Unity3D开发tile地图

## 思路

* 资源和数据分离
* Tile自主加载
* 资源缓存

**资源和数据分离**

数据中记录的是整个地图中每个tile应该显示的资源（地图纹理）的id，资源和数据并非同步更新。不同位置用到相同id的资源可直接使用，以便减少资源的请求次数。

**Tile自主加载**

Tile创建创建指令发出后就开始自我维护状态。完成创建和加载工作。

**资源缓存**

资源缓存分为两部分——物理缓存和内存缓存。内存缓存可以减少请求次数，和图片转化过程中的消耗，物理缓存用来提高加载速度，使图片的载入看起来更有效。

主要过程：数据请求——资源载入——缓存——地图显示

## Demo

地图尺寸100\*100（m），tile尺寸1\*1（m）。

用Txt文件作为服务器的数据，以网址http://image6.tuku.cn/pic/sucai/concept\_person\_199\_220/s{0}.jpg

作为资源，其中｛0｝处的有效范围为【001—160】。

图片物理缓存数量30。

运行程序后，程序的数据文件夹MapClient\_Data会多出一个图片缓存文件夹MapClient\_Data/images

和一个数据文本文件MapClient\_Data/ServerData.txt

程序根据相机计算显示范围，所以程序本身可以适应任何分辨率的选择。

拖动画面就可以实现地图的移动。

## Demo实现细节

**服务器模拟ServerData.cs**

生成一万个0-160之间的随机数，作为题图和资源的映射。

程序运行开始会判断是否有保存的数据，有保存数据直接载入，没有则通过备份生成原始数据。

脚本提供一个随机改变的功能，程序启动后每隔固定时间调用一次用来模拟数据变化。

public void RepetChange(int repetTime)

{

if (IsInvoking("SomeChange"))

CancelInvoke("SomeChange");

InvokeRepeating("SomeChange", 0, repetTime);

}

void SomeChange()

{

for (int i = 0; i <= Random.Range(3, 30); i++)

map[Random.Range(0, 10000)] = Random.Range(1, 161);

}

脚本的结束部分负责保存变化的数据

void OnApplicationQuit() { Save(); }

void Save()

{

string content = "";

for (int i = 0; i < 10000; i++)

{

content += map[i].ToString();

if (i != 9999)

content += ",";

}

Help.WriteTxt(Application.dataPath + "/ServerData.txt", content);

}

为了减少数据库的请求次数，采用了多个节点同时请求。在条件允许的情况下甚至可以载入全部数据进行一次性维护。数据请求部分在这个Demo中只是做简单的模拟。并没有添加缓存功能。通过Get方法来获得需要的数据信息。

**图片载入DownLoadImage.cs**

有两个载入途径，一个来自网络，一个来自本地的缓存。

不管从哪个途径载入图片，最后都保存在内存中为缓存。提供一个具有回调功能的方法来完成图片加载。脚本下载功能包括的宽度限制和统一同时发生多条相同请求的功能。并对下载后的图片进行Texture2D缓存。因为直接下载图片unity会进行格式转换，会占用很多内存，直接保存转换前的www类实际上并不会解决内存过大的问题。当请求的图片id在内存中时，则可以直接返回这张图片。省去的重复下载的过程。统一相同请求就是在统一时间内，对同一个id资源进行多次请求，这时之下在一次，然后对每个请求进行回调，以此来优化下载模块。

下载模块的请求顺序：

请求某个id的资源图片

是否在内存中

回调图片资源

是否在本地

下载

缓存

Y

N

Y

N

**图片显示Tile.cs**

提供一个创建借口，并对自身管理。通过借口创建tile后，会在相应位置实例化预制好的cube，并对资源进行请求。回调后记录载入时间。初始化使用时间，在管理视野的函数中来对使用时间进行更新，作为本地缓存的权重。创建自动区分是否已经存在，或者数据是否过期。在调用的时候进行处理（说明在视野内），否则不处理。程序关闭时可以对各个tile的数据进行序列化保存，作为数据缓存。本demo的已经在ServerData中使用数据保存，此处为数据结构的设计，所以没有实现。

**相机移动MoveCamera**

相机的正交尺寸为实际距离的2倍，可以计算屏幕像素到实际距离的影射。通过两帧的鼠标移动距离计算出时机移动距离，并能记录移动方向，用来推测用户的下一步行为实现预加载。

根据地图尺寸设定相机移动范围，使他不能移出地图外。

**管理模块Manager.cs**

当鼠标抬起的时候更新视野内数据，因为在移动过程中，移动速度可能很快。也不宜每阵进行请求。而且在移动范围比较大的过程中也可能经过很多tile，导致最后落点并不是正在加载的数据，反而阻碍了真正数据的加载。鼠标静止状态的时候更新视野内的tile使用时间权重。对唯一资源id进行累加，保证使用次数多时间久权重就大。通过相机移动方向推测下一个移动的可能在空闲状态的时候进行预加载，这里可以进行扩展，demo中只加载了一次，可以进行多次预加载，其次就是在行为的判断上，现在只使用了上一帧和当前帧的比较，这里可会扩大采样的时区，这就能更准确的判断用户的操作行为，这里直接用移动差值做简化。根据移动方向，可以做扇形扩展。或者根据移动速度判断加载的距离。范围可以适当放大等等。程序结束时可以对内存中已经缓存的图片进行赛选比较。选出最常用的资源进行保存。

**物理缓存机制：**

程序运行时检测已经有的物理缓存id，调用这些id的资源时可直接通过本地加载缓存的内存中。网络下载的资源也同样下载到资源中。最后通过使用时间做结算。这里也可以扩展，就是关于tile的序列化，可以保存上一次资源的使用情况，这样加到最后的权重结算中可以得到一个使用软件以来的整体比较。

**已知问题：**

下载采用队列等待，对于地图需求应该采用后劲先出的栈做等待结构更合适。