[Version number, author, save date, and task days total are fields and can be updated with F9]

Resource System Design Document, v3

Greg Sabatini

October 9, 1999

Introduction

The resource system acts as a database for the game, and handles all file loading and saving in the game. It keeps records of all assets which have been loaded and all objects that exists in the game.

Requirements

This system should be able to handle all required file system access, including loading and saving. It also should be able to parse scene information to load assets necessary for the scene, create objects, and spawn those objects. Since all assets must be cross-platform, virtually all of this system will be shared code. Specific file handling routines may be platform specific. If a specialized memory manager is required for the Mac, it will be included in this system.

This system is used by the Gameloop system for scene loading, the Sound system for streaming sounds, and the Script System for loading scripts. It uses the individual asset classes: CAnim, CBitmap, CSprite, CSound, etc. to parse the files it loads into usable formats.

Structures/Classes

Class CSceneDB;

The CSceneDB class keeps records of all the objects in the game. It uses an array of pointers to linked lists for keeping track of the objects. Each object may be in any of the multiple linked lists in this array. Each linked list contains a differently ordered list based on a object property.
The CSceneDB class also is responsible for loading in the scene files, creating objects for them, and calling the CAssetDB class to load the associated assets. This class then adds the objects to its object linked lists.

Class CAssetDB;

The CAssetDB class keeps records of which assets are currently loaded and a pointer to the loaded version of the asset. Using the following structure:

```
struct asset_record_struct {
        char[16] szLabel;
        CAsset *pAsset;
```

};

When called to load an asset, this class will check to see if it has loaded the asset previously. If it has, it will return the pointer to the asset, otherwise it will create a new CAsset subobject based on the file extension and call the object load for this object.

Class CFile;

This class will contain all the file handling routines, including loading, saving, and streaming.

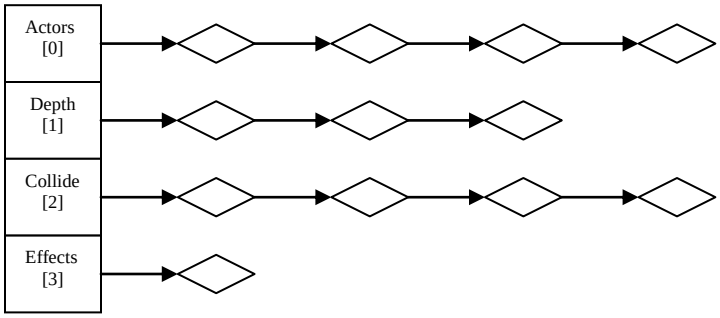
Functions/Methods

```
class CSceneDB {
protected:

public:
        CSceneDB();  // Scene Constructor
        ~CSceneDB();  // Scene Destructor (deletes CAssetDB)

        LoadScene(const char *szFilename);

        AddObject(pObject); // Adds object to linked lists
        DeleteObject(pObject); // Deletes object from linked lists

        SpawnObject(const char *szLabel); // Creates a copy of object

        // Sets active list at head of passed property
        SetListType(enum LinkListProperty);
        NextObject();  // Gets curr object and advances list
};

class CAssetDB {
protected:

public:
        CAssetDB()
        ~CAssetDB()

        LoadAsset(const char* szFile, ** ppObject);

};
```

```
class CFile {
protected:

public:
        CFile();
        ~CFile();

        GetSize(const char* szFilename);

        Load(const char* szFilename, void *pHere);
        Save(const char* szFilename, void *pHere);

        OpenStream(const char* szFilename, int Read_Write_Type);
        ReadStream(Handle, void* pHere, int bytes);
        WriteStream(Handle, void* pHere, int bytes)
        CloseStream(Handle);
};
```

Diagrams

CSceneDB Linked List Diagram



Schedule Task List

System Tasks	Duration	Dependent
Design Win32 CFile class	0.5 Days	Design Document
Design Mac CFile class	0.5 Days	Win32 CFile class designed
Code Win32 CFile class	2 Days	Win32 CFile class designed
Test & Revise Win32 CFile class	1 Day	Win32 CFile class finished
Code Mac CFile class	2 Days	Mac CFile class designed
Test & Revise Mac CFile class	1 Day	Mac CFile class finished
Design CAssetDB class	1 Day	Design Doc, CAsset and subclasses designed
Code CAssetDB class	2 Days	CAssetDB class designed, CAsset subclasses finished
Test & Revise CAssetDB class	1 Day	CAssetDB class finished
Design CSceneDB class	2 Days	Design Doc, CAssetDB class designed
Code CSceneDB class	3 Days	CSceneDB class designed, CAssetDB class finished
Test & Revise CSceneDB class	2 Days	CSceneDB class finished
Integrate Resource system	1 Day	CAssetDB, CSceneDB, CFile tested and revised
Rework #1 CFile class	1 Day	As Needed
Test & Revise CFile Rework #1	1 Day	CFile class Reworked #1
Rework #2 CFile class	1 Day	As Needed
Test & Revise CFile Rework #2	1 Day	CFile class Reworked #2
Rework #1 CAssetDB class	2 Days	As Needed
Test & Revise CAssetDB Rework #1	1 Day	CAssetDB class Reworked #1
Rework #2 CAssetDB class	1 Day	As Needed
Test & Revise CAssetDB Rework #2	1 Day	CAssetDB class Reworked #2
Rework #1 CSceneDB class	2 Days	As Needed
Test & Revise CSceneDB Rework #1	1 Day	CSceneDB class Reworked #1
Rework #2 CSceneDB class	1 Day	As Needed
Test & Revise CSceneDB Rework #2	1 Day	CSceneDB class Reworked #2
Total	33 Days	

Memory

Although this system must use considerable memory while loading files, because the files are first loaded in whole and then parsed into the correct format, the

allocation of this memory is done by other systems. While running the actual game, the memory requirements are small, due to the linked list and array of pointer to linked list data structure use.

Risk Assessment

The Resource database systems have a low level of risk. The risk lies primarily in the SAM reading/parsing being slow, in which case we can create a binary version of the format.

The CFile system also experiences a low level of risk. If memory dictates, we could build a compression system into this class. This is not accounted for in the task list.

Because this system will include the memory manager on the Mac, if required, this system has a high level of risk built in. This is where we could use the Mac consultant if necessary.

QA & Test

For the CAssetDB and CSceneDB classes the correct number of objects and assets (w/o duplications) being loaded in based on the SAM output will verify that this module is working correctly. For the CFile class, the file or portion of the file being the same in memory as it was on the disk will verify that this system works. If no file is working correctly, there is most likely a problem with the CFile class.