面向对象课程设计报告

一、题目简介

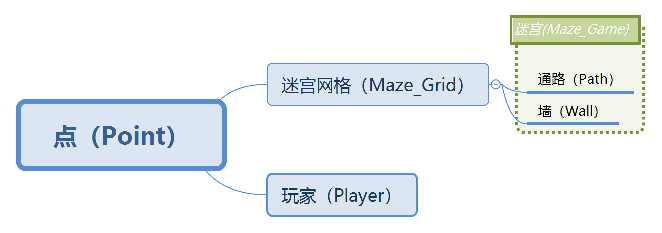
迷宫游戏是当今最为经典的一款益智类游戏。迷宫游戏简单, 容易上手, 极具创意, 虽历经数年, 仍相当受欢迎。通过随机生成迷宫地图, 玩家从入口进入迷宫, 探索前行。往往是选择一个路口, 走着走着就到了死胡同, 只能退回来, 选择其它路口再试走。就这样, 不断地折回, 反复地寻找, 最后才能找到出口, 最后成功走出迷宫。此时此刻, 玩家也从成功走出迷宫获得愉悦的心情, 从而把工作生活中的压力释放掉。本系统利用C#编程语言和WPF技术，通过面向对象编程实现了迷宫游戏的开发，并通过数据库保存了通关用户的数据。

二、系统概述

迷宫游戏根据用户选择的游戏难度，随机产生迷宫地图。游戏操作方式键盘为主，用户可以通过键盘上的方向键控制玩家到达终点。同时设置限时，用户必须在规定时间内走出迷宫。用户通关后，记录过关路线，根据用时和难度情况计分。记录所有通关用户并在排行榜中按得分从高到低排序。

三、系统设计

本系统设计中主要定义和运用了点（Point）、迷宫网格（Maze\_Grid）、通路（Path）、墙（Wall）、迷宫游戏（Maze\_Game）共5个类，其相互之间的继承、组合等关系如下：

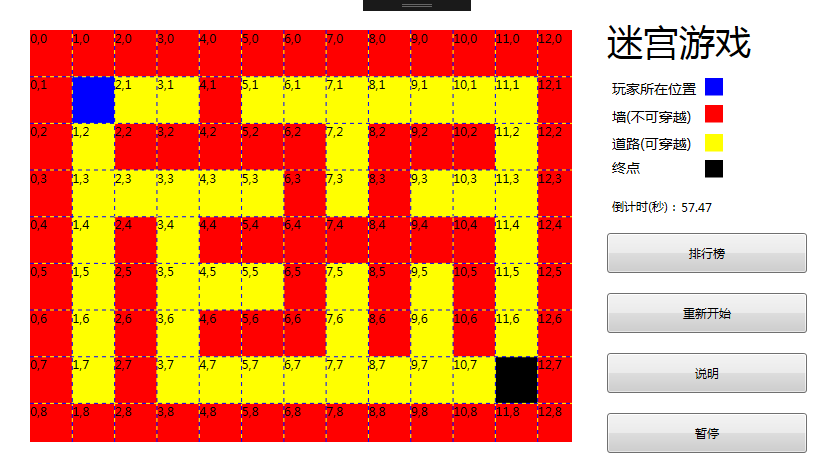


玩家在迷宫中移动不能通过墙而只能通过通路，当玩家所在位置与终点重合时，闯关成功。在用户游戏时记过关路线、过关用时、迷宫形状等数据，在用户闯关成功后，向数据库插入数据。用户查看排行榜时，向数据查询数据，查看过关路线时会根据记录的数据将过关路线重新绘制出来。

四、关键技术

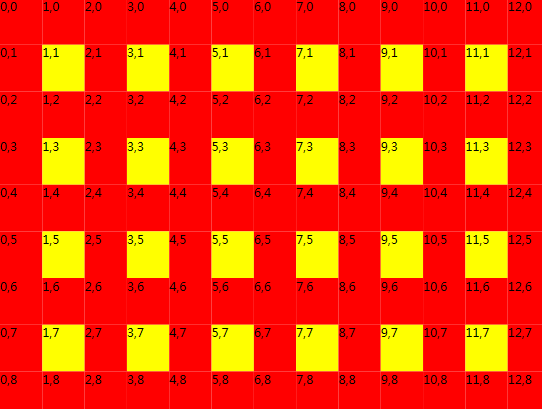
1.设计迷宫

在本系统设计中，把迷宫看成一个个独立的网格，以左上角网格为原点、水平向右方向为x轴，竖直向下方向为y轴建立二维平面直角坐标，每个网格对应一个坐标点，每个坐标点可能是通路，也可能是墙，玩家在迷宫中移动不能通过墙而只能通过通路。当玩家所在位置与终点重合时，闯关成功。迷宫标出坐标的效果如下：



2.随机生成迷宫

本系统采用深度优先遍历法来机随机生成迷宫，如下图所示，把黄色格子看做图的结点，从坐标为（1,1）的结点开始进行深度优先遍历，每次访问一个结点就把其未访问过的相邻结点入栈，接着弹出栈顶元素并访问，打通两个结点之间的“墙“，当栈为空时，即表明对整个图都完成了遍历。



3.记录用户数据

数据库建表sql如下：

CREATE TABLE [dbo].[Users] (

[Name] VARCHAR (50) NOT NULL,

[MaxScore] FLOAT (53) NOT NULL,

[Route] VARCHAR (MAX) NOT NULL,

[Wall] VARCHAR (MAX) NOT NULL,

[Path] VARCHAR (MAX) NOT NULL,

CONSTRAINT [PK\_Table] PRIMARY KEY CLUSTERED ([Name] ASC)

);

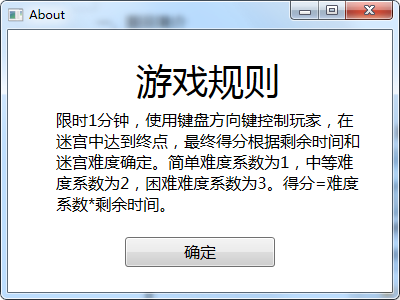
其中Name为用户姓名，MaxScore为用户最高得分，Route为用户过关路线，Wall为用户过关迷宫的墙的位置，Path为用户过关迷宫的通路位置。

在用户游戏时记录相关数据，在用户闯关成功后，向数据库插入即可。在查看排行榜时查询数据库，根据得分从高到低排序，并根据记录的数据重新画出过关路线即可。

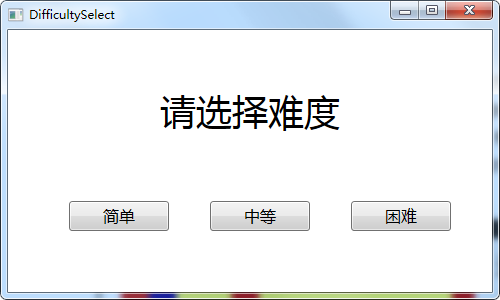
五、效果及存在问题

运行效果：

1.介绍游戏规则



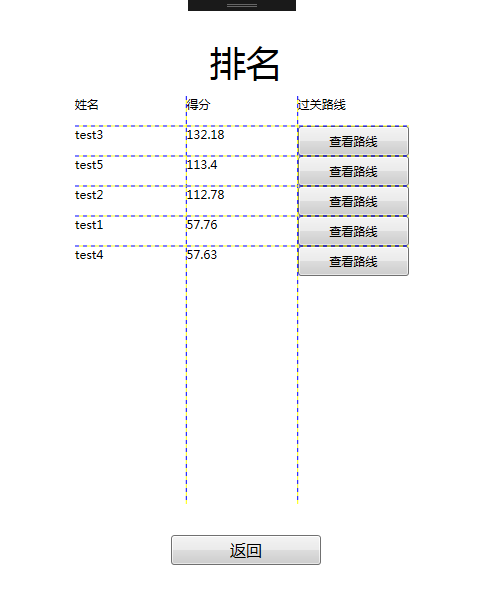
2.游戏难度选择



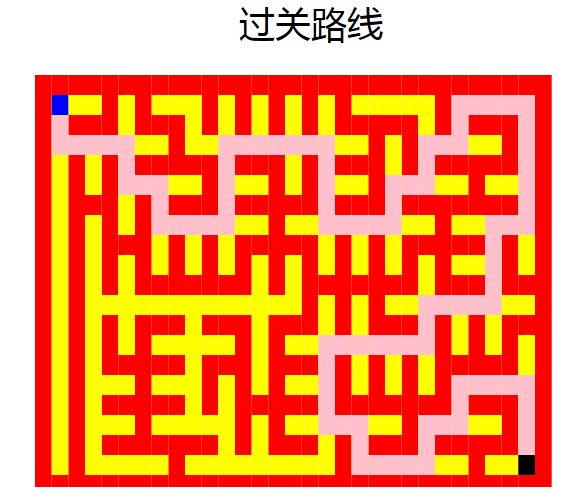
3.游戏主界面



4.排行榜



5.查看用户过关路线：



存在的不足和预期解决办法：

无法对数据库连接异常的具体情况进行分析，仅仅使用了简单的try……catch结构，对用户名重名进行了检验，而没有考虑其他情况，所以当用户连接数据库发生其他异常时只能得知连接异常不能具体分析出异常来源。预期解决方案为尽量优化代码，举出更多连接异常的情况，给出更详细的报错信息，提示用户联系管理员协助处理。

除此之外，本系统的不足还有游戏界面简单，没有动画效果，视觉效果差强人意。预期的解决方案为之后对界面美工进行优化，增加动画，增强系统的流畅性和美观性。

六、心得体会

回顾起此课程设计，至今我仍感慨颇多，从理论到实践，在这段日子里，学到很多很多的东西，同时不仅可以巩固了以前所学过的知识，而且学到了很多在书本上所没有学到过的知识。通过这次课程设计使我懂得了理论与实际相结合是很重要的，只有理论知识是远远不够的，只有把所学的理论知识与实践相结合起来，从理论中得出结论，才能真正为社会服务，从而提高自己的实际动手能力和独立思考的能力。在设计的过程中遇到问题，可以说得是困难重重，但可喜的是最终都得到了解决。

七、关键代码附录

1.对迷宫网格初始化

private void initGrid(int Columns, int Rows)

{

for (int i = 0; i < Columns; i++) Maze.ColumnDefinitions.Add(new ColumnDefinition());

for (int i = 0; i < Rows; i++) Maze.RowDefinitions.Add(new RowDefinition());

g = new Maze\_Grid[Columns, Rows];

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

{

for (int j = 0; j < Rows; j++)

{

g[i, j] = new Maze\_Grid(i, j); // 初始化坐标

if (i == 0 || j == 0 || i + 1 == Columns || j + 1 == Rows) // 标边界

{

g[i, j].Visited = true;

g[i, j].isConnected = false;

}

else if (i % 2 == 0 || j % 2 == 0) // 标记墙

{

g[i, j].Visited = false;

g[i, j].isConnected = false;

}

else // 待打通的网格

{

g[i, j].Visited = false;

g[i, j].isConnected = true;

}

}

}

2. 深度优先遍历法生成迷宫形状

private void DFS(int xi, int yi, int xo, int yo)

{

Stack<Maze\_Grid> st = new Stack<Maze\_Grid>();

st.Push(g[xi, yi]);

Random rd = new Random();

while (st.Count != 0)

{

Maze\_Grid e = st.Pop(); // 出栈

int[] direction = new int[4]; // 1, 2, 3, 4 分别代表左、上、右、下四个方向

// 打乱数组

int j = 0;

bool exist = false;

while (j < 4)

{

exist = false;

int rand = rd.Next(1, 5);

foreach(int k in direction)

{

if (k == rand) exist = true;

}

if (!exist)

{

direction[j] = rand;

j++;

}

}

foreach(int i in direction)

{

switch (i)

{

case 1:

if (e.X - 2 > 0 && e.X - 2 > 1 && g[e.X - 2, e.Y].Visited == false)

{

if (e.X - 2 > 0) g[e.X - 2, e.Y].Visited = true;

st.Push(g[e.X - 2, e.Y]); //左方网格进栈

g[e.X - 1, e.Y].isConnected = true; //打通两个方块

}

break;

case 2:

if (e.Y + 2 > 0 && e.Y + 2 < yo && g[e.X, e.Y + 2].Visited == false)

{

g[e.X, e.Y + 2].Visited = true;

st.Push(g[e.X, e.Y + 2]); //上方网格进栈

g[e.X, e.Y + 1].isConnected = true; //打通两个方块

}

break;

case 3:

if (e.X + 2 < xo && e.X + 2 < xo && g[e.X + 2, e.Y].Visited == false)

{

g[e.X + 2, e.Y].Visited = true;

st.Push(g[e.X + 2, e.Y]); // 右方网格进栈

g[e.X + 1, e.Y].isConnected = true; //打通两个方块

}

break;

case 4:

if (e.Y - 2 > 0 && e.Y - 2 < yo && g[e.X, e.Y - 2].Visited == false)

{

if (e.Y - 2 > 0) g[e.X, e.Y - 2].Visited = true;

st.Push(g[e.X, e.Y - 2]); // 下方网格进栈

g[e.X, e.Y - 1].isConnected = true; //打通两个方块

}

break;

default:

break;

}

}

}

// 去除边缘多余的墙

if (xo % 2 == 0)

{

for (int i = 1; i < yo; i++)

{

if (g[xo - 3, i].isConnected == true) g[xo-2, i].isConnected = true;

}

}

if (yo % 2 == 0)

{

for (int i = 1; i < xo; i++)

{

if (g[i, yo - 3].isConnected == true) g[i, yo - 2].isConnected = true;

}

}

}

private void mazeShape(int Columns, int Rows, int diff)

{

Maze\_ = new Maze\_Game(diff);

Maze\_.walls = new List<Wall>();

Maze\_.paths = new List<Path>();

Maze\_.route = new List<int>();

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

{

for (int j = 0; j < Rows; j++)

{

if (g[i,j].isConnected == true)

{

Path t = new Path(g[i, j].X, g[i, j].Y, g[i, j].isConnected);

g[i, j] = t;

Maze\_.paths.Add(t);

} else

{

Wall t = new Wall(g[i, j].X, g[i, j].Y, g[i, j].isConnected);

g[i, j] = t;

Maze\_.walls.Add(t);

}

}

}

}

3. 向服务器上传数据

public bool Post()

{

try

{

A.Name = scc.name;

string route = "";

for (int i = 0; i < Maze\_.route.Count; i++) { route += Maze\_.route[i].ToString() + ","; }

string walls = "";

for (int i = 0; i < Maze\_.walls.Count; i++) { walls += Maze\_.walls[i].X + "," + Maze\_.walls[i].Y + ","; }

string paths = "";

for (int i = 0; i < Maze\_.paths.Count; i++) { paths += Maze\_.paths[i].X + "," + Maze\_.paths[i].Y + ","; }

// 连接数据库

string s = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=D:\\Desktop\\maze\\WPF\\Maze\\Maze\\data\\rank\_list.mdf;Integrated Security=True;Connect Timeout=30";

SqlConnection conn = new SqlConnection(s); //创建一个连接实例

string insertStr = "INSERT INTO [dbo].[Users] ([Name], [MaxScore], [Route], [Wall], [Path]) " +

"VALUES(N'" + A.Name + "', " + Maze\_.Score.ToString() + ", N'"+ route +"', N'"+walls+ "', N'"+paths+"')";

SqlCommand insert = new SqlCommand(insertStr, conn);

conn.Open(); //打开连接

insert.ExecuteNonQuery();

conn.Close();

return true;

}

catch

{

return false;

}

}

}

}

4.重现玩家绘制过关路线

private void init(RankList.Record R)

{

string[] walls = R.getWalls().Split(',');

string[] paths = R.getPaths().Split(',');

string[] route = R.getRoute().Split(',');

string[] start = { "1", "1" };

int Columns = Convert.ToInt32(walls[walls.Length - 3]) + 1;

int Rows = Convert.ToInt32(walls[walls.Length - 2]) + 1;

string[] end = { (Columns - 2).ToString(), (Rows - 2).ToString() };

// 画出网格

drawGrid(Columns, Rows);

// 画墙

Draw(walls, Brushes.Red);

// 画通路

Draw(paths, Brushes.Yellow);

// 画过关路线

Draw(route, Brushes.Pink);

// 画起点

Draw(start, Brushes.Blue);

// 画终点

Draw(end, Brushes.Black);

}

private void drawGrid(int Columns, int Rows)

{

for (int i = 0; i < Columns; i++) Maze.ColumnDefinitions.Add(new ColumnDefinition());

for (int i = 0; i < Rows; i++) Maze.RowDefinitions.Add(new RowDefinition());

}

private void Draw(string[] walls, Brush e)

{

for (int i = 0; i < walls.Length - 1; i += 2)

{

TextBlock newArea = new TextBlock();

Maze.Children.Add(newArea);

Grid.SetColumn(newArea, Convert.ToInt32(walls[i]));

Grid.SetRow(newArea, Convert.ToInt32(walls[i+1]));

newArea.Background = e;

}

}

}

5. 查询历史数据

public Record[] ReadRecord()

{

// 连接数据库

string s = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=D:\\Desktop\\maze\\WPF\\Maze\\Maze\\data\\rank\_list.mdf;Integrated Security=True;Connect Timeout=30";

SqlConnection conn = new SqlConnection(s); //创建一个连接实例

string queryStr = "SELECT \* FROM Users";

string countStr = "SELECT COUNT(\*) FROM Users";

SqlCommand query = new SqlCommand(queryStr, conn);

SqlCommand count = new SqlCommand(countStr, conn);

conn.Open(); //打开连接

SqlDataReader counts = count.ExecuteReader();

counts.Read();

int len = counts.GetInt32(0);

counts.Close();

Record[] R = new Record[len]; // 记录

SqlDataReader res = query.ExecuteReader();

int i = 0;

while (res.Read())

{

R[i] = new Record(res.GetSqlString(0).ToString(), Convert.ToDouble(res.GetSqlDouble(1).ToString()), res.GetSqlString(2).ToString(),

res.GetSqlString(3).ToString(), res.GetSqlString(4).ToString());

i++;

}

conn.Close(); //关闭连接

return R;

}

6. 关键类设计

abstract private class Point // 坐标点

{

protected int x;

protected int y; // 表示网格的坐标

public int X

{

get { return x; }

set { if (value >= 0) x = value; }

}

public int Y

{

get { return y; }

set { if (value >= 0) y = value; }

}

public Point(int xi, int yi) // 初始化坐标

{

x = xi;

y = yi;

}

virtual public bool isOut() { return false; } // 判断该点是否为终点

}

private class Maze\_Grid : Point // 迷宫网格

{

public bool isConnected { set; get; } // 判断网格是否连通

public bool Visited { get; set; } // 用于深度优先遍历

public Maze\_Grid(int xi, int yi) : base(xi, yi) { }

}

private class Path : Maze\_Grid // 通路

{

public Path(int xi, int yi, bool co) : base(xi, yi) { isConnected = co; }

}

private class Wall : Maze\_Grid // 墙

{

public Wall(int xi, int yi, bool co) : base(xi, yi) { isConnected = co; }

}

private class Maze\_Game

{

private int difficulty; // 难度系数

public double Score { get; private set; } // 玩家得分

public void caculateScore(double time) { Score = difficulty \* time; } // 计算玩家得分

public List<Path> paths;

public List<Wall> walls;

public List<int> route; // 玩家过关路线

public Maze\_Game(int di) { difficulty = di; }

}

private class Player : Point // 玩家

{

private int xout;

private int yout; // 终点

public double xstep { get; set; } // x轴上的步长

public double ystep { get; set; } // y轴上的步长

public string Name { set; get; } // 姓名

public void MoveToLeft() { x = x - 1; }

public void MoveToTop() { y = y - 1; }

public void MoveToRight() { x = x + 1; }

public void MoveToBottom() { y = y + 1; } // 玩家移动

override public bool isOut() // 判断玩家当前位置是否为终点

{

if (x == xout && y == yout) return true;

else return false;

}

public Player(int xi, int yi, int xo, int yo) : base(xi, xi) { xout = xo; yout = yo; }

}

八、参考文献

[1]潘家富.一种简洁迷宫游戏设计[J].电子技术与软件工程,2019(13):239-240+256.