Proposta de trabalho

Usando Lua para Jogos

Agora que já vimos um poco sobre a programação LUA, que tal iniciar um RPG em LUA?
O código pode parecer longo mas com certeza pode ser bem proveitoso e útil para o aprendizado da linguagem.
O texto a seguir foi baseado num artigo da Internet e tem uma proposta de trabalho para nossa disciplina.
A idéia é implementar o RPG proposto usando a linguagem Lua que se encontra no código a seguir (alguns trechos ficaram ainda em inglês).
Elaborem o código e me enviem os fontes funcionando até dia 11/10 para izequiel@gmail.com.
Se for o caso usem uma abordagem similar para o contexto das histórias de seus trabalhos ok?

Começando

Para usar um Script Lua num jogo, vamos precisar de um jogo. Para este propósito teremos muito a ver sobre isso no texto a seguir.

Iremos para duas portas de um jogo NPC.

Na primeira porta teremos a FLORESTA NEGRA. Na segunda porta um calabouço. Na floresta negra percebemos um cavaleiro procurando tesouros que vão estar na segunda porta.

Esta é a ideia do jogo :)

CODIGO!

Vamos iniciar o setup do LUA.

O código vai ser este:

```
using System;
using LuaInterface;
```

```
namespace RPGLUA
{
 ///
 /// The main class for our superb RPG
 ///
 class RPG
 {
  Lua lua = new Lua();
```

So that's the Lua setup just waiting to be used for scripting goodness. Next up we need the basic game loop and basic game interface. Time to throw together a simple skeleton program!

```
class RPG
 {
  Lua lua = new Lua();
  bool quit = false;
  ///
  /// The main entry point for the application.
  ///
  [STAThread]
  static void Main(string[] args)
  {
   RPG rpg = new RPG();
   rpg.Go();
  }
  public void Go()
  {
   Console.WriteLine("Welcome to the cool RPG Game");
   Console.WriteLine("\tPress q to quit!");
   while(quit == false)
   {
    Console.Write(">");
    string ans = Console.ReadLine();
```

```
   if(ans == "q")
    quit = true;
  }
  Console.WriteLine("Goodbye");
 }
}
```

As you can probably guess - on running the above, we create an instance of the RPG class imaginatively called rpg. Then we tell it to *go*with the Go function.

At this point it enters a big loop that won't end until we quit. Each time it loops it waits for player input, in the form of a read line. Currently the only recognized input is "q" which will let us exit the game.

I believe I said something about rooms. Let's start making one.

```
     class Room
{
 private Lua scriptPower;
 private string scriptName;
 private string name;
 public Room(string name, Lua lua)
 {
  this.name = name;
  scriptName = name + ".lua";
  scriptPower = lua;
  scriptPower.DoFile(scriptName);
 }
}
```

Yah we're going for a generic room object. Basically you pass in the Lua script name (without the .lua). Then the room loads up the script file. We also keep a local copy of the name. That's it. How about a Lua script to give us an idea of what we're doing?

forest.lua

```
----
--Our Forest Room
---
--Create a forest Table--
forest = {};
--What to do on entering the forest--
function onEnter()
 print("You enter the forest");
end;
--Add some text--
forest.description = "A deep dark foreboding forest. A bit scary.";
forest.name = "Dark Forest";
--Add a function pointer--
forest.OnEnter = onEnter;
```

Type that into a file called forest.lua and save it to wherever your exe is created (same place you put the lua dlls). Now we have a carefully crafted forest let's update the room code, with two getters.

```
    public string Name
 {
  get
  {
   return(string) scriptPower[name + ".name"];
  }
 }
 public string Description
```

```
{
 get
 {
  return (string)scriptPower[name + ".description"];
 }
}
```

This is getting a bit exciting isn't it? Well I'm not one of those people who likes to write out reems and reems of code and then pull it all together at the end. I wanna start using the above getters right now! For this we need to upgrade our interface to version Alpha2009. No problem.

Here it is all at once.

```
        public void Go()
{
 Console.WriteLine("Welcome to the cool RPG Game");
 Console.WriteLine("\tPress q to quit!\n\r");
 while(quit == false)
 {

  Explore(currentRoom);
  Console.Write(">");
  string ans = Console.ReadLine();
  ClearScreen();
  if(ans == "q")
   quit = true;
 }
 Console.WriteLine("Goodbye");
}
public void Explore(Room r)
{
 Console.WriteLine("You are in: " + r.Name);
 Console.WriteLine(r.Description);
```

```
 }
 //Write 50 blank lines should clear the terminal
 public void ClearScreen()
 {
  for(int i = 0; i < 50; i++)
   Console.WriteLine();
 }
```

Pretty self explanatory. My clear screen is a bit crap but what are you going to do?

Run it and see how cool it is! Wooo! We've setup the dark forest as the first room. Next we're going to add an NPC in the middle of the forest : - the knight. No doubt while reading this you are thinking "Oh you could do this too or write it this way instead". Yeah there are many ways to do things and a lot of choice between what you want in the script and what you want in the C# code. Generally knowing what you're creating will make these choices easier. (Do you want the game to be extended or are you just using scripts for easier content management)

Lets create the NPC class it will be based very closely on our room class. In fact it is going to be the room class but renamed to NPC. Object Orientaness suggests we should have a parent class possibly a scriptable object and a lookable interface? ILookable and Scriptable. I'm not going to put this in though! :o I want to keep things pretty simple and direct.

```
class NPC
 {
  private Lua scriptPower;
  private string scriptName;
  private string name;
  public NPC(string name, Lua lua)
  {
   this.name = name;
```

```
 scriptName = name + ".lua";
 scriptPower = lua;
 scriptPower.DoFile(scriptName);
}

public string Name
{
 get
 {
  return (string)scriptPower[name + ".name"];
 }
}
public string Description
{
 get
 {
  return (string)scriptPower[name + ".description"];
 }
}
}
```

All quite familiar. Time to stir things up a bit. We're going to be able to add occupants to rooms and we'll add a way to interact with NPCs in said rooms. This calls for funktions.

```
   public void DescribeOccupants()
{
 foreach(NPC n in npcList)
 {
  Console.WriteLine("There is a " + n.Name + " here.");
 }
}
public void AddNPC(NPC npc)
{
```

```
 npcList.Add(npc);
}
```

Looks like we also need an array list called npcList. Therefore remember to include `using System.Collections;`. Add the npcList to the top of the room class, like so

```
private ArrayList npcList = new ArrayList();
```

We can now add NPCs to rooms! Rather than hard code this. Lets use another script that will have the power to add NPCs to rooms among other things. We'll call this script *setup*. The script will need one or two helper functions so let's write them and hook them into lua. Here's the code:

```
class RPG
{
Lua lua = new Lua();
bool quit = false;
Room currentRoom;

public Room CreateRoom(string roomName)
{
return new Room(roomName, lua);
}

public NPC CreateNPC(string npcName)
{
return new NPC(npcName, lua);
}

public void SetStartRoom(Room r)
{
currentRoom = r;
}
```

```
public void AddNPCToRoom(Room  r, NPC n)
{
r.AddNPC(n);
}

public RPG()
{
lua.RegisterFunction("CreateRoom", this,this.GetType().GetMethod("CreateRoom"));
lua.RegisterFunction("CreateNPC", this, this.GetType().GetMethod("CreateNPC"));
lua.RegisterFunction("AddNPCToRoom", this,this.GetType().GetMethod("AddNPCToRoom"));
lua.RegisterFunction("SetStartRoom", this,this.GetType().GetMethod("SetStartRoom"));
lua.DoFile("setup.lua");
```

Okay let's see the script file. It uses the simple functions, we wrote, that allow easy creation of NPCs and rooms and ways to add NPCs to rooms.

Setup.lua

```
---
--- Setup Script
---
start = CreateRoom("forest");
SetStartRoom(start);
AddNPCToRoom(start, CreateNPC("knight"));
```

After this I bet you'd quite like to see the knight.lua file too yeah? It's basically the same as room.

knight.lua

```
---
--Knight NPC
---
--Create a knight Table--
knight = {};
knight.description = "A tall knight wrapped in bulky armour. He seems to be
shaking.";
knight.name = "A Knight";
function knight.OnTalk()
end;
```

Our code creates a forest room and puts a knight in the forest. It also determines where we the player start (in the forest). Let's get the functionality of this room and npc quite complete before we add the next room. First we'll want to see (in the game) the knight. This isn't a problem.

```
public void Explore(Room r)
 {
  Console.WriteLine("You are in: " + r.Name);
  Console.WriteLine(r.Description);
  r.DescribeOccupants()
 }
```

Pretty easy. But now we need to interact with the the npc. I'm going to give the NPCs an interact function and a interact key. In the NPC class do the following:

```
public string interactKey;
  public NPC(string name, Lua lua)
```

```
{
 this.name = name;
 this.interactKey = name[0].ToString();
```

What's the interact key? Well it's so the program can say "Press K to interact with the knight". Then we'll have an interact function where the interactions actually take place. All NPCs will have Talk and Look. Basic but that's all we'll need. Currently I've added the code for what happens when you look.

```
public void Interact()
{
 string answer = "";
 while(answer != "x")
 {
  Console.WriteLine("You are interacting with " + Name);
  Console.WriteLine("\tx to exit");
  Console.WriteLine("\tt to talk");
  Console.WriteLine("\tl to look");
  Console.Write(">");
  answer = Console.ReadLine();
  if(answer == "l")
  {
   Console.WriteLine(this.Description);
  }
 }
}
```

Pretty standard stuff. Now we need to have a way to interact with these NPCs when we, the plater, are in the same room. We'll make two functions. Display Interactions and CheckInteractionKeys. I'm aware that there could be a problem where you had say a knight and king both K's :o at this point the computer could ask you to clarify. This is more of an example though, I

assume most people will be going the graphical route and interaction will be a bit more straightforward.

So in **the room class** add the following two functions.

```
public void DisplayInteractions()
 {
  foreach(NPC n in npcList)
  {
   Console.WriteLine("Press " + n.interactKey +
    " to interact with " + n.Name);
  }
 }
 public void CheckInteractions(string keypress)
 {
  foreach(NPC n in npcList)
  {
   if(n.interactKey == keypress)
   {
    n.Interact();
    return;
   }
  }
 }
```

Now we have to link these functions into the game loop. After which, we should be able to interact with NPCs! Yay for us!

DisplayInteractions looks at all the NPCs in the room. It then gets the interaction key from each NPC and says if you want to interact with npc X you must press his interaction key Y.

The second function takes the key that the user has pressed and asks each NPC in the room - is this your key? If the npc say's yes that's my key then his or

her interaction is run and they engage the player. After that's finished the user is dropped back to the room he/she was in.

That said let's add some calls to the functions.

```
public void Go()
 {
  Console.WriteLine("Welcome to the cool RPG Game");
  Console.WriteLine("\tPress q to quit!\n\r");
  while(quit == false)
  {
   Explore(currentRoom);
   Console.Write(">");
   string ans = Console.ReadLine();
   ClearScreen();
   currentRoom.CheckInteractions(ans);
   if(ans == "q")
    quit = true;
  }
  Console.WriteLine("Goodbye");
 }
 public void Explore(Room r)
 {
  Console.WriteLine("You are in: " + r.Name);
  Console.WriteLine(r.Description);
  r.DescribeOccupants();
  Console.WriteLine();
  r.DisplayInteractions();
 }
```

On exploring a room we tell all the npcs to tell the user what key needs to be pressed to interact with them. On the user pressing a key we call the check each NPC.

No problem, give it a whirl and interact with the knight.

Looking is good but there's something else I desire in my knights and that's talking! Let's make the knight talk.

Very important - Make sure to add the following two lines of code (if you're ever getting Metatable errors see if you have added these two lines).

```
public RPG()
 {
  lua.OpenTableLib();
  lua.OpenBaseLib();
```

With that done we can move to the setup file. We want the knight script to have access to the knight object. This will be done through the setup file.

Setup.lua

```
---
--- Setup Script
---
load_assembly("RPGLUA");
NPC = import_type("RPGLUA.NPC");
start = CreateRoom("forest");
SetStartRoom(start);
npc = CreateNPC("knight");
knight.this = npc;
AddNPCToRoom(start, npc);
```

We import the NPC class so we can use in our Lua scripts. When we create a NPC called knight we know it creates a table called knight (which we're using as a namespace). So into knight we add a entry called this which refers to the C# npc object. From this we can access everything to do with knight (we don't get rights to protected or private stuff though :( )

Next we'll alter the C# NPC class in two places. This will set us up for being able to talk. First the OnTalk function.

```
public void OnTalk()
{
 scriptPower.DoString("knight:OnTalk()");
}
```

Next the call to the OnTalk function. This call is made when t is pressed.

```
        public void Interact()
{
 string answer = "";
 while(answer != "x")
 {
  Console.WriteLine("You are interacting with " + Name);
  Console.WriteLine("\tx to exit");
  Console.WriteLine("\tt to talk");
  Console.WriteLine("\tl to look");
  Console.Write(">");
  answer = Console.ReadLine();
  if(answer == "l")
  {
   Console.WriteLine(this.Description);
  }
  else if(answer == "t")
  {
   this.OnTalk();
```

```
  }
 }
}
```

These two functions mean when we press T we're going to called the ontalk function in the knight script. Before we go to the on knight script though we want to add one last function to the NPC class.

```
public void Say(string say)
 {
  Console.WriteLine(Name + " says \"" + say +"\"");
 }
```

Our script will use the Say function to make our NPC talk. Here's the script in action.

knight.lua

```
----
--Knight NPC
---
--load_assembly("RPGLUA");
--NPC = import_type("RPGLUA.NPC");
--Create a knight Table--
if(knight == nil) then
 knight = {};
end
knight.description = "A tall knight wrapped in bulky armour. He seems to be shaking.";
```

```
knight.name = "A Knight";
function knight.OnTalk()
 knight.this:Say("Ah I'm so scared! I have to find the kings treasure but I'm
paralyzed with fear! Please help me!");
end;
```

Okay finally we're getting to some kind of story! Scripting seems to be working fine and I'm sure you're getting the jist of it. We're going to add rooms and pasages ways then allow the quest to be finished. A passage way is quite simple here's the code.

```
class PassageWay
 {
  public string description;
  public Room destination;
  public string key = "p";
  public PassageWay(string describe, Room leadsto)
  {
   destination =  leadsto;
   description = describe;
  }
 }
```

Righty-o that's a passage way done. Of course really we need a room for a passage way to go to! And we need rooms to "own" passageways. Okay first let's give rooms passageways much in the same way they were given NPCs. Here's all the code at once woo:

```
 public void AddPassageWay(PassageWay pw)
  {
   wayList.Add(pw);
  }
  public void DescribeOccupants()
```

```
{
 foreach(PassageWay pw in wayList)
 {
  Console.WriteLine(pw.description);
 }
 foreach(NPC n in npcList)
 {
  Console.WriteLine("There is a " + n.Name + " here.");
 }

}
public void AddNPC(NPC npc)
{
 npcList.Add(npc);
}
public void DisplayInteractions()
{
 foreach(NPC n in npcList)
 {
  Console.WriteLine("Press " + n.interactKey +
   " to interact with " + n.Name);
 }
 foreach(PassageWay pw in wayList)
 {
  Console.WriteLine("Press " + pw.key + " to use the "
   + pw.description);
 }
}
public void CheckInteractions(ref Room currentRoom, string keypress)
{
 foreach(NPC n in npcList)
 {
  if(n.interactKey == keypress)
  {
   n.Interact();
```

```
    return;
   }
  }
  foreach(PassageWay pw in wayList)
  {
   if(pw.key == keypress)
   {
    currentRoom = pw.destination;
   }
  }
 }
```

Prertty straight forward. The important thing to note is that ref. Yes we need to change rooms on a keypress (to simulate the player moving from one room to another) therefore we need to have access to the currentRoom. Hence the new argument. All this code requires a new arrayList too (to store all the passageways a room might have).

```
 private ArrayList wayList = new ArrayList();
```

Now we must ammend the code so we pass a current room reference in.

This ammendmant all goes down In the Go function.

```
while(quit == false)
  {
   Explore(currentRoom);
   Console.Write(">");
   string ans = Console.ReadLine();
   ClearScreen();
   currentRoom.CheckInteractions(ref currentRoom, ans);
   if(ans == "q")
```

Now interactions can change the room the player is in - all very cool. Let's give the scripts a few more functions to play with.

```
public PassageWay CreatePassageWay(string describe, Room r)
 {
  return new PassageWay(describe, r);
 }
```

The above goes in the RPG class and we register it like so:

```
public RPG()
 {
  lua.OpenTableLib();
  lua.OpenBaseLib();
  lua.RegisterFunction("CreateRoom", this, this.GetType().GetMethod("CreateRoom"));
  lua.RegisterFunction("CreatePassageWay", this,
this.GetType().GetMethod("CreatePassageWay"));

...
```

There it is all registered. Ready to see the new setup file? It's a beast!

setup.lua

```
---
--- Setup Script
---
load_assembly("RPGLUA");
```

```
NPC = import_type("RPGLUA.NPC");
Room = import_type("RPGLUA.Room");
Way = import_type("RPGLUA.PassageWay");
--Create the forest
start = CreateRoom("forest");
SetStartRoom(start);
--Create the Knight
npc = CreateNPC("knight");
knight.this = npc;
AddNPCToRoom(start, npc);
--Create the dungeon
room2 = CreateRoom("dungeon");
--Create the passage from the forest to the dungeon
underground = CreatePassageWay("Steps surrounded by leaf litter", room2);
start:AddPassageWay(underground);
```

Okay and here's the dungeon file. I wrote it at the start but here's a good time to introduce it!

dungeon.lua

```
----
--Our Dungeon Room
---
--Create a forest Table--
dungeon = {};
function onEnter()
 print("You enter the dungeon.");
end;
```

```
dungeon.description = "A small room with irregular stone walls. It's damp and dark. Also there are piles of glittering treasure.";
dungeon.name = "Dank Dungeon";
dungeon.OnEnter = onEnter;
```

If you actually play the game you'll find you can see a passage and use it and end up in a new room (trapped by the way!). I was suprised at this as it worked immediately without me having to fix bugs :D We are practically done! We need to improve the setup script so all rooms-scripts have a this method. While we're at we'll add a passage way back.

(I was going to have the treasure in a box you had to open it with a key from the knight but I want to keep this short. In fact right at the start I was going to have four rooms, a knight AND a witch - crazy!)

setup.lua

```
---
--- Setup Script
---
load_assembly("RPGLUA");
NPC = import_type("RPGLUA.NPC");
Room = import_type("RPGLUA.Room");
Way = import_type("RPGLUA.PassageWay");
--Create the forest
start = CreateRoom("forest");
forest.this = start;
SetStartRoom(start);
--Create the Knight
```

```
npc = CreateNPC("knight");
knight.this = npc;
AddNPCToRoom(start, npc);
--Create the dungeon
room2 = CreateRoom("dungeon");
dungeon.this = room2;
--Create the passage from the forest to the dungeon
underground = CreatePassageWay("Steps surrounded by leaf litter", room2);
start:AddPassageWay(underground);
--Create the passage from the dungeon to the forest
overground = CreatePassageWay("a stone staircase", start);
room2:AddPassageWay(overground);
```

To finish the game we need a have-you-seen-the-treasure-flag and we need to set up the OnEnter function for rooms. Because I'm amazing I know the potential problem of seeing the treasure before you get the quest but do you know what? I don't care, I guess this is how the people who made Lionheart felt.

The flag isn't a problem:

```
        public RPG()
 {
  lua["seenTreasure"] = 0;
```

Neithers the call to the OnEnter command.

```
        foreach(PassageWay pw in wayList)
  {
   if(pw.key == keypress)
   {
    currentRoom = pw.destination;
    currentRoom.OnEnter();
   }
```

```
 }
```

We're jumping around a bit here but first next let's go to knight.lua.

```
function knight.OnTalk()
 if(seenTreasure == 0) then
  knight.this:Say("Ah I'm so scared! I have to find the kings treasure but I'm
paralyzed "
   .. "with fear! Please help me!");
 else
  knight.this:Say("What you found my treasure that's great. I'm so happy. I'll get it
in a "
   .. "bit! By the way game over");
 end;
end;
```

The two dots are concatinate by the by. As you can see above we vary what the knight says depending if you've seen the treasure or not. All we have left to do is set the flag when entering the dungeon-room and we've written a cool scripted game. It's possibly quite extensible too I haven't checked. I know you can add an arbitary number of rooms.

In the Room class add

```
        public void OnEnter()
 {
  scriptPower.DoString(name +":OnEnter()");
 }
```

Wooooooooooo finally add the following to the dungeon.lua file:

```
function onEnter()
 seenTreasure = 1;
```

end;

And that's it game over, a fully scripted RPGish style game we can all enjoy. If you're still raring for more I suggest you refactor the code so its more OO friendly. Then maybe try and a more complicated quest. You might want to have things like:
OnLeave
OnPush
OnGive
OnStealFrom
OnDeath
etc etc

Also as I went along a lot of the helper functions registered with Lua became redundant -- they can be cleared out.

Go wild. Try a graphical front end too! Patch it into your latest game! Let me know how it goes.

Baseado em: http://www.godpatterns.com/2005/07/using-lua-scripting-for-games.html