Little Rubi's Revenge

Team Alien
GAM 541

Spring 2012

Vasileios	Producer/Tools /AI Programmer
Santosh	Technical Director/Engine Programmer/physics
Sree Harsha	Game Designer/Game play/Physics/Sound Programmer
Zheng	Graphics Programmer/Engine Programmer

Table of contents

Overview

We use component based design architecture for our project with message based communication. The major global components involved are:

Graphics: Handles the rendering part of the project. We use Directx10 API with shader based implementation.

Physics: We use simple 2D based discrete physics with circular and axis aligned rectangular shapes.

AI: Our game is not very AI intensive so State machines can fulfill most of our AI related requirements.

UI: We will use stylized artwork with vivid colors and matching HUD, simplistic menus, and easy controls.

File I/O: We use XML serialization for level and game data. MD5 for 3D models, and debug text output in TXT file.

Memory manager: We don't feel a need for a memory manager at this point but it will be implemented if necessary.

Audio: We use FMOD for audio processing.

The frame rate controller module of the engine is responsible for running the game loop. For every frame every system's update function is called. Individual systems can implement their update functions as needed. Systems can interact with each other by broadcasting messages.

The major object components involved in the project are:

Transform: Keeps track of the current transform values of the object.

Mesh: It's a graphics component representing a 3D model.

Sprite: It's a graphics component representing a 2D sprite.

Animation: It's a graphic component which can be animated.

Body: It's a physics component that provides physics capabilities.

AI: It is a AI component.

Composition of various components are created runtime or as described by the level files. Blueprints of the archetypes are detailed in the game data xml file which will be used to create a compositions dynamically. Factory is responsible creating, destroying, managing compositions. Each composition has an id, assigned by the factory, which can be used to access a composition. The various components inside a composition can communicate with each other by broadcasting messages on the composition which will then be forwarded by the composition to all its components. The components which care for that message can react to it and others can just ignore.

We don't plan on using multithreading as of now. We may use it for preloading assets for levels at later point.

Graphics Implementation

We are using DirectX 10 API for our graphics implementation. So everything is going to be shader based implementation. Differed shaders and geometry shaders will be used wherever necessary. Sprites and textures lists will be part of the level file. The actual textures will be loaded using the DirectX 10 native functions. We are using Blender to create our 3D models which will be of the output format md5. We are going to write a custom parser to read the md5 file into the graphics system. We think implementing particle systems, shadow maps and skeletal animations are near possibility.

Behavior Implementation

Our game is 2.5D platformer with nominal AI in it. It will make use of Pattern movements for certain enemy behaviors or animations. A* will be used wherever necessary for path finding. State machines will suffice for the rest of the enemy behaviors .

Physics Implementation

Our game will use simple 2D based physics engine which will have a look and feel of 3D. Simple circular and axis aligned rectangular collision detection will be employed. Euler integration will be used. It will also feature quad partitioning. If time permits we hope to implement random aligned rectangular collision.

Multiplayer Implementation

There is no multiplayer implementation.

Coding Methods

The various coding conventions used are:

- All member variables of a class will be preceded with 'm_', ex: m_variable1
- all local variables will begin with the smaller case acronym of the type followed by the Word describing the variable starting with upper case. If more than one word is used to describe the variable then the first letter of those words will be upper case with no spaces or underscores between the words. ex: an int which describes the right side force: iRightForce
- All function names will start with the smaller case letter. If more than one word is used to describe the function then the first letter of those words will be upper case with no spaces or underscores between the words. ex: makeThemCry()

- All class names will begin with Upper case letter: ex: class Timer
- All constants will be all upper case. ex: EPSILON
- All other code formatting is left up to the programmer's discretion.
- We will use Doxygen for documentation.
- We use SVN subversion for source control with each commit accompanied by a detailed comment.
- Files will be have the same name as the major class written in it with no more than 2 classes per file.
- We have folder structure for major systems and modules such as: Debug, Engine, Factory, Serialization, Utility, Graphics, Physics, GameLogic, etc. All code files will go in their respective folders.

Debugging

The various debugging tools that we have are

- Simple output console
- Output to text file
- Debug drawing system
- Assertions
- Simple performance viewer(shows only FPS at the moment but will be enhanced later)
- Various in-game variables can be watched runtime
- Frame pause and step-by-step frame execution
- Toggle-able debug mode

Tools:

We use GLEE2D editor to create our levels which outputs an XML file. We use XML serialization to read XML level and data files. MD5 parser to read MD5 models. We use FMOD for audio processing. Blender, Maya, Photoshop to create our models, textures.