

**www.Mcours.com**

Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)

## M2-GamaGora

intégration d'un langage de script: Lua

J.C. lehl

December 11, 2009

## résumé des épisodes précédents

boucle principale :

- ▶ déplacements,
- ▶ collisions,
- ▶ réaction aux collisions,
- ▶ comportements,
- ▶ interactions : réaction à des évènements.

## résumé des épisodes précédents

pour chaque état :

en fonction de l'évènement reçu : réagir,

- ▶ envoyer de nouveaux évènements,
- ▶ attendre un évènement particulier, etc.
- ▶ (gestion des transitions vers les autres états)

comportement, évènements :

- ▶ assez pénible à écrire,
- ▶ encore plus pénible à modifier.

programmeur nécessaire pour construire l'interaction du "monde"  
et du joueur.

## scripter le comportement du "monde"

### pourquoi ?

- ▶ simplifier l'écriture du comportement de chaque classe d'entité,
- ▶ aide pour le programmeur,
- ▶ mais surtout :  
exposer simplement la "logique" du comportement,
- ▶ et cacher la programmation "bas niveau" associée,

permettre aux LD de décrire les interactions / comportements.

# intégrer un langage de script

## pourquoi ?

- ▶ séparer le "code" et les "données",
- ▶ le comportement d'une entité est une donnée pour le jeu,
- ▶ ne pas recompiler l'application à chaque test,
- ▶ change le cycle de mise au point du jeu.

## classique :

coder, compiler, exécuter, reproduire les conditions permettant de tester la modification, tester ...

## data driven (comportement == donnée) :

exécuter, charger le script, tester, modifier le script, continuer à tester ...

## intégrer un langage de script "visuel"

autre solution pour les LD :

utiliser une représentation visuelle plutôt que du script.

exemple : kismet du udk3.

# quel langage ?

- ▶ python,
- ▶ lua,
- ▶ angelscript,
- ▶ javascript,
- ▶ ...

# présentation Lua

## pourquoi Lua ?

- ▶ l'interpréteur est une librairie C,
- ▶ facile à inclure dans une application,
- ▶ rapide, + machine virtuelle JIT (x86),
- ▶ gestion mémoire automatique (garbage collector),
- ▶ multi-threadé (coopératif) + co-routines,
- ▶ interactions entre code C/C++ et code lua interprété :
- ▶ variables globales,
- ▶ appel de fonctions C/C++ depuis lua,
- ▶ appel de fonctions lua depuis C/C++.



## présentation de Lua

```
- define a function
function Erupt()
    Camera.SetMode(ORBIT);
    Camera.SetOrbitRate(PIOVER2);
    entity= Entity.find("Volcano");
    Camera.SetTargetEntity(entity);
end

- define a main
while(1)
    Script.WaitSeconds(10);
    Erupt()
end
```

## intégration Lua

- ▶ Lua et son "hôte" communiquent à travers une pile.
- ▶ `lua_State` représente le contexte d'exécution de la fonction (pile, etc.).

les paramètres sont empilés dans l'ordre, 1, 2, 3, etc.

- ▶ lua est typé dynamiquement, une variable peut changer de type !
- ▶ donc, il faut vérifier le type des paramètres empilés.

## intégration Lua

```
script lua :  
Camera.SetTargetPos(100.0, 40.0, 230.0);  
  
code C :  
int LuaSetTargetPos( lua_State *state )  
{  
    float x, y, z;  
  
    x= (float) lua_tonumber(state, 1);  
    y= (float) lua_tonumber(state, 2);  
    z= (float) lua_tonumber(state, 3);  
  
    CameraSetTargetPos(x, y, z);  
    return 0;  
}
```

## intégration Lua

```
int LuaSetTargetPos( lua_State *state )
{
    float x, y, z;

    if(lua_isnumber(state, 1)
    && lua_isnumber(state, 2)
    && lua_isnumber(state, 3))
    {
        x= (float) lua_tonumber(state, 1);
        y= (float) lua_tonumber(state, 2);
        z= (float) lua_tonumber(state, 3);

        CameraSetTargetPos(x, y, z);
    }
    return 0;
}
```

## intégration Lua

une fonction C peut aussi renvoyer une valeur à Lua :

```
script lua :  
if Game.GetPlayerScore() > 9 then  
    TriggerEndOfLevel();  
end
```

```
code C :  
int LuaGetPlayerScore( lua_State *state )  
{  
    lua_pushnumber(state, game->playerScore);  
    return 1;  
}
```

## intégration Lua

déclarer des fonctions C/C++ à l'interpréteur Lua :

```
const luaL_reg CameraLib [] =  
{  
    { "SetMode", LuaSetMode },  
    { "GetMode", LuaGetMode },  
    { "SetTargetPos", LuaSetTargetPos },  
    { "SetTargetEntity", LuaSetTargetEntity },  
    ...  
  
    { NULL, NULL }  
};  
  
luaL_openlib(state, "Camera", CameraLib, 0);
```

# intégration Lua

exécuter du code Lua depuis du C/C++ :  
créer un contexte d'exécution, lua\_State.

```
lua_State *state= lua_open();  
lua_dofile(state, "script.lua");  
lua_dostring(state, "a=_1;_b=_4;");  
lua_close(state);
```

## intégration Lua

charger un script, exécuter une fonction :

charger le script, empiler les paramètres, appeler la fonction.

```
lua_State *state= lua_open();  
lua_loadfile(state, "script.lua");  
lua_getglobal(state, "function");  
lua_pushnumber(state, 1.0)  
lua_call(state, 1, 0);  
lua_close(state);
```