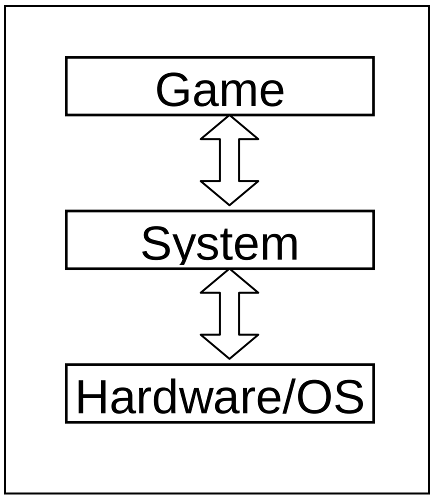
LEGO Code Design Document, v11

Randy Angle

October 28, 1999

LEGO Code Model

LEGO will be implemented as an object oriented approach to a “Director” style 2D educational product. The overall structure of the *LEGO* game will center on a typical main-loop style with each system getting called with an appropriate time slice to execute its needed functionality. There should be no “blocking” long-term processes during game play. The movie system is handled differently and will block waiting for the end of the movie or an interruption. The main-loop should process at least every 15th of a second (30th on the target) and all systems will be called each loop. Critical, interrupt, tasks will have callbacks to indicate the completion of tasks. Messaging between objects will happen by calling inter-object methods to pass information. HARDWARE and low-level functionality will be abstracted as a SYSTEM layer. The logic and process will be handled in the GAME layer.

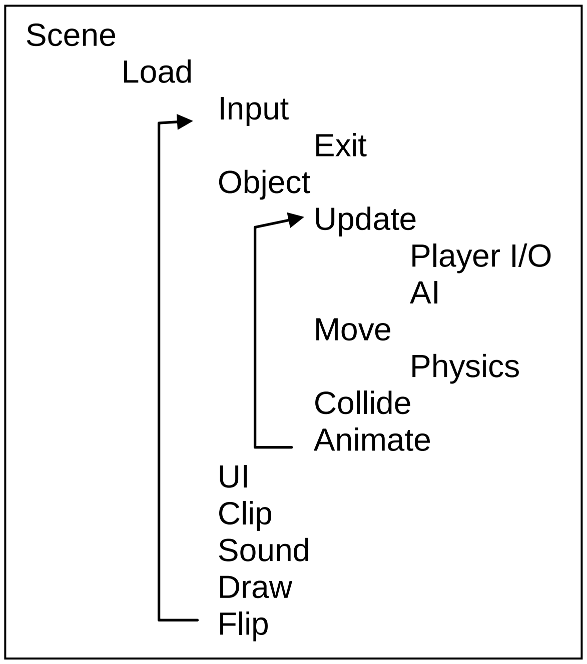


Those things that actually touch the hardware in some way or act as utilities (Graphics, Sound, File handling, Input, etc) will reside in the SYSTEM layer. The abstract or product specific code (animation, AI, hot-spot detection, UI, script, etc.) will reside in the GAME layer. The idea here is to separate the hardware functions so that it is easier to port the product from the PC to the MAC.

Game Loop

LEGO will has two central loops and a core path for passing data between the systems during object update. During game play all objects will get processed every frame for AI, Movement, Physics, Collision and if necessary Player I/O.

```
Main
    Setup Windows
    Init subsystems
    Locate CD & SaveDir
    Load Game Settings
    State
        Movie
        Scene
    Ba-Bye
```



```
Scene
    Load
        Input
            Exit
        Object
            Update
                Player I/O
                AI
            Move
                Physics
            Collide
            Animate
        UI
        Clip
        Sound
        Draw
        Flip
```

Timing Model

LEGO will use delta-time per frame for all calculations. All physics, collision, animation, AI and any other time dependant calculation will use the delta frame time for any specific frame. After each frame update, the delta-time is calculated and kept in a global variable, gDeltaTime. All systems can check the gDeltaTime for any time-based calculation. Also for use is a global variable, gGameTime, that shows the elapsed time since the game has begun.

The motion will be calculated based on the gDeltaTime value. The animation routines handle the delta time to interpolate between key-frames. For looping positions any amount that the loop length has been exceeded is used as an offset from the beginning of the loop for keeping smooth animation. The AI will have the scripted heuristics adjusted based not only on animation time for actions but any random waiting will be done based on the delta-times accumulated as each AI update is called. If an AI needs to be lowered in priority each AI can decide to lower its' update rate by returning early until its' own determined delta-time has been reached. That way a button can go down to low priority until the player presses it.

Because of *LEGO* timer model there is very little risk in short delays for single frames. If windows tasks for a moment *LEGO* will gracefully continue with no affect to the continuing game play. It is the goal of the team to achieve at least 30 frames/second on the target system with 15 frames/second on the minimum system.

LEGO Data Model

As much as is reasonable the *LEGO* software will be considered an engine. The actual game will be data driven from data sets created in various tools. *LEGO* 2 should be a simple addition of a new data file included with the *LEGO* Engine Player. One executable, an installer, a document file, a default configuration file and a new set of assets will comprise all the deliverables on a master CD-ROM. Extending or developing for *LEGO* should be easy.

Installation

The "save game" directory will contain the executable (so that it can be patched), all save games and configuration files. A typical hard-drive install will consume less than 5MB of disk space, 1.5MB for executable, 50KB for configuration files, and the rest for save game data.

On the PC *LEGO* will use *InstallShield* and Auto-play to install itself when the CD is inserted. The Auto-play executable will set a registry flag indicating it has finished installing *LEGO* and the next time it is run it will simply ask if the Player wishes to play the game. If the game is uninstalled the registry flag is cleared and the Auto-play will install it again.

On the MAC *LEGO* will use *InstallVice* a product for the MAC similar to the PC *InstallShield.*

File Access

When the game is started it determines where the "save game" directory and CD drive with the game is. During scene loading all CD access will happen at the maximum rate for the drive. During game play no data will be read except streamed songs or dialog.

File data will be loaded in pre-compiled chunks. Instead of using an IFF style file format and parsing the data, *LEGO* will use a format with a structure header that has byte offsets into the individual sections of data. Large data buffers will be filled with the contents of files and a pointer to the base structure will allow for pointer arithmetic into the actual data chunks. The reason for doing this is for speed, memory management, and avoiding the blue-text-screen-of-death. For more detail see the FILE SYSTEM.DOC.

System Memory

Size for types of objects:

Object	Size (Kbytes)
Character Animation – 128 frames	512
Sound library – 95 seconds	1024
Prop animation – Appear, Activate, Glow…	256
Executable – PC	1537.5
Scene – One learning activity	128
Script – One complex object	128
Graphics Buffers – 64x64x16Bit	8
Sound Buffers – 47 seconds	512

Approximately 256KB are used for instancing class data for game objects. Windows 95 has approximately a 16MB footprint; Windows 98 can take up to 20MB (all approximate Windows is not consistent on this matter).

The allocation for work RAM is:

Item	Size (KB)
Executable	1537.5
Data	256
Scene & Scripts	2048
Sound & Stream Buffer	1537.5
Character Animation	4096
Prop Animation	256
Graphics Buffers	2048
Total	**11779**
Available	**509**

Auxiliary Memory

DirectSound makes it possible to load the sound samples into sound memory for quick processing of audio effects without incurring additional space in System Memory. If available, *LEGO* will take advantage of this memory for its' audio samples. There is no equivalent MAC sound card advantage.

LEGO is best with at least 2MB of VRAM. This allows us to page flip and retain a background in video memory.

The allocation for 2MB of VRAM is:

Item	Item Size in Bytes
Display Buffer 640x480x16	614400
Back Buffer 640x480x16	614400
Background 720x480x16	691200
UI Elements & Font	177152
Total	**2097152**

CD-ROM Storage

Most components of the game will be used in more than one scene. There are 28 activities, one scrolling world, one sign-in scene, and one type-in name scene. The bulk of the CD will be dialog and songs. The next biggest set of assets will be animations and then backgrounds. There are eight static backgrounds at approximately 380K bytes each and one large scrolling background at approximately 2048K bytes. There

Asset	Quantity	Asset Size (MB)	Total Size (MB)
Executable & Config	2	3	6
SceneDBs & Scripts	32	2	64
Characters	10	5	50
Props	200	.25	50
Dialogs	50	3	150
Sound Libraries	50	1	50
Songs	8	3	24
Scene Backgrounds	7	.3	2.1
Scrolling Backgrounds	1	3	3
Intro & Outro	2	100	200
Total			**599.1**

PC Windows and MAC OS 8.6

LEGO is being developed as both a PC Win95/98/2000 and MAC OS 8.6 application. It will use normal C++ coding methods as interpreted by the Monkey Business Coding style document. Unless otherwise stated please refer to that document for style questions. Like any well-behaved application it will gracefully handle losing focus, saving and restoring screens as necessary. As a windows executable we will be using *Microsoft* DirectX in our system level APIs. On the MAC we will be using GameSprockets. Sound will be using the DirectSound and SoundSpockets with SoundManager 3+ libraries. User input will be interfaced using DirectInput and InputSprocket for the keyboard, joystick, and mouse.

LEGO Tools

For the data driven model to work the *LEGO* tools will have to be tightly integrated and easy to use for non-technical designers and artists. The *LEGO* tool set will include:

NOGGIN - AI Script Compiling tool

All the character and prop object behaviors will be scripted in a text editor and compiled to allow team members to manipulate the properties of these objects including responses to the players input or timed events. All the scripts will use a simple BASIC like programming language that will allow fine control over an object's behavior. The tool to do this is the key to allowing the most control over game play. If we cannot budget and schedule this tool, programming will have to modify all game variables and code all AIs at compile time.

SAM - Scene Asset Manager Tool

The *LEGO* team has decided to build a tool to construct the scenes and build asset databases. Members of the ART and ENGINEERING staff will both use the tool to create activity scenes.

SPR – Sprite Converter Tool

The sprite converter tool reads a 24bit AVI file and trims the color components back to 15bit (no dither). It performs differential frame compression using a format similar to Autodesk FLC. The resulting file is saved as a SPR.

Background – Background Conversion Tool

Converts the TGA format into a section of 64x480 pixel strips used in the game for making scrolling backgrounds work.

All other asset data in *LEGO* is standard for any Windows application and requires no additional work or modification by the engineering team.

Miscellaneous Systems – not documented elsewhere

Debug support

The PC version will support sending messages to the Debug Window in the Visual C++ debugger. It can also open a console window and send it console like output, including printing at an (X,Y) coordinate. There will also be the capability for sending messages to a log file. The plan to support dual monitors via DirectX is also being considered. The debug module has class that supports the console and several classic C-functions for the various printf-like debug routines.

The impact of debug support on the production schedule is minimal. The console and debug window support exists in the prototype, and we will add dual monitor capability over time if necessary.

Installation

As outlined in the discussion early about the installation system we will be using InstallSheild and InstallVice. A specific auto-boot program may be necessary on the PC and can be completed during the ALPHA phase of development. It will check the registry to find if the program has already been installed and call the installer or play the game based on what it finds there. It may also use Randy's system information routines to insure that the PC meets minimal requirements. Typically an auto-boot program requires a bitmap from art and approximately 4 hours of programming time to implement. On the MAC a window is automatically opened and the install or play icons will be visible.

Demo Mode

If necessary the opening animation will replay if the sign-in screen is left unattended for too long. No demo or attract mode is present in the features of LEGO.

Localization

LEGO will use an extended ASCII character set that will allow for foreign language conversion. Menu and dialog text will be kept in a resource file that holds all text. Any text in art will have to be changed by the artists. Art for fonts will follow this extended ASCII character set.

Time will be scheduled at the end of the American English version to convert assets, scenes and scripts for use in foreign language versions of *LEGO.*

Development Directory Structure

The network drive for *LEGO* is kept on NT2\MONKEY. On each engineer’s machine it is mapped to the “M:” drive. The directory structure is

NT2				The server
	Monkey			The project directory
		VSS		The Visual SourceSafe backup of all code
		Tools		Any tool executables to convert assets
		CodeTemp		A Junk directory to share code files
		Final Preschool		PC and MAC installers and executables
			Scripts	Object NOG files
			Scenes	Scene SAM files
			Characters	Character SPR files
			Backgrounds	Background TGA files
			Props	Non-Character SPR files
			Interface	Font, cursor, and particle TGA files
			Sounds	Effect WAV files
			Dialogs	Character speech and song WAV files
			Music	MOD or MIDI files with samples
			Movies	Intro and End MPG movies
		Final Kindergarten		PC and MAC installers and executables
			Scripts	Object NOG files
			Scenes	Scene SAM files
			Characters	Character SPR files
			Backgrounds	Background TGA files
			Props	Non-Character SPR files
			Interface	Font, cursor, and particle TGA files
			Sounds	Effect WAV files
			Dialogs	Character speech and song WAV files
			Music	MOD or MIDI files with samples
			Movies	Intro and End MPG movies

Milestone Finish Criteria

Just before each milestone the whole team will spend at least 3 days integrating code and polishing the deliverables. After each milestone the Lead Engineer will do a code review to insure that the code generated for the milestone was done according to project policy and standards and that no shortcuts or hacks were made just to complete the milestone. After code review any adjustments to the engineering schedule will be made.

LEGO Risk Analysis

Engineering Staffing

All engineers are hired for the beginning of production. If for some reason an engineer is no longer available to the team then a replacement will have to be acquired as soon as possible. If a replacement can not be found internally, an external solution will be sought with urgency. While the scheduling of *LEGO* includes times for vacations, conferences and some training, no extended down time is anticipated.

The Macintosh consultant is a backup plan if needed to lessen the impact the conversion to MAC will cause. The engineering team plans to do as much as possible without the consultant but can not let the conversion slip the schedule of the PC product.

NOGGIN - AI Script Compiling tool

If we cannot budget and schedule this tool, programming will have to modify all game variables and code all AIs at compile time.

SAM - Scene Asset Manager Tool

If we cannot budget and schedule this tool, programming will have to construct these as data structures in the compiled code.

SPR – Sprite Converter Tool

If we cannot budget and schedule this tool, programming will have to construct these as compressed numbered frames of TGA files.

LEGO Engineering Roles

Lead Engineer – Randy Angle

Experience:

- 21 Years Programming in assembly, BASIC, FORTH, …
- 15 Years in object oriented C
- 3 Years in C++
- 8 Years Embedded Systems Programming and Lead Project Engineer
- 22 Years Designing and Small Scale Publishing Role-playing Games
- 17 Years Amateur Designing, Programming, and doing Art for Computer Games
- 8 Years Professionally making Computer & Video Games as Lead Designer or Lead Engineer

Credits:

- Goblins – 1998 PC & Playstation, Lead Engineer
- Reapers – 1998 PC, Lead Engineer
- Last Express – 1997 Sony Playstation & PC, Lead Playstation Engineer and Conversion Designer, PC support programmer
- Aftermath Engine – 1995 Sony Playstation & PC-CDROM, Lead Engineer, Technical Designer
- Star Trek: The Next Generation – 1994 Genesis & SNES cartridge, Lead Designer & Programmer
- Chess Maniac 5 Billion & 1 – 1994 Voice acting
- Falcon 3.0 – 1992 Testing, Support Supervising and Training
- Super Tetris – 1991 Lead Tester

Skills:

- Project Leadership
- Game Design
- AI Systems
- Overall Game Architecture
- High Performance File Systems
- Compression Technologies
- Digital & Analog Electronics & Circuit Design
- In-Circuit Emulation
- Assembly Language Optimization
- Programming Language Design & Interpreters
- Structures and Algorithm Design
- Sound Design and Programming
- High Speed Graphic Optimization

Lead Engineer – Randy Angle - Continued

Responsibilities:

- Oversee the technical end of *LEGO*
- Schedule of all *LEGO* engineers
- Participate in the pre-production process with other leads
- Interface with the other leads to ensure good communication
- Hire engineering resources
- Evaluate, Research and Develop new technologies
- Code design for the overall structure of *LEGO*
- Manage communication between all of *LEGO* engineering
- Keep teams tasks on schedule as needed
- Provide for training engineers in areas that need improvement
- Design, plan and implement the Main and Game loops for *LEGO*
- Design, plan and implement the AI technology for *LEGO*
- Design, plan and implement the Particle system
- Design, plan and implement the Procedural Texture system
- Design, plan and implement the Record Keeping system
- Design, plan and implement the Debug support system
- Design, plan and implement the Installer
- Help design, plan and implement the Animation system
- Help design, plan and implement the Graphics system
- Help design, plan and implement the Scrolling Environments system
- Design, plan, implement and support AI Scripts as needed
- Support SPR tool as needed
- Support NOGGIN tool as needed
- Perform Code Reviews at each milestone
- Provide any engineering documentation during the project

Engineer – Greg Sabatini

Experience:

- 15 years programming in BASIC, QuickBASIC, PASCAL...
- 4 years C programming experience
- 3 years C++ experience
- 2 years project scheduling and coordination

Credits:

- Hot Wheels Turbo Racing, 1999, Nintendo and Cross Platform Development
- Byzantine: The Betrayal, 1997, Additional Programmer
- XOC, 1997, Programmer
- EDISON, 1997, Programmer

Skills:

- Win95, Nintendo 64
- C++, C
- DirectX
- 2D graphics (sprites, alpha matting, image processing)
- Graphics optimization
- Multi-platform Sound programming
- Audio/Video synchronization

Responsibilities:

- Evaluate, Research and Develop new technologies
- Work with other engineers to maintain uniform development procedures
- Take responsibility for version control and regular backups
- Design, plan and implement the Movie system
- Design, plan and implement the Streaming Dialog system
- Design, plan and implement the Sound system
- Design, plan and implement the Scene Asset Management tool
- Design, plan and implement the Input system
- Design, plan and implement the Resource system
- Help design, plan and implement the Scrolling Environments system
- Help design, plan and implement the Character Animation system
- Help design, plan and implement the Graphics system
- Help write scripts for activities
- Take responsibility for CD Burns & Integration before milestones

Headers & Footers

Besides using all the coding standards outlined in the Monkey Business Programming Policies document, all CPP & H files in *LEGO* will use a footer comment that has the following keyword for Visual SourceSafe:

Header:

```
////////////////////////////////////////////////////////////////////////////////////////
//
//      Project  : DUPLO Inventures
//
//      $Workfile: $
//
//      Comments :
//
//      Creator  : Randy Angle
//
//   Last $Author: $
//
//      Created  : 9/23/1999
//
//      $Revision: $
//
//   Copyright (c) 1999 Stormfront Studios
//
/////////////////////////////////////////////////////////////////////////////////////////
```

Footer:

```
/*
 * $History: $
 */
```