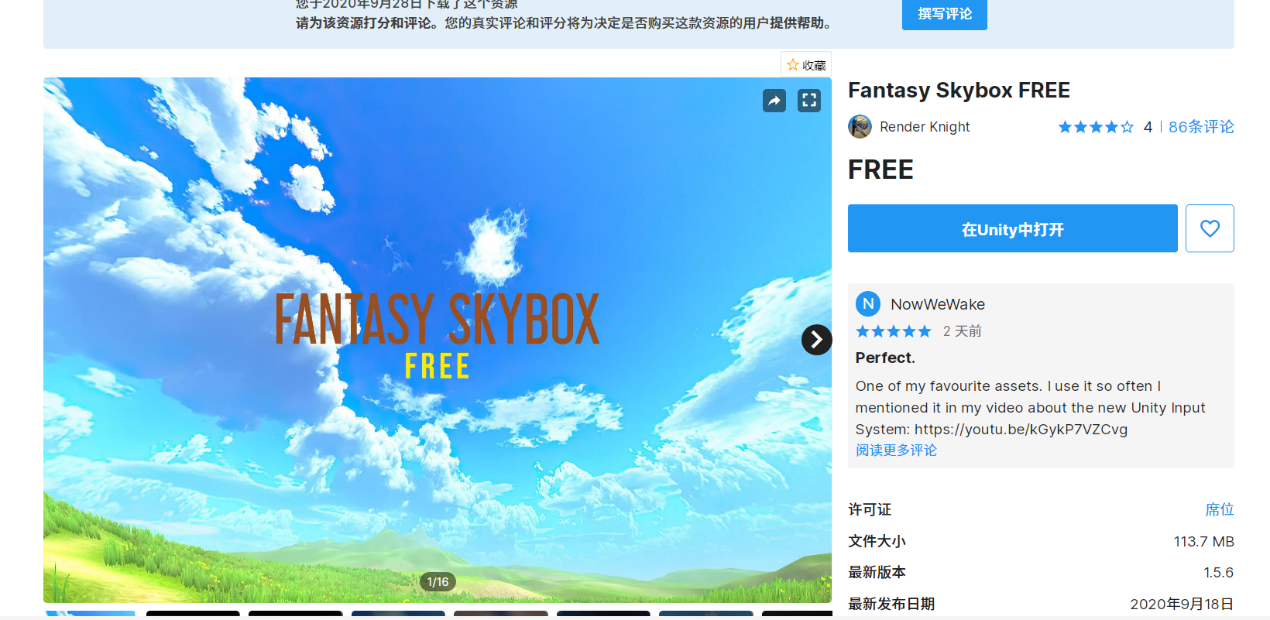
# 3d游戏作业四

# 3d游戏设计读书笔记四

## 一、基本操作演练【建议做】

1. 下载 Fantasy Skybox FREE， 构建自己的游戏场景

a. 在AssetStore中搜索Fantasy Skybox FREE并下载。



b. unity添加天空盒有两种方法，分别为向场景添加和向摄像机添加。首先我们在assert里面导入skybox素材。

（1）向camera里导入：

点击camera或者daomain camera，再点击component，选renderin,点skybox.再inspector里面会成这样，注意，第一个框框必须是skybox。然后在第二个框框里面导入素材。

（2）向scene里导入：

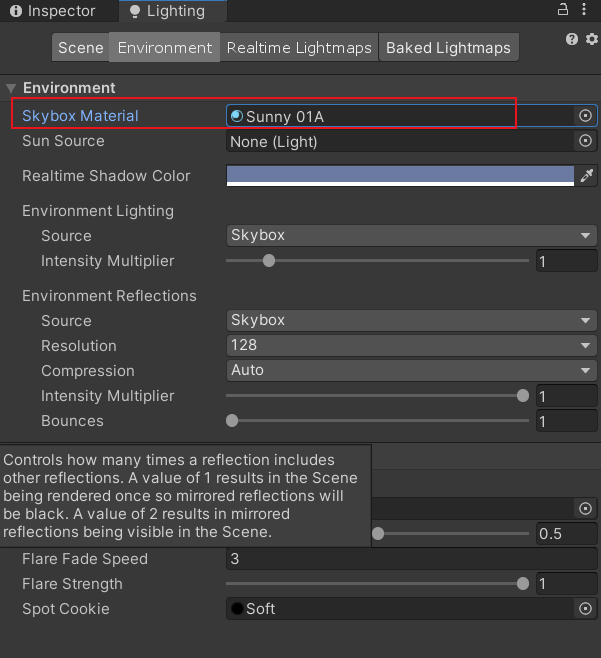
旧版：

在edit里点render setting即可。再在红框框处选择。

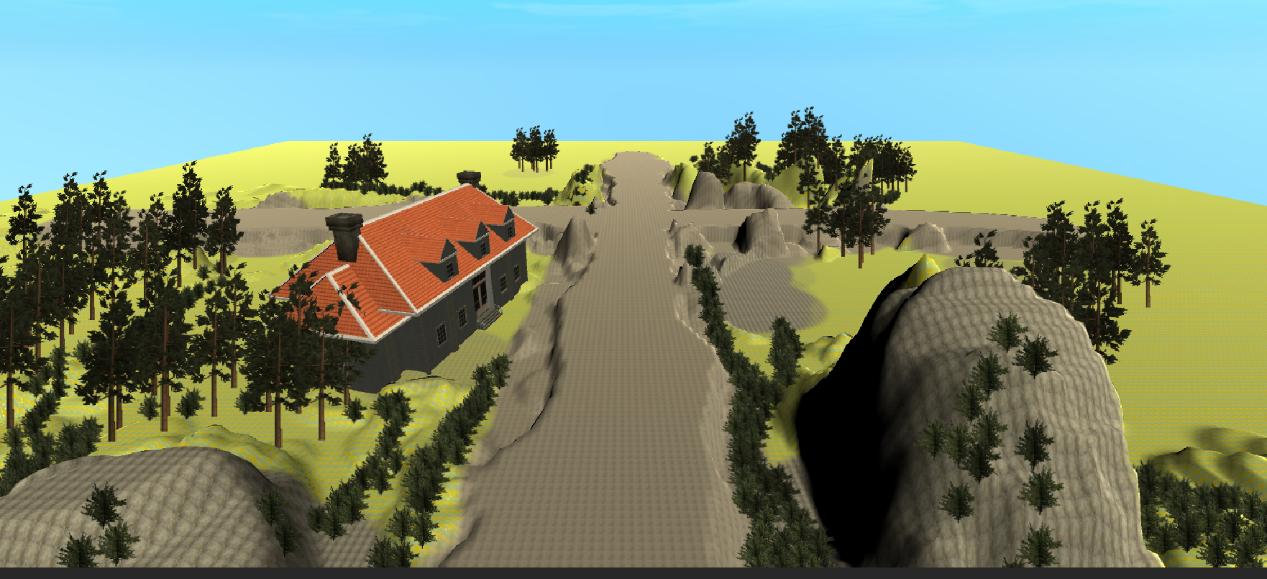
scene里面添加skybox什么时候都可以看到，camera只有在该摄像机里面可以看到。

新版：

在 window菜单下的lighting，找到Environment选项，天空盒和雾效等等都在里面



c. 在场景中添加地形并画出路、草、树、房子。这些材料都可以在AssetStore中找到。



\*\*ps\*\*：理解天空盒子（Understanding skybox）

天空盒是一个全景视图，分为六个纹理，表示沿主轴（上，下，左，右，前，后）可见的六个方向的视图。如果天空盒被正确的生成，纹理图像会在边缘无缝地拼接在一起，可以在内部的任何方向看到周围连续的图像。全景图片会被渲染在场景中的所有其他物体后面，并旋转以匹配相机的当前方向（它不会随着相机的位置而变化，而照相机的位置总是位于全景图的中心）。因此，天空盒子是在图形硬件上以最小负载向场景添加现实性的简单方式。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200929151302790.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3FxXzQzNzE5NDM3,size\_16,color\_FFFFFF,t\_70#pic\_center)

3. 写一个简单的总结，总结游戏对象的使用

Empty：空对象，常用作载体，可用来挂载脚本或作为其他游戏对象的父类

3D Object：三维实体对象，有几何属性。游戏可见的对象之一

2D Object：二维实体对象，有几何属性，游戏中可见的对象之一

Light：光源，用来营造光影效果

Audio：音频，用以创造游戏音效

Video：视频

Camera：摄像机，用以从某个视角观察游戏世界

游戏对象可通过事先创建，脚本创建，加载预设等方式进行创建，通过改变其属性实现游戏场景的切换。

游戏对象包括常规的3D和2D对象，还包括摄像机和光源等这种特殊对象。每个对象都有自己的相关属性（如材质，位置等等），还可以附加component（脚本等），对象之前也可以有附属关系，对象还可以通过脚本来创建。

## 二、编程实践（二选一）

### 牧师与魔鬼 动作分离版

【2019开始的新要求】：设计一个裁判类，当游戏达到结束条件时，通知场景控制器游戏结束

\*\*动作分离的原因\*\*：场记（控制器）管的事太多，不仅要处理用户交互事件，还要游戏对象加载、游戏规则实现、运动实现等等，而显得非常臃肿。一个最直观的想法就是让更多的人（角色）来管理不同方面的工作。显然，这就是面向对象基于职责的思考，例如：用专用的对象来管理运动，专用的对象管理播放视频，专用的对象管理规则。就和踢足球一样，自己踢5人场，一个裁判就够了，如果是国际比赛，就需要主裁判、边裁判、电子裁判等角色通过消息协同来完成更为复杂的工作。

```

该版本改进的目的:

把每个需要移动的游戏对象的移动方法提取出来，建立一个动作管理器来管理不同的移动方法。

根据原先版本，每一个可移动的游戏对象的组件都有一个Move脚本，当游戏对象需要移动时候，游戏对象自己调用Move脚本中的方法让自己移动。

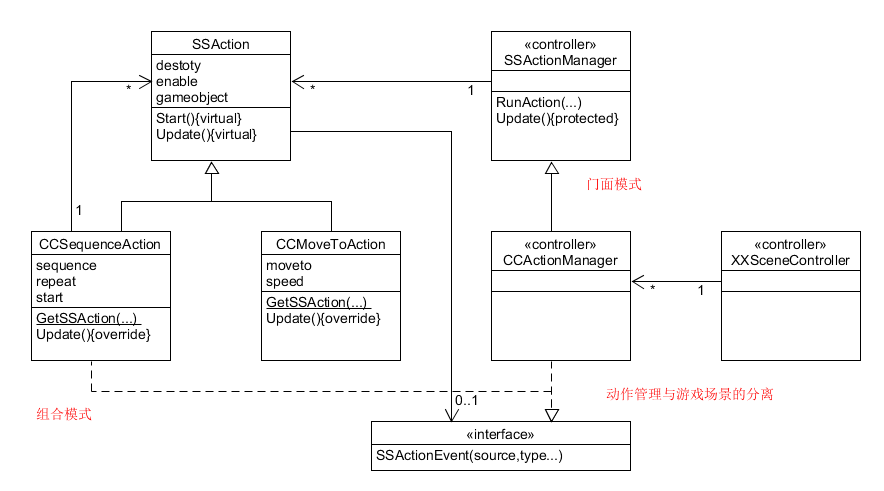
而动作分离版，则剥夺了游戏对象自己调用动作的能力，建立一个动作管理器，通过场景控制器(在我的游戏中是Controllor)把需要移动的游戏对象传递给动作管理器，让动作管理器去移动游戏对象。

当动作很多或是需要做同样动作的游戏对象很多的时候，使用动作管理器可以让动作很容易管理，也提高了代码复用性。

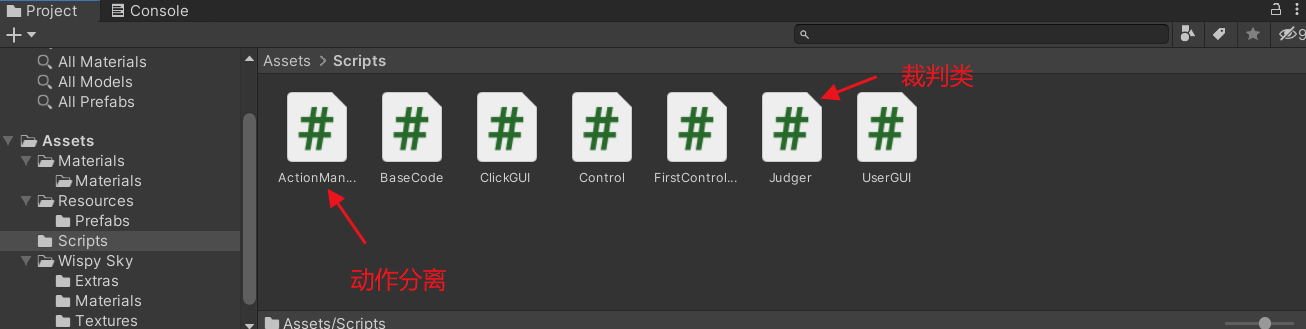
```

以下版本与上一篇版本有一些不同（原先版本我只参考了一位师兄的博客，后来再次复习老师网页上的内容时，发现在类等设置上有一些出入，所以重做了一版，同时知道了如何从商店下载模型，所以模型也有改动，同时省去了一些功能），以下只讲动作分离和裁判类，其余请参考代码。

以下是老师的UML图：



新增一个ActionManager脚本和一个Judger裁判类，将UserGUI中判断游戏是否结束以及结束时的操作全部转移给Judger，将游戏中控制小船和角色移动的动作全部改为actionManager的操作（对BaseCode.cs,Control.cs,Firstcontrol.cs需要进行一点修改）。



### ActionManage

```javascript

public class SSAction : ScriptableObject

{

public bool enable = true;

public bool destroy = false;

public GameObject GameObject { get; set; }

public Transform Transform { get; set; }

public ISSActionCallback Callback { get; set; }

protected SSAction() { }

// Use this for initialization

public virtual void Start()

{

throw new System.NotImplementedException();

}

// Update is called once per frame

public virtual void Update()

{

throw new System.NotImplementedException();

}

}

public class SSMoveToAction : SSAction{

public Vector3 target; // 移动目标

public float speed; // 移动速度

// 创建并返回动作的实例

public static SSMoveToAction GetSSMoveToAction(Vector3 target, float speed){

SSMoveToAction action = ScriptableObject.CreateInstance<SSMoveToAction>();

action.target = target;

action.speed = speed;

return action;

}

public override void Start(){

}

// 在 Update 函数中用 Vector3.MoveTowards 实现直线运动

public override void Update(){

this.Transform.position = Vector3.MoveTowards(this.Transform.position, target, speed \* Time.deltaTime);

if (this.Transform.position == target)

{

this.destroy = true;

// 完成动作后进行动作回掉

this.Callback.SSActionEvent(this);

}

}

}

public class SSSequenceAction : SSAction, ISSActionCallback{

public List<SSAction> sequence;

public int repeat = -1;

public int start = 0;

// 创建并返回动作序列的实例

public static SSSequenceAction GetSSSequenceAction(int repeat, int start, List<SSAction> sequence){

SSSequenceAction action = ScriptableObject.CreateInstance<SSSequenceAction>();

action.repeat = repeat;

action.sequence = sequence;

action.start = start;

return action;

}

// 在 Update 中执行当前动作

public override void Update(){

if (sequence.Count == 0) return;

if (start < sequence.Count)

{

sequence[start].Update();

}

}

// 更新当前执行的动作

public void SSActionEvent(SSAction source, SSActionEventType events = SSActionEventType.Complete,

int intParam = 0, string strParam = null, Object objectParam = null) {

source.destroy = false;

this.start++;

if (this.start >= sequence.Count)

{

this.start = 0;

if (repeat > 0) repeat--;

if (repeat == 0)

{

this.destroy = true;

this.Callback.SSActionEvent(this);

}

}

}

// Use this for initialization

public override void Start(){

foreach (SSAction action in sequence)

{

action.GameObject = this.GameObject;

action.Transform = this.Transform;

action.Callback = this;

action.Start();

}

}

// 执行完毕后销毁动作

void OnDestroy()

{

foreach (SSAction action in sequence)

{

DestroyObject(action);

}

}

}

public class SSActionManager : MonoBehaviour{

private Dictionary<int, SSAction> actions = new Dictionary<int, SSAction>();

private List<SSAction> waitingAdd = new List<SSAction>();

private List<int> waitingDelete = new List<int>();

protected void Update(){

foreach (SSAction action in waitingAdd){

actions[action.GetInstanceID()] = action;

}

waitingAdd.Clear();

foreach (KeyValuePair<int, SSAction> KeyValue in actions){

SSAction action = KeyValue.Value;

if (action.destroy){

waitingDelete.Add(action.GetInstanceID()); // release action

}

else if (action.enable){

action.Update(); // update action

}

}

foreach (int key in waitingDelete){

SSAction action = actions[key];

actions.Remove(key);

DestroyObject(action);

}

waitingDelete.Clear();

}

protected void Start(){

}

// 执行动作

public void RunAction(GameObject gameObject, SSAction action, ISSActionCallback callback){

action.GameObject = gameObject;

action.Transform = gameObject.transform;

action.Callback = callback;

waitingAdd.Add(action);

action.Start();

}

public void SSActionEvent(SSAction source, SSActionEventType events = SSActionEventType.Complete,

int intParam = 0, string strParam = null, Object objectParam = null){

}

}

public class ActionManager : SSActionManager, ISSActionCallback

{

private SSMoveToAction boat\_move;

private SSSequenceAction char\_move;

public FirstController sceneController;

protected new void Start()

{

sceneController = (FirstController)Director.getInstance().currentSceneController;

sceneController.actionManager = this;

}

public void moveBoat(BoatController boatCtrl, Vector3 dest, float speed)

{

boat\_move = SSMoveToAction.GetSSMoveToAction(dest, speed);

this.RunAction(boatCtrl.getGameobj(), boat\_move, this);

}

public void moveChar(MyCharacterController charCtrl,Vector3 dest, float speed)

{

//moveable.setDestion()的功能

Vector3 middle = dest;

Vector3 chara = charCtrl.getGameobj().transform.position;

if (dest.y < chara.y) { // character from bank to boat，说明角色在岸上，可以上船

middle.y = chara.y;

}else { // character from boat to bank 说明角色在船上，可以上岸

middle.x = chara.x;

}

SSAction action1 = SSMoveToAction.GetSSMoveToAction(middle, speed);

SSAction action2 = SSMoveToAction.GetSSMoveToAction(dest, speed);

char\_move = SSSequenceAction.GetSSSequenceAction (1, 0, new List<SSAction> { action1, action2 });

this.RunAction(charCtrl.getGameobj(), char\_move, this);

}

}

```

### Judger

```javascript

public class Judger : MonoBehaviour{

void Start(){

}

public void judge(UserGUI userGUI,GUIStyle style,GUIStyle buttonStyle){

if (userGUI.status == 1) {

GUI.Label(new Rect(Screen.width/2-50, Screen.height/2-85, 100, 50), "You Lose!", style);

if (GUI.Button(new Rect(Screen.width/2-70, Screen.height/2, 140, 70), "Restart", buttonStyle)) {

userGUI.status = 0;

userGUI.action.restart ();

}

} else if(userGUI.status == 2) {

GUI.Label(new Rect(Screen.width/2-50, Screen.height/2-85, 100, 50), "You win!", style);

if (GUI.Button(new Rect(Screen.width/2-70, Screen.height/2, 140, 70), "Restart", buttonStyle)) {

userGUI.status = 0;

userGUI.action.restart ();

}

}

}

}

```

结果图：

