# 3D游戏设计读书笔记七

## 智能巡逻兵

提交要求：

游戏设计要求：

创建一个地图和若干巡逻兵(使用动画)；

每个巡逻兵走一个3~5个边的凸多边型，位置数据是相对地址。即每次确定下一个目标位置，用自己当前位置为原点计算；

巡逻兵碰撞到障碍物，则会自动选下一个点为目标；

巡逻兵在设定范围内感知到玩家，会自动追击玩家；

失去玩家目标后，继续巡逻；

计分：玩家每次甩掉一个巡逻兵计一分，与巡逻兵碰撞游戏结束；

程序设计要求：

必须使用订阅与发布模式传消息

subject：OnLostGoal

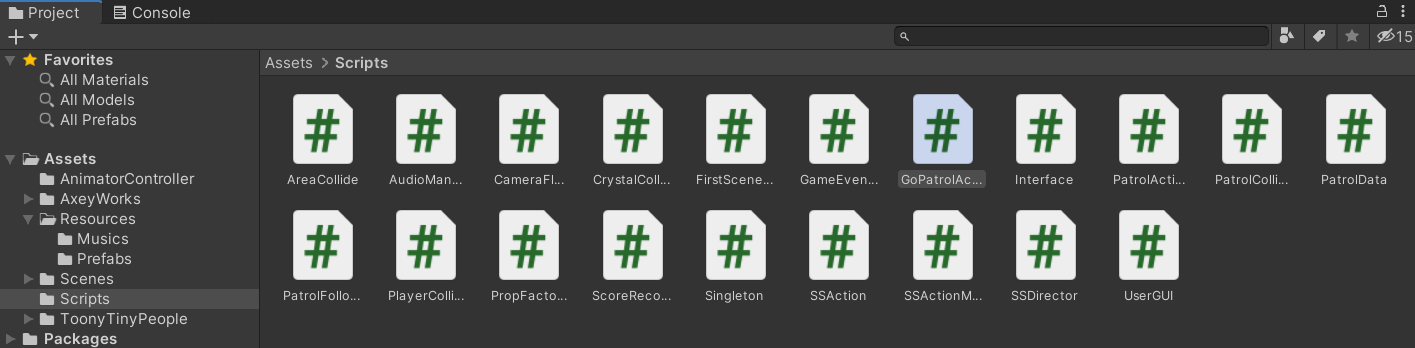
Publisher: ?

Subscriber: ?

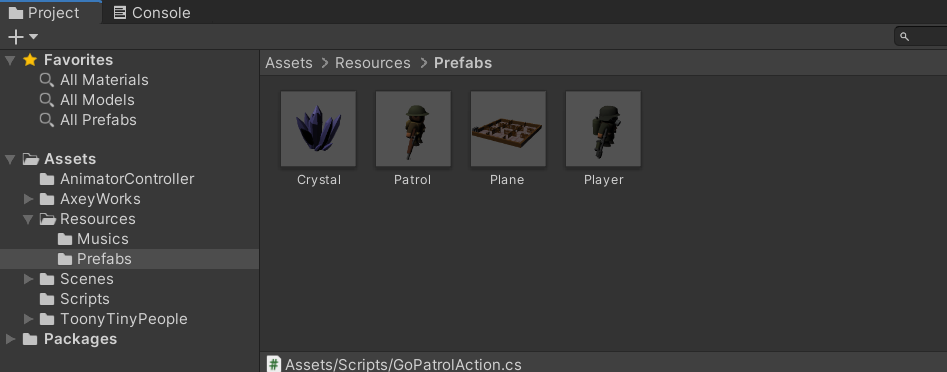
工厂模式生产巡逻兵

## 设计与实现

代码：



模型预置：



对巡逻兵的要求是有多个，且每个巡逻兵走一个3~5个边的凸多边型。同时，巡逻兵需要规避障碍物，并且感知玩家以自动追击。我们需要存储巡逻兵（Patrol）的自身位置及玩家（Player）的位置，以确认是否追击玩家。

按作业要求，我们使用工厂模式来生产巡逻兵（Patrol）。

Patrol有如下属性：

#### PatrolData：

```javascript

public class PatrolData : MonoBehaviour

{

public int sign; //标志巡逻兵在哪一块区域

public bool follow\_player = false; //是否跟随玩家

public int wall\_sign = -1; //当前玩家所在区域标志

public GameObject player; //玩家游戏对象

public Vector3 start\_position; //当前巡逻兵初始位置

}

```

使用工厂模式设置每个巡逻兵（Patrol）的初始位置。游戏中共有9个区域，每个区域各有一个巡逻兵，我们使用sign将这9个区域分别编号，生成时设置对应的sign。

#### PropFactory：

```javascript

public List<GameObject> GetPatrols()

{

int[] pos\_x = { -6, 4, 13 };

int[] pos\_z = { -4, 6, -13 };

int index = 0;

//生成不同的巡逻兵初始位置

for(int i=0;i < 3;i++)

{

for(int j=0;j < 3;j++)

{

vec[index] = new Vector3(pos\_x[i], 0, pos\_z[j]);

index++;

}

}

for(int i=0; i < 9; i++)

{

patrol = Instantiate(Resources.Load<GameObject>("Prefabs/Patrol"));

patrol.transform.position = vec[i];

patrol.GetComponent<PatrolData>().sign = i + 1;

patrol.GetComponent<PatrolData>().start\_position = vec[i];

used.Add(patrol);

}

return used;

}

```

巡逻兵的动作有巡逻（即走一个3~5个边的凸多边型）和追击玩家。我们使用move\_sign判断是否达到目的地，并且设置EAST、WEST、SOUTH、NORTH四个移动方向，如果距离目的地小于0.9则转换方向，理论上设置每次移动距离在一定范围内随机。

#### GoPatrolAction：

设置方向：

```javascript

void Gopatrol()

{

if (move\_sign)

{

//不需要转向则设定一个目的地，按照矩形移动

switch (dirction)

{

case Dirction.EAST:

pos\_x -= move\_length;

break;

case Dirction.NORTH:

pos\_z += move\_length;

break;

case Dirction.WEST:

pos\_x += move\_length;

break;

case Dirction.SOUTH:

pos\_z -= move\_length;

break;

}

move\_sign = false;

}

this.transform.LookAt(new Vector3(pos\_x, 0, pos\_z));

float distance = Vector3.Distance(transform.position, new Vector3(pos\_x, 0, pos\_z));

//当前位置与目的地距离浮点数的比较

if (distance > 0.9)

{

transform.position = Vector3.MoveTowards(this.transform.position, new Vector3(pos\_x, 0, pos\_z), move\_speed \* Time.deltaTime);

}

else

{

dirction = dirction + 1;

if(dirction > Dirction.SOUTH)

{

dirction = Dirction.EAST;

}

move\_sign = true;

}

}

```

调用：

```javascript

public class GoPatrolAction : SSAction

{

private enum Dirction { EAST, NORTH, WEST, SOUTH };

private float pos\_x, pos\_z; //移动前的初始x和z方向坐标

private float move\_length; //移动的长度

private float move\_speed = 1.2f; //移动速度

private bool move\_sign = true; //是否到达目的地

private Dirction dirction = Dirction.EAST; //移动的方向

private PatrolData data; //侦察兵的数据

private GoPatrolAction() { }

public static GoPatrolAction GetSSAction(Vector3 location)

{

GoPatrolAction action = CreateInstance<GoPatrolAction>();

action.pos\_x = location.x;

action.pos\_z = location.z;

//设定移动矩形的边长

action.move\_length = Random.Range(4, 7);

return action;

}

public override void Update()

{

//防止碰撞发生后的旋转

if (transform.localEulerAngles.x != 0 || transform.localEulerAngles.z != 0)

{

transform.localEulerAngles = new Vector3(0, transform.localEulerAngles.y, 0);

}

if (transform.position.y != 0)

{

transform.position = new Vector3(transform.position.x, 0, transform.position.z);

}

//侦察移动

Gopatrol();

//如果侦察兵需要跟随玩家并且玩家就在侦察兵所在的区域，侦查动作结束

if (data.follow\_player && data.wall\_sign == data.sign)

{

this.destroy = true;

this.callback.SSActionEvent(this,0,this.gameobject);

}

}

public override void Start()

{

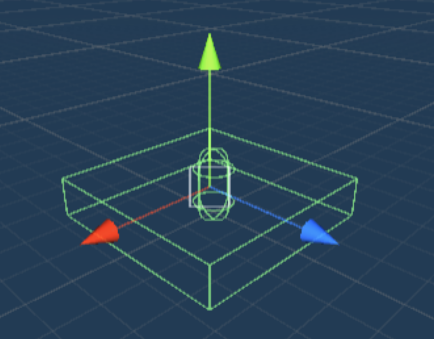
this.gameobject.GetComponent<Animator>().SetBool("run", true);

data = this.gameobject.GetComponent<PatrolData>();

}

```

给巡逻兵（Patrol）添加两个Collider，一个为Capsule Collider，用于检测巡逻兵是否与玩家产生碰撞，然后再添加一个Box Collider，用于检测玩家是否进入Patrol的跟随范围。示意图如下：



#### PatrolFollowAction

要实现巡逻兵（Patrol）朝玩家位置移动。当玩家（Player）的位置超出上面的Box Collider的检测范围，巡逻兵停止跟随行动。同时我们设置一个参数follow\_player用于确定玩家是否进入了巡逻兵的管辖区域，如果该值为0，巡逻兵也不会朝玩家的位置移动。

开始追击动作时，巡逻动作会结束，同样，结束追击动作时，巡逻动作又重新开始，所以我们需要回调函数。当游戏结束时，所有的动作也都结束，巡逻兵（Patrol）也必须停止运动。

```javascript

public class PatrolFollowAction : SSAction

{

private float speed = 2f; //跟随玩家的速度

private GameObject player; //玩家

private PatrolData data; //侦查兵数据

private PatrolFollowAction() { }

public static PatrolFollowAction GetSSAction(GameObject player)

{

PatrolFollowAction action = CreateInstance<PatrolFollowAction>();

action.player = player;

return action;

}

public override void Update()

{

if (transform.localEulerAngles.x != 0 || transform.localEulerAngles.z != 0)

{

transform.localEulerAngles = new Vector3(0, transform.localEulerAngles.y, 0);

}

if (transform.position.y != 0)

{

transform.position = new Vector3(transform.position.x, 0, transform.position.z);

}

Follow();

//如果侦察兵没有跟随对象，或者需要跟随的玩家不在侦查兵的区域内

if (!data.follow\_player || data.wall\_sign != data.sign)

{

this.destroy = true;

this.callback.SSActionEvent(this,1,this.gameobject);

}

}

public override void Start()

{

data = this.gameobject.GetComponent<PatrolData>();

}

void Follow()

{

transform.position = Vector3.MoveTowards(this.transform.position, player.transform.position, speed \* Time.deltaTime);

this.transform.LookAt(player.transform.position);

}

}

```

场景控制器调用PatrolActionManager让Patrol开始巡逻，结束时调用它停止巡逻。

```javascript

public class PatrolActionManager : SSActionManager

{

private GoPatrolAction go\_patrol; //巡逻兵巡逻

public void GoPatrol(GameObject patrol)

{

go\_patrol = GoPatrolAction.GetSSAction(patrol.transform.position);

this.RunAction(patrol, go\_patrol, this);

}

//停止所有动作

public void DestroyAllAction()

{

DestroyAll();

}

}

```

#### UserGUI

在GUI中获取方向键输入，改变Player的移动。

```javascript

void Update()

{

//获取方向键的偏移量

float translationX = Input.GetAxis("Horizontal");

float translationZ = Input.GetAxis("Vertical");

//移动玩家

action.MovePlayer(translationX, translationZ);

}

```

#### FirstSceneController：

处理键盘操作：

```javascript

//玩家移动

public void MovePlayer(float translationX, float translationZ)

{

if(!game\_over)

{

if (translationX != 0 || translationZ != 0)

{

player.GetComponent<Animator>().SetBool("run", true);

}

else

{

player.GetComponent<Animator>().SetBool("run", false);

}

//移动和旋转

player.transform.Translate(0, 0, translationZ \* player\_speed \* Time.deltaTime);

player.transform.Rotate(0, translationX \* rotate\_speed \* Time.deltaTime, 0);

//防止碰撞带来的移动

if (player.transform.localEulerAngles.x != 0 || player.transform.localEulerAngles.z != 0)

{

player.transform.localEulerAngles = new Vector3(0, player.transform.localEulerAngles.y, 0);

}

if (player.transform.position.y != 0)

{

player.transform.position = new Vector3(player.transform.position.x, 0, player.transform.position.z);

}

}

}

```

#### AreaCollide

在九个区域中各设置一个Box Collider，当Player进入区域时，修改玩家（Player）的sign值，使巡逻兵（Patrol）确定玩家在哪个区域。

```javascript

public class AreaCollide : MonoBehaviour

{

public int sign = 0;

FirstSceneController sceneController;

private void Start()

{

sceneController = SSDirector.GetInstance().CurrentScenceController as FirstSceneController;

}

void OnTriggerEnter(Collider collider)

{

//标记玩家进入自己的区域

if (collider.gameObject.tag == "Player")

{

sceneController.wall\_sign = sign;

}

}

}

```

### 订阅与发布模式

订阅与发布模式就像订阅报纸，报纸会不定期推送的邮箱，当你想看的时候，打开邮箱就可以看。

定义一个发布事件的类。

#### GameEventManager

```javascript

public class GameEventManager : MonoBehaviour

{

//分数变化

public delegate void ScoreEvent();

public static event ScoreEvent ScoreChange;

//游戏结束变化

public delegate void GameoverEvent();

public static event GameoverEvent GameoverChange;

//水晶数量变化

public delegate void CrystalEvent();

public static event CrystalEvent CrystalChange;

//玩家逃脱

public void PlayerEscape()

{

if (ScoreChange != null)

{

ScoreChange();

}

}

//玩家被捕

public void PlayerGameover()

{

if (GameoverChange != null)

{

GameoverChange();

}

}

//减少水晶数量

public void ReduceCrystalNum()

{

if (CrystalChange != null)

{

CrystalChange();

}

}

}

```

#### CrystalCollide

订阅者为场景控制器，发布者为发生的各个事件。

```javascript

public class CrystalCollide : MonoBehaviour

{

void OnTriggerEnter(Collider collider)

{

if (collider.gameObject.tag == "Player" && this.gameObject.activeSelf)

{

this.gameObject.SetActive(false);

//减少水晶数量

Singleton<GameEventManager>.Instance.ReduceCrystalNum();

}

}

}

```

#### SSActionEvent

碰撞水晶后得分。

```javascript

public class SSAction : ScriptableObject

{

public bool enable = true; //是否正在进行此动作

public bool destroy = false; //是否需要被销毁

public GameObject gameobject; //动作对象

public Transform transform; //动作对象的transform

public ISSActionCallback callback; //动作完成后的消息通知者

protected SSAction() { }

//子类可以使用下面这两个函数

public virtual void Start()

{

throw new System.NotImplementedException();

}

public virtual void Update()

{

throw new System.NotImplementedException();

}

}

```

#### PlayerCollide

逃脱后发布。

```javascript

public class PlayerCollide : MonoBehaviour

{

void OnCollisionEnter(Collision other)

{

//当玩家与侦察兵相撞

if (other.gameObject.tag == "Player")

{

other.gameObject.GetComponent<Animator>().SetTrigger("death");

this.GetComponent<Animator>().SetTrigger("shoot");

Singleton<GameEventManager>.Instance.PlayerGameover();

}

}

}

```

#### ScoreRecorder

记录分数，当玩家（Player）与巡逻兵（Patrol）相撞时游戏结束，计分也结束。

玩家通过获得水晶得分。

```javascript

public class ScoreRecorder : MonoBehaviour

{

public FirstSceneController sceneController;

public int score = 0; //分数

public int crystal\_number = 12; //水晶数量

// Use this for initialization

void Start()

{

sceneController = (FirstSceneController)SSDirector.GetInstance().CurrentScenceController;

sceneController.recorder = this;

}

public int GetScore()

{

return score;

}

public void AddScore()

{

score++;

}

public int GetCrystalNumber()

{

return crystal\_number;

}

public void ReduceCrystal()

{

crystal\_number--;

}

}

```

#### FirstSceneController

```javascript

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class FirstSceneController : MonoBehaviour, IUserAction, ISceneController

{

public PropFactory patrol\_factory; //巡逻者工厂

public ScoreRecorder recorder; //记录员

public PatrolActionManager action\_manager; //运动管理器

public int wall\_sign = -1; //当前玩家所处哪个格子

public GameObject player; //玩家

public Camera main\_camera; //主相机

public float player\_speed = 5; //玩家移动速度

public float rotate\_speed = 135f; //玩家旋转速度

private List<GameObject> patrols; //场景中巡逻者列表

private List<GameObject> crystals; //场景水晶列表

private bool game\_over = false; //游戏结束

void Update()

{

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

{

patrols[i].gameObject.GetComponent<PatrolData>().wall\_sign = wall\_sign;

}

//水晶收集完毕

if(recorder.GetCrystalNumber() == 0)

{

Gameover();

}

}

void Start()

{

SSDirector director = SSDirector.GetInstance();

director.CurrentScenceController = this;

patrol\_factory = Singleton<PropFactory>.Instance;

action\_manager = gameObject.AddComponent<PatrolActionManager>() as PatrolActionManager;

LoadResources();

main\_camera.GetComponent<CameraFlow>().follow = player;

recorder = Singleton<ScoreRecorder>.Instance;

}

public void LoadResources()

{

Instantiate(Resources.Load<GameObject>("Prefabs/Plane"));

player = Instantiate(Resources.Load("Prefabs/Player"), new Vector3(0, 9, 0), Quaternion.identity) as GameObject;

crystals = patrol\_factory.GetCrystal();

patrols = patrol\_factory.GetPatrols();

//所有侦察兵移动

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

{

action\_manager.GoPatrol(patrols[i]);

}

}

//玩家移动

public void MovePlayer(float translationX, float translationZ)

{

if(!game\_over)

{

if (translationX != 0 || translationZ != 0)

{

player.GetComponent<Animator>().SetBool("run", true);

}

else

{

player.GetComponent<Animator>().SetBool("run", false);

}

//移动和旋转

player.transform.Translate(0, 0, translationZ \* player\_speed \* Time.deltaTime);

player.transform.Rotate(0, translationX \* rotate\_speed \* Time.deltaTime, 0);

//防止碰撞带来的移动

if (player.transform.localEulerAngles.x != 0 || player.transform.localEulerAngles.z != 0)

{

player.transform.localEulerAngles = new Vector3(0, player.transform.localEulerAngles.y, 0);

}

if (player.transform.position.y != 0)

{

player.transform.position = new Vector3(player.transform.position.x, 0, player.transform.position.z);

}

}

}

public int GetScore()

{

return recorder.GetScore();

}

public int GetCrystalNumber()

{

return recorder.GetCrystalNumber();

}

public bool GetGameover()

{

return game\_over;

}

public void Restart()

{

SceneManager.LoadScene("Scenes/mySence");

}

void OnEnable()

{

GameEventManager.ScoreChange += AddScore;

GameEventManager.GameoverChange += Gameover;

GameEventManager.CrystalChange += ReduceCrystalNumber;

}

void OnDisable()

{

GameEventManager.ScoreChange -= AddScore;

GameEventManager.GameoverChange -= Gameover;

GameEventManager.CrystalChange -= ReduceCrystalNumber;

}

void ReduceCrystalNumber()

{

recorder.ReduceCrystal();

}

void AddScore()

{

recorder.AddScore();

}

void Gameover()

{

game\_over = true;

patrol\_factory.StopPatrol();

action\_manager.DestroyAllAction();

}

}

```

### 结果展示



更多请访问[github]()