Object AI Script System Design Document, v2

Randy Angle

October 19, 1999

Introduction

Every object, character or prop, in the game has a scripted behavior. The script tells the object how to react to timers, player events, and other object interactions. All the behaviors are the result of one of these stimuli. Starting a scene will cause the individual objects to be spawned into the scene. The scene and these objects have certain assets that are organized and laid out with the SAM (Scene Asset Manager) tool. The scripts for the objects are built using a programmer editor and compiled into NOG files using NOGGIN (AI script compiler) tool. This document describes the object hierarchy, the script syntax, and the interpreter model.

The object hierarchy builds from a fundamental object called the CObj, it contains a sprite that contains a bitmap, and a location. A CObj can also contain an animated sprite that has multiple frames of bitmaps that update at regular time intervals. The CObj class also contains other object properties like script name, hotspot data, object name label, and pointers to other associated object assets (sounds, etc.).

When the CObj class is first initialized for each object the code in the SPAWN section of the script is run. In the main game loop each CObj has its' update method called and the UPDATE script section is processed. Any inter-object communication like CLICK, BUMP, MESSAGE has a handler section in the object as well. Lastly there is a way to remove the object altogether and the KILL section of the script is processed.

Inter-object messaging is the primary system for activating or externally changing the state of an object. A CLICK will usually activate any animation and may cause more complex behaviors to begin. The BUMP will send a pointer of the BUMPING object to the BUMPED object. A SEND will send a pointer to the sending object and an EAIMESSAGE type from an enumerated list.

Moving an object on screen involves changing the object's position over several frames during the UPDATE cycle. Access to the current world coordinates is part of an object's properties. If an object has a pointer to another object it can find information about that object's properties too. The script system also has pointers to the properties of the scene and game global data. Completing a learning lesson will cause the "cursor" object to set the game global data, which indicates that the player has learned this subject. When the player selects Clicky or Bricky their object scripts can evaluate the status of the game global data to determine which lesson they will ask to be taught.

The support routines for character behavior can be encapsulated in specific keywords of the script language. Keywords like MOVE, MOVETOWARD, SPAWNNEW, and others will be written in C++ for speed.

Requirements

The requirements for the AI scripting system are to control the behavior of objects in the products. Each object must be able to quickly process their state and move the position of the object if necessary. The objects must also be able to react to external stimuli via the various callback handlers. The quickest way to allow scripting and be fast is to build a small virtual machine interpreter, similar to P-Code or Forth, and write fast C++ code to support the individual keywords. Because the scripts must have access to the variables in a CObj class and game global data the interpreter must abstract the access to the C++ variables.

There can be as many as 30 objects each processing their SPAWN, UPDATE and other sections each frame. As much as possible it will be necessary for long processes to happen over successive frames. There can be no unbounded looping structures that block further processing or the game will skip frame updates. Looping keywords do exist to allow the spawning of multiple random objects and coordinating multiple objects.

If each frame is $1/30^{th}$ of a second on a target machine then processing all objects should take no longer than one-third of that time or $1/90^{th}$ of a second. That would leave the rest of the time for graphics and audio processing.

Because of the direct correlation between AI scripting and approval of game design and features some of the design work involved in the AI script language is left open at this time. This document will receive more attention during the production phase when requirements are better known.

Sample Script

```
SPAWN:
        SET(ME.XPOS,100)
        SET(ME.YPOS,100)
        PLAY("JUAN_WAVE.SPR")
        SAY("JUAN_WELCOME_MATH.WAV")
        SET(ME.AISTATE,AISTATE_FREEPLAY)
        SET(ME.AIACTION,AIACTION_INTRO)
        SET(ME.TIMER,100)
        VISIBLE(TRUE)
        RETURN

UPDATE:
        IF(ME.AISTATE,AISTATE_FREEPLAY)
                IF(ME.AIACTION,AIACTION_ATTRACT)
                        PARTICLE(ME.POSITION,ME.BOX,PARTICLE_STARS)
                ENDIF
                IF(ME.ANIMLOOPING)
                        DECR(ME.TIMER)
                        IF(ME.TIMER,0)
                                PLAY("JUAN_WAVE.SPR")
                                SAY("JUAN_HI.WAV")
                        ENDIF
                ELSEIF
                        IF(ME.ANIMDONE)
                                PLAY("JUAN_STAND.SPR")
                                SET(ME.TIMER,100)
                        ENDIF
                ENDIF
                .
                .
                .
        ENDIF

        IF(ME.AISTATE,AISTATE_LESSON)
                IF(ME.AIACTION,AIACTION_ATTRACT)
                        IF(ME.CLICKABLE)
                                PARTICLE(ME.POSITION,ME.BOX,PARTICLE_MOONS)
                        ENDIF
                ENDIF
                IF(ME.ANIMDONE)
                        FIND(ME.INTEREST,"BALL")
                        MOVETOWARD(ME.INTEREST)
                ENDIF
                .
                .
                .
        ENDIF
        .
        .
        .
        RETURN

MESSAGE:
        IF(ME.MESSAGE,AIMESSAGE_MOUSEOVER)
                SET(ME.AIACTION, AIACTION_ATTRACT)
        ENDIF
        IF(ME.MESSAGE,AIMESSAGE_CLICKED)
                IF(ME.AISTATE,AISTATE_FREEPLAY)
                        SET(ME.AISTATE,AISTATE_LESSON)
                        SET(ME.AIACTION,AIACTION_CLICKED)
                ENDIF
        ENDIF
        IF(ME.MESSAGE,AIMESSAGE_BUMPED)
                IF(ME.MESSAGEWHO.NAME,"BALL")
                        SEND("BALL",AIMESSAGE_CAUGHT)
                ENDIF
        ENDIF
        RETURN

KILL:
        KILL(ME)
```

Structures/Classes

```
typedef enum
{
        AISTATE_DEAD = 0,
        AISTATE_SPAWNING,
        AISTATE_FREEPLAY,
        AISTATE_LESSON,
        AISTATE_MAX
} EAISTATE;

typedef enum
{
        AIACTION_NONE = 0,
        AIACTION_MOVING,
        AIACTION_CLICKED,
        AIACTION_ATTRACT,
        AIACTION_WAITING,
        AIACTION_TALKING,
        AIACTION_INTRO,
        AIACTION_MAX
} EAIACTION;

typedef enum
{
        AIMESSAGE_NONE = 0,
        AIMESSAGE_BUMP,
        AIMESSAGE_CLICK,
        AIMESSAGE_MOUSEOVER,
        AIMESSAGE_CAUGHT,
        AIMESSAGE_APPEAR,
        AIMESSAGE_HIDE,
        AIMESSAGE_ACTIVATE,
        AIMESSAGE_MAX
} EAIMESSAGE;

typedef enum
{
        AITOKEN_NULL = 0,
        AITOKEN_RETURN,
        AITOKEN_IFBOOL,
        AITOKEN_IFSTRING,
        AITOKEN_IFNUMBER,
        AITOKEN_KILL,
        AITOKEN_VISIBLE,
        AITOKEN_SET,
        AITOKEN_SAY,
        AITOKEN_BEEP,
        AITOKEN_PLAY,
        AITOKEN_SEND,
        AITOKEN_PARTICLE,
        AITOKEN_DEBUGOUT,
        .
        .
        .
        AITOKEN_MAX
} EAITOKEN;

typedef enum
{
        AITYPE_NONE = 0,
        AITYPE_CURSOR,
        AITYPE_CHARACTER,
        AITYPE_PROP,
        AITYPE_PARTICLE,
        .
        .
        .
        AITYPE_MAX
} EAITYPE;
```

```
class CNoggin
{
private:
        BYTE*   pScript;
        BYTE*   pSpawn;
        BYTE*   pUpdate;
        BYTE*   pMessage;
        BYTE*   pKill;
        BOOL    Parse(BYTE* Section);
public:
        CNoggin();
        ~CNoggin();
        BOOL    NogLoad(char* ScriptName);
        BOOL    NogSpawn(void);
        BOOL    NogUpdate(float DeltaTime);
        void    NogMessage(CObj* Who);
        BOOL    NogKill(void);
}
```

Schedule Task List

System Tasks	Duration	Dependent
Design AI Script System	5 Days	Design Document
Code Noggin Tool 1ST Pass	5 Days	AI System designed
Code Noggin Tool 2nd Pass	5 Days	AI System designed
Code AI System Class	5 Days	AI Class designed
Code AI Parser	3 Days	AI System designed
Code AI Keywords	5 Days	AI Class designed
Integrate AI System	3 Days	AI Class coded
Test & Revise AI System	1 Day	RK System integrated
Rework #1 AI System	2 Days	As Needed
Test & Revise AI Rework #1	1 Day	AI System Reworked #1
Rework #2 AI System	2 Day	As Needed
Test & Revise AI Rework #2	1 Day	AI System Reworked #2
Rework #3 AI System	2 Day	As Needed
Test & Revise AI Rework #3	1 Day	AI System Reworked #3
Total	41 Days	

System Tasks	Duration	Dependent
Code Pre-K Math Scripts	10 Days	AI System integrated
Code Pre-K Language Scripts	6 Days	AI System integrated
Code Pre-K Art Scripts	6 Days	AI System integrated
Code Pre-K Music Scripts	6 Days	AI System integrated
Code K Math Scripts	6 Days	AI System integrated
Code K Language Scripts	6 Days	AI System integrated
Code K Art Scripts	6 Days	AI System integrated
Code K Music Scripts	6 Days	AI System integrated
Total	52 Days	

Memory

The AI Script System uses very little memory if 20 objects have 10K bytes of script and class space each that would amount to 200K bytes of AI space.

Risk Assessment

The real risk with the AI Script System is that we might need additional revisions during development. The design time allocated during development should help to lessen that risk and the method used for this script system is based on several generations of previous script system design.

In order to guard against faulty scripts, magic numbers and checksums can be used in the files.

QA & Test

The AI Script system will probably have the greatest number of errors. The language is new and therefore the programmers will make occasional mistakes. The compiler tool, NOGGIN, will catch some of these, while the debug output will help to catch others. The CNoggin class is relatively simple code and should be easy to test in early versions. The testing of the NOGGIN tool will be done while building and integrating the CNoggin Class.