六、模型与动画

1、智能巡逻兵

- **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
- **提示：生成 3~5个边的凸多边型**
  - 随机生成矩形
  - 在矩形每个边上随机找点，可得到 3 - 5 的凸多边型

> 参考博客：https://blog.csdn.net/c486c/article/details/80153548

订阅与发布模式

> Unity官方教程：https://unity3d.com/cn/learn/tutorials/topics/scripting/events?playlist=17117

- 发布者与订阅者没有直接的耦合
- 是MVC模式实现模型与视图分离的重要手段
- 例如：数据DataSource对象，就是Subject。任何使用该数据源的显示控件，如Grid都会及时更新。

```
//EventManager.cs
using UnityEngine;
using System.Collections;
```

```
public class EventManager : MonoBehaviour
{
    public delegate void ClickAction();//利用delegate委托声明事件类型
    public static event ClickAction OnClicked;//定义subject


    void OnGUI()
    {
        if(GUI.Button(new Rect(Screen.width / 2 - 50, 5, 100, 30), "Click"))
        {
            if(OnClicked != null)
                OnClicked();//发出通知，事件由谁处理，如何处理都不需要知道！
        }
    }
}
```

```
//TeleportScript.cs
using UnityEngine;
using System.Collections;

public class TeleportScript : MonoBehaviour
{
    void OnEnable()
    {
        EventManager.OnClicked += Teleport;//注册该事件
    }


    void OnDisable()
    {
        EventManager.OnClicked -= Teleport;//取消该事件
    }


    void Teleport()//回调函数的实现
    {
        Vector3 pos = transform.position;
        pos.y = Random.Range(1.0f, 3.0f);
        transform.position = pos;
    }
}
```

```
//TurnColorScript.cs
using UnityEngine;
using System.Collections;
```

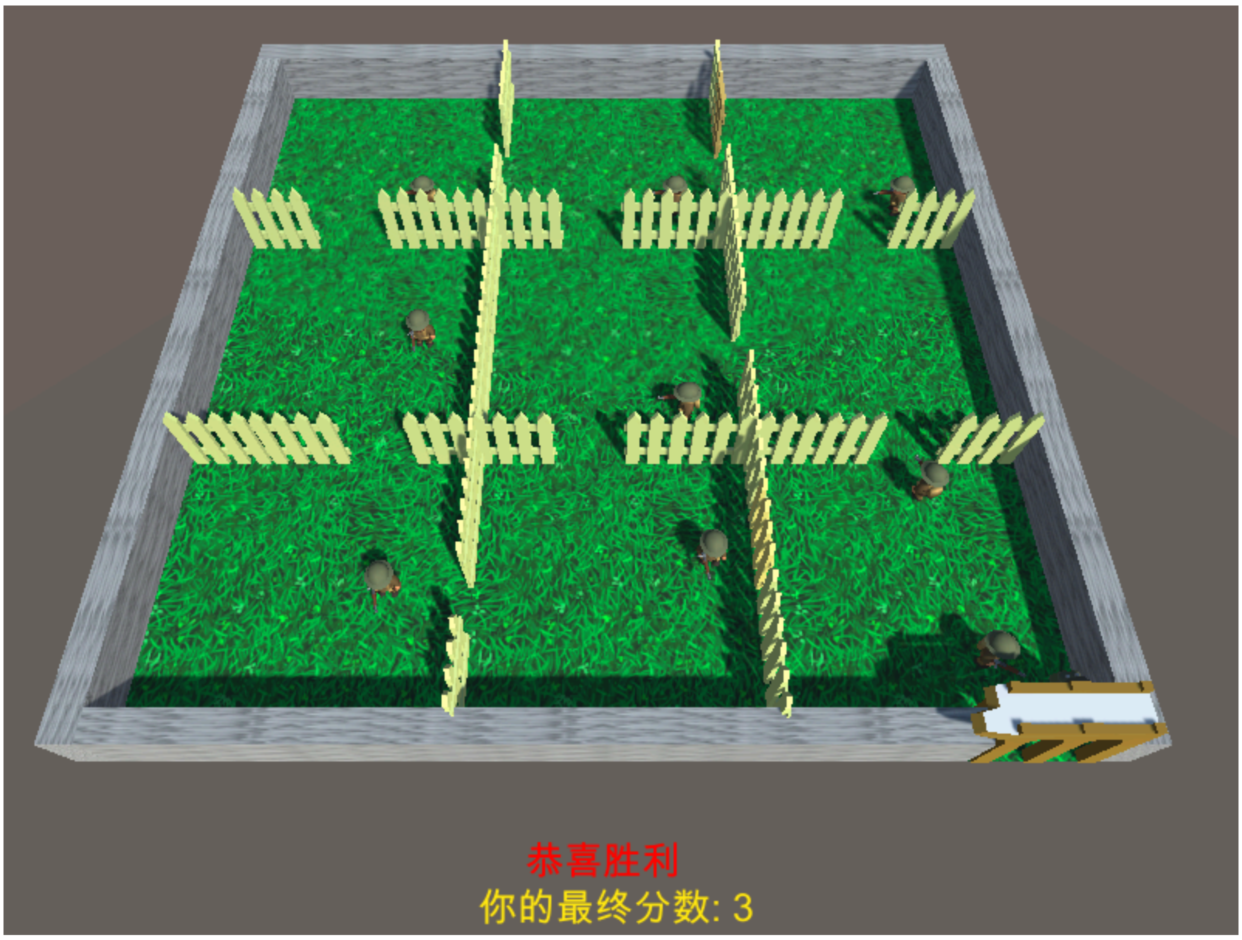
```
public class TurnColorScript : MonoBehaviour
{
    void OnEnable()
    {
        EventManager.OnClicked += TurnColor;//注册该事件
    }


    void OnDisable()
    {
        EventManager.OnClicked -= TurnColor;//取消该事件
    }


    void TurnColor()//回调函数的实现
    {
        Color col = new Color(Random.value, Random.value, Random.value);
        renderer.material.color = col;
    }
}
```

由于尝试了许多Asset Store中的模型和动画均不能得到很好的效果，所以使用了参考博客中的模型和动作。

游戏效果图



思路：

1. 同样使用之前的框架和模式，如单例模式，游戏工厂模式，动作基类，动作管理器，记分管理器，添加新的订阅发布模式负责事件的绑定和响应。
2. 利用Capsule Collider组件实现OnTriggerCollide()功能，并设置每隔半秒判断一次，防止每帧的判断。
3. 为每个运动的游戏对象添加Animator Controller实现动画，通过脚本更改参数实现动画转移，注意取消Has exit time。
4. 利用动作管理器的回调函数实现巡逻兵两个动作之间的转移。

核心代码：

1.事件管理器

Escape()为玩家逃脱的事件，Over()为游戏结束的事件

```
//EventManager.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EventManager : MonoBehaviour {

    public delegate void ScoreEvent();
    public delegate void GameOverEvent();
    public static event ScoreEvent OnScore;
    public static event GameOverEvent OnGameOver;

    public void Escape()
    {
        if (OnScore != null)
        {
            OnScore();
        }
    }

    public void Over()
    {
        if(OnGameOver != null)
        {
            OnGameOver();
        }
    }
}
```

2.事件触发

在每个巡逻兵上挂载两个组件，一个判断是否catch玩家，另一个判断是否follow玩家，分别使用OnCollisionEnter()和OnTriggerEnter()，OnTriggerExit()判断。

```
//PlayerDead.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerDead : 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("catch");
            Singleton<EventManager>.Instance.Over();//触发游戏结束事件
        }
    }
}
```

```
//CatchPlayer.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CatchPlayer : MonoBehaviour
{
    void OnTriggerEnter(Collider collider)
    {
        if (collider.gameObject.tag == "Player")
        {
            //玩家进入侦察兵追捕范围
            this.gameObject.transform.GetComponent<PatrolData>().follow_player =
true;
            this.gameObject.transform.GetComponent<PatrolData>().player =
collider.gameObject;
        }
    }
    void OnTriggerExit(Collider collider)
    {
        if (collider.gameObject.tag == "Player")
        {
            this.gameObject.transform.GetComponent<PatrolData>().follow_player =
false;
            this.gameObject.transform.GetComponent<PatrolData>().player = null;
            Singleton<EventManager>.Instance.Escape();//触发玩家逃脱事件
        }
    }
}
```

3.巡逻兵的具体动作

巡逻兵的数据

```
//PatrolData.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PatrolData : MonoBehaviour {
    public bool follow_player = false;    //是否跟随玩家
    public GameObject player;             //玩家游戏对象
    public Vector3 start_position;        //当前巡逻兵初始位置
}
```

巡逻兵有两个简单动作，一个为按矩形方向巡逻移动，一个为跟随玩家的动作。

由于设置游戏对象均有Rigidbody组件，所以重写SSAction的FixedUpdate()函数。

在进行移动时，同时置Patrol Animator Controller中的run变量为true，从而执行行走的动画。

```
//PatrolAction.cs
...
   public override void Start()
    {
        this.gameobject.GetComponent<Animator>().SetBool("run", true);
        data  = this.gameobject.GetComponent<PatrolData>();//每次得到此巡逻兵的数据
    }
...
```

```
//PatrolFollowAction.cs
...
    public override void FixedUpdate()
    {
        transform.position = Vector3.MoveTowards(this.transform.position,
  player.transform.position, speed * Time.deltaTime);
        this.transform.LookAt(player.transform.position);
    }
...
```

4.巡逻兵的动作管理器

负责执行巡逻兵的巡逻动作

```
//PatrolActionController.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PatrolActionController : SSActionManager {
    private PatrolAction patrolAction;

    public void GoPatrol(GameObject patrolObj)
    {
        patrolAction = PatrolAction.GetSSAction(patrolObj.transform.position);
        this.RunAction(patrolObj, patrolAction, this);
    }
    public void DestroyAllAction()
    {
        DestroyAll();
    }
}
```

巡逻动作与跟随动作的转换

```
//PatrolAction.cs
...
 public override void FixedUpdate()
 {
     if (data.follow_player)
     {
         //每隔0.5s检查一次巡逻兵是否需要跟随玩家，防止玩家在Trigger的边缘反复触发反复改
变foll_player的值
         if (time > 0.5) {
             this.destory = true;
             this.callback.SSActionEvent(this,0,this.gameobject);//执行
SSActionManager中的回调函数
             time = 0;
         } else{
             time += Time.deltaTime;
         }
     }
 }
...

//PatrolFollowAction.cs
...
public override void FixedUpdate()
 {
    if (!data.follow_player ){
        if (time > 0.5) {
                this.destory = true;
                this.callback.SSActionEvent(this,1,this.gameobject);
                time = 0;
            } else{
                time += Time.deltaTime;
            }
    }
}
...
```

回调函数负责切换两个动作

```
//SSActionManager.cs
...
    public void SSActionEvent(SSAction source,int intParam = 0, GameObject
objectParam = null){
        if(intParam == 0){
            //巡逻兵跟随玩家
            PatrolFollowAction follow =
PatrolFollowAction.GetSSAction(objectParam.gameObject.GetComponent<PatrolData>
().player);
            this.RunAction(objectParam, follow, this);
        }
        else{
            //巡逻兵按照初始位置开始继续巡逻
            PatrolAction run = 
PatrolAction.GetSSAction(objectParam.gameObject.GetComponent<PatrolData>
().start_position);
            this.RunAction(objectParam, run, this);
        }
}
...
```

5.巡逻兵的对象工厂

由于没有设置重置功能，所以只有从预制加载游戏对象。

```
//GameFactory.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameFactory : MonoBehaviour{
    public GameObject patrolObj;
    private List<GameObject> used = new List<GameObject>();
    private Vector3[] vec = new Vector3[9];

    public List<GameObject> GetPatrols()
    {
        int[] pos_x = { -6, 4, 13 };
        int[] pos_z = { -4, 6, -13 };
        int index = 0;

        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++)
        {
            patrolObj =Instantiate(Resources.Load<GameObject>("Prefabs/Patrol"),
Vector3.zero, Quaternion.identity) as GameObject;
            patrolObj.transform.position = vec[i];
            patrolObj.GetComponent<PatrolData>().start_position = vec[i];
            used.Add(patrolObj);
        }
        return used;
    }
}
```

6.场记

同样添加积分管理器和巡逻兵的动作管理器，且使用单例模式。

```
//FirstController.cs
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FirstController : MonoBehaviour, ISceneController, UserAction {

    public PatrolActionController actionController { get; set; }  //动作管理器
    public ScoreRecorder scoreRecorder { get; set; }  //积分管理器
    public GameFactory gf;//对象工厂，加载巡逻兵对象
    public GameObject playerObj;//玩家
    public List<GameObject> patrols;//巡逻兵
    private bool isGameOver = false;//游戏结束判断标志
    private bool isWin = false;//游戏获胜判断标志
    public float player_speed = 3; //玩家移动速度
    public float rotate_speed = 120f;//玩家旋转速度
    //注册事件
    void OnEnable(){
        EventManager.OnScore += AddScore;
        EventManager.OnGameOver += GameOver;
    }
    //取消事件
    void OnDisable(){
        EventManager.OnScore -= AddScore;
        EventManager.OnGameOver += GameOver;
    }
    //定义事件的回调函数
    void GameOver(){
        isGameOver = true;
        actionController.DestroyAllAction();
        for (int i = 0; i < patrols.Count; i++)
        {
            patrols[i].GetComponent<Animator>().SetBool("run", false);
        }
    }
    //定义事件的回调函数
    void AddScore(){
        scoreRecorder.Record ();
    }
    //初始化
    void Awake(){
        SSDirector director = SSDirector.getInstance();
        director.currentSceneController = this;
        scoreRecorder = this.gameObject.AddComponent<ScoreRecorder>();
        actionController = this.gameObject.AddComponent<PatrolActionController>();
        gf = Singleton<GameFactory>.Instance;
        director.currentSceneController.LoadResources();
    }

    void Update(){

 if(!isGameOver&&playerObj.transform.position.x>=10&&playerObj.transform.position.
z<=-13.3)       {
            Win();
        }
    }
    //移动玩家
    public void movePlayer(float translationX, float translationZ)
    {
        if(!isGameOver)
        {
            if (translationX != 0 || translationZ != 0)
            {
                playerObj.GetComponent<Animator>().SetBool("run", true);
            }
            else
            {
                playerObj.GetComponent<Animator>().SetBool("run", false);
            }
            playerObj.transform.Translate(0, 0, translationZ * player_speed *
Time.deltaTime);
            playerObj.transform.Rotate(0, translationX * rotate_speed *
Time.deltaTime, 0);
        }
    }

    public int getScore(){
        return scoreRecorder.getScore();
    }

    public bool getGameover(){
        return isGameOver;
    }

    void Win(){
        isWin = true;
        GameOver();
    }
    public bool getWin(){
        return isWin;
    }
    //加载预制
    public void LoadResources()  {
        Instantiate(Resources.Load<GameObject>("Prefabs/Plane"), Vector3.zero,
Quaternion.identity);
        playerObj = Instantiate(Resources.Load<GameObject>("Prefabs/Player"), new
Vector3(-10,0,-10), Quaternion.identity) as GameObject;
        playerObj.tag = "Player";
        Debug.Log (gf);
        patrols = gf.GetPatrols();
        for (int i = 0; i < patrols.Count; i++)
        {
            actionController.GoPatrol(patrols[i]);
        }
    }
}
```

7.UserGUI

其中UserAction有

```
public interface UserAction {
    void movePlayer(float translationX, float translationZ);//移动玩家
    int getScore();//获得当前游戏的分数
    bool getGameover();//获得游戏结束的状态
    bool getWin();//获得游戏胜利的状态
}
```

```
//UserGUI.cs
...
    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);
    }
    private void OnGUI()
    {
        GUI.Label(new Rect(10, 5, 200, 50), "分数:", text_style);
        GUI.Label(new Rect(55, 5, 200, 50), action.getScore().ToString(),
score_style);
        if(action.getWin()){
            over_style.normal.textColor = Color.red;
            GUI.Label(new Rect(Screen.width / 2 - 50, Screen.width / 2 - 100, 100, 
100), "恭喜胜利", over_style);
            over_style.normal.textColor = Color.yellow;
            GUI.Label(new Rect(Screen.width / 2 - 80, Screen.width / 2 - 70, 100, 
100), "你的最终分数: "+action.getScore().ToString(), over_style);
        }
        if(!action.getWin()&&action.getGameover())
        {
            over_style.normal.textColor = Color.black;
            GUI.Label(new Rect(Screen.width / 2 - 50, Screen.width / 2 - 100, 100, 
100), "游戏结束", over_style);
            over_style.normal.textColor = Color.yellow;
            GUI.Label(new Rect(Screen.width / 2 - 80, Screen.width / 2 - 70, 100, 
100), "你的最终分数: "+action.getScore().ToString(), over_style);
        }
        GUI.Label(new Rect(10, 5, 200, 50), "分数:", text_style);
        GUI.Label(new Rect(55, 5, 200, 50), action.getScore().ToString(), 
score_style);
    }
```