# Object Management

## 1. Persisting Objects

- 用List追踪创建的物体
- 在Application.persistentDataPath目录下，保存游戏数据，游戏内“存档”，“读档”。
- 包装读写流，自定义底层数据的保存方式

## 2. Object Variety

- ScriptableObject配置数据，[CreateAssetMenu]
- 工厂类包装生成类型
  - 不能使用构造函数时，可以这样保护只读的变量
    - public int ShapeId {
      get {
      return shapeId;
      }
      set {
      if (shapeId == int.MinValue && value != int.MinValue) {
      shapeId = value;
      }
      else {
      Debug.LogError("Not allowed to change shapeId.");
      }
      }
      }

      int shapeId = int.MinValue;
- 保存时使用“版本号”区分数据读写方式，使用新的读写方式要记得兼容旧的数据格式。
- 为了避免同样材质的物体，使用不同颜色时系统都创建一种材质，
  使用 MaterialPropertyBlock。
- GPU instancing

## 3. Reusing Objects

- 使用对象池控制物体的生成销毁，修改GameObject的active。
  不同对象不同的池。
- 销毁使用RemoveAtSwapBack提升效率

## 4. Multiple Scenes

- 创建的物体（激活与不激活）在场景根节点中，可能造成层级混乱，遍历代价增加的问题。
  1.可以使用创建在同一个节点下的方法，但是每次会对父节点更新，不推荐

1.可以使用创建在同一个节点下的方法，但是每次会对父节点更新，不推荐。
2.创建物体在新的场景中

- 场景在1帧内无法加载完毕，使用协程完成程序的加载激活。
  由于场景加载时间取决于内容，并且加载会造成卡顿，所以使用异步加载（也就是协程）。
  为避免多次加载场景，协程前后禁用开启当前组件。

```
IEnumerator LoadLevel () {
enabled = false ;
        yield return SceneManager.LoadSceneAsync(
            "Level 1", LoadSceneMode.Additive
        );

SceneManager.SetActiveScene(SceneManager.GetSceneByName("Level 1"));
enabled = true ;
    }
```

## 5. Spawn Zones

- 使用抽象类和继承，生成不同的生成区域

## 6. More Game State

- 可序列化文件可以用JSON保存读取
- 随机数可以保存种子Random.Seed和状态Random.Status

## 7. Configuring Shapes

- 相比每个物体都有一次Update更新状态，在一次Update中更新所有状态开销更小。

## 8. More Factories

- 明显不同的逻辑使用不同的工厂

## 9. Shape Behaviour...

- 条件编译