**3D在线游戏关卡**

**随机生成系统V1.0**

**设计说明书**

**一．软件设计背景**

目前大型3D多人在线网络游戏等游戏中，副本关卡等游戏内容都是先由美术制作好场景模型再由策划编辑出任务、怪物和怪物AI等；玩家多次参与后，固定的内容带来的吸引力逐渐降低。为维持玩家的新鲜感，需要持续提供新的场景资源构建新的关卡。新关卡的制作给美术和策划人员带来不小的负担，而且过多的关卡地图等对游戏服务器硬件资源的有着不能忽视的消耗。按照一定规则由服务器运算自动产生的随机关卡能让场景资源和内容资源利用更高，同时还能保证玩家每次进入关卡能有不同的体验。2D单机游戏方面对于随机关卡的应用已经较为成熟，但3D在线游戏在此方面的探索还较少。本系统将专门针对3D在线游戏提供一个完整的随机关卡自动生成解决方案。

**二. 软件设计说明**

## 关卡随机生成系统总体设计方案

关卡随机生成系统主要涉及以下几个方面技术内容:

1. 按子地图方案导出资源；
2. 产生符合规则的子地图序列；
3. 对符合条件的子地图进行地形文件和凸包文件的拼接；
4. 处理拼接边界出现的地形和模型裂缝等；
5. 生成地图缩略图；
6. 正确操纵拼接后关卡的怪物策略；

其中第1 项为关卡编辑器设计方案， 2 , 3 , 6项为服务端设计方案，3 ,4, 5项为客户端设计方案。

随机生成关卡的过程对用户来说是透明的，同正常进入关卡一样，用户点击进入关卡并等待客户端加载资源完成就能体验关卡内容了，因此要求以上所涉及的技术（除离线编辑内容外）执行高效可控，且随机关卡的进入流程作为一般化关卡进入流程的可选项和补充。参照图2.1，关卡随机生成系统将额外增加3个过程，分别为3服务端生成地图、4同步资源序列、5客户端生成地图。



图2.1进入关卡的完整流程

基本地形块subTerrain

编辑器工作区

参考地形+场景文件

A3DTerrain2 + ecwld

预处理地形文件

客户端

处理后的地形文件 +场景文件

客户端

服务器随机序列

图2.2.1 客户端数据流

## 子地图资源编辑方案

在编辑器内创建要随机的基本地形块（subTerrain），美术/策划分别编辑每个地形块，最后导出参考地形给客户端使用。客户端收到服务器随机块序列后，对参考地形块重新排列，并对地形块接缝进行缝合，形成最终的地形文件，供客户端加载。

图2.2.2可随机关卡编辑流程

地形编辑器需要提供以下功能：

* 1. 提供创建、编辑子地形块（尺寸如128米\*128米）的功能；
  2. 按子地形块分块导出NPC/控制器信息；
  3. 提供子地形块的NPC通过图编辑功能；
  4. 针对基本地形块编辑连通性、块类型等属性；
  5. 加载/导出随机属性的配置文件；
  6. 分块导出碰撞信息；



图2.2.3设置随机地图属性信息

## 生成符合规则的子地图序列

随机地图是由固定个数的子地图（本文以8x8块子地图块为例）拼接起来的，子地图有各自的连通性和其内容相关的资源属性(如出口和入口等)。生成基本规则要求只有一个入口和出口，且入口和出口必须连通；在此基础上按游戏参与者的相关信息调整从入口到出口的路径长度，并生成分叉路口和死路；最后再加上直接配置的子地图块，可用于布置固定的传送场景等。

此算法由MazeGen类封装实现，具体步骤如下（①为起点②为终点实线为连通路径虚线为死路）：

1. MazeGen::GenerateMainPath在8x7的子地图区域先选出起点和终点，生成一条最短的路线，将连线通过的Room (单元格，即一块子地图)设置成连通；然后根据主线长度要求、不断在当前路径上随机选取相邻Room去掉连接，改为向同一平行方向各自相邻Room做延伸再连接来实现；

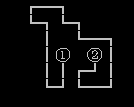


图2.3.1 主路线生成图示

1. MazeGen::AdjustBranch在生成好的主路线上添加分支：随机在主线上抽出一个非起点和终点的Room，分别对它周围4个方向做尝试添加一个新的连通Room ，要求新加如的Room不能在已有的主线上。循环以上方案直到新加数目满足[branch\_min, branch\_max] 支线数目要求；

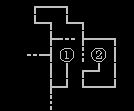


图2.3.2 直线分支添加图示

1. MazeGen::AdjustBranchRoom对AdjustBranch中产生的支线Room周围4方向上尝试添加一个新的连通Room ，要求新加入的Room不能在已有的主线上，循环以上方案直到到新加数目满足[branch\_block\_min, branch\_block\_max] 支线长度要求；

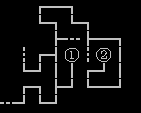


图2.3.3 复杂分支添加图示

1. GetAppropriatePiece根据已产生的Room(单元格)属性(包含连通性,房间类型)选择符合标准的子地图，完成单元格到子地图序号的映射。由于算法产生的单元格不一定能完全匹配上子地图所提供所有属性，故在匹配算法上应有一定的容错性，允许在不能完全匹配的时候能有候补的子地图；

abase::vector<int>equal\_joint\_pieces;

abase::vector<int>close\_joint\_pieces;

for(inti = 0; i< (int)piece\_desps.size(); ++i)

{

if(piece\_desps[i].type != type) continue;

int mask = piece\_desps[i].joint\_mask;

if( mask == joint\_mask)

equal\_joint\_pieces.push\_back(i);

else if ( (mask &joint\_mask ) == joint\_mask )

close\_joint\_pieces.push\_back(i);

}

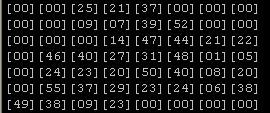


图2.3.4 随机生成的8x7子地图序列

1. FixStub在生成好的8x7的地图上，添加一个8x1从配置读取的固定子地图序列，以符合策划对随机场景中指定坐标内容可控的需求。

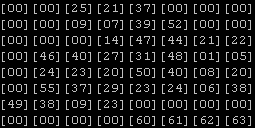


图2.3.5补充了固定规则的完整8x8子地图序列

## 拼接服务器子地图资源

服务器场景相关的资源文件类别分为碰撞，NPC寻路，怪物生成器。以随机迷宫关卡资源(大小8x8)为例,导出资源列表如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 资源目录 | 文件名 | 备注 |
| 地形高度图 | map | 1.hmp 2.hmp  … 64.hmap |  |
| 地面通过图 | movemap | 1.rmap 2.rmap  … 64.rmap  1.dhmap 2.dhmap  … 64.dhmap | 如果某个地图片不存在x.rmap和x.dhmap，则表示该地图片每一点都可达 |
| 空中通过图 | airmap | 1.octr 2.octr  …  64.octr | 如果某个地图片不存在x.octr，则表示该地图片不存在空中阻挡 |
| 水面图 | watermap | 1.wmap 2.wmap  …  64.wmap | 如果某个地图片不存在x.wamp，则表示该地图片不存在水面 |
| 布怪数据 | npcgen | npcgen.data  npcgen\_1.data  …  npcgen\_64.data | npcgen.data为怪物控制器数据  其余为区域布怪数据 |
| 碰撞检测数据 | mapbht | 1.bht 2.bht  … 64.bht |  |

图2.4.1 服务器端子地形块原始资源文件清单

MapResManager保存地图生成依赖的静态资源数据，包括地图类型、地图生成规则配置、怪物布置、寻路、地形等数据，程序启动时从资源文件中加载。对于固定地图模式提供GetUniqueXXX接口访问全局唯一地图资源；对于随机地图模式提供CreateTerrain/CreateMoveMap等接口，内部组织CNPCGenMan、CMap、trace\_manager2 、CTerrain等拼接子地图资源创建新地图数据，地图产生新的资源数据保存在所属的world类中。



图 2.4.2 地图资源管理类结构图

map\_generator(图2.4.3)保存MazeGen生成的子地图序列：



图 2.4.3 子地图序列类结构图

图 2.4.4 拼接后的地图类world与其它类的关系图

具体的地形数据拼接方式如下：

* **CTerrain-地形高度数据:**

高度图由地图上等距离的离散的点对应的高度值组成，保存为1维数组。

以1024\*1024大小的地图为例，假定每个离散点代表的地形范围是2\*2米，则共(512+1)\*(512+1)个float值。考虑到占用的内存较少，所以采用数据拷贝的方式生成新terrian，将MapResManager中的子地图块数据拷贝到新terrain的相应位置上即可；

* CNPCMoveMap**-NPC地面寻路数据**

通过二维数组来保存分块地图的数据：

1. 位图模式保存地图上的点是否可达(每个CBitImage精度可为1米\*1米)：

CBitImage \*\*m\_ReachableMap;

1. 使用压缩方式保存可达点与地形高度差：

CBlockImage<FIX16>\*\* m\_DeltaHeightMap;

由于占用内存较多，因此采用数据引用的方式生成新的CNPCMoveMap，新CNPCMoveMap的数据指针指向MapResManager中的数据。对于8\*8的子地图配置，数组长度为64；

* **CGlobalWaterAreaMap-NPC水域寻路数据**

同CNPCMoveMap；

* **CGlobalSPMap-NPC空间寻路数据**

每个子地图是正方体空间八叉树，实时从多个子地图块生成耗时间更耗存储，考虑实际地图空间阻挡较少，所以拼接后初始化为没有阻挡；

* **trace\_manager-凸包碰撞检测数据**

同CNPCMoveMap，按子地图排列安排新trace\_manager中数据指针指向MapResManager中相应的子地图静态数据；每次AABBTrace前先计算碰撞经过的所有子地图（InterSection类提供），然后逐个进行碰撞检测，直到产生碰撞为止；如果子地图块中的碰撞凸包跨越子地图边界，则会地相邻块产生影响，因此需要从两种约定中选择一种，所有子地图碰撞凸包都不会越界、或者所有子地图块碰撞凸包都包含相同的越界碰撞凸包；



图 2.4.5 InterSection类结构图

**InterSection**类使用的算法参考bresenham’s line algorithm，取得源到目的2点有向线段穿过的子地图块索引数组:

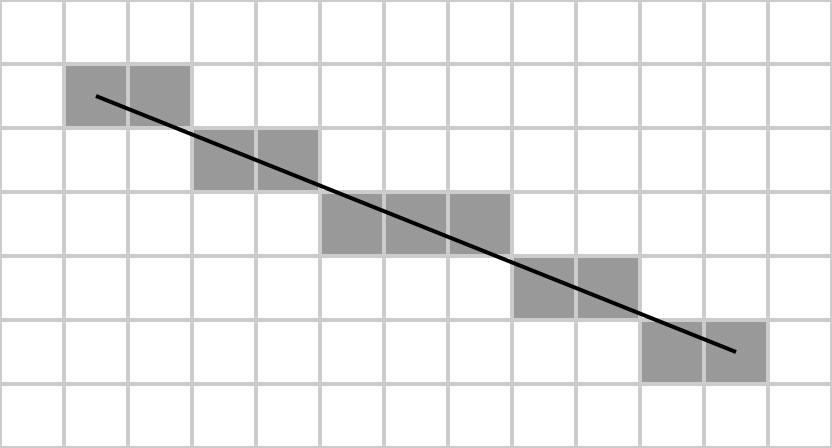


图2.4.6InterSection类判断跨越子块的算法

1. 首先按源和目的两点算出直线方程：

intstart\_u = (int) (start.x / width);

intstart\_v = (int) (start.z / height);

intend\_u = (int) ((start.x + offset.x) / width);

intend\_v = (int) ((start.z + offset.z) / height);

float k = offset.z/offset.x;

float b = start.z - k\*start.x;

boolnegative\_u = offset.x< 0;

boolnegative\_v = offset.z< 0;

intstep\_u = negative\_u ? -1 : 1;

intstep\_v = negative\_v ? -1 : 1;

1. 在与各子地图块边界重合的轴向上分别求出与线段的交点cross\_point，删除重合的交点，并按到线段起点的距离排序:

Pos2D cross\_point\_modify;

Pos2D cross\_point;

cross\_point.uaxis = true;

cross\_point.v = negative\_v ? (start\_v - 1)\*height :start\_v \* height;

for(int n = start\_v; n != end\_v; n+=step\_v, cross\_point.v += (step\_v \* height )){

cross\_point.u = (cross\_point.v - b)/k;

cross\_point.uvc = fmod(cross\_point.u, width) < 0.001f;

cross\_point\_modify = cross\_point;

cross\_point\_modify.v -= 0.001f; // 0.001f防止float->int时候的误差

if(cross\_point.uvc) cross\_point\_modify.u += 0.001f; cross\_point\_set[start.horizontal\_distance(A3DVECTOR(cross\_point.u,0,cross\_point.v))] = cross\_point\_modify;

}

cross\_point.u = negative\_u ?start\_u \* width : (start\_u + 1) \* width;

cross\_point.uaxis = false;

for(int n = start\_u; n != end\_u; n+=step\_u, cross\_point.u += (step\_u \* width)) {

cross\_point.v = cross\_point.u \*k + b;

cross\_point.uvc = -fmod(cross\_point.v, height) < 0.001f;

cross\_point\_modify = cross\_point;

cross\_point\_modify.u += 0.001f; // 0.001f防止float->int时候的误差

if(cross\_point.uvc)cross\_point\_modify.v -= 0.001f; cross\_point\_set[start.horizontal\_distance(A3DVECTOR(cross\_point.u,0,cross\_point.v))] = cross\_point\_modify;

}

1. 按交点类型获取子地图块索引坐标：
2. 横向轴向上的交点取上下两个子块；
3. 纵向轴向上的交点取左右两个子块；
4. 重合的交点取线段穿过的两个对角子块；

* **npcgenerator-NPC生成管理器**

图2.4.8 npcgenerator类与场景管理类的关系图

NPC生成管理器分为控制器和孵化器两部分，孵化器决定在何处生成多少个ID是多少的NPC,而控制器决定何时启用哪些孵化器。孵化器一般与子地形相关，可通过地图编辑器分别导出在各子地形块的npcgen\_1.data~npcgen\_64.data文件中；而控制器全图导出一份在npcgen.data中即可。在子地形块拼接中，主要面临的问题是，同一块子地形块在最终拼接的场景中多次出现的问题、以及孵化器生成的NPC重新定位的问题。对于第二个问题，生成NPC时位置按子地图块中心点和地图中心点偏移做修正即可。

对于第一个问题，首先，当子地形块被多次使用时，每使用一次都必须复制相应的孵化器、及使用了此孵化器的控制器，并按以下公式生成复制后的新控制器的ID：

unique\_ctrl\_id = (ctrl\_id&0xFFFFFF) + (block\_id<< 24)& 0xFF000000（公式 2.4.1）

其中，ctrl\_id为复制前的控制器ID，block\_id为子地形块ID，unique\_ctrl\_id为复制后的控制器惟一ID。当游戏其它模块持原控制器ID对各子地形块复制的新控制器引用时，必须先取得子地形块ID。

## 向客户端同步随机结果

在进入随机关卡场景前，客户端需要从服务器同步子地图索引序列以初始化客户端资源。客户端进入的时机有两个，一为选取角色后直接进入场景PlayerEnterWorld，另一为场景间相互跳转PlayerEnterServer，为不影响正常场景的进入逻辑，在map\_generator上提供SyncPlayerWorldGen虚接口。

图2.5.1 随机关卡场景切换组件引用图

对于非随机关卡，不需要包含map\_generator组件；随机关卡的map\_generator按派生类的实现同步客户端，需要保证此信息要在其他角色信息之前同步完成。

## 客户端地形资源拼接

主要有地形数据、建筑物等场景数据、碰撞凸包三部分：

* 1. 地形数据预处理：

引擎完全封装了对地形的处理逻辑，所以客户端必须在加载地形之前，根据收到服务器传过来的随机序列对地形文件进行预处理，把地形块重新排列，产生新的地形文件共引擎使用。处理逻辑主要由几下几个类实现：

CECRandomMapProcess: 地形数据预处理的管理类

CECTerrainPreprocessor：负责处理地形整体信息文件（.trn2）

CECTernBlockPreprocessor：负责处理地形block的信息。引擎使用block组织地形数据，整张地形被划分row\*col个block。CECTernBlockPreprocessor根据随机后的地形序列，修改block的位置信息。

CECStitchSubTern：负责缝合地形块。遍历每个subTerrain，对边缘的block块，找出处在其他subTerrain的相邻block，并进行缝合处理。

图2.6.1 客户端地形拼接处理模块结构

以上几个类对参考地形文件处理，存盘为供引擎类A3DTerrain2使用的一系列地形文件（后缀为.t2nr,.t2bk等等），交由引擎加载这些文件即可，原来的客户端逻辑都可以复用。

* 1. 场景对象（建筑物等）数据：

客户端场景内部采用block管理数据，整张地图被划分为row\*col个block。每个block都保存了处在这个区域里的对象数据，比如建筑物、特效对象等。主角在地图里移动时计算出视野范围内所有block块序号，并根据服务器传过来的子地形随机序列计算出对应的新block块，加载新的block块即可。需要注意的是：

1. 由于各种场景对象ID都是唯一的，但子地形块随机过程中有机会复用，对这种情况的对象ID需要做特殊处理；
2. 编辑器导出的场景对象位置坐标是绝对坐标，在加载block时需要进行映射处理；
   1. 建筑物碰撞凸包：

由于和每个建筑物一一对应，在动态加载建筑物模型的时候，对相应的凸包也做相应的位置平移。

## 正确操纵拼接后关卡的怪物策略

怪物策略使用控制器spawner\_ctrl操作预先配置好的孵化器base\_spawn：



图2.7.1控制器类结构图



图2.7.2孵化器基类结构图

随机关卡产生时，拼接后的多个子关卡可能会映射到同一份拼接前的子关卡上，为使怪物策略能正确控制游戏内容，需保证每个策略涉及的对象不会跨越多个子关卡，这样，每份子关卡的控制器都能独自运作、而不会因依赖其它子关卡导致未知的运行结果。

当怪物策略执行到需要操纵子地形块的控制器时，策略传入的控制器ID要先根据怪物关联的子地图ID按公式2.4.1做修正为全局惟一的ID，再进行激活等操作。