

UTILISATION DE LUA COMME LANGAGE DE SCRIPT DANS LES JEUX CODES EN JAVA

Lua est un langage de programmation qui a été bien accepté par la communauté de développement de jeux comme langage de script. C'est parce que Lua propose une série de méthodes pour permettre l'utilisation des fonctions C dans le code Lua et vice-versa. Lorsque les développeurs choisissent de coder un jeu en Java, Lua n'est apparemment plus une option. L'objectif de ce travail est de montrer que Lua peut également être utilisé comme langage de script pour des jeux codés en Java. Pour cela, les développeurs ont juste besoin de connaître la bibliothèque LuaJava et quelques conseils.

INTRODUCTION

Au fil des années de développement de jeux, l'utilisation de scripts est devenue de plus en plus populaire. Aujourd'hui, presque tous les jeux utilisent des scripts dans de nombreux aspects différents. Décrire les attributs de différents personnages, objets, créatures, coder l'intelligence artificielle, les dialogues, les événements, l'historique ou même dans les fichiers de configuration. Les scripts peuvent même fournir aux joueurs un outil pour créer leurs propres modifications de jeu.

Les scripts sont essentiels de nos jours. Lorsque le langage Lua est né, la majorité des développeurs de jeux codaient en C / C ++, et ils ont adoré Lua car avec presque aucun effort, ils pouvaient échanger des données avec un langage de script facile à écrire, à lire et même à étendre. C'est parce que Lua est open source et également codé en C. C'est ainsi que Lua est devenu populaire.

Comme un petit exemple de la popularité de Lua entre les communautés de développement de jeux, nous pouvons énumérer quelques jeux notables qui utilisent Lua comme langage de script et avec d'autres rôles. Crysis, Far Cry, Grim Fandango et Escape from Monkey Island, Grand Theft Auto San Andreas, Ragnarok Online, SimCity 4, Star Wars Battlefront et Battlefront 2 également Empire at War, World of Warcraft et bien d'autres.

De nos jours, nous pouvons voir beaucoup de jeux et de moteurs de jeux qui utilisent Java comme langage de programmation. L'idée de ce travail est d'offrir aux développeurs qui utilisent Java un moyen d'utiliser également Lua dans leurs scripts de jeu. Les avantages de l'utilisation d'un langage de script sont bien connus. Il permet un développement rapide et un déploiement facile, car vous n'avez pas à recompiler le code après chaque modification [Cassino et al., 1999].

De plus, les langages de script offrent une bonne intégration avec les technologies existantes telles que les langages de programmation et sont faciles à apprendre et à utiliser. Et, nous pouvons également dire que les langages de script fournissent un codage dynamique car son code peut être généré et exécuté en runtime.

Mais, comme tout ce qui comporte des avantages rassemble des inconvénients, les langages de script ne sont pas différents. Ils supposent la présence d'un «vrai» langage de programmation. Ils ne sont pas conformes aux meilleures pratiques en génie logiciel et en structure de code, telles que l'orientation des objets. Et ils sont également adaptés à une application spécifique, telle que PHP pour le World Wide Web. C'est là que Lua entre en main.

Le langage Lua n'a pas été créé uniquement pour être un langage de script, mais un langage de programmation court, efficace et extensible [Ierusalimschy, 2006]. Il présente donc les avantages des langages de script, mais ne présente pas d'inconvénients.

Lua offre un développement extrêmement rapide et est également très facile à apprendre, à écrire et à comprendre. C'est interprétable donc, vous n'avez pas besoin de compiler. Il propose également du code dynamique et s'intègre aux programmes C. Et avec la bibliothèque LuaJava, s'intègre également avec Java.

À propos des inconvénients du langage, nous pouvons dire qu'il est vrai que si le programmeur n'utilise pas de discipline, le code Lua peut devenir un gâchis. Mais la syntaxe du langage permet l'implémentation d'une orientation objet ou d'un développement basé sur des composants [Ierusalimschy, 2006]. Lua est également un langage à usage général, il peut donc gérer de nombreuses applications différentes à l'aide de nombreuses bibliothèques d'extensions telles que LuaSQL, CGI Lua et lup Lua et, au moins mais pas en dernier, LuaJava.

TROIS CONSEILS POUR SCRIPTER DES JEUX À L'AIDE DE LUAJAVA

LuaJava est un outil de script pour Java. Le but de cet outil est de permettre aux scripts écrits en Lua de manipuler des composants développés en Java [Cassino et al., 1999]. LuaJava permet l'intégration entre Lua et Java dans les deux sens: manipulation d'objets Java par des scripts Lua et manipulation d'objets Lua par des

programmes Java [Cassino et al., 1999]. L'accès aux composants Java se fait à partir de Lua sans avoir besoin de déclarations ou de tout type de prétraitement. Dans cet article, nous avons choisi de travailler à l'aide d'objets Java dans les scripts Lua, car de cette façon, nous pouvons créer des objets génériques et créer des scripts pour changer la valeur de ses attributs en produisant un code propre et facile à comprendre.

La première chose que nous devons faire lorsque nous voulons gérer des objets Java à l'intérieur de Lua est de créer un LuaState. Le LuaState contrôlera l'accès à l'environnement Lua.

Nous pourrions créer un LuaState en procédant comme suit:

```
LuaState luaState;  
  
luaState = LuaStateFactory.newLuaState ();
```

Après avoir créé un LuaState, nous devons maintenant ouvrir les bibliothèques LuaJava.

Cela se fera par l'instruction suivante:

```
luaState.openLibs ();
```

Maintenant que nous avons construit notre environnement, nous sommes prêts à commencer à écrire du code Lua. Cela devrait être fait dans un fichier Lua (.lua). Maintenant que nous avons un environnement et un fichier Lua, nous avons juste besoin de tout mettre ensemble. Cela devrait être fait en appelant la méthode / tt LdoFile. Cette méthode indique à l'environnement Lua de lire le fichier Lua passé en paramètre.

L'instruction devrait être:

```
luaState.LdoFile (<emplacement luafile>);
```

Une fois que nous avons vu toutes les méthodes pour créer un environnement Lua et l'utiliser dans un projet Java, rassemblons-les dans un exemple classique de Hello World.

```
void Main () {  
  
    LuaState luaState;  
  
    luaState = LuaStateFactory.newLuaState ();  
  
    luaState.openLibs ();  
  
    luaState.LdoFile ("helloworld.lua");  
  
    luaState.close ();  
  
}
```

Fichier helloworld.lua:

```
imprimer ("Bonjour tout le monde")
```

LuaJava offre également de nombreuses autres fonctionnalités. Nous allons examiner de plus près deux de ces fonctionnalités qui seront importantes pour notre schéma de script figurant dans nos deuxième et troisième conseils. La première fonctionnalité est la façon dont vous appelez une fonction Lua à exécuter dans la portée Java. La seconde est de savoir comment utiliser un objet Java dans un fichier Lua.

Pour utiliser une fonction Lua en Java, vous devez obtenir la variable globale Lua qui stocke la fonction, ce qui sera fait en appelant la méthode `getGlobal` en passant le nom de la fonction en paramètre:

```
luaState.getGlobal ();
```

Après cela, nous utilisons simplement la méthode d'appel `LuaState`. Cette méthode est particulièrement importante car son premier paramètre est le nombre de

paramètres transmis à la fonction. Mais comment passez-vous ces paramètres? Cela sera expliqué par notre troisième conseil.

Pour passer un objet Java à Lua, nous utiliserons la méthode `pushJavaObject` et le passerons comme paramètre d'une fonction.

La phrase d'instruction est:

```
luaState.pushJavaObject (
```

Ainsi, avec ces deux fonctionnalités, nous pourrons passer un objet Java en tant que paramètre à la fonction Lua. Utilisez-le dans cette fonction et appelez la fonction dans la portée Java. Cela nous permettra de créer une classe qui gérera les scripts pour nous comme nous le verrons dans la section suivante.

Construire une classe LoadScript

Maintenant que nous avons vu comment LuaJava fonctionne, construisons une classe qui gérera tous les scripts de notre jeu. Cette classe n'aura qu'un seul attribut, notre `LuaState`. Le constructeur de notre classe ne recevra qu'un seul paramètre qui sera le chemin de notre fichier de script. À l'intérieur de notre constructeur, nous allons créer l'environnement Lua avec les méthodes `LuaStateFactory.newLuaState` et `openLibs`, et exécuter le fichier Lua reçu en paramètre par le constructeur.

Notre classe aura deux méthodes: `closeScript` et `runScriptFunction.closeScript`, appelez simplement `luaState.close` pour mettre fin à l'utilisation de l'environnement Lua. `runScriptFunction` obtiendra une fonction Lua reçue comme paramètre et l'appellera en passant un objet Java, également reçu comme paramètre, à cette fonction. Nous avons construit une classe pour gérer tous nos scripts. Dans la section suivante, nous verrons comment nous utilisons cette classe.

LOADSCRIPT CLASS

```
import org.keplerproject.luajava.LuaState;  
  
import org.keplerproject.luajava.LuaStateFactory;
```

```
public class LoadScript {

    LuaState luaState;

    /**
     * Constructor
     * @param fileName File name with Lua script.
     */

    LoadScript(final String fileName) {

        this.luaState = LuaStateFactory.newLuaState();

        this.luaState.openLibs();

        this.luaState.LdoFile(filename);

    }

    /**
     * Ends the use of Lua environment.
     */

    void closeScript() {

        this.luaState.close();

    }

    /**
     * Call a Lua function inside the Lua script to insert
     * data into a Java object passed as parameter
     * @param functionName Name of Lua function.
     * @param obj A Java object.
     */
}
```



```
*/  
  
void runScriptFunction(String functionName, Object obj) {  
  
    this.luaState.getGlobal(functionName);  
  
    this.luaState.pushJavaObject(obj);  
  
    this.luaState.call(1, 0);  
  
}  
  
}  
  
}
```

Utilisation des fichiers de script Lua

Pour un bref exemple de tout ce que nous avons vu jusqu'à présent, considérons un jeu avec un grand nombre de monstres différents. Cela pourrait être quelque chose comme Diablo de Blizzard ou World of Warcraft. Nos monstres auront quatre attributs différents: race, vie, attaque et défense. Supposons que le jeu comporte cent types de monstres différents, celui-ci avec différentes valeurs d'attribut. Que feriez-vous pour modéliser ces monstres? Construire une classe de monstre et une classe particulière pour chaque type de monstre? Ce serait très mauvais. Vous devriez envisager de construire la classe monstre avec notre classe de script créée précédemment.

La classe monstre recevra un nouvel attribut appelé script. Le constructeur de classe recevra la nouvelle race de monstre en tant que paramètre et chargera son script appelant la classe LoadScript. Après cela, nous appelons simplement la méthode runScriptFunction appelant «create». Ensuite, chaque race aura un fichier de script Lua qui chargera l'instance de monstre avec les attributs de race.

Jetons un œil au code:

CLASSE MONSTER

```
public class Monster extends Creature {

    /* Info */

    protected String race;

    protected int defense;

    protected int attack;

    protected int life;

    /* Script */

    private LoadScript script;

    public Monster(String race) {

        /* Loads Lua script for this race.*/

        this.script = new LoadScript(race+".lua");

        /*Call Lua create function.*/

        script.runScriptFunction("create", this);

    }

    public String getRace() {

        return race;

    }

    public int getDefense() {

        return this.defense;

    }

    public void setDefense(int defense) {

        this.defense = defense;

    }

}
```

```

    }

    public int getLife() {

    return this.life;

    }

    public void setLife(int life) {

    this.life = life;

    }

    public void setAttack(int attack) {

    this.attack = attack;

    }

    public int getAttack() {

    return this.attack;

    }

    }

```

L'analyse du code au-dessus de nous peut voir que la première ligne est la déclaration de classe monstre. Ensuite, les quatre lignes suivantes déclarent les attributs de classe relatifs aux informations sur les monstres. Ensuite, nous avons un attribut qui est notre toute nouvelle classe LoadScript. Après les déclarations d'attribut, nous pouvons trouver le constructeur de classe.

Comme nous l'avons vu ci-dessus, le constructeur reçoit une chaîne avec la race du monstre et dans sa première ligne appelle le constructeur LoadScript pour stocker dans le script d'attribut le fichier Lua qui stocke les valeurs des attributs de cette race de monstre. La ligne suivante appelle la fonction Lua create qui définira les nouveaux

monstres avec les valeurs définies dans le script. Les lignes suivantes ne sont que quelques méthodes get et sets à utiliser dans la portée Java si nécessaire.

Le code suivant montre comment doit être un script Lua:

EXEMPLE DE FICHIER DE SCRIPT

```
fonction create (monstre)

monstre: setRace ("Sample Monster")

monstre: setDefense (10)

monstre: setAttack (10)

monstre: setLife (100)

fin
```

Le fichier de script consiste uniquement en une fonction Lua qui appelle les méthodes définies dans la classe Java de monstre, définissant les valeurs pour cette race de monstre spécifique. Pour créer un nouveau monstre, le développeur a juste besoin de copier le fichier, changez son nom en nouveau nom de race de monstre et les valeurs transmises aux méthodes qu'il contient.

RÉSULTATS ET CONCLUSIONS

La technique présentée a été utilisée pour créer des scripts de monstre et de joueur pour un MMORPG 2D expérimental encore en construction. Les scripts Lua ont rendu l'insertion de nouveaux personnages (monstre ou joueurs) assez facile, car ils ont produit un fichier de script très propre et organisé et nous ont permis de copier un script déjà fait, en remplaçant simplement les valeurs par les données du nouveau personnage. Cela a permis de produire de nouveaux monstres pour le jeu aussi rapidement que possible pour générer de nouvelles combinaisons de valeurs d'attribut.

Pour un développement rapide d'une plate-forme de test, nous avons utilisé Golden T Game Engine (GTGE) qui est un logiciel gratuit et offre une bibliothèque de programmation de jeux multiplateforme avancée écrite en langage Java. GTGE est développé par Golden T Studios. Nous avons également utilisé certains graphiques de RPG Maker XP à des fins de test uniquement. RPG Maker XP est développé par Enterbrain, Inc. (figure 1).

TRAVAUX FUTURS

Nous avons l'intention de continuer à travailler au développement du MMORPG 2D en utilisant les scripts Lua. En tant que prochains objectifs du projet, nous pouvons nous détacher: remplacer les graphiques RPG Maker XP et utiliser la bibliothèque LuaJava pour lier d'autres bibliothèques Lua telles que LuaSocket (pour la communication réseau) et LuaSQL (pour l'accès à la base de données), en code Java.



RÉFÉRENCES

K. Arnold J. Gosling, 1997, The Java Programming Language. 2e édition, Addison-Wesley.

Figure 1: Capture d'écran de l'environnement de test construit à l'aide des graphiques GTGE et RPG Maker XP. Vous pouvez voir le personnage du joueur au centre, et trois monstres différents (Corbeau, TroubleMaker, TroubleMaker Leader) créés à l'aide des fichiers de script Lua.

R. Ierusalimschy, 2006, Programmation à Lua. Seconde Édition, Lua.Org.

R. Ierusalimschy et. al, 2006, Lua 5.1 Reference Manual, Lua.Org,

C. Cassino et al., 1999, LuaJava - Un outil de script pour Java, PUC-RioInf.MCC02 / 99, février.

BIOGRAPHIE

ROBERTO DE BEAUCLAIR SEIXAS travaille avec la recherche et le développement à l'Institut des mathématiques pures et appliquées - IMPA, en tant que membre du laboratoire de vision et d'infographie - Visgraf. Il a obtenu son doctorat. diplôme en informatique à l'Université pontificale catholique de Rio de Janeiro - PUC-Rio, où il travaille avec le groupe de technologie graphique informatique - TeCGraf. Ses intérêts de recherche incluent la visualisation scientifique, l'infographie, le calcul haute performance, les SIG, les systèmes de simulation et les jeux d'entraînement à la guerre. Actuellement, il est le conseiller du Warfare Games Center du Corps des Marines de la Marine brésilienne.

GUSTAVO HENRIQUE SOARES DE OLIVEIRA LYRIO travaille avec le Computer Technology Technology Group - Tecgraf. Il a obtenu son B.Sc. en génie informatique à l'Université catholique pontificale de Rio de Janeiro - Puc-Rio. Ses intérêts incluent

l'informatique graphique et les jeux d'entraînement à la guerre. Il est actuellement développeur du Warfare Games Center du Corps des Marines de la Marine brésilienne.