团结引擎

微信小游戏（以下优化是已经被团结引擎实现的）

* 引擎轻量化
  + 托管代码精简
    - 缩减构建后的托管程序集大小，对iL2cpp优化效果明显
    - 修改player的Optimaztion的MnagedStrippingLevel到Extreme极端模式
    - 勾选LogStrppedMethodForDebug选项，通过添加打印的方式排查哪些被调用但是被移除的方法
  + 引擎代码剔除
    - 剔除引擎内部小游戏未使用到的功能点
    - 支持LightProbe环境光、多种纹理格式的运行时解压缩（DXT，ETC，ASTC）、FreeType中SFNT格式的PNG支持。这几个都可以剔除
    - 剔除设置支持3种模式：kepp保留、force strip 强制移除、auto detect 自动检查（目前自动检查仅支持环境光）
  + Build-in Package剔除
    - 内建功能模块默认是启用的
    - 在strip advice窗口中，可以disable selected ，关闭内建功能模块
  + 资产轻量化
    - 使用微信宿主提供的中文字体
    - 优化tuanjie\_default\_resources文件，从593kb到341kb
      * 调整微信小游戏平台默认纹理的压缩格式，rgba->astc
      * 移除微信游戏不支持的compute shader
  + iL2cpp元数据精简
    - iL2cpp运行时（就是在端侧运行cpp代码的虚拟机）依赖元数据获取c#类型、方法等信息。与游戏中的c#代码数量相关。保存了程序集、类型和方法等信息。默认元数据结构可以支持超过21亿个类型或方法。小游戏通常在万这个级别。可以精简元数据来减小iL2cpp元数据文件大小
    - 大概可以减少15%左右
    - 通过Publishing Settings中的Use Slim Format For global-metadata.data勾选控制，默认为开启
  + emscripten内联选项优化
    - 默认情况下，会采用较高级别的内联优化处理，一定程度上提升运行效率。但是会导致wasm中可能出现含有数千行指令但只有数十行被执行的函数。所以限制内联函数的行数大小也可以缩减包。但是值如果设置过小又会损耗性能。损耗就是符号跳转的性能
    - 在build settings中platform-weixin minigame-one caller inline maxsize，该值在20-1000，默认为200.可以根据实际情况修改比较精确的值
* 运行时内存优化
  + IL2CPP运行时内存优化
    - IL2CPP运行时，作为脚本代码运行的虚拟机，本身也会消耗内存。上述的IL2cpp元数据精简也会相应较低IL2CPP运行时使用的内存消耗。将近50%
  + 内存分配器优化
    - 在Release版本中，之前Overhead是默认开启的，开启后会在内存块中头部或尾部记录每次分配的大小，有助于检测内存使用是否存在问题，比如内存泄漏。虽然不影响功能，但会消耗一些额外的资源。
    - 现在Overhead默认在release模式下关闭。所以关闭后使用Profile.GetTotalAllocatedMemoryLong无法获取使用的内存大小，需要使用Slim接口重启应用或者在BuildingSettings-StripAdvice窗口-修改AllocateOverhead为Keep开启状态，
    - 引擎默认内存Alignment是16字节，微信小游戏不需要这么大，在Release版本下会减小为4字节。降级相关的Overhead.。这会减少每次分配的大小有较少的损失。
    - 其它平台如果使用4字节对齐，可能需要处理器多次内存访问，增加开销导致性能降低；更小的内存对齐可能会导致数据跨多个缓存行降低缓存命中率导致内存访问速度变慢；肯呢个会导致内存读取错误与兼容性变差等。
  + Shader内存优化
    - 从AB加载shader流程如下：先将压缩数据读进内存，然后解压所有shader blob数据，后续解析shader时会从解压后的blob中加载所有shader变体。完成后将原始blob数据释放。‘
    - 因为在实际游戏中，并不是所有变体都在使用。团结引擎针对shader变体做了如下处理
      * 将shader变体的blob数据修改为按需加载，解压延迟到了使用shader变体的时候
      * shader变体解析完成后，解压数据会被释放，从而降低内存占用。
    - 对比
      * 原：解压所有blob数据-加载所有shader变体
      * 现：仅解压blob数据表头-创建不包含blob数据的shader变体-按需解压指定blob并加载-卸载解压后的blob数据-
  + Remapper内存优化（重映射）
    - 引擎会给内存中的每个对象都分配一个唯一的heap ID，成为instance ID。
      * 对象的ID不会为0
      * 从序列化文件中加载的对象是正数；从代码中构造的对象是负数
      * 新对象产生时，每次递增2（或-2）；【优化为每次递增为1】
        + 为什么只增不减？

确保唯一性，简化对象查找和管理

内存管理，如果被回收或减少可能更频繁的进行内存管理，增加了系统复杂性和性能开销

历史数据的保持，保留历史数据的价值，在调试时可以追踪对象的创建和销毁过程

简化算法，查找和插入变得简单，可以通过线性查找二分查找来确认某个ID是否存在。简化了和其他数据的交互，比如映射、记录等

避免竞争条件，比如多线程中如果ID可以被回收会引入竞争条件。多个现成同时请求相同的ID时，可能会出现冲突和数据不一致的问题。

* + - * ID只增不减，不会被复用；【优化为增加版本概念，复用已有的ID】
        + 新增一个双向映射表，此表内容可以随着该对象释放而移除
        + 因为原来的只增不减的
      * 同一次启动中，对象ID始终不变，即使对象重新加载过
      * 【新增：内存紧凑数据结构代替原有哈希表】
        + 指针压缩：将64位指针压缩为32位或更小，减少存储时指针占用
        + 紧凑哈希表，减少内存开销
        + 其它技术紧凑内存
  + 绘制优化
    - GPU Skinning
      * 小游戏CPU的性能只有原生App的1/3，CPU Skinning坑存在较大开销
      * WebGL1&WebGl2不支持Compute Shader。团结针对小游戏在微信上提供了GPU计算方案Transform Feedback与Vertex Shader。其中transformer feedback方案过期了，没有Vertex Shader好。
      * 在ProjectSettings - Player - Other Settings - Mesh Skinning 选择Mesh Skinnning方式
        + CPU Skinning ：顶点蒙皮运算。在CPU上进行。每帧动画开始时，先按FK、IK计算出所有谷歌的评议和旋转变化矩阵。然后读取模型在内存中的顶点数据，依据每个顶点的位置和绑定的谷歌权重计算出变化后的顶点位置。变换后的顶点数据需要每帧重新上传到GPU，同时供shader的多个pass（阴影、光照）使用。是webGL的默认动画方案
        + Transform Feedback：使用vertexsa
        + Vertex Shader Skinning
    - 图形渲染优化
      * shader优化，fps翻倍，位于project settings - player - other settings 选择webGL2.0，勾选enable shader optimizations 会有子选项
      * 允许禁用默认的FBO的DepthStencilComponent，减少内存占用。禁用离屏渲染深度缓冲区的组件。
    - Shader变体异步warmup
      * 新增异步shader预热接口，因为支持KHR\_parallel\_shader\_compile的会立即返回状态，所以可以直接连续调用不会卡住主线程。使用GL\_COMPLETION\_STATUS\_KHR检查，知道所有都完成，再触发completed事件。complete后再进入对应的游戏场景。
    - 提高DynamicVBO对象池的复用效率
      * 修改缓存复用策略，默认开启
    - TextMesh Pro动态字体性能优化
      * 避免CPU进行纹理格式转换计算
      * GPU现存占用减少1/4
  + 启动加载优化
    - 首包wasm & data文件精简，参考上面的精简步骤，主要是减少体积
    - 引擎初始化加速
      * il2cpp初始化加速
      * shader变体延迟加载
      * 延迟创建monoscript缓存
      * GLES Gfx Device初始化优化
    - Object.InstantiateAsync，分帧构造对象。因为微信小游戏是单线程的。

Unity 小游戏

游戏试玩；或模型查看

webView 3D模型加载