



存档系统

重点部分：

1. 存档路径和存档文件格式
2. 如何加载/存储数据
3. 游戏结束时如何保存数据

核心脚本：

1. ArchiveManager.cs
2. SaveManager.cs
3. Serialization_Dict.cs
4. Serialization_Vector.cs

保存存档的时机

1. 玩家在游戏场景中点击返回主菜单时
2. 游戏意外退出时

保存存档触发机制

说明：当需要保存存档时使用事件管理器触发保存游戏事件，游戏中需要保存的数据有时间数据、玩家数据（玩家位置、玩家属性）、地图数据（地图初始化数据、地图对象数据等）、背包数据、科技树数据。因此需要在不同的Manger中将对应的保存游戏方法添加到事件管理器中，例如在时间管理器TimeManager中添加一个OnGameSave方法，方法中调用的是存档管理器里<保存时间数据>，其他Manager中保存数据的方式类似

```
// 当游戏关闭时将时间数据写入磁盘即可
private void OnGameSave() {
```

```
ArchiveManager.Instance.SaveTimeData();  
}
```

删除存档的时机

1. 由于本游戏没有玩家复活机制，所以当玩家死亡时整个存档就坏掉了，此时需要删除当前存档开始新的游戏





数据存储管理器 SaveManager

说明：

1. Windows存储数据路径：C:\Users\zgx\AppData\LocalLow\DefaultCompany

 saveData	2023/10/26 16:26	文件夹
 setting	2023/9/24 3:00	文件夹
 Player	2023/10/26 16:26	文本文档
 Player-prev	2023/10/26 16:23	文本文档

名称	修改日期	类型
 0	2023/10/26 16:26	文件夹
 SaveMangerData	2023/10/26 16:26	文件

名称	修改日期	类型
 MainInventoryData	2023/10/26 16:26	文件
 Map_(8, 8)	2023/10/26 16:26	文件
 Map_(8, 9)	2023/10/26 16:26	文件
 Map_(8, 10)	2023/10/26 16:26	文件

初始化

1. 设置数据存储路径和配置存储路径，如果路径不存则优先创建对应目录

2. 初始化saveMangerData，记录当前存档id、最后存档时间

保存文件

1. 通过传入的保存文件路径创建FileStream对象， FileMode.OpenOrCreate
2. 以二进制的方式将传入的数据对象写入到FileStream中

加载文件

1. 判断传入的文件路径是否存在，如果不存在则返回
2. 创建一个FileStream， FileMode.Open
3. 将内容解码成对象 `T obj = (T)binaryFormatter.Deserialize(file);`
4. 释放FileStream使用的资源file.Dispose()
5. 返回解码后的对象

初始化存档管理器数据

1. 从存档路径下加载SaveManagerData文件<加载文件>，如果没有对应文件则先创建存档管理器数据并<更新存档管理器数据>

更新存档管理器数据

1. 直接将SaveManagerData保存到存档目录下<保存文件>

添加存档

1. 根据当前存档id创建一份新的存档数据，并将其添加到SaveManagerData中
2. 更新当前SaveManagerData中currentId += 1并<更新存档管理器数据>

获取存档

1. 根据传入的存档id再存档数据中进行查找，如果没有返回null否则返回对应存档信息（存档id和存档时间）

获取存档路径

1. 根据传入的存档Id<获取存档>，如果当前存档不存在则创建对应文件夹

清理所有存档

1. 遍历所有的存档管理器数据，通过当前存档id拼接成存档文件名，使用
`Directory.Delete` 删除存档文件
2. 清空存档管理器数据并<更新存档管理器文件>

设置数据缓存

说明：缓存字典格式 `Dictionary<int, Dictionary<string, object>> cacheDic`

1. 根据传入的存档Id、对象名以及对象数据判断当前字典是否含有对应KeyValue，如果没有则加到字典中

从某个存档中加载对象

1. 根据当前加载对象类型获得对应Name（`typeof(T).Name`），并根据当前存档Id去对应检查存档缓存字典中是否有对应结果
2. 如果没有则<获取存档路径>
3. 将文件名与路径拼接得到具体的文件路径然后<加载文件>
4. 将加载后的数据放到缓存字典中<设置数据缓存>

保存对象到某个存档中

1. 根据传入的存档Id找到对应存档路径
2. 根据传入的对象使用`GetType().Name`获取对象名，将存档路径和文件名拼接起来作为保存对象的文件名并将对象保存到文件夹里<保存文件>

3. 更新当前存档数据中的存档时间并<更新存档管理器数据>
4. <设置数据缓存>便于后面直接使用

存档管理器 ArchiveManger

说明：

1. 游戏目前只有一个存档，因此不涉及多存档问题
2. 游戏中需要保存的数据有：时间数据、玩家数据（玩家位置、玩家属性）、地图数据（地图初始化数据、地图对象数据等）、背包数据、科技树数据

加载存档数据

1. 查看存档管理器中是否包含存档数据，如果存在存档数据则获取存第一份数据作为游戏存档数据

清空存档数据

1. 调用数据存储管理器<清理所有存档>

保存时间数据

1. 调用数据存储管理器<保存对象到某个存档中>方法将时间数据保存到存档中

保存背包数据

1. 注意：仓库属于地图对象，因此仓库数据存放到了地图对象数据中
2. 调用数据存储管理器<保存对象到某个存档中>方法将背包数据保存到存档中

保存玩家位置数据

1. 注意：玩家位置和玩家属性保存时机有些不同，分开保存会比较好
2. 调用数据存储管理器<保存对象到某个存档中>方法将玩家位置数据保存到存档中

保存玩家属性数据

1. 注意：玩家位置和玩家属性保存时机有些不同，分开保存会比较好
2. 调用数据存储管理器<保存对象到某个存档中>方法将玩家属性数据保存到存档中

添加特殊地图对象类型数据

1. 根据传入的地图对象Id将地图对象数据添加到 `mapObjectTypeDataDict` 中

获取特殊地图对象类型数据

1. 根据传入的地图对象Id从特殊地图对象数据字典中拿出数据

删除特殊地图对象类型数据

1. 根据传入的地图对象Id从特殊地图对象数据字典中移除数据

保存特殊地图对象类型数据

1. 注意：有些特殊地图对象例如篝火、仓库、浆果灌木是建筑物中比较特殊的情况，它们都有一些特有属性，因此需要在建造时动态保存方便在恢复游戏或者第一次建造时保存读取，虽然也可以通过地图管理器中的MapObjectDataDict根据Id去获得数据，但在目前实现方法中并没有这样做
2. 调用数据存储管理器<保存对象到某个存档中>方法将玩家位置数据保存到存档中

保存地图数据

1. <保存特殊地图对象类型数据>
2. 调用数据存储管理器<保存对象到某个存档中>方法将整个地图数据（地图块索引）保存到存档中

保存单个地图块数据

1. 通过传入的地图块索引拼接出需要保存的地图块文件名，使用<保存对象到某个存档中>将地图块数据保存到当前存档数据中

获取单个地图块数据

1. 根据传入的地图块索引拼接出需要加载的地图块数据文件名，使用<加载文件>加载地图块数据

添加并保存单个地图块数据

1. 将传入的地图块索引添加到地图数据的地图块索引列表中便于后续查找
2. <保存地图数据>后再<保存单个地图块数据>

创建新存档

1. 清空当前存档数据<清理所有存档>
2. 添加一份新存档<添加存档>
3. 保存当前用户设定的地图初始化数据
4. 保存<保存玩家位置数据>和<保存玩家属性数据>，新存档中玩家初始位置为地图中央，玩家属性例如血量饱食度设定为最大值
5. 初始化地图数据并保存<保存地图数据>
6. 初始化背包数据并保存<保存背包数据>
7. 初始化时间数据并保存<保存时间数据>，其中时间初始化阶段为0、重置时间计数器
8. 初始化科技数据

加载当前存档

说明：数据已保存到本地了，直接使用LoadObject从本地数据文件中读取数据

1. 加载地图初始化数据、地图数据、特殊地图对象数据
2. 加载玩家位置、属性数据
3. 加载背包数据

4. 加载时间、科技树数据

可序列化字典 `Serialization_Dict`

说明：单独创建该类主要目的是C#字典无法序列化/反序列化，因此需要设置一个单独的类去存储游戏中包含字典的各类数据结构

重要属性：

1. `keyList`：字典所有key值
2. `valList`：字典所有value值
3. `dictionary`：字典反序列化后的中间临时数据，方便外部访问

序列化方法

注意：需要加上 `OnSerializing` 的标识

1. 遍历当前中间临时字典中的Key/Value，按照一一对应的顺序分别添加到 `keyList/valList`中

反序列化方法

注意：需要加上 `OnDeserialized` 的标识

1. 遍历当前`keyList`中的结果，按照`keyList/valList`元素一一对应的顺序将Key/Value加入到中间临时字典中

可序列化向量 `Serialization_Vector2/Serialization_Vector3`

说明：

1. `vector2/vector3`方法类似，类中仅包含x, y, z三个变量，由于该类中没有无法序列化的类型，因此可以直接使用 `binaryFormatter.Serialize` 写到文件中