# 3d游戏作业第五次作业

# 3d游戏设计读书笔记五

## 一、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；

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：

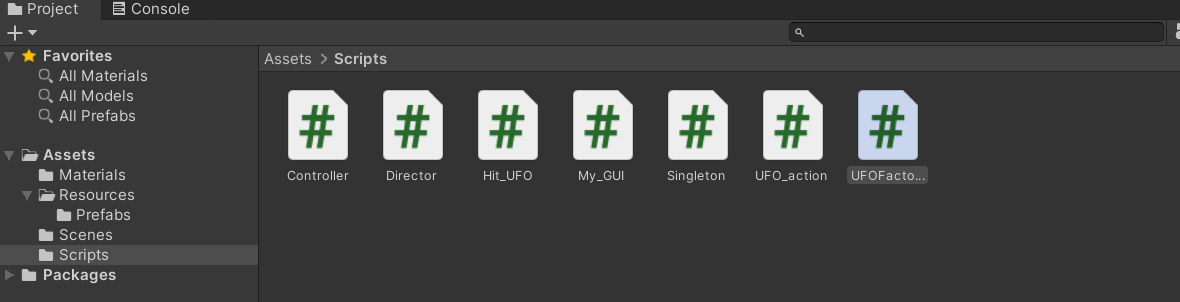
1. 使用带缓存的工厂模式管理不同飞碟的生产与回收，该工厂必须是场景单实例的！具体实现见参考资源 Singleton 模板类

2. 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；参考：Unity对象池（Object Pooling）理解与简单应用 代码质量较低，比较凌乱

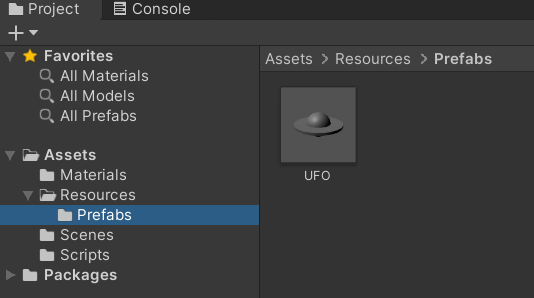
### 实现

1. \*\*脚本前览\*\*



2. \*\*飞碟预设\*\*

球+圆盘



使用标准材质

3. \*\*Director\*\*

Director 使用单例模式，设置一个 Director类 的private static成员变量，并设置一个与之配套的 getInstance() 的成员函数。同时设置一个 currentController 的成员变量。

```java

public class Director : System.Object

{

private static Director \_instance;

public Controller currentController { get; set; }

public static Director getInstance()

{

if (\_instance == null)

{

\_instance = new Director();

}

return \_instance;

}

}

```

4. \*\*Controller\*\*

Controller 负责统筹各个组件之间的关系，所以在其中我们需要设置一下几个成员变量：\_UFOfactory、\_director、my\_diskfactory ，并对其进行初始化、实例化。需要注意的是， \_UFOfactory 采用的是单例模式。

```java

public class Controller : MonoBehaviour

{

public UFOFactory \_UFOfactory;

public Director \_director;

private GameObject my\_diskfactory;

void Awake()

{

Random.InitState((int)System.DateTime.Now.Ticks);

my\_diskfactory = new GameObject("Disk\_Factory");

my\_diskfactory.AddComponent<UFOFactory>();

\_director = Director.getInstance();

\_UFOfactory = Singleton<UFOFactory>.Instance;

\_director.currentController = this;

}

}

```

5. \*\*UFOaction\*\*

UFOaction 定义了UFO的飞行动作，其中：

（1）成员变量：

```java

public Director \_director;

public GameObject player; //记录该动作所归属的对象

Vector3 start; //记录UFO飞行的初始位置

Vector3 end; //记录UFO飞行的结束位置

public int speed=3; //记录UFO飞行的速度

public bool running = true;//运行态标志位

```

(2)成员函数

a. Start()

设置UFO 开始位置 以及 结束位置 ，并调用 setColor函数 调整其颜色。

```java

public void Start()

{

\_director = Director.getInstance();

start = new Vector3(Random.Range(-20f,20f), Random.Range(-20f, 20f), 0);

if (start.x < 10 && start.x > -10)

start.x \*= 4;

if (start.y < 10 && start.y > -10)

start.y \*= 4;

end = new Vector3(-start.x, -start.y, 0);

player.transform.position = start;

setColor();

}

```

b. Update()

使用 MoveTowards函数 移动UFO。并且判定UFO是否移动到其结束位置，若是移动到其结束位置，证明其未被集中，则调用 not\_hit 函数，回收UFO。

```java

public void Update()

{

if (running)

{

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

if (player.transform.position == end)

{

this.\_director.currentController.\_UFOfactory.not\_hit(this.player);

}

}

}

```

c. setColor()

使用 Random 随机一个数，并根据这个数设置UFO的颜色。

```java

public void setColor()

{

int color = Random.Range(1, 4);

switch (color)

{

case 1:

player.GetComponent<MeshRenderer>().material.color = Color.red;

foreach (Transform child in player.transform)

{

child.gameObject.GetComponent<MeshRenderer>().material.color = Color.red;

}

break;

case 2:

player.GetComponent<MeshRenderer>().material.color = Color.yellow;

foreach (Transform child in player.transform)

{

child.gameObject.GetComponent<MeshRenderer>().material.color = Color.yellow;

}

break;

case 3:

player.GetComponent<MeshRenderer>().material.color = Color.blue;

foreach (Transform child in player.transform)

{

child.gameObject.GetComponent<MeshRenderer>().material.color = Color.blue;

}

break;

default:

break;

}

}

```

6. \*\*UFOfactory\*\*

UFOfactory整合了UFO的动作管理员。

（1）成员变量

```java

public List<GameObject> used; //存储已经放飞的UFO

public List<GameObject> not\_used; //存储飞回来了的UFO

public List<UFO\_action> actions; //存储UFO动作，小型动作管理器

public int round = 0; //阶段

public int score = 0; //得分

```

（2）成员函数

a. Start()

为各个List指针创建List对象，并为各个List添加其对应成员。并为各个 UfO\_action 绑定对应的 Player 。

```java

private void Start()

{

used = new List<GameObject>();

not\_used = new List<GameObject>();

actions = new List<UFO\_action>();

for(int i = 0; i < 10; i++)

{

not\_used.Add(Object.Instantiate(Resources.Load("Prefabs/UFO", typeof(GameObject)), new Vector3(0, -20, 0), Quaternion.identity, null) as GameObject);

actions.Add(ScriptableObject.CreateInstance<UFO\_action>());

}

for(int i = 0; i < 10; i++)

{

actions[i].player = not\_used[i];

}

}

```

b. Update()

调用各个 处于工作状态的UFO\_action 的 Update函数 ，在 round < 10 的时候调用 get\_ready函数 来更新轮次。并在 round == 11 的时候停止对 UFO\_action 的操作。

```java

private void Update()

{

if (round <= 10)

{

for (int i = 0; i < 10; i++)

{

actions[i].Update();

}

if (not\_used.Count == 10)

{

round += 1;

if (round <= 10)

get\_ready(round);

}

}

}

```

c. hitted(GameObject g)

判别被击中的UFO是什么色的，并对应给分。将击中的UFO从 used列表 移动到 not\_used列表 ，并调整其为初始位置，再将其对应的 UFO\_action 调为非running。

```java

public void hitted(GameObject g)

{

if (g.gameObject.GetComponent<MeshRenderer>().material.color == Color.red)

score += 3;

else if (g.gameObject.GetComponent<MeshRenderer>().material.color == Color.yellow)

score += 2;

else if (g.gameObject.GetComponent<MeshRenderer>().material.color == Color.blue)

score += 1;

this.used.Remove(g);

g.transform.position = new Vector3(0, -20, 0);

for(int i = 0; i < 10; i++)

{

if (actions[i].player == g)

actions[i].running = false;

}

this.not\_used.Add(g);

}

```

d. not\_hit(GameObject g)

not\_hit函数 与 hitted函数相似，但减去了加分的环节。

```java

public void not\_hit(GameObject g)

{

this.used.Remove(g);

g.transform.position = new Vector3(0, -20, 0);

for (int i = 0; i < 10; i++)

{

if (actions[i].player == g)

actions[i].running = false;

}

this.not\_used.Add(g);

}

```

e. get\_ready(int round)

用于做UFO起飞之前的准备工作，它将 not\_used列表中所有的UFO移入 used列表 并按轮次调整其飞行速度，并调用 UFO\_action 的 Start函数 做初始化，并将其设为 running态 。

```java

public void get\_ready(int round)

{

for(int i = 0; i < 10; i++)

{

used.Add(not\_used[0]);

not\_used.Remove(not\_used[0]);

actions[i].speed = round + 3;

actions[i].Start();

actions[i].running = true;

}

}

```

7. \*\*Hit\_UFO\*\*

使用 Ray 来定位鼠标的位置以及鼠标点击到的游戏物体，当点击到物体的时候调用 UFOfactory 的 hitted函数。

```java

public class Hit\_UFO : MonoBehaviour

{

public GameObject cam;

public Director director;

private void Start()

{

director = Director.getInstance();

}

// Update is called once per frame

void Update()

{

if (Input.GetButtonDown("Fire1"))

{

Vector3 mp = Input.mousePosition; //get Screen Position

//create ray, origin is camera, and direction to mousepoint

Camera ca;

if (cam != null) ca = cam.GetComponent<Camera>();

else ca = Camera.main;

Ray ray = ca.ScreenPointToRay(Input.mousePosition);

//Return the ray's hit

RaycastHit hit;

if (Physics.Raycast(ray, out hit))

{

director.currentController.\_UFOfactory.hitted(hit.transform.gameObject);

}

}

}

}

```

8. \*\*My\_GUI\*\*

创建 Score 和 Round 的label。并在轮次到达11的时候，停止显示 Score 和 Round ，而在屏幕中间显示 Final Score 。

```java

private void OnGUI()

{

int my\_round = \_director.currentController.\_UFOfactory.round;

if (my\_round == 11)

{

GUIStyle ending = new GUIStyle();

ending.normal.background = null;

ending.normal.background = null;

ending.normal.textColor = new Color(0, 0, 0);

ending.fontSize = 80;

string ending\_score = "Final Score: " + \_director.currentController.\_UFOfactory.score.ToString();

GUI.Label(new Rect(0.13f \* Screen.width, 0.4f \* Screen.height, 300, 300), ending\_score, ending);

}

else

{

string round = my\_round.ToString();

round = "Round: " + round;

GUIStyle bb = new GUIStyle(); //创建GUI的格式

bb.normal.background = null;

bb.normal.textColor = new Color(0, 0, 0);

bb.fontSize = 25;

GUI.Label(new Rect(0.8f \* Screen.width, 240, 150, 35), round, bb);

string score = \_director.currentController.\_UFOfactory.score.ToString();

score = "Score:" + score;

GUI.Label(new Rect(0.8f \* Screen.width, 270, 150, 35), score, bb);

}

}

```

9. \*\*Singleton\*\*

使用 FindObjectOfType 来寻找一个类型的单例，并返回这个单例。

```java

public class Singleton<T> : MonoBehaviour where T: MonoBehaviour

{

protected static T instance;

public static T Instance

{

get

{

if (instance == null)

{

instance = (T)FindObjectOfType(typeof(T));

if (instance == null)

{

Debug.LogError("No instance of " + typeof(T));

}

}

return instance;

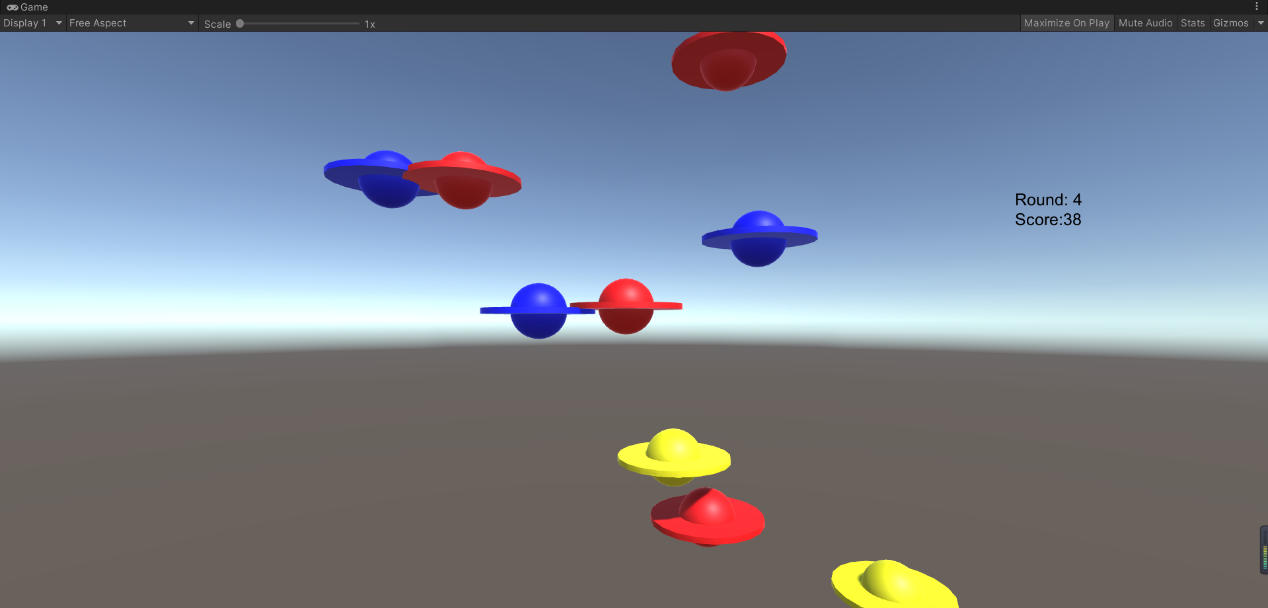
}

}

}

```

10. \*\*游戏截图\*\*



## 二、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](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

\*例如编写旋转脚本组件，让UFO使用不同的转动轴进行转动。通过，使用不同的速度与轴方向，就可以让我们的UFO按照不同的旋转轴旋转。将设计好的不同的组件挂载到不同的UFO上，就做好了几种飞碟的预制。\*

```java

public class around : MonoBehaviour

{

public float speed;

public float yangle, zangle;

void Update()

{

Vector3 axis = new Vector3(0, yangle, zangle);

this.transform.RotateAround(new Vector3(0, 0, 0), axis, speed \* Time.deltaTime);

this.transform.Rotate(Vector3.up \* 100 \* Time.deltaTime);

}

}

```