从零开始写一个武侠冒险游戏-3-地图生成

- 作者:FreeBlues
- 修订记录
  - 2016.06.13 初稿完成.
  - 2016.08.03 增加对 `xCode` 项目文件的说明.

概述

前面两章我们设计了角色的状态, 绘制出了角色, 并且赋予角色动作, 现在是时候为我们的角色创造一个舞台了, 那就是游戏地图(我们目前做的是一个 `2D` 游戏, 因此叫 `地图` , 如果是 `3D` , 则叫 `地形`).

地图生成也是游戏开发的一项基本技术, 涉及到方方面面的技能, 而且地图的数据结构要考虑到游戏里的其他景物跟角色的显示和交互, 对于整个游戏程序的效率起着决定性的影响, 不过我们这里先解决有没有的问题, 目标不太高, 能流畅运行就可以了.

最简原型

跟我们一向提倡的大思路一致, 一切从简出发, 先弄个原型跑起来再说.

> 经验之谈: 很多开发过程中的难题都是因为我们一开始就引入了过于复杂的问题, 制定了太大的目标, 试图一开始就把方方面面都考虑到, 结果无形中就增加了难度, 不得不承认, 这种顶层设计的思路是不太符合事物发展的规律的, 也不符合生物的进化规律, 所以实现起来就比较困难, 如果我们遵循从简单到复杂, 从原型到成品的开发思路, 就会发现开发过程变得顺利很多.

游戏地图原理

简单说来, 游戏地图有两个层面, 一个是显示到屏幕上的图形图像, 一个是隐藏在图像后面的数据结构, 前者是游戏跟玩家交互的界面, 后者是游戏中绘制出来的各种对象跟程序交互的接口.

比如玩家操纵一个游戏角色从左边一个位置走到右边一个位置, 玩家看到的是屏幕上角色的移动过程, 而程序在后面要记录玩家每时每刻的坐标, 以及该坐标在地图上对应的位置.

如果玩家看到地图上某个位置有一个可以操作的物体, 比如一个箱子, 玩家的角色想要靠近这个箱子然后打开它, 那么后台的地图数据库里首先要在地图的某个位置上有一个箱子, 然后再判断角色距离箱子的距离, 如果小于某个值, 那么就说明允许操作, 玩家开过箱子后, 还要把箱子的当前状态(已开启)再写回到数据库里, 等等诸如此类.

最简单的地图

最简单的地图就是一张事先画好的图, 角色在这张图上移来移去, 这个功能我们在第2章就已经实现了, 但是按照这种方法实现的地图角色很难跟地图上的物体进行交互, 而且使用事先画好的图做地图还有一个问题就是如果整个游戏场景比较大的话就需要很多画预先存储到游戏中, 这样会导致较大的体积.

所以, 我们采取另一种做法, 因为游戏场景中很多物体对象都是可以重复使用的, 比如树木, 岩石等等, 所以我们可以把这些基本对象提取出来事先绘制好, 或者使用预先做好的素材, 这样我们需要事先存储的内容就大大减少了, 然后再根据实际需要动态绘制上去, 这就是随机生成场景地图的做法.

恰好我之前写过一个简单的随机地图生成器, 虽然比较简陋, 不过为了减少工作量, 还是可以拿来用用的, 当然, 直接用是不行的, 主要是以它做一个基础来进行改写.

原型目标

首先明确一下我们这个地图原型的基本需求点:

- 可以灵活调整地图大小
- 可以随机插入 `树木` / `矿物` / `建筑` 等固定物体
- 角色可以跟地图上的这些物体交互

这是三个最基本的需求, 我们一步一步来实现这三个需求.

格子地面地图

综合性能和实现难度方面的考虑, 我们的地图以网格的形式进行绘制和保存, 也就是以我们之前写好的那个随机地图生成器为基本原型, 这样一方面可以灵活控制数据表的大小, 数据表中存储的最小单位就是一个预先设定好大小的格子, 另一方面写起来也比较简单, 还有不错的效率表现.

首先确定我们的初始化参数和数据结构, 用这个函数来实现:

```
function initParams()
    print("Simple Map Sample!!")
    textMode(CORNER)
    spriteMode(CORNER)

    --[[
    gridCount：网格数目，范围：1~100，例如，设为3则生成3*3的地图，设为100，则生成100*100的地图。
    scaleX：单位网格大小比例，范围：1~100，该值越小，则单位网格越小；该值越大，则单位网格越大。
    scaleY：同上，若与scaleX相同则单位网格是正方形格子。
    plantSeed：植物生成几率，范围:大于4的数，该值越小，生成的植物越多；该值越大，生成的植物越少。
    minerialSeed：矿物生成几率，范围:大于3的数，该值越小，生成的矿物越多；该值越大，生成的矿物越少。
    --]]
    gridCount = 50
    scaleX = 50
```

```
    scaleY = 50
    plantSeed = 20.0
    minerialSeed = 50.0

    -- 根据地图大小申请图像
    local w,h = (gridCount+1)*scaleX, (gridCount+1)*scaleY
    imgMap = image(w,h)

    -- 整个地图使用的全局数据表
    mapTable = {}

    -- 设置物体名称
    tree1,tree2,tree3 = "松树", "杨树", "小草"
    mine1,mine2 = "铁矿", "铜矿"

    -- 设置物体图像
    imgTree1 = readImage("Planet Cute:Tree Short")
    imgTree2 = readImage("Planet Cute:Tree Tall")
    imgTree3 = readImage("Platformer Art:Grass")
    imgMine1 = readImage("Platformer Art:Mushroom")
    imgMine2 = readImage("Small World:Treasure")

    -- 存放物体：名称，图像
    itemTable = {[tree1]=imgTree1,[tree2]=imgTree2,[tree3]=imgTree3,[mine1]=imgMin
e1,[mine2]=imgMine2}

    -- 3*3
    mapTable = {{pos=vec2(1,1),plant=nil,mineral=mine1},{pos=vec2(1,2),plant=nil,m
ineral=nil},
                {pos=vec2(1,3),plant=tree3,mineral=nil},{pos=vec2(2,1),plant=tree1
,mineral=nil},
                {pos=vec2(2,2),plant=tree2,mineral=mine2},{pos=vec2(2,3),plant=nil
,mineral=nil},
                {pos=vec2(3,1),plant=nil,mineral=nil},{pos=vec2(3,2),plant=nil,min
eral=mine2},
                {pos=vec2(3,3),plant=tree3,mineral=nil}}

end
```

接下来是绘制地面单位格子的函数, 现在是在每个格子上绘制一个矩形, 参数 `position` 是一个二维向量, 形如 `vec(1,2)` 则表示该格子位于第 `1` 行, 第 `2` 列, 代码如下:

```
-- 绘制单位格子地面
function drawUnitGround(position)
    local x,y = scaleX * position.x, scaleY * position.y
    pushMatrix()
    stroke(99, 94, 94, 255)
    -- 网格线宽度
    strokeWidth(1)
    -- 地面颜色
    fill(5,155,40,255)
    -- fill(5,155,240,255)
    rect(x,y,scaleX,scaleY)
    popMatrix()
end
```

用这两个函数来调用它:

```
-- 新建地图数据表, 插入地图上每个格子里的物体数据
function createMapTable()
    for i=1,gridCount,1 do
        for j=1,gridCount,1 do
            mapItem = {pos=vec2(i,j), plant=nil, mineral=nil}
            table.insert(mapTable, mapItem)
        end
    end
    updateMap()
end

-- 更新地图
function updateMap()
    setContext(imgMap)
    for i = 1,gridCount*gridCount,1 do
        local pos = mapTable[i].pos
        -- 绘制地面
        drawUnitGround(pos)
    end
    setContext()
end

-- 绘制地图
function drawMap()
    -- 绘制地图
    sprite(imgMap,-scaleX,-scaleY)
end
```

最基本原型的完整代码

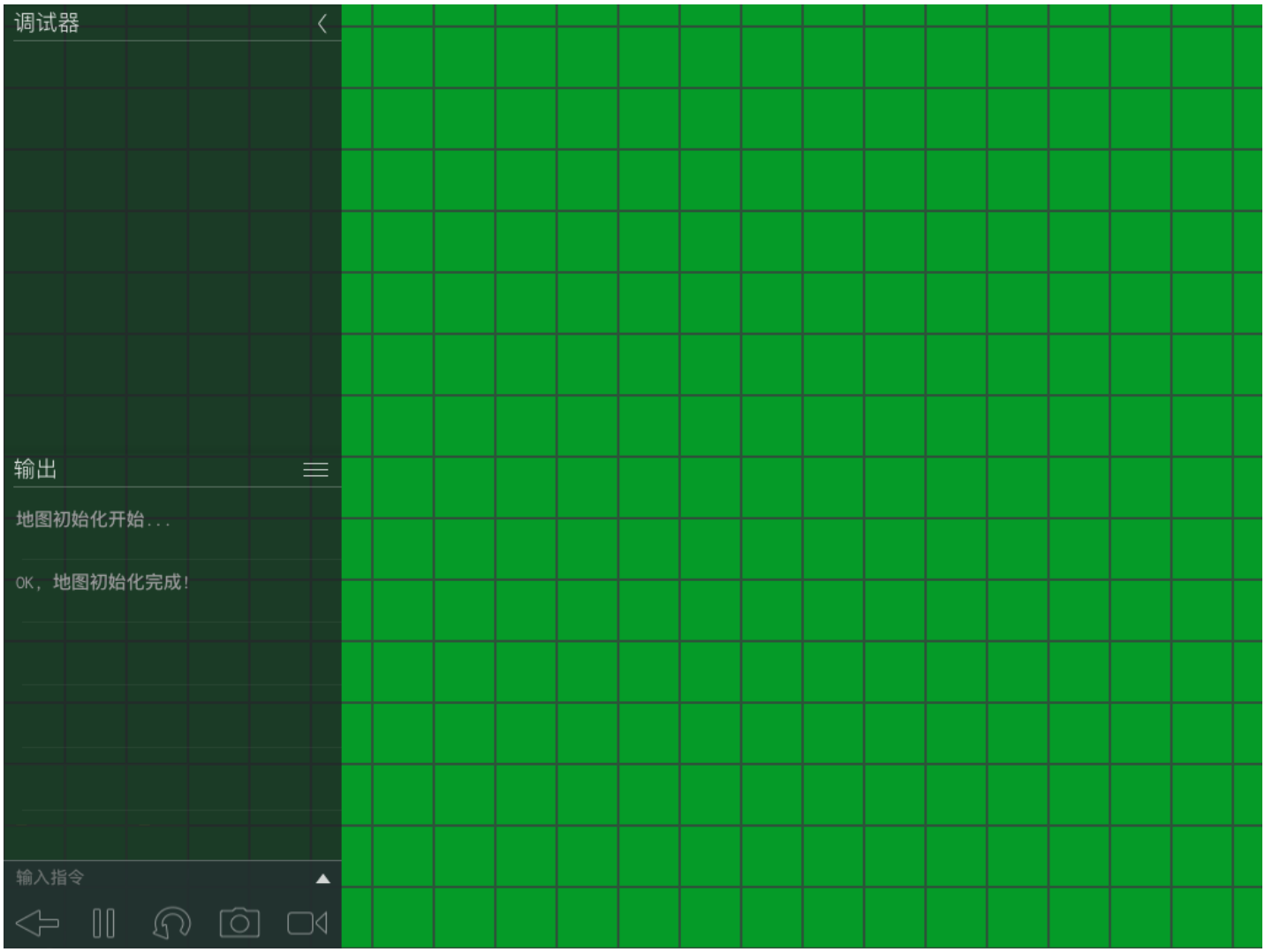
下面我们把实现这个最基本原型的完整代码列出来:

```
-- MapSample

-- 初始化地图参数
function initParams()
    print("地图初始化开始...")
    textMode(CORNER)
    spriteMode(CORNER)

    --[[ 参数说明:
    gridCount：网格数目，范围：1~100，例如，设为3则生成3*3的地图，设为100，则生成100*100的地图。
    scaleX：单位网格大小比例，范围：1~100，该值越小，则单位网格越小；该值越大，则单位网格越大。
    scaleY：同上，若与scaleX相同则单位网格是正方形格子。
    plantSeed：植物生成几率，范围:大于4的数，该值越小，生成的植物越多；该值越大，生成的植物越少。
    minerialSeed：矿物生成几率，范围:大于3的数，该值越小，生成的矿物越多；该值越大，生成的矿物越少。
    --]]
    gridCount = 50
    scaleX = 50
    scaleY = 50
    plantSeed = 20.0
    minerialSeed = 50.0

    -- 根据地图大小申请图像
    local w,h = (gridCount+1)*scaleX, (gridCount+1)*scaleY
    imgMap = image(w,h)

    -- 整个地图使用的全局数据表
    mapTable = {}

    -- 设置物体名称
    tree1,tree2,tree3 = "松树", "杨树", "小草"
    mine1,mine2 = "铁矿", "铜矿"

    -- 设置物体图像
    imgTree1 = readImage("Planet Cute:Tree Short")
    imgTree2 = readImage("Planet Cute:Tree Tall")
    imgTree3 = readImage("Platformer Art:Grass")
    imgMine1 = readImage("Platformer Art:Mushroom")
    imgMine2 = readImage("Small World:Treasure")

    -- 存放物体：名称，图像
    itemTable = {[tree1]=imgTree1,[tree2]=imgTree2,[tree3]=imgTree3,[mine1]=imgMine1,[mine2]=imgMine2}

    -- 3*3
    mapTable = {{pos=vec2(1,1),plant=nil,mineral=mine1},{pos=vec2(1,2),plant=nil,m
```

```
ineral=nil},
                {pos=vec2(1,3),plant=tree3,mineral=nil},{pos=vec2(2,1),plant=tree1
,mineral=nil},
                {pos=vec2(2,2),plant=tree2,mineral=mine2},{pos=vec2(2,3),plant=nil
,mineral=nil},
                {pos=vec2(3,1),plant=nil,mineral=nil},{pos=vec2(3,2),plant=nil,min
eral=mine2},
                {pos=vec2(3,3),plant=tree3,mineral=nil}}

end

-- 新建地图数据表，插入地图上每个格子里的物体数据
function createMapTable()
    for i=1,gridCount,1 do
        for j=1,gridCount,1 do
            mapItem = {pos=vec2(i,j), plant=nil, mineral=nil}
            table.insert(mapTable, mapItem)
        end
    end
    updateMap()
end

-- 跟据地图数据表，刷新地图
function updateMap()
    setContext(imgMap)
    for i = 1,gridCount*gridCount,1 do
        local pos = mapTable[i].pos
        -- 绘制地面
        drawUnitGround(pos)
    end
    setContext()
end

-- 绘制单位格子地面
function drawUnitGround(position)
    local x,y = scaleX * position.x, scaleY * position.y
    pushMatrix()
    stroke(99, 94, 94, 255)
    -- 网格线宽度
    strokeWidth(1)
    -- 地面颜色
    fill(5,155,40,255)
    -- fill(5,155,240,255)
    rect(x,y,scaleX,scaleY)
    popMatrix()
end

-- 游戏主程序框架
```

```
function setup()
    displayMode(OVERLAY)

    initParams()
end

function draw()
    background(40, 40, 50)

    -- 绘制地图
    drawMap()
end
```

看看截图:



很好, 基本的格子地图写好了, 接着我们来解决在格子地图上随机插入 `树木` / `矿物` / `建筑` 等固定物体的功能.

插入物体

因为我们已经在设计数据表时就考虑到了要插入固定物体, 所以现在需要做的就是写几个相关的函数, 首先是两个随机选取物体名字的函数:

```
-- 随机生成植物
function randomPlant()
    local seed = math.random(1.0, plantSeed)
    local result = nil

    if seed >= 1 and seed < 2 then result = tree1
    elseif seed >= 2 and seed < 3 then result = tree2
    elseif seed >= 3 and seed < 4 then result = tree3
    elseif seed >= 4 and seed <= plantSeed then result = nil end

    -- 返回随机选取的物体名字
    return result
end

-- 随机生成矿物
function randomMinerial()
    local seed = math.random(1.0, minerialSeed)
    local result = nil

    if seed >= 1 and seed < 2 then result = mine1
    elseif seed >= 2 and seed < 3 then result = mine2
    elseif seed >= 3 and seed <= minerialSeed then result = nil end

    -- 返回随机选取的物体名字
    return result
end
```

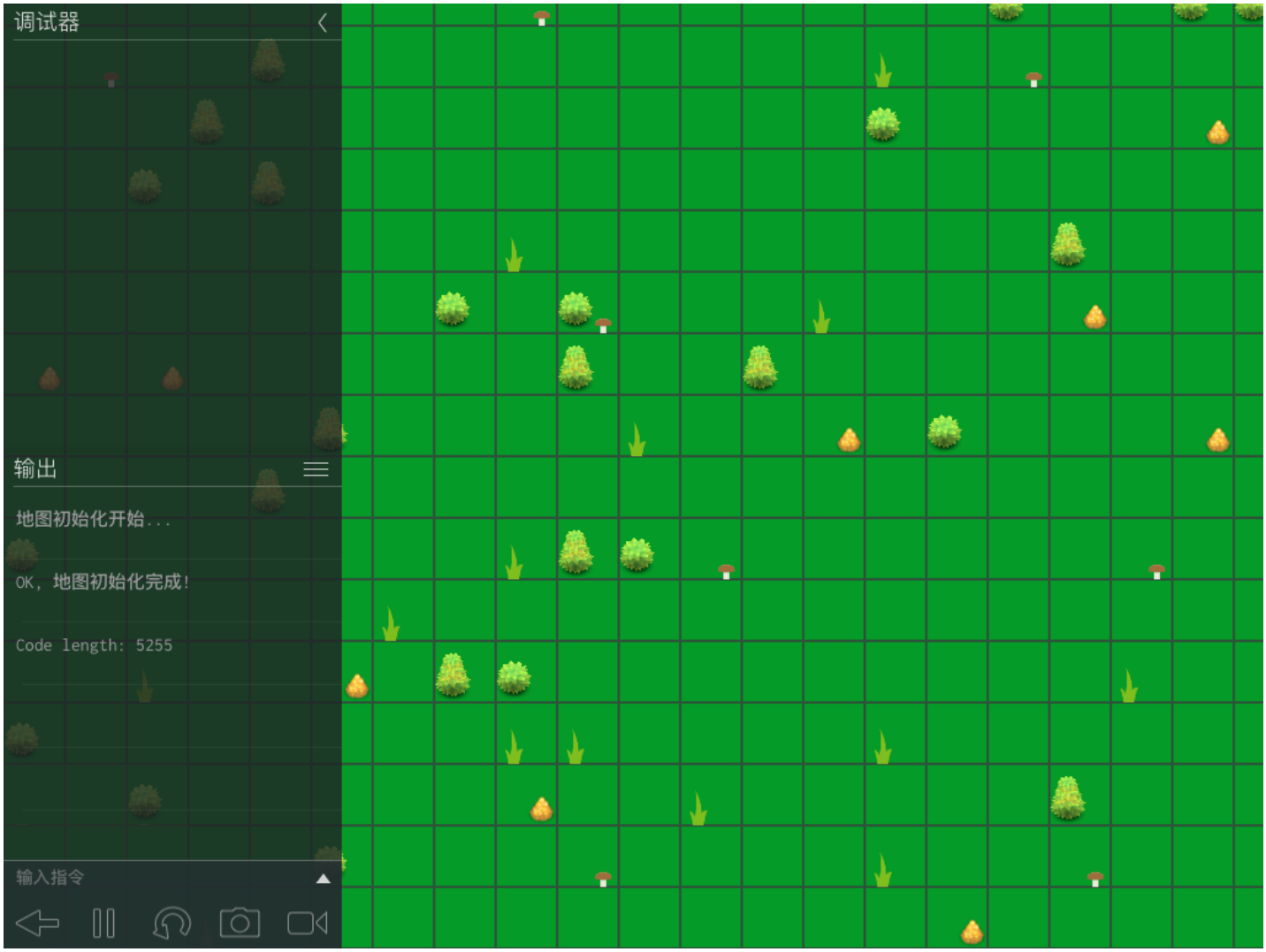
然后增加两个绘制函数, 来绘制出物体的图像:

```
-- 绘制单位格子内的植物
function drawUnitTree(position,plant)
    local x,y = scaleX * position.x, scaleY * position.y
    pushMatrix()
    -- 绘制植物图像
    sprite(itemTable[plant], x, y, scaleX*6/10,scaleY)

    --fill(100,100,200,255)
    --text(plant,x,y)
    popMatrix()
end

-- 绘制单位格子内的矿物
function drawUnitMineral(position,mineral)
    local x,y = scaleX * position.x, scaleY * position.y
    pushMatrix()
    -- 绘制矿物图像
    sprite(itemTable[mineral], x+scaleX/2, y, scaleX/2, scaleX/2)

    --fill(100,100,200,255)
    --text(mineral,x+scaleX/2,y)
    popMatrix()
end
```

最后需要修改函数 `createMapTable()` 和 `updateMap()`, 在其中增加对 `plant` 和 `mineral` 的处理, 修改后的代码如下:

```
-- 新建地图数据表，插入地图上每个格子里的物体数据，目前为 plant  和 mineral 为空
function createMapTable()
    --local mapTable = {}
    for i=1,gridCount,1 do
        for j=1,gridCount,1 do
            mapItem = {pos=vec2(i,j), plant=randomPlant(), mineral=randomMinerial(
)}
            --mapItem = {pos=vec2(i,j), plant=nil, mineral=nil}
            table.insert(mapTable, mapItem)
        end
    end
    updateMap()
end

-- 跟据地图数据表，刷新地图
function updateMap()
    setContext(imgMap)
    for i = 1,gridCount*gridCount,1 do
        local pos = mapTable[i].pos
        local plant = mapTable[i].plant
        local mineral = mapTable[i].mineral
        -- 绘制地面
        drawUnitGround(pos)
        -- 绘制植物和矿物
        if plant ~= nil then drawUnitTree(pos, plant) end
        if mineral ~= nil then drawUnitMineral(pos, mineral) end
    end
    setContext()
end
```

非常好, 第二个基本目标也完成了, 截个图:



看看现在的截图效果, 是不是感觉我们的原型正在一步步走向完善? 紧接着就要想办法实现角色跟地图上物体的交互了, 想做到这一点, 首先需要建立角色跟地图在地图数据表中的数据关联.

建立角色跟地图的关联

现在地图绘制好了, 角色也可以自由地在地图上活动了, 不过这只是我们看到的表面现象, 实际在隐藏于屏幕后面的程序代码中, 角色的位置跟地图的坐标(方格)并没有建立任何关联.

例如, 角色在地图上看到一棵树, 他想要对这棵树做一些动作(`观察` / `浇水` / `砍伐` 等)进行交互, 如果角色选择了砍伐树, 那么最终树被砍倒之后我们还需要更新地图数据表, 把对应位置的树的图片更换成树根, 而实现角色跟树的交互, 就需要根据角色位置坐标跟树的位置坐标进行判断.

我们知道树的位置坐标已经保存在地图的数据表中了, 但是角色的坐标跟地图的数据表还没有任何关系, 因为角色经常移动, 所以我们可以写一个函数, 根据角色的屏幕像素点坐标来计算所处的地图方格坐标, 代码如下:

```
-- 根据像素坐标值计算所处网格的 i,j 值
function where(x,y)
    local i = math.ceil((x+scaleX) / scaleX)
    local j = math.ceil((y+scaleY) / scaleY)
    return i,j
end
```

有了这个函数, 我们只要把角色当前位置的像素点坐标输入, 就可以得到它所处网格的坐标, 这样就把角色跟地图从数据层面建立了关联. 后续就可以方便地通过这个接口来处理他们之间的交互了.

为方便后续代码维护, 我们要把上述代码改写为一个 `地图生成` 类, 改写后的完整代码如下:

```
-- MapSample

Maps = class()

function Maps:init()
    --[[
    gridCount：网格数目，范围：1~100，例如，设为3则生成3*3的地图，设为100，则生成100*100的地图。
    scaleX：单位网格大小比例，范围：1~100，该值越小，则单位网格越小；该值越大，则单位网格越大。
    scaleY：同上，若与scaleX相同则单位网格是正方形格子。
    plantSeed：植物生成几率，范围:大于4的数，该值越小，生成的植物越多；该值越大，生成的植物越少。
    minerialSeed：矿物生成几率，范围:大于3的数，该值越小，生成的矿物越多；该值越大，生成的矿物越少。
    --]]
    self.gridCount = 50
    self.scaleX = 50
    self.scaleY = 50
    self.plantSeed = 20.0
    self.minerialSeed = 50.0

    -- 根据地图大小申请图像
    local w,h = (self.gridCount+1)*self.scaleX, (self.gridCount+1)*self.scaleY
    self.imgMap = image(w,h)

    -- 整个地图使用的全局数据表
    self.mapTable = {}

    -- 设置物体名称
    tree1,tree2,tree3 = "松树", "杨树", "小草"
    mine1,mine2 = "铁矿", "铜矿"

    -- 设置物体图像
    imgTree1 = readImage("Planet Cute:Tree Short")
    imgTree2 = readImage("Planet Cute:Tree Tall")
```

```
    imgTree3 = readImage("Platformer Art:Grass")
    imgMine1 = readImage("Platformer Art:Mushroom")
    imgMine2 = readImage("Small World:Treasure")

    -- 存放物体：名称，图像
    self.itemTable = {[tree1]=imgTree1,[tree2]=imgTree2,[tree3]=imgTree3,[mine1]=i
mgMine1,[mine2]=imgMine2}

    -- 尺寸为 3*3 的数据表示例
    self.mapTable = {{pos=vec2(1,1),plant=nil,mineral=mine1},{pos=vec2(1,2),plant=
nil,mineral=nil},
                {pos=vec2(1,3),plant=tree3,mineral=nil},{pos=vec2(2,1),plant=tree1
,mineral=nil},
                {pos=vec2(2,2),plant=tree2,mineral=mine2},{pos=vec2(2,3),plant=nil
,mineral=nil},
                {pos=vec2(3,1),plant=nil,mineral=nil},{pos=vec2(3,2),plant=nil,min
eral=mine2},
                {pos=vec2(3,3),plant=tree3,mineral=nil}}


    print("地图初始化开始...")
    -- 根据初始参数值新建地图
    self:createMapTable()
    print("OK, 地图初始化完成！")
end

-- 新建地图数据表，插入地图上每个格子里的物体数据
function Maps:createMapTable()
    --local mapTable = {}
    for i=1,self.gridCount,1 do
        for j=1,self.gridCount,1 do
            self.mapItem = {pos=vec2(i,j), plant=self:randomPlant(), mineral=self:
randomMinerial()}
            --self.mapItem = {pos=vec2(i,j), plant=nil, mineral=nil}
            table.insert(self.mapTable, self.mapItem)
        end
    end
    self:updateMap()
end

-- 根据地图数据表，刷新地图
function Maps:updateMap()
    setContext(self.imgMap)
    for i = 1,self.gridCount*self.gridCount,1 do
        local pos = self.mapTable[i].pos
        local plant = self.mapTable[i].plant
        local mineral = self.mapTable[i].mineral
        -- 绘制地面
```

```
        self:drawGround(pos)
        -- 绘制植物和矿物
        if plant ~= nil then self:drawTree(pos, plant) end
        if mineral ~= nil then self:drawMineral(pos, mineral) end
    end
    setContext()
end

function Maps:drawMap()
    sprite(self.imgMap,-self.scaleX,-self.scaleY)
end

-- 根据像素坐标值计算所处网格的 i,j 值
function Maps.where(x,y)
    local i = math.ceil((x+self.scaleX) / self.scaleX)
    local j = math.ceil((y+self.scaleY) / self.scaleY)
    return i,j
end

-- 随机生成植物
function Maps:randomPlant()
    local seed = math.random(1.0, self.plantSeed)
    local result = nil

    if seed >= 1 and seed < 2 then result = tree1
    elseif seed >= 2 and seed < 3 then result = tree2
    elseif seed >= 3 and seed < 4 then result = tree3
    elseif seed >= 4 and seed <= self.plantSeed then result = nil end

    return result
end

-- 随机生成矿物
function Maps:randomMinerial()
    local seed = math.random(1.0, self.minerialSeed)
    local result = nil

    if seed >= 1 and seed < 2 then result = mine1
    elseif seed >= 2 and seed < 3 then result = mine2
    elseif seed >= 3 and seed <= self.minerialSeed then result = nil end

    return result
end

function Maps:getImg(name)
    return self.itemTable[name]
end
```

```
-- 重置
function Maps:resetMapTable()
    self.mapTable = self:createMapTable()
end

-- 绘制单位格子地面
function Maps:drawGround(position)
    local x,y = self.scaleX * position.x, self.scaleY * position.y
    pushMatrix()
    stroke(99, 94, 94, 255)
    strokeWidth(1)
    fill(5,155,40,255)
    -- fill(5,155,240,255)
    rect(x,y,self.scaleX,self.scaleY)
    --sprite("Documents:3D-Wall",x,y,scaleX,scaleY)
    popMatrix()
end

-- 绘制单位格子内的植物
function Maps:drawTree(position,plant)
    local x,y = self.scaleX * position.x, self.scaleY * position.y
    pushMatrix()
    -- 绘制植物图像
    sprite(self.itemTable[plant],x,y,self.scaleX*6/10,self.scaleY)

    --fill(100,100,200,255)
    --text(plant,x,y)
    popMatrix()
end

-- 绘制单位格子内的矿物
function Maps:drawMineral(position,mineral)
    local x,y = self.scaleX * position.x, self.scaleY * position.y
    pushMatrix()
    -- 绘制矿物图像
    sprite(self.itemTable[mineral],x+self.scaleX/2,y,self.scaleX/2,self.scaleX/2)

    --fill(100,100,200,255)
    --text(mineral,x+self.scaleX/2,y)
    popMatrix()
end

-- 游戏主程序框架
function setup()
    displayMode(OVERLAY)

    myMap = Maps()
end
```

```
function draw()
    background(40, 40, 50)

    -- 绘制地图
    myMap:drawMap()
end
```

到目前为止, 我们在 `地图生成原型` 章节的目标基本完成, 下一章我们会尝试把 `状态` , `帧动画` 和 `地图生成` 这三个模块整合起来, 一般来说事物发展到 `三` 的阶段会由量变触发质变, 我们这个程序也一样, 会在这次整合之后, 从一个个零散简陋的原型, 一跃而成一个还能看得过去的基本框架, 是不是很期待?

激动人心的新起点

事实上, 把角色屏幕位置跟地图数据表建立关联之后, 我们的角色就真正存在于这个游戏世界中了, 它可以自由地跟地图上的每一个物体进行交互, 这意味着一个全新的激动人心的开始! 到现在为止, 我们游戏世界的基本框架已经搭建起来了, 我们可以在这个框架上试验自己对于武侠冒险游戏的各种新想法.

所有章节链接

Github项目地址

Github项目地址, 源代码放在 `src/` 目录下, 图片素材放在 `assets/` 目录下, `XCode` 项目文件放在 `MyAdventureGame` 目录下, 整个项目文件结构如下:

```
Air:Write-A-Adventure-Game-From-Zero admin$ tree
.
├── MyAdventureGame
|   ├── Assets
|   |   ├── ...
|   ├── Libs
|   |   ├── ...
|   ├── MyAdventureGame
|   |   ├──...
|   ├── MyAdventureGame.codea
|   |   ├──...
|   ├── MyAdventureGame.xcodeproj
|   |   ├──...
|   └── libversion
├── README.md
├── Vim 列编辑功能详细讲解.md
├── assets
|   ├── ...
|   └── runner.png
├── src
|   ├── c01.lua
|   ├── c02.lua
|   ├── c03.lua
|   ├── c04.lua
|   ├── c05.lua
|   ├── c06-01.lua
|   ├── c06-02.lua
|   ├── c06-03.lua
|   └── c06.lua
├── 从零开始写一个武侠冒险游戏-0-开发框架Codea简介.md
├── 从零开始写一个武侠冒险游戏-1-状态原型.md
├── 从零开始写一个武侠冒险游戏-2-帧动画.md
├── 从零开始写一个武侠冒险游戏-3-地图生成.md
├── 从零开始写一个武侠冒险游戏-4-第一次整合.md
├── 从零开始写一个武侠冒险游戏-5-使用协程.md
├── 从零开始写一个武侠冒险游戏-6-用GPU提升性能(1).md
├── 从零开始写一个武侠冒险游戏-6-用GPU提升性能(2).md
└── 从零开始写一个武侠冒险游戏-6-用GPU提升性能(3).md

2 directories, 26 files
Air:Write-A-Adventure-Game-From-Zero admin$
```