



# NPC 실습

MM4220 게임서버 프로그래밍  
정내훈

# NPC

- NPC
  - Non Playing Character
  - 예)
    - Monster
    - 상점 주인
    - 퀘스트 의뢰인
  - 서버 컴퓨터가 조작
    - 인공지능 필요

# NPC

- 인공 지능
  - 재미있는 행동/반응
  - 재미
    - Cheating사용은 재미를 떨어 뜨림
    - 너무 잘해도 곤란 : monster의 존재 목적?
  - Script로 구현
  - 가장 기본적인 인공지능
    - 길 찾기
    - 적 인식

# NPC

- Script

- 뒤에 더 자세히
- 사용 목적 : 게임 제작 파이프라인 단축
  - 프로그래머의 개입 없이 기획자가 직접 작성/Test
  - 서버 리부팅도 필요 없게 할 것.
- XML, LUA 같은 언어를 많이 사용
  - 독자적인 언어를 쓰는 곳도 있음

# NPC

- 적 자동 인식 (어그로 몬스터)
  - 몇 되지 않는 능동적 AI
  - 능동적 AI
    - 플레이어에 의하지 않는 자발적 동작
    - 서버에 막대한 부하. 절대 피해야 함
      - NPC 개수 10만
  - 실제 구현은 수동적
    - 플레이어의 이동을 근처 NPC에만 broadcast
      - 플레이어 생성, NPC생성시 검사 필요
    - 움직이지 않으면 인식 못하는 경우가 생길 수 있음.

# NPC

- NPC 구현

- NPC 서버를 따로 구현 하는가?

- NPC 서버 구현의 장점

- 안정성 : NPC 모듈이 죽어도 서버 정상 작동
    - 부하 분산 : 메모리 & CPU

- NPC 서버 구현의 단점

- 통신 overhead,
      - 공유메모리 참조로 끝날 일이 패킷통신으로 악화.
      - 서버 입장에서는 NPC도 플레이어와 비슷한 부하

# 길 찾기

- Avatar, NPC 이동의 기본
  - Avatar이동?
    - WASD이동 시 불필요
    - 마우스 클릭 이동 시 필요 => 꼭 서버에서 할 필요는 없음
  - NPC
    - 서버구현 필수
      - 서버가 지형과 장애물을 알아야 한다.
- 지형 구현과 밀접한 관련
  - 2D, 3D?
  - Tile, Polygon?

# NPC 구현

- 고려할 점
  - ID 생성 체계
    - 플레이어와 NPC의 구분
      - bool is\_npc?
        - Object 객체 내부 접근 필요.
      - ID의 영역의 구별
        - 멤버 변수를 확인 하지 않아도 됨.
        - NPC\_START 상수 필요
    - 자료구조 선택
      - 고정 자료구조 : array 또는 vector
      - 동적 자료구조 : map



# NPC 구현

- 고려할 점
  - ID 생성 체계
    - 플레이어와 NPC의 구분
      - bool is\_npc?
        - Object 객체 내부 접근 필요.
      - ID의 영역의 구별
        - 멤버 변수를 확인 하지 않아도 됨.
        - NPC\_START 상수 필요
    - 자료구조 선택
      - 고정 자료구조 : array 또는 vector
      - 동적 자료구조 : map

# NPC 구현

- 1차 구현
  - 구현 내용
    - 1초에 1번 랜덤 무브
  - AI Thread를 사용한 구현
    - AI Thread에서 시간 검사 및 이동 판단
  - HEART BEAT을 사용한 구현
    - HEART BEAT자체가 기준 시간에 맞춰 호출됨.
  - NPC 시야 구현
    - 시야 리스트를 따로 저장할 필요는 없음.
      - 클라이언트가 연결되어 있지 않기 때문

# Timer(2018 화수)

```
DO (A) ;  
Sleep(1000) ;  
DO (B) ;
```

일반 프로그램

```
Loop(true) {  
    if (false == A_done) {  
        DO (A); A_done = true;  
        B_time = current_time() + 1000; }  
    if (B_time <= current_time()) {  
        DO(B);  
        B_time = MAX_INT;  
    }  
}
```

클라이언트 프로그램

# Timer

- 앞의 동작의 문제점
  - 10만개의 NPC가 출동하면?
    - 매 루프마다 10만번의 if 문이 필요.
      - 서버는 GPU bound가 아니다.
      - **Busy Waiting**
    - 캐시 문제, pipeline stall
- 해결책?
  - NPC Class에 heart\_beat() 함수를 두고 일정 시간 간격마다 호출 되게 한다.

# Timer

```
Loop(true) {  
    if (objA.next_heal_time < current_time) {  
        objA.m_hp = objA.m_hp + 1;  
        objA.next_heal_time += HEAL_INTERVAL;  
    }  
}
```

서버 메인루프에서 검사 : busy wait

```
Cobj::heart_beat()  
{  
    m_hp += HEAL_AMOUNT;  
}
```

1초마다 호출 : heart\_beat

# Timer

- Heart\_beat 함수.

- 자율적으로 움직이는 모든 NPC를 살아있도록 하는 함수.
- 외부의 요청이 없어도 독자적으로 AI를 실행
- 구현

- Heart\_beat\_thread

```
while(true) {  
    curr_heart_beat = current_time();  
    for (int i =0 ; i < MAX_NPC; ++i)  
        NPC[i].heart_beat();  
    delay = DURATION - (current_time() - curr_heart_beat);  
    delay = MAX(0, delay);  
    Sleep(delay);  
}
```

# Timer

- Heart\_beat 함수의 문제
  - 10만개의 NPC라면?
    - busy waiting은 없지만 아무일도 하지 않는 heart\_beat이 시간을 잡아 먹는다.

```
heart_beat()  
{  
    my_hp += HEAL_AMOUNT;  
}
```

이론

실재

```
heart_beat()  
{  
    if (my_hp < my_max_hp)  
        my_hp += HEAL_AMOUNT;  
}
```

# Timer

- Heart\_beat 함수의 문제 해결 - 1
  - 필요한 경우만 heart\_beat이 불리도록 한다.
    - 복잡한 NPC의 경우 프로그래밍이 어려워 진다.
      - Heart\_beat함수안의 수많은 if
      - 불리지 않는 경우를 판단하기가 힘들다.
- Heart\_beat 함수의 문제 해결 - 2
  - heart\_beat함수를 없앤다.
  - 각 모듈에서 timer를 직접 사용한다.



# NPC 구현

---

- 1차 구현 – 실습
  - AI Thread 구현

# NPC 구현

- 1차 구현 – HEART BEAT 구현
  - AI Thread의 간략화
  - 시간 동기화 Logic의 간소화

# NPC 구현

---

- 2차 구현 – 시야 구현
  - 부하 측정 및 비교.

# Timer

- Timer 기반 동작
  - Timer : 서버 동작 기본 요소
    - Real time 동작을 위해 필요
    - Timing에 맞춘 동작들 구현
      - 이동, 마법 시전, HP회복...
  - NPC 이동
    - Timer에 동기화 되어 있다.
      - 이동시간이 될 때 까지 기다렸다가 이동 (X)
      - Timer에 다음 이동시간을 등록하고 종료 (O)

# Timer (2019 화목)

- 컨텐츠 구현의 뼈대
  - 각종 이벤트 구현
  - Timing에 맞춘 동작들 구현
    - 캐스팅 타임, 쿨타임
    - 이동, 마법 시전, HP회복...
  - NPC AI
    - Timer 기반의 Finite State Machine
    - 주기적으로 상황 파악. (그러나 현실은...)

# Timer

```
heart_beat()  
{  
    if (my_hp < my_max_hp)  
        my_hp += HEAL_AMOUNT;  
}
```



```
get_damage(int dam)  
{  
    my_hp -= dam;  
    add_timer(my_heal_event, 1000);  
}  
  
my_heal_event()  
{  
    my_hp += HEAL_AMOUNT;  
    if (my_hp < my_max_hp)  
        add_timer(my_heal_event, 1000);  
}
```

# Timer

- Timer thread의 구현

```
Event_queue timer_queue

TimerThread()
do {
    sleep(1)
    do {
        event k = peek (timer_queue)
        if k.starttime > current_time()
            break
        pop (timer_queue)
        process_event(k)
    } while true;
} while true;
```

# Timer

- Timer Thread와 Worker Thread의 연동
  - timer thread에서 할 일
    - 모든 AI
    - 이동, 길찾기 등
  - timer thread의 과부하 => 서버 랙
  - 실제 작업은 worker thread에 넘겨야 한다.
    - concurrency control도 겸한다.



# Timer

## • Timer Thread와 Worker Thread의 연동

```
NPC_Create()  
    foreach NPC  
        add_timer(my_id, MOVE_EVENT, 100)
```

```
Timer_thread()  
    ...  
    overlap_ex.command = MOVE  
    PostQueuedCompletionStatus(port, 1, id, overlapex)  
    ...
```

```
Worker_thread()  
    ...  
    if (overlap_ex.command == MOVE) move_npc(id);  
    ...
```

# 숙제 (#5)

- 게임 서버/클라이언트 프로그램 작성
  - 내용
    - 숙제 (#4)의 프로그램의 확장
    - 프로그램 수정
      - 전체 지도는 800 x 800
      - 클라이언트는 자기 말 주위 21x21 표시, 시야는 15x15
      - 200000개의 NPC : 1초 마다 한 칸 씩 이동
  - 목적
    - NPC 및 Timer 개념 사용 (PQCS와 GQCS를 사용)
  - 제약
    - Windows에서 Visual Studio로 작성 할 것
  - 제출
    - 제목을 “2019 게임서버[화목] 학번, 이름 숙제5”, “2019 게임서버[수목] 학번, 이름 숙제5” 로 할 것
    - 5월 5일[수목반은 5월 15일] 오후 1시까지 제출 (1일 당 10% 감점)
    - Zip으로 소스를 묶어서 e-mail로 제출
      - 소스만(sdf, obj, log, manifest 같은거 제외!)
    - E-mail주소는 nhjung@kpu.ac.kr

# NPC

- Timer 구현
  - Timer Thread

```
Event_queue timer_queue
```

```
TimerThread()
```

```
do
```

```
    sleep(1)
```

```
do
```

```
    event k = timer_queue.top
```

```
    if k.starttime > current_time()
```

```
        break
```

```
    timer_queue.pop(&id, &k)
```

```
    process_event_callback(id, k)
```

```
while true;
```

```
while true;
```

# Timer

- NPC
  - IOCP를 통해서 NPC제어?
    - Thread programming!
    - PC에 관련된 Event도 IOCP를 통해서!!
  - Timer Thread에서 A\*를 할 수는 없다.
    - Timer Thread과부하!

# Timer

- 이벤트 큐
  - 저장 정보
    - 어떤 오브젝트가 언제 무엇을 누구에게 해야 하는가.

```
struct event_type {  
    int obj_id;  
    high_resolution_clock::time_point wakeup_time;  
    int event_id;  
    int target_id;  
};
```

# Timer

- 이벤트 큐
  - 시간 순서대로 정렬된 우선순위 큐가 필요하다.

```
class mycomparison
{
    bool reverse;
public:
    mycomparison() {}
    bool operator() (const event_type lhs, const event_type rhs) const
    {
        return (lhs.wakeup_time > rhs.wakeup_time);
    }
};

priority_queue<event_type, vector<event_type>, mycomparison> p_queue;
```

# NPC

- NPC

- Timer Queue와 Worker Thread 만으로는 부족
- 대부분의 NPC가 timer queue로 동작한다면 timer thread의 과부하
- 플레이어가 관찰할 수 있는 NPC만 움직여야 한다.
  - Is\_alive 이외에 is\_active 필요.

# NPC

- NPC : 타이머를 사용한 이동

```
Event_queue timer_queue
```

```
NPC_Create()
```

```
    foreach NPC
```

```
        push (timer_queue, id, MOVE_EVENT, 1)
```

```
NPC_CALLBACK(id, event)
```

```
    if (event == MOVE_EVENT)
```

```
        id -> move_npc()
```

```
        push (timer_queue, id, MOVE_EVENT, 1)
```

```
MOVE_NPC()
```

```
    overlap_ex.command = MOVE
```

```
    PostQueuedCompletionStatus(port, 0, &NPC_INFO, overlap_ex)
```



# NPC

- NPC : 타이머를 사용한 이동

```
Event_queue timer_queue
```

```
NPC_Create()
```

```
MOVE_PLAYER() {
```

```
    foreach_monster_in_range( push(timer_queue, monster_id, MOVE_EVENT,1);
```

```
    foreach_monster_get_away( try_erase_timer_queue(monster_id));
```

```
}
```

```
NPC_CALLBACK(id, event)
```

```
    if (event == MOVE_EVENT)
```

```
        id -> move_npc()
```

```
        push (timer_queue, id, MOVE_EVENT, 1)
```

```
MOVE_NPC()
```

```
    overlap_ex.command = MOVE
```

```
    PostQueuedCompletionStatus(port, 0, &NPC_INFO, overlapex)
```

# NPC

---

- NPC : 실습
  - InActive상태 구현
  - 플레이어나 NPC 이동/생성 시 active 여부 검사