一、框架Framework部分:

1、单例模式：

继承于Monobehaviour的单例模式。便于协程的启动。

2、管理器Manager（易移植到其他项目）:

ResourceManager：

要换成AB包加载，现在的打包不了。

SceneLoadManager：场景加载管理器，用于场景加载，异步加载等待加载完成后加载场景信息。

ConfigManager：配置表管理器，用于加载并缓存静态配置表文件，csv格式，用来存储游戏内的复杂数值。

ControllerManager：控制器管理器，MVC组件，不好用，功能与具体游戏事件逻辑所在位置冲突，下个demo打算删掉。

ViewManager：视图管理器，使用的是UGUI，分为两种UI，一种是占全屏幕，长时间存在的页面，使用栈进行层级管理，关闭为返回逻辑，跳转为实例化新的物体逻辑。另一种是频繁开关的页面，生成一次，使用显示和隐藏逻辑。

  
第一种类型



第二种类型

3、MVC框架（只留View）:

本应该在控制器里实现游戏逻辑，最后全移到GameplayManager里了，只留下了UI事件的订阅和取消订阅。

4、状态机：

描述整个游戏所处的位置（见StateManager）

5、工具：

计算2DVector和3DVector转换，2D距离等。

射线检测区分UI和物体等。

二、Gameplay部分

1、管理器Manager（不容易移植到其他项目）:

StateManager:状态管理器，基于协程的状态机，事件驱动跳转，该demo仅用于确认状态供其他类使用，如，下述InputManger仅在游戏页面才生效。

InputManger：输入管理器，该demo主要事件为点击事件，主要实现了一个click和unclick的逻辑。主要用来控制建造页面和升级页面的显示。



GameplayManager：游戏逻辑管理器，除本身的血量，金钱等内容的管理，还在其中私有定义了一些下属管理器。

WaveManager：波次管理器，用协程处理敌人波次及胜利判定。

BuildManager：建造管理器，用来处理建造相关逻辑。

2、事件驱动：

声明静态全局事件，用来在运行中触发UI(打开关闭)，Scenes(场景加载),Game(游戏内逻辑)事件。

3、敌人

敌人使用了简单的有限状态机，在各状态中手动跳转，目前逻辑简单，只用移动和死亡两个阶段及动画，不乱，在另一个比较demo中使用混乱，后续考虑换成事件驱动的。

状态：

Run：沿着场景中的锚点行走，到达终点后销毁扣除玩家血量。

Die：生命值为0时，禁用Collider（用于处理于塔的交互逻辑），播放死亡动画，销毁。

4、塔

火球塔:

逻辑：碰撞检测，范围有目标按攻击间隔生成火球。

火球：追踪，转向，检测到敌人Collier调用敌人的受伤逻辑。若目标更换或目标Collider组件disable，飞行至上一帧记录位置附近后销毁。

激光塔：

逻辑：碰撞检测，范围有目标设置激光终止位置及显示激光。

激光：按造成伤害的间隔，敌人死亡后转火或隐藏。

5、数据相关

固定复杂数据：csv，格式容易出问题，定型后转成txt

设计数据，如关卡信息等，ScriptableObject







