

# LuaGame: A framework for developing games in Lua for the Digital TV

Diego Barboza    Débora Muchaluat-Saade\*    Esteban Clua

\*Laboratório MídiaCom

Instituto de Computação, Universidade Federal Fluminense, Brasil

## Abstract

Digital Television is a growing platform for software development. Interactive applications are an important part of this platform that aims at higher image and sound quality, as well as a communication channel with the viewer. Digital games are an engrossing media that is making its way into various devices, such as mobile phones, the Internet and Digital TV systems. However, game development may be a high demanding and complex area and, without the proper tools, the development for limited resource platforms as set-top boxes could be a harsh task. Therefore, this paper presents a framework that facilitates this process, allowing developers to use its base structure and focus on the creation of game specific content in a component-based development model. We propose a framework called *LuaGame*, present its architecture and show its usage in a practical example.

**Keywords:** Ginga, Lua, Digital TV, frameworks, games

## Authors' contact:

{dbarboza, esteban}@ic.uff.br  
\*debora@midiaacom.uff.br

## 1. Introduction

A framework can be defined as a set of classes that presents an abstract design for solutions of a family of related problems and supports reuse [Johnson and Foote 1988]. Those classes should be flexible and allow extensions, requiring low effort on the development of applications, where it should be necessary to focus only on their particularities.

In the game development context, game engines are employed as auxiliary tools, combining frameworks with higher level tools, such as visual editors and integration tools.

Interactivity is one of the guidelines for the Brazilian Digital Terrestrial Television System (SBTVD-T) [Brasil 2006] and it should be used to provide different kinds of applications, such as electronic guides, shopping channels, bank and educational services, among others [Barbosa and Soares 2008]. Digital games are an interactive application type that could help Digital TV become popular in the country, since the national industry is in an expansion and growing moment [Ferreira and Souza 2009].

In this work, we propose a novel game development framework architecture for Digital TV using *Lua* [Sant'Anna et al. 2008] as the programming language. The purpose of this framework is to model common behavior of various games and provide tools that allow the development without coding ordinary tasks, such as resource management, control loop and user input treatment. This way, we provide a basic application structure that the developer can use and focus only on filling its empty spaces, which are specific components and behavior for his game.

This paper is organized as follows: Section 2 discusses some related work and Section 3 presents some concepts about game development tools. Section 4 describes the proposed framework architecture and Section 5 briefly presents a sample application built with the *LuaGame* framework. Section 6 presents the conclusion and suggestions for future work.

## 2. Related Work

ATHUS [Segundo et al. 2010] is a framework proposal that uses *Lua* adding a layer between applications and the digital TV middleware. This framework has some basic functionality, such as drawing images to the screen, controlling application speed, checking for collision and creating menus.

*TuGa* [Ferreira and Souza 2009] presents a two-layer middleware in *Java*, to be used together with *Ginga-J*, considering its reference implementation [ABNT/CEET 2007]. The layers separate the code for hardware abstraction and the code that provides game functionalities. This is an interesting proposal, because it makes it easier to port the tool to other platforms and reuse most of its code, at the same time that games developed using the framework should not need any modification to run on different platforms.

The present work is an extension of *GingaGame* [Barboza and Clua 2009], a framework developed in *Java* using the reference implementation [ABNT/CEET 2007]. The framework's proposal is to provide an application model that controls the game's update and drawing functions, manages its resources, controls objects' lifecycle, manages scenes and handles user input. We separate the framework code from platform specific code, aiming at easier portability, and providing components for game developing. Also, we aim at software reuse, so we provide framework classes that could be extended for defining reusable

components that could be used by other applications with minimum effort. In this work, we refine *GingaGame*'s architecture. This architecture was revised and ported to work with *Lua*, so it could be executed as NCL nodes [ABNT 2007] and work with middleware *Ginga*.

### 3. Game framework architectures

There are various levels of game development tools. They range from lower abstraction levels, where different graphics APIs and libraries (for managing user input, physics, scene graph, *etc*) are combined providing a more flexible, yet complex, environment, to higher levels with drag-and-drop graphical interfaces that allows the making of a game in a visual manner, with minimal or even no programming requirement.

Game development tools typically provide a basic set of functionalities necessary for building games, and may include additional elements targeting at any specific stage of this process.

These functionalities include: **Control loop** (defines the application's life-cycle); **Scene graph** (used for organizing a game scene); **Render** (component responsible for communicating with the graphics hardware); **Input** (provides an interface with input devices); **Content management** (handles external content loaded into the game); **Physics** (updates the interaction between dynamic objects in a scene and handles collisions, friction, torque and other physics effects); **Scripting** (some tools employ scripts for game specific code, separating it from the native tool code); **Multimedia support** (controls music, sound effects and video playback); **Network** (provides an interface for connection and message exchange in multiplayer games over a network); **Artificial intelligence** (components used for defining the behavior of some game objects using an AI technique); **Editor** (graphical editors are provided by some high level tools to edit scenes, bind relations between objects, position actors and so on).

There is a relation between the degree of freedom provided by a game development tool and its construction, maintenance and usage. Low level tools offer more customization but are more complex and demand more development time. High level tools, otherwise, allow the user to create games with less programming, although those tools are usually more restrict on the kind of games they support. A complete toolset is of great importance on the game development process, providing tools and APIs that programmers should use for coding the game's logic, editor tools for level designers, animation tools for animators and so on.

### 4. Lua Game framework

This work describes a framework for game development that encapsulates common tasks and provides an application model that the developer may use to add components and define the desired behavior for its game. This framework is called *LuaGame* and it can be used for developing games for digital TV systems based on the *Ginga* middleware, such as the Brazilian Digital TV System [ABNT 2007] and IPTV systems [ITU-T 2011].

The framework's architecture was developed to provide software reuse. Its components may be employed in different contexts and can be extended for creating new components. Figure 1 presents an overview of the framework's architecture and each component is described right below. This is the current state of the framework, but more components should be added in the future.

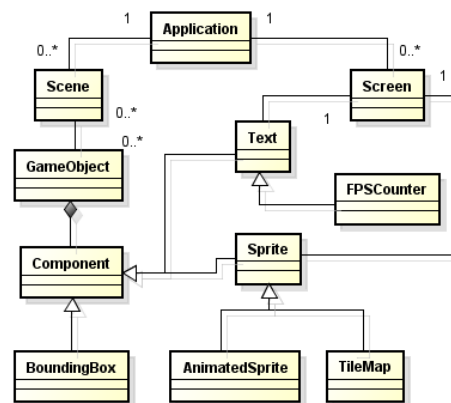


Figure 1: Architecture of the *LuaGame* framework

- **Application:** this component manages the whole game. This class handles the game loop, stores and defines which scenes will be displayed at a time, and receives input events and propagates to the remaining components. Also, it controls the game speed and provides time interval between frames to the components, so the behavior of the components should be time independent;
- **Component:** provides a common base interface for building game components. Components that inherits from this class may specify input, update and draw functions that are automatically called by the framework;
- **GameObject:** a game object is a container used for grouping various components together in a unique object. Conceptually, each component may be used to model a part of a whole and the game object is the aggregation of those parts. For instance, a game object "car" may be defined as the aggregation of components such as wheels, body, motor, steering wheel and *etc*;
- **Scene:** a scene is used for decomposing the game into logic units that are presented one at a time. A scene is the composition of various components and game objects and may be shown or hidden as

necessary. Scene samples include the main menu, each game level, high score screen, among others;

- **Screen:** this class prepares the draw area on the beginning of each frame, renders the images drawn during the frame, and sends the final resulting image to the graphics device;
- **Sprite:** provides functions such as color keying and image cropping for drawing images in the game;
- **AnimatedSprite:** an animated sprite is an extension of the Sprite class that provides an interface to define image crop and frame sequences to build animations. The animation may be controlled by the size of each frame, the frame interval and the update speed;
- **BoundingBox:** defines a bounding box around an object for intersection checks;
- **Text:** used for rendering texts on screen, allowing the definition of the string to be printed, its color, font, font size and text orientation;
- **TileMap:** a brick map used for drawing two-dimensional scenery from an image that is segmented into blocks, each one representing part of the scenery. This class allows the configuration of the block size on the source image, the map size within the game, and the content of each cell;
- **FPSCounter:** component used for debugging that prints the current frame rate on the screen.

With *LuaGame*, we provide some of the functionalities mentioned in Section 3. Our framework exposes an application model that controls the game loop, provides access to rendering and input functions, manages external content loading, handles basic collision detection and defines a common component interface for scripting in *Lua*. Artificial intelligence components, multimedia and network support, as well as a more refined physics engine and a visual editor will be handled in future work.

## 5. Application examples

This section presents a sample application developed using our framework. We chose a simple application as demonstration, so we could present its code entirely and show how the framework is used.

The application simulates a dice roll on the screen, using the component *AnimatedSprite* to represent the dice object. The application waits for the user input and rolls the dice when a key is pressed. When this happens, the dice is animated and a countdown is started. When the countdown ends, the animation is stopped and the result is presented. With this application, we demonstrate the usage of animated images and its control methods. Also, we define behavior to the scene and read user input.

For developing this application, we define a scene that will present the dice image, process user input and run the countdown. The scene is defined by its

declaration and the declaration of its attributes. Later, we need to tell to the framework that the defined scene inherits from the framework's Scene class, using the method *Scene:new*. The attributes are: a reference to the dice image, a time counter and a time limit (both in milliseconds), and a flag that tells if the dice is rolling.

Once the class is defined, the *load* method (Listing 1) is used for loading the image (Figure 2). This image has one line with six animation frames (*framesX/Y*) and its animation (*fps*), configured as five frames per second. The image is loaded with the method *load* and centralized on the screen using its width and height (*cropWidth/Height*).

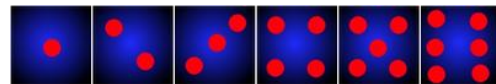


Figure 2: Complete dice image. It is cropped by the animation class and only one face is drawn at a time

```
function DiceApplication:load()
    self.dice = AnimatedSprite:new()
    self.dice.framesX = 6
    self.dice.framesY = 1
    self.dice.fps = 5
    self.dice:load('media/dice.png')
    self.dice.x
        = 320 - self.dice.cropWidth / 2
    self.dice.y
        = 240 - self.dice.cropHeight / 2
    self:addComponent(self.dice, 'dice')
end
```

Listing 1: Dice image load

User input is processed by the method *keyPressHandler*. The key is checked and, based on the key pressed, we either exit the application or roll the dice, if it is not rolling. The dice animation is started by the method *dice:start* that makes the animation frames change five times per second, until a command makes it stop. We also set *rolling* to true so the application knows the dice is rolling (Listing 2).

```
function DiceApplication:keyPressHandler(evt)
    if evt.key == 'e' then
        Application.showSceneByName(
            'ApplicationsMenu')
        return
    elseif (evt.key == 'r' and not self.rolling)
    then
        self.rolling = true
        self.dice:start()
    end
end
```

Listing 2: Handling user input

The update method controls the dice rolling time. After the dice is rolled, the elapsed time is accumulated using the delta time between frames provided by the application guaranteeing that it is a time close to the wall clock and frame rate independent, i.e., the application seems to run at the same speed, even in different environments with variable frame rates.

The accumulated time is compared to the target time, and when it is equal or larger than the target, the dice roll should stop. The animation is then paused (*dice:pause*) and the last animation frame is presented

as the roll result. We set *rolling* to false, so a new roll should be triggered by user input (Listing 3). Figure 3a shows the resulting application running.

```
function DiceApplication:update()
    if (self.rolling) then
        self.elapsedTime
        = self.elapsedTime
        + Application.deltaTime
        if (self.elapsedTime >= self.targetTime)
        then
            self.rolling = false
            self.dice:pause()
            self.elapsedTime = 0
        end
    end
    Scene:update(self)
end
```

Listing 3: Dice roll update method

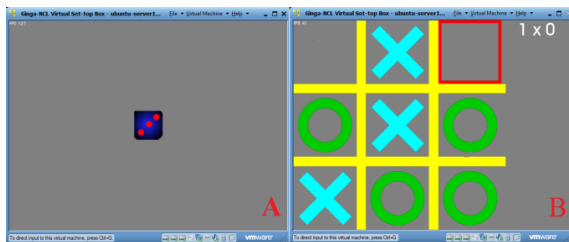


Figure 3: Sample applications developed with the framework.

We also developed a tic tac toe application (see Figure 3b) that works with multiple images (the tic tac toe frame, the red box that works as a cursor, and the circle and cross used to represent the player's actions), text output (for printing the score), more complex user input (using the keyboard arrows to move through the cells and using the confirm button to make a play, and defines a game logic that solves the tic tac toe to find out if one of the players is the winner.

Using our framework, for both this application and the previously mentioned dice application, we could focus on programming only the game specific part, defining scenes and components to be added in the game, without concerns on how the underlying framework is handling update and draw calls, how the scenes are presented, or even how images are loaded, animated and drawn on the screen.

## 6. Conclusion

In this work we presented an initial step of the creation of a generic and extensible framework for developing games for Digital TV systems using the *Lua* language. The developed framework abstracts common development task, managing scenes and resources, handling user input, drawing and updating components, *etc.* This allows the developer to focus on game specific code that models the behavior of its game entities.

Some application samples were developed to validate the framework and demonstrate its functionalities. The code of a complete application was presented and it could be used as a base sample of the framework usage.

Future work will include the definition of more components to be developed and included in the framework, enhancing its functionalities. We will also develop more examples and approach functionalities, such as a basic physics engine and interaction with NCL documents for audio and video playback.

## References

- ABNT/CEET, 2007. Televisão digital terrestre - Codificação de dados e especificações de transmissão para transmissão digital – Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais.
- ABNT, 2007. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações.
- BARBOSA, S.D.J. AND SOARES, L.F.G., 2008. TV digital interativa no Brasil se faz com Ginga: Fundamentos, Padrões, Autoria Declarativa e Usabilidade. In T. Kowaltowski & K. Brealman (orgs.) Jornada de Atualizações em Informática - JAI 2008. Rio de Janeiro, RJ: Editora PUC-RIO, 2008. pp.105-174.
- BARBOZA, D. AND CLUA, E., 2009. Ginga Game: A Framework for Game Development for the Interactive Digital Television. SBC – Proceeding of SBGames 2009. Available from: [http://usuarios.rdc.puc-rio.br/sbgames/09/\\_proceedings/dat/\\_pdfs/computing/Proceedings\\_Computing\\_Full.pdf](http://usuarios.rdc.puc-rio.br/sbgames/09/_proceedings/dat/_pdfs/computing/Proceedings_Computing_Full.pdf) [Accessed 02 Aug 2012].
- BRASIL, 2006. Decreto n 5.820, de 29 de Junho de 2006. Implantação do Sistema Brasileiro de Televisão Digital Terrestre - SBTVD-T. DOU de 27/11/2006. [http://www.planalto.gov.br/ccivil\\_03/\\_Ato2004-2006/2006/Decreto/D5820.htm](http://www.planalto.gov.br/ccivil_03/_Ato2004-2006/2006/Decreto/D5820.htm) [Accessed 01 Aug 2012].
- FERREIRA, D. A. AND SOUZA, C. T., 2009. TuGA: Um Middleware para o Suporte ao Desenvolvimento de Jogos em TV Digital Interativa. Centro Federal de Educação Tecnológica do Ceará. Available from: [http://tuga-sdk.googlecode.com/files/TuGA\\_Middleware.Jogos.TV\\_Digital\\_v1.6.pdf](http://tuga-sdk.googlecode.com/files/TuGA_Middleware.Jogos.TV_Digital_v1.6.pdf) [Accessed 01 Aug 2012].
- ITU-T, 2011. Recommendation ITU-T H.761. Nested context language (NCL) and Ginga-NCL. Available from: <http://www.itu.int/rec/T-REC-H.761-201106-I/en> [Accessed 01 Aug 2012].
- JOHNSON, R. AND FOOTE, B., 1988. Designing Reusable Classes. Journal of Object Oriented Programming Volume 1, Number 2 (June/July 1988). 22-35. Available from: <http://www.laputan.org/drc.html> [Accessed 01 Aug 2012].
- SANT'ANNA, F., CERQUEIRA, R. AND SOARES, L., 2008. NCLua: objetos imperativos lua na linguagem declarativa NCL. Proceedings of the 14th Brazilian Symposium on Multimedia and the Web, October 26-29, 2008, Vila Velha, Brazil. DOI: 10.1145/1666091.1666107.
- SEGUNDO, R., SILVA, J., TAVARES, T., 2010. ATHUS: A Generic Framework for Game Development on Ginga Middleware. SBC – Proceedings of SBGames 2010. Available from: <http://sbgames.org/sbgames2010/proceedings/computing/full/full12.pdf> [Accessed 02 Aug 2012].