# 스크립트

MM4220 게임서버 프로그래밍

정내훈

# 내용

- 스크립트
- LUA 프로그래밍

# 스크립트

- 정의
  - 사용하기 편한 언어 => 생산성이 높다.
- 특징
  - 직관적인 문법
  - 관용적인 문법
  - 성능보다는 표현력에 중심
  - 인터프리팅 (<-> 컴파일러)
- 게임에서의 활용
  - 데이터 정의
  - NPC AI
  - Non-programmer

# 스크립트

- 장점
  - 생산성
    - 빠른 개발 속도
    - Live edit, 서버 재 컴파일 불필요
- 단점
  - 성능
    - 느린 실행 속도
- 종류
  - 기존 언어
    - Visual Basic, C#, Java, Python, LUA, XML, javascript
  - Custom
    - LPC, UnrealScript

# 스크립트

- 요구
  - 배우기 쉬울 것
    - 쉬운 문법
  - Oop기능 제공
    - Object -> Living -> Monster -> Orc -> 네루바 오크
  - Multi-thread기능 제공 (옵션)
    - Multi-thread환경에서의 동작 보장
      - Re-enterant
    - 쓰레드별로 별도의 인터프리터 객체를 실행할 수도 있음.
    - 멀티쓰레드 프로그래밍이 필요한 경우
      - 개별 컨텐츠 구현이 아니라 시스템을 구현할 경우
      - 예) EVE-online

# 스크립트 (2019)

- 스크립트 언어의 특징
  - LUA : 간단, 성능이 뛰어남, 가장 많이 사용
  - Python : 생산성이 높다. Multithread성능에 문제 있음
  - Java : 높은 인지도, garbage collection으로 인한 성능 저하
  - C# : 성능, multithread지원
  - XML : 언어가 아님

# LUA

- LUA의 기원 및 설계 목적
  - 포르투갈어의 '달'에 해당하는 말.
  - 가벼운 명령형/절차식 언어로, 확장 가능한 문법을 가진 스크립팅 언어를 주 목적으로 설계되었다.
- LUA의 특징
  - 간단한 자료구조 : boolean, number, string, table
    - Array? Structure?
- LUA의 장점
  - 핵심 모듈의 크기가 120KB 이하.(cf. Python 860KB, Perl 1.1MB)
  - 게임 개발에 자주 사용되는 Python에 비해 빠른 실행속도를 가짐
  - 공짜!!(Open Source)

# LUA

- LUA

```
print "Hello, World!”
```

```
function factorial(n)
  if n == 0 then
    return 1
  end
  return n * factorial(n - 1)      -- A comment in Lua starts with a double-hyphen
end                                -- and runs to the end of the line

print(factorial(5))

print([[multiple
     lines]])                      --[[It's possible to write in multiple lines if
                                     it's used with following double square brackets]]
```

# LUA

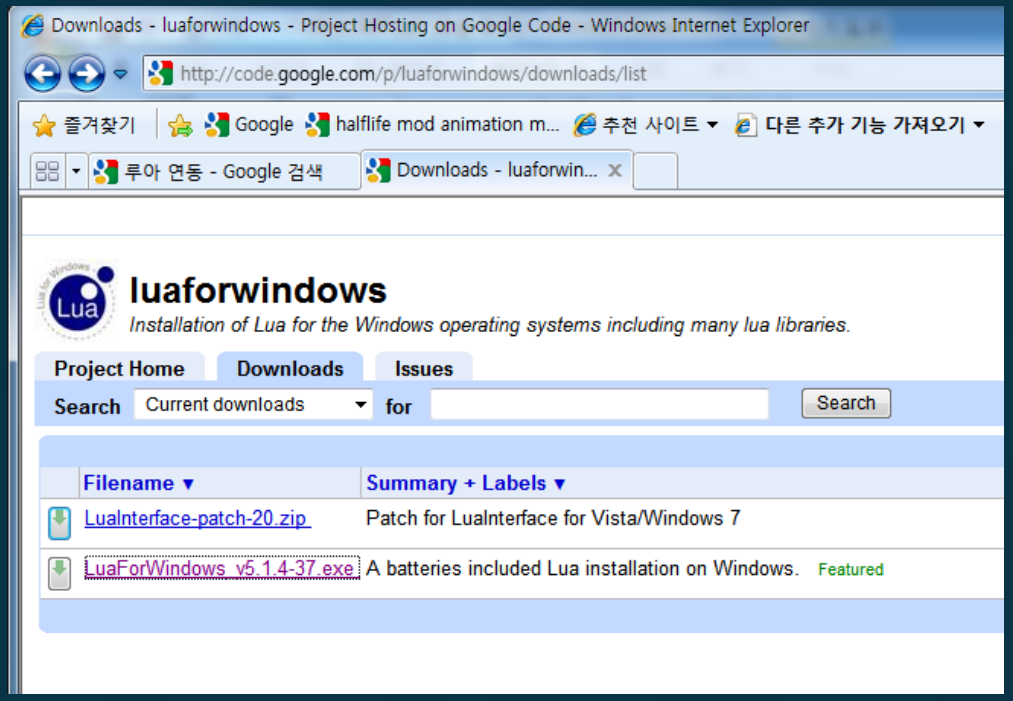
- LUA

```
-- Creates a new table, with one associated entry.
-- The string x mapping to the number 10.
a_table = {x = 10}
-- Prints the value associated with the string key,
-- in this case 10.
print(a_table["x"])
b_table = a_table
a_table["x"] = 20    -- The value in the table has been changed to 20.
print(a_table["x"])  -- Prints 20.
-- Prints 20, because a_table and b_table both refer to the same table.
print(b_table["x"])
```

```
point = { x = 10, y = 20 }    -- Create new table
print(point["x"])             -- Prints 10
print(point.x)                -- Has exactly the same meaning as
line above
```

# 스크립트

- LUA 실행
  - 공식 홈페이지 : http://www.lua.org/
    - 윈도우 바이너리를 다운 받아서 설치한다.



# 스크립트

- LUA : Visual C++와의 연동
  - include & library
    - http://www.lua.org => Download => binaries => get a binary => Windows => LuaForge => Lua5.3.4 – Release1 => Windows Libraries => Static => lua-5.3.4_Win32_vc15_lib.zip
  - 모든 include 화일을 프로젝트에 추가할 것
  - lua53.lib를 라이브러리에 추가할 것

# LUA

- LUA 실습
  - Visual Studio 프로젝트 만들기
  - Visual Studio의 Include 와 library 세팅
  - lua53.lib 필요
  - 간단한 루아 프로그램 실행

```
#include <stdio.h>
#include <string.h>
extern "C" {
#include "lua.h"
#include "lauxlib.h"
#include "lualib.h"
}

int main(void)
{
   char buff[256];
   int error;
   lua_State *L = luaL_newstate();   //루아를연다.
   luaL_openlibs(L);    //루아표준라이브러리를연다.


   while(fgets(buff, sizeof(buff), stdin) != NULL) {
     error =luaL_loadbuffer(L, buff, strlen(buff), "line")
          ||  lua_pcall(L, 0, 0, 0);
     if(error) {
       fprintf(stderr, "%s\n", lua_tostring(L, -1));
       lua_pop(L, 1);       //스택으로부터오류메시지를뽑아낸다.
     }
    }
   lua_close(L);
   return 0;
}
```

# 스크립트

- LUA 실습
  - 아래 스크립트를 C++에서 읽고 실행 후 rows, cols값을 알아내기

```
function plustwo (x)
local a;
a = 2
return x+a;
end
--here we see a simple function but it could be very complex,
--even check for user input
rows = 6; -- comments in a config file can be handy
cols = plustwo(rows);--now we can have functions in our config files.
```

Ex1. lua

# LUA

- LUA 실습
  - 소스 디렉토리에 ex1.lua 생성

```
#include <stdio.h>
 extern "C"
{
#include "lua.h"
#include "lauxlib.h"
#include "lualib.h"
}

int main(void)
{
            int rows, cols;

            lua_State *L = luaL_newstate();  //루아를연다.
            luaL_openlibs(L);   //루아표준라이브러리를연다.
            luaL_loadfile(L,"ex1.lua");
            lua_pcall(L,0,0,0);

            lua_getglobal(L,"rows");
            lua_getglobal(L,"cols");
            rows = (int) lua_tonumber(L,-2);
            cols = (int) lua_tonumber(L,-1);

            printf("Rows %d, Cols %d\n", rows, cols);

            lua_pop(L,2);
            lua_close(L);
            return 0;
}
```

# LUA

- LUA 실습
  - LUA는 Stack Machine이다.
    - C++와 LUA 프로그램간의 자료 교환은 Stack을 사용한다.
      - LUA 함수 호출시 매개 변수 Push
        - 예 ) lua_pushnumber(L, 123);
      - Stack에 저장된 값 읽기
        - 예) (int) lua_tonumber(L,-2);
      - Stack에 글로벌 변수 값 저장하기
        - 예) lua_getglobal(L,"rows");

# LUA

- LUA 실습
  - C에서 LUA함수 호출

```
void lua_pcall (     lua_State *L,
                     int nargs,
                     int nresults,
                     int msgh);
```

  - nargs는 파라미터의 개수
  - nresults는 리턴값의 개수
  - msgh는 0 (또는, 에러 메시지 핸들러)
  - 스택에 함수, 파라미터1, 파라미터2... 파라미터n을 넣어놓아야 한다.
  - 실행이 끝나면 스택에는 리턴값만 남고 다 pop된다..

# LUA

- LUA 실습
  - 원하는 함수 실행

```
…
int main(void)
{
        int result;
        int error;

        lua_State *L = luaL_newstate();
        luaL_openlibs(L);

        error = luaL_loadfile(L,"ex1.lua");
        lua_pcall(L,0,0,0);
        lua_getglobal(L,"plustwo");
        lua_pushnumber(L, 5);
        lua_pcall(L,1,1,0);
        result = (int) lua_tonumber(L,-1);

        printf("result %d\n", result);

        lua_pop(L,1);
        lua_close(L);
        return 0;
}
```

# LUA

- LUA 실습
  - 에러 검출

```
void error (lua_State *L, const char *fmt, ...) {
      va_list argp;
      va_start(argp, fmt);
      vfprintf(stderr, fmt, argp);
      va_end(argp);
      lua_close(L);
      exit(EXIT_FAILURE);
    }
```

```
if (0 != lua_pcall(L, 0, 0, 0))
  error(L, "error running function `XXX': %s\n",
                   lua_tostring(L, -1));
```

# LUA

- LUA 실습
  - LUA에서 C 함수 호출
    - // make my_function() available to Lua programs
    - lua_register(L, "my_function", my_function);
  - 파라미터와 리턴 값은 스택을 사용
- 실습
  - 두개의 값을 더하는 C 함수를 사용한 후 이를 LUA에서 호출하도록 프로그램 하라.
  - C에서 LUA 함수를 호출하고 LUA가 다시 C를 호출하는 형태

# LUA

- LUA 실습

```
function addnum_lua(a, b)
return c_addnum(a,b);
end
```

```
int addnum_c(lua_State *L)
 {
          int a = (int)lua_tonumber(L, -2);
          int b = (int)lua_tonumber(L, -1);
          int result = a + b;
          lua_pop(L,3);
          lua_pushnumber(L, result);
          return 1;
 }
```

```
lua_register(L, "c_addnum", addnum_c);

lua_getglobal(L,"addnum_lua");
lua_pushnumber(L, 100);
lua_pushnumber(L, 200);
lua_pcall(L, 2, 1, 0);
result = (int) lua_tonumber(L, -1);
printf("Result of addnum %d\n", result);
lua_pop(L,1);
```

# TODO

- LUA를 사용한 NPC AI의 구현