# 《Supernauts》动态环境中的高级实时路径查找

**Harri Hatinen**
主程序员，Grand Cru


GAME DEVELOPERS CONFERENCE™ CHINA
SHANGHAI INTERNATIONAL CONVENTION CENTER
SHANGHAI, CHINA · OCTOBER 19–21, 2014

# 问题(1/2)

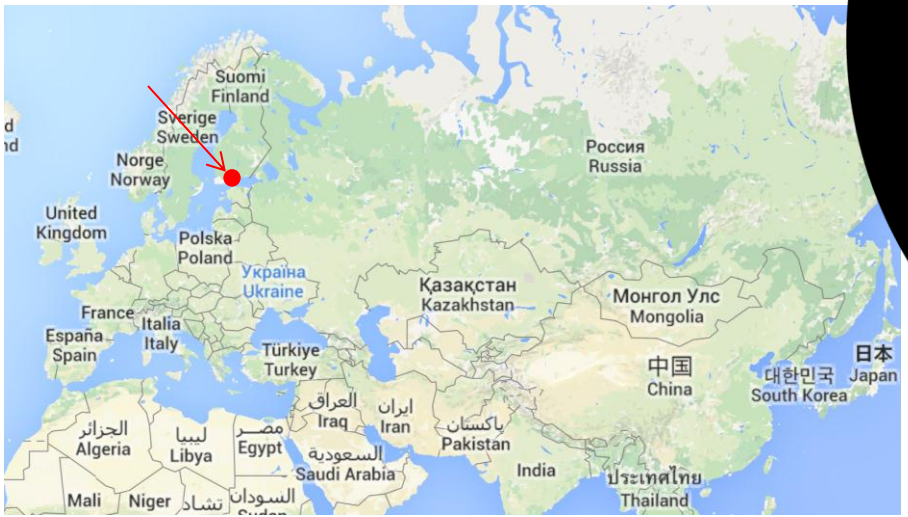
- A*算法是最佳的路径搜索算法
- 提高性能的方法:
  - 更好的试探法
  - **更小的数据结构**

# 问题(2/2)

- 导航网格是是现代游戏的一项标准
- 但是，**用户生成内容**和**不断变化的环境**并没有一个很好的标准。

# Grand Cru公司

- 20名员工
- 芬兰赫尔辛基




GC
Est. 2011
Grand Cru
Kallio, Helsinki

# Harri Hatinen

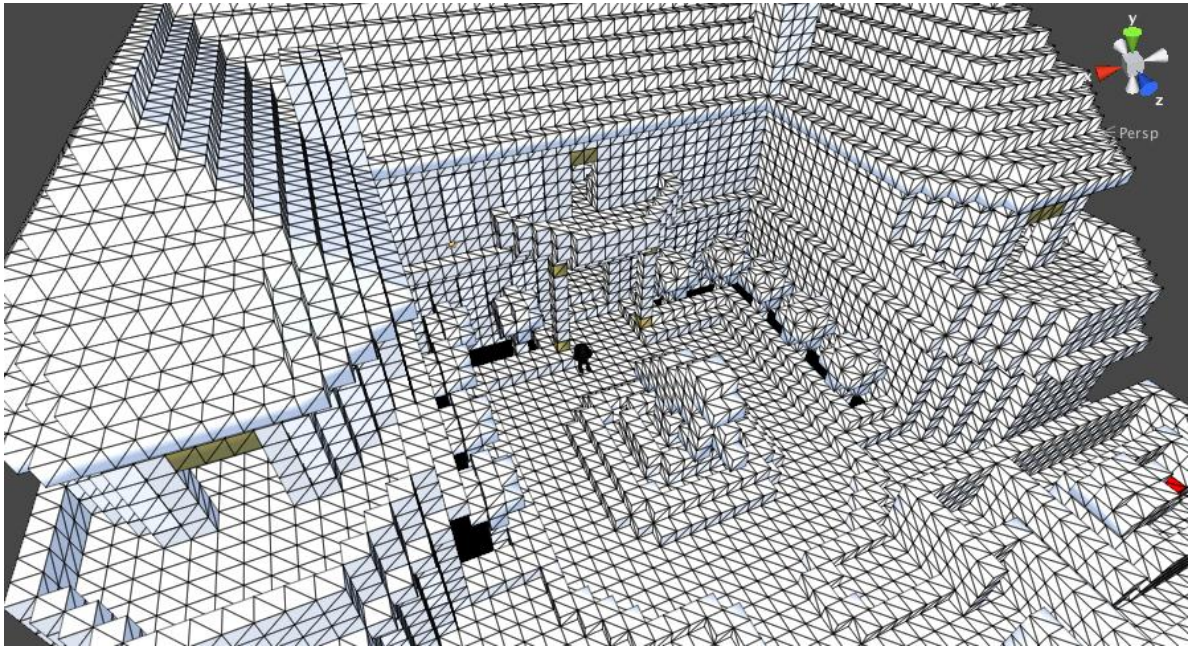

- 联合创始人
- 主程序员
- harri.hatinen@grandcrugames.com


# 《Supernauts》

- 一切都是用户生成的

# 《Supernauts》

- 一切都是用立方体构建的

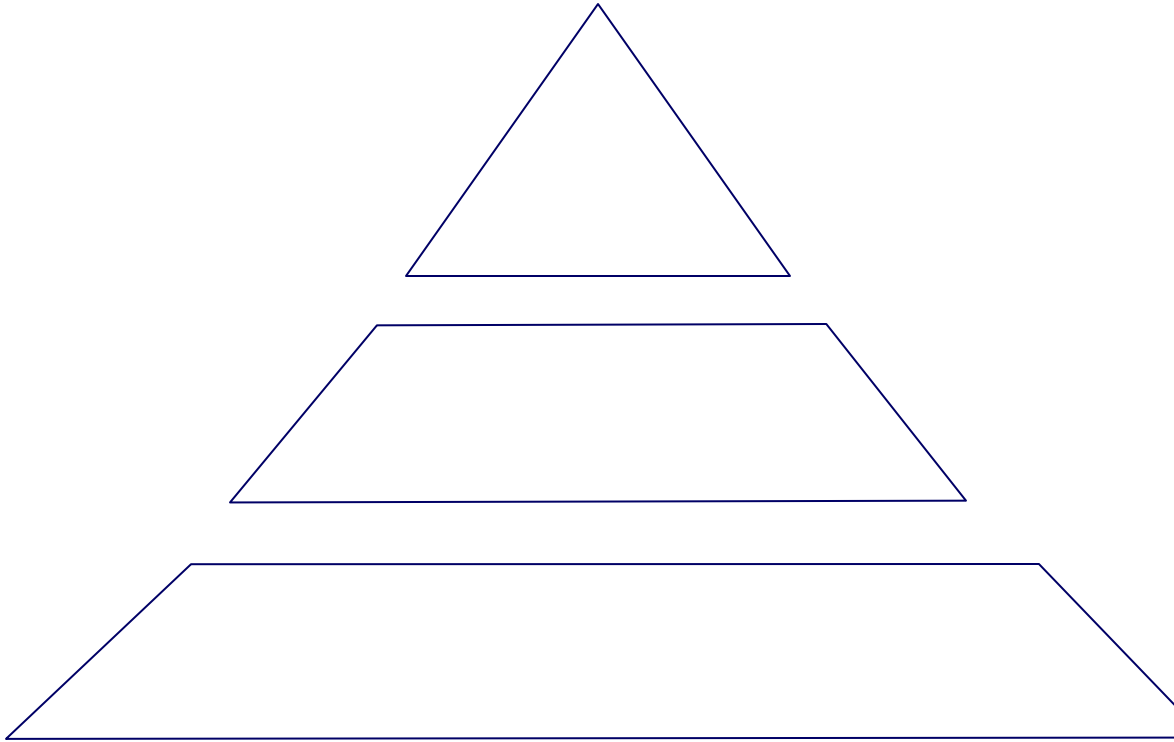



# 《Supernauts》

- 路径查找用于:
  - 运动控制
  - 人工智能(AI)
- [演示视频]

# 解决问题

- 步骤1：实验
- 步骤2：研究
- 步骤3：创新

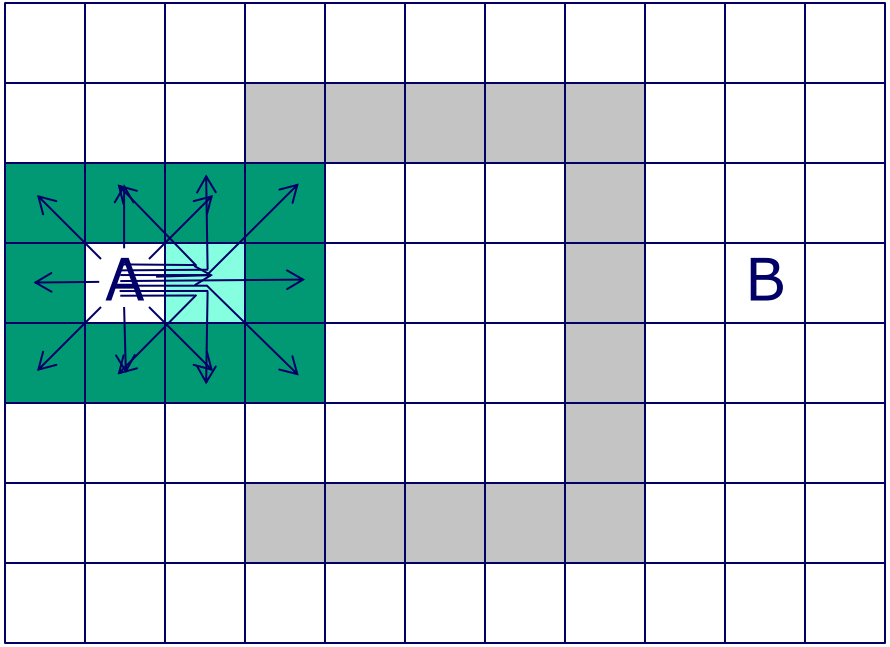


# 步骤1：实验

- 我们是否还需要昂贵的高级路径搜索算法？
- 不要把努力浪费在错误的功能上
- 简单和快速的实现

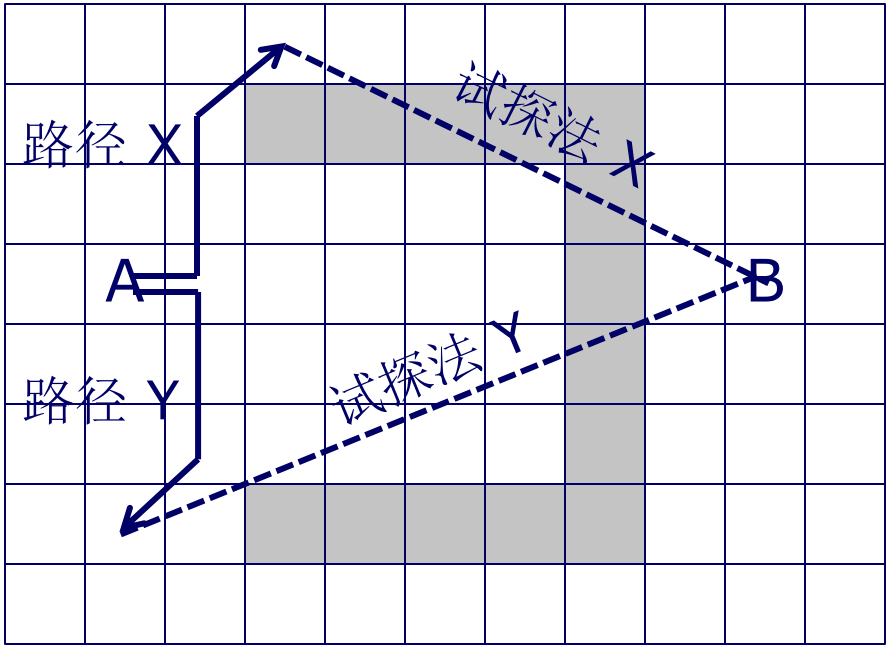
  -> 迅速找出要求和限制

# 线性网格中的A*算法



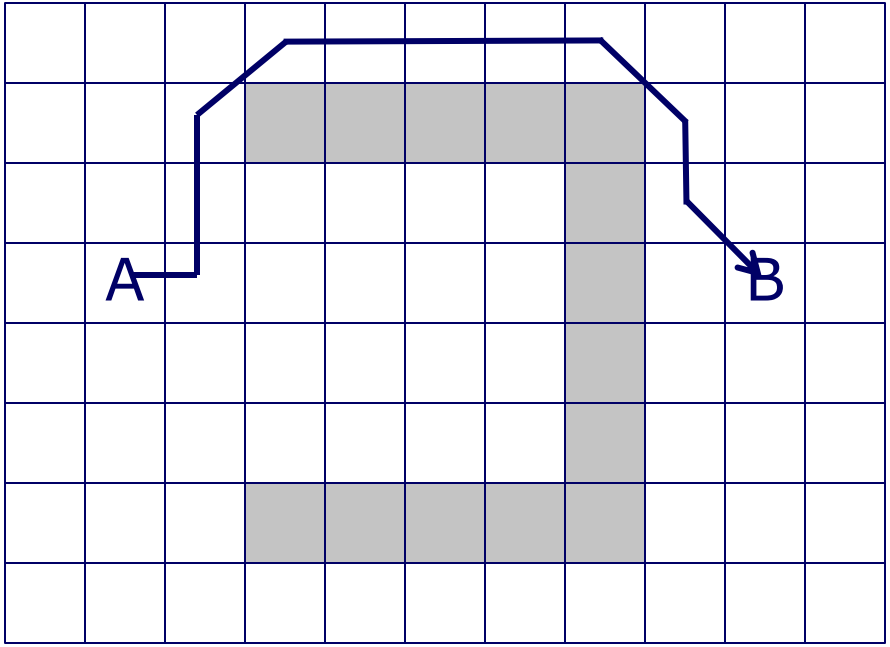

- 从当前节点，将路径扩展到附近的节点。
- 选择**最佳**路径的节点
- 重复

# 线性网格中的A*算法




- 我们如何得知最佳路径？
- X路径长度 = Y路径长度
  - 哪一个更好？
- 通过试探法
- 分值 = 路径 + 试探法
  - 分值更小 = 更好

# 线性网格中的A*算法




最佳路径可能是这样子的。

# 实验:结果

- 1到2小时用于实现
- 事实证明需要一种寻路算法。
- 不过，我们很快就发现这种方法太慢。

# 步骤2:研究

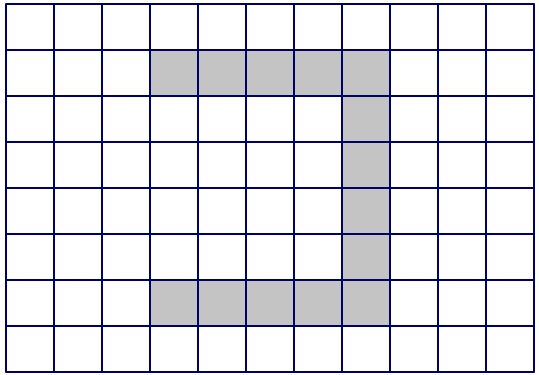
- 现在我们知道
  - 1)我们需要一种路径搜索算法
  - 2)简单的实现是不够的

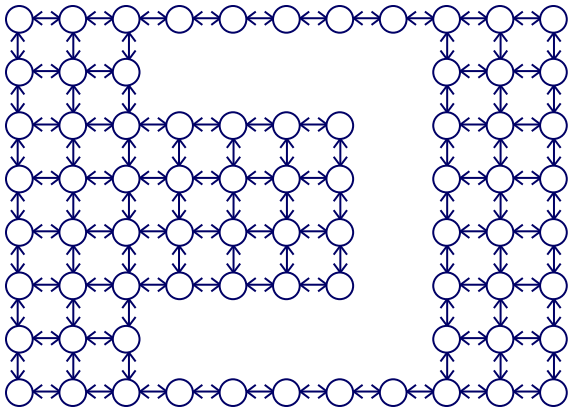
-> 找出和这一主题有关的所有既有知识

# 研究

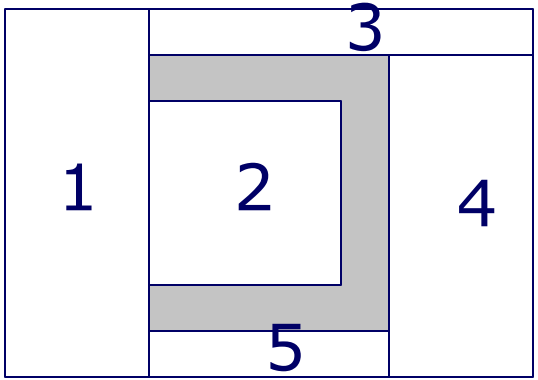
- A*算法是最优的空间搜索算法
- 优化的重点是改善**数据结构**
- 导航网格是现代电子游戏实际运用的标准

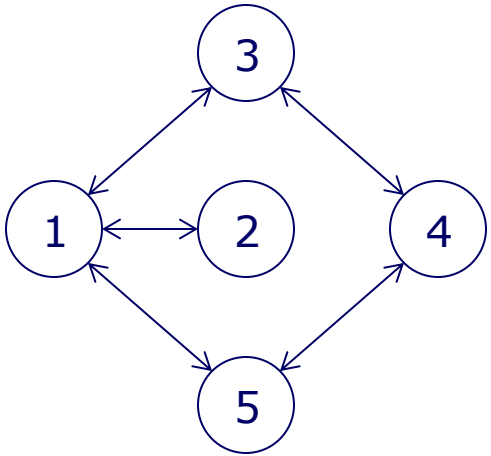
# 什么是导航网格？





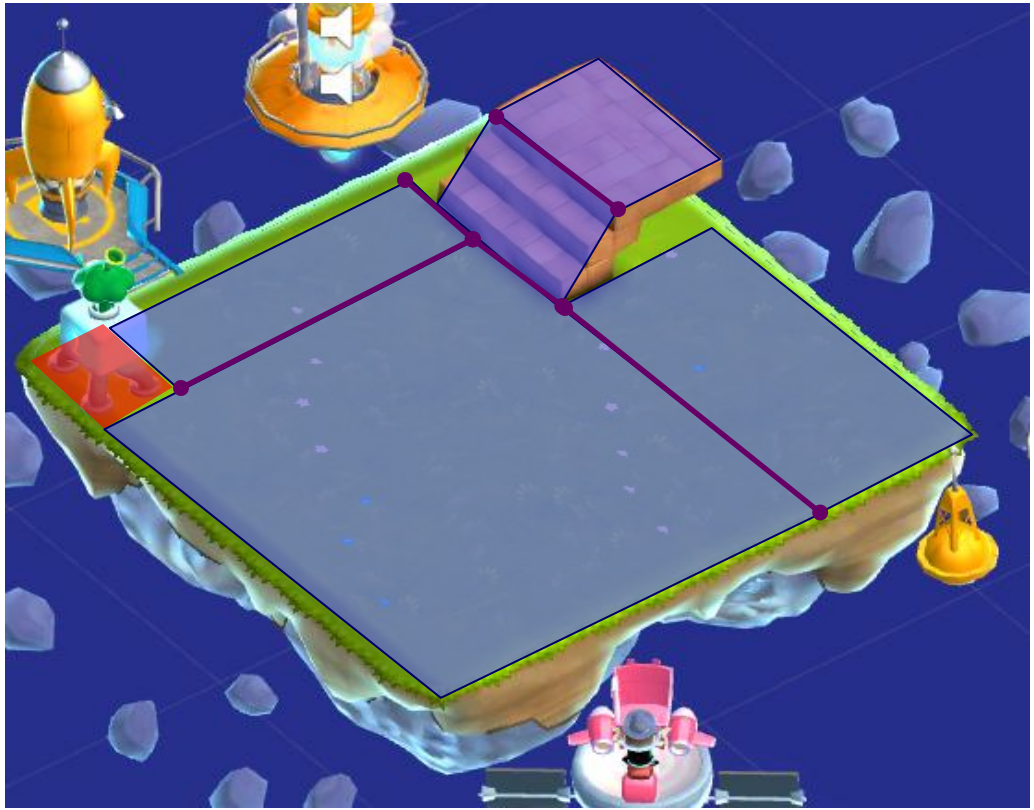
74个节点
114条边



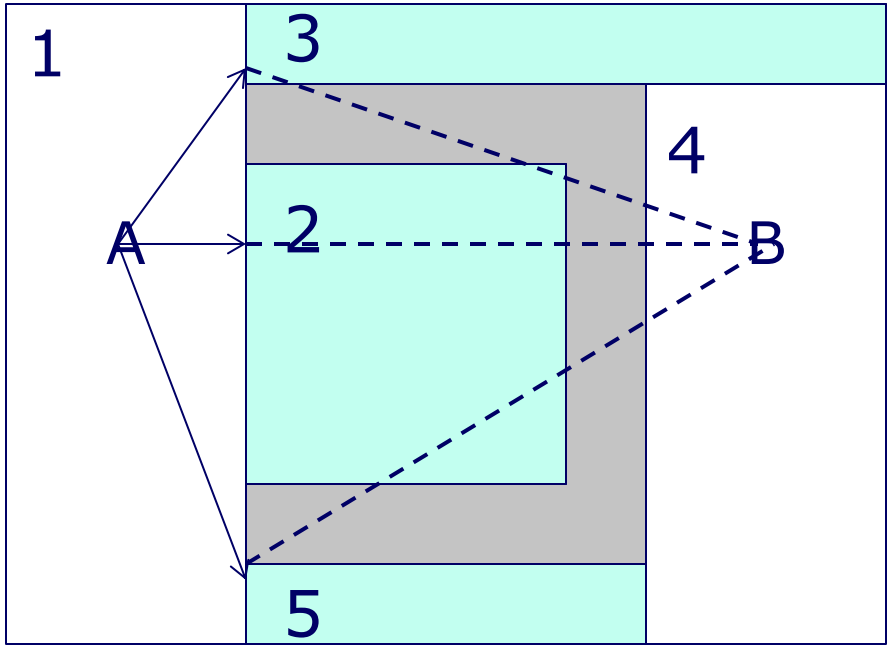




5个节点
5条边

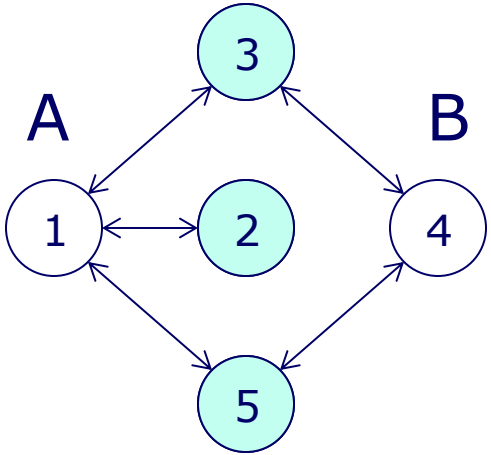
# 《Supernauts》中的导航网格



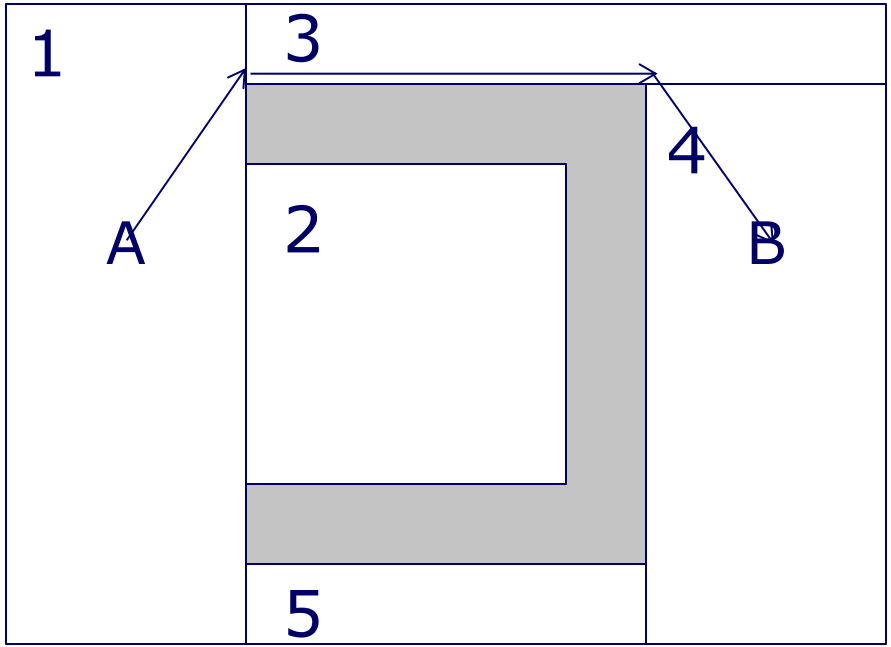
# 基于导航网格的A*算法

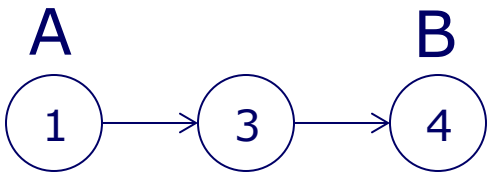



- 比在网格中更加接近
- 循环访问**相邻节点**并使用**试探法**

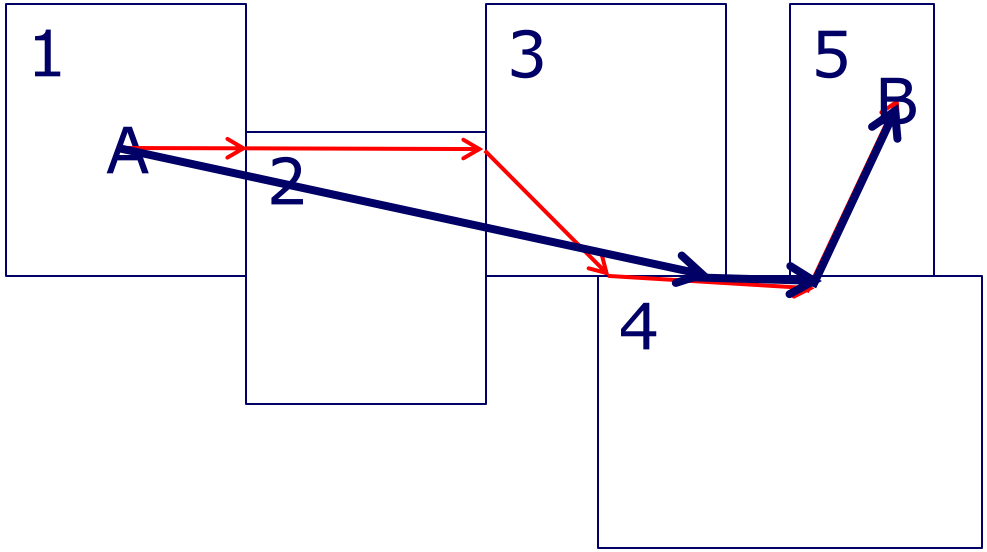


# 基于导航网格的A*算法

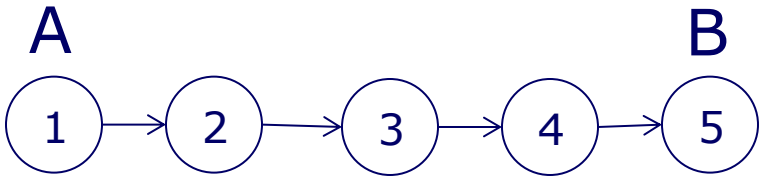


# 导航网格中的直线路径

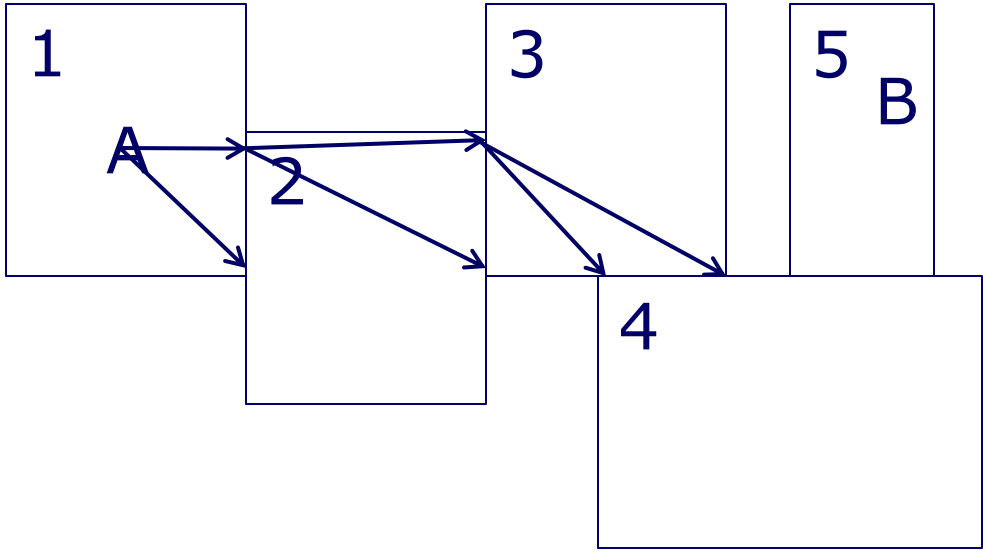



最短路径是明确的：

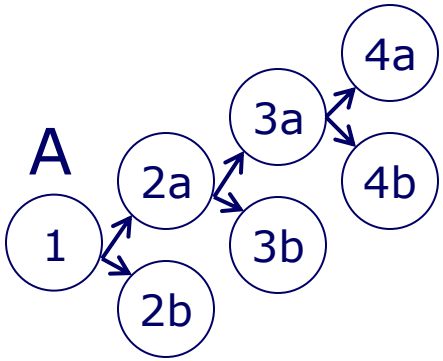



但要如何生成这一路径？

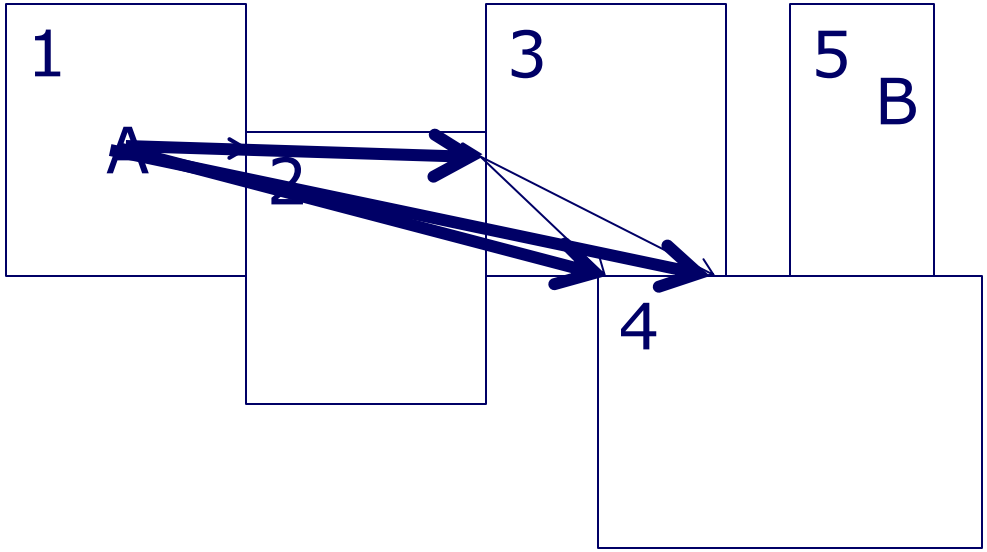
# 导航网格中的直线路径



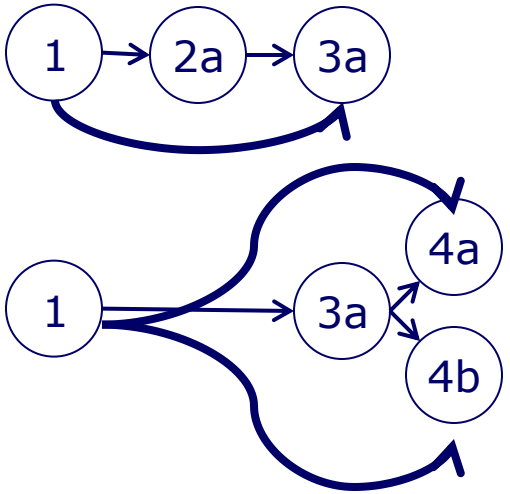

- 用两条替代路径扩展到相邻节点
- 公共边的最远端



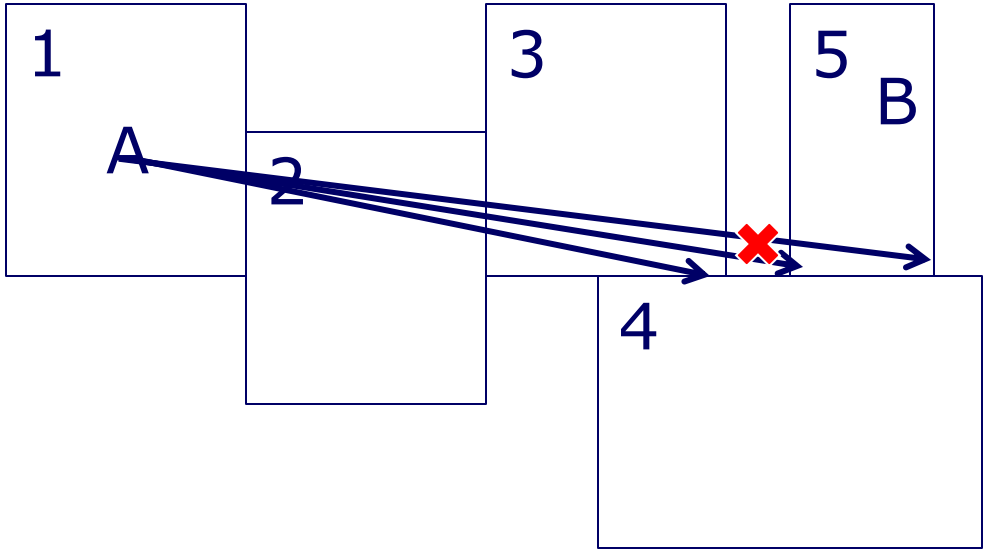
# 导航网格中的直线路径

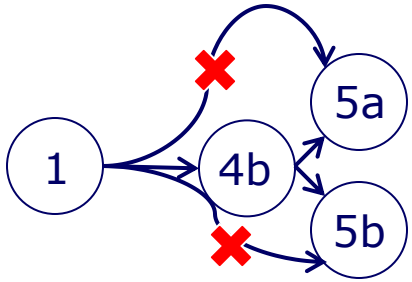



- 在每次循环访问中，跟踪路径尽可能向后



# 导航网格中的直线路径

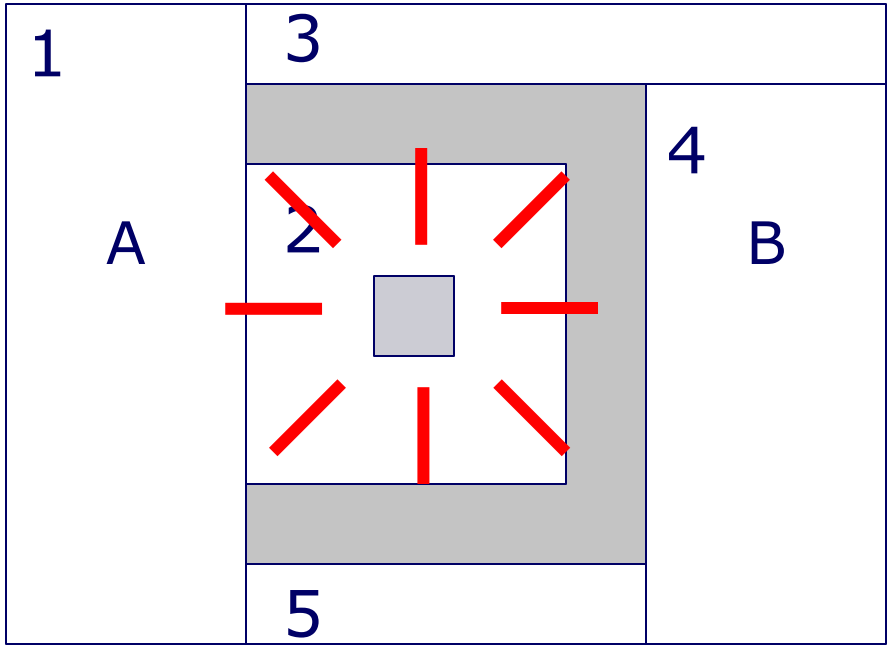


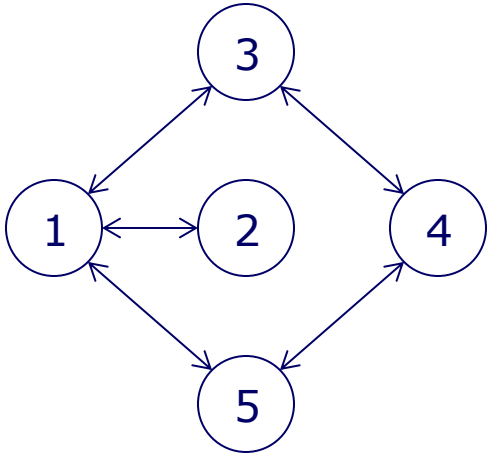



# 生成导航网格

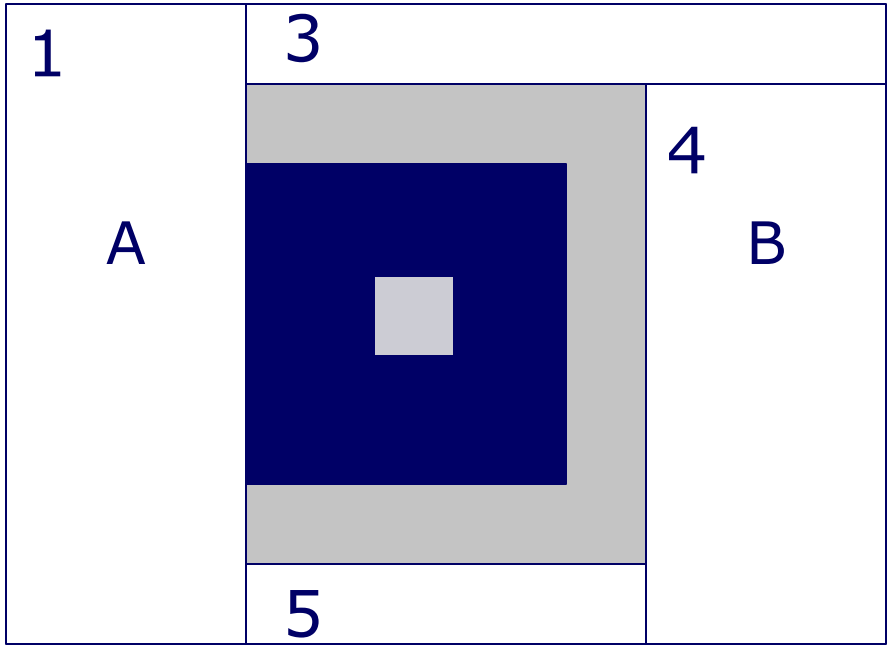
- 贪心算法选择(Greedy Selection)得出近似最优的结果
- 循环访问可走的自由位置，并尽可能地扩展
- 重复，直到所有位置都被填满

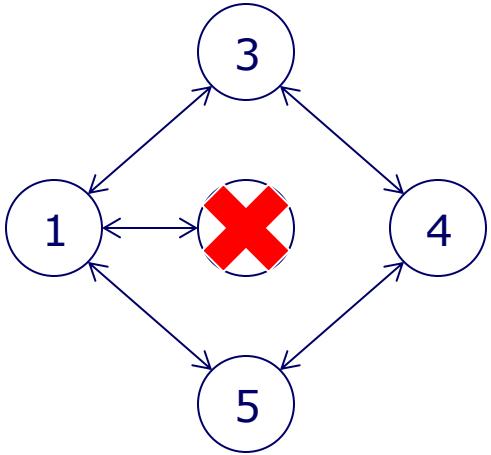
# 更新导航网格



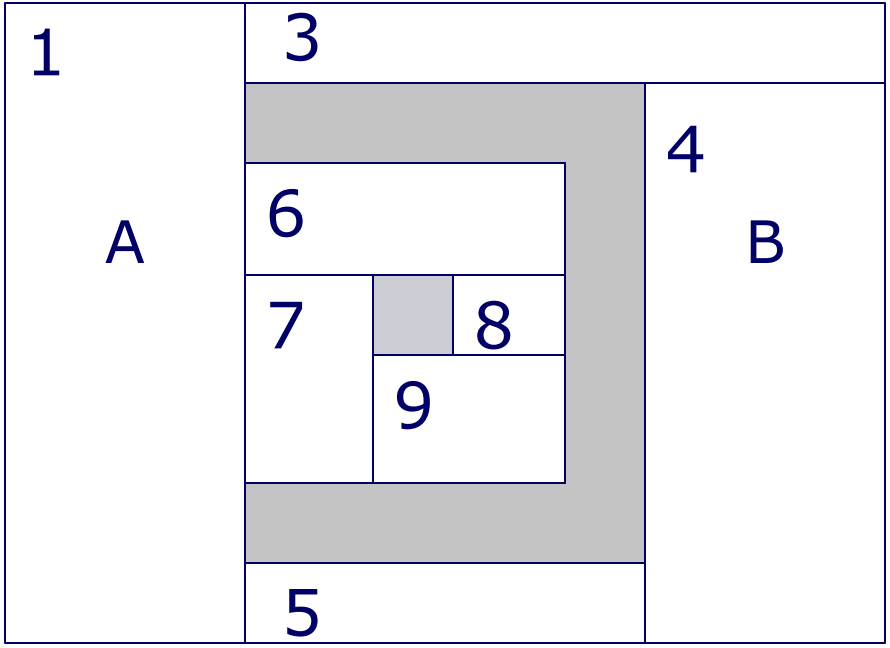



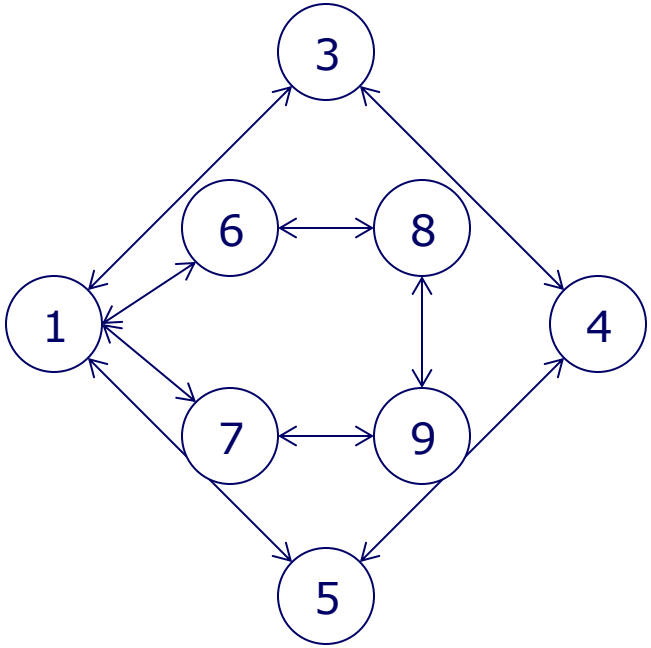
# 更新导航网格



# 更新导航网格



# 研究：结果

- 我们现在拥有足够快的寻路算法了！
- 但是：生成和更新仍然很慢！

# 步骤3：创新

- 现在是时候进行最有趣的部分了：发明新东西！
- 让我们将问题分解一下：
  - 1）缓慢的初始化
  - 2）缓慢的更新

# 1）缓慢的初始化

- 即使是最简单的贪心算法选择也要反复访问大量的方块
  - 算法需要找到所有表面并进行迭代
    - 实际上复杂度$N^3$，大量的方块
  - 由于存在用户生成的内容并且环境不断变化，预先计算是不可能的！

# 2）缓慢的更新

- 大节点环境改变时，会有大面积区域需要重新循环访问。

# 问题...

- 因此，认真分析，我们已经将问题分解成更小的问题：
  - 世界中有太多的方块，难以进行循环访问
  - 节点太大

# ...及其解决方案

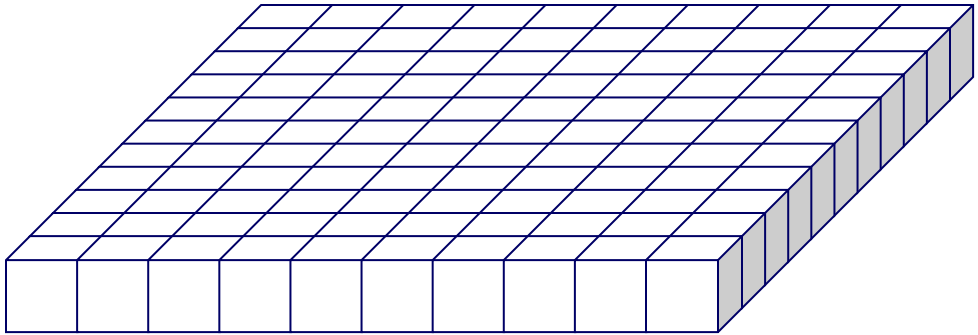
- 过多的循环访问
  - -> 只循环访问我们需要的方块
- 过大的节点
  - -> 限制节点尺寸

# 创新!

- 过多的循环访问
  - -> 限制导航网格的尺寸！
  - -> 生成节点尺寸的上限，不进行有问题的更新！
- 但小型导航网格无法涵盖整个关卡
  - -> 让我们根据需要添加尽可能多的小型导航网格！
  - -> 允许我们只对需要的区域进行循环访问！

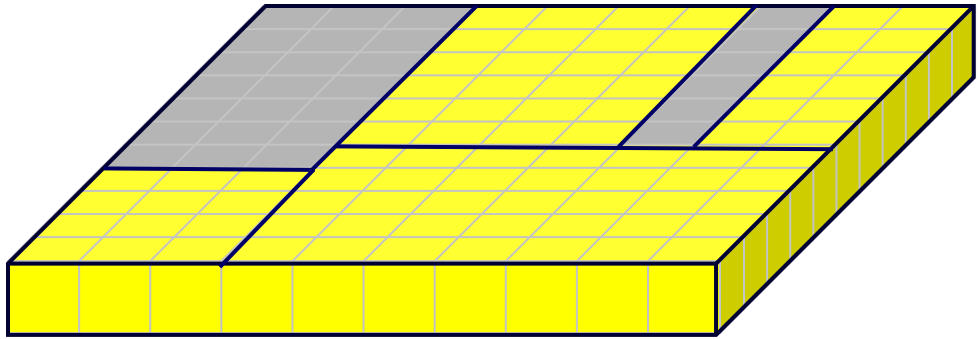
# 《Supernauts》的导航网格

- 10x1x10迷你导航网格



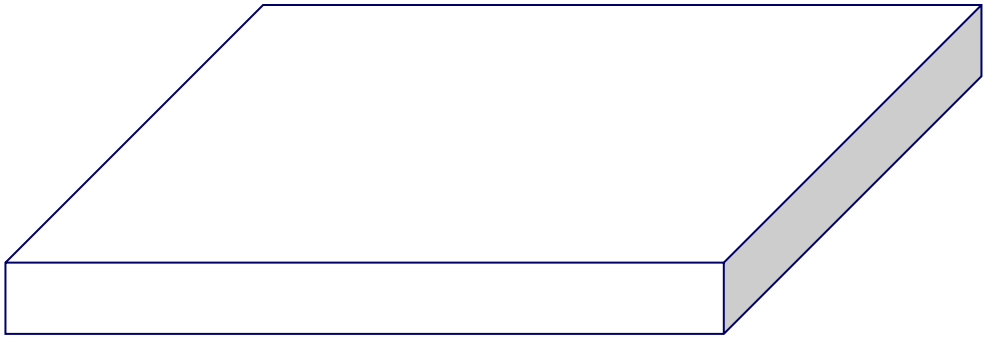
# 《Supernauts》的导航网格

- 10x1x10迷你导航网格



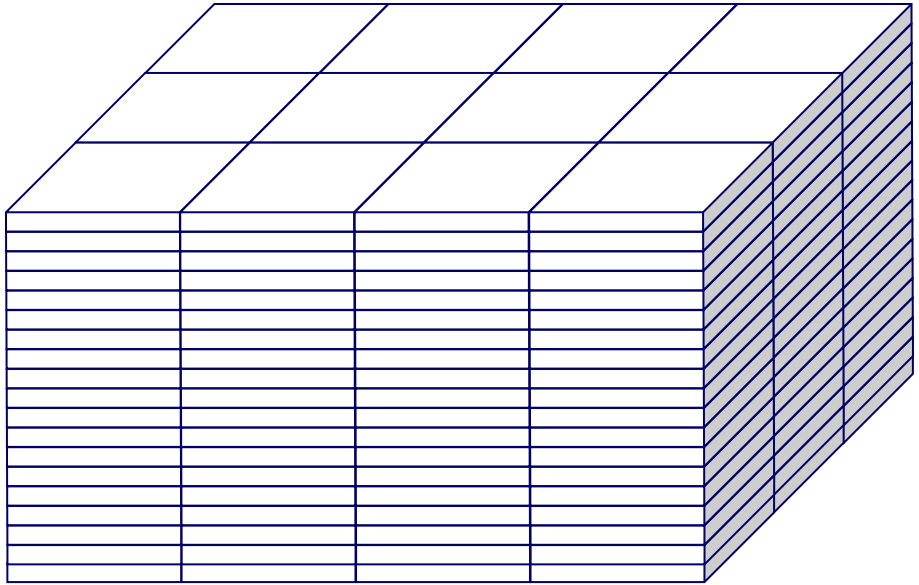
# 《Supernauts》的导航网格

- 整个世界分成均匀的迷你导航网格



# 《Supernauts》的导航网格

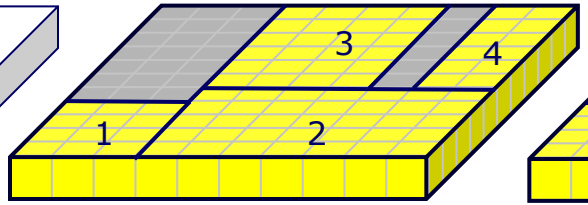
- 整个世界分成均匀的迷你导航网格



# 迷你导航网格

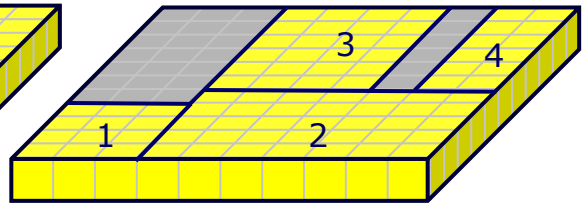
## 空白状态

可以只是一个NULL指针。



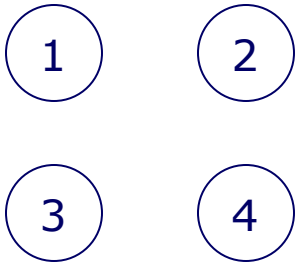

## 建造状态

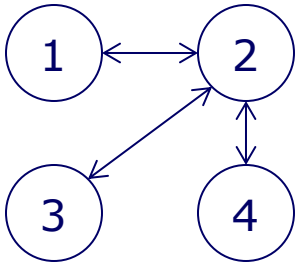
节点建成，但没有连接



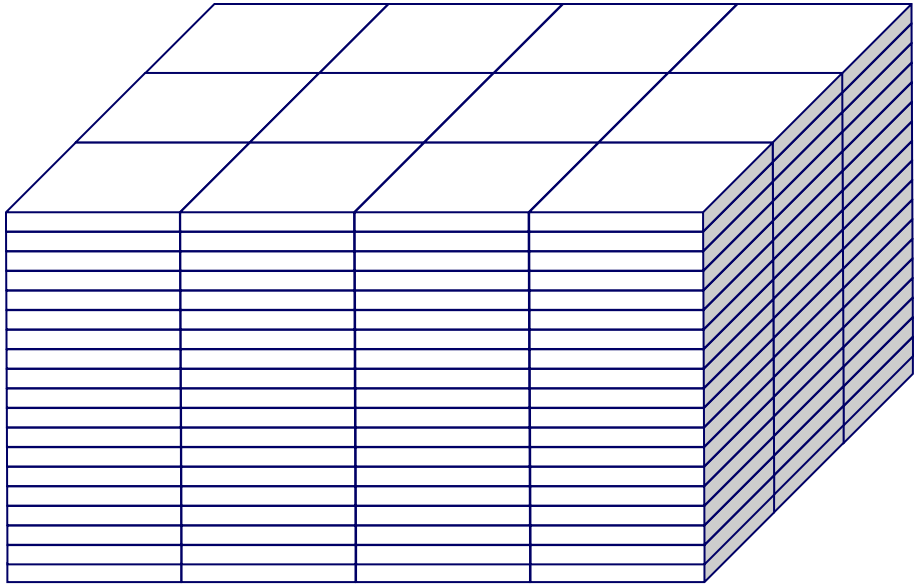

## 连接状态

节点建成并连接

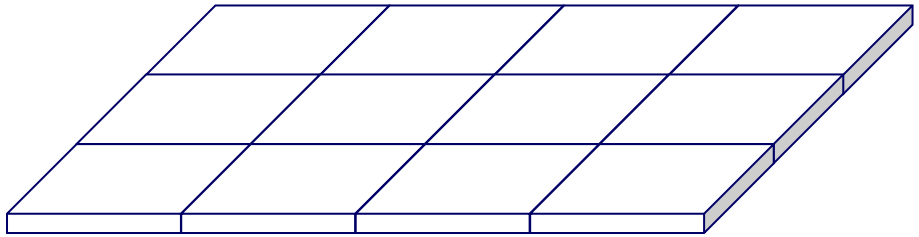


# 《Supernauts》导航网格中的A*算法

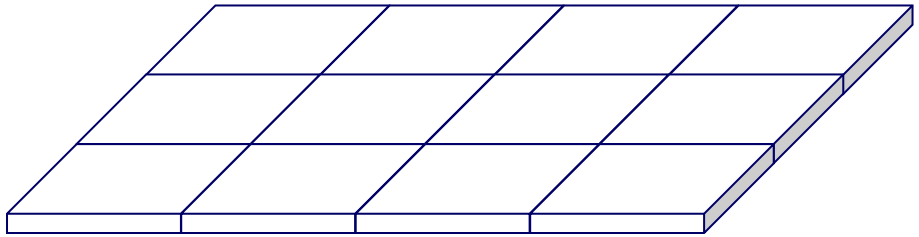


# 《Supernauts》导航网格中的A*算法

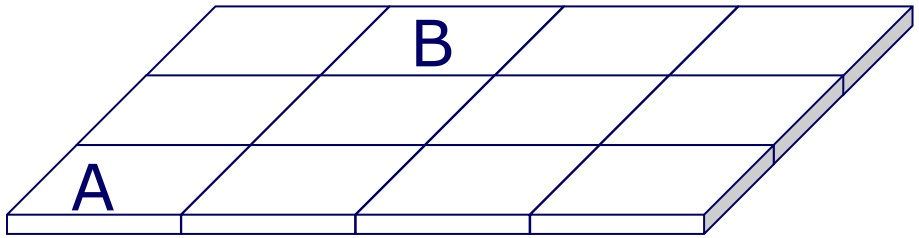


# 《Supernauts》导航网格中的A*算法

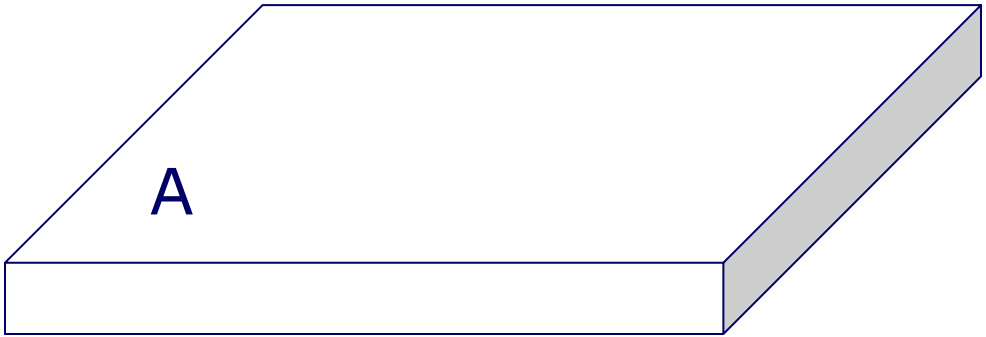
- 最初整个世界都只是空白（NULL）迷你导航网格的阵列。



# 《Supernauts》导航网格中的A*算法

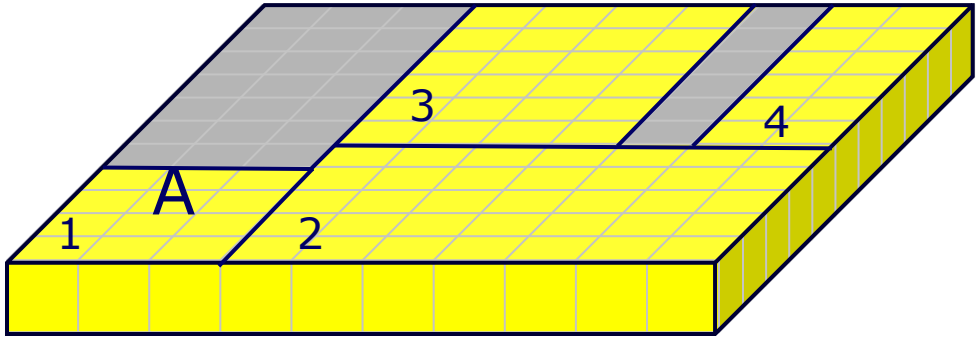


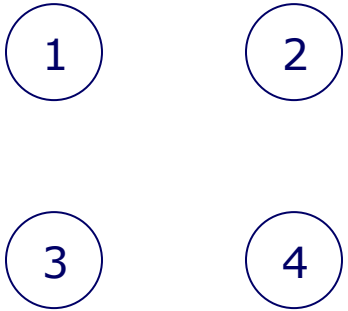
# 《Supernauts》导航网格中的A*算法



# 《Supernauts》导航网格中的A*算法

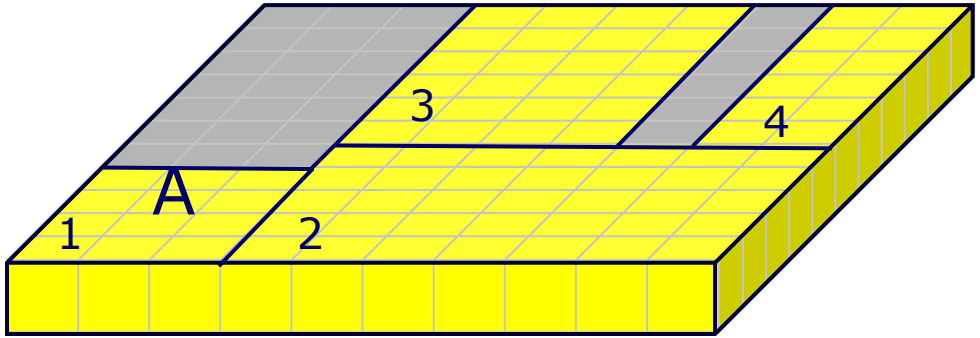
- 构建节点

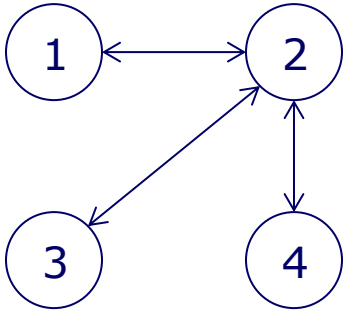


# 《Supernauts》导航网格中的A*算法

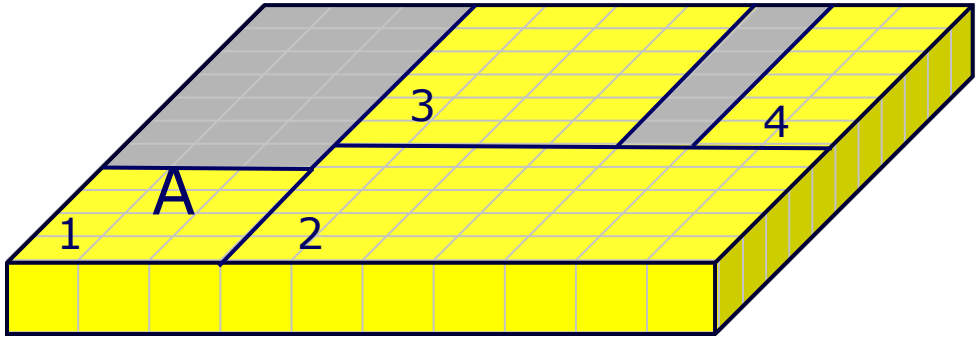
- …并连接节点
- 但是！这还不是所有。
- 我们需要找出连接到外部的链接，否则我们将永远出不去！

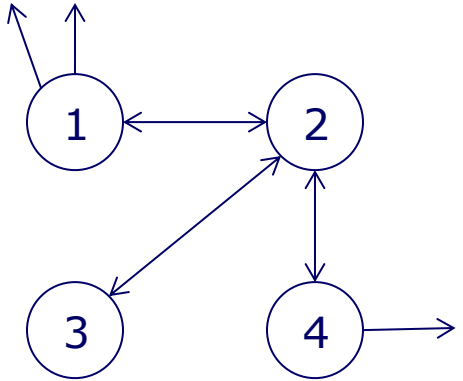


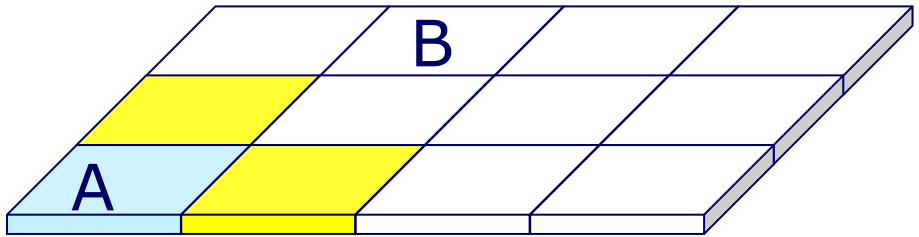
# 《Supernauts》导航网格中的A*算法

- 我们需要检查所有4 x 3的相邻节点，因为角色可能攀爬1个方块或是下降1个方块。
- 更新相邻节点至建造状态，这样我们就能够把节点连接到外部。



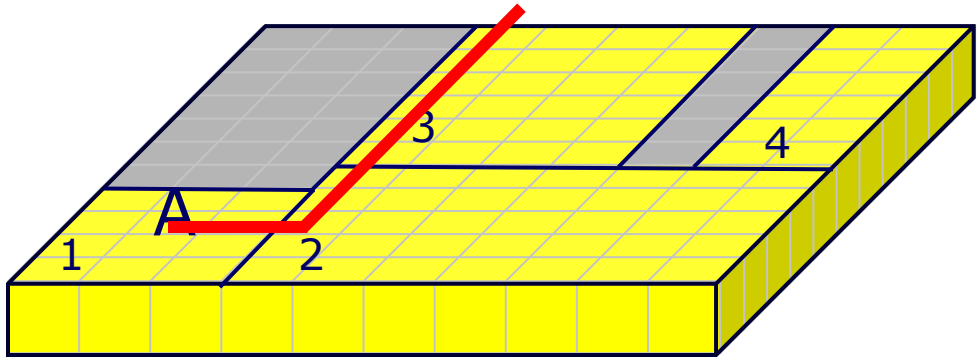
# 《Supernauts》导航网格中的A*算法

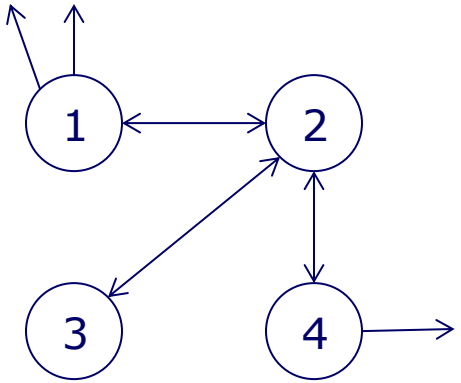



建造

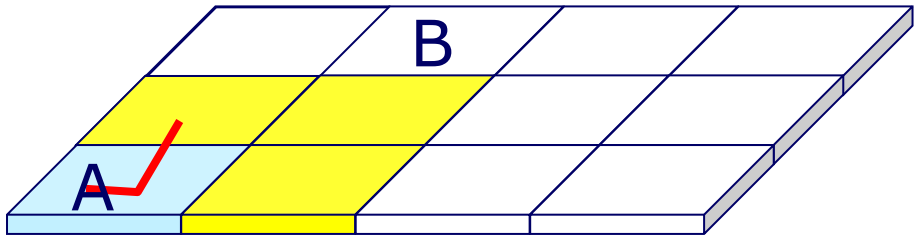
连接

# 《Supernauts》导航网格中的A*算法



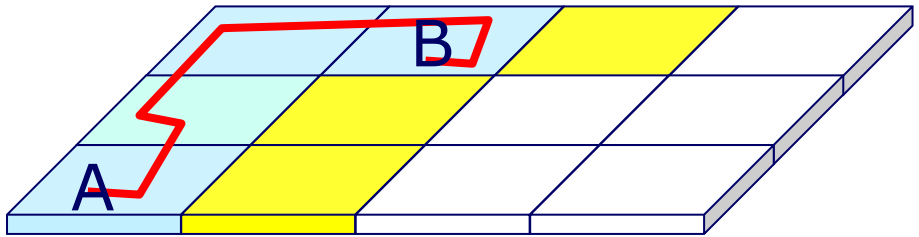
# 《Supernauts》导航网格中的A*算法




建造

连接

# 《Supernauts》导航网格中的A*算法



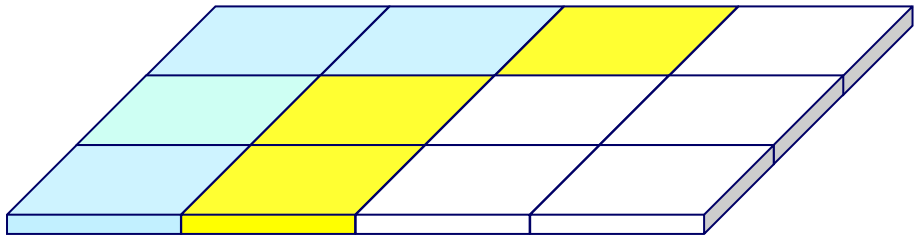

建造

连接

# 更新《Supernauts》的导航网格

- 如何更新？
  - 方法非常简单！

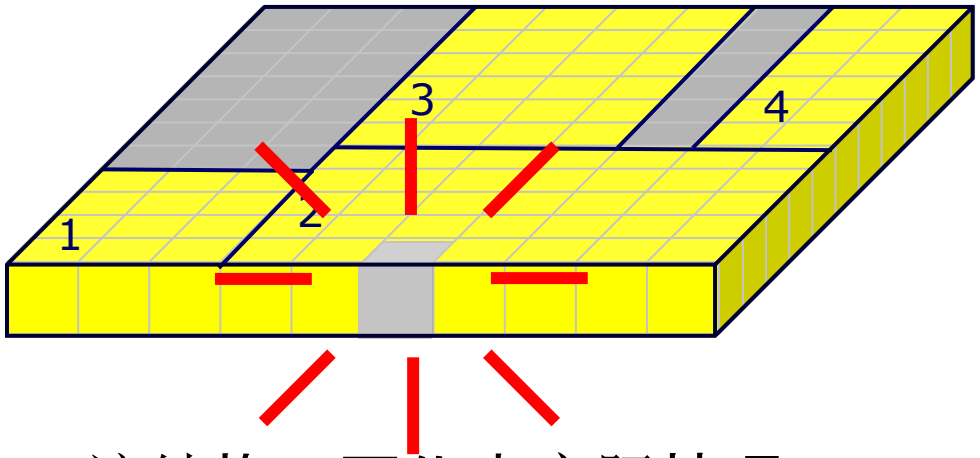
# 更新《Supernauts》的导航网格

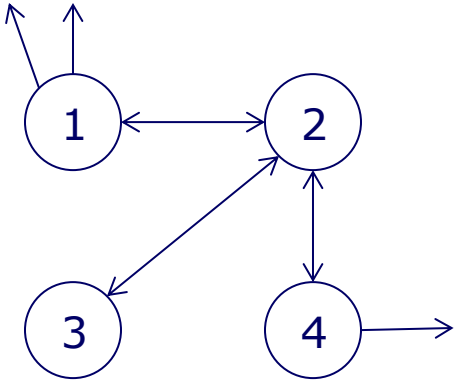


建造

连接

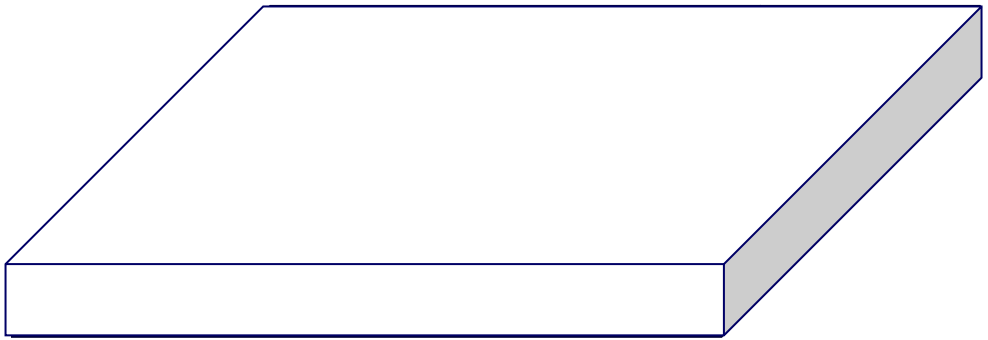
# 更新《Supernauts》的导航网格



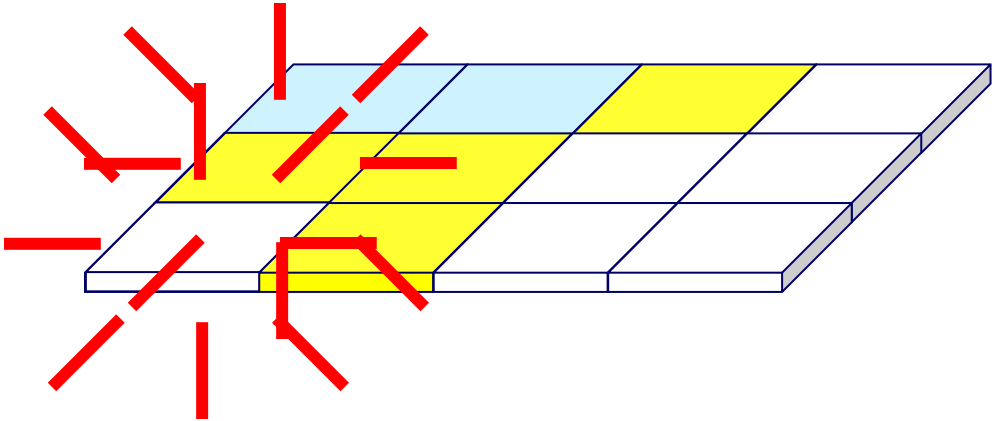

- 该结构不再代表实际情况
- 因为迷你导航网格很小...

# 更新《Supernauts》的导航网格



- 该结构不再代表实际情况
- 因为迷你导航网格很小...
- 我们可以清除掉它！

# 更新《Supernauts》的导航网格

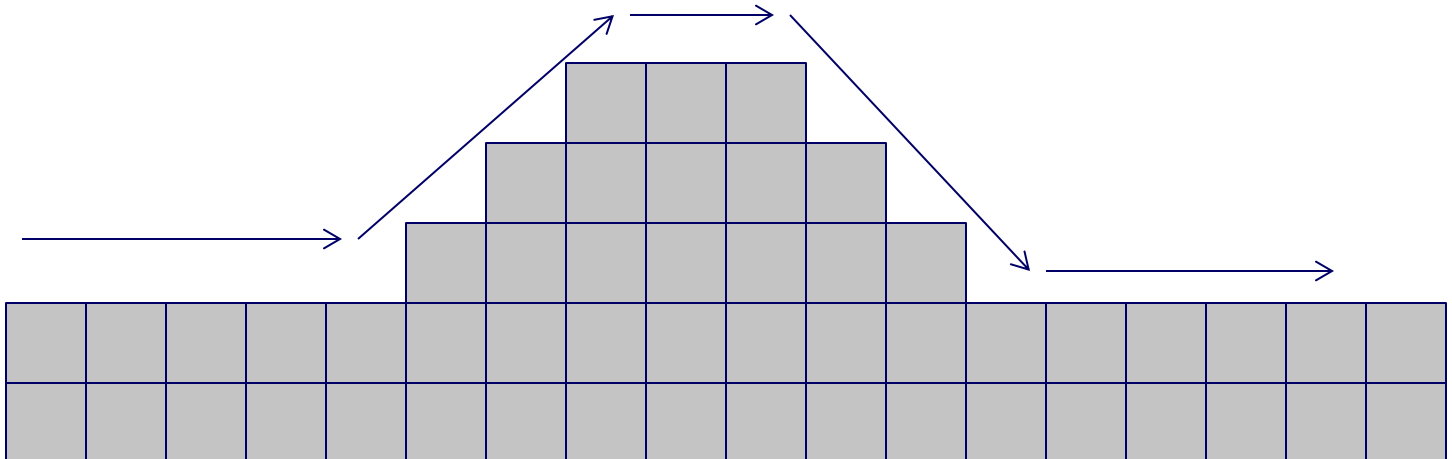


建造

连接

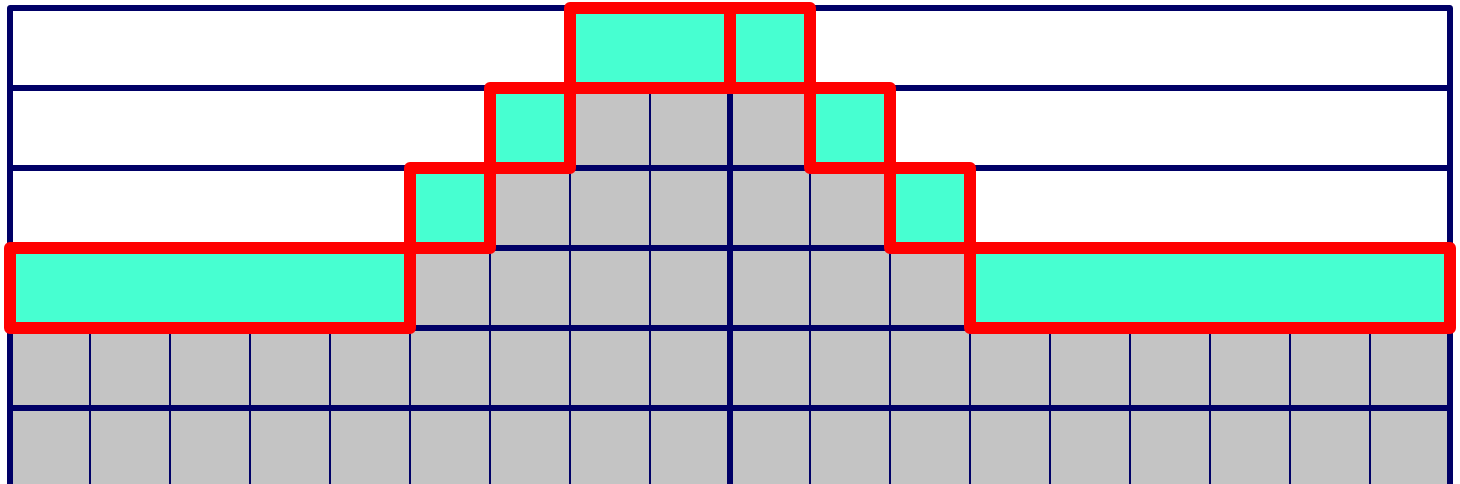
# 改进《Supernauts》的导航网格

- 在《Supernauts》中角色们可以沿着楼梯上下走动。
- 这可以用来减少节点数目



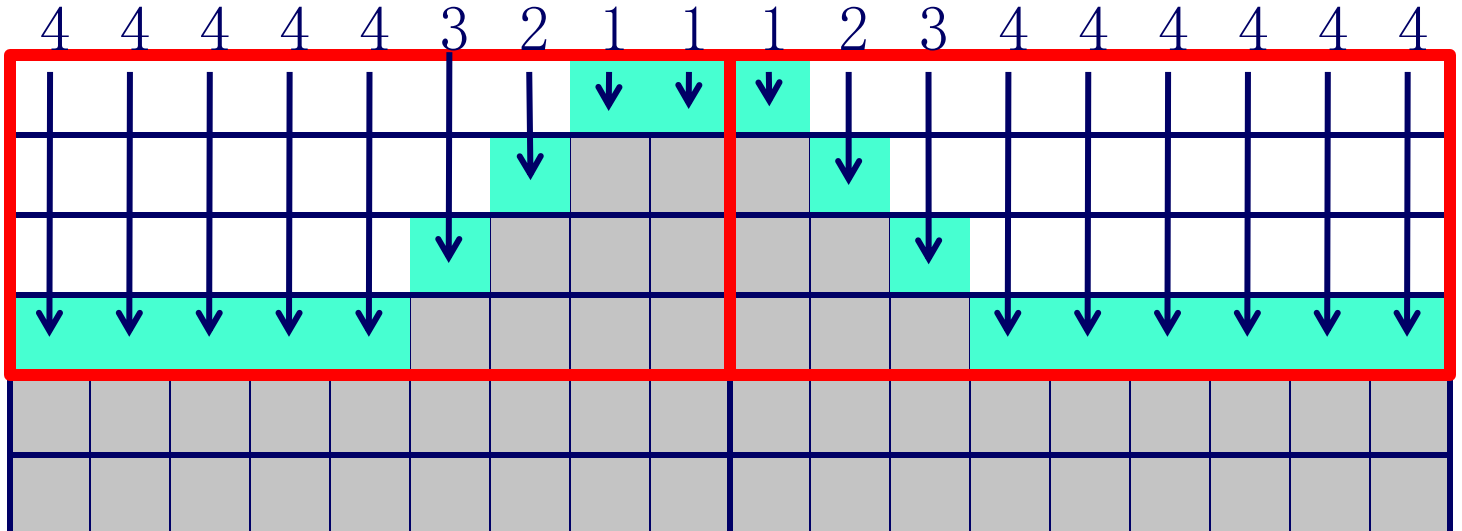
# 改进《Supernauts》的导航网格

- -> 8个节点
- 角色**可以**沿着节点**自由移**动。
- 我们是否能够用任何方法将之合并？



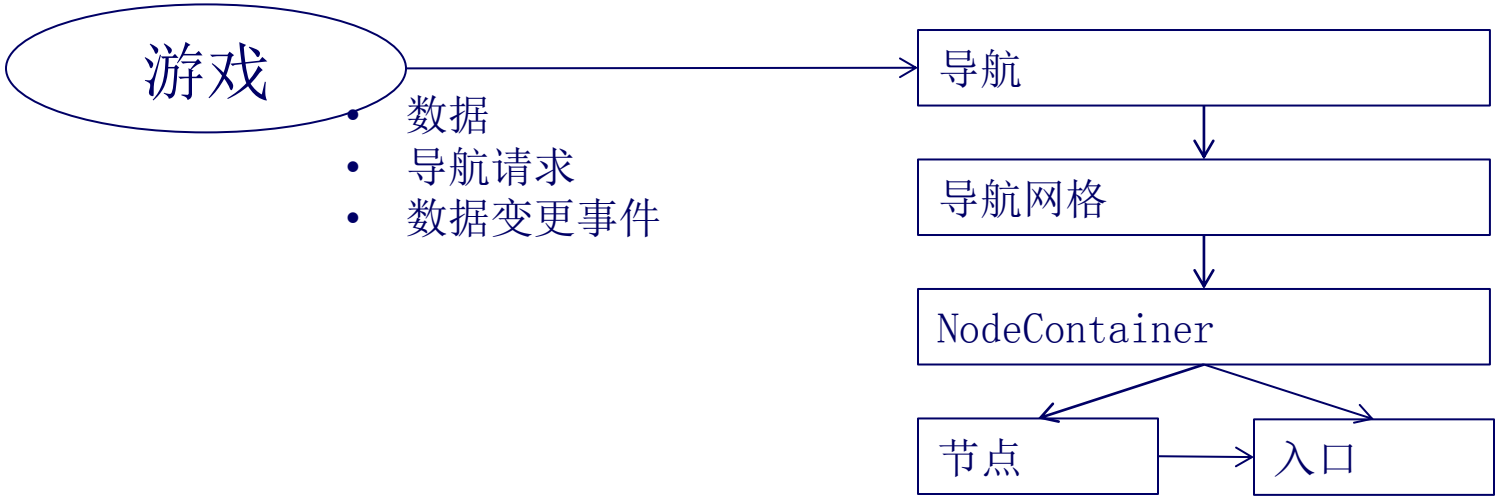
# 改进《Supernauts》的导航网格

- 多个迷你导航网格共享相同的节点
- 每个节点的**高度场（height field）**可以用来帮助区分重叠的节点。



# 实际导航网格展示视频

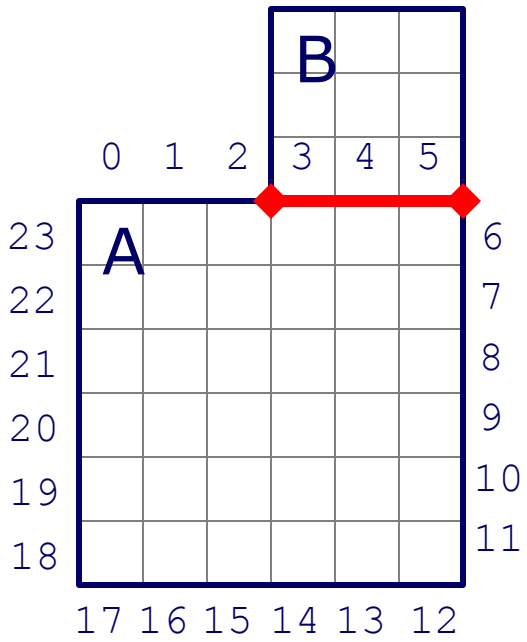
# 示例实现



```
class Node {
    int id;
    IntBounds bounds;
    int [,] heightField;
    List <Portal> connections;
};

class Portal {
    int positionA, positionB;
    int nodeA, nodeB;
};
```

# 节点 & 入口




```
new Portal {
    positionA = 3,
    positionB = 6,
    nodeA = A,
    nodeB = B
};
```

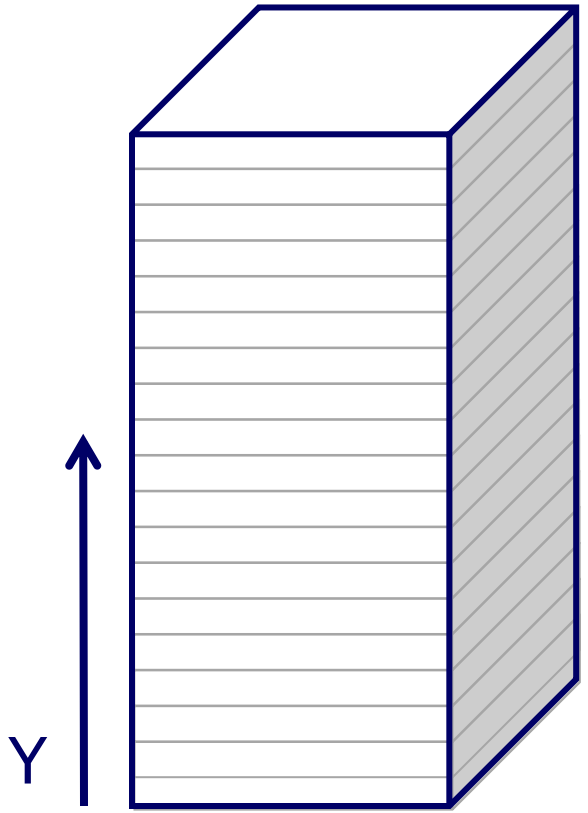
```
class NavMesh {
    private NodeContainer [,] nodeLookUp;

    public Node GetAndCreateNode(Vector3 point);
    public Node GetNode(int id);
};

class NodeContainer {
    private int [,,] positionToNode;
    private State [] sliceStates;
    private HashSet<int> [] nodeSetsPerSlice;
    private Dictionary<int, Node> nodesIndexed;
    // ...
};
```

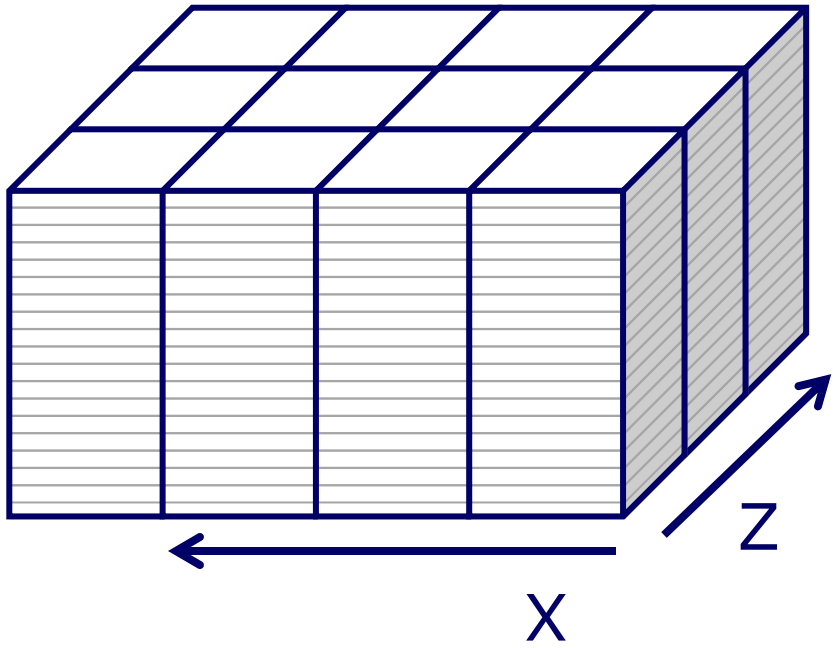
# NodeContainer

```
class NodeContainer {
    private int [,,] positionToNode;
    private State [] sliceStates;
    private HashSet<int> [] nodeSetsPerSlice;
    private Dictionary<int, Node> nodesIndexed;
    // ...
};
```



# 导航网格

```
NodeContainer [,] nodeLookUp;
```



```
class Navigation {
    private NavMesh navMesh;

    public Path FindPath(Vector3 start, Vector3 target);
    public Path FindPath(Vector3 start, Bounds area);

    public bool PathExists(Vector3 start, Vector3 target);

    public Vector3 GetRandomPosition(Bounds area);
    public Vector3 GetRandomReachablePosition(Vector3 point);

    public void Invalidate(Bounds area);
};
```

```
public Path FindPath(Vector3 start, Vector3 target);
public Path FindPath(Vector3 start, Bounds area)
```

- 目标地不一定必须是一个点，也可以是一个区域。
- 在进入边界时简单地结束算法。

```
public bool PathExists(Vector3 start, Vector3 target);
public bool PathExists(Vector3 start, Vector3 target);
```

- 路径存在时，简单查询比路径寻找要快得多，这是因为你不需要自己生成路径。

```
public Vector3 GetRandomPosition(Bounds area);
public Vector3 GetRandomReachablePosition(Vector3 point);
```

- 你也可以轻松地获得**均匀分布**的随机位置。
- 简单地利用节点表面积对随机选择进行加权计算。

# 总结

- 每个游戏都有自身独特的需求。
- 复杂的问题可以通过以下方法变得简单
  - 拆分成小问题
  - 反复发展完善解决方法而不是在最开始就研究最高级的解决方案
- 避免工作超出你的游戏的实际需要！

# 谢谢！

- 问答环节
- harri.hatinen@grandcrugames.com