hướng tới thực sự di chuyển theo hướng của cùng một đối tượng mà nhiệm vụ Trong tầm nhìn đã tìm thấy. Trong triển khai cây hành vi truyền thống, điều này được giải quyết bằng cách mã hóa bằng đen. Với Behavior Designer, bạn có thể sử dụng các biến để dàng hơn rất nhiều.

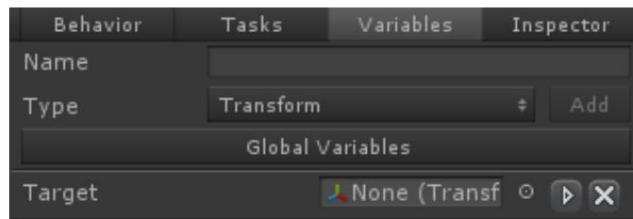
Trong ví dụ trước của chúng tôi, chúng tôi có hai nhiệm vụ: một nhiệm vụ xác định xem mục tiêu có nằm trong tầm nhìn hay không và sau đó nhiệm vụ kia di chuyển đến mục tiêu. Cây này trông giống như :



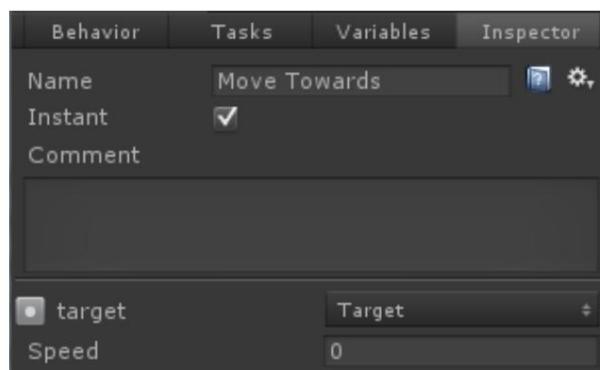
Mã cho cả hai nhiệm vụ này được thảo luận trong phần [Viết một Nhiệm vụ Mới](#) chủ đề, nhưng phần xử lý các biến nằm trong khai báo biến này:

mục tiêu SharedTransform công khai;

Với biến SharedTransform được tạo, bây giờ chúng ta có thể tạo một biến mới trong Behavior Designer và gán biến đó cho hai nhiệm vụ:

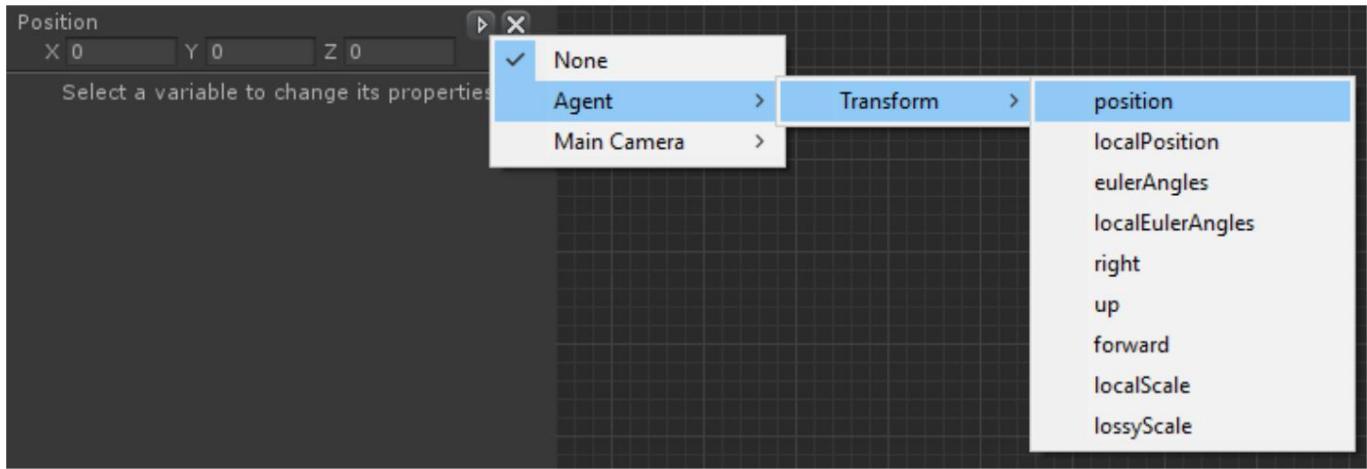


Chuyển sang trình kiểm tra tác vụ và gán biến đó cho hai tác vụ:



Và cùng với đó, hai nhiệm vụ có thể bắt đầu chia sẻ thông tin! Bạn có thể lấy / đặt giá trị của biến được chia sẻ bằng cách truy cập thuộc tính Giá trị. Ví dụ, target.Value sẽ trả về đối tượng biến đổi. Khi Within Sight chạy, nó sẽ gán biến đổi của đối tượng xuất hiện trong tầm mắt cho biến Target. Khi Move Towards chạy, nó sẽ sử dụng biến Target đó để xác định vị trí cần di chuyển.

Nhìn vào biến trong trình kiểm tra, nếu bạn nhìn sang bên trái của nút xóa, bạn sẽ thấy một hình tam giác trỏ sang phải. Đây là nút cho Ánh xạ biến.



Ánh xạ biến cho phép SharedVariable của bạn ánh xạ tới một thuộc tính cùng loại. Điều này cho phép bạn nhanh chóng lấy hoặc đặt một giá trị trên một thành phần MonoBehaviour. Ví dụ, giả sử rằng chúng tôi muốn có được vị trí của người đại diện. Có thể sử dụng tác vụ Nhận vị trí với một biến không được ánh xạ, nhưng điều đó sẽ thêm các tác vụ không cần thiết vào cây hành vi của bạn. Thay vào đó, nếu bạn ánh xạ biến tới thuộc tính Transform.position, thì bắt cứ khi nào giá trị của biến được truy cập, thay vào đó, nó sẽ sử dụng thuộc tính mà nó được ánh xạ tới. Điều này cho phép bạn lấy hoặc đặt vị trí của Biến đổi mà không cần thực hiện thêm bất kỳ tác vụ nào.

Lưu ý: Ánh xạ biến yêu cầu thuộc tính C#. Nó không hoạt động trên các trường C#. Với ví dụ Vector3 ở trên, bạn có thể thiết lập ánh xạ thuộc tính như sau:

```
lớp công khai MyMonoBehaviour: MonoBehaviour {
    riêng Vector3 m_MyVector3; public
    Vector3 MyVector3 {get {return m_MyVector3; } bộ {
        m_MyVector3 = giá trị; }} // Bắt buộc đối với ánh xạ thuộc tính. }
```

Behavior Designer hỗ trợ cả biến cục bộ và toàn cục. Biến toàn cục tự như các biến cục bộ ngoại trừ bất kỳ cây nào có thể tham chiếu đến cùng một biến. Các biến có thể được tham chiếu bởi các lớp dẫn xuất không phải Tác vụ bằng cách nhận tham chiếu từ cây hành vi.

Các loại biến được chia sẻ sau đây được bao gồm trong cài đặt Behavior Designer mặc định.

Nếu không có loại nào trong số này phù hợp với tình huống của bạn thì bạn có thể tạo biến chia sẻ của riêng mình:

- SharedAnimationCurve
- SharedBool
- SharedColor
- SharedFloat
- SharedGameObject
- SharedGameObjectList
- SharedInt
- SharedMaterial
- SharedObject

- SharedObjectList
- SharedQuaternion
- SharedRect
- Chuỗi chia sẻ
- SharedTransform
- SharedTransformList
- SharedVector2
- SharedVector3Int
- SharedVector3
- SharedVector3Int
- SharedVector4

## Biến động

Các biến động cho phép bạn sử dụng các biến tạm thời bị giới hạn trong phạm vi. Các biến này rất tốt nếu bạn muốn chia sẻ dữ liệu giữa một số tác vụ hạn chế và không cần truy cập biến bên ngoài các tác vụ đó. Các biến động có thể được tạo bằng cách nhấp vào vòng tròn ở bên phải của trường biến, sau đó chọn "(Động)":



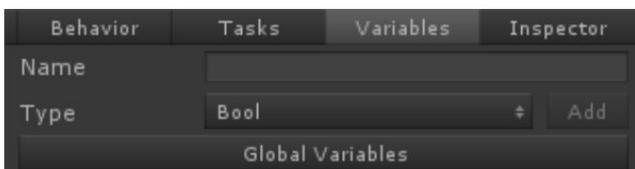
Sau khi biến động đã được tạo, bạn có thể nhập tên của biến.



Biến động bây giờ sẽ được sử dụng trong cây của bạn. Biến động sẽ có cùng giá trị cho bất kỳ trường nào tham chiếu đến cùng một tên biến động. Tên phân biệt chữ hoa chữ thường và không giống với một biến cục bộ trong cây.

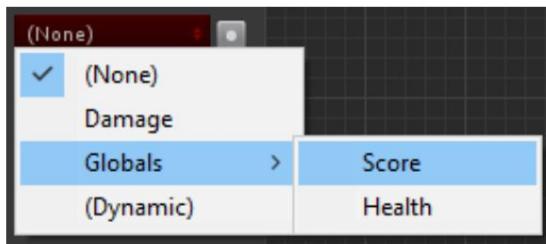
## Biến toàn cục

Các biến toàn cục tương tự như các biến cục bộ ngoại trừ bất kỳ cây hành vi nào cũng có thể truy cập một thể hiện của cùng một biến. Để truy cập các biến toàn cục, hãy điều hướng đến tùy chọn menu Window-> Behavior Designer-> Global Variables hoặc từ trong ngăn Biến:



Khi một biến toàn cục được thêm lần đầu tiên, một tệp nội dung sẽ được tạo để lưu trữ tất cả các biến toàn cục. Tệp này được tạo tại / Behavior Designer / Resources / BehaviorDesignerGlobalVariables.asset. Bạn có thể di chuyển tệp này miễn là nó vẫn nằm trong thư mục Tài nguyên.

Các biến toàn cục được gán theo cách rất giống với các biến cục bộ. Trong trình kiểm tra tác vụ, khi bạn chỉ định một biến toàn cục, các biến toàn cục được đặt trong mục trình đơn "Globals":



Các biến toàn cục cũng có thể được [truy cập từ các đối tượng dẫn xuất không phải Tác vụ](#).

## Tạo các biến được chia sẻ

Có thể tạo Biến chia sẻ mới nếu bạn không muốn sử dụng bất kỳ loại nào được tích hợp sẵn. Để tạo một Biến được chia sẻ, hãy phân lớp loại SharedVariable và triển khai các phương pháp sau. Từ khóa OBJECT\_TYPE phải được thay thế bằng loại Biến được chia sẻ mà bạn muốn tạo.

```
[System.Serializable] lớp
công khai SharedOBJECT_TYPE: SharedVariable <OBJECT_TYPE> {

    toán tử ngầm định công khai tĩnh SharedOBJECT_TYPE (OBJECT_TYPE
    value) {return new SharedOBJECT_TYPE {Value = value}; }
}
```

Điều quan trọng là thuộc tính "Giá trị" tồn tại. Trình kiểm tra biến sẽ hiển thị lỗi nếu Biến được chia sẻ mới được tạo không chính xác. Các biến được chia sẻ có thể chứa bất kỳ loại đối tượng nào mà tác vụ của bạn có thể chứa, bao gồm nguyên thủy, mảng, danh sách, đối tượng tùy chỉnh, v.v.

Ví dụ: tập lệnh sau sẽ cho phép một lớp tùy chỉnh được chia sẻ:

```
[System.Serializable] lớp
công khai CustomClass {

    public int myInt;
    public Object myObject;
}

[System.Serializable]
public class SharedCustomClass: SharedVariable <CustomClass> {

    toán tử ngầm công khai tĩnh SharedCustomClass (CustomClass
    value) {return new SharedCustomClass {Value = value}; }
}
```

## Truy cập các biến từ các đối tượng không phải nhiệm vụ

Các biến thường được tham chiếu bằng cách [gán tên](#) biến cho truy cập của nó trong bảng kiểm tra Behavior Designer. Các biến cục bộ cũng có thể được truy cập bởi các lớp dẫn xuất không phải Tác vụ (chẳng hạn như MonoBehaviour) bằng cách gọi phuơng thức:

```
behaviorTree.GetVariable ("MyVariable");
behaviorTree.SetVariable ("MyVariable", giá trị);
behaviorTree.SetVariableValue ("Tên của tôi", giá trị);
```

Khi đặt một biến, nếu bạn muốn các tác vụ tự động tham chiếu đến biến đó thì hãy đảm bảo rằng một biến được tạo với tên đó trước thời hạn. Đoạn mã sau đây cho thấy một ví dụ về việc sửa đổi một biến từ lớp MonoBehaviour:

```
sử dụng UnityEngine; sử
dụng BehaviorDesigner.Runtime;

lớp công cộng AccessVariable: MonoBehaviour {

    công khai BehaviorTree behaviorTree;

    public void Start () {

        var myIntVariable =
(SharedInt) behaviorTree.GetVariable ("MyVariable");
        myIntVariable.Value = 42;
    }
}
```

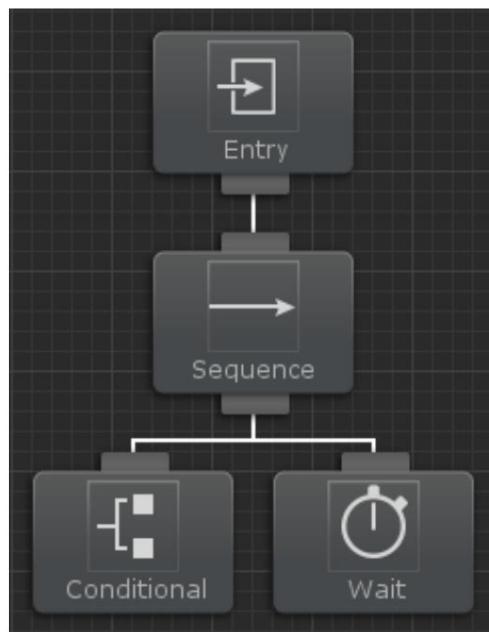
Trong ví dụ trên, chúng ta đang nhận được một tham chiếu đến biến có tên "MyVariable" trong ngăn Biến của nhà thiết kế hành vi. Ngoài ra, như thể hiện trong ví dụ, bạn có thể lấy và đặt giá trị của biến bằng thuộc tính SharedVariable.Value.

Tự ứng tự, các biến toàn cục có thể được truy cập bằng cách nhận tham chiếu đến cá thể GlobalVariable:

```
GlobalVariables.Instance.GetVariable ("MyVariable");
GlobalVariables.Instance.SetVariable ("MyVariable", giá trị);
```

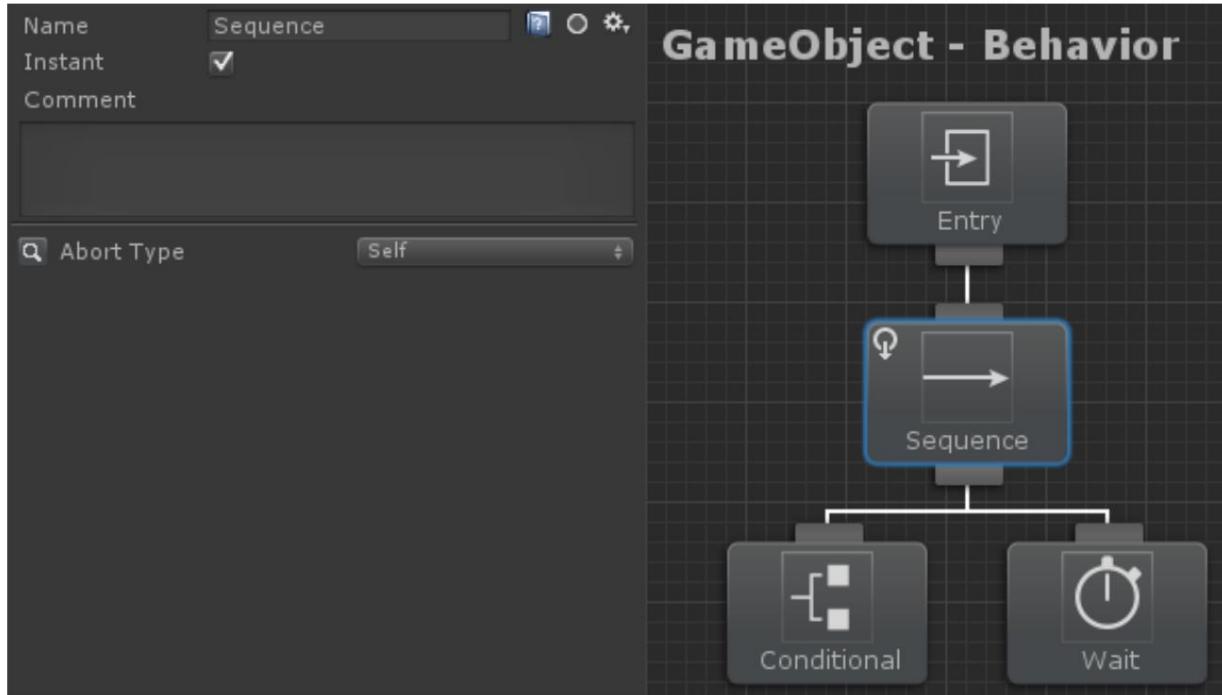
## Hủy bỏ có điều kiện

Hủy bỏ có điều kiện cho phép cây hành vi của bạn phản hồi động với các thay đổi mà không phải làm lộn xộn cây hành vi của bạn với nhiều tác vụ Ngắt / Thực hiện Ngắt. Tính năng này tương tự như Observer Aborts trong Unreal Engine 4. Hầu hết các triển khai cây hành vi đều đánh giá lại toàn bộ cây sau mỗi lần đánh dấu. Hủy bỏ có điều kiện là một tối ưu hóa để tránh phải chạy lại toàn bộ cây. Như một ví dụ cơ bản, hãy xem xét

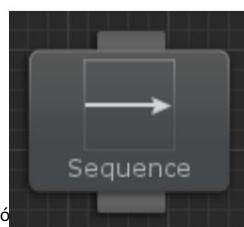


cây sau:

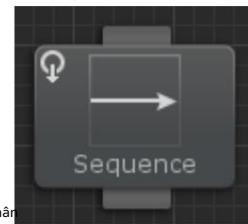
Khi cây này chạy tác vụ Điều kiện sẽ trả về thành công và tác vụ Trình tự sẽ bắt đầu chạy con tiếp theo là tác vụ Chờ. Tác vụ Chờ có thời gian chờ là 10 giây. Trong khi tác vụ chờ đang chạy, giả sử rằng tác vụ có điều kiện thay đổi trạng thái thay đổi và bây giờ trả về lỗi. Nếu Hủy bỏ có điều kiện được bật, tác vụ Điều kiện sẽ đưa ra lệnh hủy bỏ và dừng tác vụ Chờ chạy. Tác vụ Điều kiện sẽ được đánh giá lại và tác vụ tiếp theo sẽ chạy theo các quy tắc cây hành vi tiêu chuẩn. Hủy bỏ có điều kiện có thể được truy cập từ bất kỳ tác vụ Tổng hợp nào:



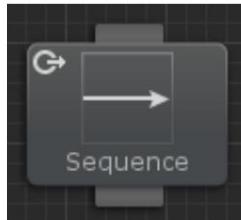
Có bốn kiểu hủy bỏ khác nhau: Không có, Bản thân, Mức độ ưu tiên thấp hơn và Cả hai.



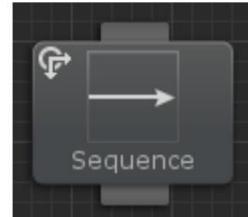
Đây là hành vi mặc định. Nhiệm vụ Điều kiện sẽ không đư ợc đánh giá lại và không có quyết định hủy bỏ nào đư ợc đư a ra.



Đây là một kiểu hủy bỏ khép kín. Nhiệm vụ có điều kiện chỉ có thể hủy bỏ một tác vụ Hành động nếu cả hai đều có cùng một tác vụ Tổng hợp mẹ.



Hành vi có mức độ ưu tiên thấp hơn có thể đư ợc sắp xếp từ các nhiệm vụ quan trọng hơn đến ít quan trọng nhất. Nếu một nhiệm vụ có điều kiện quan trọng hơn thay đổi trạng thái thì có thể đưa ra lệnh hủy bỏ sẽ dừng các tác vụ có mức độ ưu tiên thấp hơn đang chạy.



Loại hủy bỏ này kết hợp cả mức độ ưu tiên của bản thân và mức độ ưu tiên thấp hơn.

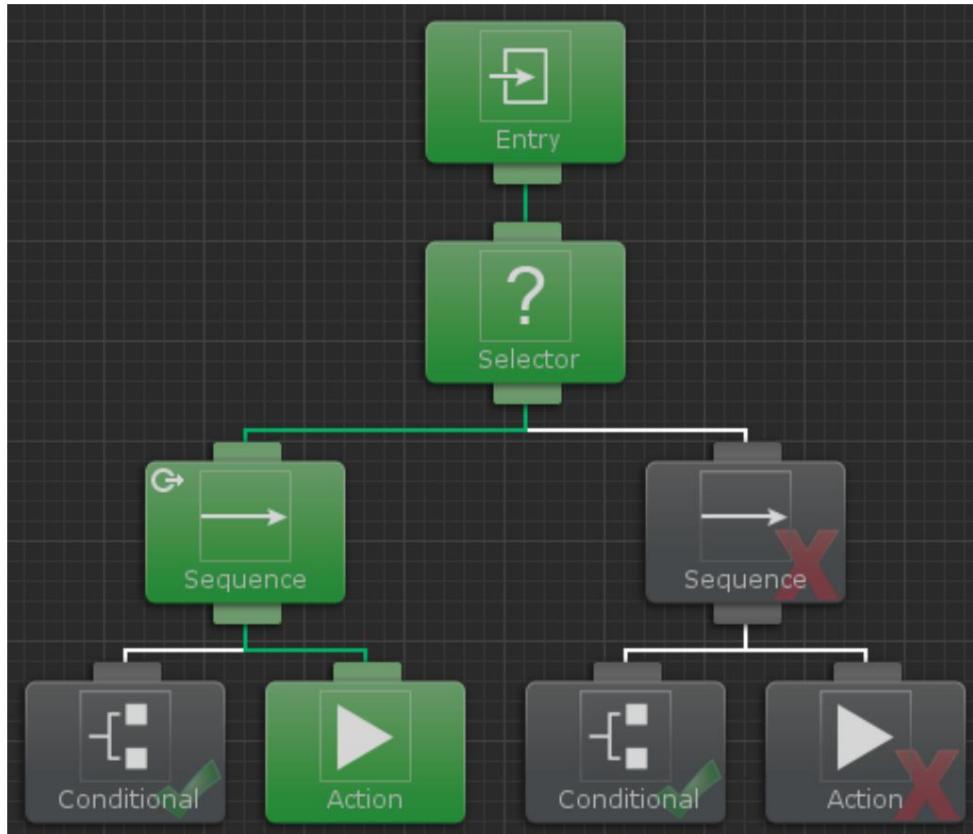
Phá thai có điều kiện cũng có thể đư ợc suy nghĩ theo cách sau:

Mức độ ưu tiên thấp hơn: sẽ đánh giá lại khi bất kỳ nhiệm vụ nào ở bên phải của nhánh hiện tại đang hoạt động.

Bản thân: sẽ đánh giá lại khi có bất kỳ nhiệm vụ nào trong nhánh hiện tại đang hoạt động.

Cả hai: sẽ đánh giá lại khi bất kỳ nhiệm vụ nào ở bên phải hoặc bên trong nhánh hiện tại đang hoạt động.

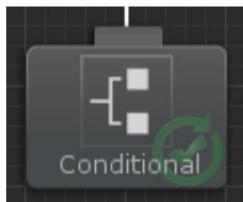
Ví dụ sau sẽ sử dụng kiểu hủy bỏ ưu tiên thấp hơn:



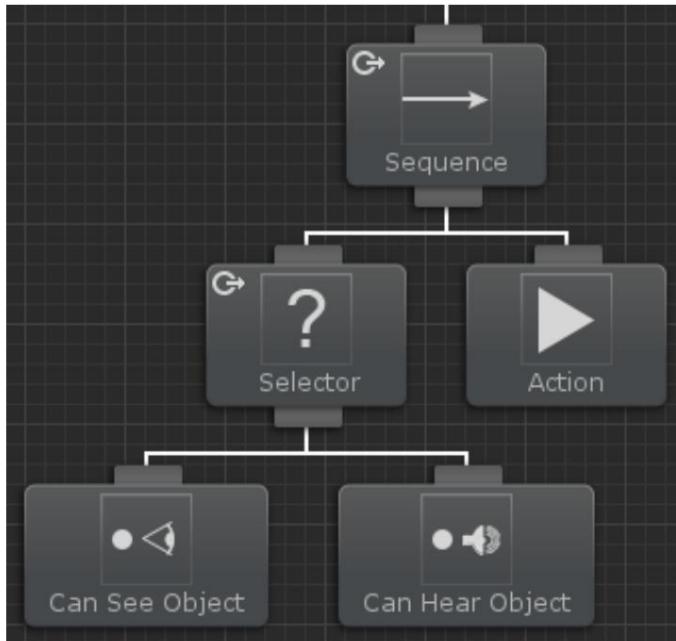
Trong ví dụ này, tác vụ Trình tự cha của nhánh bên trái có kiểu hủy bỏ là thấp hơn

sự ưu tiên. Giả sử rằng nhánh bên trái bị lỗi và di chuyển cây sang nhánh bên phải do nhiệm vụ cha của Bộ chọn. Trong khi nhánh bên phải đang chạy, tác vụ có Điều kiện đầu tiên sẽ chuyển trạng thái thành thành công. Vì trạng thái tác vụ đã thay đổi và kiểu hủy bỏ có mức độ ưu tiên thấp hơn nên tác vụ Hành động hiện đang chạy sẽ bị hủy bỏ và tác vụ có Điều kiện ban đầu được chạy lại.

Trạng thái thực thi của tác vụ có điều kiện sẽ có biểu tượng bộ lặp xung quanh trạng thái thành công hoặc thất bại để cho biết rằng nó đang được đánh giá lại bằng cách hủy bỏ có điều kiện:



Hủy bỏ có điều kiện cũng có thể được lồng vào nhau. Ví dụ: bạn có thể muốn điều hành một chi nhánh khi một trong hai điều kiện thành công, nhưng cả hai điều kiện đều không bắt buộc. Trong ví dụ này, chúng ta sẽ sử dụng các nhiệm vụ Có thể thấy Đối tư ứng và Có thể Nghe Đối tư ứng. Bạn muốn chạy tác vụ hành động khi đối tư ứng được nhìn thấy hoặc nghe thấy. Để thực hiện việc này, hai tác vụ có điều kiện này phải được tạo bởi một Bộ chọn có kiểu hủy bỏ ưu tiên thấp hơn. Nhiệm vụ hành động sau đó là một anh chị em của nhiệm vụ Bộ chọn. Một tác vụ Trình tự sau đó được gán cho hai tác vụ này bởi vì tác vụ hành động chỉ nên chạy khi một trong hai tác vụ có điều kiện thành công. Nhiệm vụ Trình tự được đặt thành loại hủy bỏ Mức độ ưu tiên thấp hơn nên hai tác vụ có điều kiện sẽ tiếp tục được đánh giá lại ngay cả khi cây đang chạy một nhánh hoàn toàn khác.

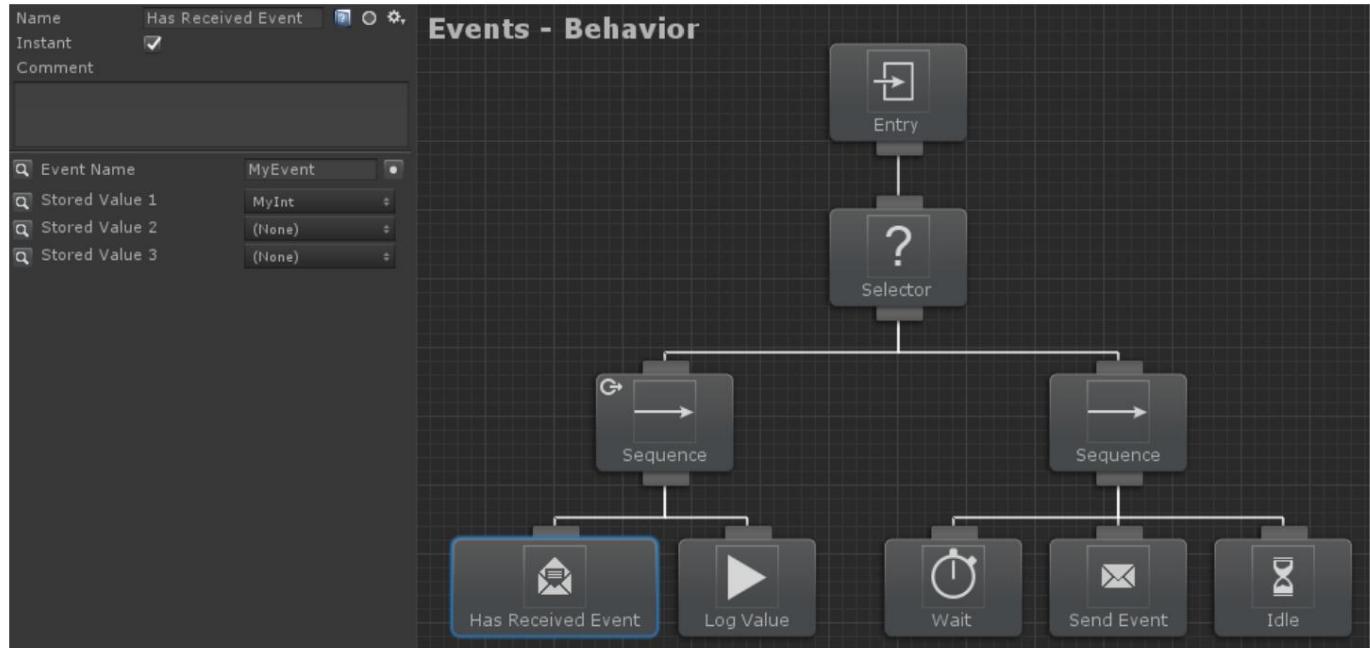


Điều quan trọng cần lưu ý với cây này là tác vụ Bộ chọn phải có kiểu hủy bỏ được đặt thành Mức độ ưu tiên thấp hơn (hoặc Cả hai). Nếu nó không có kiểu hủy bỏ được đặt thì hai tác vụ có điều kiện sẽ không được đánh giá lại.

## Sự kiện

Hệ thống sự kiện trong Behavior Designer cho phép cây hành vi của bạn dễ dàng phản ứng với các thay đổi. Hệ thống sự kiện này có thể kích hoạt một sự kiện thông qua mã hoặc thông qua các tác vụ cây hành vi.

Sự kiện có thể được báo hiệu thông qua cây hành vi với các nhiệm vụ Sự kiện gửi và Sự kiện đã nhận. Khi một sự kiện được báo hiệu, tác vụ Gửi sự kiện sẽ được sử dụng. Nhiệm vụ Sự kiện Đã Nhận là một nhiệm vụ có điều kiện và sẽ trả về thành công ngay sau khi sự kiện đã được nhận. Tên sự kiện có thể được chỉ định cho cả hai nhiệm vụ này. Để nhận đúng các sự kiện, nhiệm vụ Sự kiện Đã nhận phải được đánh giá lại với các lần hủy bỏ có điều kiện. Ví dụ, hãy xem xét cây sau:



Trong lần đánh dấu đầu tiên, nhiệm vụ Sự kiện Đã Nhận sẽ không thành công vì sự kiện chưa được nhận. Bộ chọn sau đó sẽ chuyển sang nhánh bên phải và nhiệm vụ Chờ sẽ bắt đầu. Sau một giây, Nhiệm vụ gửi sự kiện sẽ gửi sự kiện và sau đó Chế độ chờ được bắt đầu để giữ cho cây hoạt động. Trong lần đánh dấu tiếp theo, nhiệm vụ Sự kiện Đã nhận trả về trạng thái thành công do sự kiện trước đó đã được gửi. Sau đó, nó xuất ra giá trị của biến MyInt đã được gửi với tác vụ Gửi sự kiện.

Ngoài việc có thể gửi các sự kiện thông qua cây hành vi, các sự kiện có thể được gửi thông qua mã. Phư ơng thức BehaviorTree.SendEvent sẽ cho phép bạn gửi một sự kiện đến cây hành vi được chỉ định. Ví dụ:

```
var behaviorTree = GetComponent <BehaviorTree> ();
behaviorTree.SendEvent <object> ("MyEvent", 5);
```

Trong ví dụ này, sự kiện "MyEvent" sẽ được gửi đến thành phần cây hành vi với giá trị tham số là 5. Nếu cây hành vi chứa tác vụ Sự kiện Đã nhận thì nó sẽ phản ứng tương ứng. Nếu tác vụ Sự kiện Đã nhận đang nhận sự kiện thì loại mẫu phải là một đối tượng hợp lệ.

Bạn cũng có thể nhận các sự kiện từ bên ngoài cây hành vi. Để tiếp tục với ví dụ "MyEvent", bạn có thể nhận sự kiện này bằng cách sử dụng phư ơng thức BehaviorTree.RegisterEvent. BehaviorTree.UnregisterEvent sẽ ngừng lắng nghe sự kiện đó.

```
public void OnEnable () {

    var behaviorTree = GetComponent <BehaviorTree> ();
    behaviorTree.RegisterEvent <object> ("MyEvent", ReceivedEvent);
```

```

}

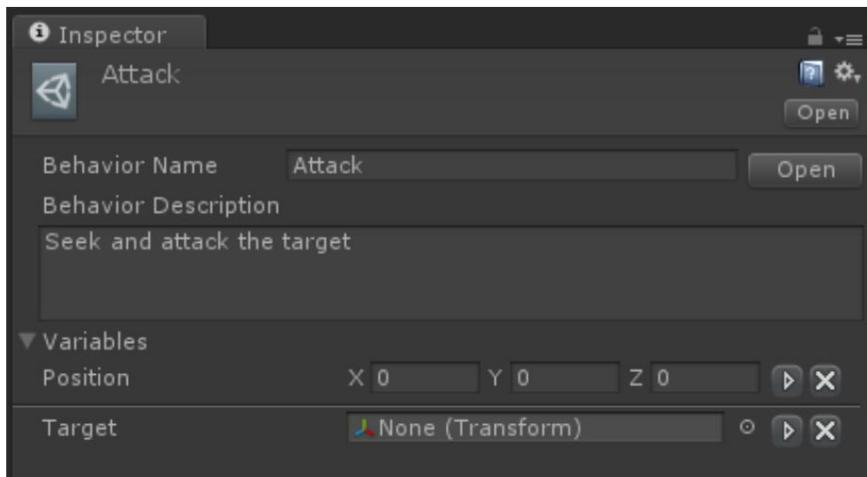
public void ReceivedEvent (object arg1) {

}

public void OnDisable ()
{
    var behaviorTree = GetComponent <BehaviorTree> ();
    behaviorTree.UnregisterEvent <object> ("MyEvent", ReceivedEvent);
}

```

## Cây Hành vi Bên ngoài



Trong một số trường hợp, bạn có thể có một cây hành vi mà bạn muốn chạy từ nhiều đối tượng. Ví dụ, bạn có thể có một cây hành vi tuần tra một căn phòng. Thay vì tạo cây hành vi riêng biệt cho từng đơn vị, bạn có thể sử dụng cây hành vi bên ngoài. Cây hành vi bên ngoài được tham chiếu bằng cách sử dụng tác vụ Tham chiếu cây hành vi. Khi cây hành vi ban đầu bắt đầu chạy, nó sẽ tải tất cả các tác vụ bên trong cây hành vi bên ngoài và hoạt động như thể chúng là của riêng nó. Bất kỳ SharedVariable nào trong cây bên ngoài có cùng tên và loại của cây mẹ của nó sẽ tự động bị ghi đè. Ví dụ: nếu cây mẹ có SharedInt có tên là "MyInt" với giá trị là 7 và Cây bên ngoài có SharedInt có tên là "MyInt" với giá trị là 0, khi cây chạy MyInt sẽ có giá trị là 7 trong Cây bên ngoài.

### Tham chiếu cây hành vi

Tác vụ Tham chiếu cây hành vi cho phép bạn chạy một cây hành vi khác trong cây hành vi hiện tại. Bạn có thể tạo cây hành vi này bằng cách lưu cây dưới dạng cây hành vi bên ngoài. Một cách sử dụng cho điều này là nếu bạn có một đơn vị đóng một loạt các nhiệm vụ để tấn công. Bạn có thể muốn đơn vị tấn công tại các điểm khác nhau trong cây hành vi và bạn muốn đơn tấn công đó luôn giống nhau. Thay vì sao chép và dán các tác vụ giống nhau lặp đi lặp lại, bạn có thể chỉ cần sử dụng một hành vi bên ngoài và sau đó các tác vụ luôn được đảm bảo giống nhau. Ví dụ này được minh họa trong dự án mẫu RTS trên [trang mẫu](#).

Phu ơng thức GetExternalBehaviors cho phép bạn ghi đè nó để bạn có thể cung cấp một mảng cây hành vi bên ngoài được xác định trong thời gian chạy.

Tác vụ Tham chiếu Cây Hành vi cho phép bạn chỉ định các biến cho mỗi tác vụ tham chiếu. Đôi với hầu hết các trường hợp, điều này là không cần thiết vì các biến sẽ tự động được chuyển từ cây cha sang cây bên ngoài. Tuy nhiên, nếu bạn có nhiều tác vụ Tham chiếu Cây Hành vi trong cùng một cây, tất cả đều trả đến cùng một Cây Bên ngoài, bạn có thể muốn chỉ định các giá trị biến khác nhau trên mỗi cây. Trong tình huống này, bạn có thể chỉ định giá trị biến trong tác vụ Tham chiếu cây hành vi và giá trị biến đó sẽ được chuyển sang Cây bên ngoài.

## Gộp chung

Các cây hành vi bên ngoài có thể được gộp chung để có hiệu suất tốt hơn khi chuyển đổi giữa nhiều cây bên ngoài. Khi cây hành vi bên ngoài được khởi tạo, phu ơng thức Init sẽ được gọi để giải mã hóa cây hành vi bên ngoài. Sau đó, cây hành vi bên ngoài được gộp chung có thể được gán cho thành phần cây hành vi giống như cây hành vi bên ngoài không được gộp chung. Ví dụ:

```
sử dụng UnityEngine; sử  
dụng BehaviorDesigner.Runtime;
```

```
public class ExternalPoolExample: MonoBehaviour {public BehaviorTree  
behaviorTree; public ExternalBehavior externalBehaviorTree;  
  
private ExternalBehavior [] externalPool; chỉ mục  
int riêng tư ;  
  
public void Awake () {  
  
    // Khởi tạo năm Cây hành vi bên ngoài sẽ được sử dụng lại. Phu ơng thức  
Init sẽ giải mã hóa Cây hành vi bên ngoài. externalPool = new ExternalBehavior [5]; for  
    (int i = 0; i <externalPool.Length; ++ i) {externalPool [i] = Object.Instantiate  
    (externalBehaviorTree);  
  
        externalPool [i] .Init ();  
    }  
}  
  
public void OnGUI () {  
  
    // Chỉ định Cây hành vi bên ngoài tiếp theo trong nhóm đơn giản. if  
(GUILayout.Button ("Chỉ định")) {  
  
        behaviorTree.DisableBehavior ();  
        behaviorTree.ExternalBehavior = externalPool [index];  
        behaviorTree.EnableBehavior ();
```

```

        index = (index + 1) % externalPool.Length;
    }
}
}

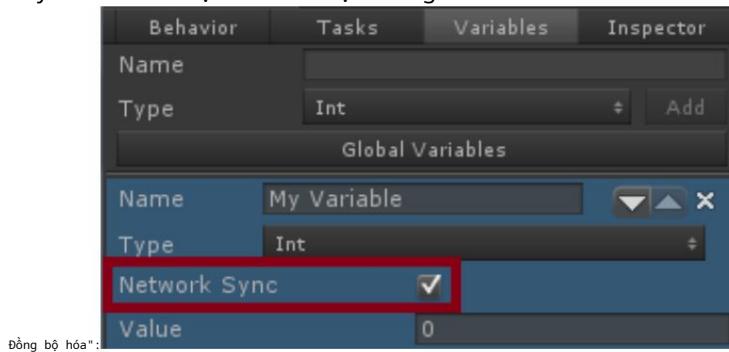
```

## Kết nối mạng

Behavior Designer hỗ trợ hệ thống mạng của Unity được giới thiệu với Unity 5.1.

Mạng là một chủ đề phức tạp, vì vậy trước tiên bạn nên xem qua [tài liệu về mạng](#) của Unity trước khi tiếp tục.

Các biến được chia sẻ có thể tự động được đồng bộ hóa qua mạng từ máy chủ đến máy khách. Điều này có thể được kích hoạt bằng cách mở chi tiết của biến và chọn "Mạng



[Định nghĩa trình biên dịch ENABLE\\_MULTIPLAYER](#) phải được thêm vào để các biến đồng bộ hóa chính xác.

Do giới hạn mạng hiện tại, các cuộc gọi ClientRPC không thể bị quá tải, vì vậy loại này phải được biết trước. Điều này có nghĩa là chỉ có thể đồng bộ hóa các loại biến sau:

- bool
- Màu
- nỗi
- GameObject
- int
- Quaternion
- 
- Chuỗi Rect
- Biến đổi
- Vector2
- Véc tơ3
- Vector4

Thành phần Hành vi từ [mã nguồn thời gian chạy](#) phải được sử dụng để cho phép các biến được đồng bộ hóa. Đây là kết quả của một lỗi mạng Unity hiện tại. Các cuộc gọi ClientRPC không thể được gọi trên lớp cha từ một lớp con. Một báo cáo lỗi đã được gửi và đang chờ bản sửa lỗi. Lỗi này tự biểu hiện bằng cách hiển thị cảnh báo sau trong bảng điều khiển:

Trong đó [RpcPath] là đường dẫn phuơng thức, [ObjectID] là ID của đối tượng và [NetID] là ID mạng.

## Tham chiếu Công việc

Khi viết một nhiệm vụ mới, trong một số trường hợp, cần phải truy cập vào một nhiệm vụ khác trong nhiệm vụ đó. Ví dụ, TaskA có thể muốn lấy giá trị của TaskB.SomeFloat. Để thực hiện điều này, TaskB cần được tham chiếu từ TaskA. Trong ví dụ này, TaskA trông giống như sau:

```
sử dụng UnityEngine; sử
dụng BehaviorDesigner.Runtime.Tasks;

lớp công khai TaskA: Hành động {

    công khai TaskB referenceTask;

    public void OnAwake () {

        Debug.Log (tham chiếuTask.SomeFloat);
    }
}
```

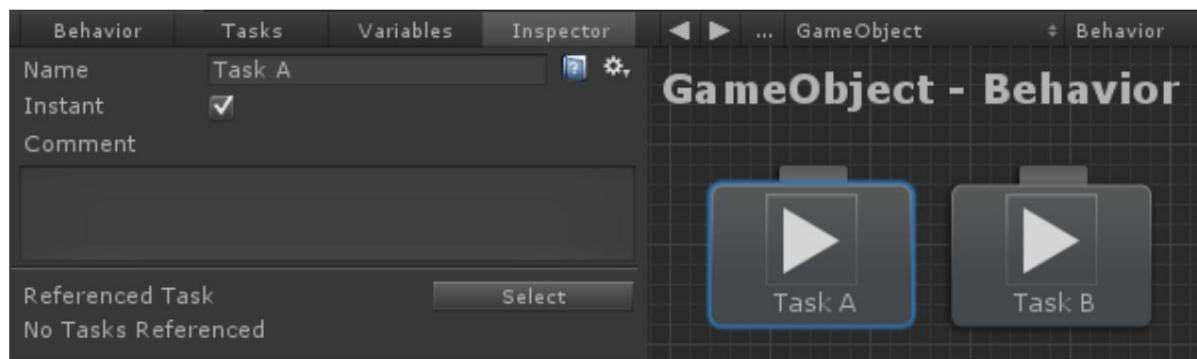
TaskB sau đó trông giống như :

```
sử dụng UnityEngine; sử
dụng BehaviorDesigner.Runtime.Tasks;

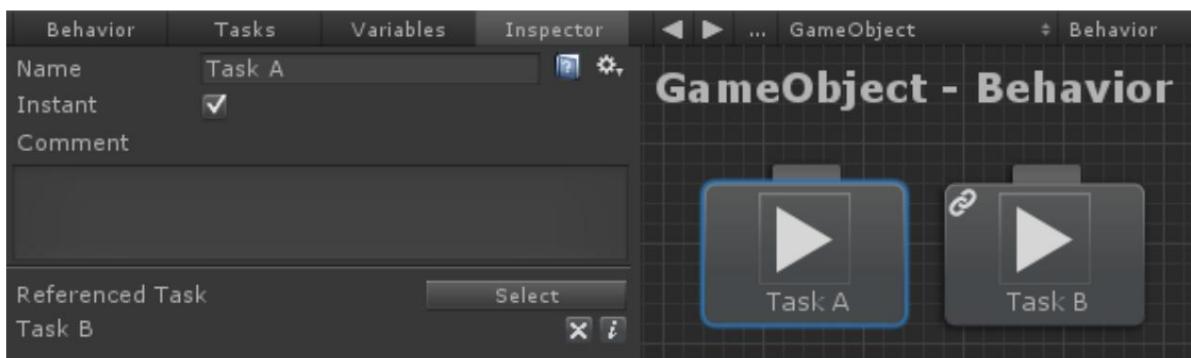
lớp công khai TaskB: Hành động {

    public float SomeFloat;
}
```

Thêm cả hai nhiệm vụ này vào cây hành vi của bạn trong Cây hành vi và chọn TaskA.



Bấm vào nút chọn. Bạn sẽ vào chế độ liên kết nơi bạn có thể chọn các tác vụ khác trong cây hành vi. Sau khi bạn chọn Nhiệm vụ B, bạn sẽ thấy rằng Nhiệm vụ B được liên kết như một nhiệm vụ được tham chiếu:



Đó là nó. Bây giờ khi bạn chạy cây hành vi TaskA sẽ có thể xuất ra giá trị của giá trị SomeFloat của TaskB. Bạn có thể xóa tham chiếu bằng cách nhập vào "x" ở bên phải của tên nhiệm vụ được tham chiếu. Nếu bạn nhập vào chữ "i" thì nhiệm vụ được liên kết sẽ đánh dấu trong



trái cam:

Các tác vụ cũng có thể được tham chiếu bằng cách sử dụng một mảng:

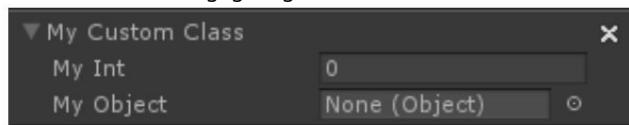
lớp công khai TaskA: Hành động {

```
    công khai TaskB [] referenceTasks;
}
```

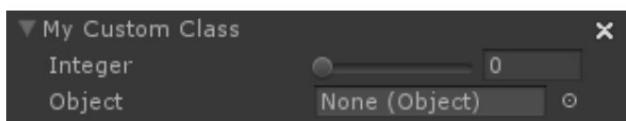
## Ngăn kéo đối tượng

Object Drawers rất giống với tính năng Unity thuộc tính [Drawers](#). Ngăn kéo đối tượng cho phép bạn tùy chỉnh giao diện của các đối tượng khác nhau trong trình kiểm tra. Để làm ví dụ, chúng tôi sẽ sửa đổi ví dụ về Đối tượng tùy chỉnh được chia sẻ trong phần [Tạo biến](#) được chia sẻ của riêng bạn chủ đề. Với trình kiểm tra mặc định, biến SharedCustomClass trông giống như

sau trong thanh tra:



Đối với ví dụ này, chúng tôi sẽ giới hạn phạm vi của số nguyên từ 0 đến 10 bằng cách sử dụng ngăn kéo đối tượng:



Ngăn kéo đối tượng sau được sử dụng để thực hiện điều này (tập lệnh này nằm trong thư mục Trình chỉnh sửa):

```
sử dụng UnityEngine; sử
dụng UnityEditor; sử dụng
BehaviorDesigner.Editor;
```

```
[CustomObjectDrawer (typeof (CustomClass))]
```

```

public class CustomClassDrawer: ObjectDrawer {

    ghi đè công khai void OnGUI (nhãn GUIContent) {

        var customClass = value as CustomClass;
        EditorGUILayout.BeginVertical (); if (FieldIns
        Inspector.DrawFoldout (customClass.GetHashCode (), label))
    {EditorGUI.indentLevel ++; customClass.myInt = EditorGUILayout.IntSlider ("Số
        nguyên",
        customClass.myInt, 0, 10);
        customClass.myObject = EditorGUILayout.ObjectField ("Đối tượng",
        customClass.myObject, typeof (UnityEngine.Object), true);
        EditorGUI.indentLevel--;
    }
    EditorGUILayout.EndVertical ();
}
}

```

Phương pháp duy nhất mà bạn cần ghi đè để ngăn kéo đối tượng hoạt động là phương thức OnGUI (nhãn GUIContent). Trong nhãn là tên của trường đang được vẽ. Cũng giống như ngăn kéo thuộc tính, bạn có thể chỉ định ngăn kéo đối tượng theo loại lớp hoặc theo thuộc tính. Ví dụ trên đang sử dụng phương thức kiểu lớp.

Như một ví dụ khác, chúng tôi sẽ chuyển đổi Thuộc tính phạm vi được sử dụng trong ví dụ của Unity thành Ngăn kéo đối tượng. Trước tiên, chúng ta cần tạo thuộc tính:

```

sử dụng UnityEngine;
sử dụng BehaviorDesigner.Runtime.Tasks; sử
dụng BehaviorDesigner.Runtime.ObjectDrawers;

public class RangeAttribute: ObjectDrawerAttribute {

    public float min;
    public float max;

    public RangeAttribute (float min, float max) {

        this.min = min;
        this.max = max;
    }
}

```

Bây giờ thuộc tính đã được tạo, chúng ta cần tạo ngăn kéo đối tượng thực tế (tập lệnh này nằm trong thư mục Editor):

```

sử dụng UnityEngine;
sử dụng UnityEditor;
sử dụng BehaviorDesigner.Editor;

```

```
[CustomObjectDrawer (typeof (RangeAttribute))] lớp công
khai RangeDrawer: ObjectDrawer {

    ghi đè công khai void OnGUI (nhãn GUIContent) {

        thuộc tính var rangeAttribute = (RangeAttribute); value =
            EditorGUILayout.Slider (nhãn, (float) giá trị,
rangeAttribute.min, rangeAttribute.max);
    }
}
```

Khi cả hai điều này đã được tạo, chúng ta có thể sử dụng nó trong một tác vụ:

```
sử dụng UnityEngine;
sử dụng BehaviorDesigner.Runtime; sử
dụng BehaviorDesigner.Runtime.Tasks; sử dụng
BehaviorDesigner.Runtime.ObjectDrawers;
```

```
public class NewAction: Action {

    [Range (5, 10)]
    public float rangedFloat; ghi đè
    công khai TaskStatus OnUpdate () {

        Debug.Log (rangedFloat); trả
        về TaskStatus.Success;
    }
}
```

Điều này sẽ hiển thị trong trình kiểm tra tác vụ dưới dạng:



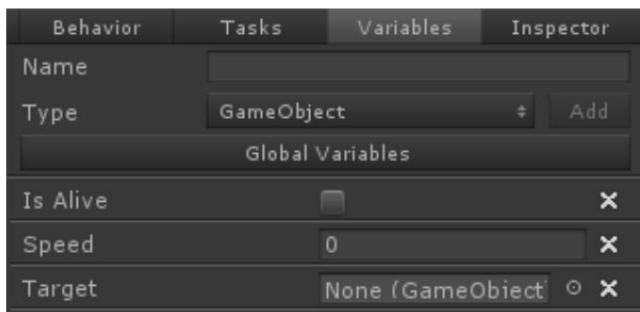
## Bộ đồng bộ hóa biến

Các biến được chia sẻ rất tuyệt vời để chia sẻ dữ liệu giữa các tác vụ và cây hành vi. Tuy nhiên, trong một số trường hợp, bạn muốn chia sẻ cùng một biến với các thành phần cây không phải là hành vi. Ví dụ: bạn có thể có một thành phần GUI Controller quản lý GUI. Bộ điều khiển GUI này hiển thị một phần tử GUI cho biết tác nhân đang được điều khiển bởi cây hành vi có còn sống hay không. Nó thực hiện điều này bằng cách có một boolean cho biết tác nhân còn sống hay không:

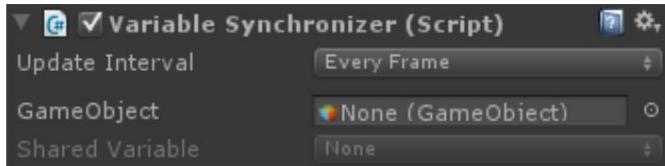
```
public bool isAlive {get; bộ; }
```

Với thành phần Bộ đồng bộ hóa biến, bạn có thể tự động giữ boolean này và Biến được chia sẻ tương ứng được đồng bộ hóa với nhau.

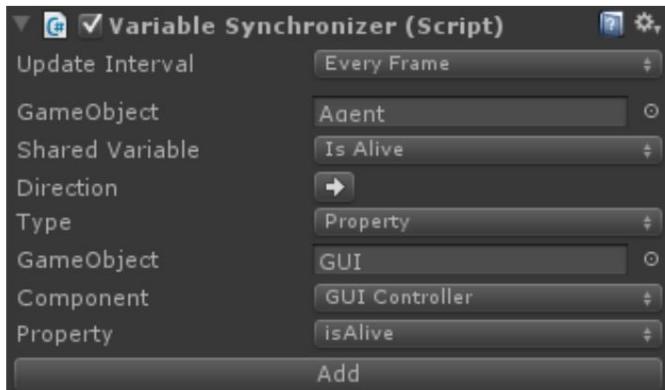
Để thiết lập Trình đồng bộ hóa biến, trước tiên hãy đảm bảo rằng bạn đã tạo Biến được chia sẻ mà bạn muốn đồng bộ hóa. Đối với ví dụ này, chúng tôi đã tạo ba Biến được chia sẻ:



Sau đó, thêm thành phần Behavior Designer / Variable Synchronizer vào GameObject.

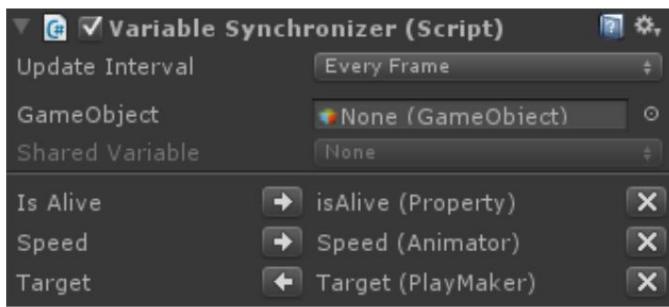


Tiếp theo, bắt đầu thêm Biến được chia sẻ mà bạn muốn tiếp tục đồng bộ hóa. Đối với ví dụ này, chúng ta sẽ thêm biến Is Alive đã được đề cập trước đó.



1. Chỉ định GameObject chứa cây hành vi có Biến được chia sẻ mà bạn muốn đồng bộ hóa.
2. Chọn từ hộp bật lên mà bạn muốn sử dụng Biến được chia sẻ.
3. Chỉ định một hướngh. Nếu mũi tên hướngh sang trái thì bạn đang đặt giá trị Biến được chia sẻ. Nếu mũi tên hướngh sang phải thì bạn đang nhận được giá trị Biến được chia sẻ.
4. Chỉ định kiểu đồng bộ hóa. Hiện tại, các loại sau được hỗ trợ: Behavior Designer, Property, Animator và PlayMaker.
5. Các bước còn lại sẽ tùy thuộc vào kiểu đồng bộ hóa được chọn. Trong này ví dụ Thuộc tính đã được chọn, vì vậy bạn sẽ cần chọn thành phần chứa thuộc tính mà bạn muốn đồng bộ hóa với Biến được chia sẻ.
6. Nhấp vào Thêm.

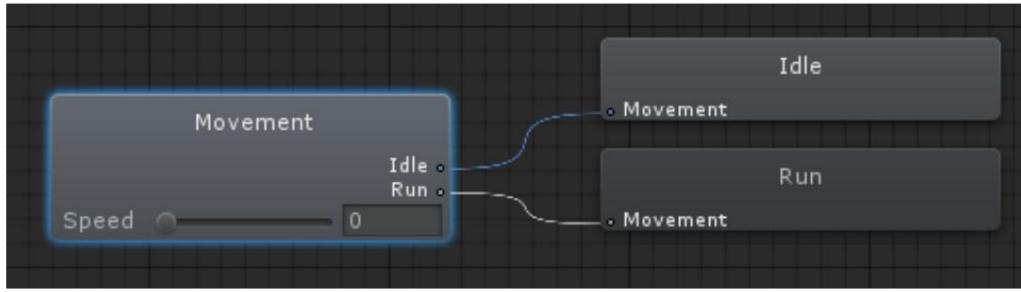
Sau khi được thêm, Biến được chia sẻ Is Alive sẽ đặt thuộc tính isAlive tại một khoảng thời gian được chỉ định bởi Khoảng thời gian cập nhật. Ảnh chụp màn hình sau đây chứa một số biến được đồng bộ hóa khác:



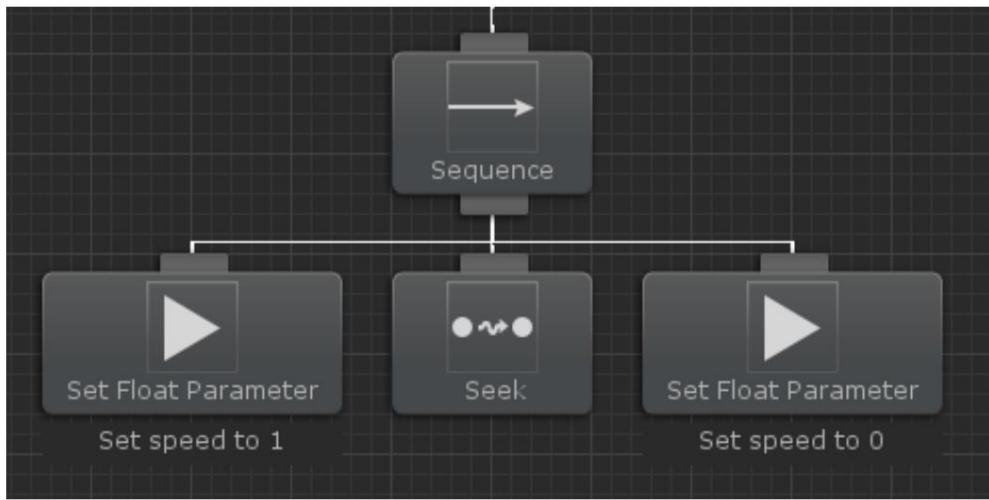
- Biến Is Alive được chia sẻ đang đặt thuộc tính isAlive.
- Biến Chia sẻ Tốc độ đang thiết lập thông số Speed Animator.
- Biến mục tiêu được chia sẻ đang được đặt bởi biến Target PlayMaker.

## Đồng bộ hóa hoạt ảnh

Behavior Designer bao gồm một tập hợp các tác vụ Animator cho phép bạn phát các hoạt ảnh trong cây. Hãy xem xét cây dưới đây:



Đây là một bộ điều khiển Animator cực kỳ đơn giản, sử dụng cây hòa trộn để trộn giữa trạng thái Chờ và Chạy dựa trên tham số Tốc độ. Nếu bạn muốn sau đó đồng bộ hóa cây pha trộn này với tác vụ Tìm kiếm thì cây hành vi của bạn sẽ trông tư ơng tự như :



Trước khi tác vụ Seek bắt đầu, tham số Tốc độ được đặt thành 1 để cây hòa trộn có thể phát hoạt ảnh chạy. Sau khi tác vụ Seek hoàn thành, tham số Tốc độ được đặt thành 0 để phát lại hoạt ảnh Không hoạt động.

Với một cây hành vi nhỏ và Bộ điều khiển Animator, quá trình này hoạt động tốt. Tuy nhiên, khi số lượng hoạt ảnh tăng lên cho tác nhân của bạn, phương pháp này bắt đầu trở nên cực kỳ cồng kềnh. Ngoài ra, cây hành vi của bạn cũng bắt đầu trở nên cực kỳ dài dòng với tất cả các tác vụ Animator và nếu bạn thay đổi quá trình chuyển đổi trong Animator Controller | 44

bạn phải làm lại cây hành vi để làm việc với thay đổi Bộ điều khiển Animator đó.

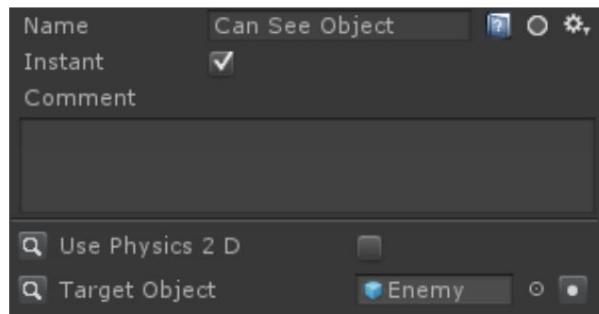
Có một cách tốt hơn.

Cách tiếp cận được khuyến nghị để đồng bộ hóa các hoạt ảnh trong cây hành vi của bạn là không đồng bộ hóa các hoạt ảnh trong cây hành vi. Thay vào đó, bộ điều khiển nhân vật của đại lý của bạn sẽ làm điều đó cho bạn. Sử dụng NavMeshAgent của Unity làm ví dụ, khi cây hành vi đặt đích đến NavMeshAgent (chẳng hạn như với tác vụ Seek), vận tốc của NavMeshAgent sẽ thay đổi để di chuyển tới đích. Sau đó, bộ điều khiển nhân vật sẽ sử dụng vận tốc và chuyển các chuyển động thành các tham số mà Bộ điều khiển hoạt hình có thể hiểu được. MecWarriors đã có một hướng dẫn tuyệt vời về cách thực hiện điều này. Trang MecWarriors không còn hoạt động nhưng bạn có thể tìm thấy phiên bản được lưu trong bộ nhớ cache của hướng dẫn [tại đây](#). Unity cũng có một hướng dẫn tương tự trong [tài liệu của họ](#). Việc triển khai tương tự có thể được thực hiện với [Apex Path](#) và A \* Pathfinding Project.

Ưu điểm của cách tiếp cận này là cây hành vi của bạn sau đó hoàn toàn không nhận thức được các hoạt ảnh và nó thậm chí có thể hoạt động với [chuyển động gốc](#). Bạn cũng không làm phức tạp cây của mình với các tác vụ Animator và làm cho cây của bạn sạch sẽ hơn rất nhiều. Bộ điều khiển ký tự cuối cùng sử dụng phương pháp này và đối với nội dung đó, chúng tôi đã tạo một cây hành vi khá lớn. Bạn có thể xem cây hành vi trên [trang này](#) và bạn sẽ nhận thấy rằng không có nhiệm vụ Animator nào trong cây. Một thành phần cầu nối đang liên kết vận tốc của NavMeshAgent với đầu vào của bộ điều khiển nhân vật, sau đó cho phép Bộ điều khiển ngửi thứ ba xử lý các hoạt ảnh.

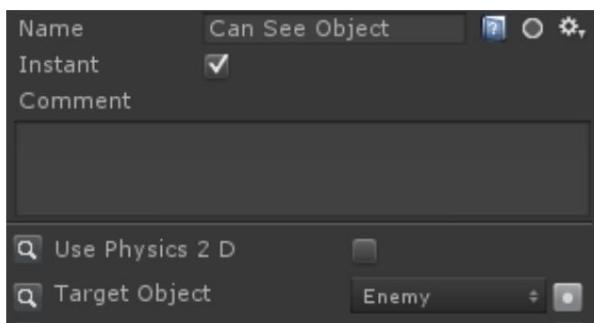
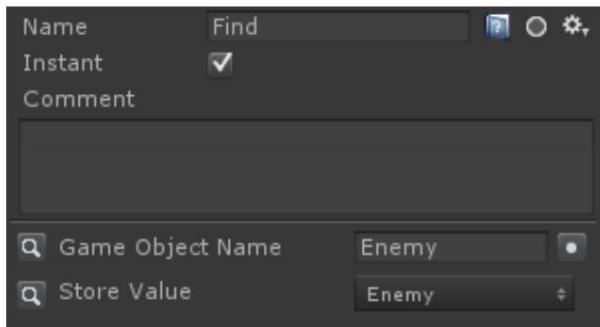
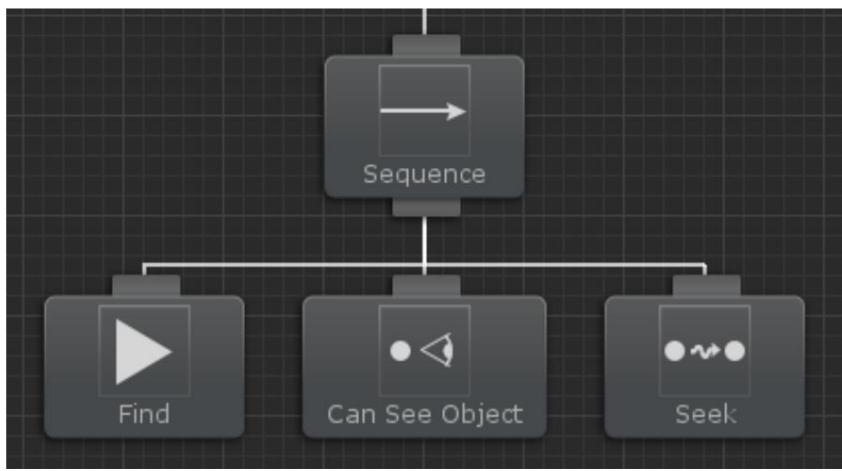
## Tham chiếu các đối tượng cảnh

Khi bạn đang tạo cây của mình, thông lệ thường là tham chiếu các đối tượng trong khung cảnh từ một Biến được chia sẻ hoặc nhiệm vụ. Ví dụ, trong cây có thể thấy Đối tượng này đang xác định xem đối tượng Kẻ thù có thể được nhìn thấy hay không:



Khi bạn đang kiểm tra, bạn thấy rằng mọi thứ đang hoạt động tốt và bây giờ bạn muốn tạo một nhà lấp ghép từ nó để bạn có thể có nhiều tác nhân trong cảnh của mình. Ngay sau khi bạn biến tác nhân thành nhà lấp ghép và xóa nó khỏi hiện trường, bạn sẽ thấy rằng tham chiếu Đối tượng mục tiêu đã bị thiếu. Lý do điều này xảy ra là vì với tư cách là tài sản cấp dự án (tệp prefab, hoặc tệp biến toàn cục) không thể tham chiếu đến các đối tượng trong cảnh. Đây là một hạn chế của Unity và cách để khắc phục nó là điền biến vào thời gian chạy sau khi bạn đã tạo nhà lấp ghép.

Đối tượng tham chiếu cảnh có thể được điền vào thời gian chạy theo nhiều cách khác nhau. Một cách là sử dụng tác vụ Tìm và tìm kiếm đối tượng theo tên:



Khi tác vụ Tìm chạy, nó sẽ tìm kiếm một GameObject có tên là "Kẻ thù" và sau đó đặt nó vào một SharedVariable. Khi Can See Object chạy, nó sẽ sử dụng SharedVariable đó để tìm kiếm đối tượng địch.

Một cách khác để điền giá trị vào thời gian chạy là có một thành phần đã có trong khung cảnh có một tham chiếu đến đối tượng. Sau khi nhà lắp ghép được tạo ra, thành phần này sau đó có thể đặt giá trị của SharedVariable mà Đối tượng Có thể thấy sử dụng.

```

using UnityEngine;
using BehaviorDesigner.Runtime;
  
```

```

lớp công cộng Spawner: MonoBehaviour {
```

```

public GameObject m_Enemy;

public void Start () {

    var behaviorTree = GetComponent <BehaviorTree> ();
    behaviorTree.SetVariableValue ("Kẻ thù", m_Enemy);
}

}
```

## Thuộc tính nhiệm vụ

Behavior Designer hiển thị các thuộc tính nhiệm vụ sau: HelpURL, TaskIcon, TaskCategory, TaskDescription và LinkedTask.

### URL trợ giúp

Nếu bạn mở bảng trình kiểm tra tác vụ, bạn sẽ thấy trên biểu tượng tài liệu ở trên cùng bên phải. Biểu tượng tài liệu này cho phép bạn liên kết trang web trợ giúp với một công việc. Bạn tạo liên kết này với thuộc tính HelpURL. Thuộc tính HelpURL nhận một tham số là liên kết đến trang web.

```
[HelpURL ("http://www.example.com")] lớp công khai
```

```
MyTask: Hành động {
```

### Biểu tượng Nhiệm vụ

Các biểu tượng tác vụ được hiển thị trong cây hành vi bằng cách sử dụng thuộc tính TaskIcon và được sử dụng để giúp hình dung tác vụ thực hiện. Các đường dẫn có liên quan đến thư mục dự án gốc. Từ khóa {SkinColor} sẽ được thay thế bằng màu da Unity hiện tại, "Sáng" hoặc "Tối".

```
[TaskIcon ("Assets / Path / To / {SkinColor} Icon.png")] lớp công khai
```

```
MyTask: Action {
```

### Hạng mục công việc

Tổ chức bắt đầu trở thành một vấn đề khi bạn tạo ra ngày càng nhiều nhiệm vụ. Để làm được điều đó, bạn có thể sử dụng thuộc tính TaskCategory:

```
[TaskCategory ("Common")] public
class Seek: Action {
```

Nhiệm vụ này bây giờ sẽ được phân loại theo danh mục chung:



Các danh mục có thể đư ợc lồng vào nhau bằng cách tách tên danh mục bằng dấu gạch chéo:

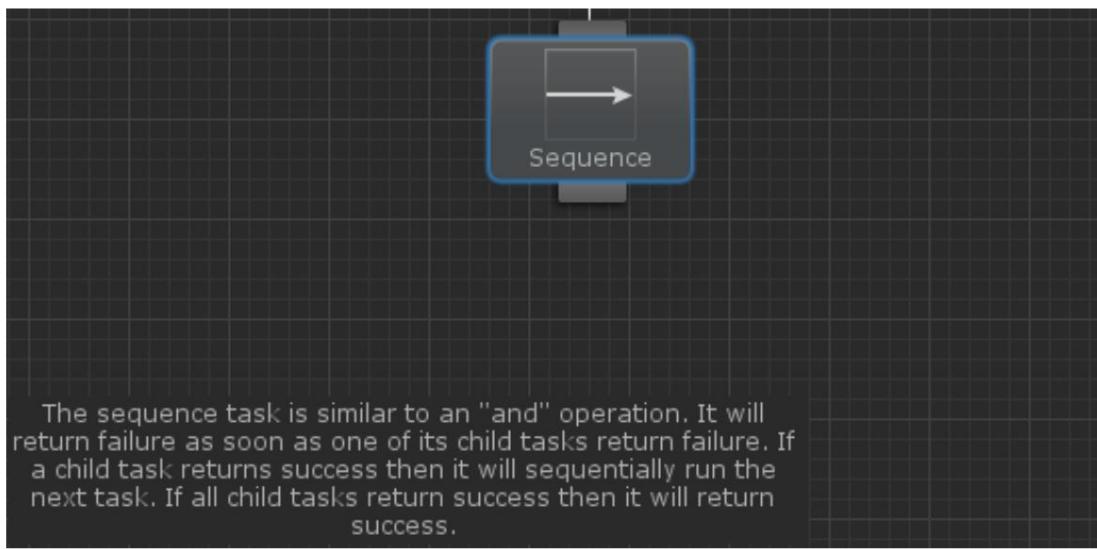
```
[TaskCategory ("RTS / Harvester")] lớp
công khai HarvestGold: Hành động {
```

## Mô tả công việc

Thuộc tính TaskDescription cho phép bạn hiển thị nhận xét cấp lớp của mình trong chế độ xem biểu đồ. Ví dụ: mô tả trình tự bắt đầu bằng:

```
[TaskDescription ("Nhiệm vụ trình tự tương tự như thao tác \" và \" . . .")]
public class Sequence: Composite {
```

Mô tả này sau đó sẽ đư ợc hiển thị ở khu vực dư ới cùng bên trái của biểu đồ:



Tác vụ đư ợc Liên kết

Biến rất tốt khi bạn muốn chia sẻ thông tin giữa các tác vụ. Tuy nhiên, bạn sẽ nhận thấy rằng không có cái gọi là "SharedTask". Khi bạn muốn một nhóm nhiệm vụ chia sẻ các nhiệm vụ giống nhau, hãy sử dụng thuộc tính LinkedTask. Ví dụ, hãy xem nhiệm vụ bảo vệ tác vụ. Khi bạn tham chiếu một nhiệm vụ với nhiệm vụ bảo vệ, tác vụ tương tự đó sẽ tham chiếu trở lại nhiệm vụ bảo vệ tác vụ ban đầu. Việc liên kết các tác vụ là không cần thiết, nó là một thuộc tính thuyết phục hơn để đảm bảo các tru ờng có các giá trị đư ợc đồng bộ hóa. Thêm thuộc tính sau vào tru ờng của bạn để bật liên kết tác vụ:

[LinkedTask]

```
public TaskGuard [] linkedTaskGuards = null;
```

Để thực hiện một liên kết trong trình chỉnh sửa, hãy thực hiện các bước tương tự [tham chiếu một tác vụ khác](#).

## Tích hợp

Behavior Designer bao gồm nhiều tác vụ tích hợp với nội dung của bên thứ ba. Đối với hầu hết các tích hợp đó, không cần thực hiện thêm bước nào và chúng có thể đư ợc thêm vào cây hành vi và sau đó gán giá trị của chúng. Tuy nhiên, các tích hợp dưới đây có giải thích chi tiết hơn về cách chúng hoạt động.

[Hệ thống đối thoại cho sự thông nhất](#)

[Playmaker](#)

[Bộ điều khiển ký tự cuối cùng](#)

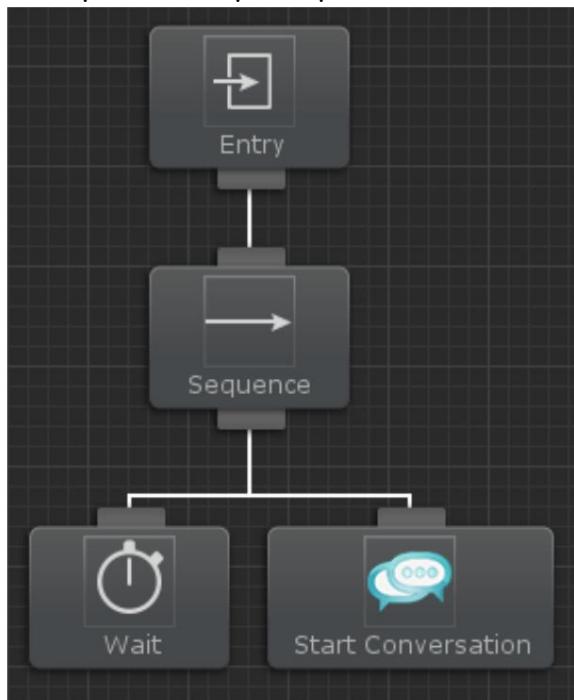
[uScript](#)

## Hệ thống đối thoại

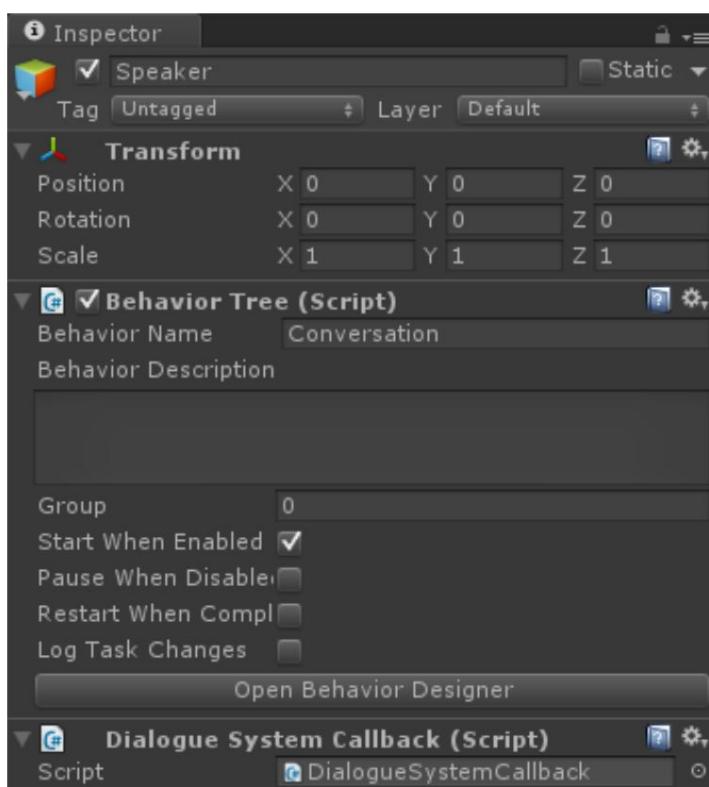
Hệ [thống đối thoại](#) là một hệ thống đối thoại hoàn chỉnh cho Unity. Behavior Designer đư ợc tích hợp với Hệ thống Đối thoại bằng cách cho phép bạn quản lý các cuộc trò chuyện, sửa, trình tự và nhiệm vụ trong cây hành vi của mình. Ngoài ra, Hệ thống đối thoại đư ợc tích hợp với Trình thiết kế hành vi để nó có thể đồng bộ hóa các biến với Lua và cây hành vi bắt đầu / dừng | 49

với các lệnh tuần tự. Có thể tìm thấy thêm thông tin về mặt này của quá trình tích hợp [tại đây](#). Tất cả các tệp tích hợp Hệ thống Đồi thoại đều nằm trên [trang tải xuống](#).

Để bắt đầu, trước tiên hãy đảm bảo rằng bạn đã cài đặt Hệ thống Đồi thoại cho Unity và đã nhập gói tích hợp. Sau khi các tệp đó được nhập, bạn đã sẵn sàng để bắt đầu tạo cây hành vi với Hệ thống đồi thoại. Hãy tạo một cây rất cơ bản với tác vụ tuần tự có tác vụ Chờ và tác vụ Bắt đầu hội thoại:



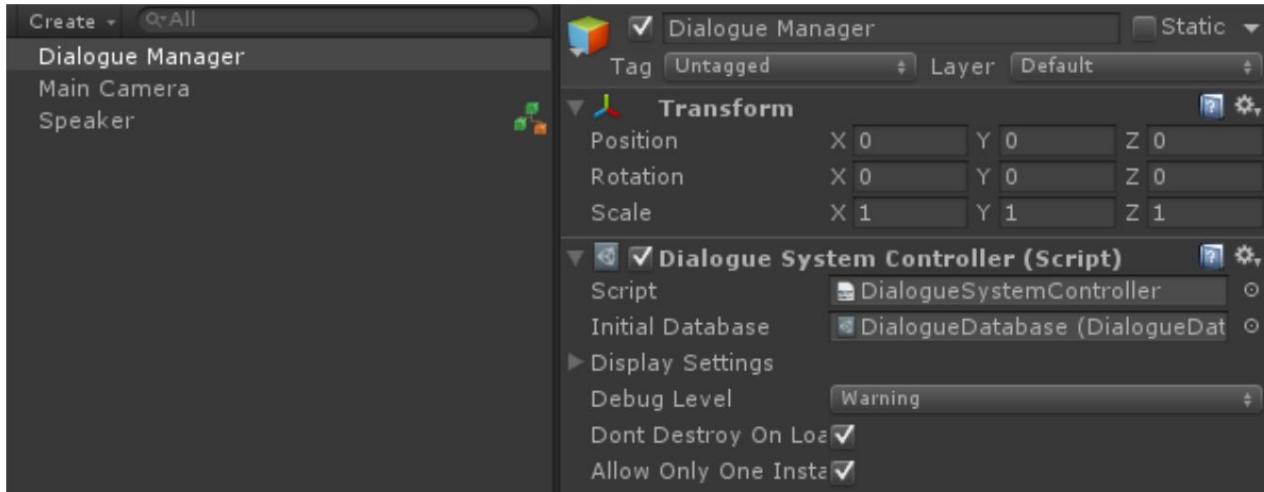
Khi Hệ thống đồi thoại kết thúc bằng một cuộc hội thoại hoặc trình tự, nó sẽ gọi lại cho Nhà thiết kế hành vi để cho Nhà thiết kế hành vi biết rằng việc đó đã hoàn tất. Để điều này xảy ra, thành phần Gọi lại Hệ thống Đồi thoại phải được thêm vào cùng một GameObject mà cây hành vi của bạn đang sử dụng:



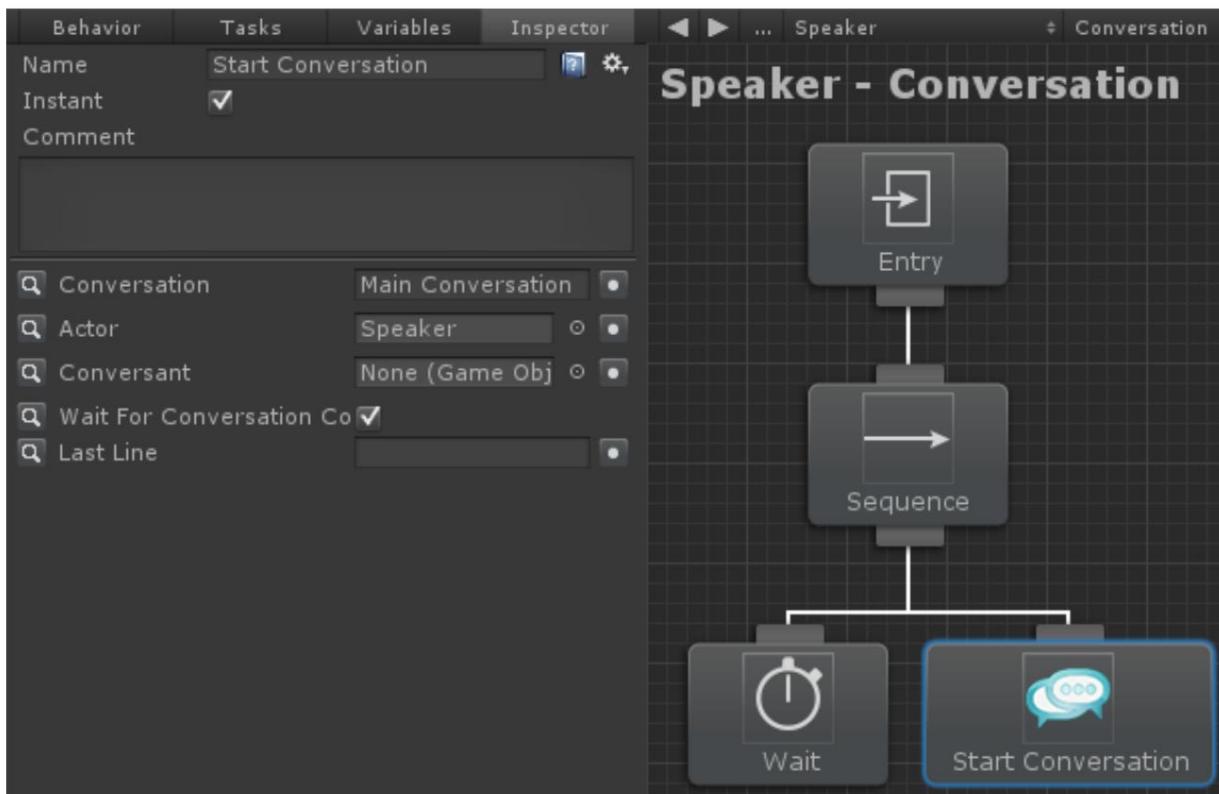
Cơ sở dữ liệu và tạo một cuộc trò chuyện cơ bản:



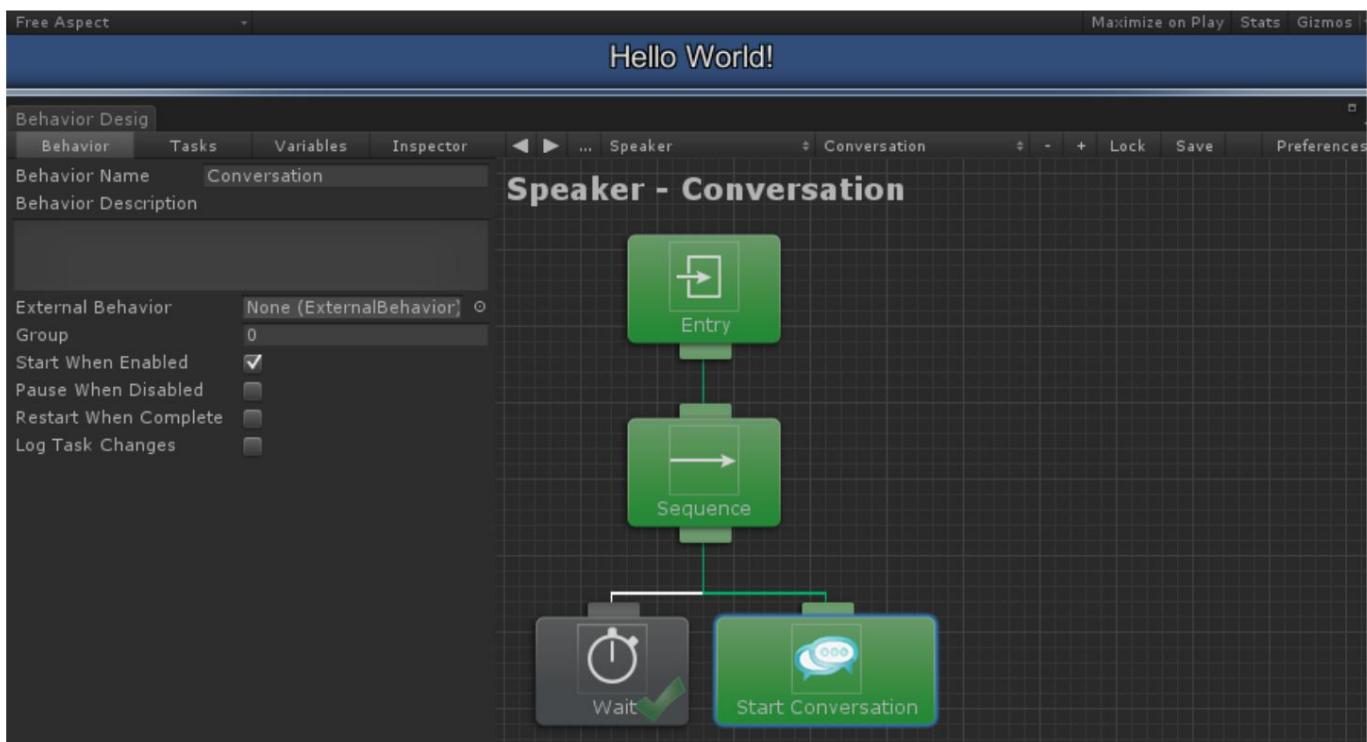
Ghi lại tên cuộc trò chuyện vì điều đó sẽ cần thiết sau này. Gán cơ sở dữ liệu đó cho Bộ điều khiển Hệ thống Đôi thoại:



Bước cuối cùng là chỉ cần gán các giá trị trong tác vụ Bắt đầu cuộc trò chuyện. Hai giá trị duy nhất được yêu cầu là tên cuộc hội thoại và tác nhân GameObject:



Sau khi các giá trị đó đã được chỉ định, hãy nhấn play và bạn sẽ thấy dòng chữ "Hello World" xuất hiện ở đầu màn hình trò chơi:



Chủ đề này hầu như không làm trầy xư ớc bề mặt cho những gì có thể với tích hợp Hệ thống đối thoại / Thiết kế hành vi. Để có một ví dụ phức tạp hơn, hãy xem dự án mẫu Hệ thống đối thoại.

## Bộ điều khiển nhân vật phụ

Gói tích hợp này có thể được tải xuống trên [trang tải xuống](#).

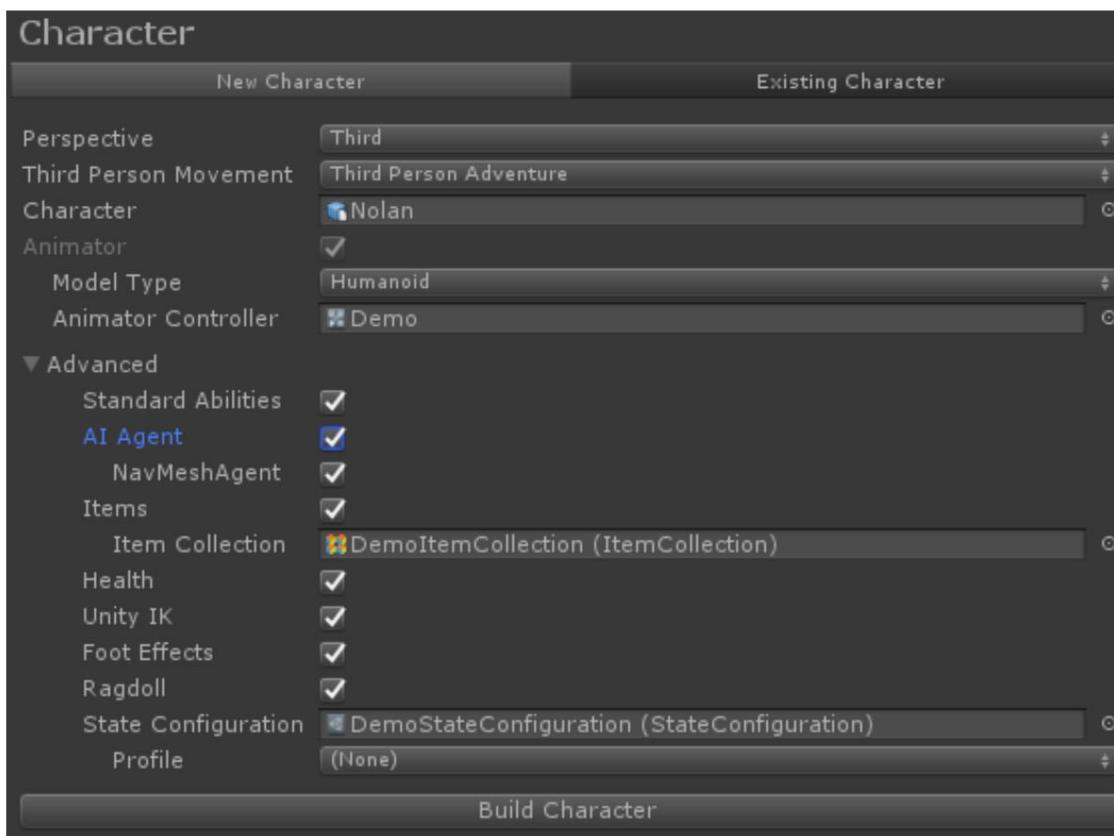
Bộ điều khiển Nhân vật Opsive được tích hợp với Nhà thiết kế hành vi cho phép AI của bạn thực hiện bất kỳ chức năng nào mà một nhân vật do người chơi điều khiển có thể thực hiện. Tích hợp Ultimate Character Controller chứa cảnh demo được thiết kế để hoạt động với cả vũ khí cận chiến và bắn súng.

Khi tích hợp được nhập, sẽ có bốn cảnh mẫu được bao gồm:

- Cảnh cận chiến góc nhìn thứ nhất
- Cảnh bắn súng góc nhìn thứ nhất
- Cảnh cận chiến góc nhìn thứ ba
- Cảnh bắn súng góc nhìn thứ ba

Cây hành vi cho bốn cảnh này hoàn toàn giống nhau và điểm khác biệt duy nhất là góc nhìn của người chơi và vũ khí mà các nhân vật đã trang bị.

Khi bạn đang tạo một nhân vật mới sẽ được sử dụng với Behavior Designer, hãy đảm bảo rằng bạn đã bật AI Agent trong Trình quản lý nhân vật. Nếu bạn đang sử dụng Lưới điều hướng của Unity, bạn cũng nên bật chuyển đổi NavMeshAgent.



Sau khi nhân vật đã được tạo, các thành phần sau cũng sẽ được thêm vào:

- Cây hành vi
- Tác nhân cây hành vi

Thành phần Tác nhân Cây Hành vi có thể được tìm thấy trong [gói tích hợp Bộ điều khiển Nhân vật Opsive](#). Trong [cảnh demo](#), bạn sẽ nhận thấy rằng cũng có một thành phần Demo Agent được thêm vào để bổ sung sức khỏe và đạn của nhân vật dưới dạng thuộc tính cho [Bản đồ thuộc tính](#) của nhà thiết kế hành vi tính năng.

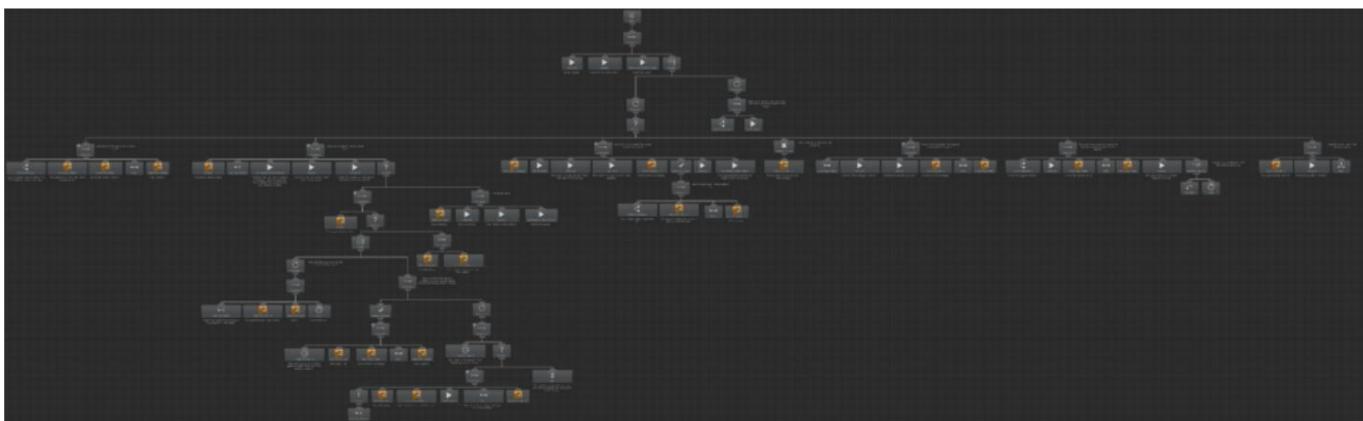
---

Sau khi nhân vật đã được thiết lập, đến lúc tạo cây hành vi của riêng bạn. Cây hành vi được bao gồm cung cấp một cái nhìn tổng quan tuyệt vời về luồng cây hành vi, vì vậy phần còn lại của tài liệu này sẽ mô tả cách cây hành vi đó hoạt động. Để tận dụng tối đa Behavior Designer, bạn nên tạo cây hành vi của riêng mình.

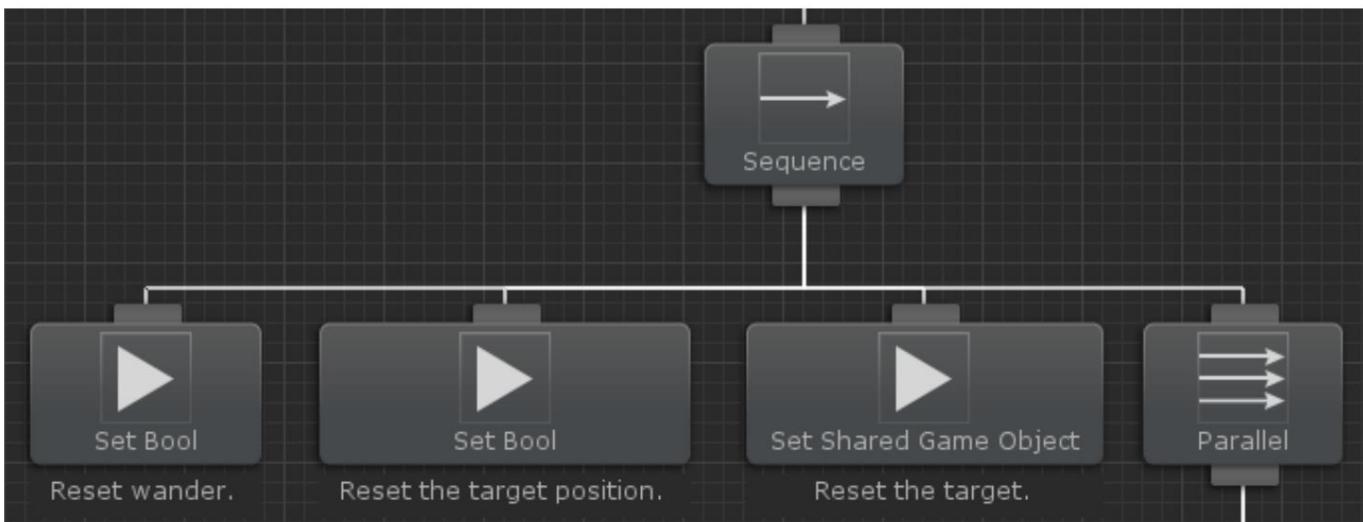
Cây hành vi được bao gồm sẽ thực hiện các chức năng sau:

- Tấn công nếu người chơi trong tầm nhìn.
- Tấn công nếu người chơi tấn công tác nhân.
- Di chuyển đến vị trí nguồn âm thanh nếu nghe thấy âm thanh từ đầu phát.
- Tìm kiếm người chơi nếu người chơi bị mất.
- Nhận sức khỏe nếu cần.
- Nhận đạn nếu cần.
- Tuần tra nếu không có hành động nào khác là cần thiết.

Toàn bộ cây hành vi trông giống như :



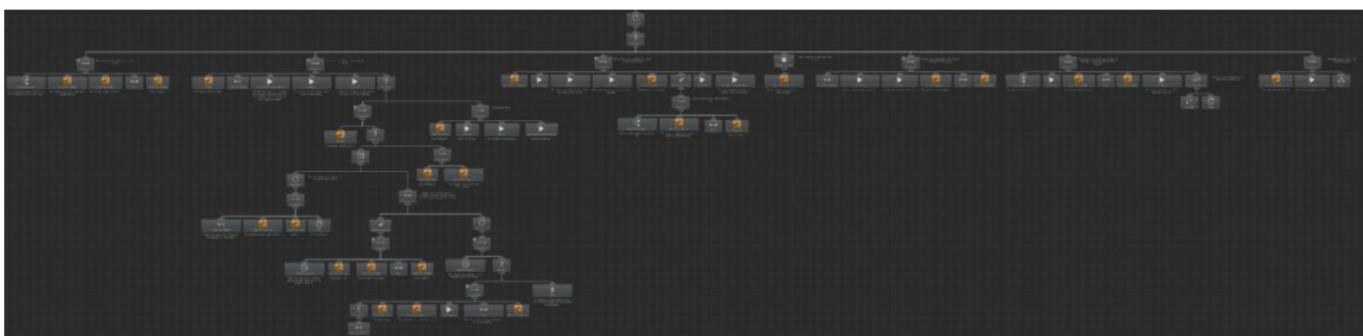
Cây hành vi đánh giá từ trên xuống dưới, từ trái qua phải. Điều này có nghĩa là tác vụ đầu tiên chạy là tác vụ Trình tự ngay bên dưới tác vụ Entry:



Nhiệm vụ Trình tự là một nhiệm vụ Tổng hợp sẽ đánh giá tất cả các con của nó cho đến khi một con trả về trạng thái thất bại. Nhiệm vụ Trình tự có thể được coi là một AND giữa các nhiệm vụ con. Ba tác vụ đầu tiên mà tác vụ Trình tự này sẽ thực thi liên quan đến việc đặt lại cây hành vi để sẵn sàng cho lần chạy tiếp theo. Các biến mà các tác vụ này đang đặt lại sẽ được mô tả sau. Một tác vụ song song sau đó được sử dụng để thực thi nhiều con cùng một lúc. Có hai nhánh mà tác vụ Song song này sẽ thực thi:

- Nhánh trái sẽ thực hiện các hành động thực tế như di chuyển hoặc tấn công.
- Nhánh bên phải sẽ cập nhật vị trí của người chơi khi biến Cập nhật Vị trí là đúng.

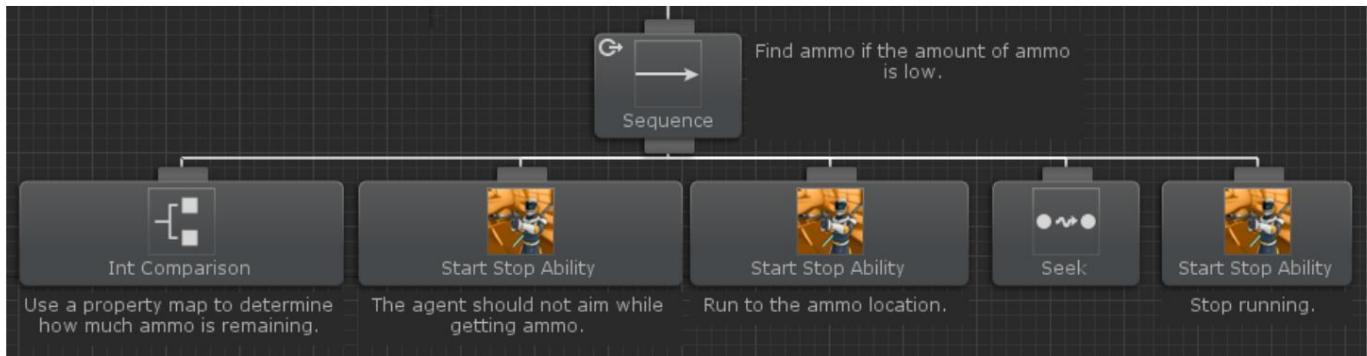
Chức năng chính của cây được chứa trong nhánh bên trái:



Nhánh này được bổ trợ bằng nhiệm vụ Bộ lặp được đặt thành Lặp lại Mãi mãi. Bộ lặp | 54

task là một Decorator sẽ tiếp tục thực thi nhánh con cho đến khi được yêu cầu. Vì Lặp lại mãi mãi được bật, Bộ lặp sẽ giữ cho nhánh hoạt động cho đến khi cây bheavior kết thúc. Một Bộ chọn được gắn với Bộ lặp. Nhiệm vụ Selector sẽ thực hiện các phần tử con của nó cho đến khi một phần tử con trả về thành công. Bộ chọn có thể được coi là HOẶC giữa các nhiệm vụ con.

Bởi vì cây hành vi đánh giá từ trái sang phải nên các nhánh ở bên trái có mức độ ưu tiên cao hơn so với các nhánh ở bên phải. Điều này có nghĩa là nhánh đầu tiên sẽ thực thi là nhánh kiểm tra để xác định xem đại lý có hết đạn hay không:

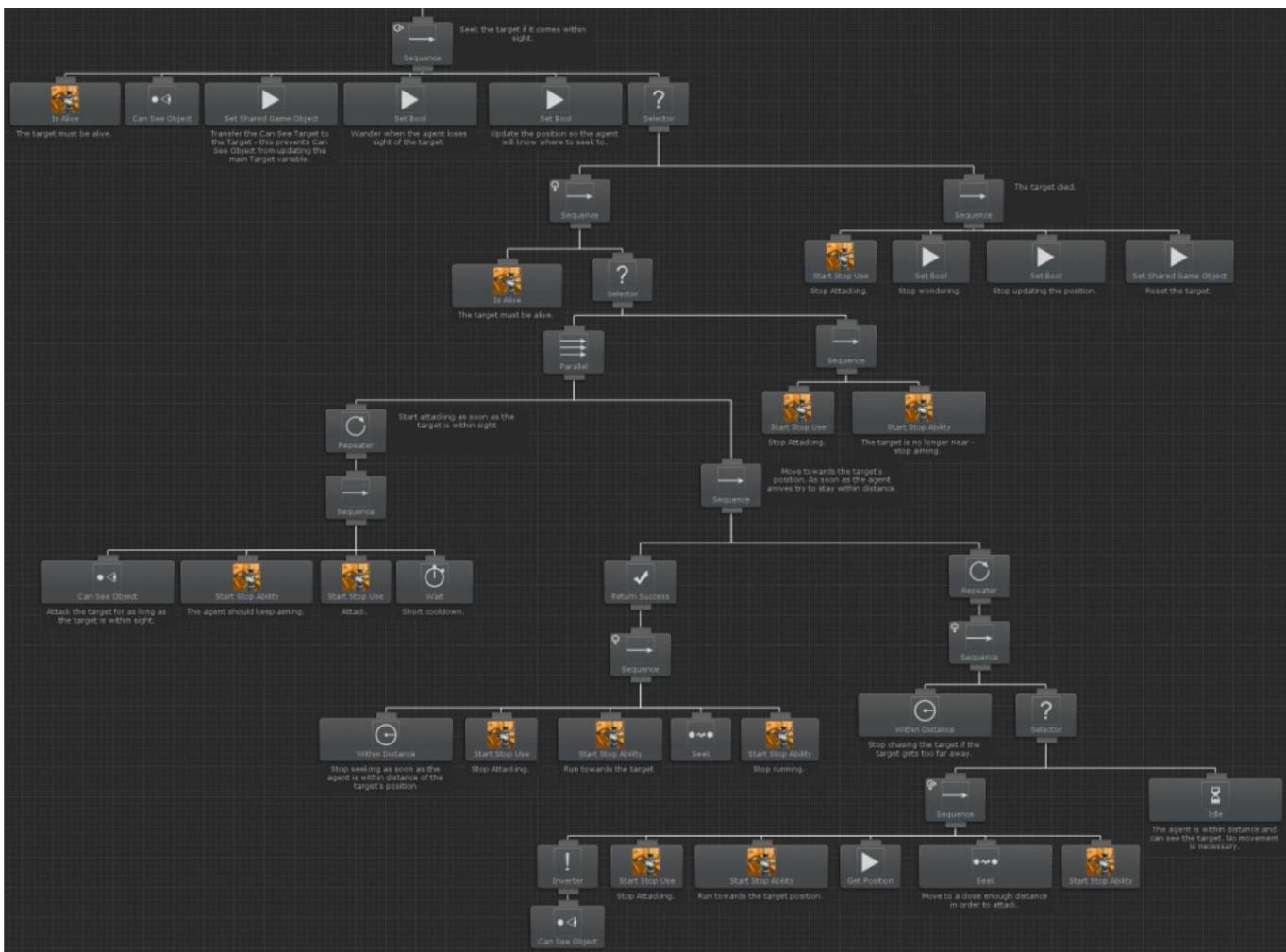


Nhiệm vụ trình tự của nhánh đạn có [Điều kiện hủy bỏ](#) ưu tiên thấp hơn bộ.

Bỏ qua có điều kiện cho phép cây hành vi đánh giá lại các tác vụ con Các nhiệm vụ có điều kiện khi các nhánh khác đang hoạt động. Nếu nhân vật hết đạn thì nhân vật đó không thể làm gì được nên nhánh này có quyền ưu tiên cao nhất. Nếu nhánh ưu tiên thấp hơn (bất kỳ nhánh nào ở bên phải nhánh đạn) đang hoạt động và tác nhân sắp hết đạn thì lệnh hủy bỏ có điều kiện sẽ hủy nhánh ưu tiên thấp hơn và bắt đầu thực hiện nhánh đạn.

Nhánh đạn là một nhánh tư ơng đối đơn giản, đầu tiên kiểm tra để xác định nhân vật có bao nhiêu đạn với nhiệm vụ Int Comparison Conditional. Nhiệm vụ So sánh Int này sẽ so sánh đạn của đặc vụ với một số lư ợng không đổi. Lư ợng đạn được lấy thông qua thiết lập ánh xạ thuộc tính Đạn thông qua thành phần Demo Agent. Nếu số tiền này nhỏ hơn hằng số thì tác vụ Int Comparison sẽ trả về thành công và nhánh sẽ thực thi.

Nếu đặc vụ còn ít đạn thì nó sẽ ngừng nhắm đến nhiệm vụ Start Stop Ability. Đây là một nhiệm vụ được tạo cho Ultimate Character Controller và nó sẽ bắt đầu hoặc dừng một khả năng dựa trên các giá trị được chỉ định. Trong trường hợp này, nhiệm vụ sẽ dừng khả năng Aim vì nhân vật không nên nhắm khi lấy đạn. Một nhiệm vụ Start Stop Khả năng khác được chạy và nhiệm vụ này sẽ bắt đầu nhiệm vụ Thay đổi tốc độ để nhân vật bắt đầu chạy về vị trí đạn. Nhiệm vụ Tìm kiếm (từ [Gói chuyển động](#)) sẽ sử dụng lư ới điều hướng của Unity để di chuyển nhân vật đến vị trí được chỉ định. Sau khi nhân vật đến vị trí, nhiệm vụ Start Stop Ability chỉ được thực thi lần nữa, lần này nó sẽ dừng khả năng Speed Change để nhân vật ngừng chạy.



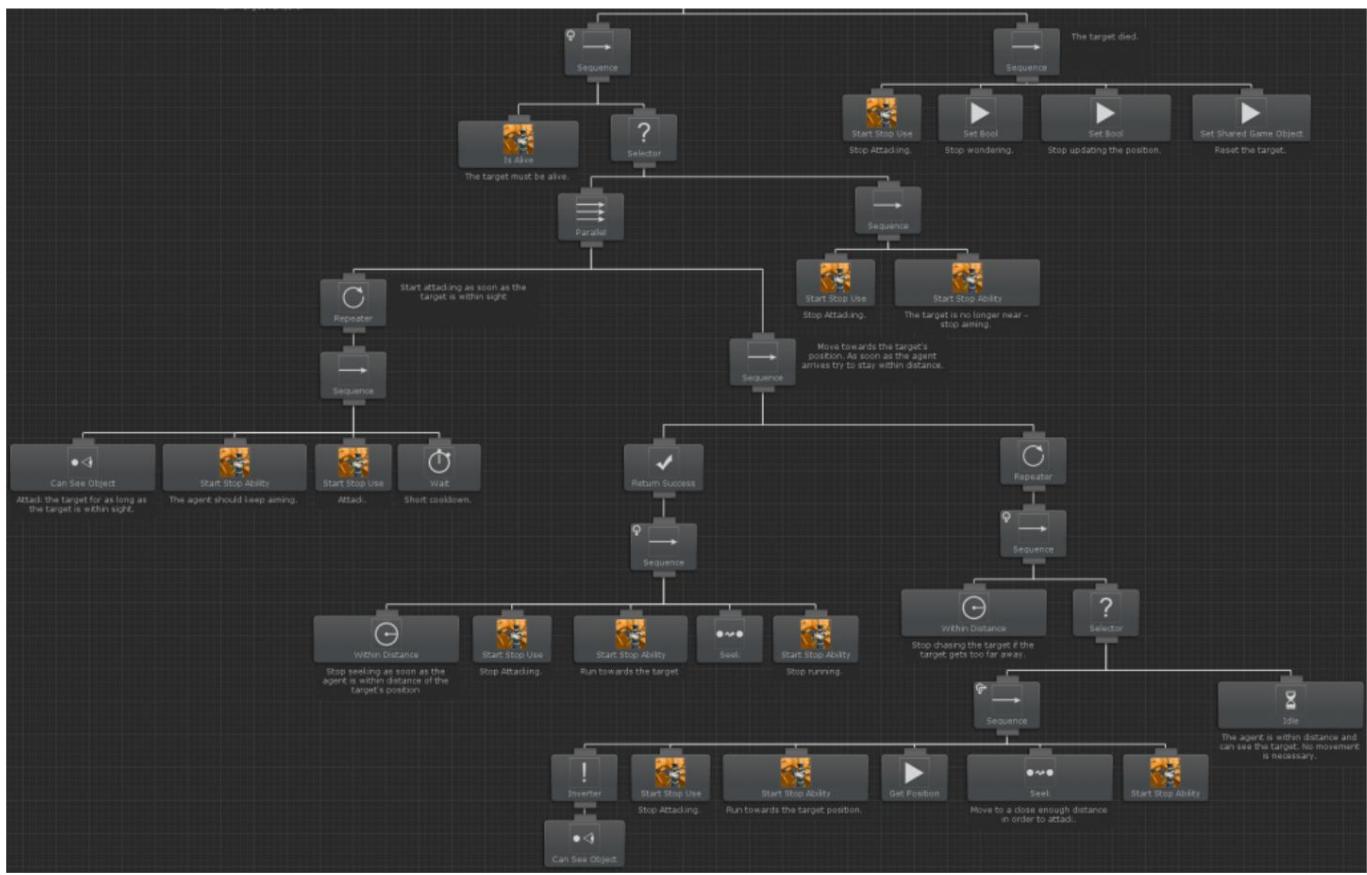
Nếu đặc vụ không cần đạn thì nhánh có mức ưu tiên cao nhất tiếp theo là nhánh Có thể nhìn thấy.

Nhánh này chưa hành động chính trong đó nó sẽ thực sự tấn công người chơi khi người chơi ở trong tầm nhìn. Tự ơng tự như nhánh Đạn, nhánh có thể thấy sử dụng Hủy bỏ có điều kiện mức độ ưu tiên thấp hơn nên nó sẽ hủy bỏ bất kỳ nhánh nào có mức ưu tiên thấp hơn. Nhánh Can See sẽ có thể hủy bỏ bất kỳ nhánh nào ở bên phải của nó, có nghĩa là nó sẽ không thể hủy bỏ nhánh Đạn vì nhánh Đạn nằm ở bên trái của nhánh Cần Xem.

Nhiệm vụ đầu tiên trong nhánh Có thể thấy là nhiệm vụ Có thể thấy Đôi tư ợng từ Gói chuyển động.

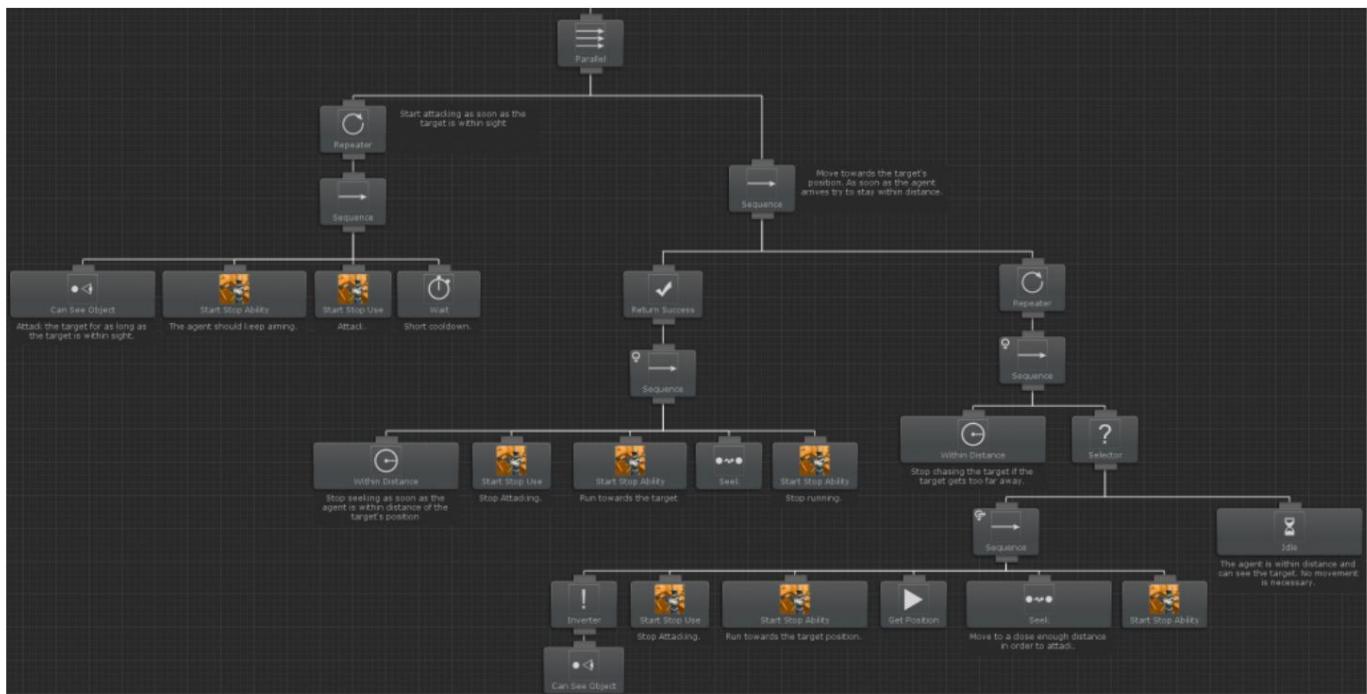
Can See Object sẽ sử dụng mặt nạ lớp để tìm kiếm bất kỳ đối tượng nào trong tầm nhìn. Người chơi đang ở trên lớp Nhân vật nên nhiệm vụ sẽ chỉ tìm kiếm các đối tượng nằm trên lớp đó. Khi nhiệm vụ tìm thấy người chơi, nhiệm vụ Is Alive sẽ thực hiện để xác định xem người chơi còn sống hay không.

Không cần phải tấn công một nhân vật đã chết. Sau đó, tác vụ Set Shared GameObject là một tác vụ trợ giúp sẽ chuyển giá trị Đôi tư ợng có thể thấy được tìm thấy sang biến Target. Điều này được thực hiện để khi các tác vụ khác hoạt động trên biến trình phát tìm thấy, đối tượng ban đầu sẽ không bị mất. Hai nhiệm vụ Set Bool sau đó được thực hiện, điều này cho thấy rằng đặc vụ sẽ đi lang thang nếu người chơi bị lạc và vị trí của mục tiêu phải được cập nhật. Nhánh bên phải bên dưới nhiệm vụ Song song ban đầu sau đó sẽ đặt vị trí của mục tiêu thành biến Vị trí cuối cùng.



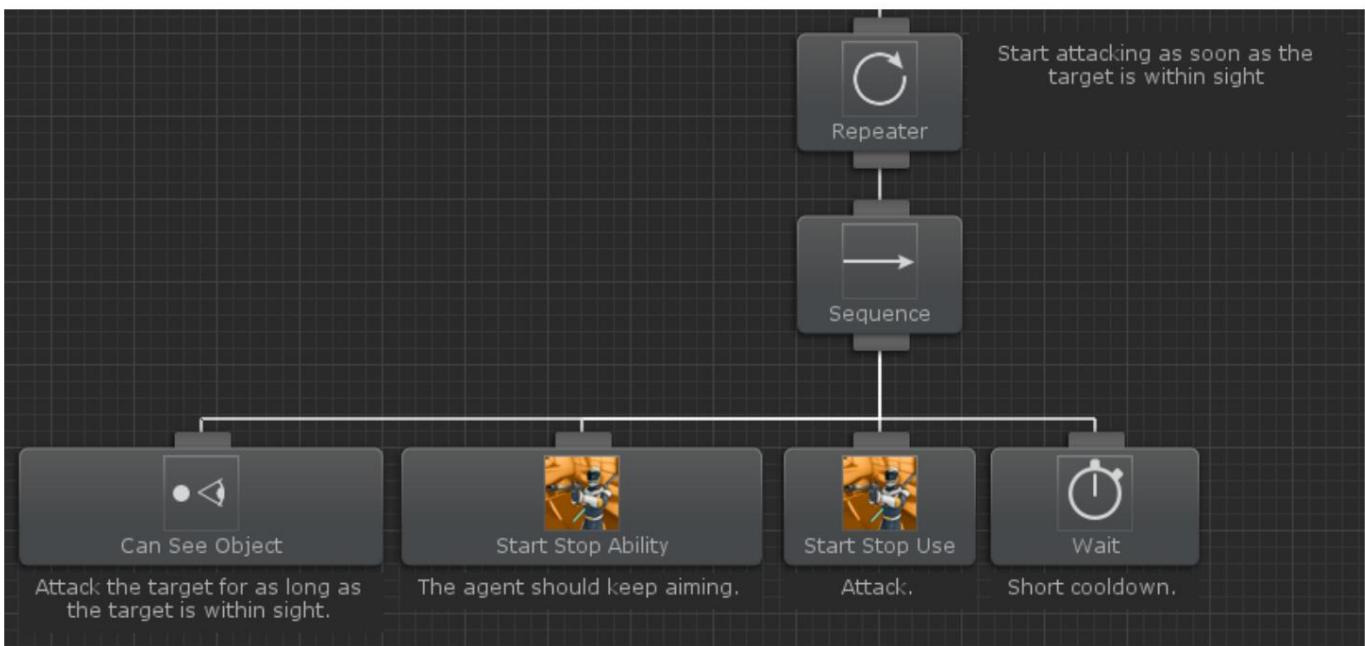
Sau đó, một tác vụ Bộ chọn đư ợc chạy, nhiệm vụ này đầu tiên sẽ đánh giá nhánh bên trái.

Mục tiêu của nhánh trái là tấn công mục tiêu chừng nào mục tiêu còn trong tầm ngắm. Để thực hiện điều này, một tác vụ Trình tự đư ợc sử dụng với Tự hủy bỏ có điều kiện. Hủy bỏ tự có điều kiện là một loại hủy bỏ mới và loại hủy bỏ này sẽ đánh giá lại tác vụ Có điều kiện miễn là nhánh hiện tại đang hoạt động. Trong trường hợp này, nhánh hiện tại chưa nhiệm vụ Is Alive cùng với một Selector, do đó nó sẽ đánh giá lại Is Alive miễn là đặc vụ có thể nhìn thấy mục tiêu. Nhiệm vụ Is Alive sẽ thất bại khi mục tiêu không còn sống.

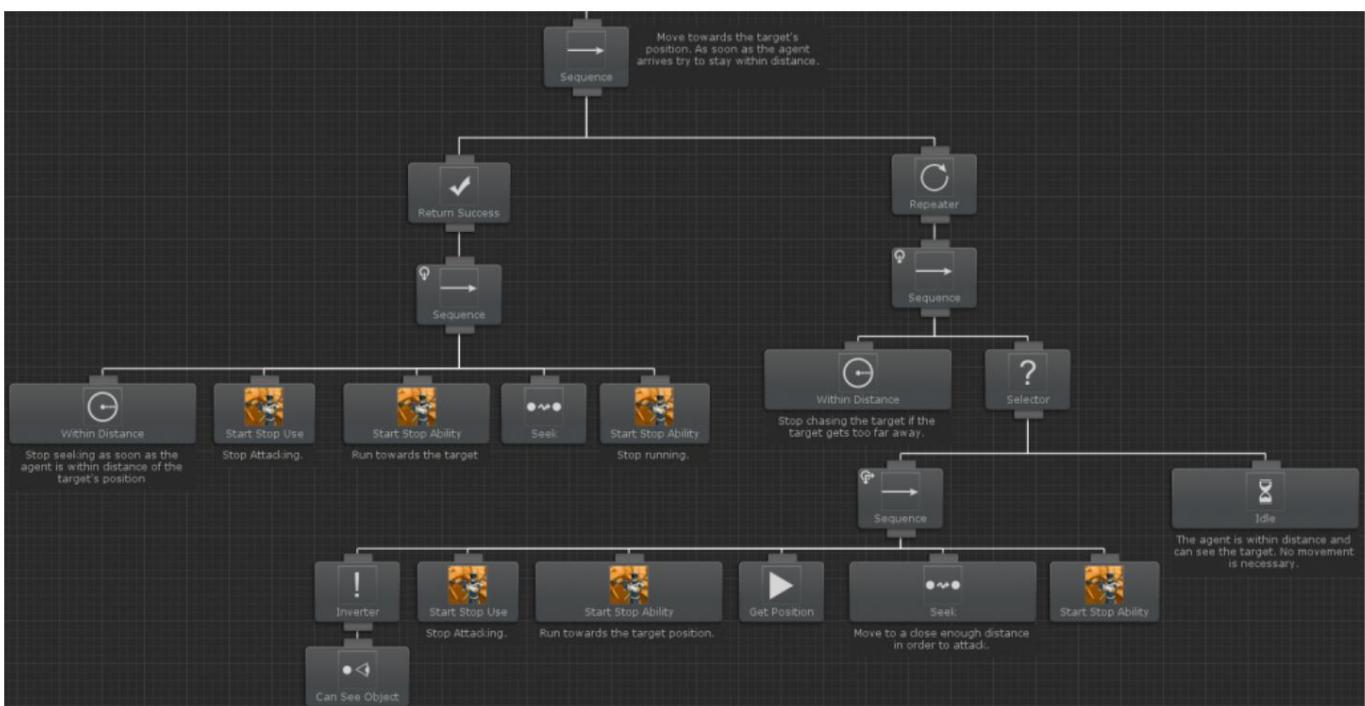


Nhánh đầu tiên mà nhiệm vụ Bộ chọn sẽ đánh giá có một nhiệm vụ Song song là cha. Tác vụ song song này chạy hai nhánh:

- Nhánh trái tấn công nếu mục tiêu trong tầm nhìn
- Chi nhánh bên phải sẽ giữ cho đại lý ở gần mục tiêu

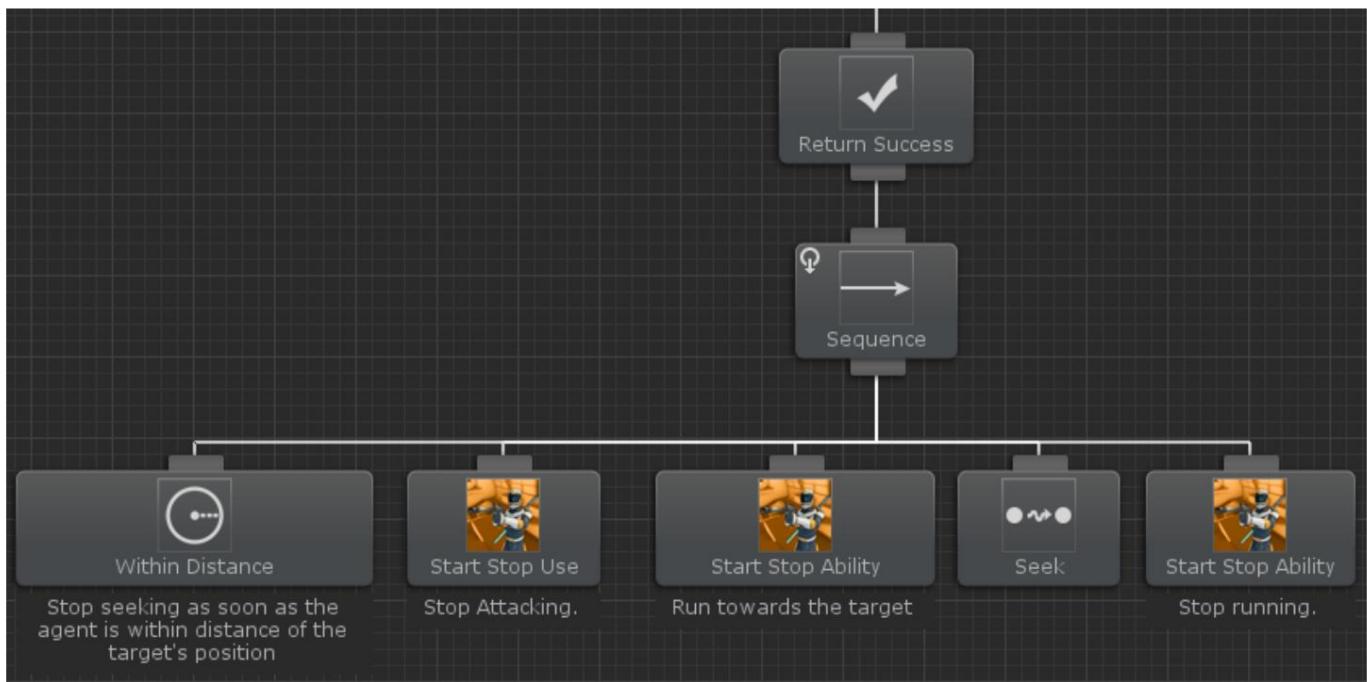


Nhánh Tấn công bên trái sử dụng một nhiệm vụ Có thể Xem Đôi tư ợng khác để xác định xem tác nhân có ở gần mục tiêu hay không. Nhiệm vụ có thể nhìn thấy đôi tư ợng này khác với nhiệm vụ có thể nhìn thấy đôi tư ợng đầu tiên vì nó có độ lớn khoảng cách nhỏ hơn nên tác nhân sẽ chỉ tấn công khi ở gần mục tiêu. Nếu mục tiêu ở gần Start Stop Khả năng nhiệm vụ sẽ thực thi để giữ cho nhân vật nhắm mục tiêu. Sau khi nhân viên nhắm mục tiêu, nhiệm vụ Start Stop Use sẽ thực hiện, nhiệm vụ sẽ thực sự sử dụng vật phẩm đó, sẽ bắn súng trù ờng tấn công hoặc vung kiếm. Tác vụ Wait sau đó đư ợc sử dụng để ngăn tác nhân cố gắng tấn công mọi khung hình.

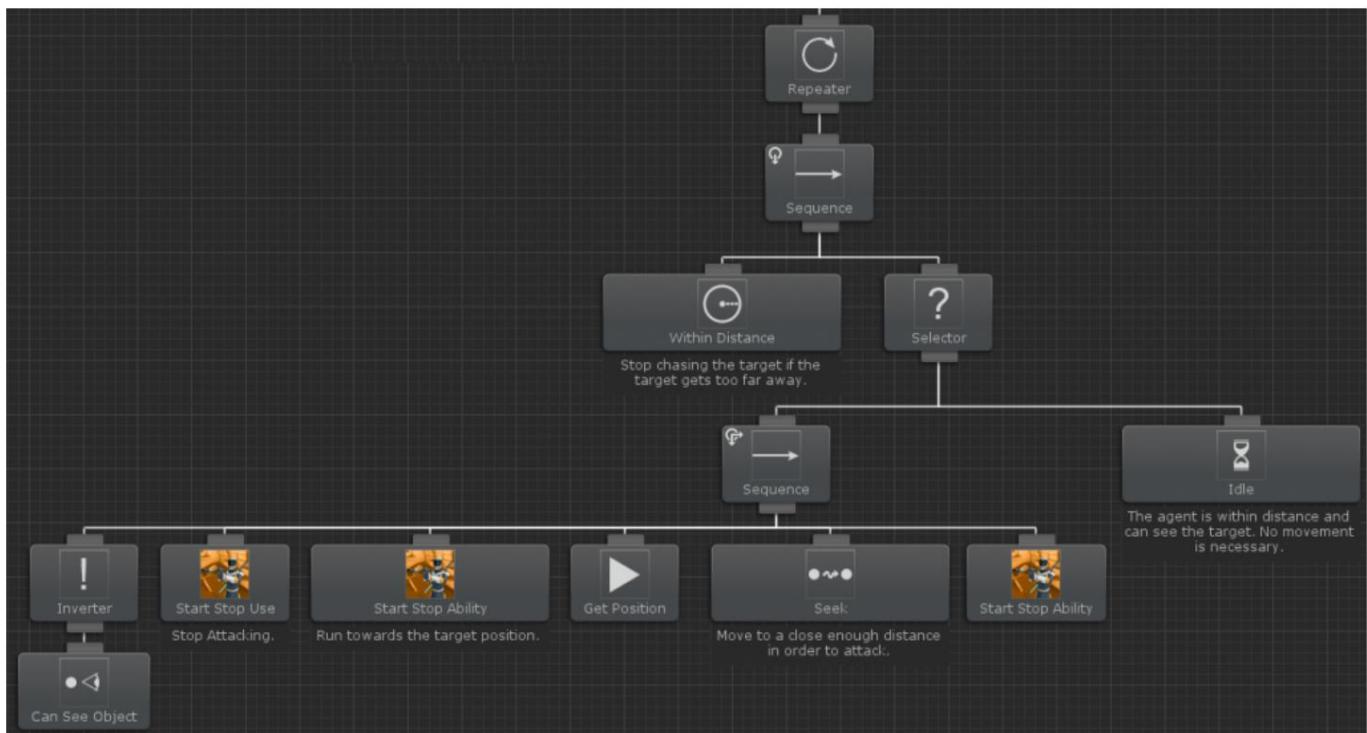


Trong khi nhánh Tấn công đang chạy, nhánh Tiến tới cũng sẽ chạy. Nhánh này bao gồm hai phần:

- Di chuyển vào vị trí sau khi đạt được mục tiêu đầu tiên.
- Giữ nguyên vị trí sau khi vào vị trí.



Để di chuyển vào vị trí, nhiệm vụ Trong Khoảng cách được sử dụng để xác định xem tác nhân có ở trong khoảng cách của mục tiêu hay không. Tự hủy bỏ có điều kiện được sử dụng vì vậy nhiệm vụ Trong khoảng cách sẽ đánh giá lại miễn là bất kỳ nhiệm vụ nào trong nhánh hiện tại đang hoạt động, trong trường hợp này nó sẽ là nhiệm vụ Tìm kiếm. Ngay sau khi Seek di chuyển tác nhân đến gần mục tiêu, nhiệm vụ Trong Khoảng cách sẽ trả về thành công và Biến tần sẽ thay đổi thành công đó thành thất bại, do đó nhánh sẽ ngừng thực hiện. Trong khi nhiệm vụ Tìm kiếm đang hoạt động, khả năng Thay đổi Tốc độ sẽ được kích hoạt để đặc vụ sẽ chạy về phía mục tiêu.

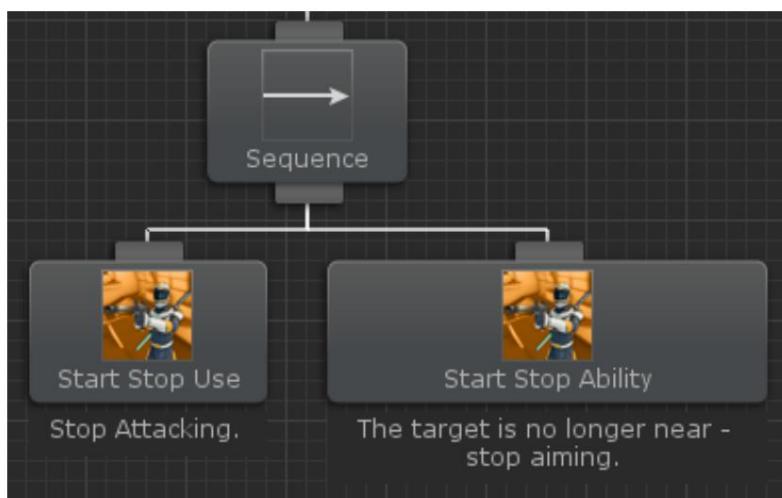


Sau khi đạt lý di chuyển đến được trong khoảng cách với mục tiêu, nhánh bên phải sẽ giữ đạt lý ở gần mục tiêu. Có ba điểm khác biệt chính giữa nhánh này và nhánh Tiên tới trước đó:

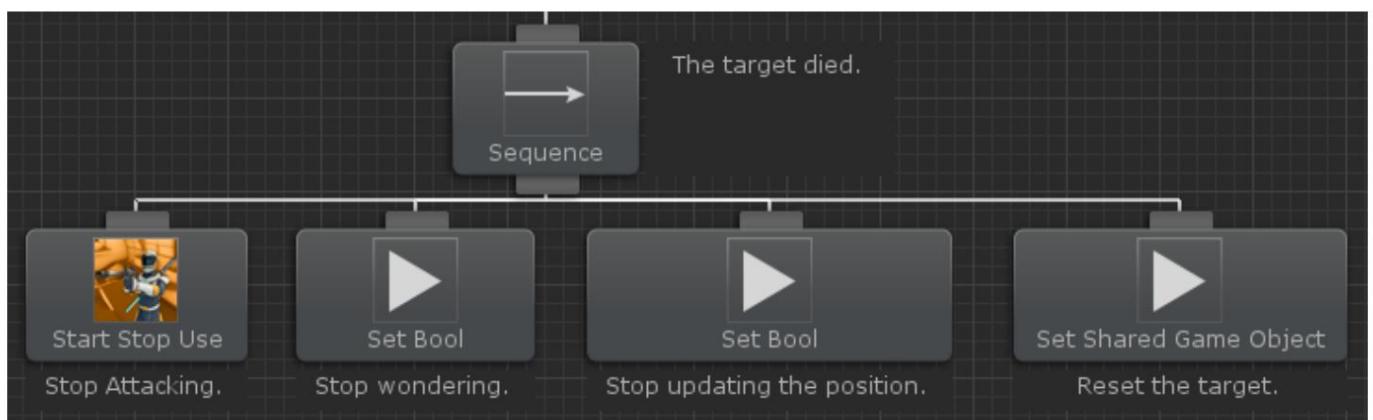
- Nhánh hiện tại sẽ có nhân vật đi bộ thay vì chạy trong khi tìm kiếm mục tiêu.
- Một nhiệm vụ Có thể Nhìn thấy Đối tượng được sử dụng cùng với Trong Khoảng cách, do đó tác nhân sẽ luôn

ở trong tầm nhìn của mục tiêu trong khi tấn công. Như một ví dụ, mục tiêu có thể di chuyển bên trong một cấu trúc, điều này có thể khiến tác nhân không nhìn thấy mục tiêu mặc dù mục tiêu có thể vẫn ở gần tác nhân.

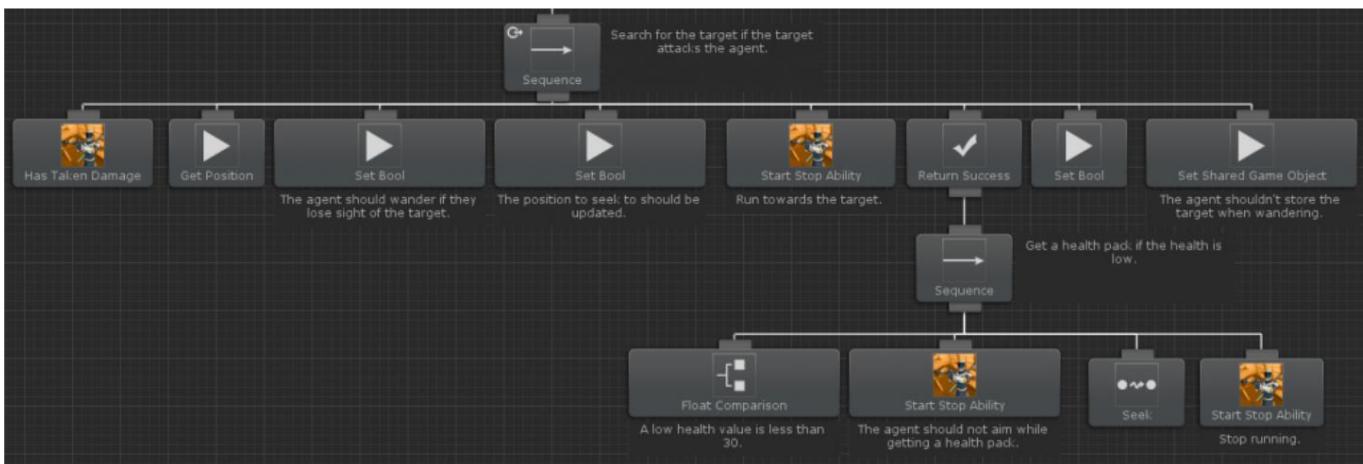
- Nhiệm vụ nhàn rỗi được sử dụng nếu nhân viên không cần tìm kiếm ở bất kỳ vị trí nào. Nếu mục tiêu đang đứng yên và trong tầm nhìn thì đặc vụ có thể tiếp tục tấn công mà không cần phải thay đổi vị trí.



Tác nhân bây giờ sẽ tấn công và di chuyển về phía mục tiêu khi ở trong khoảng cách. Nếu mục tiêu rời xa tác nhân và di chuyển quá xa thì nhánh Chuyển hướng bên phải sẽ trả lại lỗi và thực hiện nhánh Đặt lại. Nhánh Reset này chạy hai Hành động sẽ đảm bảo tác nhân không còn tấn công và không còn nhắm mục tiêu.



Khi mục tiêu lần đầu tiên được lấy, các biến Vị trí Đi lang thang và Cập nhật đã được thiết lập để tác nhân sẽ tìm kiếm mục tiêu nếu mục tiêu bị mất. Nếu mục tiêu bị giết bởi tác nhân thì tác nhân không cần phải đi lang thang vì vậy nhánh Mục tiêu chết sẽ đặt lại tất cả các biến liên quan đến tấn công về mặc định.

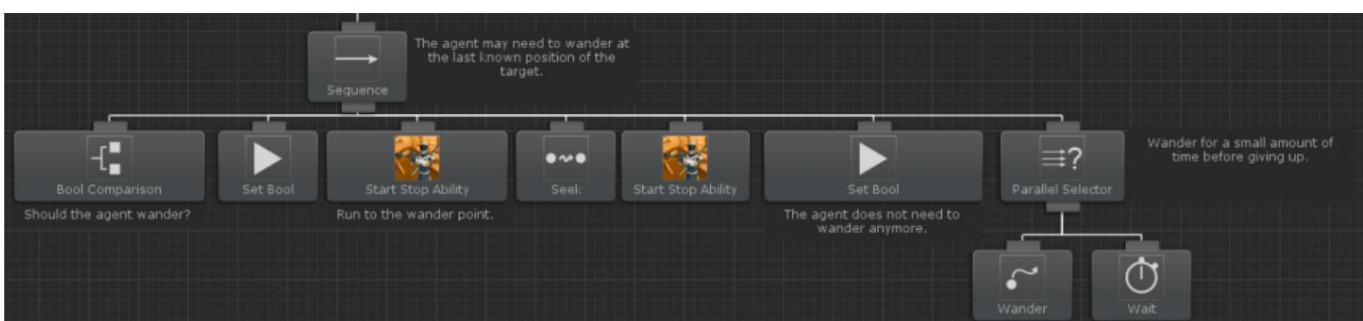


Nếu đặc vụ không cần lấy đạn và ngửi mùi chơi không có trong tầm nhìn thì nhánh ưu tiên cao nhất tiếp theo sẽ chạy. Nhánh này sử dụng hệ thống sự kiện của Ultimate Character Controller để xác định xem đặc vụ có bị sát thương hay không. Nếu đặc vụ đã bị thiệt hại thì nhiệm vụ Đã nhận Thiết hại sẽ thành công trở lại và chi nhánh sẽ thực hiện. Khi đặc vụ đã gây sát thương, đầu tiên nhánh sẽ lấy vị trí của đối tượng mà nó gây sát thương và đặt Wander thành true. Trước khi tác nhân di chuyển về phía mục tiêu, đầu tiên nó sẽ xác định xem lưu lượng máu của họ có thấp đến mức nghiêm trọng hay không. Nếu tình trạng sức khỏe thấp nghiêm trọng thì nhân viên sẽ tìm kiếm một gói sức khỏe.

Sau khi tác nhân lấy được một gói sức khỏe hoặc xác định rằng không cần thêm sức khỏe, nhánh sẽ kết thúc. Hãy nhớ rằng biến Wander đã được đặt thành true nên nhánh Wander sẽ bắt đầu thực thi nếu không có nhánh nào có mức độ ưu tiên cao hơn.



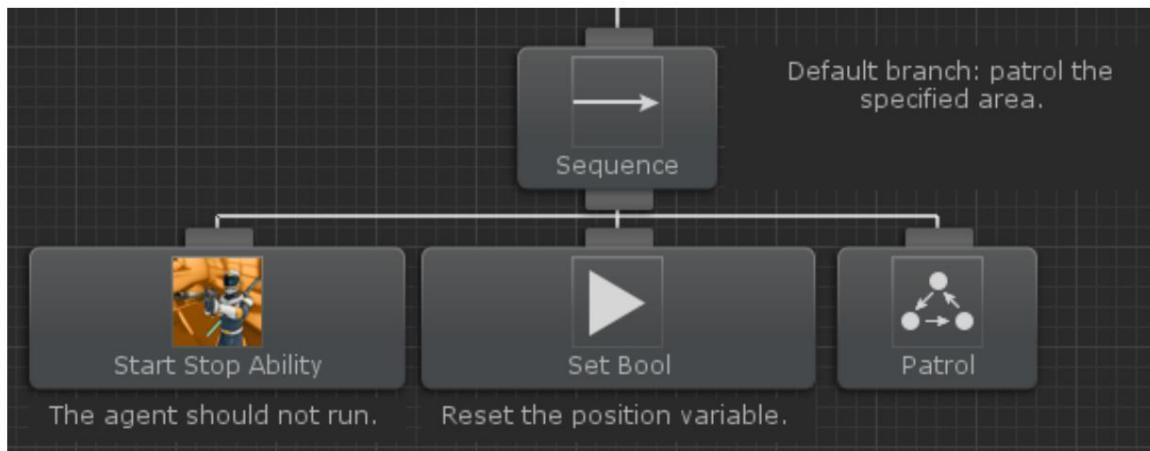
Nhánh tiếp theo có mức ưu tiên cao nhất là nhánh Can Hear. Nhánh này sử dụng nhiệm vụ Có thể nghe Đôi tai từ Gói Chuyển động để xác định xem có nguồn âm thanh nào được phát ra âm thanh hay không. Nếu bất kỳ âm thanh nào được nghe thấy thì tác nhân sẽ chạy về vị trí nguồn âm thanh bằng cách bắt đầu tác vụ Thay đổi tốc độ và sử dụng tác vụ Tìm kiếm để di chuyển vào vị trí. Biến Wander cũng được đặt thành true nên đặc vụ sẽ đi lang thang nếu không thể nhìn thấy mục tiêu vào thời điểm đặc vụ đến địa điểm Seek.



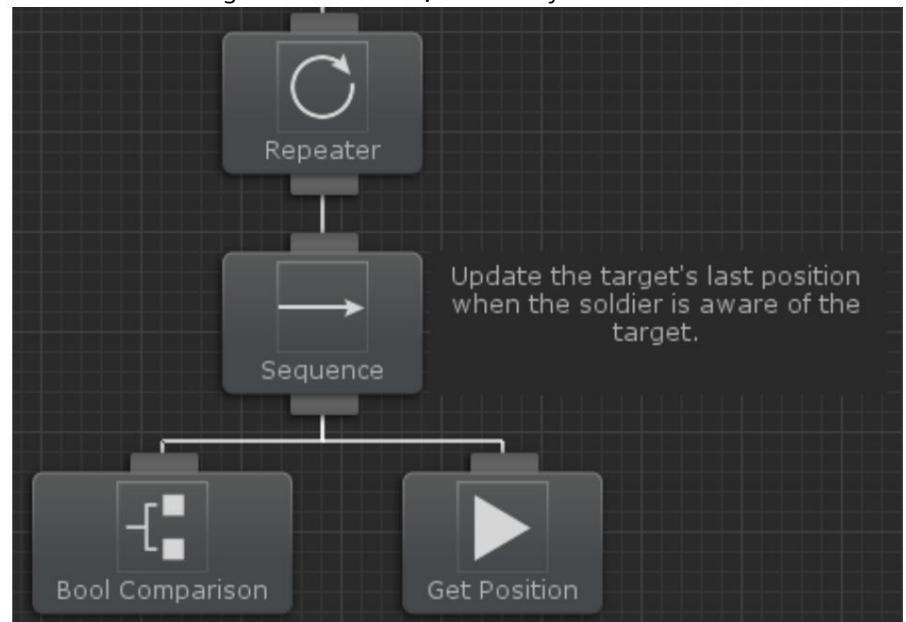
Trong suốt quá trình thực thi cây, đã có nhiều trường hợp nhiệm vụ Wander được đặt thành true. Nhiệm vụ so sánh Bool sẽ so sánh biến nhiệm vụ Wander này thành true với | 61

xác định xem tác nhân có nên đi lang thang hay không. Trước khi đặc vụ có thể đi lang thang, trước tiên họ phải tìm đến Vị trí Cuối cùng. Biến này sẽ được đặt bất cứ khi nào biến Vị trí Cập nhật là true và nó sẽ cho biết vị trí cuối cùng mà mục tiêu được đặt. Khi đặc vụ đến Vị trí cuối cùng, nhiệm vụ Đi lang thang sẽ thực hiện, cho phép đặc vụ tìm kiếm mục tiêu.

Hãy nhớ rằng trong thời gian này, các nhánh có mức độ ưu tiên cao hơn đang được đánh giá lại nên nếu người chơi trong tầm nhìn của tác nhân bất kỳ lúc nào thì nhánh Có thể xem sẽ hủy bỏ nhánh hiện tại để nhân vật có thể bắt đầu tấn công. Nhiệm vụ Wander được gắn với nhiệm vụ Bộ chọn song song sẽ thực hiện tất cả các con cho đến khi một con trở lại thành công. Ở bên phải của nhiệm vụ Wander là nhiệm vụ Chờ sẽ trả lại thành công sau một khoảng thời gian định trước. Điều này sẽ ngăn đặc vụ lang thang mãi mãi và giới hạn lựợng thời gian tác nhân lang thang.



Nếu không có nhánh nào khác cần thực hiện thì cây sẽ rơi trở lại nhánh Tuần tra. Nhánh này sẽ di chuyển nhân vật giữa các điểm tuần tra và giữ cho nhân vật di chuyển cho đến khi



nhánh ưu tiên cao hơn hủy bỏ nó.

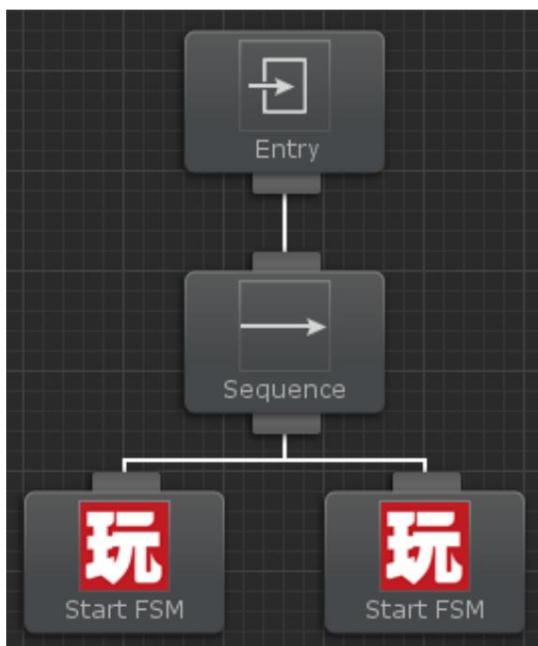
Khi các nhánh Đạn, Có thể Nhìn thấy, Gây sát thương, Có thể Nghe, Đi lang thang và Tuần tra được thực hiện gần ngọn cây thì nhánh Cập nhật Vị trí cũng sẽ thực hiện. Nhánh này sẽ cập nhật vị trí của mục tiêu khi vị trí cập nhật là đúng. Nhánh này được tách ra khỏi phần còn lại của cây bởi vì không chỉ một nhánh duy nhất có thể đặt Cập nhật Vị trí thành true, do đó để tránh phải thêm các tác vụ giống nhau nhiều lần, nhánh này đã được thêm vào gần trên cùng dưới Tác vụ song song.

Bây giờ bạn đã hoàn thành phần tổng quan này, bạn sẽ có đủ kiến thức để bắt đầu tạo cây của riêng bạn với Bộ điều khiển ký tự cuối cùng. Đảm bảo rằng bạn xem qua cây demo một vài lần để làm quen với trình chỉnh sửa trực quan của Behavior Designer.

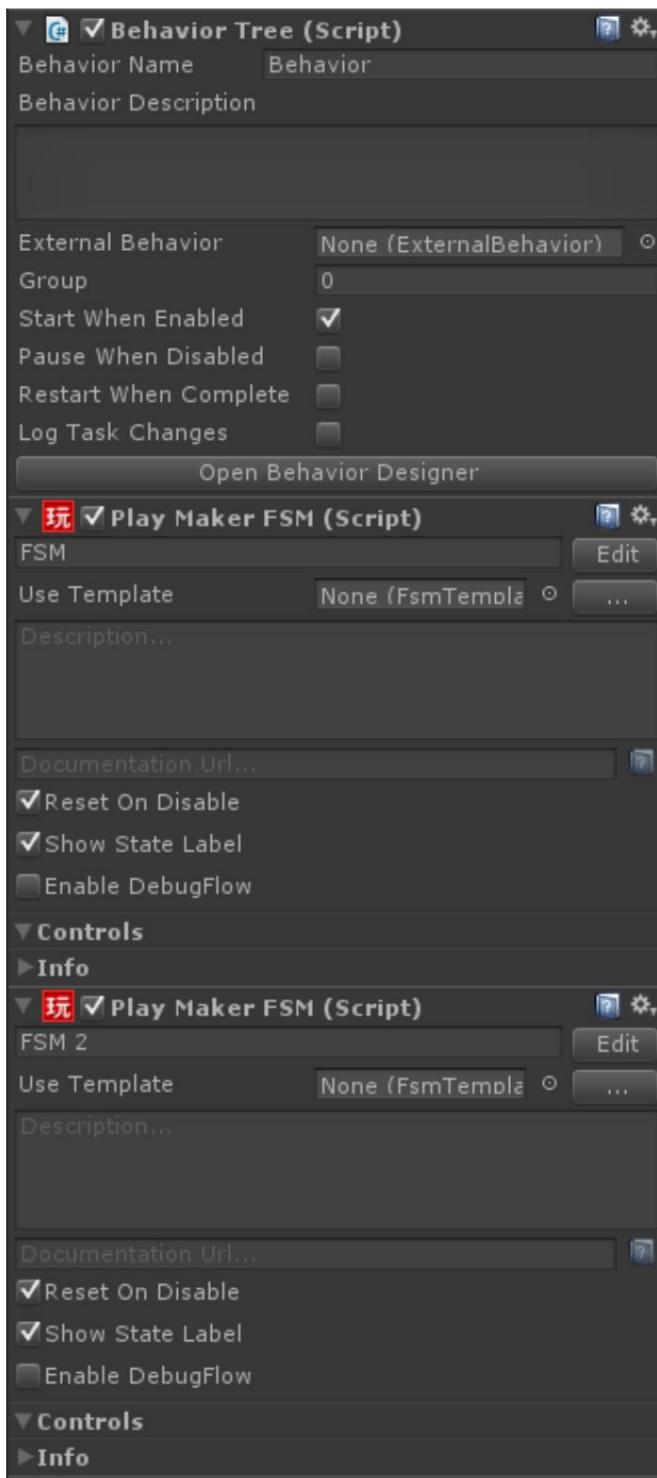
# Playmaker

[Playmaker](#) là một công cụ tạo kịch bản trực quan phổ biến cho phép bạn dễ dàng tạo các máy trạng thái hữu hạn. Behavior Designer tích hợp trực tiếp với PlayMaker bằng cách cho phép PlayMaker thực hiện hành động hoặc tác vụ có điều kiện, sau đó tiếp tục cây hành vi từ nơi nó đã dừng lại. Tệp tích hợp PlayMaker nằm trên [trang tải xuống bởi vì Playmaker](#) không bắt buộc để Behavior Designer hoạt động.

Để bắt đầu, trước tiên hãy đảm bảo rằng bạn đã cài đặt Playmaker và đã nhập gói tích hợp. Sau khi các tệp đó được nhập, bạn đã sẵn sàng để bắt đầu tạo cây hành vi với Playmaker. Để bắt đầu, hãy tạo một cây rất cơ bản với một nhiệm vụ tuần tự có hai nhiệm vụ con Bắt đầu FSM:



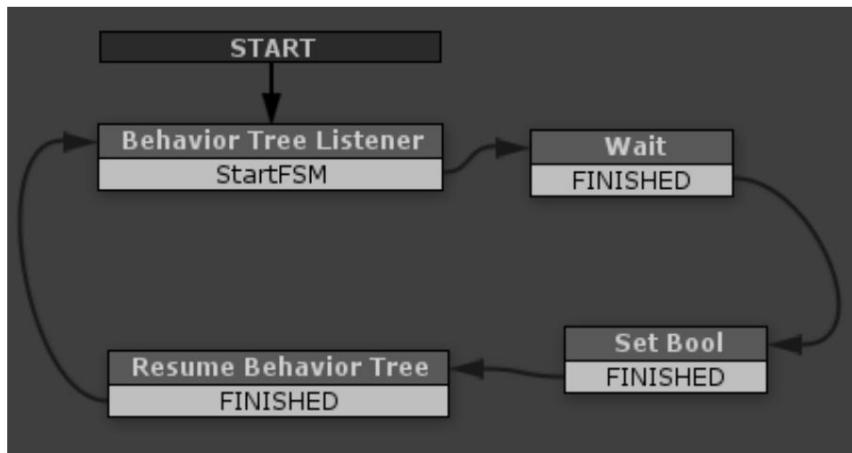
Tiếp theo, thêm hai thành phần Playmaker FSM vào cùng một đối tượng trò chơi mà bạn đã thêm cây hành vi vào.



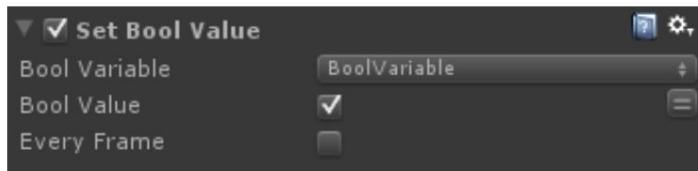
Mở Playmaker và bắt đầu tạo FSM mới. FSM này sẽ là một FSM đơn giản để chỉ ra cách Nhà thiết kế hành vi tương tác với Playmaker. Đối với FSM phức tạp hơn, hãy xem dự án mẫu Playmaker. Behavior Designer khởi động Playmaker FSM bằng cách gửi cho nó một sự kiện. Tạo sự kiện này bằng cách thêm một trạng thái mới được gọi là "Trình xử lý cây hành vi" và thêm một sự kiện toàn cầu mới có tên "StartFSM". Sự kiện phải mang tính toàn cầu nếu không Nhà thiết kế hành vi sẽ không bao giờ có thể khởi động FSM.



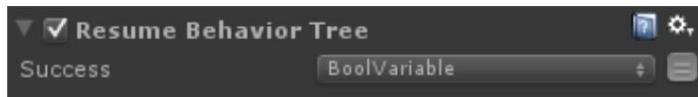
Thêm chuyển tiếp từ sự kiện đó cùng với trạng thái chờ, trạng thái bool đã đặt và trạng thái cây hành vi tiếp tục. Đảm bảo rằng bạn chuyển từ trạng thái Sơ đồ hành vi sang trạng thái Trình xử lý cây hành vi để FSM có thể được khởi động lại từ Trình thiết kế hành vi.



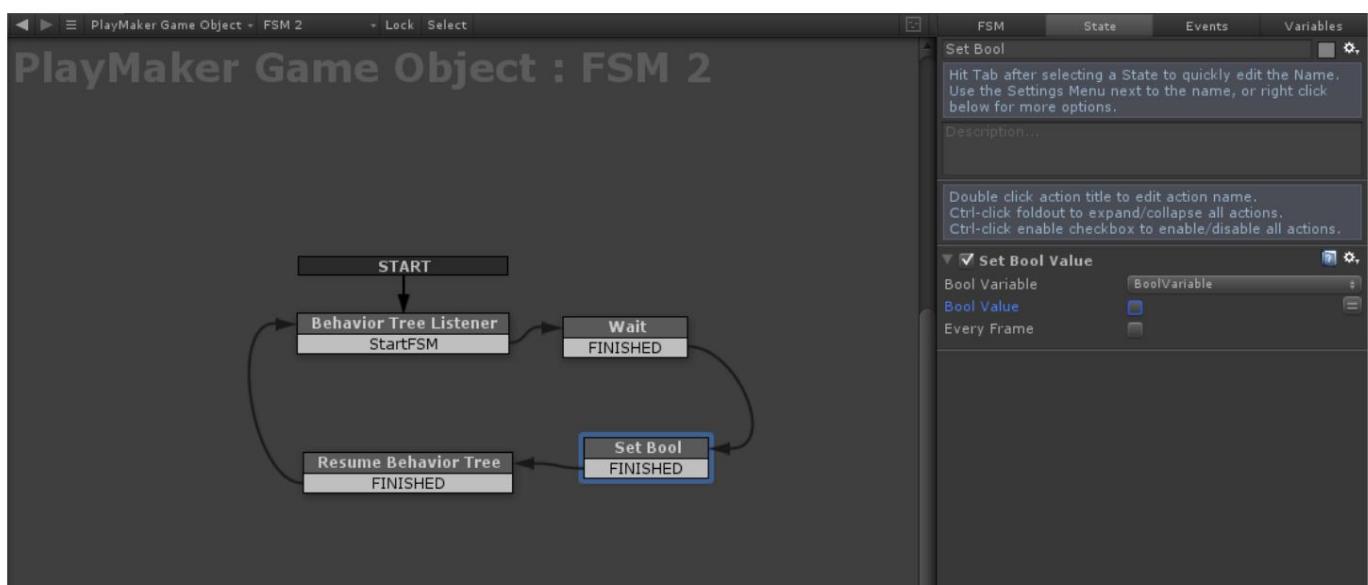
Tạo một biến mới trong trạng thái Đặt Bool và đặt giá trị đó thành true.



Sau đó, trong trạng thái Cây hành vi tiếp tục, chúng tôi muốn trả lại thành công dựa trên giá trị bool đó:

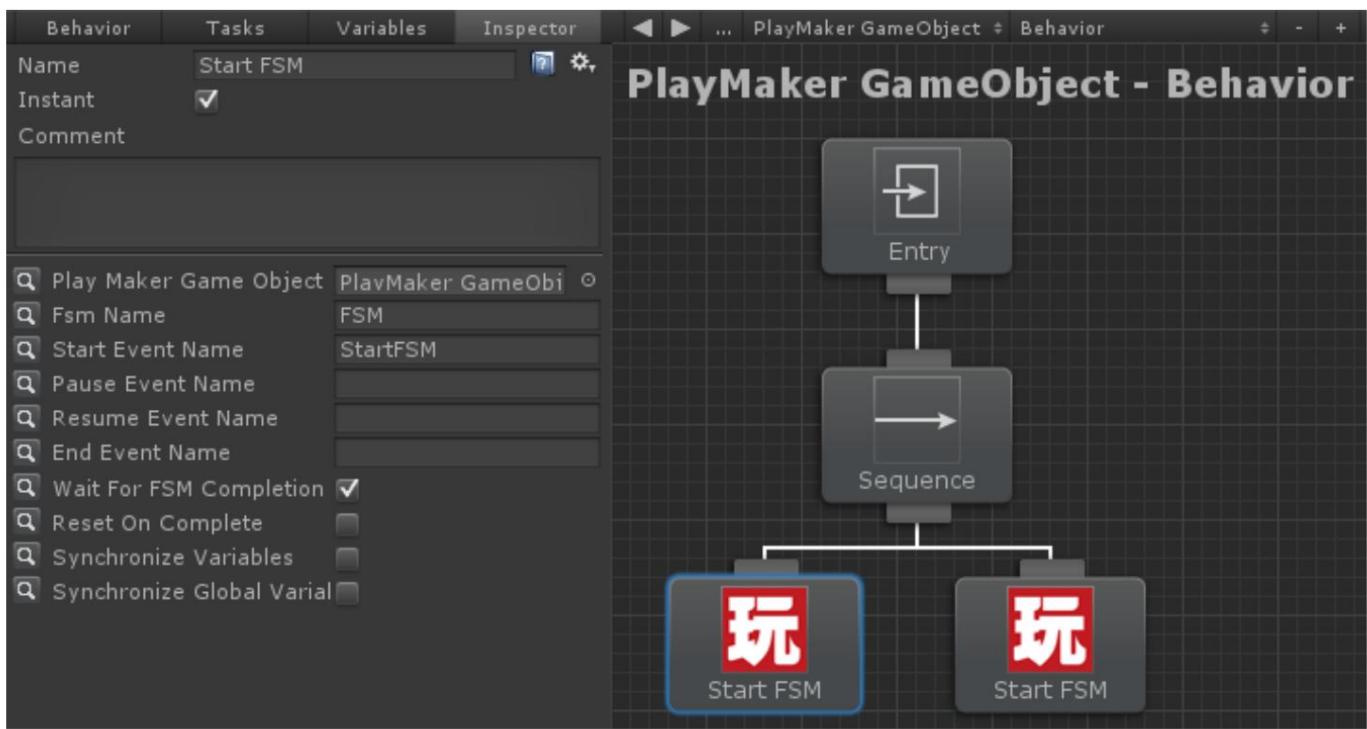


Đó là nó cho FSM này. Tạo các trạng thái và biến tương tự cho FSM thứ hai mà chúng ta đã tạo trước đó. Không đặt biến bool thành true cho FSM này.

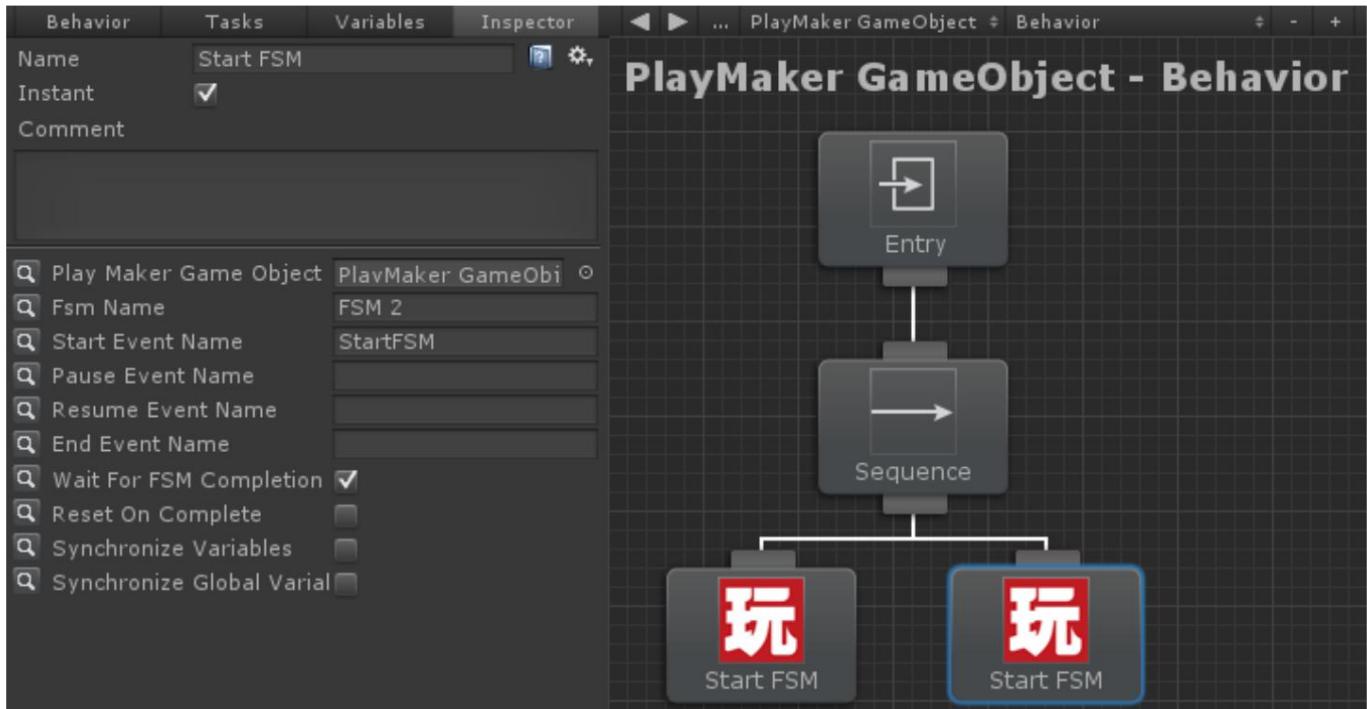


Bây giờ chúng tôi đã hoàn thành công việc trong Playmaker. Mở cây hành vi của bạn sao lưu trong Behavior Designer. Chọn nhiệm vụ Playmaker bên trái và bắt đầu gán giá trị cho các biến. Đối tượng trò chơi Playmaker được chỉ định cho đối tượng trò chơi mà chúng tôi đã thêm các thành phần Playmaker FSM vào. Tên FSM là tên của FSM Playmaker. Tên sự kiện là tên của | 65

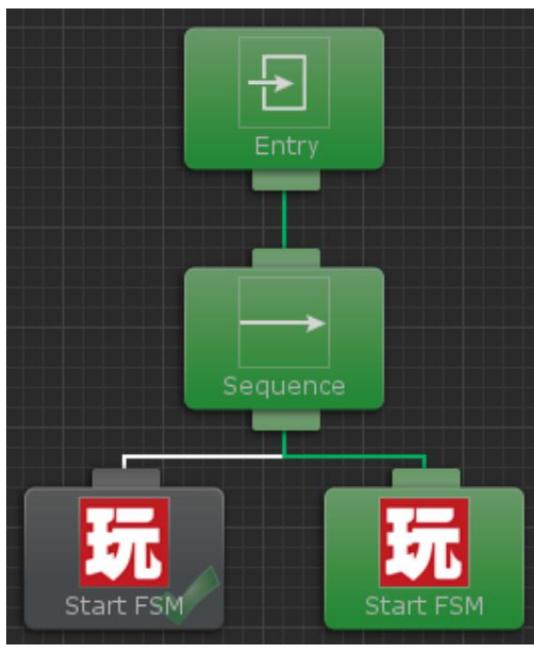
sự kiện toàn cầu mà chúng tôi đã tạo trong Playmaker.



Bây giờ chúng ta cần gán các giá trị cho nhiệm vụ Playmaker phù hợp. Các giá trị phải giống với tác vụ Playmaker bên trái ngoại trừ Tên FSM khác.



Đó là nó! Khi bạn nhấn chơi, bạn sẽ thấy nhiệm vụ Playmaker đầu tiên chạy trong một giây và sau đó nhiệm vụ Playmaker thứ hai sẽ bắt đầu chạy.

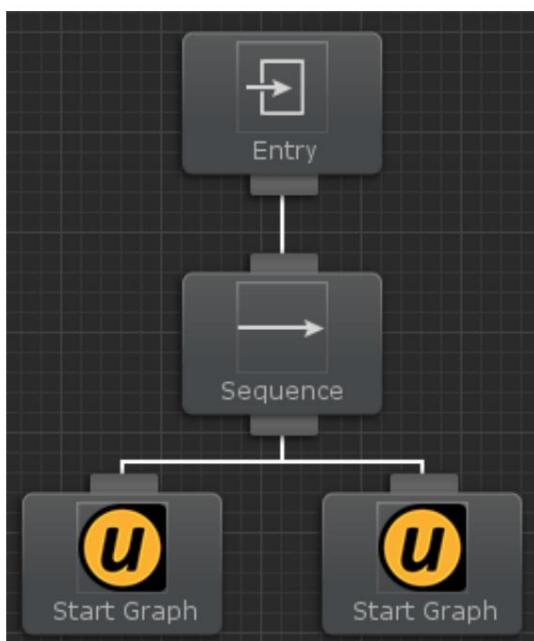


Nếu bạn hoán đổi các nhiệm vụ để tác vụ Playmaker thứ hai chạy trước Playmaker FSM đầu tiên, cây hành vi sẽ không bao giờ đến đư ợc với Playmaker FSM đầu tiên vì Playmaker FSM thứ hai trả lại lỗi và tác vụ trình tự ngừng thực hiện các con của nó.

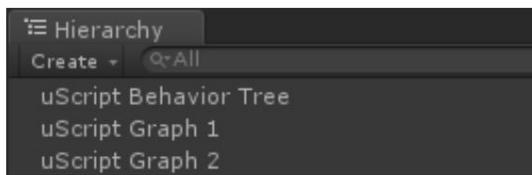
## uScript

[uScript](#) là một công cụ tạo kịch bản trực quan cho phép bạn tạo các thiết lập phức tạp mà không cần viết một dòng mã nào. Behavior Designer tích hợp trực tiếp với uScript bằng cách cho phép uScript thực hiện các tác vụ hành động hoặc các tác vụ có điều kiện và sau đó tiếp tục cây hành vi từ nơi nó đã dừng lại. Các tệp tích hợp uScript nằm trên [trang tải xuống](#) bởi vì uScript không bắt buộc để Behavior Designer hoạt động.

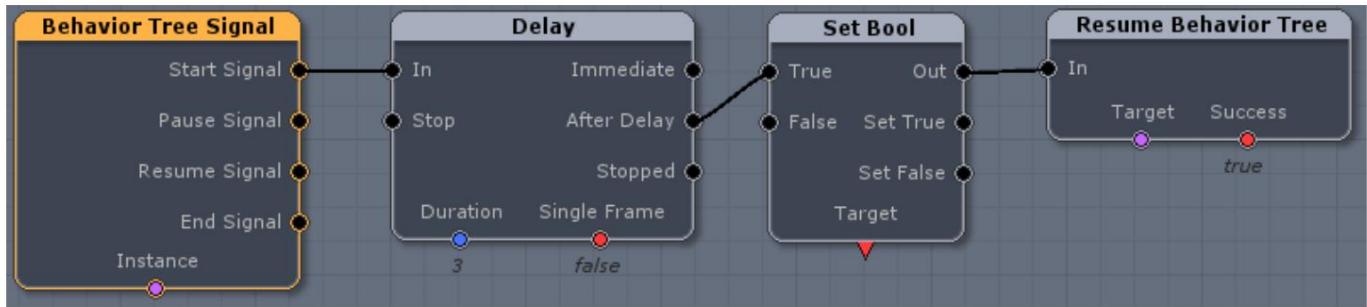
Để bắt đầu, trước tiên hãy đảm bảo rằng bạn đã cài đặt uScript và đã nhập gói tích hợp. Sau khi các tệp đó đư ợc nhập, bạn đã sẵn sàng để bắt đầu tạo cây hành vi với uScript. Để bắt đầu, hãy tạo một cây rất cơ bản với một nhiệm vụ tuần tự có hai nhiệm vụ con Start Graph:



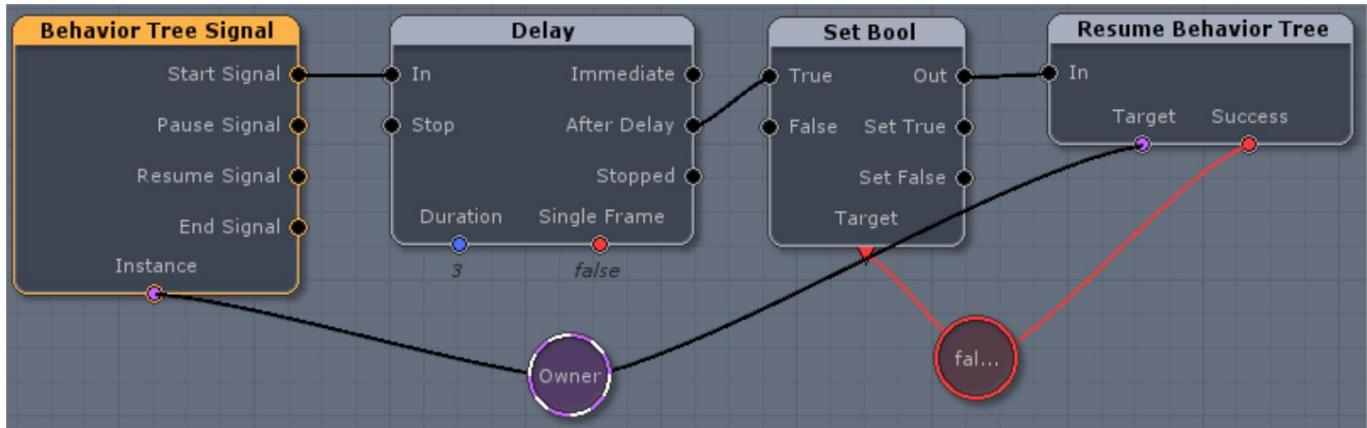
Bây giờ chúng ta cần tạo hai GameObjects sẽ chứa đồ thị uScript đã biên dịch:



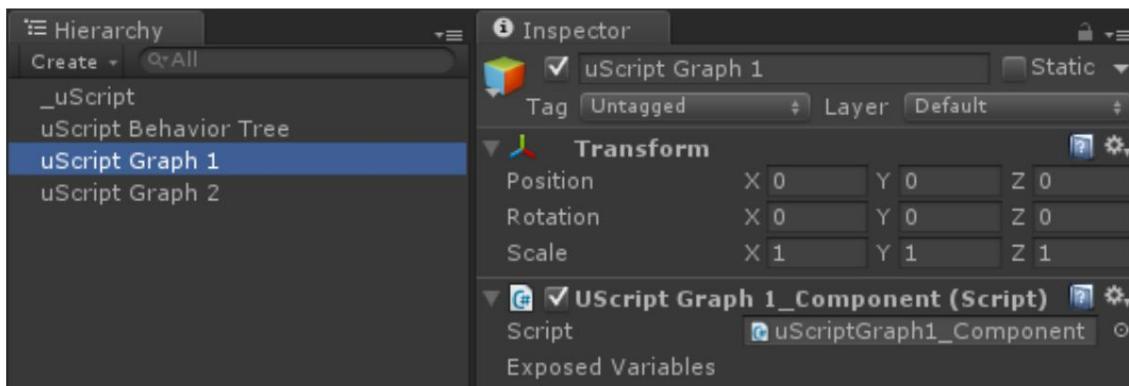
Mở uScript và bắt đầu tạo một đồ thị mới. Thêm nút Tín hiệu cây hành vi, nằm trong Sự kiện / Tín hiệu. Khi Behavior Designer muốn bắt đầu thực thi một biểu đồ uScript, nó sẽ bắt đầu từ nút này. Nút này chứa bốn sự kiện: Tín hiệu Bắt đầu, Tín hiệu Tạm dừng, Tín hiệu Tiếp tục và Tín hiệu Kết thúc. Tín hiệu Bắt đầu đư ợc sử dụng khi tác vụ cây hành vi bắt đầu chạy. Tín hiệu tạm dừng đư ợc gọi khi cây hành vi bị tạm dừng và Tín hiệu tiếp tục đư ợc gọi khi cây hành vi tiếp tục bị tạm dừng. Cuối cùng, End Signal đư ợc gọi khi tác vụ uScript kết thúc. Đối với biểu đồ của chúng tôi, chúng tôi sẽ chỉ tạo một vài nút, dự án mẫu uScript hiển thị một biểu đồ uScript phức tạp hơn. Tạo một nút có độ trễ 3 giây, đặt bool, sau đó tiếp tục cây hành vi. Nút Cây hành vi Resume nằm trong Actions / Behavior Designer:



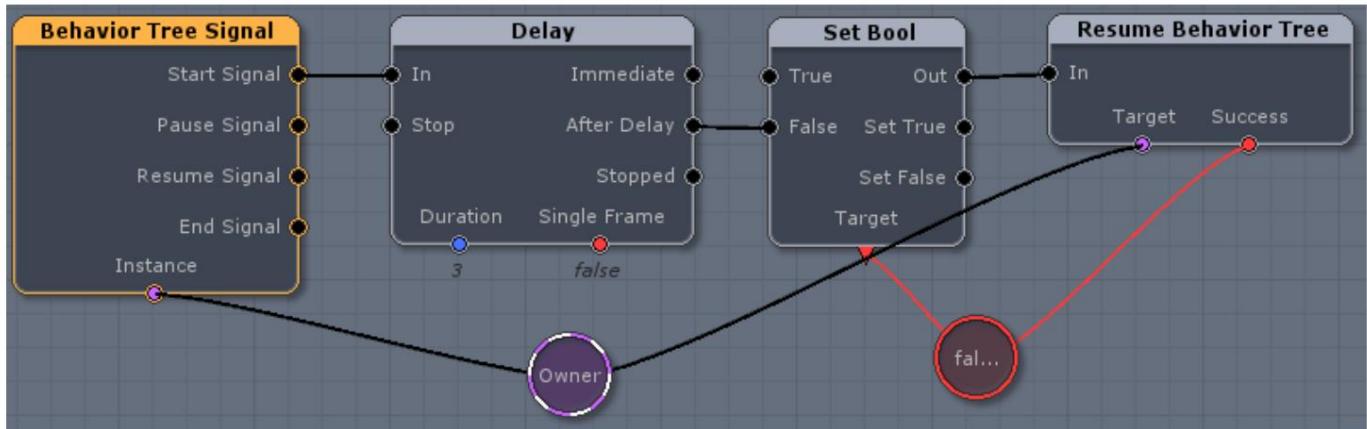
Bây giờ chúng ta cần tạo một Owner GameObject và biến bool.



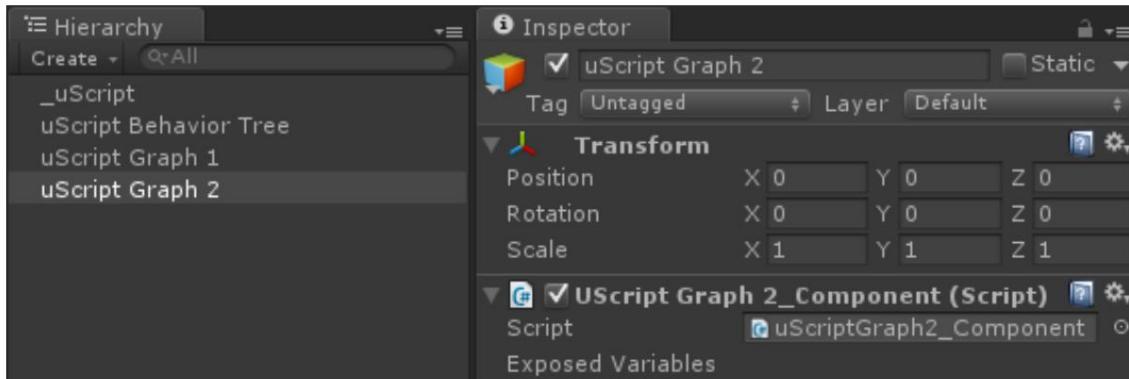
Lưu đồ thị uScript và gán thành phần cho GameObject đồ thị uScript đầu tiên của bạn. Trả lời không nếu uScript hỏi bạn có muốn gán thành phần cho GameObject chính hay không.



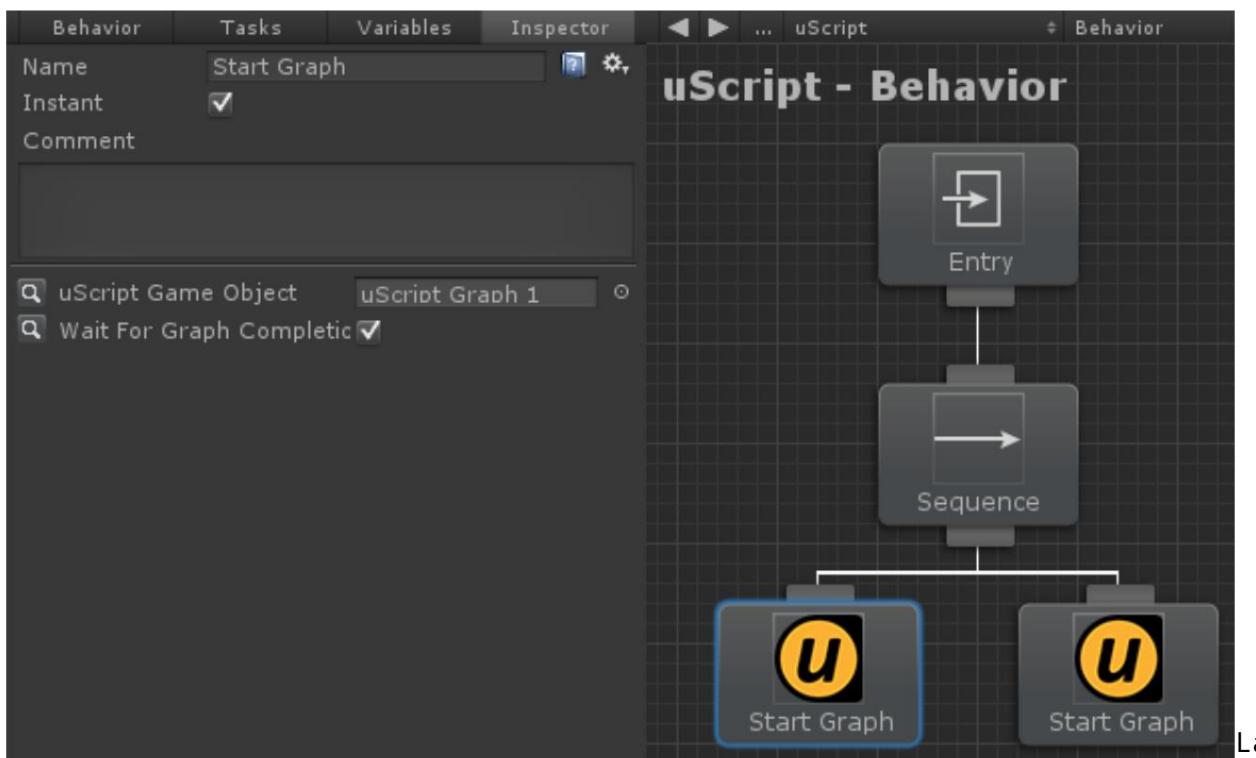
Tạo thêm một biểu đồ uScript. Làm cho nó giống như biểu đồ cuối cùng ngoại trừ đặt bool thành false:



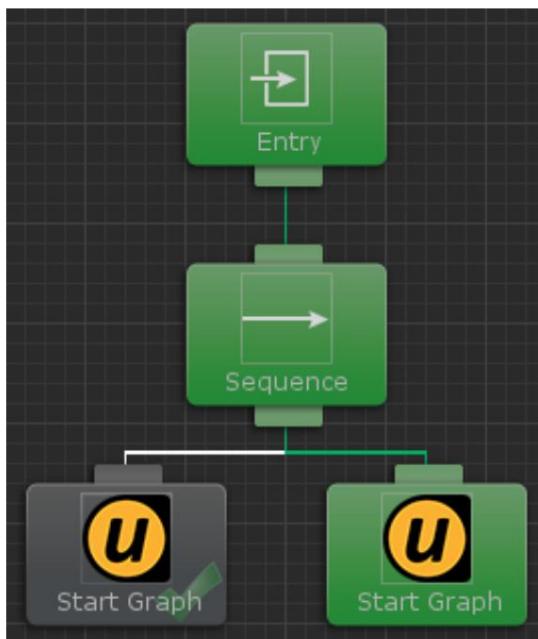
Cuối cùng lưu biểu đồ đó và gán thành phần cho GameObject uScript thứ hai:



Chúng tôi sắp hoàn thành. Điều duy nhất cần làm là gán đúng GameObject uScript cho các tác vụ trong Behavior Designer. Mở lại cây hành vi của bạn trong Behavior Designer. Nhập vào tác vụ uScript bên trái và gán GameObject uScript cho GameObject đồ thị uScript đầu tiên của bạn.



tự ứng tự cho tác vụ uScript bên phải, chỉ gán GameObject uScript cho GameObject đồ thị uScript thứ hai của bạn. Đó là nó! Khi nhấn play, bạn sẽ thấy tác vụ uScript đầu tiên chạy trong ba giây, tiếp theo là tác vụ uScript thứ hai.



Nếu bạn hoán đổi các tác vụ để biểu đồ uScript thứ hai chạy trước biểu đồ uScript đầu tiên, cây hành vi sẽ không bao giờ đến được biểu đồ uScript đầu tiên vì biểu đồ uScript thứ hai trả về lỗi và tác vụ trình tự ngừng thực hiện các tác vụ con của nó.

## Video

Các video sau đây sẽ mô tả cách sử dụng Behavior Designer: