高塔与藏书者_技术文档

by 232studio 2024.11.19

一、游戏概况

- 开发环境：Unity 2022.3.34f1c1
- 游戏平台：PC
- 版本号：v1.919810
- 项目托管地址：https://github.com/tmszzzzz/Project451 （将在2024.11.21及以后开放为public可见）

二、游戏基本逻辑

此处作简单介绍以便后续的描述，更详细的运行逻辑可以参考策划案。

游戏的基本类型可以被概括为“回合制策略游戏”，以回合为主要单位进行游戏逻辑的运行。

可交互的游戏部分主要的组成有两种：“节点”和“连接线”，他们是“（潜在）成员”和“人际关系”的抽象。“书”是最基本的游戏资源，它仅被节点所持有与转移。

“节点”、“书”、“连接线”是游戏基本逻辑的构成部分。

节点拥有状态，分别是未加入、已加入、暴露。游戏的全局目标是将初始未加入的所有节点转变为已加入。状态可以发生变更，其根本的判定标准是”受影响力“，这是属于节点的属性，它的值恒等于一个节点与它的所有邻节点所持有的书的总和。一个节点的邻节点是所有与这个节点使用连接线相连的节点。

每个节点有三个阈值属性：转化、暴露、维持阈值，简单来说，影响力与这些阈值的大小关系产生变化时便会导致状态的变更。

整个游戏部分的展现十分类似于人际关系网络，在游戏背景上它也确实代表着这种含义。

转变节点状态是游戏的基本目标，而状态依赖于受影响力，受影响力又完全取决于节点所持有的书的情况，所以对”书“的操作管理是游戏核心玩法。每一回合，我们可以依一定的规律或限制，在诸多节点之间创建书的分配流，以达成对不同节点的状态管控。每次书的分配与节点的转变是以回合为单位的，也就是说节点的受影响力发生任何变化时，其产生的状态变更都需要下一回合才生效。

存在全局的属性”暴露值“，在一些情况下这个值将会增长，例如，每个暴露节点每回合会增加一定的暴露值。暴露值满则游戏失败。

存在一些特殊节点：藏书者首次加入将会提供书；消防员在加入后开始每回合提供书；关键人物则与游戏主流程相链接，是游戏进程推进的基础与重要目标。

存在需要被解锁的连接线：它们在吸纳并维持一定数量的关键人物时才会开启。

游戏进行一定回合后，出现附加规则”看火者“：他们在不同节点之间随机游走，仅在回合切换时统一地展示此回合所在的位置，随后便继续移动一次。这意味着我们无法得知他们的具体位置，只能根据一些信息进行推测。他们监控当前所在位置的书籍流转进出，一旦被监控则会增长较多的暴露值。

每条连接线会有一定概率指示其两侧看火者的数量信息。

随着每一回合的推进，众多节点不断在各种状态中转化，使用”书“管理平衡节点状态，保持吸纳新成员的同时又不致暴露，最终完成整个场景的转化，从而达成游戏胜利。

游戏有配套的自制剧情系统，随游戏进程不断揭示游戏相关背景故事。

游戏有自制任务系统，给予基本的目标引导。

游戏提供符合世界观的文字教程（设定上为系统操作手册）以及更简略直观的基于timeline的动态教程。

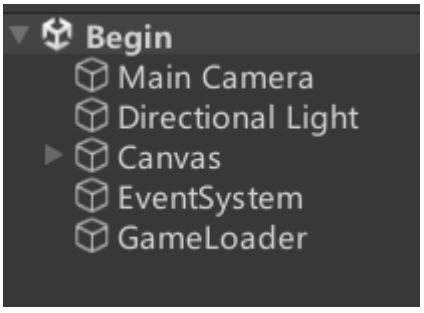
游戏有自制序列化json存档系统，支持随时退出与继续而不影响进度。

存在基于基本存档系统的自动保存应急恢复存档，仅为出现影响游戏继续的重大恶性bug时提供，需要手动替换存档文件。虽然提供了此系统，但经多次测试与修改，目前已经没有已知的此类恶性bug待修复。因此这个应急恢复系统只是作为最后的保障。

二、场景架构

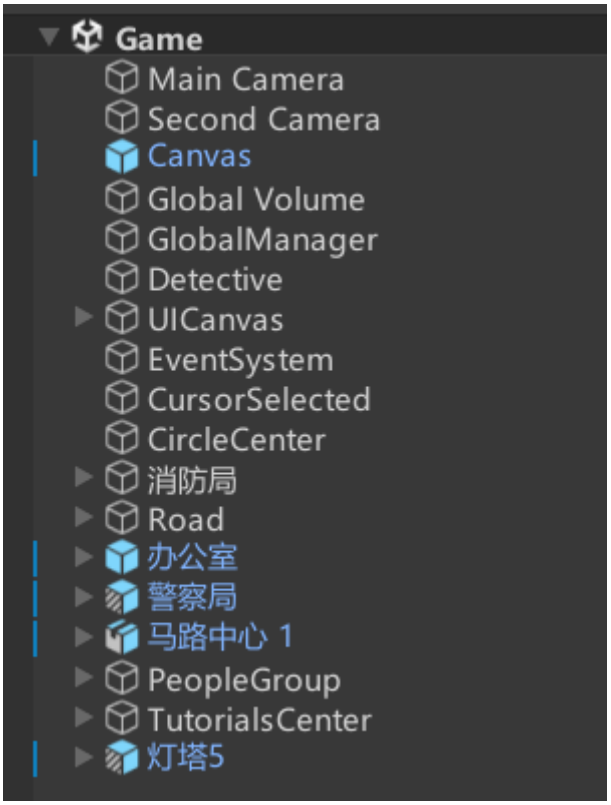
共3个游戏场景：

- Begin - 游戏主页面，管理进入游戏的入口。



其中，GameLoader在开始游戏后进入DontDestroyOnLoad，在整个游戏周期中存在。它承担从开始页面携带信息加载进入游戏主场景的职责，同时也作为场景切换时数据传达的中转。

- Game - 游戏主场景，整个游戏流程的绝大部分运行场景。



  - Canvas - 并非UGUI的Canvas组件，此处的Canvas是所有节点（后文可能称为node）和连接（后文可能称为connection）物体的共同父级。它搭载的脚本持有所有node、所有connection的引用并开放数个接口供外部获取相关信息。同时它也分管从json文件中加载所有node和connection的初始状态、node的状态转换、connection的指示更新等控制信号的发

出。总而言之，Canvas是统筹所有node和connection物体的中枢管理器，其之所以存在是因为整个场景的节点、连接数可能达到数百之多，需要一个共同父级以处理它们之间的关联与数据流。

- GlobalManager - 搭载了所有单例脚本组件的空物体，作为全局的数个模块的管理器存在。其搭载的脚本均以xxManager的形式命名，这代表这些脚本分管其相关模块的全局管理器职责，且只需要存在实例而不需要一个依附确定的游戏物体便可单独发挥效用。其包含的模块包括但不限于如下的数个部分：
  - GlobalVar - 唯一一个不以Manager结尾的脚本，它搭载了所有游戏进程中使用的全局变量，且可以随游戏进程发生变更。
  - RoundManager - 轮次调度管理器，其决定了一次轮次转换中的所有行为以及它们之间的先后顺序。
  - PlotManager - 剧情推送管理器，其解析本地的剧情文件并依此与剧情ui侧交互以展示剧情。
  - etc.

  这部分会在程序架构一节详述。
- Detective - 所有看火者的父级。与Canvas相似，由于看火者也是在场上大量存在的物体，所以同样也需要一个共同父级去管理其逻辑。不过，看火者仅有行为而不持有属性，这在面向对象设计中不应当作为一个单独的对象，所以在Detective物体中我们统一地控制它们的行为，统一地执行它们的逻辑判断。
- UICanvas - UGUI的Canvas组件，管理全部2dUI。
- TutorialsCenter - 全部教程物体的父级。教程展示基于timeline，timeline在此物体下进行展示。
- 其余未提及的部分或较为显然，或仅作为静态场景而存在。

- End - 游戏结束场景，在达成胜利或失败后会由GameLoader搭载相关信息并加载到此场景，由此场景展示游戏结果。此场景仅包含非常基础的显示效果，以及按esc回到主页面的最基本的交互，故不再过多叙述了。

三、程序脚本组织架构

下图为项目脚本文件的基本构成：

```
Assets/Scripts
├——General - 通用脚本，表示可被不同部分使用的单独的简单操作模块。例如
DestroyMyselfInAnimator
├——InGame - 游戏运行时脚本
|   ├——Behavior - 表示不同对象的行为操作
|   |   ├——Connections - 基本connection行为及其扩展类
|   |   └——Nodes - 基本node行为及其扩展类
|   ├——Manager - 表示管理器的行为操作
|   ├——ScriptableObject - 使用ScriptableObject实现某些配置文件的管理
|   └——UI - 游戏运行时的ui部分逻辑
|       ├——2dUI - 游戏运行时的2dUI逻辑
|       |   ├——Book - 游戏中“菲尼克斯的笔记”ui元素的控制逻辑
|       |   ├——PlotUI - 剧情ui的控制逻辑
|       |   ├——Quests - 任务ui的控制逻辑
|       |   └——TextDisplay - 一些杂项文字显示效果
|       └——inGameUI - 渲染于3d场景内的ui显示逻辑
├——MapEditor - 初期编辑地图使用的MapEditor场景内物体专属脚本，中期后已不再提供支持
```

```
└——Prelude - 主页面脚本的雏形，已全部弃用
```

四、程序模块概述

- **游戏基础逻辑模块**

  我们把节点、连接、书的流转三部分以及相关的转化规则、游戏的基本进程管理划分为游戏的基础逻辑。

  我们提到节点有其拓展类型，连接也有需要解锁的特殊种类，这部分脚本使用了面向对象的程序设计。在node部分，基本的接口于BaseNodeBehavior脚本中声明，基本的逻辑大量地在NodeBehavior内实现，而后特殊的三种node继承NodeBehavior并实现自身特殊的逻辑。在connection部分，UnlockableConnectionBehavior继承了基类ConnectionBehavior并重写了关于邻居判断的逻辑，ConnectionBehavior提供了一个统一接口供UnlockableConnectionBehavior重写关于连接解锁与上锁的部分，但是这个接口在ConnectionBehavior内是空实现，这使得调用此接口不再需要判断对象的类型。

  书的流转逻辑位于RoundManager，控制了创建书的转移流动的操作逻辑与方式。

  Roundmanager内的NextRound方法完全控制了一次回合变更内会发生的所有行为。

  GameProcessManager在全局进程中通过判断转化的节点个数或者其它可以反映游戏进程的标志，达成了许多随着游戏进程而开启的游戏内容。

  GlobalVar管理着所有全局变量，它们也是可随进程而修改的。

- **扩展规则：看火者模块**

  DetectiveBehavior是此模块的主要实现，它管理所有看火者的移动、侦察逻辑，使用异步方法实现了动画效果的时序与等待。

- **操作输入与视角移动模块**

  我们将玩家与游戏系统的交互分为两部分：与游戏物体的交互和操控视角的交互，前者完全由鼠标点击实现，后者主要由键盘操作实现并附带少量鼠标操作。

  与游戏物体的交互使用了CursorManager脚本来进行统一的管理。此脚本控制从屏幕发射射线，将碰撞信息传递给它所统筹的方法，以此集中了所有与游戏物体的交互判定逻辑。这部分的操作屏蔽使用ui层进行管理，当需要屏蔽鼠标与3d场景交互时，激活一个透明但接受raycast的2d全屏image。

  控制操作视角的逻辑实现于Main Camera搭载的脚本CameraBehavior，实现了一套完善的视角移动交互逻辑。此外，此脚本还提供了数个转场镜头计算逻辑，这使得某些时候可以产生非常炫酷的镜头效果。

- **地图加载模块**

  我们在初期的mapeditor场景内进行了地图编辑器的实现，我们可以在此场景中便捷地添加节点与建立连接，而后把编辑好的地图进行序列化供主场景使用。

  使用加载形式的地图而非固定的场景的主要考虑是直接编辑场景来获得地图会对地图编辑环节连接的创建与管理带来相当大的麻烦。

  地图文件被序列化为json文件放在Assets/Maps文件夹内，当主场景加载时，会反序列化并初始化地图。一个地图文件包含了节点列表与连接列表，它们还分别附带了一些特有的属性。

- **存档模块**

  每次退出游戏时，会进行存档。存档文件位于Assets/Saves，名为save.json。

  存档文件对五个方面进行了序列化：

  - 全局变量 - GlobalVar整合了所有与游戏进程相关的全局变量，存档对其序列化。

- 节点属性与状态 - 存档读取所有节点的需要序列化的（即会由于游戏进程推进而变化的）属性与状态并序列化。
- 连接属性与状态 - 存档读取所有连接的展示信息状态并序列化。同时读取连接的解锁状态，这一条仅适用于Unlockable的连接。
- 看火者位置信息 - 存档读取所有看火者的位置信息并序列化。
- 任务信息 - 存档读取当前所有任务的激活情况并序列化。

- **剧情模块**

  剧情模块的实现是本项目的重点。虽然这部分有许多现成的解决方案可以借鉴，但为了更符合本项目的需求，我们选择了自己实现这一模块。

  这一模块可以分为三个部分：剧情文件、剧情管理器、剧情ui。

  我们设计了一种十分具有时间上的性价比的剧情编辑方式。

  在剧情设计中，有一些结构会为剧情编辑工具带来相当的复杂度：剧情分支、剧情事件。我们从编译原理中对if分支语句的编译操作获得启发，实现了一种基于flag跳转的剧情分支解析方式。同时，为剧情管理器外接了一个自定义函数管理器，这可以使用剧情文件中解析到的函数名直接调用相应的函数。

  具体来说，我们可以用一个此项目中的剧情文件来做例子。

```
define 1 '旁白'
define 2 '艾米'
define 3 '诺瓦'
define 4 '菲尼克斯'

say 1 '在你找回了一定数量的成员后，你们的组织终于能正常运转起来了，书籍的流通、正常的例行会议，所有这些事情以一种貌似平淡的方式进行着。'

say 1 '但在今天的会议上伦纳德抛出了这个问题——读书会遭到了一个敲诈者的骚扰。'

say 2 '最近有个问题...有人在试图利用我们。他声称自己知道我们的一些活动，暗示如果不给他一部分“好处”，他可能会把消息泄露出去。'

say 3 '（冷笑）又是一个只想着自己利益的家伙。这种人不过是想榨取点金钱，好继续躲在暗处看我们挣扎。'

Isay 4 '他是谁？我们对他有多少了解？'

say 2 '他叫马丁，以前和一些地下交易圈子有来往，算是个走私者。根据我打听到的，他了解一些关于我们书籍流通的事情。他还威胁说如果不“合作”，就会把情报卖给消防局。'

say 3 '简直是个无耻的勒索者。他根本不在乎我们的理念，只想从中获利。菲尼克斯，我们不能向这种人妥协。'

Isay 4 '（沉思片刻）妥协当然不可取，但如果他真的掌握了敏感信息，我们的整个网络就有暴露的风险。'

say 2 '他提出的要求并不高，只要我们提供几本书给他，他就“保证”不会走漏消息。'

say 3 '这只是开始！今天几本书，明天呢？每次他都可以提更高的要求。菲尼克斯，你清楚这种人只会变本加厉。'

Isay 4 '可是，如果我们拒绝他，风险会立即浮现。他真敢告发我们吗？我们还不知道他的底线。'
```

```
say 2 '一些成员觉得应该妥协，避免冲突。他们认为这笔“交易”只是暂时的，能让我们在重新站稳脚跟前平安无事。'

say 3 '妥协？这就意味着，我们读书会的目标被一个贪婪之徒牵制了。我们是为了知识而聚集，不是为了满足他的胃口。'

Isay 4 '那么，我们就面临两种选择：要么给他书，维持短暂的平静；要么拒绝，冒着可能暴露的风险。'

say 3 '菲尼克斯，最终决定在你手里。但我还是要说，我们的底线不能轻易动摇。如果我们一开始就让步，未来的威胁只会更多。'

Isay 4 '我明白。这是我们的考验，必须慎重对待。给我一点时间，我会做出决定。读书会要存活下去，我们需不需要为了一些利益而向某些人妥协，这件事情值得思考。'

selectItem '选择妥协（损失一本书）' flag1
selectItem '我们不应该向这种人妥协（暴露值大量增加）' flag2
select

flag flag1
func lostABook
goto end

flag flag2
func gainHugeAmountOfExposureValue
goto end

flag end

exit
```

就如同将汇编语言转换为机器码那样，我们需要进行两次扫描。第一次扫描将define语句解析为从int到string的映射map，并建立从flag名称到行号的对应表。第二次扫描开始逐行解析剧情文件，根据解析的内容不断向ui侧推送剧情。

进行选择时，我们将一个selectItem命令解析为：向选项栈中压入一个选项，每个选项对应一个flag。select命令就是立刻弹出所有选项栈的选项，并将它们作为一次选择操作推送到剧情ui侧。

剧情管理器与ui侧的具体交互逻辑大量地使用了事件通信：

- 剧情管理器接收到开启一段新剧情的请求，将读取剧情文件，执行上文所说的第一次扫描，而后向ui侧推送去一个“PushStart”信号。
- ui收到此信号将进行ui展示的初始化以及显示动效等部分。当它完成了初始化，可以接受剧情输入时，它触发事件Start，剧情管理器监听这个事件，于是开始推送第一段剧情组件。
- ui侧会根据时间或是用户操作获得当前剧情组件已经展示完毕的信号，然后向管理器侧发去信息。管理器根据上次推送的组件是否为一个选择项，判定是否需要根据传回的tag进行跳转。跳转完毕后，便继续解析行，直到解析到一个需要进行推送的行，推送完毕后退出方法继续等待下一步。
- 这样，ui侧将完全不需要持有具体的剧情分支逻辑与自定义函数事件，它只需要接收剧情推送，然后按部就班地完成展示即可。这体现了基本的MVC模型解耦思想。

- **任务模块**

使用了基本的面向对象思想，每一个任务类继承“任务单元”基类，重写各自的判断任务完成、完成后的动作等方法，由此实现了具有非常高灵活度的任务系统。

- **场景加载模块**

  使用了异步加载场景，场景加载的模式始终为Single。