# 工作总结报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 | 顾家铖 | 项目及岗位名称 | 米哈游-游戏开发助理 |
| 项目起止时间:2025.6.30-2025.8.4 | | | |
| **每周工作总结** | | | |
| （包含参与项目具体内容、项目中遇到的困难、以及收获）  第一周：   1. 布置任务：   熟悉ugui，找一个多端游戏例子，拼一个PC端的技能图标界面以及一个同一个技能在移动端下的图标。  使用git bash将项目上传到github。  使用cursor，了解下如何在unity中自定义inspector。  思考如何在游戏运行时，实现一键将PC端的图标切换成移动端的图标。   1. 完成情况、困难和收获：   完成情况：  熟悉了unity 的 ui 组件，初步掌握text，image，button三大金刚ui组件。  PC端：    移动端  找到多端游戏例子：三角洲行动，暗区突围，第五人格  使用git bash 将项目上传到github，掌握以下git指令：  git config --global user.name \*\*\*  git config --global user.email \*\*\*  git init  git add .  git commit -m "。。。"  git remote add origin git@github.com:n1c0-hi/rain.git  git branch -M master  git push -u origin master    在Unity中自定义Inspector的主要思路如下：  1. 创建自定义编辑器类  - 在项目中创建一个 Editor 文件夹（必须是这个名字）  - 编辑器脚本必须放在 Editor 文件夹中  - 使用 [CustomEditor(typeof(YourComponent))] 特性来指定要自定义的组件  - 继承 UnityEditor.Editor 类  2. 重写OnInspectorGUI方法  - 这是自定义Inspector界面的核心方法  - 可以使用EditorGUILayout或GUILayout类提供的各种控件  - 可以完全自定义布局和显示方式  3. 常用功能实现方式  - 自定义字段显示：使用EditorGUILayout.PropertyField  - 分组显示：使用EditorGUILayout.BeginVertical/EndVertical  - 折叠面板：使用EditorGUILayout.Foldout  - 自定义按钮：使用GUILayout.Button  - 撤销/重做支持：使用Undo.RecordObject  4. 高级功能  - 自定义预览：重写OnPreviewGUI方法  - 场景视图交互：实现OnSceneGUI方法  - 属性验证：使用SerializedProperty的相关方法  - 自定义属性绘制器：使用PropertyDrawer  5. 最佳实践  - 使用SerializedObject和SerializedProperty来处理序列化数据  - 合理使用EditorGUI.BeginChangeCheck和EndChangeCheck检测变化  - 需要时调用serializedObject.Update()和serializedObject.ApplyModifiedProperties()  - 考虑使用EditorPrefs保存编辑器的状态  实现一键将PC端的图标切换成移动端的图标：  我在游戏中设置了一个change button ，玩家按下之后会改变预设的分辨率。可以在游戏中切换pc和移动端模式。  困难：  对于在Unity中自定义Inspector这部分的内容了解基于ai生成，并没有真正的理解所说的内容，且停留在概念阶段，不会实际操作。  在pc和移动端模式切换过程中，由于电脑屏幕比例的原因，切换pc和移动端需要手动翻转屏幕才能将窗口显示完整。  我不太清楚如何改进。  收获：  知道了ugui组件的用法；会用脚本控制ui组件；学习了不同分辨率下如何调整ui界面使其能适配pc和移动端的需求；  学习了git相关指令。  第二周：   1. 布置任务：   将unity项目常用的gitignore放进仓库，忽略掉不必要上传的文件；  学习观察者模式这个设计模式，并且将切换平台后，按钮跟着变化这个逻辑用观察者模式实现，用枚举定义平台；  扩展recttransform的inspector，增加一个＂添加多平台数据＂按钮。   1. 完成情况、困难和收获：   完成情况：  顺利push unity项目到github，github地址：<https://github.com/n1c0-hi/week2.git>；  学习了观察者模式这个设计模式，并且将切换平台后，按钮跟着变化这个逻辑用观察者模式实现，用枚举定义平台；        扩展了recttransform的inspector，增加一个＂添加多平台数据＂按钮。    点击添加的按钮添加具体的ui组件的位置大小，输入具体的数据用于控制不同平台下ui组件的变化情况    困难：  并未遇到特别的困难。  收获：  了解了gitignore，解决了第一周因为unity一些不必要上传的很多文件导致上传github失败的问题；  学习了观察着模式的运作流程：观察者通过注册（RegisterObserver）添加到被观察者的列表中。被观察者状态变化时调用NotifyObservers()遍历列表，触发所有观察者的更新方法；  第三周：   1. 布置任务：   更新数据存储结构，每一个平台存储一份属性覆盖数据  将多平台的数据展示上移到recttransform中，并显示当前平台  允许用户选择哪些属性是需要多平台的  研究如何将上述这些多平台数据以最精简的二进制方式存储   1. 完成情况、困难和收获：   完成了每一个平台存储一份属性覆盖数据，上方有一个统一的大的选择项来确定哪些属性是要属性覆盖的，下方可以具体配置具体哪个平台的哪几个属性是需要覆盖的。      将多平台的数据展示上移到recttransform中，并显示当前平台，不仅显示了当前平台，还有平台切换按钮，方便设计师进行平台切换，更加方便配置ui组件。      可以看到有自动记录设置，例如：设计师可以点击ps5平台，切换到ps5平台后，拖动Scene中的ui组件，相应的数据会自动采集并且记录到平台覆盖设置的对应平台的对应变化的属性的数据中去。  实现上述这些多平台数据以最精简的二进制方式存储。压缩比达到3。  实现思路：  1. 位标志压缩  - 6个布尔值压缩到1字节 ( OverrideFlags )  - 空间节省83%  2. 紧凑数据结构  - CompactPlatformSettings : 65字节/平台  - 传统方式: ~200字节/平台  - 直接内存布局，无对象开销  3. 智能缓存机制  - 写时序列化 : 修改时立即持久化  - 读时反序列化 : 首次访问才加载  - 缓存复用 : 避免重复操作  架构设计：  MultiPlatformRectData  ├── useBinaryStorage (开关)  ├── platformData (传统List存储)  └── binaryContainer (二进制存储)  ├── binaryData (序列化字节数组)  └── cachedData (内存缓存)    第四周：   1. 布置任务：   editor切换平台的时候能显示对应平台的效果  将ui保存成prefab，用文本编辑器打开prefab文件，理解其中的一些布局方式。然后理解一下，把多平台数据存成二进制是什么含义  切换平台的时候显示动画过渡效果，比如位移缩放和透明度  研究一下打包后的prefab是以什么形式存在的，是怎么序列化的？   1. 完成情况、困难和收获：   在recttransform面板中点击切换平台的按钮，在scene中可以看到ui组件根据在属性覆盖中所配置好的数据进行自身变换，**实现了editor切换平台的时候能显示对应平台的效果**。  public class TraditionalFlags  {  public bool enablePosition = false; // 4字节（C#中bool实际占用4字节）  public bool enableSize = false; // 4字节  public bool enableAnchors = false; // 4字节  public bool enablePivot = false; // 4字节  public bool enableRotation = false; // 4字节  public bool enableScale = false; // 4字节  // 总计：24字节  }  To  [Flags]  public enum OverrideFlags : byte  {  None = 0, // 0000 0000  Position = 1 << 0, // 0000 0001  Size = 1 << 1, // 0000 0010  Anchors = 1 << 2, // 0000 0100  Pivot = 1 << 3, // 0000 1000  Rotation = 1 << 4, // 0001 0000  Scale = 1 << 5 // 0010 0000  }  `BinaryPlatformData.cs` 中的 `GetStorageSize` 方法：  public int GetStorageSize()  {  return binaryData?.Length ?? 0;  }  `MultiPlatformRectData.cs` 中的 `GetCompressionRatio` 方法：  public float GetCompressionRatio()  {  if (!useBinaryStorage || platformData == null || platformData.Count == 0)  return 1.0f;  int traditionalSize = CalculateActualTraditionalSize();  int binarySize = binaryContainer.GetStorageSize();    return binarySize > 0 ? (float)traditionalSize / binarySize : 1.0f;  }  压缩效果主要来自：  位标志压缩 : 6个布尔值从24字节压缩到1字节  紧凑数据结构 : CompactPlatformSettings (65字节) vs 传统结构 (88字节)  二进制格式 : 消除YAML格式的冗余文本  数据对齐优化 : 减少内存对齐造成的空间浪费  **prefab是以什么形式存在的，是怎么序列化的：**  **YAML序列化格式**  %YAML 1.1  %TAG !u! tag:unity3d.com,2011:  **GameObject层次结构**  --- !u!1 &4394435116926139837 # GameObject  --- !u!224 &7985788154231934326 # RectTransform  --- !u!222 &6160467552727658526 # CanvasRenderer  --- !u!114 &8485603406260987749 # Image组件  --- !u!114 &8780663252545177489 # Button组件  --- !u!114 &3328966160315984761 # MultiPlatformRectData组件  --- !u!114 &7989406212562674780 # BinaryStorageTest组件  **关于切换平台的时候显示动画过渡效果，比如位移缩放和透明度：**  因为我在实现scene在editor切换平台时能实现显示对应平台效果，我的方法采用的是强制刷新scene的内容。是直接切换的，我的思路是在强制刷新执行前加入动画机的触发bool变量，添加协程等动画播放完毕再进行刷新，但是这实现需要在游戏运行时才能实现，在编辑条件下我觉得不太可行。 | | | |
| **工作收获** | | | |
| 项目中所使用的工具 | | Unity2023.2.20f1c1,trae，google | |
| 请列举工作过程中的其他收获  （包括但不限于知识、技能、见识、沟通能力，职业素养等） | | 可以通过代码对inspecter进行修改，熟悉unity的ui系统，学习跨平台的知识。熟练运营ai提升工作效率。有效的沟通工作进程。 | |
| 参与后是否会将此类工作为未来求职的目标，为什么? | | 考虑作为未来求职的目标之一，我觉得技术类的工作还蛮适合我的。 | |
| 身边的导师是否有让你觉得佩服的，他们有哪些特点？ | | 导师专业知识层次很深入 | |
| 自己还有哪些方面需要提升？ | | 对于跨平台数据处理还需提升，动画制作部分也需要提升 | |
| 请客观陈述一下自己的项目感受 | | 项目内容对unity引擎的开发更加的深入，项目体量适当 | |
| 你希望下次项目的时间和时长是何时、多久？是实地还是远程？ | | 大三暑假，暂定 | |
| 备注：如有，请在反馈此报告时附上项目过程中参与完成的报告（如PPT、策划方案等） | | | |