代理人、目标和行为

在很多类型的游戏中，实体实时移动，具有一定程度的自主性。例如，游戏可能允许玩家点击或点击将角色移动到所需位置。角色将自动绕过障碍物到达该位置。敌人角色可能会移动攻击，计划在当前位置之前拦截玩家角色。盟友角色可能会与玩家角色成队飞行，保持与玩家角色的距离和平行的飞行方向。在所有这些情况下，每个实体的运动也可能被限制在现实的限制下，这样游戏角色仍然可以现实地移动和改变方向。

GameplayKit中的代理系统提供了一种有效实现自主移动的方法。*代理*代表一个游戏实体，该实体根据现实的约束在二维空间中移动：其大小、位置和速度，以及它对速度变化的抵抗力。（请注意，尽管代理的运动基于物理模型，但代理不受Sprite Kit物理模拟的约束。）代理人的行为由激励其运动的目标决定。

*目标*代表着一种独特的动机，随着时间的推移会影响代理人的运动。以下数字显示了一些目标的例子：

图7-1 A向特定位置移动的目标

图7-2 A避免障碍的目标

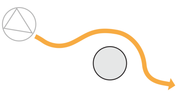
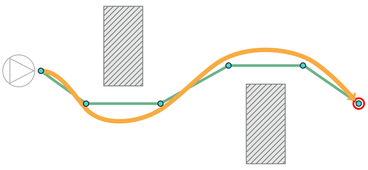


图7-3 A与其他特工一起聚集的目标

图7-4 A沿着点顺序走一条路的目标



一种*行为*将一个或多个目标联系在一起，以推动代理人的运动。例如，一种行为可能会将移动到目标点的目标与避免沿途障碍的目标结合起来。当您在行为中组合多个目标时，您可以为每个目标分配一个权重，以决定其相对影响。

为代理商设计你的游戏

GameplayKit中的代理架构扩展了实体和组件中讨论的实体组件架构——`GKAgent`对象是一个组件。在将游戏设计为每个移动角色使用实体后，您可以通过向每个相关实体添加`GKAgent`组件并使用指定`GKGoal`对象列表的`GKBehavior`对象来创建基于代理的行为。与任何其他组件一样，您在游戏循环中的每个模拟步骤或动画帧上调用`updateWithDeltaTime:`方法（直接或通过收集游戏中所有代理的`GKComponentSystem`对象）。

> 注
> 您也可以使用没有实体组件架构的代理。在这种情况下，您必须管理自己的一组`GKAgent`实例，并为每个代理调用`updateWithDeltaTime:`方法。

每个`GKGoal`对象代表一个基本、可重用的行为单位。当调用代理的`updateWithDeltaTime:`方法时，它会评估其关联的`GKBehavior`对象中的每个目标。结果生成一个矢量，该矢量表示实现该目标所需的代理旋转和速度的变化——或者更确切地说，在时间增量和代理的最大速度范围内朝着实现该目标的方向迈进。然后，代理组合这些矢量，每个矢量由`GKBehavior`对象定义的相应权重缩放，以获得组合调整，然后应用于代理的位置、方向和速度。有关可能目标的完整集合，请参阅 *GKGoal 类参考*。

`GKGoal`对象可以在多个行为中重用。例如，两个敌人的特工可能有着相同的目标，即移动拦截玩家特工，但其中一个可能会平衡这个目标和避开第三个特工。同样，一个`GKBehavior`对象可以被多个代理重用。例如，几个敌方特工可能具有相同的行为，即在躲避障碍的同时追捕玩家，并且每个人都会独立行动以实现目标组合，无论其当前状态如何。

在游戏中使用代理

AgentsCatalog示例代码项目包括几个SpriteKit场景，这些场景说明了具有一个或多个目标的代理的基本使用。

> 注
> 本节讨论示例代码项目*AgentsCatalog*的功能：*在GameplayKit中使用Agents System*。下载它，在Xcode中跟随。

例如，Fleeing场景（在`FleeingScene`类中找到）演示了如何使用代理和目标让一个游戏角色跟随鼠标或触摸位置，而另一个角色试图躲避第一个角色。这个效果是使用三个代理创建的：

- 一个“跟踪代理”，它没有屏幕上的表示形式，但其位置总是更新，以匹配鼠标点击/拖动事件（OS X）或触摸开始/移动事件（iOS）事件，如下所示：

```
- (void)touchesMoved:(nonnull NSSet<UITouch *> *)touches withEvent:(nullable
  UIEvent *)event {
    UITouch *touch = [touches anyObject];
    CGPoint position = [touch locationInNode:self];
    self.trackingAgent.position = (vector_float2){position.x, position.y};
}
```

- 一个“玩家特工”，其唯一目标是寻找跟踪特工的当前位置：

```
self.seekGoal = [GKGoal goalToSeekAgent:self.trackingAgent];
[self.player.agent.behavior setWeight:1 forGoal:self.seekGoal];
```

- 一个“敌人特工”，其唯一的目标是逃离玩家特工的位置：

```
self.fleeGoal = [GKGoal goalToFleeAgent:self.player.agent];
[self.enemy.agent.behavior setWeight:1 forGoal:self.fleeGoal];
```

The SpriteKit scene's `update:` method calls the `updateWithDeltaTime:` method of each agent. Each agent then updates its position and velocity to meet its goals, and calls the `agentDidUpdate:` method, which in turn updates the position and rotation of the SpriteKit node that serves as the agent's visual representation.

> 注
> 在本示例中，从未调用跟踪代理的`updateWithDeltaTime:`方法。由于该代理的位置始终由鼠标或触摸事件控制，GameplayKit代理模拟从来不需要移动它。由于代理没有可视化表示，因此不需要委托。

检查此示例代码项目中的其他场景，以演示其他类型的代理行为。例如，`FlockingScene`类演示了如何将分离、对齐和内聚目标结合起来，使一组代理一起移动。

代理和寻路

GameplayKit包含两个系统，用于帮助您管理游戏角色的移动：代理，在本章中讨论，以及寻路，在上一章中讨论（请参阅寻路）。每个系统以不同的方式管理运动。寻路涉及大规模描述游戏世界，以便您可以提前非常详细地规划路线，而代理模拟则关注狭隘的时间和空间，以在游戏过程中对动态情况做出反应。根据您想要实现的游戏功能，一个或另一个系统可能更合适，或者您可以将两者结合起来。

One common pattern is to use pathfinding to plan a route for a game character, then use the agent simulation to make the character follow that route—possibly while also considering other goals. The `goalToFollowPath:maxPredictionTime:forward:` goal motivates an agent to move from point to point along the path described by a `GKPath` object, and the `initWithGraphNodes:radius:` initializer conveniently creates a `GKPath` object from the result of a pathfinding operation.

DemoBots示例代码项目演示了这种模式。为了使用寻路，游戏首先创建一个`GKObstacleGraph`对象，表示游戏世界的不可通行区域，如上一章的清单6-3所示。然后，当一个敌人的机器人角色需要追逐另一个角色时，游戏的`TaskBotBehavior`类会通过图表找到一条路径（如上一章的清单6-4所示）。

> 注
> 本节讨论示例代码项目*DemoBots：使用SpriteKit和GameplayKit构建跨平台游戏*的功能。下载它，在Xcode中跟随。

Pathfinding results in an array of graph nodes—the `addFollowAndStayOnPathGoalsForPath` method shown in Listing 7-1 creates `GKGoal` objects from those nodes so that an agent can automatically follow the path.

清单7-1 使用寻路创建目标

```
private func addFollowAndStayOnPathGoalsForPath(path: GKPath) {
    // The "follow path" goal tries to keep the agent facing in a forward direction
  when it is on this path.
    setWeight(1.0, forGoal: GKGoal(toFollowPath: path, maxPredictionTime:
  GameplayConfiguration.TaskBot.maxPredictionTimeWhenFollowingPath, forward: true))

    // The "stay on path" goal tries to keep the agent on the path within the
  path's radius.
    setWeight(1.0, forGoal: GKGoal(toStayOnPath: path, maxPredictionTime:
  GameplayConfiguration.TaskBot.maxPredictionTimeWhenFollowingPath))
}
```

> 注
> 在DemoBots项目中，`GameplayConfiguration`对象是影响游戏性的全局常量的存储库。

除了寻路行为外，`TaskBotBehavior`类还创建其他目标，并将其添加到代表敌方机器人的代理中：

- The `behaviorAndPathPointsForAgent` method adds separation, alignment, and coherence goals to make groups of robots move together as a unit without overlapping one another.
- `addAvoidObstaclesGoalForScene`方法增加了一个目标，防止机器人试图通过关卡的不可通行区域。这个目标是必要的，因为代理通常不精确地遵循多边形路径——其`maxAcceleration`、`maxSpeed`和其他属性以及其他目标，导致代理遵循大致近似路径的自然、平滑的轨迹。避免障碍目标会影响代理人的行为，引导大致路径远离任何障碍。