# 第十一章 游戏中的人工智能

# 迷宫寻宝

## 实验步骤

1.使用TiledMap编辑瓦片地图

2.修改命令行版本的A\*算法

3.建立实例场景类MazeScene

4.初始化游戏场景

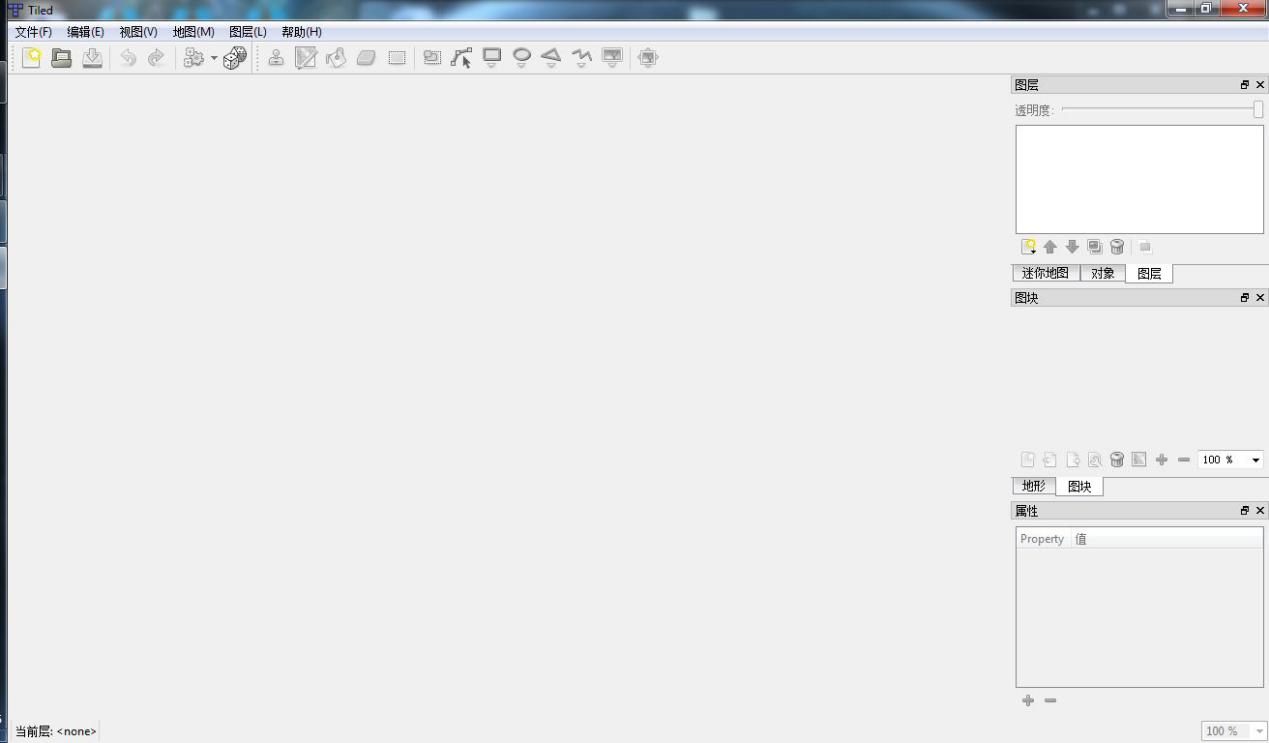
5.设置触摸监听函数

6.添加寻路处理

7.添加定时器更新

### 1.使用TiledMap编辑瓦片地图

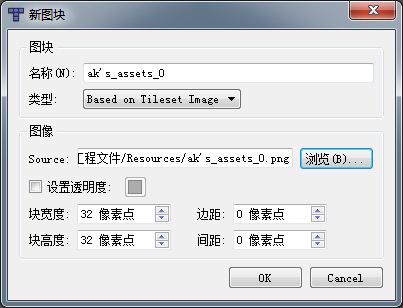
（1）打开TileMap地图编辑器



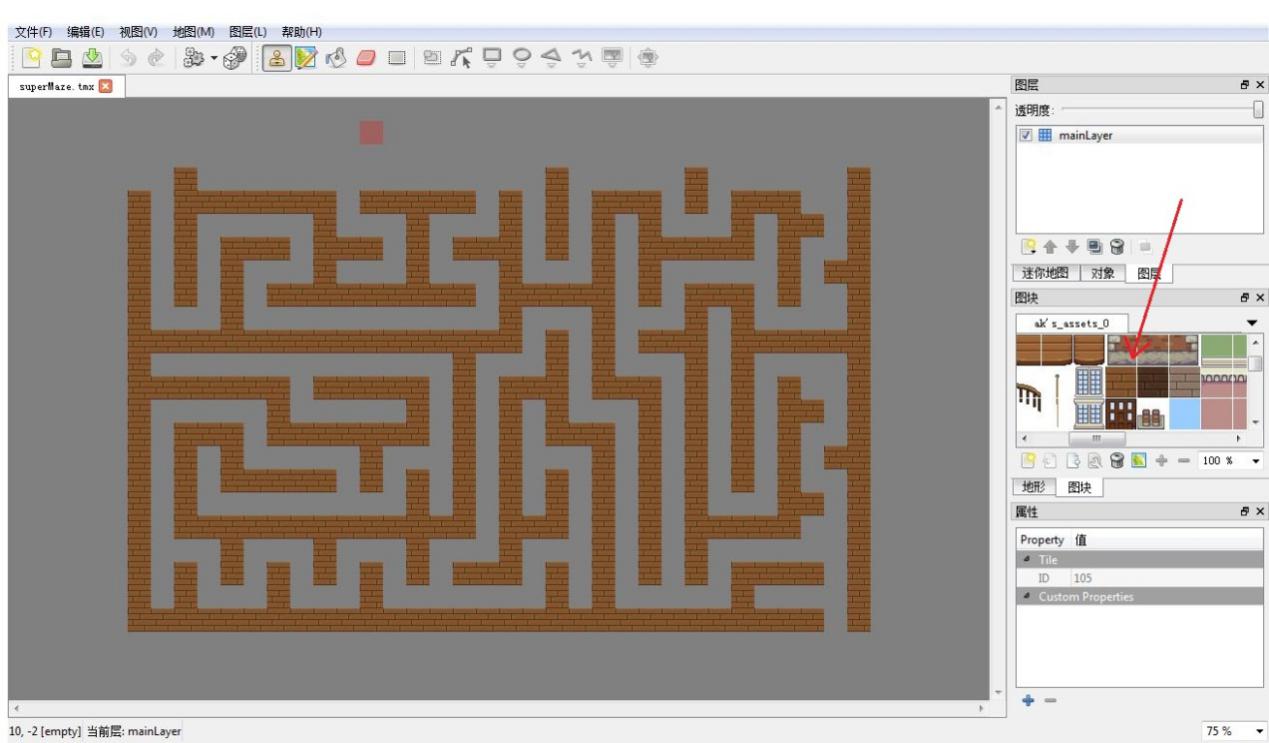
（2）选择文件->新文件，弹出新地图窗口，设置地图大小为宽度：30块，高度：20块，点击OK



（3）选择地图->新图块，弹出新图块窗口，选择地图来源为Resource目录下的ak’s\_assets\_0.png，点击OK



（4）选中构造迷宫所用图块，将其围成需要的迷宫形状



### 2.修改命令行版本的A\*算法

（1）修改计算估价函数的calculateValues函数如下

void calculateValues(mapNode\*\* map, int x, int y, int i, mapNode\* node, mapNode\* destination)

{

//若沿对角线移动，g值增加1.4个单位边长，否则g值增加1个单位边长

if (i == 0 || i == 2 || i == 5 || i == 7)

map[x][y].gValue = node->gValue + 1.4\*UNIT;

else

map[x][y].gValue = node->gValue + UNIT;

//h值为该相邻节点沿水平、竖直方向移动到终点的距离

map[x][y].hValue = (abs(destination->xCoordinate - x) + abs(destination->yCoordinate - y))\*UNIT;

map[x][y].fValue = map[x][y].gValue + map[x][y].hValue;

}

（2）修改父节点发生改变的ifChangeParent函数如下：

bool ifChangeParent(mapNode\*\* map, int x, int y, int i, mapNode \* node)

{

//若沿对角线移动

if (i == 0 || i == 2 || i == 5 || i == 7)

{

//若相邻节点g值大于当前节点g值超过1.4个单位

if (map[x][y].gValue> node->gValue + 1.4\*UNIT)

{

//设置当前节点为父节点

map[x][y].parent = node;

//修改节点的g，f值，h值不变

map[x][y].gValue = node->gValue + 1.4\*UNIT;

map[x][y].fValue = map[x][y].gValue + map[x][y].hValue;

}

return true;

}

//若沿水平、垂直方向移动，同理

else

{

if (map[x][y].gValue > node->gValue + UNIT)

{

map[x][y].parent = node;

map[x][y].gValue = node->gValue + UNIT;

map[x][y].fValue = map[x][y].gValue + map[x][y].hValue;

}

return true;

}

return false;

}

（3）修改检查周围节点的函数checkNeighboringNodes如下：

bool checkNeighboringNodes(mapNode\*\* map, openList\* open, mapNode\* node, mapNode\* destination)

{

//定义当前节点相邻8个方向的坐标变化

const int neighborDirection[8][2] = { { -1, -1 }, { -1, 0 }, { -1, 1 }, { 0, -1 }, { 0, 1 }, { 1, -1 }, { 1, 0 }, { 1, 1 } };

for (int i = 0; i<8; i++)

{

//得到相邻节点的x，y坐标

int neighborX = node->xCoordinate + neighborDirection[i][0];

int neighborY = node->yCoordinate + neighborDirection[i][1];

//若相邻节点位于地图之外、或不可到达、或已在COLSED表中则不考虑

if (neighborX >= 0 && neighborY >= 0 && neighborX <MAP\_WIDTH && neighborY <MAP\_HEIGHT

&&map[neighborX][neighborY].status != NOT\_ACCESS

&&map[neighborX][neighborY].status != IN\_CLOSEDLIST)

{

//若相邻节点已在OPEN表中

if (map[neighborX][neighborY].status == IN\_OPENLIST)

//判断是否有更小的g值，有则改变父节点

ifChangeParent(map, neighborX, neighborY, i, node);

//若相邻节点为可到达

else

{

//将当前节点设为其父节点

map[neighborX][neighborY].parent = node;

//计算该相邻节点的f，g，h值

calculateValues(map, neighborX, neighborY, i, node, destination);

//将该相邻节点插入OPEN表

insertToOpenList(open, &map[neighborX][neighborY]);

}

}

}

return true;

}

（4）修改A\*算法的主函数aStar函数如下：

int MazeScene::aStar(mapNode\*\* map, mapNode\* origin, mapNode\* destination)

{

openList\* open = new openList;

open->next = nullptr;

open->openNode = origin;

closedList\* close = new closedList;

close->next = nullptr;

close->closedNode = nullptr;

//循环检验8个方向的相邻节点

while (checkNeighboringNodes(map, open, open->openNode, destination))

{

//从OPEN表中选取节点插入CLOSED表

insertNodeToClosedList(close, open);

//若OPEN表为空，表明寻路失败

if (open == nullptr)

{

//屏幕中央显示寻路失败的提示

auto failedTip = Label::createWithBMFont("fonts/futura-48.fnt", "No Way!");

failedTip->setPosition(m\_visibleSize.width / 2, m\_visibleSize.height / 2);

this->addChild(failedTip, 2, TIP\_TAG);

break;

}

//若终点在OPEN表中，表明寻路成功

if (open->openNode->status == DESTINATION)

{

mapNode\* tempNode = open->openNode;

//调用moveOnPath（）函数控制精灵在路径上移动

moveOnPath(tempNode);

break;

}

}

return 0;

}

（5）在实例中，地图大小为32\*20，每个单元为30，因此修改宏定义如下：

#define MAP\_WIDTH 32

#define MAP\_HEIGHT 20

#define UNIT 30

### 3.建立实例场景类MazeScene

场景类MazeScene定义的具体代码如下：

class MazeScene : public cocos2d::Layer

{

private:

//瓦片地图的图层对象

TMXLayer\* m\_mapLayer;

//绘制路径的绘图节点对象

DrawNode\* m\_draw;

//屏幕的可见尺寸

Size m\_visibleSize;

//地图数组指针

mapNode\*\* m\_map = nullptr;

//寻路起点指针

mapNode\* m\_origin = nullptr;

//寻路终点指针

mapNode\* m\_destination = nullptr;

//判断是否可以在地图上移动

bool m\_isReadyToMove = true;

public:

static cocos2d::Scene\* createScene();

virtual bool init();

CREATE\_FUNC(MazeScene);

//初始化地图数组

void initMap();

//控制精灵沿路径移动

void moveOnPath(mapNode\* tempNode);

//A\*算法主函数

int aStar(mapNode\*\* map, mapNode\* origin, mapNode\* destination);

//添加屏幕触摸监听

void addTouchListener();

//定时器更新

void update(float delta);

bool onTouchBegan(Touch \*touch, Event \*unused\_event);

void onTouchEnded(Touch \*touch, Event \*unused\_event);

};

### 4.初始化游戏场景

（1）在init函数中对场景类初始化如下，完成载入场景地图、初始化地图数组、预置提示标签、添加精灵等操作：

bool MazeScene::init()

{

if (!Layer::init())

{

return false;

}

//获得屏幕尺寸

m\_visibleSize = Director::getInstance()->getVisibleSize();

//载入地图，将放置到适当位置

auto tileMap = TMXTiledMap::create("superMaze.tmx");

tileMap->setPosition(0, m\_visibleSize.height - MAP\_HEIGHT \* UNIT);

this->addChild(tileMap, 1);

//获取地图中的主图层

m\_mapLayer = tileMap->getLayer("mainLayer");

//初始化地图数组

initMap();

//在屏幕底端中央显示闪烁的操作提示

auto playTip = LabelTTF::create("Click Map To Put Treasure", "Arial", 40);

playTip->setPosition(m\_visibleSize.width / 2, 0.5\*(m\_visibleSize.height - MAP\_HEIGHT \* UNIT));

this->addChild(playTip, 1);

auto blink = Blink::create(5, 10);

playTip->runAction(blink);

//预置找到宝箱的提示，将其设置为不可见

auto successTip = Label::createWithBMFont("fonts/futura-48.fnt", "Found Treasure!");

successTip->setPosition(m\_visibleSize.width / 2, m\_visibleSize.height / 2);

this->addChild(successTip, 2,TIP\_TAG);

successTip->setVisible(false);

//在地图起点放置寻宝精灵

auto smileSprite = Sprite::create("smile.png");

smileSprite->setPosition(0.5\*UNIT, m\_visibleSize.height - 0.5\*UNIT);

this->addChild(smileSprite, 3, SMILE\_TAG);

//预置宝箱精灵，将其设置为不可见

auto boxSprite = Sprite::create("box.png");

boxSprite->setPosition(m\_visibleSize.width / 2, m\_visibleSize.height / 2 );

boxSprite->setVisible(false);

this->addChild(boxSprite, 3, BOX\_TAG);

//预置禁止放置图标，将其设置为不可见

auto forbidSprite = Sprite::create("forbid.png");

forbidSprite->setPosition(m\_visibleSize.width / 2 , m\_visibleSize.height / 2 );

forbidSprite->setVisible(false);

this->addChild(forbidSprite, 4, FORBID\_TAG);

//添加绘图节点

m\_draw = DrawNode::create();

this->addChild(m\_draw, 2, DRAW\_TAG);

//添加屏幕触摸监听

addTouchListener();

//调用定时器更新

this->scheduleUpdate();

return true;

}

（2）初始化地图数组函数initMap定义如下：

void MazeScene::initMap()

{

//根据地图宽、高分配数组空间

m\_map = new mapNode\*[MAP\_WIDTH];

for (int n = 0; n < MAP\_WIDTH; n++)

m\_map[n] = new mapNode[MAP\_HEIGHT];

//依次扫描地图数组每一个单元

for (int i = 0; i< MAP\_WIDTH; i++)

{

for (int j = 0; j< MAP\_HEIGHT; j++)

{

//若当前位置为墙体瓦片设置为不可通过

if (m\_mapLayer->getTileGIDAt(Vec2(i, j)) == NOT\_ACCESS\_TILE)

m\_map[i][j] = {NOT\_ACCESS, i, j, 0, 0, 0, nullptr };

//否则设置为可以通过

else

m\_map[i][j] = { ACCESS, i, j, 0, 0, 0, nullptr };

}

}

//初始化起点为（0，0）单元

m\_map[0][0].status = ORIGIN;

m\_origin = &m\_map[0][0];

}

（3）监听屏幕触摸事件的函数addTouchListener定义如下：

void MazeScene::addTouchListener()

{

//获得监听器

auto listener = EventListenerTouchOneByOne::create();

//为触摸开始和触摸结束设置回调函数

listener->onTouchBegan = CC\_CALLBACK\_2(MazeScene::onTouchBegan, this);

listener->onTouchEnded = CC\_CALLBACK\_2(MazeScene::onTouchEnded, this);

//设置触摸监听优先级

Director::getInstance()->getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener, this);

}

### 5.设置触摸监听函数

（1）在onTouchBegan函数中，首先将触摸点坐标转化为地图数组坐标，若该坐标为墙体或超出地图范围，或者不允许移动时，在触摸点设置禁止图标；否则在触摸点放置宝箱，以该坐标点为终点调用A\*算法开始寻路。其中，设置是否允许移动变量m\_isReadyToMove，是为阻止在一次寻路未到达终点前开始第二次移动。

bool MazeScene::onTouchBegan(Touch \*touch, Event \*unused\_event)

{

//将触摸点坐标转化为地图数组坐标

int x = (int)(touch->getLocation().x / UNIT);

int y = (int)((m\_visibleSize.height - touch->getLocation().y) / UNIT);

//若该地图单元为墙体或超出地图范围，设置禁止图标

if (m\_map[x][y].status == NOT\_ACCESS || y >= MAP\_HEIGHT)

{

auto forbid = this->getChildByTag(FORBID\_TAG);

forbid->setPosition(0.5\*UNIT + x\*UNIT, m\_visibleSize.height - 0.5\*UNIT - y\*UNIT);

forbid->setVisible(true);

return true;

}

else

{

//若该地图单元可到达且允许移动

if (m\_isReadyToMove)

{

//为到达终点前不再允许移动

m\_isReadyToMove = false;

//将当前地图单元设置为终点

m\_map[x][y].status = DESTINATION;

m\_destination = &m\_map[x][y];

//在当前地图单元添加宝箱精灵

auto box= this->getChildByTag(BOX\_TAG);

box->setPosition(0.5\*UNIT + x\*UNIT, m\_visibleSize.height - 0.5\*UNIT - y\*UNIT);

box->setVisible(true);

//调用A\*算法开始寻路

aStar(m\_map, m\_origin, m\_destination);

//将上一次寻路成功的提示隐藏

auto tip = this->getChildByTag(TIP\_TAG);

tip->setVisible(false);

return false;

}

//若不允许移动，在该地图单元设置禁止图标

else

{

auto forbid = this->getChildByTag(FORBID\_TAG);

forbid->setPosition(0.5\*UNIT + x\*UNIT, m\_visibleSize.height - 0.5\*UNIT - y\*UNIT);

forbid->setVisible(true);

return true;

}

}

}

（2）当onTouchBegan函数中设置禁止图标时，将会返回true，然后调用onTouchEnd函数，在onTouchEnd函数中，只需将禁止图标和提示隐藏即可，具体代码如下：

void MazeScene::onTouchEnded(Touch \*touch, Event \*unused\_event)

{

//隐藏禁止图标

auto forbid = this->getChildByTag(FORBID\_TAG);

forbid->setVisible(false);

//隐藏寻路成功提示

auto tip = this->getChildByTag(TIP\_TAG);

tip->setVisible(false);

}

### 6.添加寻路处理

（1）在A\*算法的主函数中，与上一节的代码框架基本相同，主要区别在于对寻路失败和寻路成功的处理。在本次实例中，若寻路失败，则在屏幕中央显示提示标签；若寻路成功则调用moveOnPath函数控制精灵在路径上移动。具体代码已在第2步修改命令行版本的A\*算法中列出，此处不再赘述。

（2）在控制精灵沿路径移动的moveOnPath函数中，需要存储由父节点指针得到的路径信息，保存在结构体数组中。然后从结构体数组末尾反向提取，根据路径上的每一个单元坐标，创建精灵移动的动作序列，并绘制移动路径。具体代码如下：

void MazeScene::moveOnPath(mapNode\* tempNode)

{

//声明存储路径坐标的结构体

struct pathCoordinate{ int x; int y; };

//分配路径坐标结构体数组

pathCoordinate\* path = new pathCoordinate[MAP\_WIDTH\*MAP\_HEIGHT];

//利用父节点信息逆序存储路径坐标

int loopNum = 0;

while (tempNode != nullptr)

{

path[loopNum].x = tempNode->xCoordinate;

path[loopNum].y = tempNode->yCoordinate;

loopNum++;

tempNode = tempNode->parent;

}

//将笑脸精灵的坐标存为绘制线段起点

auto smile = this->getChildByTag(SMILE\_TAG);

int fromX = smile->getPositionX();

int fromY = smile->getPositionY();

//声明动作向量存储动作序列

Vector<FiniteTimeAction\*> actionVector;

//从结构体数组尾部开始扫描

for (int j = loopNum - 2; j >= 0; j--)

{

//将地图数组坐标转化为屏幕实际坐标

int realX = (path[j].x + 0.5)\*UNIT;

int realY = m\_visibleSize.height - (path[j].y + 0.5)\*UNIT;

//创建移动动作并存入动作向量

auto moveAction = MoveTo::create(0.2, Vec2(realX, realY));

actionVector.pushBack(moveAction);

//绘制从起点到下一个地图单元的线段

m\_draw->drawLine(Vec2(fromX, fromY), Vec2(realX, realY), Color4F(1.0, 1.0, 1.0, 1.0));

//将当前坐标保存为下一次绘制的起点

fromX = realX;

fromY = realY;

}

//创建动作序列

auto actionSequence = Sequence::create(actionVector);

//笑脸精灵执行移动动作序列

smile->runAction(actionSequence);

}

### 7.添加定时器更新

最后，需要通过定时器更新update函数判断精灵是否到达目的地，若到达，则将宝箱隐藏，并将地图数组恢复初始状态，将目的地保存为新的起点等等，为下一次移动做好准备。具体代码如下：

void MazeScene::update(float delta)

{

auto smile = this->getChildByTag(SMILE\_TAG);

auto box = this->getChildByTag(BOX\_TAG);

//如果笑脸精灵达到宝箱位置

if ( smile->getBoundingBox().intersectsRect( box->getBoundingBox() ) )

{

//隐藏宝箱

box->setVisible(false);

//将地图数组恢复初始状态

for (int i = 0; i< MAP\_WIDTH; i++)

{

for (int j = 0; j< MAP\_HEIGHT; j++)

{

if (m\_map[i][j].status != NOT\_ACCESS)

{

m\_map[i][j] = { ACCESS, i, j, 0, 0, 0, nullptr };

}

}

}

//将当前单元设为起点

m\_origin = m\_destination;

m\_origin->status = ORIGIN;

//清除绘制的路径

m\_draw->clear();

//允许再次移动

m\_isReadyToMove = true;

//显示寻路成功的标签提示

auto tip = this->getChildByTag(TIP\_TAG);

tip->setVisible(true);

}

}

游戏实例的运行效果如下图所示：

