**神灵的遗产**

目录：

1. [游戏概述](#游戏概述)
2. [游戏内容](#游戏内容)
3. [团队分工](#团队分工)
4. [概念设计](#概念设计)
5. [详细设计](#详细设计)

1.[迷雾控制模块](#迷雾控制模块)

2.[随机事件模块](#随机事件模块)

3.[普通怪物模块](#普通怪物模块)

4.[装备查看穿戴模块](#装备查看穿戴模块)

5.[英雄模块](#英雄模块)

6.[数据库控制模块](#数据库控制模块)

1. [界面设计](#界面设计)

1.[主菜单](#主菜单)

2.[初始选择界面](#初始选择界面)

3.[角色选择](#角色选择)

4.[探索地宫界面](#探索地宫界面)

5.[背包界面](#背包界面)

6.[关卡间动画](#关卡间动画)

7.[暂停选择界面](#暂停选择界面)

8.[Boss关界面](#Boss关界面)

1. [总结](#总结)

**一．****游戏概述：**

游戏名称：神灵的遗产

面向人群：14——30

游戏平台：Windows

游戏类型：回合制冒险类游戏

**二．****游戏内容：**

**游戏背景**：

天使对宇宙的混乱已经感觉到了厌倦，于是对地狱主动发起了进攻。可是遭殃的却是人间界。当恶魔与天使的战火愈演愈烈，世界的边缘开始崩塌，群星开始从天空中坠落，人类面临灭绝。

**游戏剧情：**

在经历世界的终结之后，一位幸存者受到命运的召唤来到了一处地宫的门前，这是一座从无尽高处坠落下来的地宫，地宫的墙壁上铭刻着许多远古的符文，他在地宫还感受到了种种神秘力量的召唤。他不由自主地朝地宫中走去，他在心中对自己说：这里肯定有世界终极的奥秘，我也希望自己还可以拯救这个没有希望的世界。

**操作方法：**

鼠标点击，点开房间中的迷雾，点开一个格子表示前进一个回合。当翻到怪物时，点击怪物与他发生战斗，点击一次算一个回合。总而言之，在这个游戏中，每一次点击鼠标都代表着一个回合的进行。

**三．****团队分工：**

**四．****概念设计：**

**模块划分：**





**五．****详细设计：**

**主要模块设计：**

**1.****迷雾控制模块：**

**1.**初始化变量，并实例化雾图

void Awake()

{

width = Mathf.FloorToInt(fog.width / cols);

height = Mathf.FloorToInt(fog.height / rows);

//初始化半雾无雾透明度

fullColor = new Color[width \* height];

for (int index = 0; index < width \* height; index++)

fullColor[index] = new Color(1, 1, 1, 1);

halfColor = new Color[width \* height];

for (int index = 0; index < width \* height; index++)

halfColor[index] = new Color(1, 1, 1, 0.5f);

initalize();

}

public void initalize()

{

//实例化以防修改源资源 ???

FogOfWar = Instantiate(fog);

mater.SetTexture("\_Mask", FogOfWar);

fogTable = new int[rows \* cols];

for (int index = 0; index < fogTable.Length; index++)

fogTable[index] = 0;

}

**2.**运行后鼠标点击地砖，执行OnMouseDown()调用canDeleteFog(int p)，deleteFog(int []pos,bool isBlock)来处理迷雾，处理原则为：

若点击处为浓雾，无雾则不处理，为淡雾则去雾并处理

void OnMouseDown()

{

Fog fog = gameObject.GetComponentInParent<Fog>();

if (!fog.canDeleteFog(near[0])) //点在black or white

return;

//可探索

// \*\*判断是否不可前进即不可探索此块周边\*\*

//

Manager manager = gameObject.GetComponentInParent<Manager>();

GameObject obj = manager.randomObject(transform, near,ref isblack);

fog.deleteFog(near, isblack);

}

//判断是否可去雾

public bool canDeleteFog(int p)

{

if (fogTable[p] == 1) ////==1

return true;

return false;

}

//去雾

public void deleteFog(int []pos,bool isBlock) //isblock 是否有阻挡物体

{

int length = pos.Length;

int x;

int y;

int col;

int row;

if (isBlock)

length = 1;

for (int index = 0; index < length; index++)

{

if (index == 0) //点击位置去掉全部雾

{

color = fullColor;

fogTable[pos[index]] = 2;

}

else

{

if (fogTable[pos[index]] != 0) //fog alreadly had been clear

continue;

color = halfColor;

fogTable[pos[index]] = 1;

}

row = (pos[index] / rows);

col = pos[index] % rows;

x = col \* width;

y = FogOfWar.height - (row + 1) \* height;

FogOfWar.SetPixels(x, y, width, height, color);

}

FogOfWar.Apply();

}

**2.****随机事件模块：**

去除雾的同时，在格子处出现随机事件，

1. 根据当前关卡，决定随机状态。
2. 判断此处是否为出口位置，若为出口，则显示出口。
3. 若不为出口，则创建随机数来模拟概率。
4. 70%此处无事件，10%概率出现怪物，20%概率出现其他事件如，出现补血道具，陷阱等。若为怪物，其周围雾不可通过以红X标记，且其属性与所处关卡有关。
5. 再从各类中随机选择某一对象实例化。

代码：

public GameObject randomObject(Transform t,int[] near,ref bool isBlock)

{

int loc = near[0];

GameObject others;

//不同级别 不同怪物道具

switch (level)

{

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

{

//先判断是否为出口，不为在随机返回物体

if (loc == entryIdex)

{

others = Instantiate(entry, t.position, t.rotation) as GameObject;

others.SetActive(false);

others.SetActive(true);

others.transform.parent = GameObject.FindGameObjectWithTag("manager").transform; //挂在manager下

//播放音效

//others.GetComponent<AudioSource>().Play();

return others;

}

int r = Random.Range(1, 11);

if (r <= 7)

{

others = Instantiate(smoke, t.position, t.rotation) as GameObject;

Destroy(others, 2);

//播放音效

// others.GetComponent<AudioSource>().Play();

return others;

}

else

{//划分怪物与物品等

if (r <= 8) //为怪物

{

//isBlock = true;

others = Instantiate(monster[Random.Range(0, monster.Length)], t.position, t.rotation) as GameObject;

others.transform.parent = GameObject.FindGameObjectWithTag("manager").transform; //挂在manager下，为了显示怪物属性

others.GetComponent<Monster>().setLevel(level);

//播放音效

// others.GetComponent<AudioSource>().Play();

//创建阻挡区

GameObject gm;

Fog fog = gameObject.GetComponent<Fog>();

int[] locs;

if (loc % cols == 0)

{

locs = new int[] { loc - cols, loc - cols + 1, loc + 1, loc + cols, loc + cols + 1 };

}

else

if(loc % cols ==cols-1)

locs = new int[] { loc - cols, loc - cols - 1, loc - 1, loc + cols, loc + cols - 1 };

else

locs = new int[] { loc - cols - 1, loc - cols, loc - cols + 1, loc - 1, loc + 1, loc + cols - 1, loc + cols, loc + cols + 1 };

for (int i = 0; i < locs.Length; i++)

{

if (fog.hasFog(locs[i]))

{

Debug.Log("" + locs[i]);

gm = Instantiate(block, floor[locs[i]].transform.position, floor[locs[i]].transform.rotation) as GameObject;

gm.transform.parent = others.transform;

}

}

return others;

}

else //为物品

{

int index = Random.Range(0, obj.Length);

if (index == 2)

isBlock = true;

others = Instantiate(obj[index], t.position, t.rotation) as GameObject;

others.transform.parent = GameObject.FindGameObjectWithTag("manager").transform; //挂在manager下，为了显示怪物属性

//播放音效

//others.GetComponent<AudioSource>().Play();

return others;

}

}

}

default: return null;

}

}

**3.****普通怪物模块：**

怪物拥有

public int attackValue; //攻击力

public int healthValue; //生命值

public int armorValue; //护甲值

public int dodgeRate; //闪避值 0-100

public int hitRate; //命中值 0-100

public bool ignoreArmor; //穿透护甲

属性。

当点击攻击怪物时，

1.根据英雄命中值及怪物闪避率计算怪物闪避此次攻击的概率。

2.用随机数模拟闪避率若命中播放命中动画，为命中播放Miss

3.若为魔法攻击即穿透护甲，直接减少生命值，若不为，先减少护甲

4.是否死亡，若死亡销毁怪物，随机获得宝箱，其中有装备，若没死，对英雄造成伤害。

void OnMouseDown()

{

Hero hero= GameObject.Find("heroPanel").GetComponent<Hero>();

int h = hero.getHitValue(); //获取命中值

/\*闪避判断\*/

int hit = (h \* (100 - dodgeRate)) / 100; //(h \* (100-dodgeValue)) / 10000 因要与随机数比 \*100

int random = Random.Range(1, 101);

if (random > hit)

{

//dont hit

return;

}

if (hitMonster != null)

{//伤害效果显示

GameObject go = Instantiate(hitMonster, transform.position, transform.rotation) as GameObject;

Destroy(go, 0.2f);

}

/\*自身受到伤害 无抗性处理，无护甲穿透\*/

//当为物理攻击不穿透护甲，护甲会随攻击减少

if (!Hero.ignoreArmor&&armorValue != 0)

{

int damage = armorValue - hero.getAttackValue();

if (damage > 0)

{

armorValue = damage;

//更新护甲

}

else

{

armorValue = 0;

healthValue += damage;

}

updateArmor();

}

else

{

healthValue -= hero.getAttackValue();

}

///检测并处理死亡

Dead();

//不死更新血量

updateHealth();

/\*对英雄造成伤害\*/

hero.getDamagedByMonster(attackValue, hitRate);

}

**4.****装备查看穿戴模块：**

1. 查看：

点击未装备的装备，显示此装备信息及已装备信息，点击装备的装备，只显示已装备信息。

/// <summary>

/// 点击已装备的装备显示此装备信息

/// </summary>

/// <param name="dic"></param>

public void showProperty(Dictionary<string, int> dic,int location)

{

currentEquipment = location; ////储物当前显示属性的装备位置

hideProperty();

wearedProperty.gameObject.transform.parent.gameObject.SetActive(true);

string text="";

foreach (KeyValuePair<string, int> kvp in dic)

{

if(kvp.Value!=0)

text += (kvp.Key + ":" + kvp.Value+"\n");

}

wearedProperty.text = text;

}

/// <summary>

/// 点击储物格装备显示已装备信息与当前装备信息

/// </summary>

/// <param name="wear"></param>

/// <param name="cur"></param>

public void showProperty(Dictionary<string, int> wear, Dictionary<string, int> cur,int location)

{

currentEquipment = location; //储存当前显示属性的装备位置

hideProperty();

string text = "";

if (wear.Count > 0)

{

wearedProperty.gameObject.transform.parent.gameObject.SetActive(true);

foreach (KeyValuePair<string, int> kvp in wear)

{

if (kvp.Value != 0)

text += (kvp.Key + ":" + kvp.Value + "\n");

}

wearedProperty.text = text;

}

curProperty.gameObject.transform.parent.gameObject.SetActive(true);

text = "";

foreach (KeyValuePair<string, int> kvp in cur)

{

if (kvp.Value != 0)

text += (kvp.Key + ":" + kvp.Value + "\n");

}

curProperty.text = text;

}

1. 穿戴：

通过拖动装备穿戴装备及交换存储位置

1. 点击处是否存在，若不存在，结束
2. 若存在，装备获得鼠标和对象之间的偏移量offset
3. 判断鼠标状态
4. 若鼠标仍为点击状态，则设置装备新位置为鼠标位置加offset，执行3，若不为点击状态则转5
5. 判断是否只是点击而不是拖动，若是，则处理显示装备信息
6. 若不是，计算当前装备与可放位置距离，与最近的位置交换装备，若位置为装备栏即为穿戴装备。

/// <summary>

/// 使用协程实现拖拽效果

/// </summary>

/// <returns></returns>

IEnumerator OnMouseDown()

{

if (gameObject.GetComponent<SpriteRenderer>().sprite == null)

{

Debug.Log("null");

GameObject.FindGameObjectWithTag("BagManager").GetComponent<BagManager>().hideProperty();

yield break;

}

var camera = Camera.main;

if (camera)

{

//转换对象到当前屏幕位置

Vector3 screenPosition = camera.WorldToScreenPoint(transform.position);

//鼠标屏幕坐标

Vector3 mScreenPosition = new Vector3(Input.mousePosition.x, Input.mousePosition.y,screenPosition.z);

//获得鼠标和对象之间的偏移量,拖拽时相机应该保持不动

Vector3 offset = transform.position - camera.ScreenToWorldPoint(mScreenPosition);

//offset.z = offset.z-30; //for clearly see

//若鼠标左键一直按着则循环继续 ,拖拽

while (Input.GetMouseButton(0))

{

//鼠标屏幕上新位置

mScreenPosition = new Vector3(Input.mousePosition.x, Input.mousePosition.y, screenPosition.z);

// 对象新坐标

transform.position = offset + camera.ScreenToWorldPoint(mScreenPosition);

//协同，等待下一帧继续

yield return new WaitForEndOfFrame();

}

//transform.position.= 30;

//判断是否只是点击

if (transform.position == orignPosition)

if(location<0) //点击的是未装备的还是已装备的

GameObject.FindGameObjectWithTag("BagManager").GetComponent<BagManager>().showProperty(information,location);

else

GameObject.FindGameObjectWithTag("BagManager").GetComponent<BagManager>().showProperty(

wear.gameObject.GetComponent<Equipment>().information, information, location);

else

dragMoveing();

}

}

**5.****英雄模块：**

属性：

public static int attackValue=10; //攻击力

public static int maxHealthValue=100; //生命上限

public static int maxMagicValue=100; //魔力上限

public static int maxGhostValue=100; //灵魂上限

public static int armorValue=0; //护甲值

public static int dodgeValue=0; //闪避值

public static int hitValue=100; //命中值

//public

public static int healthValue=100; //生命值

public static int magicValue=100; //魔力值

public static int ghostValue=0; //灵魂值

public static bool ignoreArmor = false; //穿透护甲

判断伤害部分与怪物差不多，在此逻辑处理完后，更新血条，

1. 根据状态条百分比value计算状态条显示部分高度，
2. 然后到此高度为止，逐像素设置颜色

/// <summary>

/// 根据状态条百分比value，及具体值更新显示状态条

/// </summary>

/// <param name="value"></param>

/// <param name="v"></param>

public void updateBar( float value,int v)

{

barValue.text = ""+v; //更新bar的值

int d;

d = (int)(value \* height);

//d = height / 2;

//image.sprite = bar;

int w = width;

int h = height - d;

//Color[] transparent = new Color[w \* h];

for (int x = 0; x < width; x++)

{

for (int y = 0; y < d; y++)

{

if (!isHealth)

{

bar.texture.SetPixel(x, y, orignColor[x + y \* width]);

}

else

{

Color c = bar.texture.GetPixel(x, y);

//transparent[x + (y - d) \* w] = new Color(c.r, c.g, c.b, 0);

bar.texture.SetPixel(x, y, new Color(c.r, c.g, c.b, 0));

}

}

}

for (int x = 0; x < width; x++)

{

for (int y = d; y < height; y++)

{

if (!isHealth)

{

Color c = bar.texture.GetPixel(x, y);

//transparent[x + (y - d) \* w] = new Color(c.r, c.g, c.b, 0);

bar.texture.SetPixel(x, y, new Color(c.r, c.g, c.b, 0));

}

else

{

bar.texture.SetPixel(x, y, orignColor[x + y \* width]);

}

}

}

//bar.texture.SetPixels(0, d, w, h, transparent);

bar.texture.Apply();

// Debug.Log("" + bar.texture.GetPixel(50, 59).a);

}

**6.****数据库控制模块：**

使用sqlite数据库来管理数据

1.数据保存模块：

在每次通过一个关卡时保存数据，需保存装备数据，游戏状态数据（包含关卡状态）

GameObject.FindGameObjectWithTag("hero").GetComponent<Hero>().saveDate();

BagManager.save();

保存流程：

1. 与指定数据库建立连接，
2. 执行更新操作
3. 断开连接

//

string appDBPath = Application.dataPath + "/mainDate.db";

//--------------------------

DbAccess db = new DbAccess(@"Data Source=" + appDBPath);

//path = appDBPath;

db.UpdateInto("status", new string[] { "maxHealth", "maxMagic", "maxGhost", "curHealth", "curMagic",

"curGhost","hero" , "armorValue", "dodgeValue", "hitValue" ,"level"},

new string[] { maxHealthValue.ToString(), maxMagicValue.ToString(), maxGhostValue.ToString(),

healthValue.ToString(), magicValue.ToString(),ghostValue.ToString(),"'"+curHeroName+"'",

armorValue.ToString(),dodgeValue.ToString(),hitValue.ToString(),Manager.level.ToString()},"rowid","1");

db.CloseSqlConnection();

2.数据读取模块：

继续游戏时读取保存的游戏数据，选择角色时也许读取角色信息

读取流程：

1. 与指定数据库建立连接：
2. 执行查询操作，把数据存入SqliteDataReader 中
3. 在SqliteDataReader 中读取所需信息，存入相应变量
4. 断开连接

代码：

string appDBPath = Application.dataPath + "/mainDate.db";

//--------------------------

DbAccess db = new DbAccess("Data Source=" + appDBPath);

//读取当前状态或初始状态

using (SqliteDataReader sqReader = db.ExecuteQuery("SELECT \* FROM initStatus WHERE hero='" + name+"'"))

{

while (sqReader.Read())

{

//if(sqReader.GetString(sqReader.GetOrdinal("hero"))==name) //start

maxHealthValue = sqReader.GetInt32(sqReader.GetOrdinal("maxHealth"));

maxMagicValue = sqReader.GetInt32(sqReader.GetOrdinal("maxMagic"));

maxGhostValue = sqReader.GetInt32(sqReader.GetOrdinal("maxGhost"));

healthValue = sqReader.GetInt32(sqReader.GetOrdinal("curHealth"));

magicValue = sqReader.GetInt32(sqReader.GetOrdinal("curMagic"));

ghostValue = sqReader.GetInt32(sqReader.GetOrdinal("curGhost"));

armorValue = sqReader.GetInt32(sqReader.GetOrdinal("armorValue"));

dodgeValue = sqReader.GetInt32(sqReader.GetOrdinal("dodgeValue"));

hitValue = sqReader.GetInt32(sqReader.GetOrdinal("hitValue"));

hName = sqReader.GetString((sqReader.GetOrdinal("hero")));

//name = sqReader.GetString(sqReader.GetOrdinal("hero"));

}

}

db.CloseSqlConnection();

**六．****界面设计：**

**主菜单 ：**

**开始：重新游戏**

**继续：继续存档的游戏**

**退出：结束游戏**



**初始选择界面：**

**左上角角色选择：**



**角色选择：**



**探索地宫界面：**

点击去除迷雾，攻击怪物击中或为击中都有动画







**背包界面：**

**属性面板可滚动查看**



**关卡间动画，关开门：**





**暂停选择界面：**

按esc键跳出



**失败，胜利界面：**



**Boss关界面：**



**七．****总结：**

一开始以为符石守护者模仿起来应该不是太难，但当实际去做的时候才发现有许多细节，做起来并不如想象的快，最终只实现了其中的一部分功能，不过也算是一个完整的游戏了。