四、与游戏世界交互

1、编写一个简单的鼠标打飞碟（Hit UFO）游戏

- **游戏内容要求：**

1. 游戏有 n 个 round，每个 round 都包括10 次 trial；
2. 每个 trial 的飞碟的色彩、大小、发射位置、速度、角度、同时出现的个数都可能不同。它们由该 round 的 ruler 控制；
3. 每个 trial 的飞碟有随机性，总体难度随 round 上升；
4. 鼠标点中得分，得分规则按色彩、大小、速度不同计算，规则可自由设定。

- **游戏的要求：**

  - 使用带缓存的工厂模式管理不同飞碟的生产与回收，该工厂必须是场景单实例的！具体实现见参考资源 Singleton 模板类
  - 近可能使用前面 MVC 结构实现人机交互与游戏模型分离

如果你的使用工厂有疑问，参考：[弹药和敌人：减少，重用和再利用]

2、编写一个简单的自定义 Component （选做）

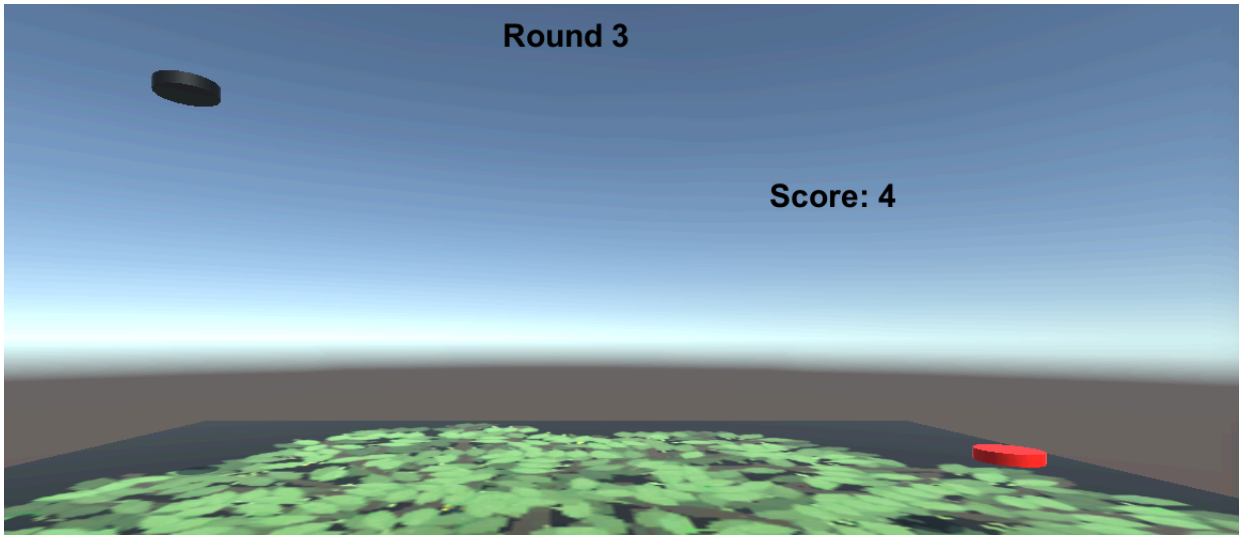
- **用自定义组件定义几种飞碟，做成预制**

  - 参考官方脚本手册 https://docs.unity3d.com/ScriptReference/Editor.html
  - 实现自定义组件，编辑并赋予飞碟一些属性

如果你想了解跟多自定义插件或编辑器的话题，请参考：Unity3d自定义一个编辑器组件/插件的简易教程

> 参考博客：https://blog.csdn.net/x2_yt/article/details/66969242
>
> 以及课件中的基本框架

游戏效果图：



游戏规则：

1. 共有3个回合，每个回合随机抛出10个飞碟，未击中飞碟不会扣分。
2. 难度每回合递增，击中黄色+1，击中红色+2，击中黑色+4。

3、知识准备：

（1）游戏对象的创建

- new GameObject();
- new GameObject(string name);
- new GameObject(string name, params Type[] components);
- GameObject CreatePrimitive(PrimitiveType type); //创建基础类型游戏对象
- Instantiate (brick, new Vector3(x, y, 0), Quaterni on.identity); //从已知对象或预制克隆（主要方法）

（2）游戏对象的销毁

- Object.Destroy(Object obj, float t = 0.0F); //对象销毁

  如果是组件对象，则从游戏对象中立即摘除；

  如果是游戏对象，则不会在update期间立即销毁，通常在 render 前销毁它的部件以及所有子对象；

- Object. DestroyImmediate；//立即销毁

  Unity 建议不要立即销毁，这可能导致离散引擎并发的行为 之间依赖关系产生不可预知错误；

（3）自定义的游戏对象属性

创新游戏对象“子类”

- Unity 不建议用户继承 GameObject
- 通过改游戏对象添加行为，作为内部数据存储

检测一个对象是否拥有数据属性

- GetComponent()

自定义游戏对象的属性

类似自定义组件

创建一个脚本 DiskData:MonoBehaiver

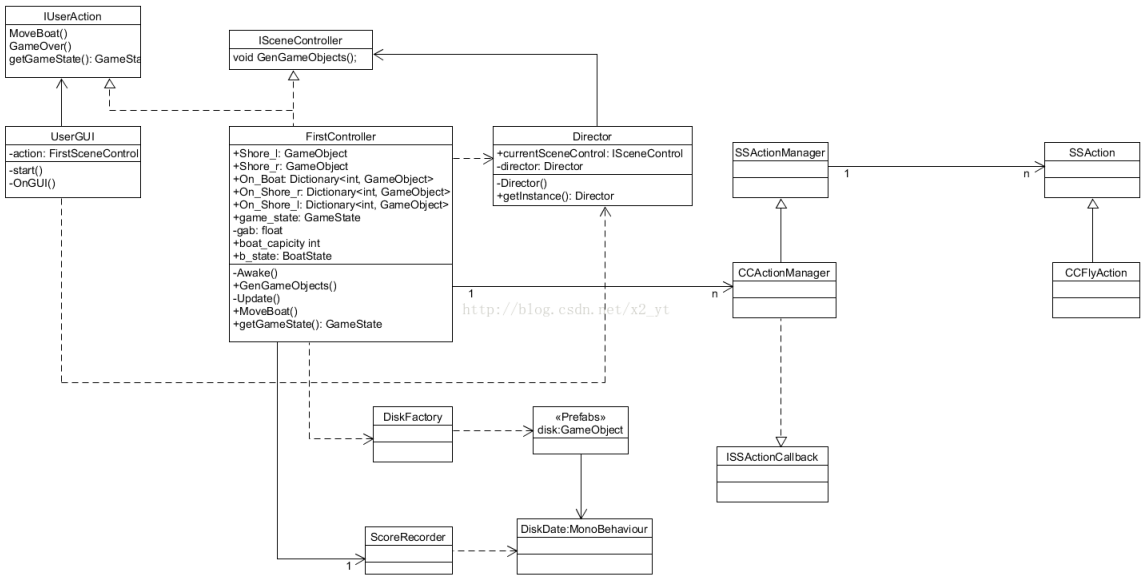
添加公共属性与方法，删除 start 和 update 方法

挂载到一个 Disk 的游戏对象（圆柱）

制作成预制，即可添加飞碟的属性

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

public class DiskData : MonoBehaviour {
    public Vector3 direction;
    public Vector3 size;
    public Color color;
    public float speed;
}
```



道具缓存工厂模式的好处

- 游戏对象的创建与销毁高成本，必须减少销毁次数。如： 游戏中子弹
- 屏蔽创建与销毁的业务逻辑，使程序易于扩展

- 通过场景单实例，构建了方便可取获取DISK的类；
- 包装了复杂的Disk生产与回收逻辑，易于使用；
- 它包含Disk产生规则（控制每个round的难度），可以积 极应对未来游戏规则的变化，减少维护成本

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DiskFactory : MonoBehaviour{

    public GameObject diskObj;  //保存飞碟游戏对象

    private List<DiskData> used = new List<DiskData>();   //用来保存正在使用的飞碟
    private List<DiskData> free = new List<DiskData>();   //用来保存未使用的飞碟

    private void Awake()
    {
        //实例化预制
        diskObj =Instantiate(Resources.Load<GameObject>("Prefabs/Disk"),
Vector3.zero, Quaternion.identity) as GameObject;
        diskObj.SetActive(false);
    }

    //判断free有没有空余飞碟，有则使用，无则生成
    //然后进行
```

```
    public GameObject GetDisk(int round)
    {
        GameObject new_disk = null;
        if (free.Count > 0)
        {
            new_disk = free[0].gameObject;
            free.Remove(free[0]);
        }
        else
        {
            new_disk = Instantiate<GameObject>(diskObj, new Vector3(2,-4,0),
Quaternion.identity);
            new_disk.AddComponent<DiskData>();
        }

        /**
        * 以下几句代码是用来随机生成飞碟的颜色的，并根据回合数来限制飞碟可用的颜色
        * 第一回合智能生成黄色的飞碟，第二回合飞碟可以有黄色和红色，第三回合黄，红
        * 黑三种颜色的飞碟都可以出现，
        */

        //frequency表示每round出现飞碟的频率
        int frequency = 0;
        if (round == 1) frequency = 100;
        if (round == 2) frequency = 250;

        //随机生成飞碟的颜色
        int selectedColor = Random.Range(frequency, round * 499);

        if (selectedColor > 500)  //250-998
            round = 2;
        else if (selectedColor > 300)  //100-499
            round = 1;
        else
            round = 0;


        //根据回合数来生成相应的飞碟

        switch (round){
            case 0:
                {
                    new_disk.GetComponent<DiskData>().color = Color.yellow;
                    new_disk.GetComponent<DiskData>().speed = 8.0f;
                    float RanX = UnityEngine.Random.Range(-1f, 1f) < 0 ? -0.5f :
0.5f;
                    new_disk.GetComponent<DiskData>().direction = new
Vector3(RanX, 1, 0);
```

```
                    new_disk.GetComponent<Renderer>().material.color =
Color.yellow;
                    break;
                }
            case 1:
                {
                    new_disk.GetComponent<DiskData>().color = Color.red;
                    new_disk.GetComponent<DiskData>().speed = 9.0f;
                    float RanX = UnityEngine.Random.Range(-1f, 1f) < 0 ? -0.7f :
0.7f;
                    new_disk.GetComponent<DiskData>().direction = new
Vector3(RanX, 1, 0);
                    new_disk.GetComponent<Renderer>().material.color =
Color.red;
                    break;
                }
            case 2:
                {
                    new_disk.GetComponent<DiskData>().color = Color.black;
                    new_disk.GetComponent<DiskData>().speed = 10.0f;
                    float RanX = UnityEngine.Random.Range(-1f, 1f) < 0 ?-0.9f :
0.9f;
                    new_disk.GetComponent<DiskData>().direction = new
Vector3(RanX, 1, 0);
                    new_disk.GetComponent<Renderer>().material.color =
Color.black;
                    break;
                }
        }

        used.Add(new_disk.GetComponent<DiskData>());
        new_disk.SetActive(true);
        new_disk.name = new_disk.GetInstanceID().ToString();
        return new_disk;
    }

    public void FreeDisk(GameObject disk)
    {
        DiskData tmp = null;
        foreach (DiskData i in used)
        {
            if (disk.GetInstanceID() == i.gameObject.GetInstanceID())
            {
                tmp = i;
            }
        }
        if (tmp != null) {
```

```
            tmp.gameObject.SetActive(false);
            free.Add(tmp);
            used.Remove(tmp);
        }
    }

}
```

场景单例模式

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

public class Singleton<T> where T : MonoBehaviour
{
    private static T instance;

    public static T Instance {
        get {
            if (instance == null) {
                instance = (T)Object.FindObjectOfType(typeof(T));
                if (instance == null) {
                    Debug.LogError("Can't find instance of " + typeof(T));
                }
            }
            return instance;
        }
    }
}
```

添加积分管理器

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

public class ScoreRecorder : MonoBehaviour {

    public int score;

    //不同颜色得分不同
    private Dictionary<Color, int> scoreTable = new Dictionary<Color, int>();

    void Start () {
        score = 0;
        scoreTable.Add(Color.yellow, 1);
        scoreTable.Add(Color.red, 2);
        scoreTable.Add(Color.black, 4);
    }

    public void Record(GameObject disk)  {
        score += scoreTable[disk.GetComponent<DiskData>().color];
    }

    public void Reset() {
        score = 0;
    }
}
```

场记的管理器增加

其他类似动作分离版的框架，在场记中添加动作管理器和积分管理器。

场记负责实现UserAction接口，以及资源实例化。

添加GameState枚举类型记录游戏状态。

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

public enum GameState { ROUND_START, ROUND_FINISH, RUNNING, END, START};

public class FirstController : MonoBehaviour, ISceneController, UserAction {


    public SceneActionManager actionManager { get; set; }  //动作管理器
```

```
    public ScoreRecorder scoreRecorder { get; set; }  //积分管理器

    public Queue<GameObject> diskQueue = new Queue<GameObject>();  //每回合抛出的飞
碟

    private int disk_number;  //抛出的飞碟总数

    private int currentRound = -1;  //当前回合

    public int round = 3;  //回合总数

    private float time = 0;  //抛出时间间隔

    private GameState gameState = GameState.START;  //当前游戏状态

    void Awake () {
        SSDirector director = SSDirector.getInstance();
        director.currentSceneController = this;
        disk_number = 10;
        this.gameObject.AddComponent<ScoreRecorder>();
        this.gameObject.AddComponent<DiskFactory>();
        scoreRecorder = Singleton<ScoreRecorder>.Instance;
        director.currentSceneController.LoadResources();
    }

    private void Update() {
        //每回合结束
        if (actionManager.diskNumber == 0 && gameState == GameState.RUNNING) {
            if (currentRound == 2)
                gameState = GameState.END;
            else
                gameState = GameState.ROUND_FINISH;
            diskQueue.Clear ();
        }

        //每回合开始
        if (actionManager.diskNumber == 0 && gameState == GameState.ROUND_START)
{
            currentRound = (currentRound + 1) % round;  //0,1,2
            NextRound();
            actionManager.diskNumber = disk_number;
            gameState = GameState.RUNNING;
        }

        //设置每次时间间隔为1s
        if (time > 0.5)  {
```

```
            ThrowDisk();
            time = 0;
        }
        else
            time += Time.deltaTime;
    }

    //从工厂加载disk_number个飞碟，并入队
    private void NextRound()  {
        DiskFactory df = Singleton<DiskFactory>.Instance;
        for (int i = 0; i < disk_number; i++)  {
            diskQueue.Enqueue(df.GetDisk(currentRound));
        }
//      actionManager.Fly(diskQueue);
    }

    //每隔0.5s抛出一个飞碟
    void ThrowDisk()  {
        if (diskQueue.Count != 0) {
            GameObject disk = diskQueue.Dequeue();

            //飞碟出现的位置
            Vector3 position = new Vector3(0, 0, 0);
            float y = UnityEngine.Random.Range(0f, 2f);
            position = new Vector3(-disk.GetComponent<DiskData>().direction.x *
7, y, 0);
            disk.transform.position = position;

            actionManager.Fly(disk);  //抛出飞碟

            disk.SetActive(true);
        }

    }
    //击中飞碟
    public void hit(Vector3 pos) {
        Ray ray = Camera.main.ScreenPointToRay(pos);

        RaycastHit[] hits;
        hits = Physics.RaycastAll(ray);
        for (int i = 0; i < hits.Length; i++)
        {
            RaycastHit hit = hits[i];

            if (hit.collider.gameObject.GetComponent<DiskData>() != null)
            {
                scoreRecorder.Record(hit.collider.gameObject);
```

```
                //飞碟被击中，使其落地，然后被回收
                hit.collider.gameObject.transform.position = new Vector3(0, -4,
0);
            }
        }
    }

    private void OnGUI(){
        if (gameState == GameState.END) {
            GameOver ();
        }

    }

    public void LoadResources()  {
        Instantiate(Resources.Load<GameObject>("Prefabs/Ground"));
    }

    public void GameOver() {
        GUIStyle button_style = new GUIStyle {
            fontSize=30,
            fontStyle=FontStyle.Bold
        };
        button_style.normal.textColor = Color.red;
        GUIStyle score_style = new GUIStyle {
            fontSize=30,
            fontStyle=FontStyle.Bold
        };

        GUI.Label(new Rect(Screen.width / 2 -50, Screen.height / 2 - 200, 100,
50), "GAMEOVER",button_style);

        GUI.Label(new Rect(Screen.width / 2 + 200, Screen.height / 2-150 , 100,
50), "Your final Score: "+this.GetScore().ToString(),score_style);
    }

    public int GetScore()  {
        return scoreRecorder.score;
    }

    public GameState getGameState() {
        return gameState;
    }

    public int getGameRound() {
        return currentRound+1;
```

```
    }

    public void setGameState(GameState gs)  {
        gameState = gs;
    }
}
```

飞碟的简单动作

根据水平速度和重力加速度分别改变水平和竖直位置。

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

public class CCMoveToActions : SSAction {

    float gravity;  //重力加速度

    float horizonSpeed;  //水平速度

    Vector3 direction;  //初始飞行方向

    float time;  //飞行时间

    //重写基类
    public override void Start () {
        enable = true;
        gravity = 9.8f;
        time = 0;
        horizonSpeed = gameobject.GetComponent<DiskData>().speed;
        direction = gameobject.GetComponent<DiskData>().direction;
    }

    public override void Update () {
        if (gameobject.activeSelf)
        {
            time += Time.deltaTime;

            transform.Translate(Vector3.down * gravity * time * Time.deltaTime);
 //竖直位移

            transform.Translate(direction * horizonSpeed * Time.deltaTime);  //水
平位移

            if (this.transform.position.y < -5) //落地并回收
```

```
            {
                this.destory = true;
                this.enable = false;
                this.callback.SSActionEvent(this);
            }
        }

    }

    public static CCMoveToActions GetSSAction()
    {
        CCMoveToActions action = ScriptableObject.CreateInstance<CCMoveToActions>
();
        return action;
    }
}
```

飞碟动作的管理器

注意这回实现回调函数。

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

public class SceneActionManager : SSActionManager, ISSActionCallback {

    public FirstController sceneController;
    public List<CCMoveToActions> actions;
    public int diskNumber = 0;

    private List<SSAction> used = new List<SSAction>();  //used是用来保存正在使用的
动作
    private List<SSAction> free = new List<SSAction>();  //free是用来保存未使用的动
作

    //飞碟动作的缓存工厂模式
    SSAction GetSSAction() {
        SSAction action = null;
        if (free.Count > 0)
        {
            action = free[0];
            free.Remove(free[0]);
        }
        else
```

```
            action = ScriptableObject.Instantiate<CCMoveToActions>(actions[0]);

        used.Add(action);
        return action;
    }

    public void FreeSSAction(SSAction action){
        SSAction tmp = null;
        foreach (SSAction i in used) {
            if (action.GetInstanceID() == i.GetInstanceID())
                tmp = i;
        }
        if (tmp != null) {
            tmp.reset();
            free.Add(tmp);
            used.Remove(tmp);
        }
    }

    //场记的动作管理器为此场景
    protected new void Start() {
        sceneController = (FirstController)SSDirector.getInstance
().currentSceneController;
        sceneController.actionManager = this;
        actions.Add(CCMoveToActions.GetSSAction());

    }

    //执行完每次飞碟动作执行的回调函数
    //即飞碟落地后执行飞碟工厂的回收
    public new void SSActionEvent(SSAction source){
        if (source is CCMoveToActions) {
            diskNumber--;
            DiskFactory df = Singleton<DiskFactory>.Instance;
            df.FreeDisk(source.gameobject);
            FreeSSAction(source);
        }
    }

    //抛出一系列飞碟，执行简单动作
    public void Fly(GameObject disk){
        RunAction(disk, GetSSAction(), (ISSActionCallback)this);
    }
}
```

UserGUI

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

public class UserGUI : MonoBehaviour
{
    //与之前一样，UserAction代表用户的动作
    //接口在Interface中，实现在场记里
    private UserAction action;
    bool isStart = false;

    void Start () {
        action = SSDirector.getInstance().currentSceneController as UserAction;
    }

    private void OnGUI()
    {
        GUIStyle score_style = new GUIStyle {
            fontSize=30,
            fontStyle=FontStyle.Bold
        };

        GUIStyle text_style = new GUIStyle {
            fontSize=30,
            fontStyle=FontStyle.Bold
        };

//      GUIStyle button_style = new GUIStyle {
//          fontSize=30
//      };
//
        //鼠标左键点击
        if (Input.GetButtonDown("Fire1")) {
            Vector3 pos = Input.mousePosition;
            action.hit(pos);
        }

        if (isStart && action.getGameState () == GameState.RUNNING)
            GUI.Button (new Rect (Screen.width / 2 - 50, Screen.height / 2 - 300,
100, 50), "Round " + action.getGameRound (), text_style);

        if(action.getGameState()!=GameState.END)
            GUI.Label(new Rect(Screen.width / 2 + 200, Screen.height / 2-150 ,
100, 50), "Score: "+action.GetScore().ToString(),score_style);
```

```
        if (!isStart && GUI.Button(new Rect(Screen.width / 2 -50, Screen.height /
2 - 300, 100, 50), "Start")) {
            isStart = true;
            action.setGameState(GameState.ROUND_START);
        }

        if (isStart && action.getGameState() == GameState.ROUND_FINISH &&
GUI.Button(new Rect(Screen.width / 2 -50, Screen.height / 2 -300, 100, 50), "Next
Round")) {
            action.setGameState(GameState.ROUND_START);
        }
    }
}
```

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

public interface ISceneController {
    void LoadResources ();
}

public interface UserAction {
    void hit(Vector3 pos);
    void GameOver();
    GameState getGameState();
    void setGameState(GameState gs);
    int GetScore();
    int getGameRound();
}
```