bool CRandMapProp::SaveSev(const char\* szPath)

{

if(!szPath || strlen(szPath)<1) return false;

AString str;

str.Format("%s\\map\_desp.sev",szPath);

FILE\* pf = fopen(str,"wb");

if(pf==NULL) return false;

fwrite(&g\_randMapVersion,sizeof(int),1,pf);

fwrite(&m\_header,1,sizeof(m\_header),pf);

for (int i=0;i<m\_header.iTileCount;i++)

{

MAP\_INFO info = m\_GridProp[i];

fwrite(&info.index,sizeof(int),1,pf);

fwrite(&info.type,sizeof(int),1,pf);

fwrite(&info.connection,sizeof(int),1,pf);

}

fclose(pf);

return true;

}

bool CRandMapProp::SaveClt(const char\* szPath)

{

if(!szPath || strlen(szPath)<1) return false;

AString str;

str.Format("%s\\map\_desp.clt",szPath);

FILE\* pf = fopen(str,"wb");

if(pf==NULL) return false;

// Write Unicode header

wchar\_t wcFlag = 0xfeff;

fwprintf(pf, L"%c", wcFlag);

fwprintf(pf, L"// Element random map file (client version)\n\n");

// Write version

fwprintf(pf, L"version %d\n", g\_randMapVersion);

fwprintf(pf,L"%d\n",m\_GridProp.size());

for (int i=0;i<m\_GridProp.size();i++)

{

MAP\_INFO info = m\_GridProp[i];

AWString strName = AS2WC(info.name);

if (!strName.GetLength())

strName = L"Unknown Area";

fwprintf(pf,L"%d %s\n", info.index,strName);

}

fclose(pf);

return true;

}

bool CRandMapProp::Save()

{

CString str;

str.Format("%s\\randommap.dat",g\_szEditMapDir);

if(AUX\_FileIsReadOnly(str)) return false;

// if(AUX\_CheckOutRandomMapInfoFile())

{

AString str;

str.Format("%s\\randommap.dat",g\_szEditMapDir);

FILE\* pf = fopen(str,"w");

if(pf==NULL) return false;

fprintf(pf,"version: %d\n",g\_randMapVersion);

str.Format("%d %d %d %d %d %d %d %d\n",m\_header.iTileCount,m\_header.iTileSize,m\_header.mainLineMaxLen,m\_header.mainLineMinLen,m\_header.branchLineNumMax,m\_header.branchLineNumMin,

m\_header.branchLineMaxLen,m\_header.branchLineMinLen);

fprintf(pf,str.GetBuffer(str.GetLength()));

str.Format("%f,%f,%f\n",m\_header.fPosX,m\_header.fPosY,m\_header.fPosZ);

fprintf(pf,str.GetBuffer(str.GetLength()));

for(int i = 0; i < m\_GridProp.size(); i++)

{

char szLine[128];

MAP\_INFO info = m\_GridProp[i];

sprintf(szLine,"%d %d %d %s\n",info.index+1,info.type,info.connection,info.name);

fprintf(pf,szLine);

}

fclose(pf);

}

return true;

}

bool CRandMapProp::InitGird(int c)

{

CString str;

str.Format("%s%s",g\_szWorkDir,g\_szEditMapDir);

g\_VSS.SetWorkFolder(str);

g\_VSS.SetProjectPath(str);

str.Format("%s\\randommap.dat",g\_szEditMapDir);

g\_VSS.GetFile(str);

AScriptFile sf;

if(!sf.Open(str))

return false;

CElementMap trnMap;

AString mapName;

mapName.Format("%sEditMaps\\%s\\1\\1.elproj",g\_szWorkDir,g\_Configs.szCurProjectName);

bool bHasTerrain = true;

if(!trnMap.Load(mapName))

{

CString err;

err.Format("打开%s失败",mapName);

AfxMessageBox(err);

// return false;

bHasTerrain = false;

}

int terrainSize = bHasTerrain ? (int)trnMap.GetTerrain()->GetTerrainSize() : 128;

trnMap.Release();

m\_GridProp.clear();

memset(&m\_header,0,sizeof(m\_header));

int i,mapVer = 0;

if(sf.MatchToken("version:",false))

{

mapVer = sf.GetNextTokenAsInt(false);

if (mapVer>1)

{

m\_header.iTileCount = sf.GetNextTokenAsInt(true);

m\_header.iTileSize = sf.GetNextTokenAsInt(false);

m\_header.mainLineMaxLen = sf.GetNextTokenAsInt(false);

m\_header.mainLineMinLen = sf.GetNextTokenAsInt(false);

m\_header.branchLineNumMax = sf.GetNextTokenAsInt(false);

m\_header.branchLineNumMin = sf.GetNextTokenAsInt(false);

m\_header.branchLineMaxLen = sf.GetNextTokenAsInt(false);

m\_header.branchLineMinLen = sf.GetNextTokenAsInt(false);

if (mapVer>=3)

{

m\_header.fPosX = sf.GetNextTokenAsFloat(true);

m\_header.fPosY = sf.GetNextTokenAsFloat(false);

m\_header.fPosZ = sf.GetNextTokenAsFloat(false);

}

}

m\_header.iTileCount = c;

m\_header.iTileSize = terrainSize;

for (i=0;i<c;i++)

{

if(sf.IsEnd())

break;

int mapID,mapType,mapCon;

mapID = sf.GetNextTokenAsInt(true);

mapType = sf.GetNextTokenAsInt(false);

mapCon = sf.GetNextTokenAsInt(false);

sf.GetNextToken(false);

if (mapID>=1)

{

MAP\_INFO info;

info.index = mapID - 1;

info.type = mapType;

info.connection = mapCon;

info.name = sf.m\_szToken;

m\_GridProp.push\_back(info);

}

}

}

sf.Close();

return i>=c;

}

bool CRandMapProp::OnCreateOneGrid(const char\* szProjName)

{

if(!AUX\_CheckOutRandomMapInfoFile())

{

AfxMessageBox("CDlgNewMap::OnOK(),Check out randommap.dat failed!");

return false;

}

char szProj[256];

sprintf(szProj,"%s\\randommap.dat",g\_szEditMapDir);

FILE\* fp = fopen(szProj, "a");

if(!fp)

return false;

fprintf(fp,"%s 3 0 maze\n",szProjName); // 名字必须为数字 1,2,3,4,5。。。 保持以往惯例

fclose(fp);

if(!AUX\_CheckInRandomMapInfoFile())

{

AfxMessageBox("CDlgNewMap::OnOK(),Check in randommap.dat failed!");

return false;

}

return true;

}

bool CRandMapProp::CreateRandomMapFile(const char\* szDst)

{

FILE\* mapFile = fopen(szDst,"w");

if (mapFile==NULL)

return false;

fprintf(mapFile,"version: %d\n",g\_randMapVersion);

AString str;

str.Format("0 0 0 0 0 0 0 0\n");

fprintf(mapFile,str.GetBuffer(str.GetLength()));

fclose(mapFile);

return true;

}

#elif \_ELEMENTCLIENT

#include <AWScriptFile.h>

bool CRandMapProp::LoadClt(const char\* szPath)

{

if(!szPath) return false;

// AString str;

// str.Format("%s\\map\_desp.clt",szPath);

AWScriptFile cltFile;

if (!cltFile.Open(szPath))

return false;

// Get version

if (!cltFile.MatchToken(L"version", false))

return false;

m\_GridProp.clear();

int iVersion = cltFile.GetNextTokenAsInt(false);

int count = cltFile.GetNextTokenAsInt(false);

while (cltFile.PeekNextToken(true))

{

MAP\_INFO info;

info.index = cltFile.GetNextTokenAsInt(true);

cltFile.GetNextToken(false);

info.name = cltFile.m\_szToken;

m\_GridProp.push\_back(info);

}

cltFile.Close();

return m\_GridProp.size() == count;

}

#else

bool CRandMapProp::LoadSev(const char\* szPath)

{

if(!szPath) return false;

// char filePath[128] = {0};

// sprintf(filePath,"%s\\map\_desp.sev",szPath);

FILE\* pf = fopen(szPath,"rb");

if(!pf) return false;

m\_GridProp.clear();

int ver=0;

fread(&ver,sizeof(int),1,pf);

if(ver>=3)

fread(&m\_header,sizeof(FILEHEADER),1,pf);

else

{

FILEHEADER2 h;

fread(&h,sizeof(h),1,pf);

m\_header.iTileCount = h.iTileCount;

m\_header.iTileSize = h.iTileSize; // 每块大小(米)

m\_header.mainLineMaxLen = h.mainLineMaxLen; // 主线最大长度

m\_header.mainLineMinLen = h.mainLineMinLen; // 主线最小长度

m\_header.branchLineNumMax = h.branchLineNumMax; // 支线数最大值

m\_header.branchLineNumMin = h.branchLineNumMin; // 支线数最小值

m\_header.branchLineMaxLen = h.branchLineMaxLen; // 支线长度最大值

m\_header.branchLineMinLen = h.branchLineMinLen;

m\_header.fPosX = 0.0f;

m\_header.fPosY = 0.0f;

m\_header.fPosZ = 0.0f;

}

for (int i=0;i<m\_header.iTileCount;i++)

{

MAP\_INFO info;

fread(&info.index,sizeof(int),1,pf);

fread(&info.type,sizeof(int),1,pf);

fread(&info.connection,sizeof(int),1,pf);

m\_GridProp.push\_back(info);

}

fclose(pf);

return true;

}

#endif

bool CRandMapProp::GetGirdProp(int iGrid,MAP\_INFO& info)

{

if(iGrid<0 || iGrid>=m\_GridProp.size())

return false;

info = m\_GridProp[iGrid];

return true;

}

void CRandMapProp::SetGridProp(int iGrid, MAP\_INFO& info)

{

if(iGrid<0 || iGrid> m\_GridProp.size())

return;

m\_GridProp[iGrid] = info;

}

bool CSingleMapGridSelectPolicy::OnDataChanged(int iOld,int iNew)

{

CString str;

str.Format("%s\\randommap.dat",g\_szEditMapDir);

if(m\_randmapProp==NULL || m\_pDlgSetting == NULL) return false;

m\_pDlgSetting->UpdateData(TRUE);

CRandMapProp::MAP\_INFO info;

info.type = m\_pDlgSetting->m\_iMapType;

info.connection = m\_pDlgSetting->GetMask();

info.name = m\_pDlgSetting->m\_strMapName;

if (iOld != -1)

{

CRandMapProp::MAP\_INFO oldInfo;

bool bHas = m\_randmapProp->GetGirdProp(iOld,oldInfo);

if (bHas && !(info == oldInfo))

{

if (info.name.GetLength()<1)

{

AfxMessageBox("名字不能为空");

return false;

}

if(AUX\_FileIsReadOnly(str))

{

AfxMessageBox("修改数据请先签出");

return false;

}

int ret = AfxMessageBox("确定要保存修改吗?",MB\_OKCANCEL);

if (ret == MB\_OKCANCEL)

{

info.index = oldInfo.index;

m\_randmapProp->SetGridProp(iOld,info);

}

}

}

bool bHas = m\_randmapProp->GetGirdProp(iNew,info);

if(!bHas)

{

info.connection = CRandMapProp::GRID\_CONNECT\_NONE;

info.type = CRandMapProp::GRID\_TYPE\_NOUSE; // no use

info.name = "maze";

}

m\_pDlgSetting->m\_iMapType = info.type;

m\_pDlgSetting->SetMask(info.connection);

m\_pDlgSetting->m\_strMapName = info.name;

m\_pDlgSetting->UpdateData(FALSE);

return true;

}

bool CSingleMapGridSelectPolicy::OnRButtonDown(int iTileMouseOn, UINT nFlags)

{

OnDataChanged(m\_iSelectedIndex);

ClearSelect();

return true;

}

bool CSingleMapGridSelectPolicy::OnLButtonDown(int iTileMouseOn, UINT nFlags)

{

bool bRet(false);

if (CanSelect(iTileMouseOn) && m\_iSelectedIndex != iTileMouseOn)

{

if(OnDataChanged(m\_iSelectedIndex,iTileMouseOn))

{

m\_iSelectedIndex = iTileMouseOn;

bRet = true;

}

}

return bRet;

}

bool CSingleMapGridSelectPolicy::OnMouseMove(int iTileMouseOn, UINT nFlags)

{

return false;

}

void CSingleMapGridSelectPolicy::ResetMore()

{

ClearSelect();

}

void CSingleMapGridSelectPolicy::ClearSelectEnableMore(bool bEnable)

{

ClearSelect();

}

BEGIN\_MESSAGE\_MAP(CRandomMapGird, CMapGrid)

//{{AFX\_MSG\_MAP(CMapGrid)

ON\_WM\_RBUTTONDOWN()

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

void CRandomMapGird::OnRButtonDown(UINT nFlags, CPoint point)

{

if (m\_pSelectPolicy)

{

m\_pSelectPolicy->OnRButtonDown(-1,nFlags);

Invalidate();

}

}

int CRandomMapGird::GetSelectIndex()

{

if (m\_pSelectPolicy)

{

CSingleMapGridSelectPolicy\* p = (CSingleMapGridSelectPolicy\*)m\_pSelectPolicy;

if(p)

return p->GetSelectIndex();

}

return -1;

}

// Draw back buffer

bool CRandomMapGird::DrawBackBuffer(RECT\* lprc)

{

// Clear background

ClearBackBuffer(0xffffffff);

if (!m\_iNumTile)

return true;

RECT rcClient;

GetClientRect(&rcClient);

int iMaxRow = m\_iOffsetY + (rcClient.bottom - SIZE\_RULERTHICK + m\_iTileSize - 1) / m\_iTileSize;

int iMaxCol = m\_iOffsetX + (rcClient.right - SIZE\_RULERTHICK + m\_iTileSize - 1) / m\_iTileSize;

a\_ClampRoof(iMaxRow, m\_iNumRow);

a\_ClampRoof(iMaxCol, m\_iNumCol);

int i, x, y;

HPEN hOldPen, hPen;

hPen = ::CreatePen(PS\_SOLID, 1, RGB(0, 0, 0));

hOldPen = (HPEN)::SelectObject(m\_hMemDC, hPen);

// Draw horizonal table line

x = SIZE\_RULERTHICK + (m\_iNumCol - m\_iOffsetX) \* m\_iTileSize + 1;

x = a\_Min(x, (int)rcClient.right);

y = SIZE\_RULERTHICK;

for (i=m\_iOffsetY; i <= iMaxRow; i++, y+=m\_iTileSize)

{

MoveToEx(m\_hMemDC, 0, y, NULL);

LineTo(m\_hMemDC, x, y);

}

// Draw vertical table line

x = SIZE\_RULERTHICK;

y = SIZE\_RULERTHICK + (m\_iNumRow - m\_iOffsetY) \* m\_iTileSize + 1;

y = a\_Min(y, (int)rcClient.bottom);

for (i=m\_iOffsetX; i <= iMaxCol; i++, x+=m\_iTileSize)

{

MoveToEx(m\_hMemDC, x, 0, NULL);

LineTo(m\_hMemDC, x, y);

}

// Draw horizonal ruler scale

char szText[100];

RECT rc = {SIZE\_RULERTHICK, 0, 0, SIZE\_RULERTHICK};

for (i=m\_iOffsetX; i < iMaxCol; i++)

{

itoa(i, szText, 10);

rc.right = rc.left + m\_iTileSize;

::DrawText(m\_hMemDC, szText, strlen(szText), &rc, DT\_SINGLELINE | DT\_VCENTER | DT\_CENTER);

rc.left += m\_iTileSize;

}

// Draw vertical ruler scale

::SetRect(&rc, 0, SIZE\_RULERTHICK, SIZE\_RULERTHICK, 0);

for (i=m\_iOffsetY; i < iMaxRow; i++)

{

itoa(i, szText, 10);

rc.bottom = rc.top + m\_iTileSize;

::DrawText(m\_hMemDC, szText, strlen(szText), &rc, DT\_SINGLELINE | DT\_VCENTER | DT\_CENTER);

rc.top += m\_iTileSize;

}

::SelectObject(m\_hMemDC, hOldPen);

::DeleteObject(hPen);

// Fill map tiles

for (y=m\_iOffsetY; y < iMaxRow; y++)

{

int iCount = y \* m\_iNumCol + m\_iOffsetX;

rc.left = SIZE\_RULERTHICK + 1;

rc.top = SIZE\_RULERTHICK + (y - m\_iOffsetY) \* m\_iTileSize + 1,

rc.bottom = rc.top + m\_iTileSize - 1;

for (x=m\_iOffsetX; x < iMaxCol; x++, iCount++)

{

rc.right = rc.left + m\_iTileSize - 1;

CString projName;

GetProjName(iCount,projName);

if(m\_pGridDrawPolicy)

m\_pGridDrawPolicy->DrawGird(projName,iCount,IsSelected(iCount), m\_hMemDC,rc);

rc.left += m\_iTileSize;

}

}

return true;

}

//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////

bool CRandomMapGridDrawPolicy::GetConnectMark(int grid\_con,int c,RECT& rc)

{

if ((c & grid\_con)==c)

{

switch (c)

{

case CRandMapProp::GRID\_CONNECT\_TOP:

rc.top += 2;

rc.bottom = rc.top + 6;

rc.left = (rc.left + rc.right) / 2 - 3;

rc.right = rc.left + 6;

break;

case CRandMapProp::GRID\_CONNECT\_BOTTOM:

rc.bottom -= 2;

rc.top = rc.bottom - 6;

rc.left = (rc.left + rc.right) / 2 - 3;

rc.right = rc.left + 6;

break;

case CRandMapProp::GRID\_CONNECT\_LEFT:

rc.left += 2;

rc.right = rc.left + 6;

rc.top = (rc.top + rc.bottom) / 2 - 3;

rc.bottom = rc.top + 6;

break;

case CRandMapProp::GRID\_CONNECT\_RIGHT:

rc.right -= 2;

rc.left = rc.right - 6;

rc.top = (rc.top + rc.bottom) / 2 - 3;

rc.bottom = rc.top + 6;

break;

}

return true;

}

return false;

}

static COLORREF brushColor[CRandMapProp::GRID\_TYPE\_NUM+1] = {

RGB(255, 255, 255), // normal

RGB(255, 255, 0), //start

RGB(0,255,255), // end

RGB(64, 64, 64), // no use

RGB(128,128,255),

RGB(128,255,128),

RGB(0,128,255),

RGB(255, 0, 0), // sel

};

static COLORREF penColor = RGB(0,0,0);

CRandomMapGridDrawPolicy::CRandomMapGridDrawPolicy(CRandMapProp\* prop):m\_randMapProp(prop)

{

for (int i=0;i<=CRandMapProp::GRID\_TYPE\_NUM;i++)

{

hBrush[i] = AUX\_CreateBrush(BS\_SOLID,brushColor[i],0);

}

hPen = ::CreatePen(PS\_SOLID, 1, penColor);

}

CRandomMapGridDrawPolicy::~CRandomMapGridDrawPolicy()

{

for(int i=0;i<=CRandMapProp::GRID\_TYPE\_NUM;i++)

::DeleteObject(hBrush[i]);

::DeleteObject(hPen);

}

void CRandomMapGridDrawPolicy::DrawGird(const CString& name,int iGrid,bool bSel,HDC hDC,RECT& rc)

{

COLORREF col = ::GetBkColor(hDC);

if (m\_randMapProp)

{

CRandMapProp::MAP\_INFO info;

m\_randMapProp->GetGirdProp(iGrid,info);

::SetBkColor(hDC, brushColor[info.type]);

::FillRect(hDC, &rc, hBrush[info.type]);

}

if (bSel)

{

::SetBkColor(hDC, brushColor[CRandMapProp::GRID\_TYPE\_NUM]);

::FillRect(hDC, &rc, hBrush[CRandMapProp::GRID\_TYPE\_NUM]);

}

if (m\_randMapProp)

{

CRandMapProp::MAP\_INFO info;

m\_randMapProp->GetGirdProp(iGrid,info);

HPEN oldPen = (HPEN)::SelectObject(hDC,hPen);

int con = CRandMapProp::GRID\_CONNECT\_LEFT;

while(con<=CRandMapProp::GRID\_CONNECT\_BOTTOM)

{

RECT markRC = rc;

if(GetConnectMark(info.connection,con,markRC))

::Ellipse(hDC,markRC.left,markRC.top,markRC.right,markRC.bottom);

con<<=1;

}

::SelectObject(hDC,oldPen);

}

::DrawText(hDC, name, name.GetLength(), &rc, DT\_SINGLELINE | DT\_VCENTER | DT\_CENTER);

::SetBkColor(hDC, col);

}

#ifdef \_DEBUG

#define new DEBUG\_NEW

#undef THIS\_FILE

static char THIS\_FILE[] = \_\_FILE\_\_;

#endif

/////////////////////////////////////////////////////////////////////////////

// CDlgRandomMapInfoSetting dialog

CDlgRandomMapInfoSetting::CDlgRandomMapInfoSetting(CWnd\* pParent /\*=NULL\*/)

: CDialog(CDlgRandomMapInfoSetting::IDD, pParent)

{

//{{AFX\_DATA\_INIT(CDlgRandomMapInfoSetting)

m\_iMapType = 3;

for (int i=0;i<CRandMapProp::GRID\_CON\_NUM;i++)

{

m\_mkCon[i] = FALSE;

}

m\_iMainLineMinLen = 0;

m\_iMainLineMaxLen = 0;

m\_iBranchMinCount = 0;

m\_iBranchMaxCount = 0;

m\_iBranchMinLen = 0;

m\_iBranchMaxLen = 0;

m\_fPosX = 0.0f;

m\_fPosY = 0.0f;

m\_fPosZ = 0.0f;

//}}AFX\_DATA\_INIT

}

void CDlgRandomMapInfoSetting::DoDataExchange(CDataExchange\* pDX)

{

CDialog::DoDataExchange(pDX);

//{{AFX\_DATA\_MAP(CDlgRandomMapInfoSetting)

DDX\_Radio(pDX, IDC\_RAD\_TYPE, m\_iMapType);

DDX\_Check(pDX, IDC\_CHK\_CONNECT1, m\_mkCon[0]);

DDX\_Check(pDX, IDC\_CHK\_CONNECT2, m\_mkCon[1]);

DDX\_Check(pDX, IDC\_CHK\_CONNECT3, m\_mkCon[2]);

DDX\_Check(pDX, IDC\_CHK\_CONNECT4, m\_mkCon[3]);

DDX\_Control(pDX, IDC\_RAD\_TYPE, m\_crtlMapType[0]);

DDX\_Control(pDX, IDC\_RAD\_TYPE2, m\_crtlMapType[1]);

DDX\_Control(pDX, IDC\_RAD\_TYPE3, m\_crtlMapType[2]);

DDX\_Control(pDX, IDC\_RAD\_TYPE4, m\_crtlMapType[3]);

DDX\_Control(pDX, IDC\_RAD\_TYPE5, m\_crtlMapType[4]);

DDX\_Control(pDX, IDC\_RAD\_TYPE6, m\_crtlMapType[5]);

DDX\_Control(pDX, IDC\_RAD\_TYPE7, m\_crtlMapType[6]);

DDX\_Text(pDX,IDC\_EDIT\_NAME,m\_strMapName);

DDX\_Text(pDX,IDC\_MAINLINE\_MIN\_LEN,m\_iMainLineMinLen);

DDX\_Text(pDX,IDC\_MAINLINE\_MAX\_LEN,m\_iMainLineMaxLen);

DDX\_Text(pDX,IDC\_BRANCH\_MIN\_COUNT,m\_iBranchMinCount);

DDX\_Text(pDX,IDC\_BRANCH\_MAX\_COUNT,m\_iBranchMaxCount);

DDX\_Text(pDX,IDC\_BRANCH\_MIN\_LEN,m\_iBranchMinLen);

DDX\_Text(pDX,IDC\_BRANCH\_MAX\_LEN,m\_iBranchMaxLen);

DDX\_Text(pDX,IDC\_START\_POSX,m\_fPosX);

DDX\_Text(pDX,IDC\_START\_POSY,m\_fPosY);

DDX\_Text(pDX,IDC\_START\_POSZ,m\_fPosZ);

//}}AFX\_DATA\_MAP

}

BEGIN\_MESSAGE\_MAP(CDlgRandomMapInfoSetting, CDialog)

//{{AFX\_MSG\_MAP(CDlgRandomMapInfoSetting)

ON\_WM\_PAINT()

ON\_BN\_CLICKED(IDC\_BTN\_OK, OnBtnOk)

ON\_BN\_CLICKED(IDC\_BTN\_CHECKOUT, OnBtnCheckout)

ON\_BN\_CLICKED(IDC\_BTN\_CHECKIN, OnBtnCheckin)

ON\_EN\_CHANGE(IDC\_MAINLINE\_MIN\_LEN, OnChangeEdit)

ON\_EN\_CHANGE(IDC\_MAINLINE\_MAX\_LEN, OnChangeEdit)

ON\_EN\_CHANGE(IDC\_BRANCH\_MIN\_COUNT, OnChangeEdit)

ON\_EN\_CHANGE(IDC\_BRANCH\_MAX\_COUNT, OnChangeEdit)

ON\_EN\_CHANGE(IDC\_BRANCH\_MIN\_LEN, OnChangeEdit)

ON\_EN\_CHANGE(IDC\_BRANCH\_MAX\_LEN, OnChangeEdit)

ON\_EN\_CHANGE(IDC\_START\_POSX,OnChangeEdit)

ON\_EN\_CHANGE(IDC\_START\_POSY,OnChangeEdit)

ON\_EN\_CHANGE(IDC\_START\_POSZ,OnChangeEdit)

ON\_WM\_DESTROY()

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

/////////////////////////////////////////////////////////////////////////////

// CDlgRandomMapInfoSetting message handlers

void CDlgRandomMapInfoSetting::OnBtnCheckout()

{

bool ret = AUX\_CheckOutRandomMapInfoFile();

AfxMessageBox(ret ? "签出成功":"签出失败");

}

void CDlgRandomMapInfoSetting::OnBtnCheckin()

{

if (m\_pMapGridSelectPolicy)

{

m\_pMapGridSelectPolicy->OnRButtonDown(-1,0);

}

bool ret = !m\_randMapGridProp.Save()|| !AUX\_CheckInRandomMapInfoFile();

AfxMessageBox(ret ? "签入失败":"签入成功");

m\_mapGrid.Invalidate(FALSE);

}

void CDlgRandomMapInfoSetting::OnCancel()

{

OnBtnOk();

}

void CDlgRandomMapInfoSetting::OnBtnOk()

{

UpdateData(TRUE);

if (m\_pMapGridSelectPolicy)

{

m\_pMapGridSelectPolicy->OnRButtonDown(-1,0);

}

if (m\_randMapGridProp.Save())

{

AUX\_CheckInRandomMapInfoFile();

}

CDialog::OnOK();

}

BOOL CDlgRandomMapInfoSetting::OnInitDialog()

{

CDialog::OnInitDialog();

//

m\_pMapGridSelectPolicy = NULL;

m\_pMapGridDrawProlicy = NULL;

if (!m\_projList.Init()){

g\_Log.Log("CDlgRandomMapInfoSetting::OnInitDialog, Failed to init ProjList.dat.");

EndDialog(IDCANCEL);

return FALSE;

}

m\_randMapGridProp.InitGird(m\_projList.GetListCount());

//

RECT rcWnd;

GetDlgItem(IDC\_STATIC\_MAP)->GetWindowRect(&rcWnd);

ScreenToClient(&rcWnd);

m\_pMapGridSelectPolicy = new CSingleMapGridSelectPolicy(this,&m\_randMapGridProp);

m\_mapGrid.SetSelectPolicy(m\_pMapGridSelectPolicy);

if (!m\_mapGrid.Create(this, rcWnd, 40))

{

g\_Log.Log("CDlgRandomMapInfoSetting::OnInitDialog, Failed to create map grid window.");

EndDialog(IDCANCEL);

return FALSE;

}

m\_mapGrid.SetTileAndRow(m\_projList.GetList(), m\_projList.GetNumCol(), true);

m\_pMapGridDrawProlicy = new CRandomMapGridDrawPolicy(&m\_randMapGridProp);

m\_mapGrid.SetDrawPolicy(m\_pMapGridDrawProlicy);

m\_mapGrid.InvalidateRect(NULL);

CRandMapProp::MAP\_INFO info;

if(m\_randMapGridProp.GetGirdProp(m\_mapGrid.GetSelectIndex(),info))

{

m\_iMapType = info.type;

SetMask(info.connection);

}

SetHeaderInfo(m\_randMapGridProp.GetHeader());

UpdateData(FALSE);

return TRUE; // return TRUE unless you set the focus to a control

// EXCEPTION: OCX Property Pages should return FALSE

}

void CDlgRandomMapInfoSetting::OnDestroy()

{

CDialog::OnDestroy();

delete m\_pMapGridSelectPolicy;

delete m\_pMapGridDrawProlicy;

}

void CDlgRandomMapInfoSetting::SetMask(int m)

{

for (int i=0;i<CRandMapProp::GRID\_CON\_NUM;i++)

{

int flag = 1<<i;

m\_mkCon[i] = (m & flag) == flag;

}

}

int CDlgRandomMapInfoSetting::GetMask()

{

int ret = 0;

for (int i=0;i<CRandMapProp::GRID\_CON\_NUM;i++)

{

if (m\_mkCon[i])

{

ret |= 1<<i;

}

}

return ret;

}

void CDlgRandomMapInfoSetting::SetHeaderInfo(const CRandMapProp::FILEHEADER& header)

{

m\_iMainLineMinLen = header.mainLineMinLen;

m\_iMainLineMaxLen = header.mainLineMaxLen;

m\_iBranchMinCount = header.branchLineNumMin;

m\_iBranchMaxCount = header.branchLineNumMax;

m\_iBranchMinLen = header.branchLineMinLen;

m\_iBranchMaxLen = header.branchLineMaxLen;

m\_fPosX = header.fPosX;

m\_fPosY = header.fPosY;

m\_fPosZ = header.fPosZ;

UpdateData(FALSE);

}

void CDlgRandomMapInfoSetting::GetHeaderInfo(CRandMapProp::FILEHEADER& header)

{

UpdateData(TRUE);

header.mainLineMinLen = m\_iMainLineMinLen;

header.mainLineMaxLen = m\_iMainLineMaxLen;

header.branchLineNumMin = m\_iBranchMinCount;

header.branchLineNumMax = m\_iBranchMaxCount;

header.branchLineMinLen = m\_iBranchMinLen;

header.branchLineMaxLen = m\_iBranchMaxLen;

header.fPosX = m\_fPosX;

header.fPosY = m\_fPosY;

header.fPosZ = m\_fPosZ;

}

static int pic\_id[] = {

IDC\_COL\_1,

IDC\_COL\_2,

IDC\_COL\_3,

IDC\_COL\_4,

IDC\_COL\_5,

IDC\_COL\_6,

IDC\_COL\_7

};

void CDlgRandomMapInfoSetting::OnPaint()

{

CDialog::OnPaint();

for (int i=0;i<CRandMapProp::GRID\_TYPE\_NUM;i++)

{

RECT rc;

GetDlgItem(pic\_id[i])->GetClientRect(&rc);

CDC\* pDC = GetDlgItem(pic\_id[i])->GetDC();

pDC->FillSolidRect(rc.left,rc.top,rc.right,rc.bottom,brushColor[i]);

}

}

void CDlgRandomMapInfoSetting::OnChangeEdit()

{

CString str;

str.Format("%s\\randommap.dat",g\_szEditMapDir);

if(AUX\_FileIsReadOnly(str))

{

MessageBox("请先签出");

return;

}

CRandMapProp::FILEHEADER header = m\_randMapGridProp.GetHeader();

GetHeaderInfo(header);

m\_randMapGridProp.SetHeader(header);

}

struct map\_piece\_desp //地图片的描述信息

{

int type;

//...

int joint\_mask; // 联通性mask 1左2上4右8下

};

enum

{

MAPRES\_TYPE\_ORIGIN,

MAPRES\_TYPE\_RANDOM,

MAPRES\_TYPE\_MAZE,

MAPRES\_TYPE\_SEQUENCE,

MAPRES\_TYPE\_COUNT

};

struct npcgen\_data\_node\_t

{

CNPCGenMan\* npcgen;

unsigned char blockid;

A3DVECTOR offset;

npcgen\_data\_node\_t() : npcgen(NULL),blockid(0),offset(0.f,0.f,0.f) {}

npcgen\_data\_node\_t(CNPCGenMan\* ng,unsigned char bid,const A3DVECTOR& of) : npcgen(ng), blockid(bid), offset(of) {}

};

typedef abase::vector<npcgen\_data\_node\_t > npcgen\_data\_list;

struct maze\_info

{

int start\_idx;

int start\_dir;

int end\_idx;

int end\_dir;

int step\_min;

int step\_max;

int branch\_min;

int branch\_max;

int branch\_block\_min;

int branch\_block\_max;

int ring\_min;

int ring\_max;

int empty\_piece;

};

struct map\_res

{

float width;

float height;

A3DVECTOR bpos\_offset;

maze\_info \_maze\_info;

abase::vector<map\_piece\_desp> \_piece\_desps;

map\_res() : width(0.f),height(0.f),bpos\_offset(0,0,0)

{

memset(&\_maze\_info, 0 , sizeof(\_maze\_info));

}

};

class MapResManager;

class map\_generator

{

protected:

int \_row; //生成的地图行数

int \_col; //生成的地图列数

float \_piece\_width; //小块的长

float \_piece\_height; //小块的宽

float \_map\_left; //左上角x值

float \_map\_top; //左上角z值

abase::vector<int> \_piece\_indexes; //生成的地图每一个小片对应于原始资源的索引

map\_generator():\_row(0),\_col(0),\_piece\_width(0.f),\_piece\_height(0.f),\_map\_left(0.f),\_map\_top(0.f) {}

public:

virtual ~map\_generator(){}

int GetRow() const { return \_row; }

int GetCol() const { return \_col; }

const abase::vector<int> & GetPieceIndexes() const { return \_piece\_indexes; }

public:

virtual bool Generate(const rect & region, const map\_res& mapres) = 0;

virtual void SyncPlayerWorldGen(gplayer\_imp\* pPlayer) const {}

virtual bool GetTownPosition(gplayer\_imp \*pImp, const A3DVECTOR &opos, A3DVECTOR &pos, int & tag) const { return false; }

virtual bool SetIncomingPlayerPos(gplayer \* pPlayer, const A3DVECTOR & origin\_pos) const { return false; }

public:

bool Init(const rect & region, float piece\_width, float piece\_height)

{

//不支持区域中心不为(0,0)的情况

float centerx = (region.left + region.right)\*0.5f;

float centerz = (region.top + region.bottom)\*0.5f;

if(centerx < -1e-6 || centerx > 1e-6 || centerz < -1e-6 || centerz > 1e-6) return false;

\_row = (int)(region.Height()/piece\_height + 0.9f);

\_col = (int)(region.Width()/piece\_width + 0.9f);

\_piece\_width = piece\_width;

\_piece\_height = piece\_height;

\_map\_left = -\_piece\_width \* \_col \* 0.5f;

\_map\_top = \_piece\_height \* \_row \* 0.5f;

ASSERT(\_row > 0 && \_col > 0);

return true;

}

int GetBlockID(float x, float z) const

{

int u = int((x - \_map\_left)/\_piece\_width);

int v = int((\_map\_top - z)/\_piece\_height);

return v\*\_col + u + 1;

}// block 0 for origin

virtual int GetRoomIndex(float x, float z) const { return 0; }

public:

static A3DVECTOR CalcCenterOffset(int col,int row,int maxcol,int maxrow, float piece\_width, float piece\_height)

{

// cur block center (pw\*col+ pw\*/2, ph\*row+ ph\*/2)

// origin center (pw\*maxcol/2, ph\*maxrow/2)

return A3DVECTOR(piece\_width\*0.5f\*(2\*col+1-maxcol),

0,

-piece\_height\*0.5f\*(2\*row+1-maxrow)); // negative Y coordinate

}

};

class random\_map\_generator : public map\_generator

{

public:

virtual bool Generate(const rect & region, const map\_res& mapres);

};

class sequence\_map\_generator : public map\_generator

{

public:

virtual bool Generate(const rect & region, const map\_res& mapres);

virtual void SyncPlayerWorldGen(gplayer\_imp\* pPlayer) const;

};

/\*

\* 初始进入坐标 SetIncomingPlayerPos 由 map\_desp.sev 给出

\* 回城坐标 GetTownPosition 1处于起点块返回GetLogoutPos 2处于其他块返回SetIncomingPlayerPos坐标

\*/

class maze\_map\_generator : public map\_generator

{

public:

virtual bool Generate(const rect & region, const map\_res& mapres);

virtual void SyncPlayerWorldGen(gplayer\_imp\* pPlayer) const;

virtual bool SetIncomingPlayerPos(gplayer \* pPlayer, const A3DVECTOR & origin\_pos) const;

virtual bool GetTownPosition(gplayer\_imp \*pImp, const A3DVECTOR &opos, A3DVECTOR &pos, int & tag) const;

virtual int GetRoomIndex(float x, float z) const

{

int block = GetBlockID(x,z);

return (block > \_col\*\_row || block < 1) ? 0 : \_room\_indexes[block-1];

}

protected:

bool \_GenMaze(const map\_res& mapres);

int \_GetAppropriatePiece(const abase::vector<map\_piece\_desp> & piece\_desps,int joint\_mask,int type) const;

protected:

A3DVECTOR \_birth\_pos; // 出生点

abase::vector<int> \_room\_indexes; // 每一块对应主线的序号1开始，非主线为0

};

class MapResManager

{

int \_mapres\_type;

map\_res \_mapres\_info;

abase::vector<CTerrain \*> \_terrain\_pieces;

abase::vector<NPCMoveMap::CMap \*> \_movemap\_pieces;

abase::vector<trace\_manager2 \*> \_traceman\_pieces;

struct

{

CNPCGenMan\* main\_data;

abase::vector<CNPCGenMan \*> spawn\_pieces;

}\_npcgen\_info;

public:

MapResManager():\_mapres\_type(-1)

{

\_npcgen\_info.main\_data = NULL;

}

~MapResManager();

/\*

\* @param servername e.g. "is05"

\* @param base\_path e.g. "/home/game/game/config/a05/"

\* @param region, local region of the grid

\*/

int Init(std::string servername, std::string base\_path, const rect & region, world \* plane);

int GetType() const { return \_mapres\_type; }

void SetType(int t);

const map\_res& GetMapResInfo() const { return \_mapres\_info; }

CTerrain \* GetUniqueTerrain(){ ASSERT(\_mapres\_type == MAPRES\_TYPE\_ORIGIN); return \_terrain\_pieces[0]; }

NPCMoveMap::CMap \* GetUniqueMoveMap(){ ASSERT(\_mapres\_type == MAPRES\_TYPE\_ORIGIN); return \_movemap\_pieces[0]; }

trace\_manager2 \* GetUniqueTraceMan(){ ASSERT(\_mapres\_type == MAPRES\_TYPE\_ORIGIN); return \_traceman\_pieces[0]; }

CTerrain \* CreateTerrain(map\_generator \* pGenerator);

NPCMoveMap::CMap \* CreateMoveMap(map\_generator \* pGenerator, world \* plane);

trace\_manager2 \* CreateTraceMan(map\_generator \* pGenerator);

bool BuildNpcGenerator(world\* pWorld);

};

int MapResManager::Init(std::string servername, std::string base\_path, const rect & region, world \* plane)

{

ONET::Conf \*conf = ONET::Conf::GetInstance();

std::string section = "Terrain\_";

section += servername;

std::string str = conf->find(section, "mapres\_type");

if(str == "random")

\_mapres\_type = MAPRES\_TYPE\_RANDOM;

else if(str == "maze")

\_mapres\_type = MAPRES\_TYPE\_MAZE;

else

\_mapres\_type = MAPRES\_TYPE\_ORIGIN;

\_\_PRINTINFO("加载地图资源, servername:%s base\_path:%s mapres\_type:%d\n", servername.c\_str(), base\_path.c\_str(), \_mapres\_type);

if(\_mapres\_type == MAPRES\_TYPE\_ORIGIN)

{

//加载地形

TERRAINCONFIG config;

config.nAreaWidth = atoi(conf->find(section,"nAreaWidth").c\_str());

config.nAreaHeight = atoi(conf->find(section,"nAreaHeight").c\_str());

config.nNumAreas = atoi(conf->find(section,"nNumAreas").c\_str());

config.nNumCols = atoi(conf->find(section,"nNumCols").c\_str());

config.nNumRows =atoi(conf->find(section,"nNumRows").c\_str());

config.vGridSize = atof(conf->find(section,"vGridSize").c\_str());

config.vHeightMin = atof(conf->find(section,"vHeightMin").c\_str());

config.vHeightMax = atof(conf->find(section,"vHeightMax").c\_str());

memset(config.szMapPath,0,sizeof(config.szMapPath));

std::string path = base\_path + conf->find(section,"szMapPath");

strncpy(config.szMapPath, path.c\_str(),sizeof(config.szMapPath) - 1);

CTerrain \* pTerrain = new CTerrain();

ASSERT(region.left <= region.right && region.top <= region.bottom);

if(!pTerrain->Init(config,region.left,region.top,region.right,region.bottom))

{

\_\_PRINTINFO("无法初始化地形数据\n");

return -101;

}

\_terrain\_pieces.push\_back(pTerrain);

//加载寻路

std::string str1 = base\_path + conf->find("MoveMap","Path");

std::string str2 = base\_path + conf->find("MoveMap","WaterPath");

std::string str3 = base\_path + conf->find("MoveMap","AirPath");

NPCMoveMap::CMap \* movemap = path\_finding::InitMoveMap(str1.c\_str(),str2.c\_str(),str3.c\_str(),plane);

if(!movemap)

{

\_\_PRINTF("无法读入NPC通路图或者无法读入水域图n");

return -102;

}

\_movemap\_pieces.push\_back(movemap);

//加载碰撞

std::string trace\_path = base\_path + conf->find("Template","CollisionFile");

float map\_width = config.nAreaWidth \* config.vGridSize \* config.nNumCols;

float map\_height = config.nAreaHeight \* config.vGridSize \* config.nNumRows;

trace\_manager2 \* pTraceMan = new trace\_manager2();

if(!pTraceMan->Load(map\_width, map\_height, trace\_path.c\_str()))

{

printf("加载凸包数据'%s'失败\n",trace\_path.c\_str());

}

else

{

printf("加载'%s'完成\n",trace\_path.c\_str());

}

\_traceman\_pieces.push\_back(pTraceMan);

//加载NPCGen

std::string npcgenfile = base\_path + conf->find("Template","NPCGenFile");

\_npcgen\_info.main\_data = new CNPCGenMan();

if(!\_npcgen\_info.main\_data->Load(npcgenfile.c\_str()))

{

\_\_PRINTINFO("无法打开 主npc分布文件\n");

return -103;

}

\_\_PRINTINFO("主地图一共有%d个怪物区域\n", \_npcgen\_info.main\_data->GetGenAreaNum());

}

else

{

int piece\_num = atoi(conf->find(section,"nPiece").c\_str());

TERRAINCONFIG config;

config.nAreaWidth = atoi(conf->find(section,"nAreaWidth").c\_str());

config.nAreaHeight = atoi(conf->find(section,"nAreaHeight").c\_str());

config.nNumAreas = 1;

config.nNumCols = 1;

config.nNumRows = 1;

config.vGridSize = atof(conf->find(section,"vGridSize").c\_str());

config.vHeightMin = atof(conf->find(section,"vHeightMin").c\_str());

config.vHeightMax = atof(conf->find(section,"vHeightMax").c\_str());

memset(config.szMapPath,0,sizeof(config.szMapPath));

std::string path = base\_path + conf->find(section,"szMapPath");

strncpy(config.szMapPath, path.c\_str(),sizeof(config.szMapPath) - 1);

//加载地图片描述信息

\_mapres\_info.width = config.nAreaWidth \* config.vGridSize;

\_mapres\_info.height = config.nAreaHeight \* config.vGridSize;

//加载随机生成条件

CRandMapProp randmap\_prop;

std::string desp\_file = base\_path + conf->find("Template","MapDespFile");

randmap\_prop.LoadSev(desp\_file.c\_str());

ASSERT(piece\_num==randmap\_prop.GetGridCount()&&"gs.conf和map\_desp\_sev描述地图大小要一致");

const CRandMapProp::FILEHEADER& mz\_info = randmap\_prop.GetHeader();

\_mapres\_info.bpos\_offset = A3DVECTOR(mz\_info.fPosX,mz\_info.fPosY,mz\_info.fPosZ);

\_mapres\_info.\_maze\_info.start\_idx = -1;

\_mapres\_info.\_maze\_info.start\_dir = -1;

\_mapres\_info.\_maze\_info.end\_idx = -1;

\_mapres\_info.\_maze\_info.end\_dir = -1;

\_mapres\_info.\_maze\_info.step\_min = mz\_info.mainLineMinLen;

\_mapres\_info.\_maze\_info.step\_max = mz\_info.mainLineMaxLen;

\_mapres\_info.\_maze\_info.branch\_min = mz\_info.branchLineNumMin;

\_mapres\_info.\_maze\_info.branch\_max = mz\_info.branchLineNumMax;

\_mapres\_info.\_maze\_info.branch\_block\_min = mz\_info.branchLineMinLen;

\_mapres\_info.\_maze\_info.branch\_block\_max = mz\_info.branchLineMaxLen;

\_mapres\_info.\_maze\_info.empty\_piece = piece\_num - 1; // 默认规则最后一个房间为空

for(int i=0; i<piece\_num; i++)

{

map\_piece\_desp desp;

//load file ...

CRandMapProp::MAP\_INFO info;

randmap\_prop.GetGirdProp(i,info);

desp.type = info.type;

desp.joint\_mask = info.connection;

\_mapres\_info.\_piece\_desps.push\_back(desp);

}

//加载地图片地形

for(int i=0; i<piece\_num; i++)

{

CTerrain \* pTerrain = new CTerrain();

if(!pTerrain->InitPiece(config,i))

{

\_\_PRINTINFO("无法初始化地形数据\n");

return -201;

}

\_terrain\_pieces.push\_back(pTerrain);

}

//加载地图片寻路

std::string str1 = base\_path + conf->find("MoveMap","Path");

std::string str2 = base\_path + conf->find("MoveMap","WaterPath");

std::string str3 = base\_path + conf->find("MoveMap","AirPath");

for(int i=0; i<piece\_num; i++)

{

NPCMoveMap::CMap \* movemap = path\_finding::InitMoveMapPiece(str1.c\_str(),str2.c\_str(),str3.c\_str(),plane,i);

if(!movemap)

{

\_\_PRINTF("无法读入NPC通路图或者无法读入水域图or天空图\n");

return -202;

}

\_movemap\_pieces.push\_back(movemap);

}

//加载地图片碰撞

for(int i=0; i<piece\_num; i++)

{

char trace\_path[256];

sprintf(trace\_path, "%smapbht/%d.bht", base\_path.c\_str(), i+1);

float piece\_width = config.nAreaWidth \* config.vGridSize;

float piece\_height = config.nAreaHeight \* config.vGridSize;

trace\_manager2 \* pTraceMan = new trace\_manager2();

if(!pTraceMan->LoadPiece(piece\_width, piece\_height, trace\_path))

{

printf("加载凸包数据'%s'失败\n",trace\_path);

}

else

{

printf("加载'%s'完成\n",trace\_path);

}

\_traceman\_pieces.push\_back(pTraceMan);

}

//加载NPCGen

std::string npcgenfile = base\_path + conf->find("Template","NPCGenFile");

\_npcgen\_info.main\_data = new CNPCGenMan();

if(!\_npcgen\_info.main\_data->Load(npcgenfile.c\_str()))

{

\_\_PRINTINFO("无法打开 主npc分布文件\n");

return -203;

}

\_\_PRINTINFO("主地图一共有%d个怪物区域\n", \_npcgen\_info.main\_data->GetGenAreaNum());

for(int i=0; i<piece\_num; i++)

{

char szSpawnFile[256];

sprintf(szSpawnFile, "%snpcgen/npcgen\_%d.data",base\_path.c\_str(),i+1);

CNPCGenMan\* spawndata = new CNPCGenMan();

if(!spawndata->Load(szSpawnFile))

{

\_\_PRINTINFO("无法打开 分支地图%dnpc分布文件\n",i);

return -203;

}

\_\_PRINTINFO("分支地图%d一共有%d个怪物区域\n", i,spawndata->GetGenAreaNum());

\_npcgen\_info.spawn\_pieces.push\_back(spawndata);

}

}

return 0;

}

CTerrain \* MapResManager::CreateTerrain(map\_generator \* pGenerator)

{

CTerrain \* pTerrain = new CTerrain();

if(!pTerrain->Init(pGenerator->GetRow(), pGenerator->GetCol(), pGenerator->GetPieceIndexes().begin(), \_terrain\_pieces.begin()))

{

delete pTerrain;

return NULL;

}

return pTerrain;

}

NPCMoveMap::CMap \* MapResManager::CreateMoveMap(map\_generator \* pGenerator, world \* plane)

{

NPCMoveMap::CMap \* movemap = path\_finding::InitMoveMap(plane, pGenerator->GetRow(), pGenerator->GetCol(), pGenerator->GetPieceIndexes().begin(), \_movemap\_pieces.size(), \_movemap\_pieces.begin());

return movemap;

}

trace\_manager2 \* MapResManager::CreateTraceMan(map\_generator \* pGenerator)

{

trace\_manager2 \* pTraceMan = new trace\_manager2;

pTraceMan->Load(pGenerator->GetRow(), pGenerator->GetCol(), pGenerator->GetPieceIndexes().begin(), \_traceman\_pieces.begin(), false);

return pTraceMan;

}

bool MapResManager::BuildNpcGenerator(world\* pWorld)

{

const map\_generator \* pGenerator = pWorld->GetMapGen();

if(\_mapres\_type == MAPRES\_TYPE\_ORIGIN)

{

return pWorld->InitNPCGenerator(\*\_npcgen\_info.main\_data);

}

else

{

const abase::vector<int>& piece\_indexes = pGenerator->GetPieceIndexes();

npcgen\_data\_list spawn\_list;

spawn\_list.reserve(piece\_indexes.size());

npcgen\_data\_node\_t spawn\_data;

int row = pGenerator->GetRow();

int col = pGenerator->GetCol();

for(int v = 0; v < row; ++v)

{

for(int u = 0; u < col; ++u)

{

int idx = v \* col + u;

int destidx = piece\_indexes[idx];

if((size\_t)destidx >= \_npcgen\_info.spawn\_pieces.size())

return false;

spawn\_data.npcgen = \_npcgen\_info.spawn\_pieces[destidx];

spawn\_data.blockid = idx + 1;

spawn\_data.offset = map\_generator::CalcCenterOffset(u,v,col,row,\_mapres\_info.width,\_mapres\_info.height);

spawn\_list.push\_back(spawn\_data);

}

}

return pWorld->InitNPCGenerator(\*\_npcgen\_info.main\_data, spawn\_list);

}

return true;

}

//////////////////////////////////////////////////////////////

bool random\_map\_generator::Generate(const rect & region, const map\_res& mapres)

{

if(!Init(region, mapres.width, mapres.height)) return false;

size\_t piece\_num = mapres.\_piece\_desps.size();

ASSERT(piece\_num);

\_piece\_indexes.insert(\_piece\_indexes.end(), \_col\*\_row, -1);

for(size\_t i=0; i<\_piece\_indexes.size(); i++)

{

\_piece\_indexes[i] = abase::Rand(0, piece\_num-1);

}

return true;

}

//////////////////////////////////////////////////////////////

bool sequence\_map\_generator::Generate(const rect & region, const map\_res& mapres)

{

if(!Init(region, mapres.width, mapres.height)) return false;

size\_t piece\_num = mapres.\_piece\_desps.size();

ASSERT(piece\_num == (size\_t)(\_col\*\_row) && "地图资源和生成大小不匹配");

\_piece\_indexes.insert(\_piece\_indexes.end(), piece\_num, -1);

for(size\_t i=0; i< piece\_num; i++)

{

\_piece\_indexes[i] = i;

}

return true;

}

void sequence\_map\_generator::SyncPlayerWorldGen(gplayer\_imp\* pPlayer) const

{

gplayer\_dispatcher\* runner = (gplayer\_dispatcher \*)pPlayer->\_runner;

runner->randommap\_order\_init(\_row,\_col,\_piece\_indexes.begin());

}

//////////////////////////////////////////////////////////////

bool maze\_map\_generator::Generate(const rect & region, const map\_res& mapres)

{

if(!Init(region, mapres.width, mapres.height)) return false;

if(!\_GenMaze(mapres)) return false;

return true;

}

bool maze\_map\_generator::\_GenMaze(const map\_res& mapres)

{

struct timeval tv\_beg,tv\_end;

gettimeofday(&tv\_beg,NULL);

RandomMaze::MazeGen generator;

const maze\_info& minfo = mapres.\_maze\_info;

int ret = generator.Init(\_col,\_row - 1, // 最后一行为固定的隐藏房间

minfo.start\_idx, (RandomMaze::Direction)minfo.start\_dir,

minfo.end\_idx, (RandomMaze::Direction)minfo.end\_dir,

minfo.step\_min, minfo.step\_max,

minfo.branch\_min, minfo.branch\_max,

minfo.branch\_block\_min, minfo.branch\_block\_max,

minfo.ring\_min, minfo.ring\_max);

if(ret != 0)

{

\_\_PRINTINFO("随机迷宫生成参数错误retcode=%d(%s)\n",

ret,RandomMaze::MazeGen::TranslateErrorCode(ret).c\_str());

return false;

}

//生成的迷宫不一定严格满足所有指定参数

ret = generator.Generate(true);

gettimeofday(&tv\_end,NULL);

int costtime = 1000000\*(tv\_end.tv\_sec - tv\_beg.tv\_sec) + tv\_end.tv\_usec - tv\_beg.tv\_usec;

if (ret != 0 )

{

\_\_PRINTINFO("随机迷宫生成警告retcode=%d(%s)\n",

ret,RandomMaze::MazeGen::TranslateErrorCode(ret).c\_str());

}

const RandomMaze::Maze& maze = generator.GetMaze();

\_\_PRINTINFO("生成%dx%d 分支数%d 分支房间数%d 环数%d 主路径总步数%d 的随机迷宫耗时%d 微秒\n",

maze.GetHorizRoomCount(),maze.GetVertRoomCount(),

maze.GetBranchCount(),maze.GetBranchRoomCount(),

maze.GetRingCount(),maze.GetStepCount(),costtime);

\_piece\_indexes.insert(\_piece\_indexes.end(), \_col\*\_row, minfo.empty\_piece);

\_room\_indexes.insert(\_room\_indexes.end(), \_col\*\_row, 0);

\_birth\_pos = A3DVECTOR(0,0,0);

std::vector<int>::const\_iterator citer = generator.GetMainPath().begin();

std::vector<int>::const\_iterator ciend = generator.GetMainPath().end();

for(int i = 1; citer != ciend; ++citer, ++i)

{

\_room\_indexes[\*citer] = i;

}

int total\_room = \_col\*(\_row-1);

for(int index = 0; index < total\_room; ++index)

{

const RandomMaze::Room& room = maze.GetRoom(index);

int roommask = room.GetDoorMask();

if(!roommask) continue;

int roomtype = CRandMapProp::GRID\_TYPE\_NORMAL;

if(index == maze.GetEntranceRoomNo())

{

roomtype = CRandMapProp::GRID\_TYPE\_START;

\_birth\_pos = CalcCenterOffset(index%\_col, index/\_col, \_col, \_row, mapres.width, mapres.height);

\_birth\_pos += mapres.bpos\_offset;

}

else if(index == maze.GetExitRoomNo())

roomtype = CRandMapProp::GRID\_TYPE\_END;

\_piece\_indexes[index] = \_GetAppropriatePiece(mapres.\_piece\_desps,roommask,roomtype);

if(\_piece\_indexes[index] < 0)

{

GLog::log(GLOG\_ERR,"随机地图未找到匹配房间%d 联通条件%d 类型%d",index,roommask,roomtype);

\_piece\_indexes[index] = minfo.empty\_piece;

}

}

generator.ShowRoom();

// 资源固定的房间

ASSERT(\_col\*\_row > 4 && "隐藏规则房间需要符合个数");

for(int index = \_col\*\_row - 1; index > \_col\*\_row - 5; --index)

\_piece\_indexes[index] = index;

// 输出结果

for(int v = 0; v < \_row; ++v)

{

for(int u = 0; u < \_col; ++u)

{

printf("[%02d]",\_piece\_indexes[u+v\*\_col]);

}

printf("\n");

}

return true;

}

int maze\_map\_generator::\_GetAppropriatePiece(const abase::vector<map\_piece\_desp> & piece\_desps,int joint\_mask, int type) const

{

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

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

for(int i = 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);

}

}

int iNum;

if (equal\_joint\_pieces.size())

{

int rand\_idx = 0;

iNum = (int)equal\_joint\_pieces.size();

if(iNum > 1)

rand\_idx = abase::Rand(rand\_idx,iNum-1);

return equal\_joint\_pieces[rand\_idx];

}

if (close\_joint\_pieces.size())

{

int rand\_idx = 0;

iNum = (int)close\_joint\_pieces.size();

if(iNum > 1)

rand\_idx = abase::Rand(rand\_idx,iNum-1);

GLog::log(GLOG\_ERR,"警告：随机地图未找到最优联通匹配类型%d 房间%d 条件%d 实际%d",

type, rand\_idx, joint\_mask, piece\_desps[close\_joint\_pieces[rand\_idx]].joint\_mask );

return close\_joint\_pieces[rand\_idx];

}

return -1;

}

void maze\_map\_generator::SyncPlayerWorldGen(gplayer\_imp\* pPlayer) const

{

gplayer\_dispatcher\* runner = (gplayer\_dispatcher \*)pPlayer->\_runner;

runner->randommap\_order\_init(\_row,\_col,\_piece\_indexes.begin());

}

bool maze\_map\_generator::SetIncomingPlayerPos(gplayer \* pPlayer, const A3DVECTOR & origin\_pos) const

{

pPlayer->pos = \_birth\_pos;

return true;

}

bool maze\_map\_generator::GetTownPosition(gplayer\_imp \*pImp, const A3DVECTOR &opos, A3DVECTOR &pos, int & tag) const

{

if(!pImp->\_parent->IsZombie() && GetRoomIndex(opos.x,opos.z) == 1) // 处于起点块 非死亡状态

{

world\_manager::GetInstance()->GetLogoutPos(pImp,tag,pos);

}

else if(GetBlockID(opos.x,opos.z) > 60) // 隐藏房间配置随precinct.sev

{

return false;

}

else

{

pos = \_birth\_pos;

tag = world\_manager::GetWorldTag();

}

return true;

}

class trace\_manager2

{

struct brush\_data

{

SvrCD::CBrushMan \* brush\_man;

SvrCD::CSMTree \* sm\_tree;

int sm\_ref;

brush\_data():brush\_man(NULL), sm\_tree(NULL), sm\_ref(0){}

};

bool \_valid;

float \_submap\_width;

float \_submap\_height;

float \_submap\_width\_inv;

float \_submap\_height\_inv;

int \_row;

int \_col;

float \_map\_left;

float \_map\_top;

int \_element\_count;

abase::vector<brush\_data> \_submap\_brushes;

static SvrCD::CNmdChd \*\_nmd\_element;

inline int GetBlockIndex(const A3DVECTOR & pos)

{

if(\_submap\_brushes.size() == 1) return 0;

int u = (int)((pos.x - \_map\_left) \* \_submap\_width\_inv);

int v = (int)((\_map\_top - pos.z) \* \_submap\_height\_inv);

if(u < 0) u = 0;

if(u > \_col - 1) u = \_col - 1;

if(v < 0) v = 0;

if(v > \_row - 1) v = \_row - 1;

return v \* \_col + u;

}

public:

trace\_manager2();

~trace\_manager2();

bool Load(float width, float height, const char \* filename);

bool LoadPiece(float width, float height, const char \* filename);

bool Load(int row, int col, const int \* piece\_indexes, trace\_manager2 \*\* trace\_pieces, bool enable\_nmdtree = false);

bool Valid() const { return \_valid;}

bool AABBTrace(const A3DVECTOR & start, const A3DVECTOR & offset, const A3DVECTOR & ext, bool & in\_solid, float &ratio, const abase::vector<char>\* element\_flags=NULL);

static bool LoadElement(const char \* filename); //装载单独文件

static void ReleaseElement();

int RegisterElement(int tid, int mid, const A3DVECTOR & pos, float dir0, float dir, float up);

void EnableElement(int cid, bool active, abase::vector<char>\* element\_flags=NULL);

int GetElementCount(){ return \_element\_count; }

};

SvrCD::CNmdChd \*trace\_manager2::\_nmd\_element = NULL;

trace\_manager2::trace\_manager2():\_valid(false),\_submap\_width(0.f),\_submap\_height(0.f),\_submap\_width\_inv(0.f),\_submap\_height\_inv(0.f),\_row(0),\_col(0),\_map\_left(0.f),\_map\_top(0.f),\_element\_count(0)

{

}

trace\_manager2::~trace\_manager2()

{

for(size\_t i=0; i<\_submap\_brushes.size(); i++)

{

brush\_data & brush = \_submap\_brushes[i];

if(brush.brush\_man)

{

brush.brush\_man->Release();

delete brush.brush\_man;

}

if(!brush.sm\_ref && brush.sm\_tree)

{

brush.sm\_tree->Release();

delete brush.sm\_tree;

}

}

\_submap\_brushes.clear();

}

bool trace\_manager2::Load(float width, float height, const char \* filename)

{

SvrCD::CSMTree \* sm\_tree = new SvrCD::CSMTree();

if(!sm\_tree->Load(filename))

{

delete sm\_tree;

return false;

}

\_submap\_width = width;

\_submap\_height = height;

\_submap\_width\_inv = 1 / width;

\_submap\_height\_inv = 1 / height;

\_row = 1;

\_col = 1;

\_map\_left = -0.5f \* width;

\_map\_top = 0.5f \* height;

brush\_data brush;

brush.sm\_tree = sm\_tree;

brush.sm\_ref = 0;

brush.brush\_man = new SvrCD::CBrushMan();

brush.brush\_man->SetSmTree(brush.sm\_tree, true);

\_submap\_brushes.push\_back(brush);

\_valid = true;

return true;

}

bool trace\_manager2::LoadPiece(float width, float height, const char \* filename)

{

SvrCD::CSMTree \* sm\_tree = new SvrCD::CSMTree();

if(!sm\_tree->Load(filename))

{

delete sm\_tree;

return false;

}

\_submap\_width = width;

\_submap\_height = height;

\_submap\_width\_inv = 1 / width;

\_submap\_height\_inv = 1 / height;

\_row = 1;

\_col = 1;

\_map\_left = -0.5f \* width;

\_map\_top = 0.5f \* height;

brush\_data brush;

brush.sm\_tree = sm\_tree;

brush.sm\_ref = 0;

brush.brush\_man = new SvrCD::CBrushMan();

brush.brush\_man->SetSmTree(brush.sm\_tree, false);

\_submap\_brushes.push\_back(brush);

\_valid = true;

return true;

}

bool trace\_manager2::Load(int row, int col, const int \* piece\_indexes, trace\_manager2 \*\* trace\_pieces, bool enable\_nmdtree)

{

\_submap\_width = trace\_pieces[0]->\_submap\_width;

\_submap\_height = trace\_pieces[0]->\_submap\_height;

\_submap\_width\_inv = trace\_pieces[0]->\_submap\_width\_inv;

\_submap\_height\_inv = trace\_pieces[0]->\_submap\_height\_inv;

\_row = row;

\_col = col;

\_map\_left = -0.5f \* col \* \_submap\_width;

\_map\_top = 0.5f \* row \* \_submap\_height;

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

{

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

{

brush\_data brush;

brush.sm\_tree = trace\_pieces[piece\_indexes[i \* col + j]]->\_submap\_brushes[0].sm\_tree;

brush.sm\_ref = 1;

brush.brush\_man = new SvrCD::CBrushMan();

brush.brush\_man->SetSmTree(brush.sm\_tree, enable\_nmdtree);

\_submap\_brushes.push\_back(brush);

}

}

\_valid = true;

return true;

}

struct InterSection

{

struct Pos2D{

float u;

float v;

bool uaxis; // 相交点是否处于u轴上 ，否则则为v轴上

bool uvc; // 是否处在uv轴心上, 0.001f范围内

};

A3DVECTOR start; // 以左上为0,0点的坐标系中的起始点

A3DVECTOR offset;

float width;

float height;

int max\_index;

int max\_col;

int max\_row;

std::map<float,Pos2D> cross\_point\_set;

enum EIS\_DIR{

ISD\_UP,

ISD\_UP\_RIGHT,

ISD\_RIGHT,

ISD\_DOWN\_RIGHT,

ISD\_DOWN,

ISD\_DOWN\_LEFT,

ISD\_LEFT,

ISD\_UP\_LEFT

};

int GetNearIndex(int col,int row,EIS\_DIR dir)

{

if(col > max\_col || row > max\_row || col < 0 || row < 0) return -1;

switch(dir)

{

case ISD\_UP:

case ISD\_UP\_RIGHT:

return row && col < max\_col ? (row -1) \* max\_col + col : -1;

case ISD\_RIGHT:

case ISD\_DOWN\_RIGHT:

case ISD\_DOWN:

return col < max\_col && row < max\_row ? row \* max\_col + col : -1;

case ISD\_DOWN\_LEFT:

case ISD\_LEFT:

return row < max\_row && col ? row \* max\_col + col - 1 : -1;

case ISD\_UP\_LEFT:

return row && col ? (row -1) \* max\_col + col -1 : -1;

default:

return -1;

}

}

InterSection(const A3DVECTOR & sta, const A3DVECTOR & off,float wdt, float hgt)

: start(sta), offset(off), width(wdt), height(hgt)

{

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

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

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

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

float k = offset.z/offset.x;

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

bool negative\_u = offset.x < 0;

bool negative\_v = offset.z < 0;

int step\_u = negative\_u ? -1 : 1;

int step\_v = negative\_v ? -1 : 1;

Pos2D cross\_point;

Pos2D cross\_point\_modify;

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

cross\_point.uaxis = true;

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;

}

}

void PushUnique(std::vector<int>& sids,int index)

{

if(index < 0 || index >= max\_index) return;

if(sids.end() == std::find(sids.begin(),sids.end(),index))

sids.push\_back(index);

}

void GetInterSectionId(std::vector<int>& sids,int mxc,int mxr)

{

max\_col = mxc;

max\_row = mxr;

max\_index = max\_col \* max\_row;

std::map<float,Pos2D>::iterator iter = cross\_point\_set.begin();

std::map<float,Pos2D>::iterator iend = cross\_point\_set.end();

#define INSERT\_SECTION\_ID(dir1,dir2) { PushUnique(sids,GetNearIndex(col,row,dir1)); PushUnique(sids,GetNearIndex(col,row,dir2)); }

for(; iter != iend; ++iter)

{

Pos2D& pos = iter->second;

int col = (int)(pos.u/width);

int row = (int)(pos.v/-height);

if(pos.uvc) // 交点过uv轴心

{

if(offset.x > 0 && offset.z > 0) // (0,90)

INSERT\_SECTION\_ID(ISD\_DOWN\_LEFT,ISD\_UP\_RIGHT)

else if(offset.x < 0 && offset.z > 0)// ( 90, 180)

INSERT\_SECTION\_ID(ISD\_DOWN\_RIGHT,ISD\_UP\_LEFT)

else if(offset.x < 0 && offset.z < 0) // (180,270)

INSERT\_SECTION\_ID(ISD\_UP\_RIGHT,ISD\_DOWN\_LEFT)

else if(offset.x > 0 && offset.z < 0) // (270,360)

INSERT\_SECTION\_ID(ISD\_UP\_LEFT,ISD\_DOWN\_RIGHT)

else if(offset.x == 0 && offset.z > 0) // 90

INSERT\_SECTION\_ID(ISD\_DOWN,ISD\_UP)

else if(offset.x == 0 && offset.z < 0) // 270

INSERT\_SECTION\_ID(ISD\_UP,ISD\_DOWN)

else if(offset.z == 0 && offset.x > 0) // 0

INSERT\_SECTION\_ID(ISD\_LEFT,ISD\_RIGHT)

else if(offset.z == 0 && offset.x < 0) // 180

INSERT\_SECTION\_ID(ISD\_RIGHT,ISD\_LEFT)

// else

//ASSERT(false);

}

else if(pos.uaxis)

{

if(offset.z > 0)

INSERT\_SECTION\_ID(ISD\_DOWN,ISD\_UP)

else

INSERT\_SECTION\_ID(ISD\_UP,ISD\_DOWN)

}

else

{

if(offset.x > 0)

INSERT\_SECTION\_ID(ISD\_LEFT,ISD\_RIGHT)

else

INSERT\_SECTION\_ID(ISD\_RIGHT,ISD\_LEFT)

}

}

#undef INSERT\_SECTION\_ID

}

};

A3DVECTOR GetBlockCenter(int col,int row,int maxcol,int maxrow, float piece\_width, float piece\_height)

{

// cur block center (pw\*col+ pw\*/2, ph\*row+ ph\*/2)

// origin center (pw\*maxcol/2, ph\*maxrow/2)

return A3DVECTOR(piece\_width\*0.5f\*(2\*col+1-maxcol),

0,

-piece\_height\*0.5f\*(2\*row+1-maxrow)); // negative Y coordinate

}

bool trace\_manager2::AABBTrace(const A3DVECTOR & start, const A3DVECTOR & offset, const A3DVECTOR & ext, bool & in\_solid, float &ratio, const abase::vector<char>\* element\_flags)

{

if(!\_valid) return false;

A3DVECTOR end = start;

end += offset;

int start\_index = GetBlockIndex(start);

int end\_index = GetBlockIndex(end);

if(start\_index == end\_index)

{

A3DVECTOR bc = GetBlockCenter(start\_index%\_col,start\_index/\_col,\_col,\_row, \_submap\_width, \_submap\_height); // 中心点相对偏移

SvrCD::BrushTraceInfo info;

info.Init( A3DVECTOR3(start.x - bc.x,start.y + ext.y ,start.z - bc.z), A3DVECTOR3(offset.x,offset.y,offset.z), A3DVECTOR3(ext.x,ext.y,ext.z));

if(!\_submap\_brushes[start\_index].brush\_man->Trace(&info,element\_flags)) return false;

in\_solid = info.bStartSolid;

ratio = info.fFraction;

return true;

}

else

{

InterSection is(A3DVECTOR(start.x - \_map\_left,0,start.z - \_map\_top),A3DVECTOR(offset.x,0,offset.z),\_submap\_width, \_submap\_height);

std::vector<int> vt; // 按方向排序后的经过序号集合

is.GetInterSectionId(vt,\_col,\_row);

for(size\_t n = 0; n < vt.size(); ++n)

{

int index = vt[n];

ASSERT(index < \_col\*\_row);

A3DVECTOR bc = GetBlockCenter(index%\_col,index/\_col,\_col,\_row, \_submap\_width, \_submap\_height); // 中心点相对偏移

SvrCD::BrushTraceInfo info;

info.Init( A3DVECTOR3(start.x - bc.x,start.y + ext.y ,start.z - bc.z), A3DVECTOR3(offset.x,offset.y,offset.z), A3DVECTOR3(ext.x,ext.y,ext.z));

if(!\_submap\_brushes[index].brush\_man->Trace(&info,element\_flags)) continue;

in\_solid = info.bStartSolid;

ratio = info.fFraction;

return true;

}

return false;

}

}

bool trace\_manager2::LoadElement(const char \* filename)

{

if(\_nmd\_element) return false;

\_nmd\_element = new SvrCD::CNmdChd();

if(!\_nmd\_element->Load(filename))

{

delete \_nmd\_element;

\_nmd\_element = NULL;

return false;

}

return true;

}

void trace\_manager2::ReleaseElement()

{

if(!\_nmd\_element) return;

\_nmd\_element->Release();

delete \_nmd\_element;

\_nmd\_element = NULL;

}

int trace\_manager2::RegisterElement(int tid, int mid, const A3DVECTOR & pos, float dir0, float dir1, float rad)

{

if(!\_valid) return 0;

int index = GetBlockIndex(pos);

//判断是否支持动态碰撞

if(\_submap\_brushes[index].brush\_man->GetNmdTree() == NULL) return 0;

A3DVECTOR bc = GetBlockCenter(index%\_col, index/\_col, \_col, \_row, \_submap\_width, \_submap\_height);

A3DVECTOR3 target(pos.x - bc.x, pos.y, pos.z - bc.z);

A3DVECTOR3 dir,up;

if(dir1 <1e-3 && rad <1e-3)

{

dir.x = (float)cos(dir0);

dir.y = 0;

dir.z = (float)sin(dir0);

up.x = 0;

up.y = 1;

up.z = 0;

}

else

{

//计算Dir和UP

A3DVECTOR3 vAxis;

float p = (float)sin(dir1);

vAxis.x = p \* (float)cos(dir0);

vAxis.z = p \* (float)sin(dir0);

vAxis.y = (float)cos(dir1);

A3DQUATERNION q(vAxis, rad \*256.0f/255.0f);

A3DMATRIX4 matTran;

q.ConvertToMatrix(matTran);

dir = matTran.GetRow(2);

up = matTran.GetRow(1);

}

if(tid)

{

\_submap\_brushes[index].brush\_man->AddNpcMine(\_nmd\_element,tid, \_element\_count + 1, target, up, dir);

\_element\_count ++;

}

else if(mid)

{

\_submap\_brushes[index].brush\_man->AddDynObj(\_nmd\_element,mid, \_element\_count + 1, target, up, dir);

\_element\_count ++;

}

else

{

return 0;

}

return \_element\_count;

}

void trace\_manager2::EnableElement(int cid, bool active, abase::vector<char>\* element\_flags)

{

if(!\_valid) return;

if(element\_flags)

(\*element\_flags)[cid] = active?1:0;

else

{

//效率有点低，可以考虑建立cid->brush idx查询表

for(size\_t i=0; i<\_submap\_brushes.size(); i++)

{

\_submap\_brushes[i].brush\_man->EnableNmd(cid,active);

}

}

}

class npc\_spawner : public base\_spawner

{

public:

struct entry\_t

{

int npc\_tid;

int msg\_mask\_or;

int msg\_mask\_and;

int alarm\_mask; //未用

int enemy\_faction; //未用

bool has\_faction;

int faction; //未用

bool ask\_for\_help;

int monster\_faction\_ask\_help;

bool accept\_ask\_for\_help;

int monster\_faction\_accept\_for\_help;

int reborn\_time\_upper; //重生时间上限

int reborn\_time\_lower; //重生时间下限

int path\_id; //巡逻路线，如果是0则未用

int path\_type; //巡论类型

int corpse\_delay; //尸体残留时间，单位为秒

bool speed\_flag; //速度标记

size\_t mobs\_count;

float offset\_terrain; //和地形高度的偏移量

};

protected:

int \_reborn\_time; //总的重生时间，给group\_spawner使用

abase::vector<entry\_t, abase::fast\_alloc<> > \_entry\_list;//本生成器中的所有entry列表

abase::vector<gnpc \*, abase::fast\_alloc<> > \_npc\_pool; //npc的生成池，所有释放的npc 会被缓冲到这里

int \_mobs\_total\_gen\_num; //能够生成多少对象

int \_mobs\_cur\_gen\_num; //当前生成了多少对象

int \_mob\_life; //创建怪物的寿命

public:

npc\_spawner():\_reborn\_time(0),\_mobs\_total\_gen\_num(0),\_mobs\_cur\_gen\_num(0),\_mob\_life(0){}

virtual ~npc\_spawner() {}

virtual int Init(const void \* buf, size\_t len) { return 0;}

virtual void OnHeartbeat(world \* pPlane) = 0;

virtual bool Reclaim(world \* pPlane, gnpc \* pNPC, gnpc\_imp \* imp,bool is\_reset) = 0;

virtual bool CreateMobs(world \*pPlane) {return false;}

virtual void ForwardFirstAggro(world \* pPlane,const XID & id, int rage) {}

virtual bool CreateObjects(world \*pPlane)

{

\_mobs\_cur\_gen\_num = 0;

return CreateMobs(pPlane);

}

virtual void ClearObjects(world \* pPlane);

static gnpc \* CreateMobBase(npc\_spawner \* \_\_this,world \* pPlane,const entry\_t & et,

int spawn\_index, const A3DVECTOR & pos,const int cid[3],unsigned char dir,

int ai\_policy\_cid,int aggro\_policy, gnpc \* orign\_npc,int life = 0);

static gnpc \* CreatePetBase(gplayer\_imp \*pImp, const pet\_data \* pData, const A3DVECTOR & pos, char inhabit\_mode,

const int cid[3],unsigned char dir, unsigned char pet\_stamp,

int ai\_policy\_cid, int aggro\_policy);

static gnpc \* CreatePetBase2(gplayer\_imp \*pImp, const pet\_data \* pData, const A3DVECTOR & pos, char inhabit\_mode,

const int cid[3],unsigned char dir, unsigned char pet\_stamp,

int ai\_policy\_cid, int aggro\_policy, int skill\_level);

static gnpc \* CreatePetBase3(gplayer\_imp \* pMaster, const pet\_data \* pData, const A3DVECTOR & pos, char inhabit\_mode, const int cid[3],

unsigned char dir, unsigned char pet\_stamp,

int ai\_policy\_cid , int aggro\_policy);

static gnpc \* CreateNPCBase(npc\_spawner \* \_\_this, world \* pPlane, const entry\_t & et,

int spawn\_index, const A3DVECTOR & pos,const int cid[3], unsigned char dir,

int ai\_policy\_cid = -1,int aggro\_policy = 0,gnpc \* origin\_npc = NULL, int life = 0);

static void AdjustPropByCommonValue(gnpc\_imp \* pImp, world \* pPlane, npc\_template \* pTemplate);

void RegenAddon(gnpc\_imp \* pImp, int npc\_id);

void SetGenLimit(int life, int total\_num)

{

\_mob\_life = life;

\_mobs\_total\_gen\_num = total\_num;

}

public:

bool AddEntry(const entry\_t & ent)

{

\_entry\_list.push\_back(ent);

\_npc\_pool.push\_back(NULL);

return true;

}

virtual void BuildRegionCollision(world \* plane, int region\_idx); //固定地图使用

virtual void BuildRegionCollision2(world \* plane); //随机地图使用

int CutRegion(const rect & rt)

{

float oa = \_region.GetArea();

if(oa <= 0.01f)

{

//对于面积非常小的区域

//认为在其中

int total\_count = 0;

for(size\_t i = 0; i < \_entry\_list.size(); i ++)

{

total\_count += \_entry\_list[i].mobs\_count;

}

return total\_count;

}

float na = rt.GetArea();

float factor = na/oa;

int total\_count = 0;

//根据面积比例调整数值

for(size\_t i = 0; i < \_entry\_list.size(); i ++)

{

ASSERT(\_npc\_pool[i] == 0);

int mobs\_count = (int)(\_entry\_list[i].mobs\_count \* factor + 0.5f);

if(mobs\_count <= 0)

{

//应该删除这个条目

\_entry\_list.erase(\_entry\_list.begin() + i);

\_npc\_pool.erase(\_npc\_pool.begin() + i);

--i;

continue;

}

\_entry\_list[i].mobs\_count = mobs\_count;

total\_count += mobs\_count;

}

\_region = rt;

\_pos\_min.x = rt.left; \_pos\_min.z = rt.top;

\_pos\_max.x = rt.right; \_pos\_max.z = rt.bottom;

return total\_count;

}

void SetRebornTime(int rtime)

{

\_reborn\_time = rtime;

}

};

class mine\_spawner : public base\_spawner

{

public:

struct entry\_t

{

int mid;

int mine\_count;

int reborn\_time;

};

protected:

abase::vector<entry\_t,abase::fast\_alloc<> > \_entry\_list;//本生成器中的所有entry列表

abase::vector<gmatter \*,abase::fast\_alloc<> > \_mine\_pool;//npc的生成池，所有释放的npc 会被缓冲到这里

float \_offset\_terrain;

int \_mine\_total\_gen\_num; //能够生成多少对象

int \_mine\_cur\_gen\_num; //当前生成了多少对象

public:

mine\_spawner():\_offset\_terrain(0),\_mine\_total\_gen\_num(0),\_mine\_cur\_gen\_num(0) {}

virtual ~mine\_spawner() {}

virtual void OnHeartbeat(world \* pPlane);

virtual bool Reclaim(world \* pPlane,gmatter \* pMatter, gmatter\_mine\_imp \* imp);

virtual bool CreateMines(world \* pPlane);

static gmatter \* CreateMine(mine\_spawner\* ,const A3DVECTOR & pos, world \* pPlane,int index,const entry\_t & ent);

static void GenerateMineParam(gmatter\_mine\_imp \* imp, npc\_template \* pTemplate);

void Reborn(world \* pPlane,gmatter \* header, gmatter \* tail,int mid,int index);

void SetOffsetTerrain(float offset) { \_offset\_terrain = offset; }

void Release();

void SetGenLimit(int total\_num)

{

\_mine\_total\_gen\_num = total\_num;

}

virtual bool CreateObjects(world \*pPlane)

{

\_mine\_cur\_gen\_num = 0;

return CreateMines(pPlane);

}

virtual void ClearObjects(world \* pPlane);

public:

bool AddEntry(const entry\_t & ent)

{

\_entry\_list.push\_back(ent);

\_mine\_pool.push\_back(NULL);

return true;

}

virtual void BuildRegionCollision(world \* plane, int region\_idx); //固定地图使用

virtual void BuildRegionCollision2(world \* plane); //随机地图使用

int CutRegion(const rect & rt)

{

float oa = \_region.GetArea();

if(oa <= 0.01f)

{

//对于面积非常小的区域

//认为在其中

int total\_count = 0;

for(size\_t i = 0; i < \_entry\_list.size(); i ++)

{

total\_count += \_entry\_list[i].mine\_count;

}

return total\_count;

}

float na = rt.GetArea();

float factor = na/oa;

int total\_count = 0;

//根据面积比例调整数值

for(size\_t i = 0; i < \_entry\_list.size(); i ++)

{

ASSERT(\_mine\_pool[i] == 0);

int mine\_count = (int)(\_entry\_list[i].mine\_count \* factor + 0.5f);

if(mine\_count <= 0)

{

//应该删除这个条目

\_entry\_list.erase(\_entry\_list.begin() + i);

\_mine\_pool.erase(\_mine\_pool.begin() + i);

--i;

continue;

}

\_entry\_list[i].mine\_count = mine\_count;

total\_count += mine\_count;

}

\_region = rt;

\_pos\_min.x = rt.left; \_pos\_min.z = rt.top;

\_pos\_max.x = rt.right; \_pos\_max.z = rt.bottom;

return total\_count;

}

};

namespace RandomMaze

{

enum Direction

{

DirectionInvalid = -1,

DirectionLeft = 0,

DirectionUp = 1,

DirectionRight = 2,

DirectionDown = 3,

DirectionCount

};

class Room

{

public:

static int InvalidRoomNo;

public:

Room(int no);

void SetRoomNo(int value) { m\_roomNo = value; }

int GetRoomNo() const { return m\_roomNo; }

bool CanWalkOut(Direction dir) const { return m\_sides[dir]; }

bool CanWalkOut() const { return m\_sides[0] || m\_sides[1] || m\_sides[2] || m\_sides[3]; }

void SetSide(Direction dir, bool canWalkOut) { m\_sides[dir] = canWalkOut; }

void Enclose() { m\_sides[0] = m\_sides[1] = m\_sides[2] = m\_sides[3] = false; }

int GetDoorCount() const;

int GetWallCount() const;

bool IsBranchEndRoom() const;

bool IsEnclosedRoom() const;

int GetDoors(std::vector<Direction>& doors) const;

int GetWalls(std::vector<Direction>& walls) const;

int GetDoorMask() const

{

int mask = 0;

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

{

if(m\_sides[i])

mask |= (1<<i);

}

return mask;

}

private:

int m\_roomNo;

bool m\_sides[4];

}; // class Room

class MazeGen;

class Maze

{

friend class MazeGen;

public:

Maze(void);

~Maze(void);

Maze(const Maze & data);

Maze& operator = (const Maze & rhs);

bool Init(int horizRoomCount, int vertRoomCount, int entranceRoomNo, int exitRoomNo);

void Clear();

Room& GetRoom(int roomNo) { return m\_rooms[roomNo]; }

const Room& GetRoom(int roomNo) const { return m\_rooms[roomNo]; }

int GetNextRoomNo(int roomNo, Direction dir) const;

bool SetDoor(int room1, int room2);

bool SetWall(int room1, int room2);

bool IsValidRoom(int roomNo) const;

bool IsNeighbor(int room1, int room2) const;

bool IsEntranceRoom(int roomNo) const { return roomNo == m\_entranceRoomNo; }

bool IsExitRoom(int roomNo) const { return roomNo == m\_exitRoomNo; }

void GetRoomCoord(int roomNo, int& x, int& y) const;

int GetRoomNo(int x, int y) const;

int GetManhDistance(int roomNo1, int roomNo2) const;

int GetHorizRoomCount() const { return m\_horizRoomCount; }

int GetVertRoomCount() const { return m\_vertRoomCount; }

int GetTotalRoomCount() const { return m\_rooms.size(); }

int GetEntranceRoomNo() const { return m\_entranceRoomNo; }

int GetExitRoomNo() const { return m\_exitRoomNo; }

int GetStepCount() const { return m\_stepCount; }

int GetBranchCount() const { return m\_branchCount; }

int GetBranchRoomCount() const { return m\_branchRoomCount; }

void SetExitRoomNo(int roomNo) { m\_exitRoomNo = roomNo; }

void SetStepCount(int value) { m\_stepCount = value; }

void SetBranchCount(int value) { m\_branchCount = value; }

void SetBranchRoomCount(int value) { m\_branchRoomCount = value; }

void IncStepCount() { ++m\_stepCount; }

void DecStepCount() { --m\_stepCount; }

void IncBranchCount() { ++m\_branchCount; }

void DecBranchCount() { --m\_branchCount; }

void IncBranchRoomCount() { ++m\_branchRoomCount; }

void DecBranchRoomCount() { --m\_branchRoomCount; }

protected:

bool SetRelation(int room1, int room2, bool canRoom1ToRoom2, bool canRoom2ToRoom1);

protected:

int m\_horizRoomCount;

int m\_vertRoomCount;

int m\_entranceRoomNo;

int m\_exitRoomNo;

int m\_stepCount;

int m\_branchCount;

int m\_branchRoomCount;

std::vector<Room> m\_rooms;

}; // class Maze

class MazeGen

{

public:

enum ErrorCode

{

ERR\_PARAM\_MAZE\_SIZE = 1,

ERR\_GEN\_ENTRANCE\_OR\_EXIT,

ERR\_PARAM\_ENTRANCE\_OR\_EXIT,

ERR\_PARAM\_ENTRANCE\_DOOR\_DIRECTION,

ERR\_PARAM\_EXIT\_DOOR\_DIRECTION,

ERR\_PARAM\_STEP\_COUNT,

ERR\_PARAM\_BRANCH,

ERR\_VALID\_ROOM\_COUNT,

ERR\_MAZE\_INIT,

ERR\_NOT\_INITIALIZED,

ERR\_MAIN\_PATH\_LENGTH,

ERR\_BRANCH\_COUNT,

ERR\_BRANCH\_ROOM\_COUNT,

};

enum State

{

MGS\_INVALID = 0,

MGS\_INITIALIZED = 1,

MGS\_GEN\_MAIN\_PATH = 2,

MGS\_GEN\_BRANCH = 3,

MGS\_ADJUST\_BRANCH\_ROOM = 4,

MGS\_DONE = 5,

};

public:

MazeGen();

int Init(int horizRoomCount, int vertRoomCount,

int entranceRoomNo, Direction entranceDoorDirection,

int exitRoomNo, Direction exitDoorDirection,

int minStepCount, int maxStepCount,

int minBranchCount, int maxBranchCount,

int minBranchRoomCount, int maxBranchRoomCount);

int Generate(bool showdetail = false);

const int GetCurState() const { return m\_state; }

const Maze& GetMaze() const { return m\_maze; }

const std::vector<int>& GetMainPath() const { return m\_mainPath; }

void ShowRoom();

static std::string TranslateErrorCode(int errorCode);

protected:

static bool GenerateEntranceAndExitRoom(int& entranceRoomNo, int& exitRoomNo,

int entranceDoorDirection, int exitDoorDirection,

int horizRoomCount, int vertRoomCount,

int minStepCount, int maxStepCount);

bool ConnectDirectly(int startRoomNo, int startRoomDoorDirection, int endRoomNo, int endRoomDoorDirection, std::vector<int>& visitedRooms);

bool GenerateMainPath();

bool AdjustBranch(int branchCount);

bool AdjustBranchRoom();

void SetState(State state);

private:

int m\_entranceDoorDirection;

int m\_exitDoorDirection;

int m\_minStepCount;

int m\_maxStepCount;

int m\_minBranchCount;

int m\_maxBranchCount;

int m\_minBranchRoomCount;

int m\_maxBranchRoomCount;

Maze m\_maze;

int m\_state;

std::vector<int> m\_mainPath; // rooms on main path

}; // class MazeGen

} // namespace RandomMaze

#endif

int Room::InvalidRoomNo = -1;

static bool IsValidDirection(int dir)

{

return (dir >= DirectionLeft && dir <= DirectionDown);

}

static bool CheckRoomDoor(int roomNo, int doorDirection, int horizRoomCount, int vertRoomCount)

{

if ((roomNo < horizRoomCount && DirectionUp == doorDirection) ||

(roomNo / horizRoomCount == vertRoomCount - 1 && DirectionDown == doorDirection) ||

(roomNo % horizRoomCount == 0 && DirectionLeft == doorDirection) ||

(roomNo % horizRoomCount == horizRoomCount - 1 && DirectionRight == doorDirection))

{

return false;

}

return true;

}

Room::Room(int no) : m\_roomNo(no)

{

m\_sides[0] = m\_sides[1] = m\_sides[2] = m\_sides[3] = false; // default enclose

}

int Room::GetDoorCount() const

{

int doorCount = 0;

for (int dir = DirectionLeft; dir < DirectionCount; ++dir)

{

if (m\_sides[dir])

{

++doorCount;

}

}

return doorCount;

}

int Room::GetWallCount() const

{

return DirectionCount - GetDoorCount();

}

bool Room::IsBranchEndRoom() const

{

return GetDoorCount() == 1;

}

bool Room::IsEnclosedRoom() const

{

return GetDoorCount() == 0;

}

int Room::GetDoors(std::vector<Direction>& doors) const

{

doors.clear();

for (int dir = DirectionLeft; dir < DirectionCount; ++dir)

{

if (m\_sides[dir])

{

doors.push\_back(static\_cast<Direction>(dir));

}

}

return doors.size();

}

int Room::GetWalls(std::vector<Direction>& walls) const

{

walls.clear();

for (int dir = DirectionLeft; dir < DirectionCount; ++dir)

{

if (!m\_sides[dir])

{

walls.push\_back(static\_cast<Direction>(dir));

}

}

return walls.size();

}

Maze::Maze(void)

: m\_horizRoomCount(0), m\_vertRoomCount(0), m\_entranceRoomNo(-1), m\_exitRoomNo(-1),

m\_stepCount(0), m\_branchCount(0), m\_branchRoomCount(0)

{

}

Maze::~Maze(void)

{

}

Maze::Maze(const Maze & data)

{

m\_horizRoomCount = data.m\_horizRoomCount;

m\_vertRoomCount = data.m\_vertRoomCount;

m\_entranceRoomNo = data.m\_entranceRoomNo;

m\_exitRoomNo = data.m\_exitRoomNo;

m\_stepCount = data.m\_stepCount;

m\_branchCount = data.m\_branchCount;

m\_branchRoomCount = data.m\_branchRoomCount;

m\_rooms = data.m\_rooms;

}

Maze& Maze::operator = (const Maze & rhs)

{

if (this == &rhs) return \*this;

m\_horizRoomCount = rhs.m\_horizRoomCount;

m\_vertRoomCount = rhs.m\_vertRoomCount;

m\_entranceRoomNo = rhs.m\_entranceRoomNo;

m\_exitRoomNo = rhs.m\_exitRoomNo;

m\_stepCount = rhs.m\_stepCount;

m\_branchCount = rhs.m\_branchCount;

m\_branchRoomCount = rhs.m\_branchRoomCount;

m\_rooms = rhs.m\_rooms;

return \*this;

}

bool Maze::Init(int horizRoomCount, int vertRoomCount, int entranceRoomNo, int exitRoomNo)

{

assert(horizRoomCount > 0 && vertRoomCount > 0 &&

entranceRoomNo >= 0 && entranceRoomNo < horizRoomCount \* vertRoomCount &&

exitRoomNo >= 0 && exitRoomNo < horizRoomCount \* vertRoomCount &&

entranceRoomNo != exitRoomNo);

m\_horizRoomCount = horizRoomCount;

m\_vertRoomCount = vertRoomCount;

m\_entranceRoomNo = entranceRoomNo;

m\_exitRoomNo = exitRoomNo;

m\_stepCount = 0;

m\_branchCount = 0;

m\_branchRoomCount = 0;

int totalRoomCount = m\_horizRoomCount \* m\_vertRoomCount;

m\_rooms.clear();

m\_rooms.reserve(totalRoomCount);

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

{

/\*Room aRoom(i);

if (i < m\_horizRoomCount)

{

aRoom.SetSide(DirectionUp, false);

}

if (i + m\_horizRoomCount >= totalRoomCount)

{

aRoom.SetSide(DirectionDown, false);

}

if ((i % m\_horizRoomCount) == 0)

{

aRoom.SetSide(DirectionLeft, false);

}

if ((i % m\_horizRoomCount) == m\_horizRoomCount - 1)

{

aRoom.SetSide(DirectionRight, false);

}\*/

m\_rooms.push\_back(Room(i));

}

return true;

}

void Maze::Clear()

{

m\_horizRoomCount = 0;

m\_vertRoomCount = 0;

m\_entranceRoomNo = 0;

m\_exitRoomNo = 0;

m\_stepCount = 0;

m\_branchCount = 0;

m\_branchRoomCount = 0;

m\_rooms.clear();

}

int Maze::GetNextRoomNo(int roomNo, Direction dir) const

{

assert(IsValidRoom(roomNo));

int cur\_x, cur\_y;

GetRoomCoord(roomNo, cur\_x, cur\_y);

int nextRoomNo = -1;

switch (dir)

{

case DirectionLeft:

nextRoomNo = (0 == cur\_x ? -1 : roomNo - 1);

break;

case DirectionUp:

nextRoomNo = (0 == cur\_y ? -1 : roomNo - m\_horizRoomCount);

break;

case DirectionRight:

nextRoomNo = (cur\_x + 1 == m\_horizRoomCount ? -1 : roomNo + 1);

break;

case DirectionDown:

nextRoomNo = (cur\_y + 1 == m\_vertRoomCount ? -1 : roomNo + m\_horizRoomCount);

break;

default:

break;

}

return nextRoomNo;

}

bool Maze::SetRelation(int room1, int room2, bool canRoom1ToRoom2, bool canRoom2ToRoom1)

{

if (room1 == room2 || !IsValidRoom(room1) || !IsValidRoom(room2))

{

return false;

}

int room1\_x, room1\_y, room2\_x, room2\_y;

GetRoomCoord(room1, room1\_x, room1\_y);

GetRoomCoord(room2, room2\_x, room2\_y);

if (room1\_x != room2\_x && room1\_y != room2\_y)

{

return false;

}

if (room1\_x + 1 == room2\_x) // room2 is on the right side of room1

{

GetRoom(room1).SetSide(DirectionRight, canRoom1ToRoom2);

GetRoom(room2).SetSide(DirectionLeft, canRoom2ToRoom1);

return true;

}

else if (room1\_x - 1 == room2\_x) // room2 is on the left side of room1

{

GetRoom(room1).SetSide(DirectionLeft, canRoom1ToRoom2);

GetRoom(room2).SetSide(DirectionRight, canRoom2ToRoom1);

return true;

}

else if (room1\_y + 1 == room2\_y) // room2 is under room1

{

GetRoom(room1).SetSide(DirectionDown, canRoom1ToRoom2);

GetRoom(room2).SetSide(DirectionUp, canRoom2ToRoom1);

return true;

}

else if (room1\_y - 1 == room2\_y) // room2 is above room1

{

GetRoom(room1).SetSide(DirectionUp, canRoom1ToRoom2);

GetRoom(room2).SetSide(DirectionDown, canRoom2ToRoom1);

return true;

}

return false;

}

bool Maze::SetDoor(int room1, int room2)

{

return SetRelation(room1, room2, true, true);

}

bool Maze::SetWall(int room1, int room2)

{

return SetRelation(room1, room2, false, false);

}

bool Maze::IsValidRoom(int roomNo) const

{

return (roomNo >= 0 && roomNo < GetTotalRoomCount());

}

bool Maze::IsNeighbor(int room1, int room2) const

{

if (room1 == room2 || !IsValidRoom(room1) || !IsValidRoom(room2))

{

return false;

}

int room1\_x, room1\_y, room2\_x, room2\_y;

GetRoomCoord(room1, room1\_x, room1\_y);

GetRoomCoord(room2, room2\_x, room2\_y);

return (abs(room2\_x - room1\_x) + abs(room2\_y - room1\_y) == 1);

}

void Maze::GetRoomCoord(int roomNo, int& x, int& y) const

{

assert(roomNo >= 0 && (size\_t)roomNo < m\_rooms.size());

x = roomNo % m\_horizRoomCount;

y = roomNo / m\_horizRoomCount;

}

int Maze::GetRoomNo(int x, int y) const

{

if (x < 0 || x >= m\_horizRoomCount || y < 0 || y >= m\_vertRoomCount)

{

return -1;

}

return x + y \* m\_horizRoomCount;

}

int Maze::GetManhDistance(int roomNo1, int roomNo2) const

{

if (!IsValidRoom(roomNo1) || !IsValidRoom(roomNo2))

{

return 0;

}

int room1\_x, room1\_y, room2\_x, room2\_y;

GetRoomCoord(roomNo1, room1\_x, room1\_y);

GetRoomCoord(roomNo2, room2\_x, room2\_y);

return abs(room2\_x - room1\_x) + abs(room2\_y - room1\_y);

}

static int GetManhattanRooms(int room1, int room2, int horizRoomCount)

{

assert(room1 >= 0 && room2 >= 0);

int x1 = room1 % horizRoomCount;

int y1 = room1 / horizRoomCount;

int x2 = room2 % horizRoomCount;

int y2 = room2 / horizRoomCount;

return abs(x2 - x1) + abs(y2 - y1) + 1;

}

struct Node

{

int roomNo;

Node\* parent;

std::map<int, Node> children;

Node(Node\* \_parent = NULL, int \_no = -1)

: roomNo(\_no), parent(\_parent)

{

}

};

static int GetUnvisitedNeighborRooms(const Maze& maze, int curRoomNo,

const std::vector<int>& visitedRooms,

const std::map<int, Node>\* visitedTree,

std::vector<int>& unvisitedNeighborRooms)

{

unvisitedNeighborRooms.clear();

for (int dir = DirectionLeft; dir <= DirectionDown; ++dir)

{

int nextRoomNo = maze.GetNextRoomNo(curRoomNo, static\_cast<Direction>(dir));

if (maze.IsValidRoom(nextRoomNo) &&

(NULL == visitedTree || visitedTree->find(nextRoomNo) == visitedTree->end()) &&

std::find(visitedRooms.begin(), visitedRooms.end(), nextRoomNo) == visitedRooms.end()) // unvisited

{

unvisitedNeighborRooms.push\_back(nextRoomNo);

}

}

return unvisitedNeighborRooms.size();

}

static int GetUnvisitedNeighborCount(const Maze& maze, int curRoomNo)

{

int count = 0;

for (int dir = DirectionLeft; dir <= DirectionDown; ++dir)

{

int nextRoomNo = maze.GetNextRoomNo(curRoomNo, static\_cast<Direction>(dir));

if (maze.IsValidRoom(nextRoomNo) &&

maze.GetRoom(nextRoomNo).IsEnclosedRoom()) // unvisited

{

++count;

}

}

return count;

}

MazeGen::MazeGen()

: m\_entranceDoorDirection(DirectionInvalid), m\_exitDoorDirection(DirectionInvalid),

m\_minStepCount(0), m\_maxStepCount(0), m\_minBranchCount(0), m\_maxBranchCount(0),

m\_minBranchRoomCount(0), m\_maxBranchRoomCount(0),m\_state(MGS\_INVALID)

{

}

void MazeGen::SetState(State state)

{

m\_state = state;

}

bool MazeGen::GenerateEntranceAndExitRoom(int& entranceRoomNo, int& exitRoomNo,

int entranceDoorDirection, int exitDoorDirection,

int horizRoomCount, int vertRoomCount,

int minStepCount, int maxStepCount)

{

assert(horizRoomCount > 0 && vertRoomCount > 0);

assert(minStepCount <= maxStepCount && minStepCount >= 0);

assert(entranceRoomNo < 0 || exitRoomNo < 0);

int roomCount = horizRoomCount \* vertRoomCount;

if (minStepCount >= roomCount)

{

return false;

}

if ((1 == horizRoomCount && (DirectionLeft == entranceDoorDirection ||

DirectionRight == entranceDoorDirection ||

DirectionLeft == exitDoorDirection ||

DirectionRight == exitDoorDirection)) ||

(1 == vertRoomCount && (DirectionUp == entranceDoorDirection ||

DirectionDown == entranceDoorDirection ||

DirectionUp == exitDoorDirection ||

DirectionDown == exitDoorDirection)))

{

return false;

}

srand((unsigned int)time(NULL));

int originalEntranceRoomNo = entranceRoomNo;

int originalExitRoomNo = exitRoomNo;

int retryTimes = 0;

while (retryTimes++ < 200)

{

if (originalEntranceRoomNo < 0)

{

entranceRoomNo = (rand() % roomCount);

if (!CheckRoomDoor(entranceRoomNo, entranceDoorDirection, horizRoomCount, vertRoomCount))

{

continue;

}

}

if (originalExitRoomNo < 0)

{

exitRoomNo = (rand() % roomCount);

if (!CheckRoomDoor(exitRoomNo, exitDoorDirection, horizRoomCount, vertRoomCount))

{

continue;

}

}

int tmpManhDistance = GetManhattanRooms(entranceRoomNo, exitRoomNo, horizRoomCount);

if (minStepCount > 2 && tmpManhDistance > 2 && tmpManhDistance < maxStepCount)

{

break;

}

}

return true;

}

int MazeGen::Init(int horizRoomCount, int vertRoomCount,

int entranceRoomNo, Direction entranceDoorDirection,

int exitRoomNo, Direction exitDoorDirection,

int minStepCount, int maxStepCount,

int minBranchCount, int maxBranchCount,

int minBranchRoomCount, int maxBranchRoomCount)

{

SetState(MGS\_INVALID);

if (!IsValidDirection(entranceDoorDirection))

{

entranceDoorDirection = DirectionInvalid;

}

if (!IsValidDirection(exitDoorDirection))

{

exitDoorDirection = DirectionInvalid;

}

if (horizRoomCount <= 0 || vertRoomCount <= 0)

{

return ERR\_PARAM\_MAZE\_SIZE;

}

if (entranceRoomNo < 0 || exitRoomNo < 0)

{

// Randomize entrance room and exit room

if (!GenerateEntranceAndExitRoom(entranceRoomNo, exitRoomNo,

entranceDoorDirection, exitDoorDirection,

horizRoomCount, vertRoomCount,

minStepCount, maxStepCount))

{

return ERR\_GEN\_ENTRANCE\_OR\_EXIT;

}

}

if (entranceRoomNo < 0 || entranceRoomNo >= horizRoomCount \* vertRoomCount ||

exitRoomNo < 0 || exitRoomNo >= horizRoomCount \* vertRoomCount ||

entranceRoomNo == exitRoomNo)

{

return ERR\_PARAM\_ENTRANCE\_OR\_EXIT;

}

if (!CheckRoomDoor(entranceRoomNo, entranceDoorDirection, horizRoomCount, vertRoomCount))

{

return ERR\_PARAM\_ENTRANCE\_DOOR\_DIRECTION;

}

if (!CheckRoomDoor(exitRoomNo, exitDoorDirection, horizRoomCount, vertRoomCount))

{

return ERR\_PARAM\_EXIT\_DOOR\_DIRECTION;

}

if (minStepCount > maxStepCount || minStepCount <= 2 ||

maxStepCount < GetManhattanRooms(entranceRoomNo, exitRoomNo, horizRoomCount) ||

minStepCount > horizRoomCount \* vertRoomCount)

{

return ERR\_PARAM\_STEP\_COUNT;

}

if ((1 == horizRoomCount || 1 == vertRoomCount) && minStepCount > GetManhattanRooms(entranceRoomNo, exitRoomNo, horizRoomCount))

{

return ERR\_PARAM\_STEP\_COUNT;

}

if (minBranchCount > maxBranchCount || minBranchCount < 0 ||

minBranchRoomCount > maxBranchRoomCount || minBranchRoomCount < 0 ||

minBranchCount > maxBranchRoomCount)

{

return ERR\_PARAM\_BRANCH;

}

if (minStepCount + minBranchRoomCount > horizRoomCount \* vertRoomCount)

{

return ERR\_VALID\_ROOM\_COUNT;

}

if (!m\_maze.Init(horizRoomCount, vertRoomCount, entranceRoomNo, exitRoomNo))

{

return ERR\_MAZE\_INIT;

}

if (m\_maze.GetManhDistance(entranceRoomNo, exitRoomNo) == 1)

{

if (IsValidDirection(entranceDoorDirection) && IsValidDirection(exitDoorDirection) &&

(m\_maze.GetNextRoomNo(entranceRoomNo, entranceDoorDirection) != exitRoomNo ||

m\_maze.GetNextRoomNo(exitRoomNo, exitDoorDirection) != entranceRoomNo))

{

return ERR\_PARAM\_ENTRANCE\_OR\_EXIT;

}

}

m\_mainPath.clear();

m\_entranceDoorDirection = entranceDoorDirection;

m\_exitDoorDirection = exitDoorDirection;

m\_minStepCount = minStepCount;

m\_maxStepCount = maxStepCount;

m\_minBranchCount = minBranchCount;

m\_maxBranchCount = maxBranchCount;

m\_minBranchRoomCount = minBranchRoomCount;

m\_maxBranchRoomCount = maxBranchRoomCount;

srand((unsigned int)time(NULL));

SetState(MGS\_INITIALIZED);

return 0;

}

int MazeGen::Generate(bool showdetail)

{

if (MGS\_DONE == m\_state)

{

return 0;

}

if (MGS\_INITIALIZED != m\_state)

{

return ERR\_NOT\_INITIALIZED;

}

SHOW\_TOKEN("============== GenerateMainPath =================\n")

// Try to satisfy step count condition

if (!GenerateMainPath())

{

return ERR\_MAIN\_PATH\_LENGTH;

}

SHOW\_DETAIL();

// Try to satisfy the branch count condition

SHOW\_TOKEN("============== AdjustBranch =================\n")

int expectedBranchCount = m\_minBranchCount + (rand() % (m\_maxBranchCount - m\_minBranchCount + 1));

if (!AdjustBranch(expectedBranchCount))

{

return ERR\_BRANCH\_COUNT;

}

SHOW\_DETAIL();

// Try to satisfy branch room count condition

while (m\_maze.GetBranchCount() <= m\_maxBranchCount)

{

SHOW\_TOKEN("============== AdjustBranchRoom =================\n")

if (AdjustBranchRoom())

{

SHOW\_DETAIL();

break;

}

if (m\_maze.GetBranchCount() == m\_maxBranchCount)

{

return ERR\_BRANCH\_ROOM\_COUNT;

}

int deltaBranchCount = rand() % (m\_maxBranchCount - m\_maze.GetBranchCount() + 1);

if (!AdjustBranch(m\_maze.GetBranchCount() + deltaBranchCount))

{

return ERR\_BRANCH\_COUNT;

}

SHOW\_DETAIL();

}

SetState(MGS\_DONE);

return 0;

}

bool MazeGen::ConnectDirectly(int startRoomNo, int startRoomDoorDirection, int endRoomNo, int endRoomDoorDirection, std::vector<int>& visitedRooms)

{

visitedRooms.clear();

int tmpStartRoomNo = startRoomNo;

int tmpEndRoomNo = endRoomNo;

// If the entrance door direction or the exit door direction is specified

if (IsValidDirection(startRoomDoorDirection))

{

tmpStartRoomNo = m\_maze.GetNextRoomNo(startRoomNo, (Direction)startRoomDoorDirection);

if (!m\_maze.IsValidRoom(tmpStartRoomNo))

{

return false;

}

m\_maze.SetDoor(startRoomNo, tmpStartRoomNo);

visitedRooms.push\_back(startRoomNo);

m\_maze.IncStepCount();

}

if (IsValidDirection(endRoomDoorDirection))

{

tmpEndRoomNo = m\_maze.GetNextRoomNo(endRoomNo, (Direction)endRoomDoorDirection);

if (!m\_maze.IsValidRoom(tmpEndRoomNo))

{

return false;

}

}

int start\_x, start\_y, end\_x, end\_y;

m\_maze.GetRoomCoord(tmpStartRoomNo, start\_x, start\_y);

m\_maze.GetRoomCoord(tmpEndRoomNo, end\_x, end\_y);

int cur\_x = start\_x;

int cur\_y = start\_y;

int step\_x = (end\_x - cur\_x > 0 ? 1 : (end\_x - cur\_x == 0 ? 0 : -1));

int step\_y = (end\_y - cur\_y > 0 ? 1 : (end\_y - cur\_y == 0 ? 0 : -1));

int curRoomNo = tmpStartRoomNo;

int prevRoomNo = curRoomNo;

visitedRooms.push\_back(curRoomNo);

m\_maze.IncStepCount();

while (curRoomNo != tmpEndRoomNo)

{

if (m\_maze.GetManhDistance(curRoomNo, tmpEndRoomNo) > m\_maze.GetManhDistance(startRoomNo, tmpEndRoomNo) ||

m\_maze.GetManhDistance(curRoomNo, tmpEndRoomNo) > m\_maze.GetManhDistance(curRoomNo, endRoomNo))

{

std::vector<int> unvisitedNeighborRooms;

GetUnvisitedNeighborRooms(m\_maze, curRoomNo, visitedRooms, NULL, unvisitedNeighborRooms);

if (tmpEndRoomNo != endRoomNo)

{

// Make sure endRoomNo won't be reached before tmpEndRoomNo

std::vector<int>::iterator it\_EndRoom = std::find(unvisitedNeighborRooms.begin(), unvisitedNeighborRooms.end(), endRoomNo);

if (unvisitedNeighborRooms.end() != it\_EndRoom)

{

unvisitedNeighborRooms.erase(it\_EndRoom);

}

}

if (unvisitedNeighborRooms.empty())

{

return false;

}

int tmpMinManhDistance = 0x0FFFFFFF;

curRoomNo = -1;

for (std::vector<int>::iterator it = unvisitedNeighborRooms.begin(); it != unvisitedNeighborRooms.end(); ++it)

{

int tmpManhDistance = m\_maze.GetManhDistance(\*it, tmpEndRoomNo);

if (tmpManhDistance < tmpMinManhDistance)

{

tmpMinManhDistance = tmpManhDistance;

curRoomNo = \*it;

}

}

if (-1 == curRoomNo)

{

return false;

}

if (tmpEndRoomNo != endRoomNo && curRoomNo == endRoomNo)

{

continue;

}

m\_maze.GetRoomCoord(curRoomNo, cur\_x, cur\_y);

}

else

{

int delta\_x = 0, delta\_y = 0;

if (cur\_x != end\_x)

{

if (cur\_y != end\_y)

{

delta\_x = ((rand() % 100) > 50 ? step\_x : 0);

}

else

{

delta\_x = step\_x;

}

}

if (cur\_y != end\_y)

{

if (cur\_x != end\_x && delta\_x != 0)

{

delta\_y = 0;

}

else

{

delta\_y = step\_y;

}

}

curRoomNo = m\_maze.GetRoomNo(cur\_x + delta\_x, cur\_y + delta\_y);

if (std::find(visitedRooms.begin(), visitedRooms.end(), curRoomNo) != visitedRooms.end())

{

continue;

}

if (tmpEndRoomNo != endRoomNo && curRoomNo == endRoomNo)

{

continue;

}

cur\_x += delta\_x;

cur\_y += delta\_y;

}

m\_maze.SetDoor(prevRoomNo, curRoomNo);

m\_maze.IncStepCount();

visitedRooms.push\_back(curRoomNo);

prevRoomNo = curRoomNo;

// Adjust step

step\_x = (end\_x - cur\_x > 0 ? 1 : (end\_x - cur\_x == 0 ? 0 : -1));

step\_y = (end\_y - cur\_y > 0 ? 1 : (end\_y - cur\_y == 0 ? 0 : -1));

}

if (tmpEndRoomNo != endRoomNo)

{

m\_maze.SetDoor(tmpEndRoomNo, endRoomNo);

visitedRooms.push\_back(endRoomNo);

m\_maze.IncStepCount();

}

return true;

}

bool MazeGen::GenerateMainPath()

{

SetState(MGS\_GEN\_MAIN\_PATH);

int curRoomNo;

// Generate the shortest path from entrance room to exit room

if (!ConnectDirectly(m\_maze.GetEntranceRoomNo(), m\_entranceDoorDirection,

m\_maze.GetExitRoomNo(), m\_exitDoorDirection,

m\_mainPath))

{

return false;

}

assert((size\_t)m\_maze.GetStepCount() == m\_mainPath.size());

if (m\_maze.GetStepCount() > m\_maxStepCount)

{

return false;

}

else if (m\_maze.GetStepCount() == m\_maxStepCount)

{

return true;

}

if (m\_maze.GetStepCount() <= 1)

{

return false;

}

if (m\_maze.GetHorizRoomCount() > 1 && m\_maze.GetVertRoomCount() > 1)

{

int retryTimes = 0;

do

{

// Searching depth should be small enough to avoid time-consuming backtracking

int expectedStepCount = 0;

if (m\_maze.GetStepCount() >= m\_minStepCount)

{

expectedStepCount = m\_minStepCount + (rand() % ((m\_maxStepCount - m\_minStepCount + 1) / 3));

if (m\_maze.GetStepCount() >= expectedStepCount)

{

break;

}

}

else

{

expectedStepCount = m\_maze.GetStepCount() + (rand() % 6);

if (expectedStepCount > m\_maxStepCount)

{

expectedStepCount = m\_maxStepCount;

}

if (m\_maze.GetStepCount() >= expectedStepCount)

{

continue;

}

}

assert(m\_mainPath.size() > 0);

int roomNo1, roomNo2;

if (m\_mainPath.size() > 2)

{

int index1 = (rand() % m\_mainPath.size());

roomNo1 = m\_mainPath[index1];

if ((m\_maze.GetEntranceRoomNo() == roomNo1 && IsValidDirection(m\_entranceDoorDirection)) ||

(m\_maze.GetExitRoomNo() == roomNo1 && IsValidDirection(m\_exitDoorDirection)))

{

// If the entrance door direction or the exit door direction is specified

continue;

}

Room& room1 = m\_maze.GetRoom(roomNo1);

std::vector<Direction> doors;

room1.GetDoors(doors);

assert(doors.size() > 0);

roomNo2 = m\_maze.GetNextRoomNo(roomNo1, doors[rand() % doors.size()]);

if ((m\_maze.GetEntranceRoomNo() == roomNo2 && IsValidDirection(m\_entranceDoorDirection)) ||

(m\_maze.GetExitRoomNo() == roomNo2 && IsValidDirection(m\_exitDoorDirection)))

{

// If the entrance door direction or the exit door direction is specified

continue;

}

m\_maze.SetWall(roomNo1, roomNo2);

doors.clear();

if (m\_maze.GetRoom(roomNo2).GetDoors(doors) == 0)

{

int tmpRoomNo = roomNo1;

roomNo1 = roomNo2;

roomNo2 = tmpRoomNo;

m\_maze.GetRoom(roomNo2).GetDoors(doors);

assert(doors.size() == 1);

}

int roomNo3 = m\_maze.GetNextRoomNo(roomNo2, doors[rand() % doors.size()]);

if ((m\_maze.GetEntranceRoomNo() == roomNo3 && IsValidDirection(m\_entranceDoorDirection)) ||

(m\_maze.GetExitRoomNo() == roomNo3 && IsValidDirection(m\_exitDoorDirection)))

{

// If the entrance door direction or the exit door direction is specified

//m\_maze.SetDoor(roomNo1, roomNo2);

//continue;

}

else

{

m\_maze.SetWall(roomNo2, roomNo3);

m\_mainPath.erase(std::find(m\_mainPath.begin(), m\_mainPath.end(), roomNo2));

m\_maze.DecStepCount();

roomNo2 = roomNo3;

}

}

else

{

roomNo1 = m\_mainPath[0];

roomNo2 = m\_mainPath[1];

if (((m\_maze.GetEntranceRoomNo() == roomNo1 || m\_maze.GetEntranceRoomNo() == roomNo2) && IsValidDirection(m\_entranceDoorDirection)) ||

((m\_maze.GetExitRoomNo() == roomNo1 || m\_maze.GetExitRoomNo() == roomNo2) && IsValidDirection(m\_exitDoorDirection)))

{

// If the entrance door direction or the exit door direction is specified

break;

}

m\_maze.SetWall(roomNo1, roomNo2);

m\_maze.DecStepCount();

m\_maze.DecStepCount();

}

// Make sure room1 is ahead of room2

if (std::find(m\_mainPath.begin(), m\_mainPath.end(), roomNo1) > std::find(m\_mainPath.begin(), m\_mainPath.end(), roomNo2))

{

int tmpRoomNo = roomNo1;

roomNo1 = roomNo2;

roomNo2 = tmpRoomNo;

}

std::stack<int> wayPointStack;

std::vector<int> unvisitedNeighborRooms;

Node visitedTree;

curRoomNo = roomNo1;

wayPointStack.push(curRoomNo);

Node\* curNode = & visitedTree;

curNode->roomNo = curRoomNo;

curNode->parent = NULL;

while (curRoomNo != roomNo2)

{

unvisitedNeighborRooms.clear();

if (m\_maze.GetStepCount() + GetManhattanRooms(curRoomNo, roomNo2, m\_maze.GetHorizRoomCount()) - 2 <= expectedStepCount)

{

GetUnvisitedNeighborRooms(m\_maze, curRoomNo, m\_mainPath, &(curNode->children), unvisitedNeighborRooms);

if (m\_maze.IsNeighbor(curRoomNo, roomNo2))

{

unvisitedNeighborRooms.push\_back(roomNo2);

}

}

if (unvisitedNeighborRooms.size() > 0)

{

int nextRoomNo = unvisitedNeighborRooms[rand() % unvisitedNeighborRooms.size()];

m\_maze.SetDoor(curRoomNo, nextRoomNo);

if (nextRoomNo != roomNo2)

{

m\_maze.IncStepCount();

m\_mainPath.insert(std::find(m\_mainPath.begin(), m\_mainPath.end(), roomNo2), nextRoomNo);

}

curNode->children[nextRoomNo] = Node(curNode, nextRoomNo);

wayPointStack.push(nextRoomNo);

curRoomNo = nextRoomNo;

curNode = &curNode->children[curRoomNo];

continue;

}

else if (wayPointStack.size() > 0)

{

wayPointStack.pop();

}

if (wayPointStack.size() > 0)

{

m\_maze.SetWall(wayPointStack.top(), curRoomNo);

m\_maze.DecStepCount();

m\_mainPath.erase(std::find(m\_mainPath.begin(), m\_mainPath.end(), curRoomNo));

curRoomNo = wayPointStack.top();

curNode = curNode->parent;

}

else

{

// ATTENTION: NEVER REACH HERE

break;

}

}

} while (m\_maze.GetStepCount() < m\_minStepCount && ++retryTimes <= 500);

}

return (m\_maze.GetStepCount() >= m\_minStepCount && m\_maze.GetStepCount() <= m\_maxStepCount);

}

bool MazeGen::AdjustBranch(int branchCount)

{

SetState(MGS\_GEN\_BRANCH);

if (m\_maze.GetHorizRoomCount() > 1 && m\_maze.GetVertRoomCount() > 1)

{

// Try to satisfy the branch count condition

if (m\_maze.GetBranchCount() < branchCount)

{

std::vector<int> candidateRooms;

for (int roomNo = 0; roomNo < m\_maze.GetTotalRoomCount(); ++roomNo)

{

Room& room = m\_maze.GetRoom(roomNo);

if (!room.IsEnclosedRoom() && !room.IsBranchEndRoom())

{

if ((m\_maze.GetEntranceRoomNo() == roomNo && IsValidDirection(m\_entranceDoorDirection)) ||

(m\_maze.GetExitRoomNo() == roomNo && IsValidDirection(m\_exitDoorDirection)))

{

// If the entrance door direction or the exit door direction is specified

continue;

}

candidateRooms.push\_back(roomNo);

}

}

while (m\_maze.GetBranchCount() < branchCount && candidateRooms.size() > 0)

{

int index = (rand() % candidateRooms.size());

int roomNo = candidateRooms[index];

std::vector<Direction> walls;

m\_maze.GetRoom(roomNo).GetWalls(walls);

while (walls.size() > 0)

{

int index = (rand() % walls.size());

int nextRoomNo = m\_maze.GetNextRoomNo(roomNo, walls[index]);

if (m\_maze.IsValidRoom(nextRoomNo) && m\_maze.GetRoom(nextRoomNo).IsEnclosedRoom())

{

m\_maze.SetDoor(roomNo, nextRoomNo);

m\_maze.IncBranchCount();

m\_maze.IncBranchRoomCount();

break;

}

else

{

walls.erase(walls.begin() + index);

}

}

if (walls.size() == 0)

{

candidateRooms.erase(candidateRooms.begin() + index);

}

}

}

else if (m\_maze.GetBranchCount() > branchCount)

{

std::vector<int> branchEndRooms;

for (int roomNo = 0; roomNo < m\_maze.GetTotalRoomCount() && branchEndRooms.size() < (size\_t)m\_maze.GetBranchCount(); ++roomNo)

{

if (m\_maze.GetRoom(roomNo).IsBranchEndRoom() && !m\_maze.IsEntranceRoom(roomNo) && !m\_maze.IsExitRoom(roomNo))

{

branchEndRooms.push\_back(roomNo);

}

}

while (m\_maze.GetBranchCount() > branchCount && branchEndRooms.size() > 0)

{

int index = (rand() % branchEndRooms.size());

int branchEndRoomNo = branchEndRooms[index];

std::vector<Direction> doors;

m\_maze.GetRoom(branchEndRoomNo).GetDoors(doors);

assert(doors.size() == 1);

int nextRoomNo = m\_maze.GetNextRoomNo(branchEndRoomNo, doors[0]);

if (m\_maze.IsValidRoom(nextRoomNo))

{

m\_maze.SetWall(nextRoomNo, branchEndRoomNo);

m\_maze.DecBranchRoomCount();

branchEndRooms.erase(branchEndRooms.begin() + index);

if (m\_maze.GetRoom(nextRoomNo).IsBranchEndRoom())

{

branchEndRooms.push\_back(nextRoomNo);

}

else

{

m\_maze.DecBranchCount();

}

}

}

}

}

return (m\_maze.GetBranchCount() >= m\_minBranchCount && m\_maze.GetBranchCount() <= m\_maxBranchCount);

}

bool MazeGen::AdjustBranchRoom()

{

SetState(MGS\_ADJUST\_BRANCH\_ROOM);

if (m\_maze.GetBranchRoomCount() >= m\_minBranchRoomCount && m\_maze.GetBranchRoomCount() <= m\_maxBranchRoomCount)

{

return true;

}

if (m\_maze.GetHorizRoomCount() > 1 && m\_maze.GetVertRoomCount() > 1)

{

std::vector<int> branchEndRooms;

for (int roomNo = 0; roomNo < m\_maze.GetTotalRoomCount() && branchEndRooms.size() < (size\_t)m\_maze.GetBranchCount(); ++roomNo)

{

if (m\_maze.GetRoom(roomNo).IsBranchEndRoom() && !m\_maze.IsEntranceRoom(roomNo) && !m\_maze.IsExitRoom(roomNo))

{

std::vector<Direction> walls;

m\_maze.GetRoom(roomNo).GetWalls(walls);

for (std::vector<Direction>::iterator it = walls.begin(); it != walls.end(); ++it)

{

int nextRoomNo = m\_maze.GetNextRoomNo(roomNo, \*it);

if (m\_maze.IsValidRoom(nextRoomNo) && m\_maze.GetRoom(nextRoomNo).IsEnclosedRoom())

{

branchEndRooms.push\_back(roomNo);

break;

}

}

}

}

// Try to satisfy branch room count condition

// TODO: How if all branch end rooms have no unvisited neighbors?

if (m\_maze.GetBranchRoomCount() < m\_minBranchRoomCount)

{

while (m\_maze.GetBranchRoomCount() < m\_minBranchRoomCount && branchEndRooms.size() > 0)

{

std::vector<Direction> walls;

int index = (rand() % branchEndRooms.size());

int nextRoomNo = 0;

int branchEndRoomNo = branchEndRooms[index];

assert(branchEndRoomNo >= 0);

m\_maze.GetRoom(branchEndRoomNo).GetWalls(walls);

while (walls.size() > 0)

{

int randWallIndex = (rand() % walls.size());

nextRoomNo = m\_maze.GetNextRoomNo(branchEndRoomNo, walls[randWallIndex]);

if (m\_maze.IsValidRoom(nextRoomNo) && m\_maze.GetRoom(nextRoomNo).IsEnclosedRoom())

{

m\_maze.SetDoor(branchEndRoomNo, nextRoomNo);

m\_maze.IncBranchRoomCount();

branchEndRooms.erase(branchEndRooms.begin() + index);

branchEndRooms.push\_back(nextRoomNo);

break;

}

else

{

walls.erase(walls.begin() + randWallIndex);

}

}

if (walls.size() == 0)

{

branchEndRooms.erase(branchEndRooms.begin() + index);

}

}

}

else if (m\_maze.GetBranchRoomCount() > m\_maxBranchRoomCount)

{

while (m\_maze.GetBranchRoomCount() > m\_maxBranchRoomCount && branchEndRooms.size() > 0)

{

std::vector<Direction> doors;

int index = (rand() % branchEndRooms.size());

int nextRoomNo = 0;

int branchEndRoomNo = branchEndRooms[index];

assert(branchEndRoomNo >= 0);

m\_maze.GetRoom(branchEndRoomNo).GetDoors(doors);

while (doors.size() > 0)

{

int randDoorIndex = (rand() % doors.size());

nextRoomNo = m\_maze.GetNextRoomNo(branchEndRoomNo, doors[randDoorIndex]);

if (m\_maze.IsValidRoom(nextRoomNo) && m\_maze.GetRoom(nextRoomNo).GetDoorCount() == 2)

{

m\_maze.SetWall(nextRoomNo, branchEndRoomNo);

m\_maze.DecBranchRoomCount();

branchEndRooms.erase(branchEndRooms.begin() + index);

branchEndRooms.push\_back(nextRoomNo);

break;

}

else

{

doors.erase(doors.begin() + randDoorIndex);

}

}

if (doors.size() == 0)

{

branchEndRooms.erase(branchEndRooms.begin() + index);

}

}

}

}

return (m\_maze.GetBranchRoomCount() >= m\_minBranchRoomCount && m\_maze.GetBranchRoomCount() <= m\_maxBranchRoomCount);

}

std::string MazeGen::TranslateErrorCode(int errorCode)

{

switch (errorCode)

{

case 0:

return "成功";

case MazeGen::ERR\_PARAM\_MAZE\_SIZE:

return "迷宫大小设定错误";

case MazeGen::ERR\_GEN\_ENTRANCE\_OR\_EXIT:

return "生成入口/出口错误";

case MazeGen::ERR\_PARAM\_ENTRANCE\_OR\_EXIT:

return "入口/出口设定错误";

case MazeGen::ERR\_PARAM\_ENTRANCE\_DOOR\_DIRECTION:

return "入口门位置设定错误";

case MazeGen::ERR\_PARAM\_EXIT\_DOOR\_DIRECTION:

return "出口门位置设定错误";

case MazeGen::ERR\_PARAM\_STEP\_COUNT:

return "主路径总房间数设定错误";

case MazeGen::ERR\_PARAM\_BRANCH:

return "分支设定错误";

case MazeGen::ERR\_VALID\_ROOM\_COUNT:

return "最小有效房间数设定错误";

case MazeGen::ERR\_MAZE\_INIT:

return "迷宫初始化失败";

case MazeGen::ERR\_NOT\_INITIALIZED:

return "迷宫生成器未初始化";

case MazeGen::ERR\_MAIN\_PATH\_LENGTH:

return "主路径长度错误";

case MazeGen::ERR\_BRANCH\_COUNT:

return "分支数错误";

case MazeGen::ERR\_BRANCH\_ROOM\_COUNT:

return "分支房间数错误";

default:

return "未知错误";

}

return "";

}