# ET框架学习笔记

|  |  |  |
| --- | --- | --- |
| 修改日期 | 修改人员 | 修改内容 |
| 2020/07/03 | 刘刚 |  |
|  |  |  |

## ET框架概述

### 资源下载

代码：<https://gitee.com/egametang/ET>

论坛：<http://bbs.etframework.net/forum.php>

QQ群：474643097

### 代码架构

##### Server

这个目录是服务器的主目录。

##### Config

这个目录是Server的数据所在目录。

##### Unity

这个目录是客户端Unity编辑器的主目录。

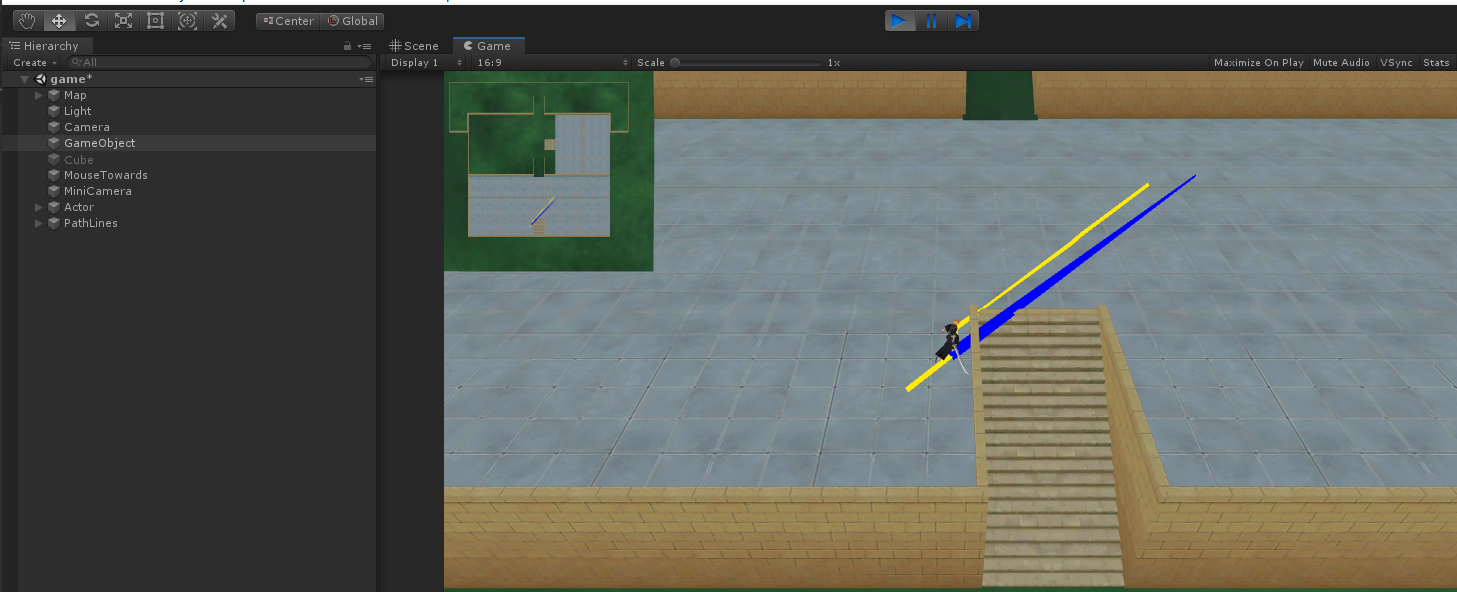
### DotNet2.2

## 更换NavMesh

### 问题

ET作者说：

ET的寻路Unity2019有问题，可以用这个寻路替代，这个寻路非常简单，很实用 <https://github.com/genechiu/NavMesh> 我现在改成用recastnavigation寻路了，c#包cpp，前后端都这样用,astar那个不支持2019要重新移植，太麻烦了. et6是不会上recast了，实在是没时间弄



<https://github.com/genechiu/NavMesh>

这是ET框架开发者（？）制作的一个navmesh的例子。自己保存navmesh数据，然后读取。

* 运行Game场景。
* 要想增加人物的话，你需要创建一个空物体，然后把main.cs脚本挂上去，就可以运行了。
* 运行以后，要在【小地图】上点击一下，才能把人种上

ET的一个使用者说：

目前ET的寻路，8K\*8K是能寻路，但寻路数据与地形不吻合。所以，如果真的能完全重合。充其量就只能4k\*4k而已。不知道这个，会是怎么样

### 改造

#### 拷贝新版NavMesh文件

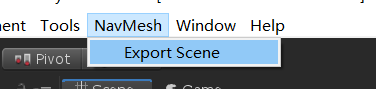
下载<https://github.com/genechiu/NavMesh>

新的代码，都以【**NavMesh**】作为关键字，与之前的PathFinding做区分。

##### 客户端

将Assets\Scripts\NavMesh目录

拷贝到\Unity\Assets\Model\Module\Pathfinding\NavMesh目录下。



拷贝后，Unity编辑器会增加这个菜单。

##### 服务器

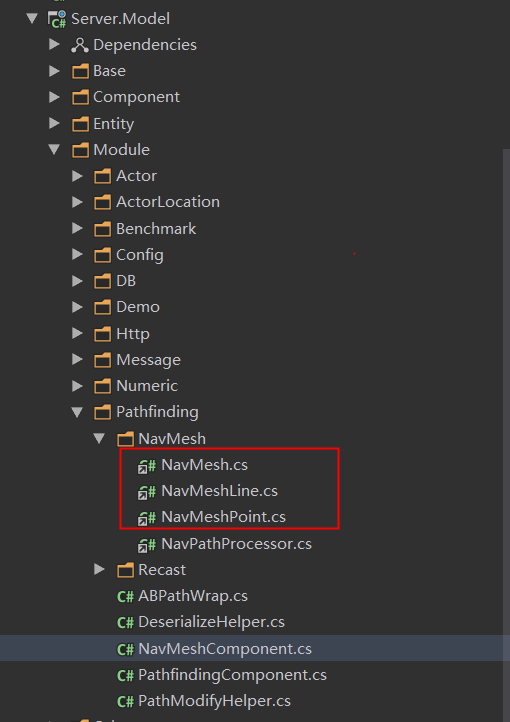
将客户端Unity\Assets\Model\Module\Pathfinding\NavMesh目录中的

NavMesh.cs

NavMeshLine

NavMeshPoint

三个文件，以【**链接**】的形式添加到服务器Server工程的Server.Model项目的Module/PathFinding/NavMesh目录下。



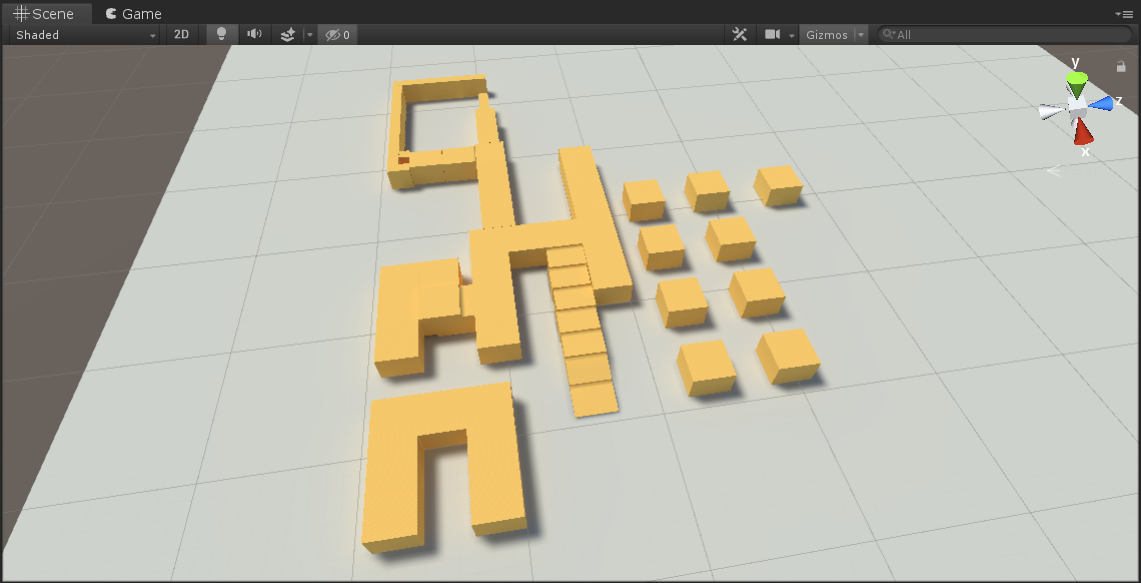
***注意：是以链接的方式，而不是拷贝。***

这是本ET框架方便的地方之一：服务器和客户端可以共享代码。

服务器代码只需要这三个文件，而不需要【Editor】目录里的编辑器所需要的文件。

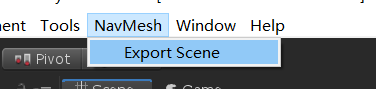
#### 生成新地图数据

##### 打开场景

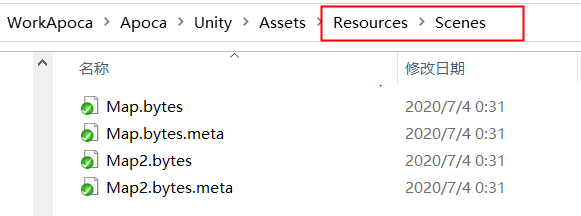


打开需要被输出的场景，例如：map.unity场景。

##### 输出



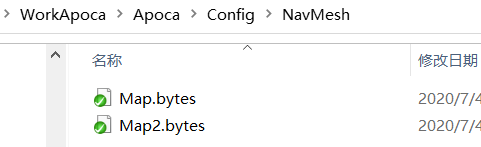
在Unity编辑器的菜单NavMesh/Export Scene，点击即可。



地图数据文件生成在Resources/Scenes目录下。

文件名为：场景名.bytes。

##### 拷贝到服务器



把bytes文件拷贝到服务器端Config/NavMesh目录下。

##### 附录：将navmesh转换成navmesh开源工具可以识别的格式

<https://blog.csdn.net/huutu/article/details/52672505>

如果需要可以把数据文件转换成\*.obj的格式，这样navmesh的Demo工具可以读取出来。

#### 修改旧文件

##### AStarConfig.cs

[BsonIgnore]

public NavGraph[] graphs;

//替换寻路。July.3.2020. Liu Gang.

[BsonIgnore]

public NavMesh navMesh;

添加navMesh成员，与graphs类似。

##### Program.cs

Game.Scene.AddComponent<MailboxDispatcherComponent>();

Game.Scene.AddComponent<ActorMessageDispatcherComponent>();

// 替换寻路。July.3.2020. Liu Gang.

//Game.Scene.AddComponent<PathfindingComponent>();

Game.Scene.AddComponent<NavMeshComponent>();

break;

添加NavMeshComponent组件，与PathfindingComponent类似。

##### UnitPathComponentHelper

// 替换寻路。July.3.2020. Liu Gang.

//PathfindingComponent pathfindingComponent = Game.Scene.GetComponent<PathfindingComponent>();

NavMeshComponent navmeshComponent = Game.Scene.GetComponent<NavMeshComponent>();

self.ABPath = ComponentFactory.Create<ABPathWrap, Vector3, Vector3>(unit.Position, new Vector3(target.x, target.y, target.z));

navmeshComponent.Search(self.ABPath);

Log.Debug($"find result: {self.ABPath.Result.ListToString()}");

#### 新建文件

##### NavMeshComponent.cs

在Server端的Server\Model\Module\Pathfinding目录下，新建文件NavMeshComponent.cs。

这个文件类似原来的PathfindingComponent.cs文件。

是所有寻路的基础类。

public override void Awake(NavMeshComponent self)

{

self.AStarConfig = new AStarConfig(); //MongoHelper.FromJson<AStarConfig>(File.ReadAllText("./pathfinding.config"));

// 更换新的地图数据。July.3.2020. Li

// 读取地图数据.

//self.AStarConfig.graphs = DeserializeHelper.Load("../Config/graph.bytes");

const string pathName = "../Config/NavMesh/map.bytes";

FileInfo fi = new FileInfo(pathName);

BinaryReader reader = new BinaryReader(new FileStream(pathName, FileMode.Open));

long count = fi.Length;

self.AStarConfig.navMesh = new NavMesh(reader.ReadBytes((int)count));

self.PathReturnQueue = new PathReturnQueue(self);

self.PathProcessor = new PathProcessor(self.PathReturnQueue, 1, false);

// July.3.2020. Liu Gang.

self.NavPathProcessor = new NavPathProcessor(self.AStarConfig.navMesh);

// 读取寻路配置

self.AStarConfig.pathProcessor = self.PathProcessor;

// July.3.2020. Liu Gang.

self.AStarConfig.navPathProcessor = self.NavPathProcessor;

}

##### NavPathProcessor.cs

在Unity端的Unity\Assets\Model\Module\Pathfinding\NavMesh目录下，新建文件NavPathProcessor.cs。

这个文件类似原来的PathProcessor.cs。

是寻路Search的主要文件。

public class NavPathProcessor

{

internal readonly ThreadControlQueue queue;

internal NavMesh \_navMesh;

internal NavPathProcessor(NavMesh navMesh)

{

queue = new ThreadControlQueue(1);

this.\_navMesh = navMesh;

}

public void CalculatePath()

{

Path path = this.queue.Pop();

ABPath abPath = path as ABPath;

Vector3 vecStart = abPath.startPoint;

int faceStart = this.\_navMesh.FindFaceAt(vecStart.x, vecStart.z);

NavMeshPoint startPoint = new NavMeshPoint(this.\_navMesh, faceStart, vecStart.x, vecStart.y, vecStart.z);

Vector3 vecEnd = abPath.endPoint;

int faceEnd = this.\_navMesh.FindFaceAt(vecEnd.x, vecEnd.z);

NavMeshPoint endPoint = new NavMeshPoint(this.\_navMesh, faceEnd, vecEnd.x, vecEnd.y, vecEnd.z);

List<NavMeshLine> outPath = new List<NavMeshLine>();

if (!\_navMesh.FindShortestPath(startPoint, endPoint, outPath))

return;

List<Vector3> outPath2 = new List<Vector3>();

foreach (NavMeshLine line in outPath)

{

NavMeshPoint point = line.startPoint;

Vector3 node2 = new Vector3(point.x, point.y, point.z);

outPath2.Add(node2);

}

outPath2.Add(vecEnd);

path.vectorPath = outPath2;

}

}

主要是因为新版的寻路，它所使用的的数据结构和之前所使用的的数据结构不一样，所以大量代码都在准备这些数据结构，实际上的寻路只有一句：FindShortestPath（）。