# 0.监听事件

|  |
| --- |
|  |
|  |

# 1.监听触摸事件

|  |
| --- |
|  |
|  |

# 2.监听键盘事件

|  |
| --- |
|  |

## 要知道那个按键被按下，最好的方法就是使用macro.KEY,它里面什么按键都有，都是以按键的名字来命名的非常方便

|  |
| --- |
|  |
|  |
|  |

# 3.监听重力感应

|  |
| --- |
|  |

## 监听重力感应需要一个acc参数，他的类型如下

|  |
| --- |
|  |

## 如果你不再需要监听某一个事件，你可以使用systemEvent.off方法来取消监听

|  |
| --- |
|  |

## 还有，我们需要在删除一个节点的时候在他的onDestroy方法里面添加关闭监听的代码

|  |
| --- |
|  |

# 4.生成预制体与预制体实例化

## 所谓的预制体，其实就相当于一个模板，而且他可以保存到系统中下一次有需要还可以继续使用

|  |
| --- |
|  |

# 5.演练

# 先来学习事件监听

## 1.新建一个项目，起名classNine-event-and-prefab,然后在assets文件夹里面新建3个文件夹，AssetsPackage，Scenes和Scripts，然后把当前场景起名main.scene并且保存到Scenes文件夹里面

|  |
| --- |
|  |

## 2.在场景在场景一个立方体和一个球体

|  |
| --- |
|  |

## 3.新建一个GameMgr脚本并且添加到Cube节点上

|  |
| --- |
|  |

## 4.双击这个GameMgr脚本用vscode打开，然后把一些多余的内容删除，然后先导入SystemEvent类和systemEvent系统双击实例对象

|  |
| --- |
|  |

## 5.然后我们来调用systemEvent对象的监听方来监听触摸事件，（其实我们用鼠标点击页面就相当于触摸了屏幕）

|  |
| --- |
| import { \_decorator, Component, EventTouch, Node,SystemEvent,systemEvent, SystemEventType } from 'cc';  const { ccclass, property } = \_decorator;  @ccclass('GameMgr')  export class GameMgr extends Component {          start () {          //监听触摸事件          systemEvent.on(SystemEventType.TOUCH\_START,this.onTouchStart,this)      }      //定义回调函数      private onTouchStart(touch:Touch,event:EventTouch):void{         console.log("Touch Started...");         console.log("touch:",touch);         console.log("event:,",event);        }      // update (dt: number) {        // }  } |

### 效果

|  |
| --- |
|  |

## 6.还可以监听触摸弹起，然后需要添加处理触摸弹起的回调函数

|  |
| --- |
| import { \_decorator, Component, EventTouch, Node,SystemEvent,systemEvent, SystemEventType } from 'cc';  const { ccclass, property } = \_decorator;  @ccclass('GameMgr')  export class GameMgr extends Component {          start () {          //监听触摸事件          //触摸开始          systemEvent.on(SystemEventType.TOUCH\_START,this.onTouchStart,this)          //触摸弹起          systemEvent.on(SystemEventType.TOUCH\_END,this.onTouchEnd,this)        }      //定义回调函数      private onTouchStart(touch:Touch,event:EventTouch):void{         console.log("Touch Started...");      //    console.log("touch:",touch);      //    console.log("event:,",event);        }      private onTouchEnd(touch:Touch,event:EventTouch):void{         console.log("Touch Ended...");      //    console.log("touch:",touch);      //    console.log("event:,",event);        }      // update (dt: number) {        // }  } |

### 7.还可以添加触摸移动事件监听, 注意：需要把cc.Touch导入进来，否则会有很多红色波浪线

|  |
| --- |
| import { \_decorator, Component, EventTouch, Node,SystemEvent,systemEvent, SystemEventType,Touch } from 'cc';  const { ccclass, property } = \_decorator;  @ccclass('GameMgr')  export class GameMgr extends Component {          start () {          //监听触摸事件          //触摸开始          systemEvent.on(SystemEventType.TOUCH\_START,this.onTouchStart,this)          //触摸移动          systemEvent.on(SystemEventType.TOUCH\_MOVE,this.onTouchMove,this)          //触摸弹起          systemEvent.on(SystemEventType.TOUCH\_END,this.onTouchEnd,this)        }      //定义回调函数      private onTouchStart(touch:Touch,event:EventTouch):void{         console.log("Touch Started...");      //    console.log("touch:",touch);      //    console.log("event:,",event);        }      private onTouchEnd(touch:Touch,event:EventTouch):void{         console.log("Touch Ended...");      //    console.log("touch:",touch);      //    console.log("event:,",event);        }      private onTouchMove(touch:Touch,event:EventTouch):void{         console.log("Touch moved...");      //    console.log("touch:",touch);      //    console.log("event:,",event);        }      // update (dt: number) {        // }  } |

### 效果，如果只是点击不移动就只会触发触摸开始和触摸结束

|  |
| --- |
|  |

### 如果按下鼠标左键不送，一直移动是这样子的效果

|  |
| --- |
|  |

### 结束后又会触发触摸结束，注意，还有一个触摸取消事件，这个事件在全局状态下面不会触发

## 8.如果需要获取触摸位置的坐标，可以从touch参数获得，我们可以查看一些Touch类型的成员

|  |
| --- |
|  |

### 注意：如果你没有导入Touch类，编译器找到的是Touch接口，这样子是不对的。使用需要导入Touch类

### 我们可以输出一下这个touch对象，发现他有一个getLocation方法

|  |
| --- |
|  |

## 9.我们使用他的这个方法来获取一下触摸坐标

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

### 注意：这个触摸坐标是以屏幕的左下角维原点的，这个和屏幕坐标不一样

## 10.这个touch对象还有一个getDelta方法可以获取本次点击的点和上一次点击的点的距离

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

### 这个getDleta是非常有用的，在游戏中经常用到。了EventTouch类有2个比较有用的方法：getTouchs()和getEventCode()

|  |
| --- |
|  |

## 11.下面我们来监听键盘事件，有按键按下和按键弹起事件代码如下，它的回调函数有一个event参数是EventKeyboard类型，开源获取按下的按键的keyCode

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

## 12.然后我们就可以根据keyCode来获取到底按了哪个按键，代码如下

|  |
| --- |
|  |

### 运行程序，效果如下

|  |
| --- |
|  |

### 在游戏中，我们就可以根据用户按下的按键来控制游戏角色的移动

## 13.当然，使用数字并不好用，很难记忆，使用cocos creator定义了一个macro里面有一个KEY宏，使用它就比较方便，我们可以当然macro然后使用他的KEY里面的按键来代替keyCode

|  |
| --- |
|  |

### 运行程序，效果如下

|  |
| --- |
|  |

## 14.然后我们来学习监听重力感应事件，其实就是监听 SystemEventType.DEVICEMOTION，代码如下，不过测试还是比较困难

|  |
| --- |
|  |

## 15.可以监听事件就可以取消监听事件，如我们来取消触摸弹起事件，使用systemEvent.off方法，参数必须和对应的on方法的参数完全一样，如果不一样可能会误删除一些监听功能

|  |
| --- |
|  |

### 运行程序，效果如下

|  |
| --- |
|  |

### 此时就看不见然后Touch ended。。。的输出了

## 16.当然其实我们取消监听是在onDestroy函数里面写是比较好的，我们需要在onDestroy函数里面把我们设置的使用事件监听全部关闭，这样子当一个节点被删除了，他的事件监听也一并删除，否则，节点删除了，事件监听还在这是不对的。

|  |
| --- |
|  |

# 下面我们来演练预制体

# 通过界面操作删除预制体

## 1.先把Sphere拖到Cube里面成为他的孩子

|  |
| --- |
|  |

## 2生成预制体很简单，比如我们想把这个Cube，注意他有一个Sphere孩子，做成预制体，我们只需要把他拖拽到AssetsPackage里面，就可以生成一个预制体，我们可以看到场景中的Cube和Sphere变成绿色了。在AssetsPackage里面也应该Cube

|  |
| --- |
|  |

## 3.我们先吧场景中的Cube删除

|  |
| --- |
|  |

## 4.然后把AssetsPackage里面的Cube往Scene一拖，又会在场景在生成应该Cube，此时他还是绿色的

|  |
| --- |
|  |

## 5.然后我们再拖拽一次，又会生成另外应该Cube

|  |
| --- |
|  |

## 6.预制体其实是一个资源文件，在预制体上面点击右键-》在资源管理器打开，就会看到Cube.prefab,然后我们可以用一个文本编辑器把他打开

|  |
| --- |
|  |

## 7.通过把预制体拖放的场景中生成的节点是绿色的，因为他和普通节点不一样，他关联了预制体，你点击这个节点，发现他多了一个预制体选项，而且你是可以取消关联的

|  |
| --- |
|  |

## 8.当你点击了这个取消关联按钮后，它就不再是绿色的，说明他此时已经和预制体没有关联了，此时你在点击这个节点，发现他已经没有预制体选项了。

|  |
| --- |
|  |
|  |

## 9.预制体选项的第二个按钮可以查看节点和那个预制体关联，如我们选中绿色的Cube点击这个按钮

|  |
| --- |
|  |

### 当你这个按钮，节点关联的预制体会抖动。这样子你就可以快速找到关联预制体

## 10，预制体选项的第三个按钮就是从资源还原，比如我们给绿色的Cube节点添加一些孩子

|  |
| --- |
|  |

## 11.此时如果我有不需要这些孩子，此时我们就可以点击从资源还原按钮，把它还原回来

|  |
| --- |
|  |

### 不过注意，每一次之后回退一步，如果有多个孩子需要点击多次，这样子其实还是有好处的，点击四次后还原到最初状态

|  |
| --- |
|  |

## 12.预制体选项里面的最后一个按钮是更新到资源也就是用现在的结构来修改预制体的结构。为了安全，我们先给预制体创建一个副本。

|  |
| --- |
|  |

## 13.然后我们来修改节点结构，如图

|  |
| --- |
|  |

## 14.然后我们来点击最后一个按钮，效果如下

|  |
| --- |
|  |
|  |

## 15.此时你会发现，新建的节点也变成绿色的了，说明他们也和预制体关联上了，我们把他们的位置调整一下，便于观察

|  |
| --- |
|  |

## 16，然后我们把那个普通的Cube节点删除，拖拽预制体到场景，发现他就是我们修改后的结构，效果如下

|  |
| --- |
|  |

### 注意，如果在你更新预制体之前，场景中有多个节点和预制体有关联，当你更新了预制体后，这些绿色的节点也会自动更新到最新状态

# 通过代码来生成预制体

## 0.先把所有节点和预制体的关联取消

|  |
| --- |
|  |

## 1.需要在代码中先获取预制体的资源，有2种方法，第一种是编辑器绑定资源，第二种是代码加载资源，我们先使用第一种方法，我们在Scripts文件夹里面新建一个GamePrefabGen脚本

|  |
| --- |
|  |

## 2.为了方便学习，我们使用Cube001预制体

|  |
| --- |
|  |

## 3.把这个新脚本拖拽到摄像机节点上面

|  |
| --- |
|  |

## 4.双击这个脚本用vscode打开，然后把多余的内容删除

|  |
| --- |
|  |

## 5.给我们的组件类添加一个数据成员，然后给他添加@property装饰器，这样子他就会出现在编辑器上面

|  |
| --- |
|  |

## 6.此时查看一下编辑器，发现摄像机节点多了一个CubePrefab属性

|  |
| --- |
|  |

## 7.然后我们需要把Cube001预制体拖拽到这个CubePrefab上面

|  |
| --- |
|  |
|  |

## 8.然后，我们就可以在代码中使用这个预制体了，需要先使用instantiate函数来实例化预制体，注意，实例化后需要添加到场景树中，还需要调整位置的坐标

|  |
| --- |
|  |

### 运行程序，修改如下，此时场景中就有3组节点

|  |
| --- |
|  |

## 9.在实际开发中，我们可以把很多界面做成预制体，然后在代码中使用他们来场景节点

# 预制体是非常有用的，其实我们可以到一些开源的游戏项目里面获取预制体，然后我们在我们的项目中使用这些预制体。

# 作业

|  |
| --- |
|  |