1. **总体设计**
2. **基本技术路线**

运用面向对象的设计理念，设计了病人类，牙医类，候诊椅类，诊疗椅类等，通过对这些类的封装，创立各个类的对象，并调用类的成员函数。

1. **总体结构**

软件总体结构如图3-1所示，包括：界面设计、算法设计、结果显示等。

界面设计方面包括使用Unity3D进行场景地图布置以及对病人和医生等人物建模，设置人物属性，设置地图属性等。

算法设计方面主要包括创建多线程进行多道缓冲区协同操作，利用信号量机制进行对临界资源的互斥同步访问。

结果显示方面主要包括界面实时参数显示和对实时数据进行保存。

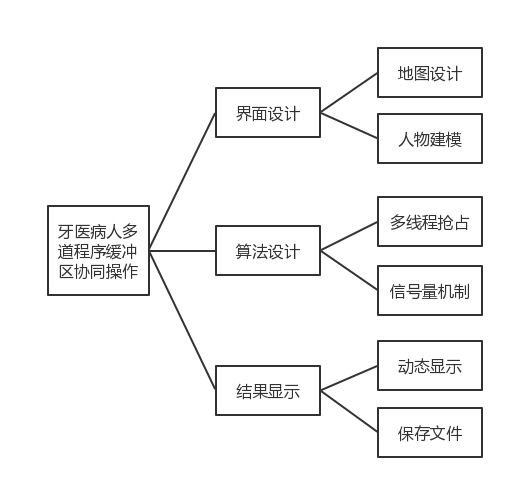
****

图3-1 软件总体结构图

**Unity的Update主线程**

Loom类（子线程对UI的操作放到主线程的处理队列中）

**主控类（maincontrol）**

候诊椅(chair)，病人(patient)，治疗椅各一个类，包含以下几个属性：

两种椅子被占有的bool,病人主动占有椅子的bool

候诊椅有一个列表，包含了未被占有的椅子的GameObject对象，

人物有一个列表，包含了占有椅子的人物的GameObject对象，

线程数组，

计时变量，

多种mutex，lock，以及一个对诊断椅子的semaphore的专用信号量。

可重用的读写文件类

1. **模块功能及关系**

主要分为病人线程模块、临界资源抢占模块、数据处理模块。

模块之间关系为临界资源抢占模块调用病人线程，线程模块用于临界资源的抢占模块，而数据处理模块是基于其他两个模块之上的。总的来说，各个模块之间互相影响，相辅相成。

具体模块见图3-2。

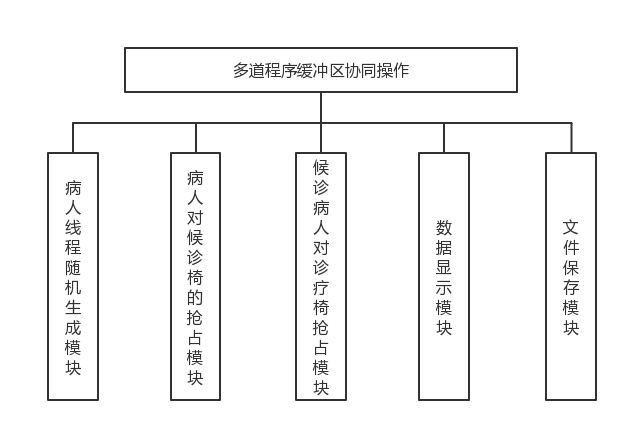
****

图3-2 功能模块图

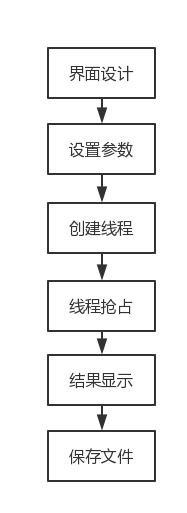
1. **总体流程**

图3-3 总体流程

1. **线程创立**

**在Update里面调用loom类的函数可以保留子线程的操作到的消息处理队列中，然后主线程对队列的内容进行处理并且不会中断子线程的状态。**

1. Loom.RunAsync(() =>//子线程在里面写可以保持子线程状态
2. {
3. if (allbeoccupied == true)//如果队列小于5 证明有一个座位被占了
4. {
5. //Debug.Log(onchairpersons.Count);
6. //抢占开始
7. for (int i = 0; i < onchairpersons.Count; i++)
8. {
9. th[i] = new Thread(Refresh2);//线程创建
10. th[i].Start(i); //线程创建
11. if (timer1 > time1)
12. {
14. timer1 = 0;//计时器清零
15. }
16. }
18. }
19. });
20. public void Refresh2(object o2)
21. {
22. temp = (int)o2;
23. Loom.QueueOnMainThread((parama) =>//操作放到主线程队列，对Action进行加锁回调，所以可以返回到在主线程操作UI
24. {
25. se.WaitOne();//信号量P操作
27. //Debug.Log(temp + "抢占成功" + Time.deltaTime);
28. //Debug.Log(onchairpersons.Count + "shuliang");
29. if (onchairpersons.Count > 0)//onchairpersons.Count >-1&&
30. {
31. g = onchairpersons[0];
32. //onchairpersons.RemoveAt(0);
33. if (onchairpersons[0].GetComponent<person>().gettwo == false)//没走到workchair就走
34. onchairpersons[0].GetComponent<person>().walk2();
35. if (onchairpersons[0].transform.position == GameObject.Find("work\_chair").transform.position)
36. {
37. onchairpersons[0].GetComponent<person>().gettwo = true;
39. Debug.Log("到达了椅子");
40. //g = onchairpersons[0];
41. }
42. if (onchairpersons[0].GetComponent<person>().gettwo == true && onchairpersons[0].GetComponent<person>().getthreed == false)//向死亡点走路之前的判断
43. {
44. sleepcount += speed;
45. //Debug.LogError(sleepcount);
46. GameObject.Find("Canvas").GetComponent<Power>().mutex = true;
47. if (sleepcount >100)
48. {
49. onchairpersons[0].GetComponent<person>().walk3();
50. if (onchairpersons[0].transform.position == GameObject.Find("death").transform.position)
51. {
52. Debug.Log("到达了终点");
53. onchairpersons[0].GetComponent<person>().getthreed = true;
54. havedPer--;
55. ch.Add(onchairs[0]);//空椅子++
56. onchairpersons.RemoveAt(0);
57. onchairs.RemoveAt(0);
58. per.RemoveAt(0);
59. sleepcount = 0;
61. }
62. }
63. }
64. }
65. else
66. th[temp].Abort();//子线程中断
67. se.Release(1);//信号量V操作
68. }, null);
69. **详细设计**
70. **进程操作函数、原语、API**

**进程操作函数：**

Loom.RunAsync(() =>//进程的状态保持进队列

{

}

Loom.QueueOnMainThread((parama) =>//对action进行加锁回调，在主线程的消息队列操作UI

{}）;

th[i] = new Thread(Refresh2);//子线程创建

th[i].Start(i);

在unity中没有对话框，需要调用win32API进行弹出对话框的操作等。

public class Messagebox

{

[DllImport("User32.dll", SetLastError = true, ThrowOnUnmappableChar = true, CharSet = CharSet.Auto)]

public static extern int MessageBox(IntPtr handle, String message, String title, int type);}

1. **模块内部流程和实现算法**

首先是多人对候诊椅的抢占，成功就可以更改状态为，失败继续找座位，人数超过座位数则加到等待队列中。然后再诊断椅状态位为未被使用的时候，所有在椅子上的人对诊断椅子进行抢占，抢占成功的执行走路函数，诊断结束释放后下一组人方可进行抢占。

实现算法：多线程抢占，抢占成功进行步行操作。

Semaphore se = new Semaphore(1,1);//初始允许请求为1，最大请求为1

Loom.QueueOnMainThread((parama) =>

{

se.WaitOne();//信号量P操作

if (onchairpersons.Count > 0)//onchairpersons.Count >-1&&

{

g = onchairpersons[0];

//onchairpersons.RemoveAt(0);

if (onchairpersons[0].GetComponent<person>().gettwo == false)//没走到候诊椅workchair就走

onchairpersons[0].GetComponent<person>().walk2();//走路函数

if (onchairpersons[0].transform.position == GameObject.Find("work\_chair").transform.position)

{

onchairpersons[0].GetComponent<person>().gettwo = true;

Debug.Log("到达了诊断椅");

}

if (onchairpersons[0].GetComponent<person>().gettwo == true && onchairpersons[0].GetComponent<person>().getthreed == false)

{

sleepcount += speed;

GameObject.Find("Canvas").GetComponent<Power>().mutex = true;

if (sleepcount >100)

{

onchairpersons[0].GetComponent<person>().walk3();

if (onchairpersons[0].transform.position == GameObject.Find("death").transform.position)

{

Debug.Log("到达了终点");

onchairpersons[0].GetComponent<person>().getthreed = true;

havedPer--;//人物个数

ch.Add(onchairs[0]);//空椅子++

onchairpersons.RemoveAt(0);

onchairs.RemoveAt(0);

per.RemoveAt(0);

sleepcount = 0;

}

}

}

}

else

th[temp].Abort();

se.Release(1);//信号量V操作

}, null);

1. **编码设计**
2. **开发环境的设置和建立**

Unity3D开发环境，VS2017环境，下载unity中专用Visual Studio Tools for Unity

1. **程序设计时要注意的事项**

在Unity中，Update是每一帧更新的，所以在这个里面设计的代码会被实时刷新到UI，而且只有这一个入口函数，如果在此写线程代码则会提示：UI操作只能在主线程调用。此时应该着重注意子线程调用主线程UI操作的问题。需要把对UI的操作，放到主线程得到UI消息处理队列中，进而让主线程去调用。

Awake函数是在Start函数之前运行，也就是游戏的初始化的问题。每次重新加载场景或者重新调用awake就可以初始化游戏参数。

1. **关键构件/插件的特点和使用**

Loom在工程上的使用，类似于协程，但是他是在子线程的操作。

但是其与协程的区别在于，Loom里面可以做主线程的工作，但是协程不可以。典型如，事件检测硬件的变化（0到1,1到0），我平常会定义一个静态的标志位，用来传出变化，但是，用Loom的话，不用传出变化，直接可以在里面进行操作。再比如，UI显示和3D物体的形态变化，直接可以loom里面写，不用转到主线程，进行操作。

1. **主要程序的代码设计及注释**

**Maincontrol类：（作用：主控类，包含各种属性状态，比如线程，人的列表，空椅子队列，在椅子上的人的队列，信号量互斥Semaphore，时间计时变量，以及各种用于互斥的，开关的，人物椅子个数数组的变量等等）**

**关键代码：**using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System.Threading;

using UnityEngine.UI;

public class maincontrol : MonoBehaviour {

public int max\_patient=8;

public int min\_patient=3;

public int chair\_number;

public int havedPer=0;//已经存在的人

public Thread[] th=new Thread[9];//测试线程

public int create\_num;

public List<GameObject> per;//人的列表

public GameObject[] chairs;//椅子

public List<GameObject> ch;//空椅子队列

public List<GameObject> ou;//外面排队空队列

public List<GameObject> onchairpersons;//在椅子上的人的队列

public List<GameObject> onchairs;//在椅子上的人的椅子的队列

public bool allbeoccupied = false;

Semaphore se = new Semaphore(1,1);//初始允许请求为1，最大请求为1

Semaphore se2 = new Semaphore(1, 1);//初始允许请求为1，最大请求为1

public GameObject g;//保存中介用来访问三条条路

int mutex = 1;

int ii = 0;

int chansheng=0;//产生

private float timer = 0;//计时变量//等候的人

public int time = 3;//计时频率

private float timer1 = 0;//计时变量//医生

public int time1 = 3;//计时频率

public string thname;//当前线程的名字

public int temp;//线程编号

public float sleepcount = 0;

public float speed;

public bool lock1 = false;

public int waitnumber=0;

public int waitmax=5;//等待人数的最大值

void Awake()

{

chair\_number = GameObject.FindGameObjectsWithTag("chairs").Length;//得到椅子数量

chairs = GameObject.FindGameObjectsWithTag("chairs");

ch.Clear();

for (int i = 0; i < chair\_number; i++)//椅子是否占有队列初始化 chair\_number

{

if (chairs[i].GetComponent<chair>().isOccupied() == false)//没有被占的椅子保存

{

ch.Add(chairs[i]);

}

}

//Debug.Log(chair\_number+"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

void OnEnable()

{

}

void Start()

{

}

//public void lthmdo()//less than min do 小于最小该做的

//{

// create\_num = Random.Range(1, 5);//max\_patient - havedPer

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

// {

// GameObject g = (GameObject)GameObject.Find("Camera").GetComponent<create\_patient>().create();

// per.Add(g);//增加人

// //Debug.Log(per.Count+"拥有的人："+havedPer);

// }

//}

public void lthmdo(int num)//less than min do 小于最小该做的

{

for (int i = 0; i < num; i++)//create\_num

{

GameObject g = (GameObject)GameObject.Find("Camera").GetComponent<create\_patient>().create();

per.Add(g);//增加人

//Debug.Log(per.Count+"拥有的人："+havedPer);

}

}

public void onenable()

{

Awake();

}

void Update()

{

if (lock1 == true)

{

//Debug.Log(chair\_number+"QQQQQQQQQQQQQQQQQQQ");

speed = GameObject.Find("Slider").GetComponent<Slider>().value + 0.1f;

max\_patient = GameObject.Find("Canvas").GetComponent<Power>().bottom1;

waitmax = GameObject.Find("Canvas").GetComponent<Power>().bottom2;

if (ii == 0)

{

lthmdo(5);

}

ii++;

//if (havedPer < 5)

// lthmdo(create\_num);//产生

outdo();

lthmdo(create\_num);//产生

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

{

//Debug.Log(havedPer+"、、、、、"+per.Count);

if (per[i].GetComponent<person>().nothavein == false)

{

for (int j = 0; j < chair\_number; j++)

{

if (per[i].GetComponent<person>().transform.position == GameObject.Find("Camera").GetComponent<maincontrol>().chairs[j].transform.position)

{

GameObject.Find("Camera").GetComponent<maincontrol>().onchairpersons.Add(per[i]);

GameObject.Find("Camera").GetComponent<maincontrol>().onchairs.Add(GameObject.Find("Camera").GetComponent<maincontrol>().chairs[j]);

per[i].GetComponent<person>().isgetchair = true;

per[i].GetComponent<person>().nothavein = true;

}

}

}

}

// Refresh();

Loom.RunAsync(() =>

{

if (allbeoccupied == true)//如果队列小于5 证明有一个座位被占了

{

//Debug.Log(onchairpersons.Count);

//抢占开始

for (int i = 0; i < onchairpersons.Count; i++)

{

th[i] = new Thread(Refresh2);

th[i].Start(i);

//timer1 += Time.deltaTime;

if (timer1 > time1)

{

//th[i].Resume();

timer1 = 0;//计时器清零

}

}

Loom.QueueOnMainThread((parama) =>

{

}, null);

//抢占成功的人开始走路

}

});

}

}

void outdo()

{

if (waitnumber == 0)

{

if (create\_num > ch.Count)

{

waitnumber += create\_num - ch.Count;

create\_num = ch.Count;

}

}

if (waitnumber>0)

{

if (waitnumber > ch.Count)

{

waitnumber -= ch.Count;

if (waitnumber + create\_num > waitmax)

{

waitnumber = waitmax;

create\_num = 0;

}

else

{

waitnumber += create\_num;

create\_num = 0;

}

}

else

{

waitnumber = create\_num - (ch.Count - waitnumber);

create\_num = ch.Count - waitnumber;

if (waitnumber > waitmax)

{

waitnumber = waitmax;

}

}

}

}

void LateUpdate()

{

timer += Time.deltaTime;

if (timer > time)

{

create\_num = Random.Range(0, max\_patient);//max\_patient - havedPer

Debug.Log("门外等候的人" + create\_num);

timer = 0;//计时器清零

Debug.Log("计时1次");

}

}

public void Refresh2(object o2)

{

temp = (int)o2;

//se.WaitOne(1); //Thread.Sleep(30);

Loom.QueueOnMainThread((parama) =>

{

se.WaitOne();

//thname = "线程编号：{" + temp.ToString() + "}";

//th[temp].Join();

//Debug.Log(temp + "抢占成功" + Time.deltaTime);

//Debug.Log(onchairpersons.Count + "shuliang");

if (onchairpersons.Count > 0)//onchairpersons.Count >-1&&

{

g = onchairpersons[0];

//onchairpersons.RemoveAt(0);

if (onchairpersons[0].GetComponent<person>().gettwo == false)//没走到workchair就走

onchairpersons[0].GetComponent<person>().walk2();

if (onchairpersons[0].transform.position == GameObject.Find("work\_chair").transform.position)

{

onchairpersons[0].GetComponent<person>().gettwo = true;

Debug.Log("到达了椅子");

//g = onchairpersons[0];

}

if (onchairpersons[0].GetComponent<person>().gettwo == true && onchairpersons[0].GetComponent<person>().getthreed == false)

{

//Debug.Log("asdasdassssssssssssssssssssssssssssssssssssssssssssss");

sleepcount += speed;

//Debug.LogError(sleepcount);

GameObject.Find("Canvas").GetComponent<Power>().mutex = true;

if (sleepcount >100)

{

onchairpersons[0].GetComponent<person>().walk3();

if (onchairpersons[0].transform.position == GameObject.Find("death").transform.position)

{

Debug.Log("到达了终点");

onchairpersons[0].GetComponent<person>().getthreed = true;

havedPer--;

ch.Add(onchairs[0]);//空椅子++

onchairpersons.RemoveAt(0);

onchairs.RemoveAt(0);

per.RemoveAt(0);

sleepcount = 0;

// Destroy(g, 0);

//th[temp].Abort();

}

}

}

}

else

th[temp].Abort();

se.Release(1);

}, null);

//if (onchairpersons[0].GetComponent<person>().getthreed == true)

//{

// th[temp].Suspend();

//}

}

public void Refresh()

{

Loom.QueueOnMainThread((parama) =>

{

if (mutex == 1)

{

mutex--;

for (int i = 0; i < onchairpersons.Count; i++)

{

if (onchairpersons[i] != null)

{

if (onchairpersons[i].GetComponent<person>().gettwo == false)//没走到workchair就走，走到第三个椅子

onchairpersons[i].GetComponent<person>().walk2();

Debug.Log("开始走");

if (onchairpersons[i].transform.position == GameObject.Find("work\_chair").transform.position)

{

onchairpersons[i].GetComponent<person>().gettwo = true;

Debug.Log("到达了椅子");

//g = onchairpersons[0];

}

if (onchairpersons[i].GetComponent<person>().gettwo == true && onchairpersons[0].GetComponent<person>().getthreed == false)

{

onchairpersons[i].GetComponent<person>().walk3();

if (onchairpersons[i].transform.position == GameObject.Find("death").transform.position)

{

Debug.Log("到达了终点");

//g = onchairpersons[0];

onchairpersons[i].GetComponent<person>().getthreed = true;

g = onchairpersons[i];

//onchairpersons.RemoveAt(i);

onchairpersons[i] = null;

sleepcount = 0;

//Destroy(g, 0);

}

}

}

}

mutex++;

}

else

{

//跳过这个for的一个

}

},null);

}

}

**LOOM类：（作用：将子线程中操作UI的消息，放入到主线程操作队列中，在Unity流程管理中Update方法下检查需要回调的Action进行加锁回调，确保主线程执行，回调序列本身又作为静态数据保存，在任意线程调用添加）**

**关键代码：**

public static void Initialize()

{

if (!initialized)

{

if (!Application.isPlaying)

return;

initialized = true;

var g = new GameObject("Loom");

\_current = g.AddComponent<Loom>();

#if !ARTIST\_BUILD

UnityEngine.Object.DontDestroyOnLoad(g);

#endif

}

}

public struct NoDelayedQueueItem

{

public Action<object> action;

public object param;

}

private List<NoDelayedQueueItem> \_actions = new List<NoDelayedQueueItem>();

public struct DelayedQueueItem

{

public float time;

public Action<object> action;

public object param;

}

private List<DelayedQueueItem> \_delayed = new List<DelayedQueueItem>();

List<DelayedQueueItem> \_currentDelayed = new List<DelayedQueueItem>();

public static void QueueOnMainThread(Action<object> taction, object tparam)

{

QueueOnMainThread(taction, tparam, 0f);

}

public static void QueueOnMainThread(Action<object> taction, object tparam, float time)

{

if (time != 0)

{

lock (Current.\_delayed)

{

Current.\_delayed.Add(new DelayedQueueItem { time = Time.time + time, action = taction, param = tparam });

}

}

else

{

lock (Current.\_actions)

{

Current.\_actions.Add(new NoDelayedQueueItem { action = taction, param = tparam });

}

}

}

public static Thread RunAsync(Action a)

{

Initialize();

while (numThreads >= maxThreads)

{

Thread.Sleep(100);

}

Interlocked.Increment(ref numThreads);

ThreadPool.QueueUserWorkItem(RunAction, a);

return null;

}

private static void RunAction(object action)

{

try

{

((Action)action)();

}

catch

{

}

finally

{

Interlocked.Decrement(ref numThreads);

}

}

**Message类：(作用：调用win32API，进行messagebox输出，因为unity不包含文本框)**

**关键代码：**

public class Messagebox

{

[DllImport("User32.dll", SetLastError = true, ThrowOnUnmappableChar = true, CharSet = CharSet.Auto)]

public static extern int MessageBox(IntPtr handle, String message, String title, int type);

}

**主界面类：（主要是动态按钮的生成，比如：动态椅子生成删除，或者操作按钮生成等等）**

**关键代码：**

public void OnGUI()

{

//just to show how to make a button on GUI - this is hardcoded, not dynamical

if (GUI.Button(new Rect(230, 10, 100, 70), "暂停/恢复"))

{

print("暂停/恢复");

if (Time.timeScale == 0) Time.timeScale = 1; // pause toggle / 1 = 100%, 0 = 0%

else Time.timeScale = 0;

// Time.timeScale = 1;

}

if (GUI.Button(new Rect(340, 10, 100, 70), "保存记录"))

{

DateTime dt = DateTime.Now;

GameObject.Find("Camera").GetComponent<write>().WriteIntoTxt("当前保存时间："+dt);

Messagebox.MessageBox(IntPtr.Zero, "已经保存完毕", "提示", 0);

}

if (GUI.Button(new Rect(10, 10, 100, 70), "开始"))

{

//SceneManager.LoadScene(1);

GameObject.Find("Camera").GetComponent<maincontrol>().lock1 = true;

}

if (GUI.Button(new Rect(120, 10, 100, 70), "结束"))

{

SceneManager.LoadScene(0);

//GameObject.Find("Canvas").GetComponent<maincontrol>().lock1 = true;

}

if (GUI.Button(new Rect(450, 10, 100, 70), "增加椅子"))

{

switch (use)

{

case 0:

GameObject Gg = Instantiate(g1, v3[0], Quaternion.identity);

G.Add(Gg);

Gg.transform.SetParent(GameObject.Find("chairs").transform);

GameObject.Find("Camera").GetComponent<maincontrol>().onenable();

use++;

break;

case 1:

GameObject Gg1 = Instantiate(g1, v3[1], Quaternion.identity);

G.Add(Gg1);

Gg1.transform.SetParent(GameObject.Find("chairs").transform);

GameObject.Find("Camera").GetComponent<maincontrol>().onenable();

use++;

break;

case 2:

GameObject Gg2 = Instantiate(g1, v3[2], Quaternion.identity);

G.Add(Gg2);

Gg2.transform.SetParent(GameObject.Find("chairs").transform);

GameObject.Find("Camera").GetComponent<maincontrol>().onenable();

use++;

break;

case 3:

GameObject Gg3 = Instantiate(g1, v3[3], Quaternion.identity);

G.Add(Gg3);

Gg3.transform.SetParent(GameObject.Find("chairs").transform);

GameObject.Find("Camera").GetComponent<maincontrol>().onenable();

use++;

break;

}

}

if (GUI.Button(new Rect(560, 10, 100, 70), "删除椅子"))

{

switch (G.Count)

{

case 4:

GameObject a = G[0];

Destroy(a,0);

GameObject.Find("Camera").GetComponent<maincontrol>().onenable();

G.RemoveAt(0);

use--;

break;

case 3:

GameObject a1 = G[0];

Destroy(a1, 0);

GameObject.Find("Camera").GetComponent<maincontrol>().onenable();

G.RemoveAt(0);

use--;

break;

case 2:

GameObject a2 = G[0];

Destroy(a2, 0);

GameObject.Find("Camera").GetComponent<maincontrol>().onenable();

G.RemoveAt(0);

use--;

break;

case 1:

Debug.Log("adasd");

GameObject a3 = G[0];

DestroyImmediate(a3);

GameObject.Find("Camera").GetComponent<maincontrol>().onenable();

G.RemoveAt(0);

use--;

break;

}

}

}

**Create\_patient类：（作用：通过应用预设体，动态的生成人物）**

**关键代码：**

public Object create()//创建人物

{

Obj = null;//防止引用上一个

prefabs = Resources.LoadAll("person") as Object[];

swpan = GameObject.FindGameObjectsWithTag("create");

int tempa, tempb;

tempa = Random.Range(0, 5);

tempb = Random.Range(0, 1);//1个出生点

Vector3 vec = new Vector3(0,0.5f,0);

Obj =Instantiate(prefabs[tempa],swpan[tempb].transform.position,Quaternion.identity);

if (Obj != null)

{

GameObject.Find("Camera").GetComponent<maincontrol>().havedPer += 1;

}

return Obj;

}

**Person类：（作用：包含三种方式的步行函数（走向候诊椅，走向诊断椅），自身是否拥有椅子，自身速度，和疼痛条等属性）**

**关键代码：**

void Awake()

{

gos[0] = GameObject.Find("fir\_obj").transform.position;

gos[1] = GameObject.Find("sec\_obj").transform.position;

if (GameObject.Find("Camera").GetComponent<maincontrol>().ch.Count != 0)

{

gos[2] = GameObject.Find("Camera").GetComponent<maincontrol>().ch[0].transform.position;

GameObject.Find("Camera").GetComponent<maincontrol>().ch.RemoveAt(0);

}

bloodbar = Instantiate(Resources.Load("Slider/Slider"),this.transform.position, Quaternion.identity) as GameObject; //生成预设体

bloodbar.GetComponent<Slider>().fillRect.transform.GetComponent<Image>().color = new Color(0, 255, 0, 255);

}

// Use this for initialization

void Start () {

bloodbar.transform.SetParent(GameObject.Find("Canvas").transform);

speedd = Random.Range(0.1f,0.5f);

}

public float count=0;

float speedd;

// Update is called once per frame

void Update () {

if (count < 500)

{

count += speedd;

bloodbar.GetComponent<Slider>().value = count / 100.0f;

}

if (isgetchair == false)//是否得到过第一个椅子

{

walk();

}

bloodbar.transform.position = transform.position + new Vector3(0, 0.5f, 0);

}

public void walk()

{

Vector3 v = (gos[i] - transform.position).normalized;//向量

transform.right = v;//朝向目标点

des = Vector3.Distance(this.transform.position, gos[i]);

transform.localPosition = Vector3.MoveTowards(this.transform.position, gos[i], Time.deltaTime \* movespeed);

if (this.transform.position == gos[2])//gos[2]是座位

{

this.GetComponent<Rigidbody2D>().isKinematic = true;//刚体运动学防抖动

}

if (des < 1f && i < 2)

{

i++;

}

}

public void walk2()

{

Vector3 v = (GameObject.Find("work\_chair").transform.position - transform.position).normalized;//向量

transform.right = v;//朝向目标点

des = Vector3.Distance(this.transform.position, GameObject.Find("work\_chair").transform.position);

//Debug.Log(this.transform.position+"+++++++++++++"+ GameObject.Find("work\_chair").transform.position);

transform.localPosition = Vector3.MoveTowards(this.transform.position, GameObject.Find("work\_chair").transform.position, Time.deltaTime \* movespeed);

}

public void walk3()

{

Vector3 v = (GameObject.Find("death").transform.position - transform.position).normalized;//向量

transform.right = v;//朝向目标点

des = Vector3.Distance(this.transform.position, GameObject.Find("death").transform.position);

transform.localPosition = Vector3.MoveTowards(this.transform.position, GameObject.Find("death").transform.position, Time.deltaTime \* movespeed);

}

}

**Chair类：（作用：是否被占有等属性）**

**关键代码：**

public bool occupied = false;

public GameObject[] person= {null};

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

this.occupied = isOccupied();//每次都判断当前被占用情况

}

public bool freeChair()

{

return false;

}

public void occupyChair(GameObject patient)

{

}

public bool isOccupied()//椅子被占，人和椅子的状态都改变

{

person = GameObject.FindGameObjectsWithTag("Player");

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

{

if (this.transform.position == person[i].transform.position)

{

person[i].GetComponent<person>().isgetchair = true;

GameObject.Find("Camera").GetComponent<maincontrol>().allbeoccupied = true;//被占一个就改变状态

return true;

}

}

return false;

}

**Text类：（作用：输出到界面变化的值）**

**关键代码：**

private Text uiText;

private Text uiText1;

private Text uiText2;

//储存中间值

private string words;

private string words2;

private float timer;

//限制条件，是否可以进行文本的输出

// Use this for initialization

void Start () {

uiText = GameObject.Find("Text").GetComponent<Text>();

uiText1 = GameObject.Find("Text (1)").GetComponent<Text>();

uiText2 = GameObject.Find("Text (2)").GetComponent<Text>();

}

// Update is called once per frame

void Update () {

words = "当前病人数量："+GameObject.Find("Camera").GetComponent<maincontrol>().havedPer+"\n"+ "候诊椅的剩余情况：" + (GameObject.Find("Camera").GetComponent<maincontrol>().ch.Count) +"\n"+ "候诊椅的使用情况：" + (GameObject.Find("Camera").GetComponent<maincontrol>().chair\_number - GameObject.Find("Camera").GetComponent<maincontrol>().ch.Count) + "\n" +

"当前运行线程编号：" + GameObject.Find("Camera").GetComponent<maincontrol>().temp + "\n" + "阻塞线程个数：" + (GameObject.Find("Camera").GetComponent<maincontrol>().onchairpersons.Count)+"\n"+"当前外面等待人数："+ GameObject.Find("Camera").GetComponent<maincontrol>().waitnumber;

if(words!=words2)

GameObject.Find("Camera").GetComponent<write>().WriteIntoTxt(words+"游戏时间"+Time.deltaTime);

uiText1.text = "随机生成的最大人数：" + GameObject.Find("Canvas").GetComponent<Power>().bottom1;

uiText2.text = "室外等待最大人数：" + GameObject.Find("Canvas").GetComponent<Power>().bottom2;

uiText.text = words;

words2 = words;

}

**Write类：（作用：写文件的作用）**

**关键代码：**

public void WriteIntoTxt(string message)

{

FileInfo file = new FileInfo(Application.dataPath + "/mytxt.txt");

if (!file.Exists)

{

writer = file.CreateText();

}

else

{

writer = file.AppendText();

}

writer.WriteLine(message);

writer.Flush();

writer.Dispose();

writer.Close();

}

**Power类：（作用：包含各种进度条(人物头上的疼痛长条，以及牙医身上的工作圆条)）**

**关键代码：**

public float speed;

public float count;

public bool mutex = false;//不用圆盘

public int bottom1;

public int bottom2;//底下的第一条，第二条

void Awake()

{

// gg.GetComponent<Slider>().fillRect.transform.GetComponent<Image>().color = new Color(255, 0, 0, 255);

//gg.SetParent(GameObject.Find("Canvas").transform);

}

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

bottom1 = (int)(GameObject.Find("Slider (1)").GetComponent<Slider>().value \* 10);

bottom2 = (int)(GameObject.Find("Slider (2)").GetComponent<Slider>().value \* 5);

Debug.Log(bottom1+"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

//speed= GameObject.Find("Slider").GetComponent<Slider>().value + 0.1f;

count = GameObject.Find("Camera").GetComponent<maincontrol>().sleepcount;

if (mutex == true)

{

if (count <= 100)

{

//count += speed;

//Debug.Log(count);

GameObject.Find("Image").GetComponent<Image>().fillAmount = count / 100.0f;

GameObject.Find("Text (3)").GetComponent<Text>().text = ((int)count).ToString() + "%";

}

if (count >= 99)

{

GameObject.Find("Image").GetComponent<Image>().color = new Color(255, 0, 0, 255);

}

if (count > 100)

{

//count = 0;

GameObject.Find("Image").GetComponent<Image>().color = new Color(0, 255, 0, 255);

GameObject.Find("Text (3)").GetComponent<Text>().text = "0%";

GameObject.Find("Image").GetComponent<Image>().fillAmount = 0;

mutex = false;

}

}

1. **测试时出现的问题及其解决方法**

**（1） Unity本不支持多线程**

因为unity制作的游戏是以单线程和协程的方式，以时间动画帧的形式更新运行，所以在UI采用多线程是不支持的，但是即使需要多线程的，也只是在自动寻路，比如 Star A算法中，或者需要后台网络通信的，比如需要下载图片，下载文件等等的时候需要。更因为在unity中每一帧都要调用update，他会在上一帧没解决的问题会被覆盖中断，然后下一个update里面会重新执行，对多线程状态的保存和更新会受到极大的困扰，在Unity的update里面加UI的多线程无疑使难上加难

可以在多线程运行的内容分别是

1. 变量(都能指向相同的内存地址)都是共享的

2. 不是UnityEngine的API能在分线程运行

3. UnityEngine定义的基本结构(int,float,Struct定义的数据类型)可以在分线程计算，如 Vector3(Struct)可以 ， 但Texture2d(class,根父类为Object)不可以。

4. UnityEngine定义的基本类型的函数可以在分线程运行

但是如何在子线程调用主线程UI呢？就需要引用loom类对UI操作的保存处理。

经常犯的错误无疑使对各种共享数据的处理，此时应该加入信号量和各种状态位的判断，防止其他操作在此操作未使用完之前更改信息。

**（2）子线程无法调用主线程的UI并且在Update里面会更改这一帧子线程的状态**

解决方法：

我们只需要引用loom类的两个函数：RunAsync(Action)和QueueOnMainThread(Action, [optional] float time) 就可以轻松实现一个函数的两段代码在C#线程和Unity的主线程中交叉运行。原理也很简单：用线程池去运行RunAsync(Action)的函数，在Update中运行QueueOnMainThread(Acition, [optional] float time)传入的函数。将子线程处理UI的操作放入主线程处理操作的队列中，并且保持此子线程的状态不被破坏而且还能实时刷新，update不受影响。就可以解决无法调用主线程UI的问题。

**（3） 动态的增加椅子之后，全局初始化问题**

解决awake函数在其他类无法调用的问题，可以在此类封装好awake函数，然后在其他函数中调用即可。

**（4）人物拥挤椅子的问题**

解决方法：给人物加上碰撞体刚体属性，采用内插值的方法解决人物碰撞拥挤和碰撞后抖动问题。

1. **软件使用说明**
2. **基本功能**

* 提供良好图形界面，显示整个系统操作过程，可以暂停和继续系统的执行；
* 可以设定候诊椅容量；
* 可以设定Patient 到达的数度和Dentist 治疗一名患者的最小时间；
* 实时显示候诊椅的使用情况量、空闲空间的数量、室外Patient 的数量、已经治疗的Patient的数量；
* 实时显示线程、进程所处于等待（阻塞）状态的个数；
* 程序运行结束，显示汇总数据：

总的运行时间；处理Patient 的个数；平均候诊椅中的Patient 数量，平均室外Patient 的数量，已经治疗的Patient 的数量；

* 能够将每次的实验输入和实验结果存储起来，随时可查询。

1. **需要运行的环境**

Windows7及以上版本

1. **安装**

Unity 3D支持直接生成可执行文件，无需安装。

1. **运行**



图7-1 开始界面



图7-2 运行界面

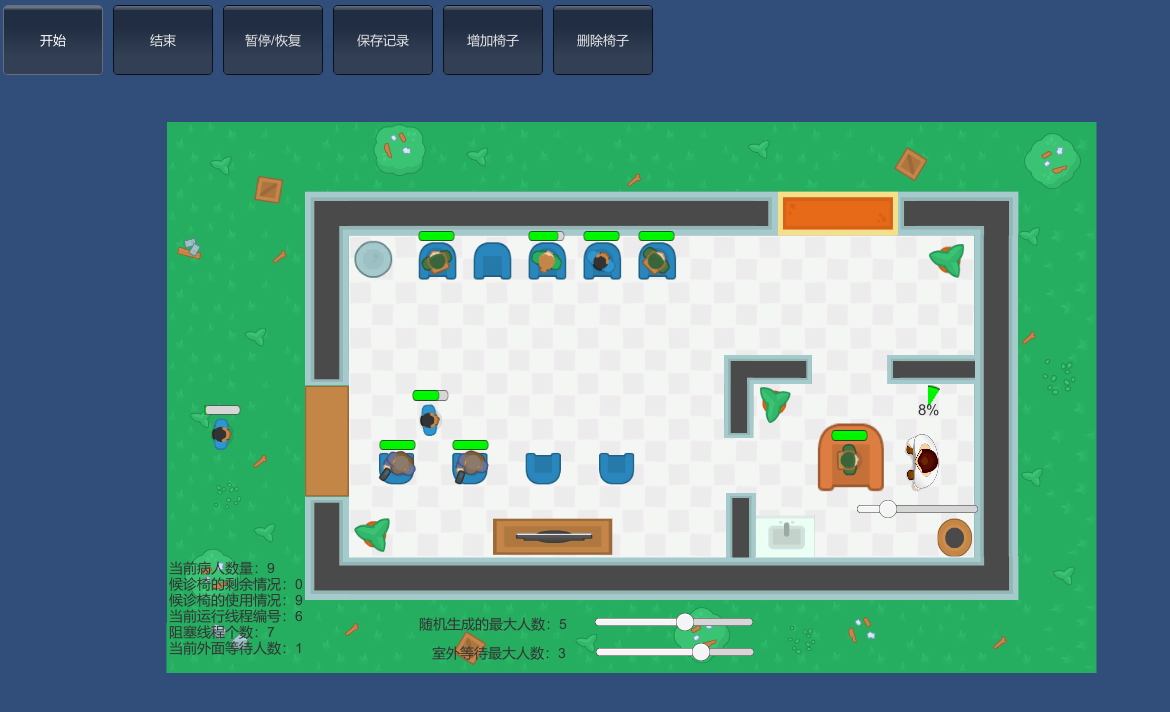


图7-3 增加椅子



图7-4 记录文件保存

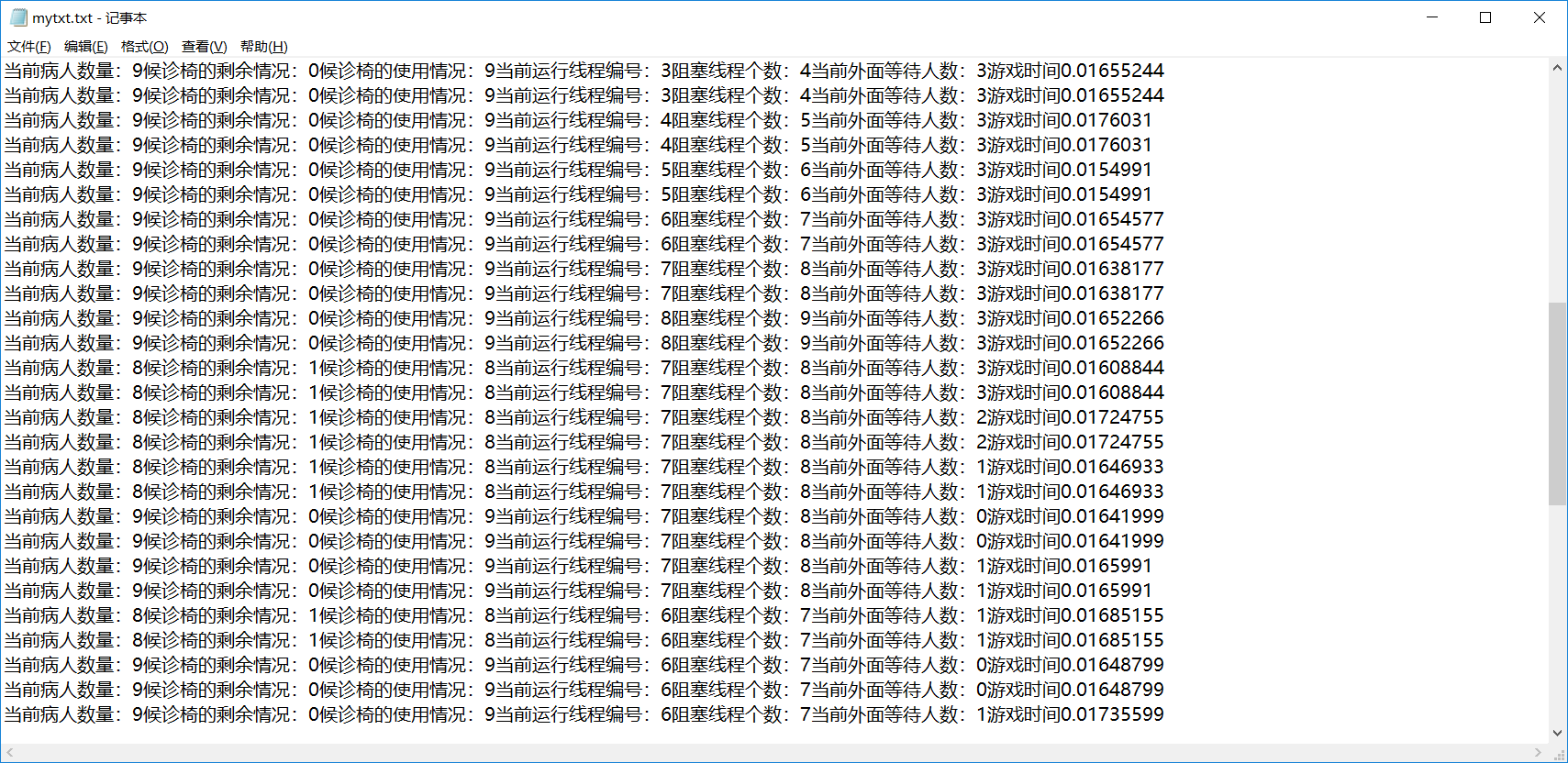


图7-5 文件保存结果

1. **操作**
   1. 拖动底下滚动条设置初始参数
   2. 点击开始按钮开始运行
   3. 点击结束按钮结束当前页面并重新加载
   4. 点击暂停/恢复按钮暂停及恢复当前页面
   5. 点击增加椅子可以增加候诊椅数量
   6. 点击删除椅子可以删除候诊椅数量
   7. 点击保存记录可以将实时数据保存成txt文件
2. **总结**
3. **完成的部分**

* 提供良好图形界面，显示整个系统操作过程，可以暂停和继续系统的执行；
* 可以设定候诊椅容量；
* 可以设定Patient 到达的数度和Dentist 治疗一名患者的最小时间；
* 实时显示候诊椅的使用情况量、空闲空间的数量、室外Patient 的数量、已经治疗的Patient的数量；
* 实时显示线程、进程所处于等待（阻塞）状态的个数；
* 程序运行结束，显示汇总数据：

总的运行时间；处理Patient 的个数；平均候诊椅中的Patient 数量，平均室外Patient 的数量，已经治疗的Patient 的数量；

* 能够将每次的实验输入和实验结果存储起来，随时可查询。

1. **创新功能**
   1. 交互较C#窗体程序更好
   2. 实时显示人物动态运动情况
   3. 可以设置各种初始参数
   4. 设定病人忍耐度机制