寻路

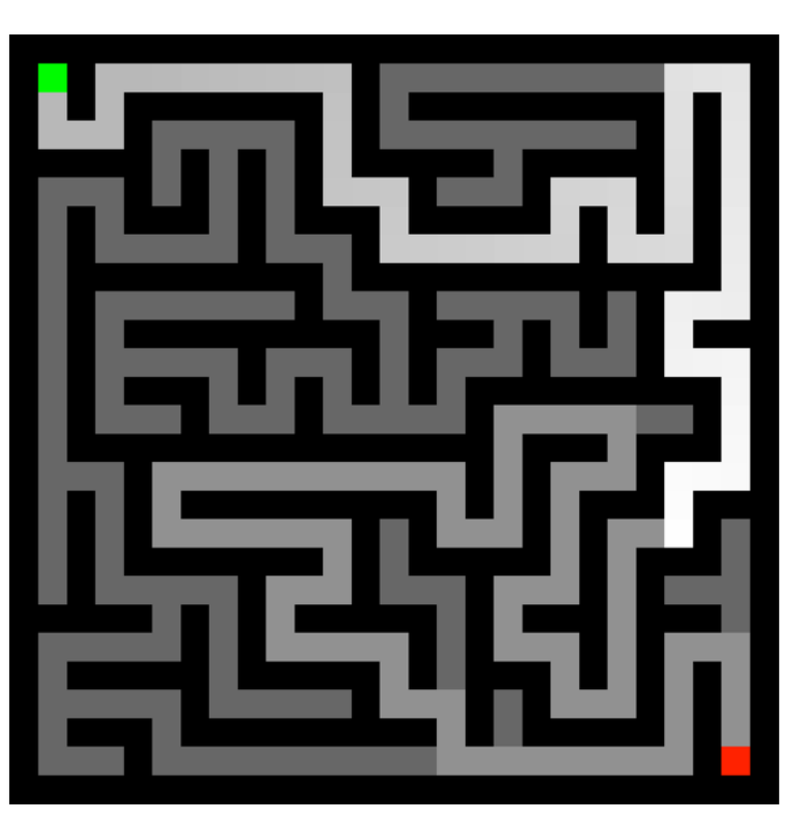
导航是许多游戏中的一个重要概念。一些回合制游戏要求玩家根据棋盘上的位置选择通往目标的最佳路径。许多动作和冒险游戏将角色置于某种形式的迷宫中——玩家角色必须解决迷宫才能实现游戏目标，而敌人角色必须努力通过迷宫向玩家提出挑战。确定如何穿越游戏板、迷宫或其他可导航空间的过程称为*寻路*。GameplayKit提供了一套实用程序，用于映射您的游戏世界并找到通过它找到路径，然后您可以使用这些实用程序来移动角色或其他游戏实体。

使用GameplayKit中的寻路实用程序需要将游戏中的导航区域描述为一个*图表*——不同位置或*节点*的集合，以及指定实体如何从一个位置导航到另一个位置的连接。由于大多数涉及二维运动的游戏都符合此描述，因此调整此类游戏以使用寻路方式通常是找到一种方便的方法来创建游戏世界的图形描述。

示例：探路者

在在本章中探索GameplayKit寻路功能的全部功能之前，您可以通过下载示例代码项目*Pathfinder: GameplayKit寻路基础知识*快速体验。这个简单的游戏会自动创建迷宫，然后使用GameplayKit解决它们，突出显示每个迷宫的最短路径，如图6-1所示。

图6-1 行之探路者样本代码项目在行动

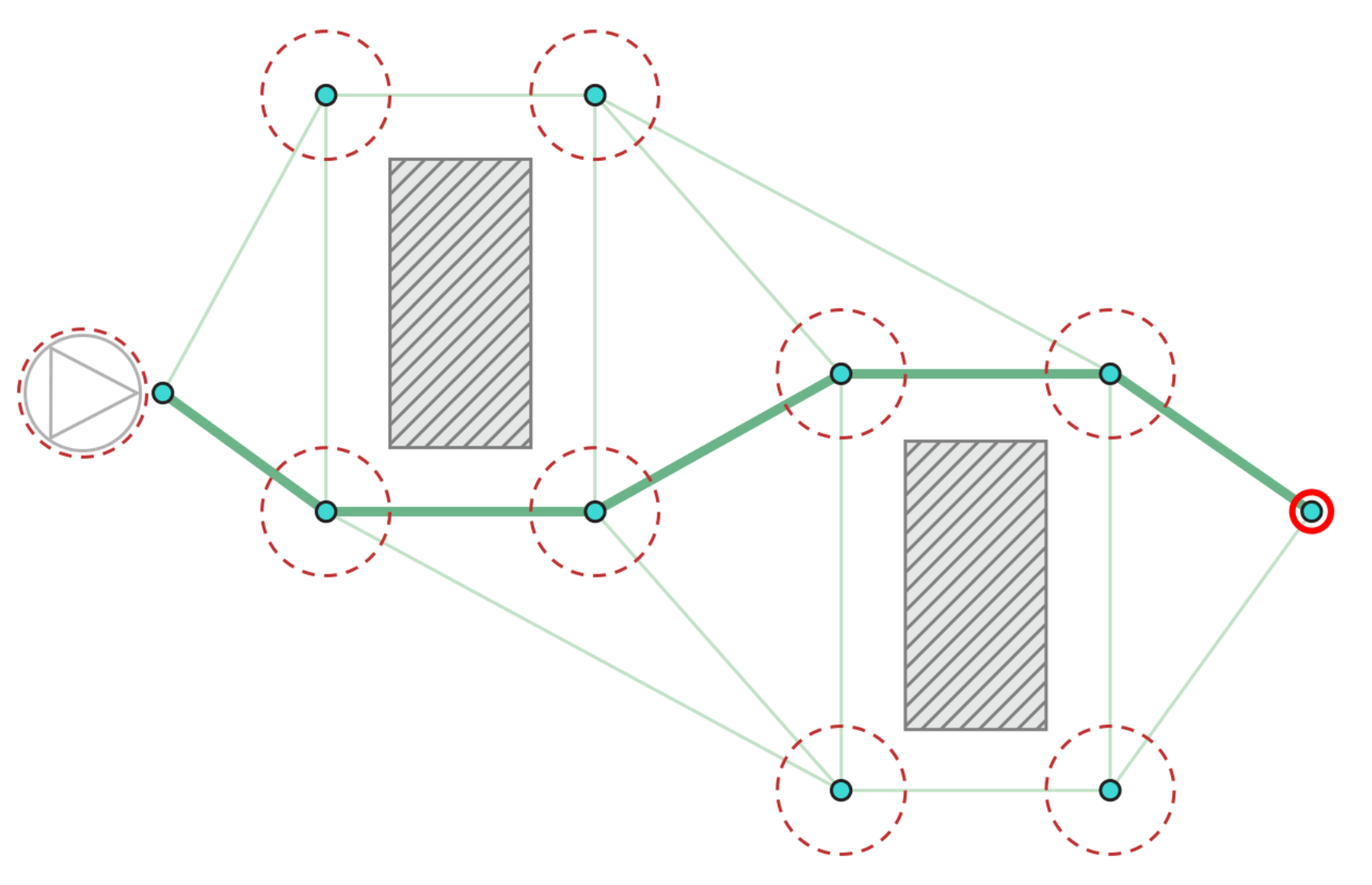


为寻路设计你的游戏

您使用GKGraph类和相关API描述图表，这些API提供了三种建模可导航区域的方法。

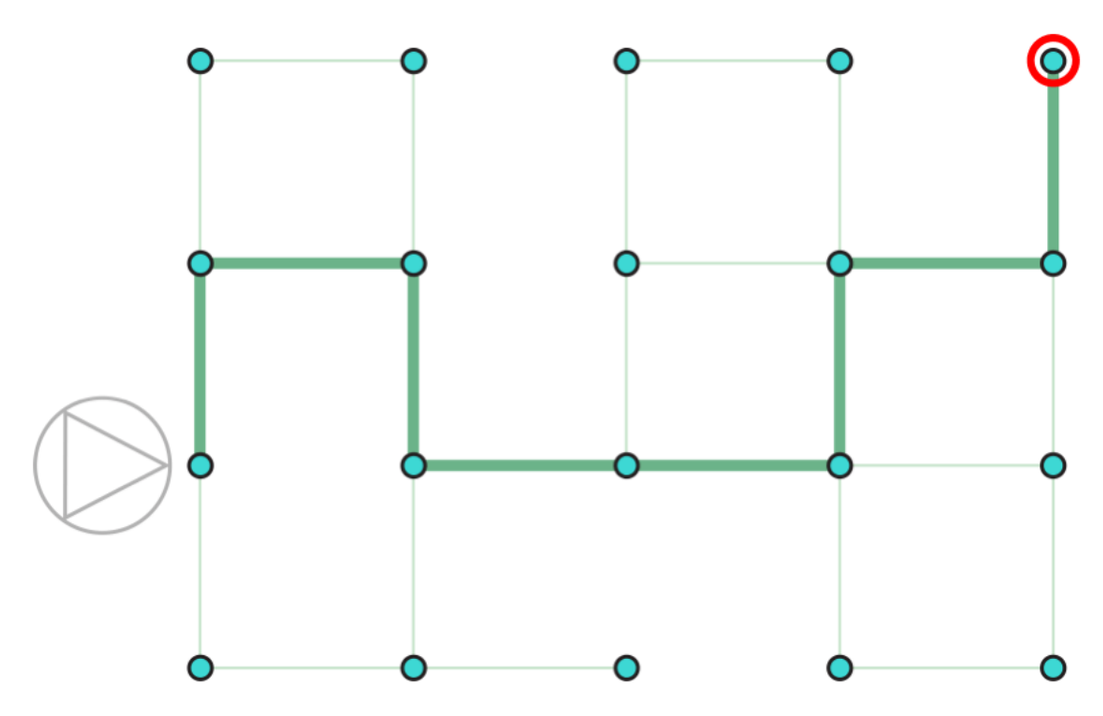
- **作为一个连续的空间被障碍物打断。**使用GKPolygonObstacle类来建模不可通行区域，使用GKGraphNode2D类来建模开放空间中感兴趣的位置，使用GKObstacleGraph类来创建包含两者的图。图6-2说明了这种格式，这可以在许多2D动作、冒险和益智游戏中找到，以及与游戏相关的运动严格为2D的3D游戏。

图6-2 找到障碍物周围的路径



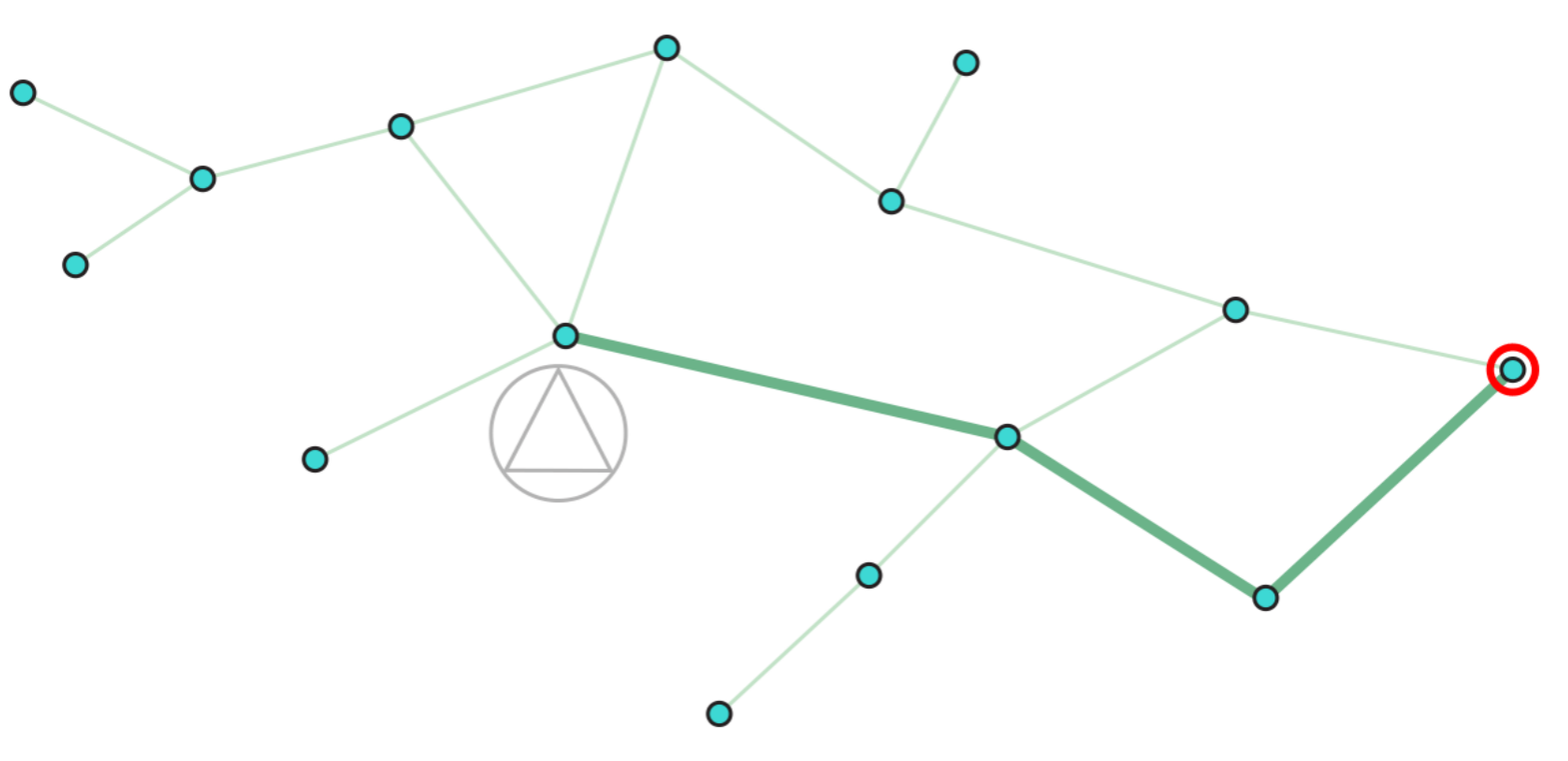
  When building games with SpriteKit, you can generate a collection of GKPolygonObstacle objects based on the contents of a scene or node. With this technique, you can define navigable regions using methods you might already be using for other game mechanics. For example, you can design a level with the SpriteKit Scene Editor in Xcode and use physics bodies to mark regions that the player (or other game entities) cannot pass through, then use the obstaclesFromNodePhysicsBodies: method to generate GKPolygonObstacle objects marking impassable regions.
- **作为一个离散的二维网格，其中唯一有效的位置是那些具有整数坐标的位置。**使用GKGridGraph和GKGridGraphNode类创建基于网格的图。图6-3说明了这种格式，它出现在许多经典的街机游戏、棋盘游戏和战术角色扮演游戏中。

图6-3 通过网格找到一条路径



  在一些此类游戏中，角色移动似乎是连续的，但为了游戏逻辑的目的，所有位置都限制在一个整数网格中。
- **作为离散位置及其之间连接的集合。**使用GKGraphNode类建模没有关联几何信息的节点和连接，然后使用GKGraph类处理整个图形。这种风格适用于实体在标记清晰的空间之间移动的游戏，但它们在空间中的实际位置与游戏无关。图6-4说明了这种格式，它适用于某些类型的棋盘游戏和策略游戏。

图6-4 在任意图形中查找路径



  节点之间的连接是有*方向*的——也就是说，从节点A到节点B的连接仅表示可以从节点A导航到节点B。为了使从节点B导航到节点A也成为可能，还必须存在从节点B到节点A的单独连接。

创建图后，您可以使用该GKGraph对象根据新节点的关系将其连接到图中的现有节点，从图中删除节点，无论它们与现有节点的关系如何，最重要的是，找到从图中一个节点到另一个节点的路径。

为您的游戏添加寻路

在游戏中使用寻路的典型工作流程涉及三到四个步骤：

1. 在游戏开始之前，使用《为寻路设计游戏》中描述的任何一种风格，创建一个GKGraph类或其子类的实例来表示游戏场景的静态内容。
2. 当您需要在游戏中找到动态实体使用的路径时——例如，要让玩家角色绕过指向点击或点击事件位置的障碍物时——要么找到与这些实体位置匹配的现有节点，要么创建代表这些实体的临时节点并将其连接到图表。
3. 使用findPathFromNode:toNode:方法通过图找到路径。

   This method returns an array of GKGraphNode objects representing the path to follow. (The subclass of GKGraphNode in the array depends is the same subclass you used to create the graph.) You can use the position data contained in the nodes along the path to move your game entities—for example, a SpriteKit game can create a sequence of SKAction objects to move a game character to the locations on the path.
4. （可选）找到路径后，从图中断开任何临时节点，以免干扰未来的寻路。

以下各节说明了寻路方法在两个示例游戏中的使用。

网格寻路

迷宫示例代码项目（已经在实体、组件和状态机器章节中看到）实现了几种经典街机游戏的变体。在游戏中，玩家和敌人角色都在整数网格上穿过迷宫。移动似乎之所以连续，只是因为动画在网格位置之间插值。

> 注
>
> 本节讨论示例代码项目迷宫：GameplayKit入门的功能。下载它，在Xcode中跟随。

本游戏创建一个GKGridGraph对象来表示迷宫，如清单6-1所示。

清单6-1 生成网格图

```
GKGridGraph *graph = [GKGridGraph graphFromGridStartingAt:(vector_int2){0, 0} width:AAPLMazeWidth height:AAPLMazeHeight diagonalsAllowed:NO];
NSMutableArray *walls = [NSMutableArray arrayWithCapacity:AAPLMazeWidth*AAPLMazeHeight];
NSMutableArray *spawnPoints = [NSMutableArray array];
for (int i = 0; i < AAPLMazeWidth; i++) {
    for (int j = 0; j < AAPLMazeHeight; j++) {
        int tile = [self tileAtRow:i column:j];
        if (tile == TileTypeWall) {
            [walls addObject:[graph nodeAtGridPosition:(vector_int2){i, j}]];
        } else if (tile == TileTypePortal) {
            [spawnPoints addObject:[graph nodeAtGridPosition:(vector_int2){i, j}]];
        } else if (tile == TileTypeStart) {
            _startPosition = [graph nodeAtGridPosition:(vector_int2){i, j}];
        }
    }
}
// Remove wall tiles from the graph so that paths go around them.
[graph removeNodes:walls];
```

graphFromGridStartingAt:width:height:diagonalsAllowed:方法为指定矩形网格中的每个位置创建一个具有节点的图。然后，该代码将迷宫中与墙壁对应的网格位置的节点移除，仅将迷宫的可遍历区域作为图形节点。

当敌方角色处于追逐状态时，他们使用寻路获取通往玩家当前位置的路线。清单6-2显示了这种行为的实现。

列出6-2个 已知网格位置的路径查找

```
- (NSArray<GKGridGraphNode *> *)pathToNode:(GKGridGraphNode *)node {
    GKGridGraph *graph = self.game.level.pathfindingGraph;
    GKGridGraphNode *enemyNode = [graph nodeAtGridPosition:self.entity.gridPosition];
    NSArray<GKGridGraphNode *> *path = [graph findPathFromNode:enemyNode toNode:node];
    return path;
}

- (void)startFollowingPath:(NSArray<GKGridGraphNode *> *)path {
    // Set up a move to the first node on the path, but
    // no farther because the next update will recalculate the path.
    if (path.count > 1) {
        GKGridGraphNode *firstMove = path[1]; // path[0] is the enemy's current position.
        AAPLSpriteComponent *component = (AAPLSpriteComponent *)[self.entity componentForClass:[AAPLSpriteComponent class]];
        component.nextGridPosition = firstMove.gridPosition;
    }
}
```

在这个例子中，敌人和玩家都占据了图表中已经的位置。（同样，即使角色的动作是动画显示连续的，出于游戏目的，每个角色总是处于整数网格位置。）因此，要找到敌人追逐玩家的路径，此方法只需查找与敌人和玩家当前位置对应的GKGridGraphNode对象，然后使用findPathFromNode:toNode:方法查找路径。

To move the enemy character, this method examines only the first two nodes in the path. This method is called from an updateWithDeltaTime: method, so it executes for every frame of animation—possibly finding a new path in response to the changes in the player's location. Because it takes longer to animate the enemy character's move along the first step of the path than to calculate a new path, this method simply starts that animation by setting a target position for the game's SpriteComponent class. (See the Entities and Components chapter for discussion of this class.)

围绕障碍寻找路径

DemoBots示例代码项目实现了一个不同的游戏，角色可以在开放空间自由移动，因此基于网格的寻路不可行。相反，这款游戏使用基于障碍的寻路。

> 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。

To prepare for this style of pathfinding, this example uses a set of nodes in each level's scene file (created with the SpriteKit Scene Editor in Xcode) whose physics bodies designate areas of the game world that should be impassable to game characters. Then, the LevelScene class implements a graph property that builds a GKObstacleGraph object, as shown in Listing 6-3.

清单6-3 创建基于障碍的图表

```
lazy var obstacleSpriteNodes: [SKSpriteNode] = self["world/obstacles/*"] as! [SKSpriteNode]

lazy var polygonObstacles: [GKPolygonObstacle] = SKNode.obstaclesFromNodePhysicsBodies(self.obstacleSpriteNodes)

lazy var graph: GKObstacleGraph = GKObstacleGraph(obstacles: self.polygonObstacles, bufferRadius: GameplayConfiguration.TaskBot.pathfindingGraphBufferRadius)
```

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

The graph property's initializer first reads the polygonObstacles property, which searches the level's node tree (see Searching the Node Tree in *SKNode Class Reference*) to provide an array of all child nodes of the obstacles node. Then, the SKNode method obstaclesFromNodePhysicsBodies: creates an array of GKPolygonObstacle objects corresponding to the shapes of each obstacle node's physics body. Finally, these objects create a graph.

当需要为游戏角色规划路线时，游戏的TaskBotBehavior类使用清单6-4中显示的方法。

列出6-4 在基于障碍的图表中查找路径

```
private func addGoalsToFollowPathFromStartPoint(startPoint: float2, toEndPoint endPoint: float2, pathRadius: Float, inScene scene: LevelScene) -> [CGPoint] {

    // Convert the provided `CGPoint`s into nodes for the `GPGraph`.
    let startNode = connectedNodeForPoint(startPoint, onObstacleGraphInScene: scene)
    let endNode = connectedNodeForPoint(endPoint, onObstacleGraphInScene: scene)

    // Find a path between these two nodes.
    let pathNodes = scene.graph.findPathFromNode(startNode, toNode: endNode) as! [GKGraphNode2D]

    // Create a new `GKPath` from the found nodes with the requested path radius.
    let path = GKPath(graphNodes: pathNodes, radius: pathRadius)

    // Add "follow path" and "stay on path" goals for this path.
    addFollowAndStayOnPathGoalsForPath(path)

    // Remove the "start" and "end" nodes now that the path has been calculated.
    scene.graph.removeNodes([startNode, endNode])

    // Convert the `GKGraphNode2D` nodes into `CGPoint`s for debug drawing.
    let pathPoints: [CGPoint] = pathNodes.map { CGPoint($0.position) }
    return pathPoints
}
```

addGoalsToFollowPathFromStartPoint方法遵循与清单6-2所示相似的步骤集，但基于障碍的图有一些变化：

1. 此方法创建并连接表示路径所需起点和终点的临时节点。

   与基于网格的图不同，连续二维空间图尚未包含这些位置的节点。（connectedNodeForPoint调用是该项目中的一个方便函数，它既创建一个新节点，也使用connectNodeUsingObstacles:方法将其添加到图中。）
2. findPathFromNode:toNode:方法返回表示临时节点之间路由的图形节点数组。
3. 该项目的addFollowAndStayOnPathGoalsForPath方法然后使用GameplayKit的代理功能沿路径移动游戏角色。

   For a discussion of the addFollowAndStayOnPathGoalsForPath method's implementation, see Listing 7-1.
4. 完成这项工作后，startNode和endNode节点将从图中删除，这样它们就不会干扰未来的寻路操作。
5. 最后，该方法还返回一组CGPoint结构，以便游戏可以选择绘制一个覆盖物，说明用于调试的路径。