스크립트

MM4220 게임서버 프로그래밍

정내훈

내용

- 스크립트
- LUA 프로그래밍
- LUA 연동

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 (2023-월화)

- 게임에 사용 시 장점
  - 생산성
    - 빠른 개발 속도, **서버 프로그래머의 해방**
    - Live edit, 서버 재 컴파일 불필요
- 게임에 사용 시 단점
  - 성능 : 느린 실행 속도, 가비지 컬렉션 랙
- 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
      - Reentrant : 하나의 객체를 여러 쓰레드에서 동시에 메소드 호출해도 문제없이 실행
    - 쓰레드별로 별도의 인터프리터 객체를 실행할 수도 있음.
    - 멀티쓰레드 프로그래밍이 필요한 경우
      - 개별 객체 구현이 아니라 시스템을 구현할 경우
      - 예) EVE-online

스크립트

- 스크립트 언어의 특징
  - LUA : 간단, 성능이 뛰어남, 게임에서 가장 많이 사용
  - Python : 생산성이 높다. 덩치가 큰 부담이 있다. Multithread성능에 문제 있음
  - Java, javascript: 높은 인지도, garbage collection으로 인한 성능 저하
  - C# : 성능, C에서 호출 오버헤드가 있다. multithread지원
  - XML : 언어가 아님 => 최근에는 JSON이 대신 사용됨

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

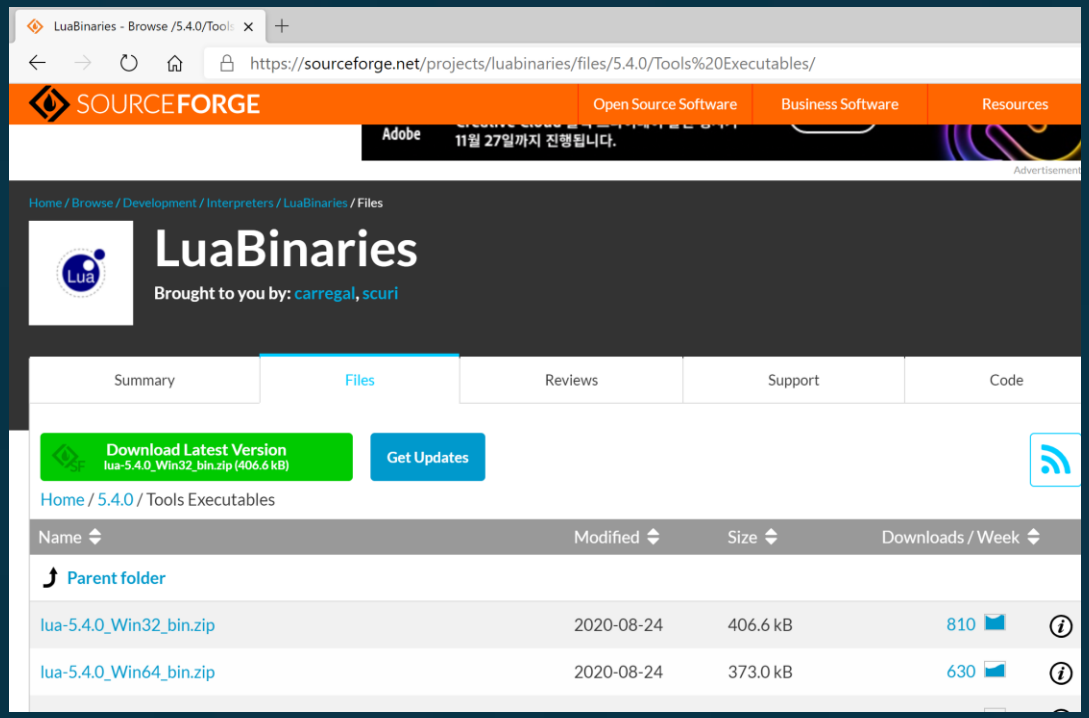
- LUA

```
-- 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/
    - Download->Building->binary->
    - Download->Building->Live demo



스크립트

- LUA : Visual C++와의 연동
  - include & library
    - http://www.lua.org => Download => binaries => download => lua-5.4.0 – Release 1 => Windows libraries => static => lua-5.4.0_Win64_vc16_lib.zip
  - 모든 include 화일을 프로젝트에 추가할 것
  - lua54.lib를 라이브러리에 추가할 것

- 참조 자료 :
  - https://www.slideshare.net/sunhyouplee/c-lua-script

LUA (2023 화목)

- LUA 실습
  - Visual Studio 프로젝트 만들기
  - Visual Studio의 Include 와 library 세팅
  - lua54.lib 필요
  - 간단한 루아 프로그램 실행

```
#include <iostream>
#include  “lua.hpp”

#pragma comment (lib, "lua54.lib")
using namespace std;

int main()
{
   const char *buff = "print ₩"Hello from Lua.₩"₩n";

   lua_State* L = luaL_newstate();
   luaL_openlibs(L);
   luaL_loadbuffer(L, buff, strlen(buff), "line");
   int error = lua_pcall(L, 0, 0, 0);
   if (error) {
      cout << "Error:" << lua_tostring(L, -1);
      lua_pop(L, 1);
   }
   lua_close(L);
}
```

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
pos_x = 6; -- comments in a config file can be handy
pos_y = plustwo(pos_x);--now we can have functions in our config files.
```

dragon_move. 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
  - LUA의 가상머신은 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

```
int 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..
  - 에러없이 종료되면 lua_pcall은 0을 리턴한다.

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

```
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);
```

LUA

- 참조
  - https://www.slideshare.net/sunhyouplee/c-lua-script

NPC SCRIPT 연동 (2023월화)

- NPC의 동작을 스크립트 언어로 정의
  - 설정 : 몬스터의 기본값 설정
    - HP, Level, 기본 반응, 기본 대사, 좌표, 종족…
  - 반응 : 어떠한 이벤트가 일어 났을 때 그것에 대한 반응
    - 플레이어 출현, 공격 받음
  - 자체 동작
    - 주위 이동, Idle message, 토끼 잡는 늑대

NPC SCRIPT 구현

- NPC의 동작을 스크립트 언어로 정의
  - 기본적으로 FSM이다.
    - event driven구현
    - Event의 출처
      - 시작, 종료, packet receive
      - timer, 다른 객체로 부터의 호출
  - VM(Virtual Machine에 상태 저장)
    - 스크립트로 관리되는 객체 정보는 VM에 저장
      - 예) LUA의 lua_State
    - 스크립트언어 인터프리터는 VM의 내용을 업데이트

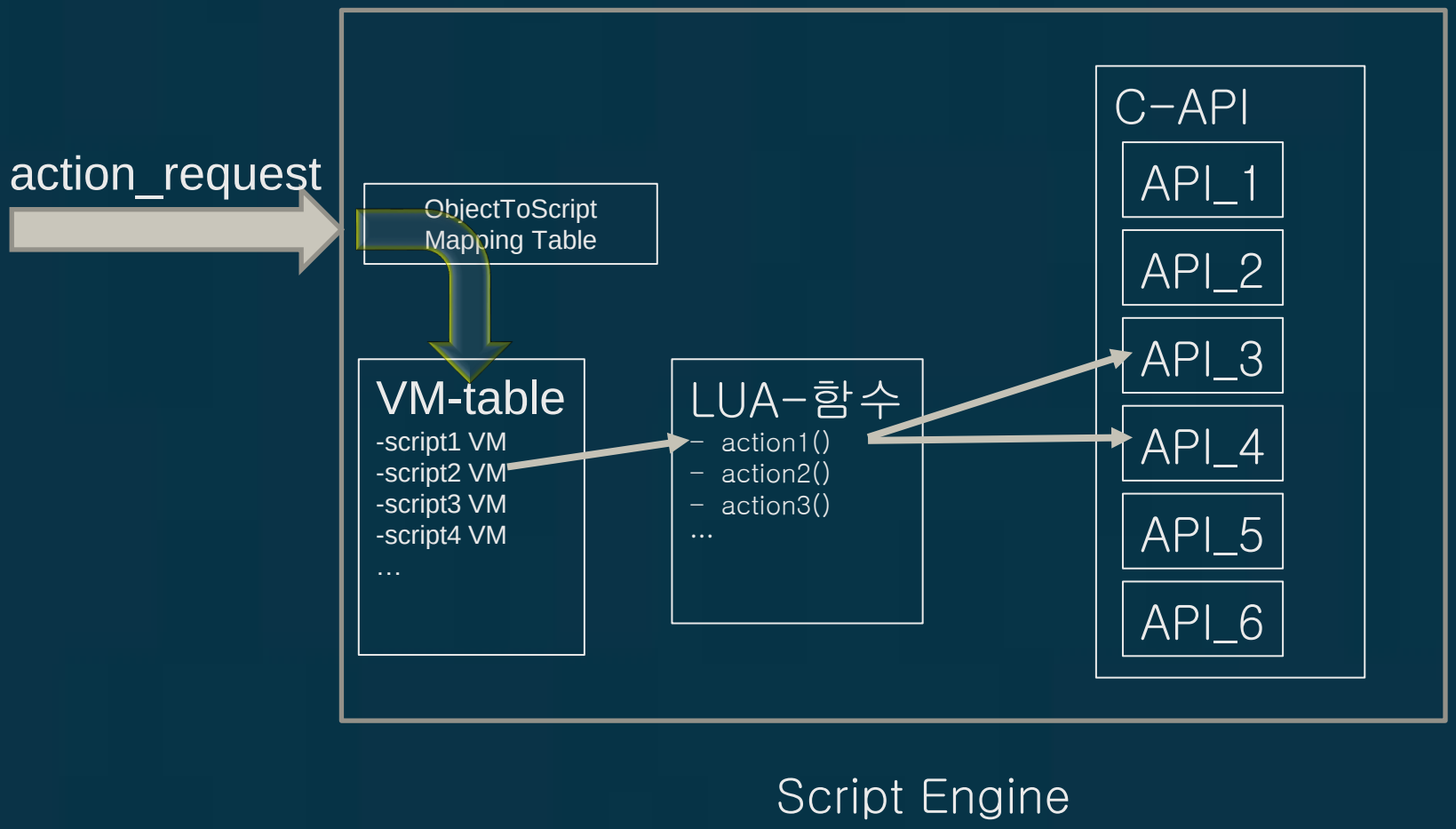
NPC SCRIPT 구현

- 성능 문제
  - 모든 NPC를 script로 돌리는 것은 거대한 삽질
    - 간단한 AI는 하드코딩으로.
  - 멀티 쓰레드 최대한 활용
    - 1 Thread & 1 VM : 하나의 VM이 모든 스크립트 객체를 콘트롤, 성능 문제, bottle neck
    - N Thread & 1 VM : 멀티 쓰레드에서 동시 호출이 가능한 스크립트 언어 필요, Worker Thread들과의 스케쥴링 충돌
    - N Thread & N VM : 하나의 VM이 여러 개의 스크립트객체를 실행, Load Balancing 문제, Worker Thread들과의 충돌
      - 객체가 존재하는 worker thread 선택 문제
    - 0 Thread & MM VM : 객체 하나당 하나의 VM, VM은 worker thread에서 실행, 대부분의 VM이 대기상태, 메모리 낭비

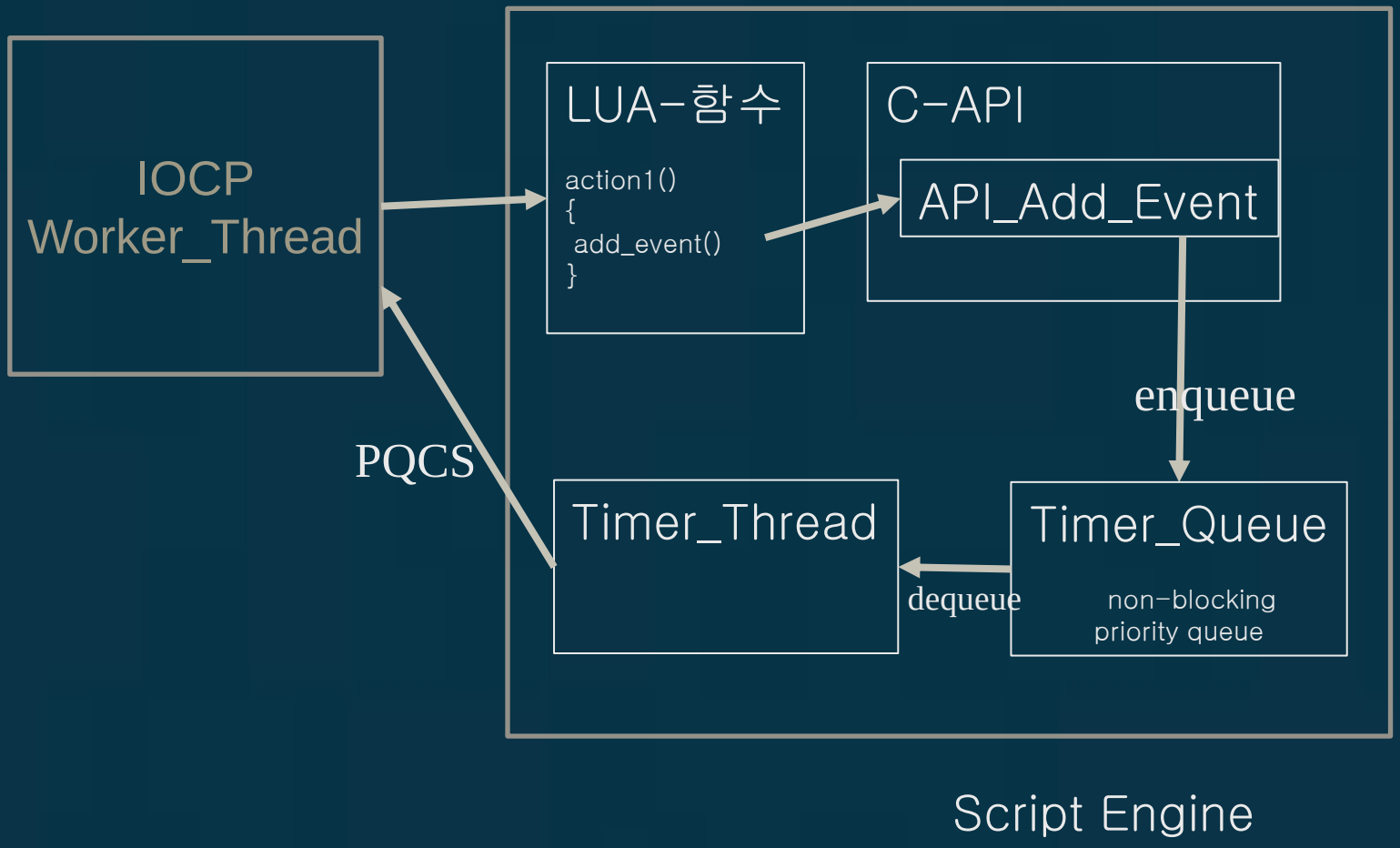
NPC SCRIPT 구현

- Massively Multi Virtual Machine
  - 스크립트 컨트롤되는 객체마다 하나의 가상머신
  - LUA니까 가능
  - Event 별로 해당 객체의 적절한 LUA함수 호출
  - Script에서 사용할 API를 게임서버에서 제공해 주어야 함.
    - Script언어가 실행하기 곤란한 기능들
    - 예) 이동, 메시지, 다른 객체 이벤트 호출, 주변 Object 검색, 클라이언트에 명령 전달, 실행시간이 많이 걸리는 기능

NPC SCRIPT 구현



NPC SCRIPT 구현



NPC SCRIPT 구현

- 구현
  - NPC 정보 구조체에 LUA_STATE를 하나 할당.

```
struct CLIENT_INFO {
   int   id;
   bool in_use;
   SOCKET my_socket;
   int   x;
   int   y;
   EXOVER        recv_over_ex;
   std::set<int>  view_list;
   CRITICAL_SECTION  vl_lock;
   lua_State*L;
};
```

NPC SCRIPT 구현

- AI 구현
  - 플레이어가 바로 옆에 다가가면 “Hello” 메시지 출력.

NPC SCRIPT 구현

- AI 구현 단계
  1. 플레이어 이동 시 주위 NPC에게 Event 생성
     - PLAYER MOVE 패킷 처리 로직에서 시작
     - PLAYER_MOVE 이벤트를 만들어서 주위 NPC에게 뿌린다. (Event는 확장 Overlapped 구조체에 넣는다.)
     - PQCS로 WorkerThread에게 처리 넘긴다
  2. LUA Script 실행
     - 스크립트에서 위치를 검사한 후 클라이언트에 “Hello”, 전송

NPC SCRIPT 구현

- 프로토콜 추가

```
#define MAX_STR_SIZE  100

#define SC_CHAT  4

struct sc_packet_chat{
   BYTE size;
   BYTE type;
   WORD id;
   char message[MAX_STR_SIZE];
};
```

- 클라이언트 수정
  - 채팅 메시지 표시

NPC SCRIPT 구현

- Chatting Packet 처리 함수 추가

```
void send_chat_packet(int to, int id, WCHAR *message)
{
   sc_packet_chat  pos_packet;

   pos_packet.id = id;
   pos_packet.size = sizeof(sc_packet_chat);
   pos_packet.type = SC_CHAT;
   strncpy(pos_packet.message, message, MAX_STR_SIZE);
   send_packet(to, reinterpret_cast<char*>(&pos_packet));
}
```

NPC SCRIPT 구현

- EVENT 정의
  - WorkerThread에서 처리해야 하는 이벤트들

```
#define IOTYPE_SEND        1
#define IOTYPE_RECV        2
#define IOTYPE_PLAYER_MOVE 3
#define IOTYPE_MOVE_SELF   4
```

NPC SCRIPT 구현

- 몬스터 LUA 프로그램

```
myid = 99999;

function set_uid(x)
   myid = x;
end

function event_player_move(player)
   player_x = API_get_x(player);
   player_y = API_get_y(player);
   my_x = API_get_x(myid);
   my_y = API_get_y(myid);
   if (player_x == my_x) then
      if (player_y == my_y) then
         API_SendMessage(myid, player, "HELLO");
      end
   end
end
```

NPC SCRIPT 구현

- 초기화

```
void Initialize_VM()
{
   CreateThread(NULL, 0, timer_thread, NULL, 0, NULL);

   for (int i=0;i < NUM_OF_NPC;++i)
      Create_Monster();

}
```

NPC SCRIPT 구현

- 초기화

```
void Create_Monster()
{
…
    lua_State *L = players[my_id].L = luaL_newstate();
    luaL_openlibs(L);

    int error = luaL_loadfile(L, "monster.lua") ||
    lua_pcall(L,0,0,0);

    lua_getglobal(L,"set_uid");
    lua_pushnumber(L, my_id);
    lua_pcall(L,1,1,0);
    lua_pop(L, 1);// eliminate set_uid from stack after call
…
```

NPC SCRIPT 구현

- 초기화

```
…

    lua_register(L, "API_SendMessage", API_SendMessage);
    lua_register(L, "API_get_x", API_get_x);
    lua_register(L, "API_get_y", API_get_y);
}
```

NPC SCRIPT 구현

- API

```
int API_get_y(lua_State *L)
{
   int user_id =
        (int)lua_tointeger(L, -1);
   lua_pop( L, 2 );
   int y = players[user_id].y;
   lua_pushnumber(L, y);
   return 1;
}
```

```
int API_SendMessage(lua_State *L)
{
   int my_id = (int)lua_tointeger(L, -3);
   int user_id = (int)lua_tointeger(L, -2);
   char *mess = (char *)lua_tostring( L, -1);

   lua_pop( L, 4 );

   send_chat_packet( user_id, my_id, mess);
   return 0;
}
```

NPC SCRIPT 구현

- Worker Thread

```
} else if (overlapped->io_type == IOTYPE_SEND) {
          delete overlapped;
} else if (overlapped->io_type == IOTYPE_PLAYER_MOVE) {
          lua_getglobal( players[id].L, "event_player_move" );
          lua_pushnumber( players[id].L, overlapped->ioc_caller_id );
          lua_pcall( players[id].L, 1, 1,0 );

          //lua_pop( players[id].L, 1 );
          delete overlapped;
} else if (overlapped->io_type == IOTYPE_MOVE_SELF) {
          // 랜덤 이동 호출
          // AddTimer
} else {
          printf("Error invalid overlapped\n");
          exit(-1);
}
```

NPC SCRIPT 구현

- Event 생성
  - Process Packet에서 이동 후 주위 NPC에게 IOTYPE_PLAYER_MOVE event를 PQCS로 보냄

숙제 (#7)

- AI스크립트 Timer 연동
  - 내용
    - 실습프로그램 확장
      - 플레이어가 다가가면 “Hello”란 메시지를 출력하고 랜덤한 방향으로 3칸 이동
        - 1초에 한 칸 씩 이동
        - 이동 종료 후 “BYE”란 메시지 출력
      - 위의 AI는 LUA로 구현
      - 첨부파일의 스크립트 연동을 참조할 것. (반드시 첨부파일을 수정해서 제출하는 것은 아님)
    - Dummy Client를 사용한 성능 측정 Report
      - LUA 스크립트를 사용하지 없을 때, 숙제 #5와 성능 비교
  - 목적
    - Timer와 LUA 스크립트 연동 학습
  - 제약
    - Windows에서 Visual Studio로 작성 할 것
  - 제출 Eclass