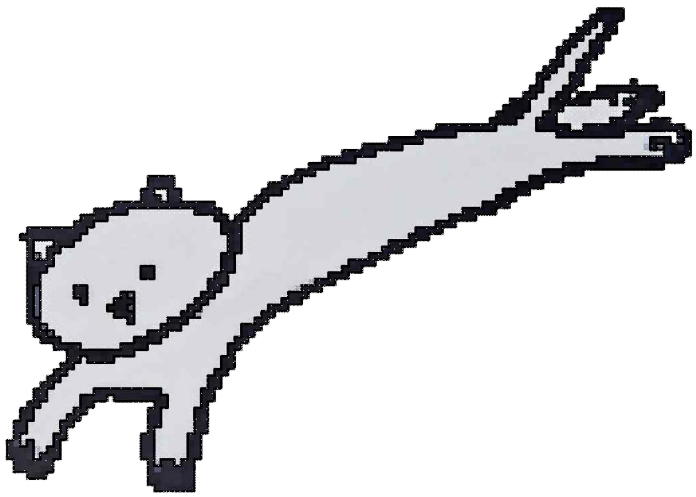
개발기간: 2025. 6. 20. ~ 2025. 9. 25.

Simple KO WAR GAME

(Discrete Event System Specification을 활용한 시뮬레이터)

(Ver 1.0)



2025. 10. 01.

고 재 현

목 차

표 목 차

그 림 목 차

1. 프로그램 개요

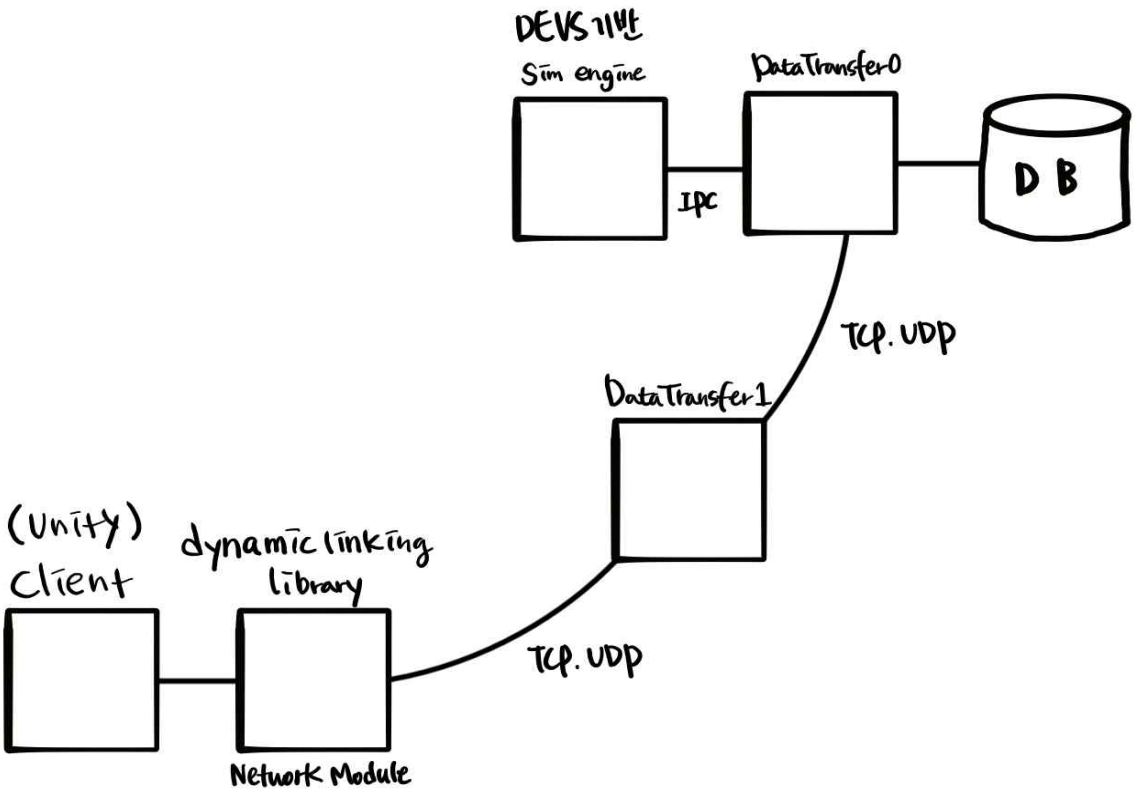
1.1 서론

이 워게임은 국방 지휘관 훈련용 modeling & simulation을 약간의 게임적 요소를 첨가하여 개발되었습니다. 시뮬레이션 엔진은 DEVS(Discrete Event System Specification)을 기반으로 만들어졌으며 한국과학기술원(KAIST) 김탁곤 교수님의 강의, 논문을 참고하였습니다.

-1994 "DEVSim++ : C++ Environment for Modeling/Simulation of DEVS Models" 한국시뮬레이션학회

등 6.2 참고 문헌 및 미디어에 기재

1.2 소프트웨어(SW) 구조



[그림 1] 소프트웨어 전체 구조

■ 클라이언트 프로그램:

- 운영체제 : windows os
- 사용언어 : C#(unity), C++(NetworkModule.dll)
- 모의통제도구, 전장상황도 등을 통합하여 구현

■ 데이터 전달 처리기

- 운영체제 : linux os(ubuntu 22.04)
- 사용언어 : C++
- 클라이언트와 시뮬레이션 엔진, DB서버 간 패킷 송수신 담당

- internal server: 시뮬레이션 엔진과 ipc(공유메모리), 하위서버와 멀티플렉싱(epoll) 방식으로 통신
- external server: 클라이언트, 상위서버와 멀티플렉싱(epoll) 방식으로 통신

■ 시뮬레이션 엔진

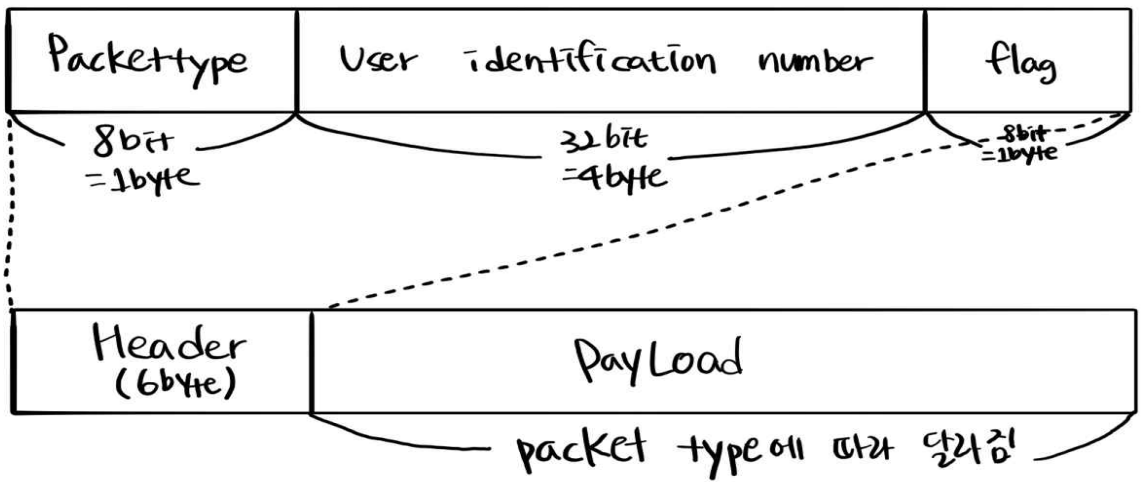
- 운영체제 : linux os(ubuntu 22.04)
- 사용언어 : C++
- 이벤트 관리: 이산 사건 이벤트를 관리하고 결과를 도출
- 시간 동기화: 실제 시간과 시뮬레이션을 동기화
- 모델 관리: 정의된 모델의 상호작용을 이벤트로 관리
- 이산 사건 시스템: DEVS를 기반으로 시뮬레이션을 구현

■ DB server

- MySQL 서버 사용

1.3 프로토콜(protocol) 정의

■ Packet Header 구조



[그림 2] Packet header structure

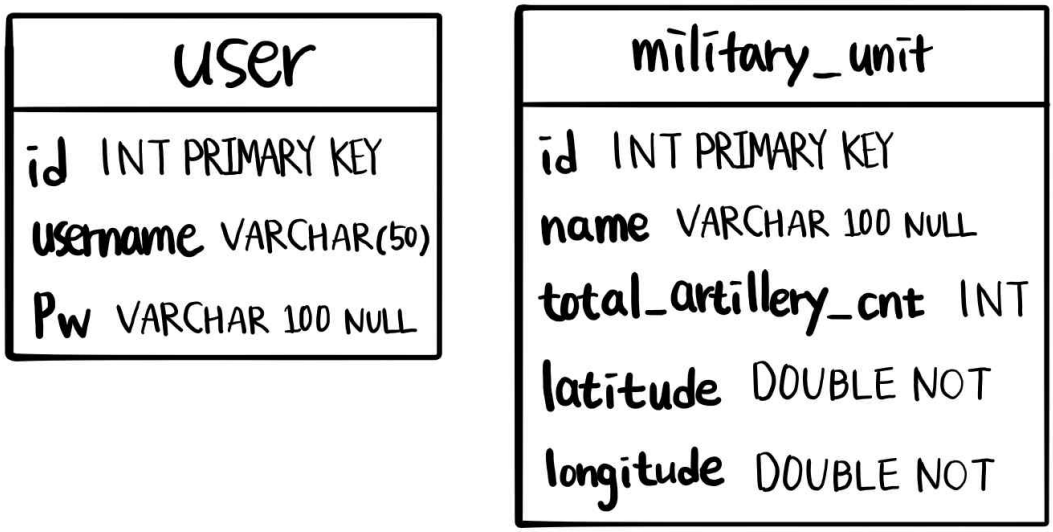
■ 프로토콜 세부내용

[표 1] 프로토콜 세부내용

packet type			pay load (bytes)	세부사항
프로토콜	연결	REQUEST_LOGIN	132	• client의 로그인, 로그아웃 • client udp sockaddr 정보 전달
		RESPONSE_LOGIN	7	
		DISCONNECTED_LOGIN_SYSTEM	0	
		HELLO_UDP	0	
		HELLO_UDP_RESPONSE	0	
		REQUEST_CONNECT	0	• 데이터 전달 처리간 연결
		RESPONSE_CONNECT	0	
		UPDATE_ROUTING_INFO	4	• 클라이언트 라우팅 info 전달
	시뮬레이션 통제	REQUEST_SIM_START	0	• 시뮬레이션 시작, 중단 요청
		REQUEST_SIM_STOP	0	
		RESPONSE_SIM_START	0	
	객체 이벤트 명령	REQUEST_ORDER_IN_SALLY	14	• 각 모델에 외부 이벤트 주입 • 미사일 발사 명령, 출격 명령 등
		RESPONSE_ORDER_IN_SALLY	40	
		REQUEST_ORDER_IN_ATTACK	14	
		RESPONSE_ORDER_IN_ATTACK	34	
	시뮬레이션 동기화	SYNC_ARTILLERY_COORDINATE	22	• 기본적으로 udp통신, 도착시 tcp • 미사일, 자주포 위치 동기화
		SYNC_MISSILE_COORDINATE	16	
		SYNC_ARTILLERY_ATTACKED	22	• 부대, 자주포 피격 이벤트 수신
		SYNC_MILITARY_UNIT_ATTACKED	22	

packet type			pay load (bytes)	세부사항
프로토콜		SYNC_SIM_TIME	4	• 시뮬레이션 시각 동기화
	객체 정보 요청	REQUEST_LOAD_MILITARY_UNIT_FROM_DB	0	• DB에 초기 부대 정보 요청
		RESPONSE_LOAD_MILITARY_UNIT_FROM_DB	17	
		REQUEST_MILITARY_UNIT_INFO	6	• 서버에 로드된 부대 정보 요청
		RESPONSE_MILITARY_UNIT_INFO	23	

1.4 ERD (Database table)



[그림 3] 데이터 베이스 테이블 설계

■ User
- user의 id, password를 저장하는 테이블

■ Military_unit
- 부대 정보를 저장하는 테이블

✓ 많은 기능을 넣지 않아 간단하게 정의하였습니다.

2. 소프트웨어(SW) 실행 매뉴얼

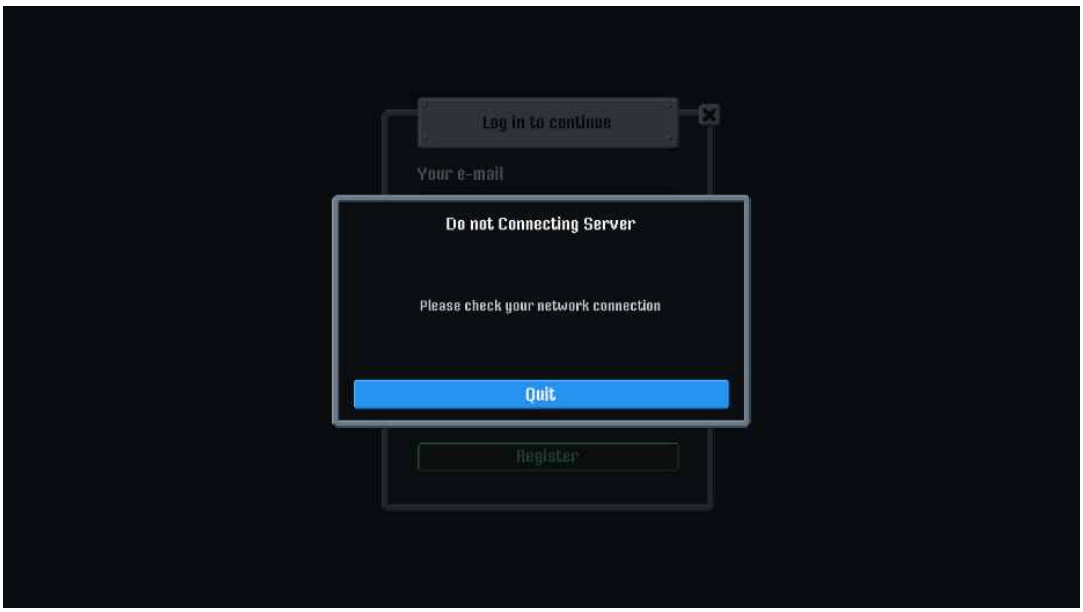
2.1 클라이언트

■ Dynamic Linking Library 업데이트

• \KoWarGame_Data\Plugins\x86_64 경로에 NetworkModule.dll을 덮어쓴다.

■ 서버 tcp, udp 포트와 ip를 확인하고 수정 후 build된 응용 프로그램을 실행.

• 반드시 서버를 먼저 열린 상태에서 접속을 해야 동작한다.



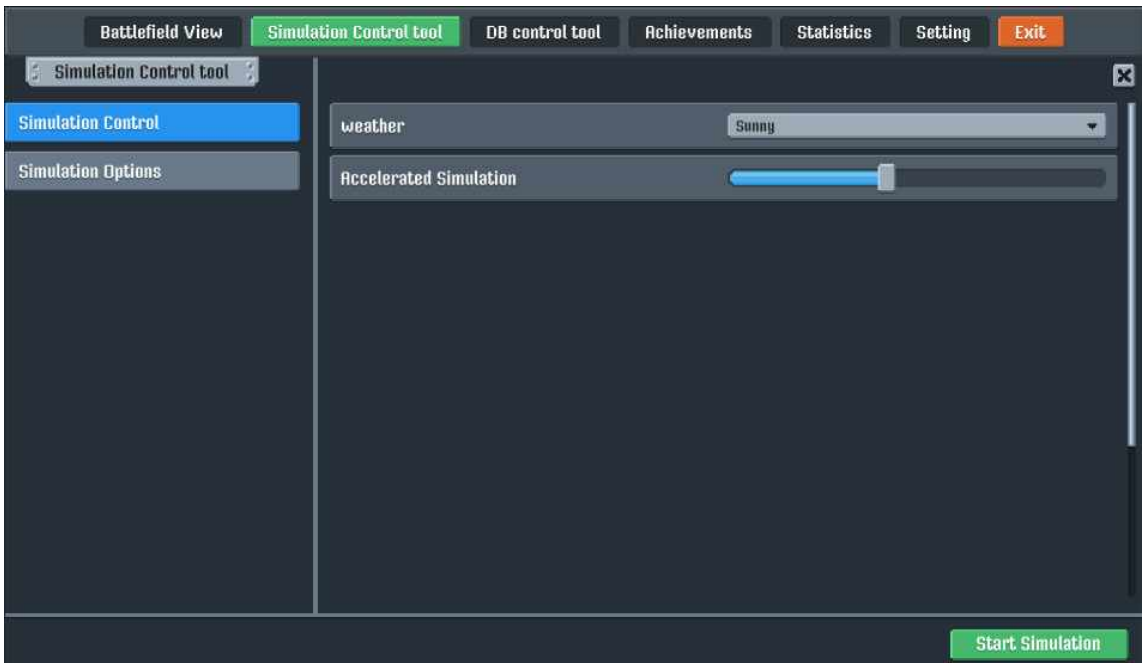
[그림 4] 네트워크 연결 실패 시 클라이언트 화면

■ 로그인 화면에서 ID: admin PW: admin123을 입력하고 접속한다.

[그림 5] 클라이언트 로그인 화면

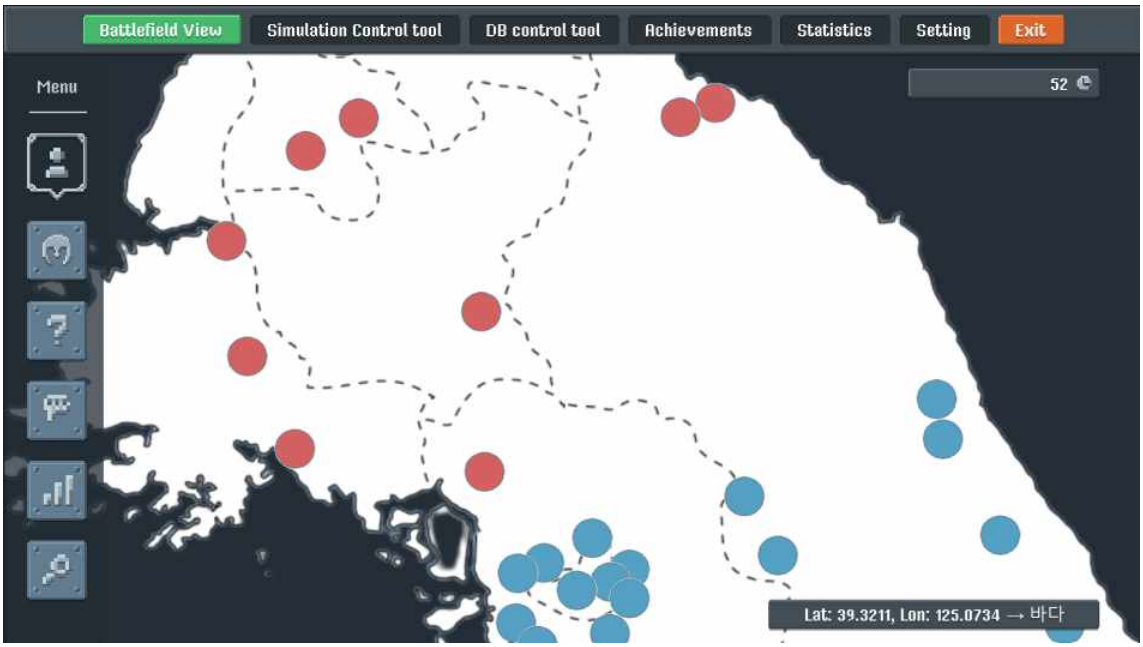
■ [Simulation Control Tool] tab에서 시뮬레이션 시작을 누른다.

• 시뮬레이션이 시작되면 [BattleField View] tab이 활성화 된다.



[그림 6] 클라이언트 시뮬레이션 컨트롤 탭 화면

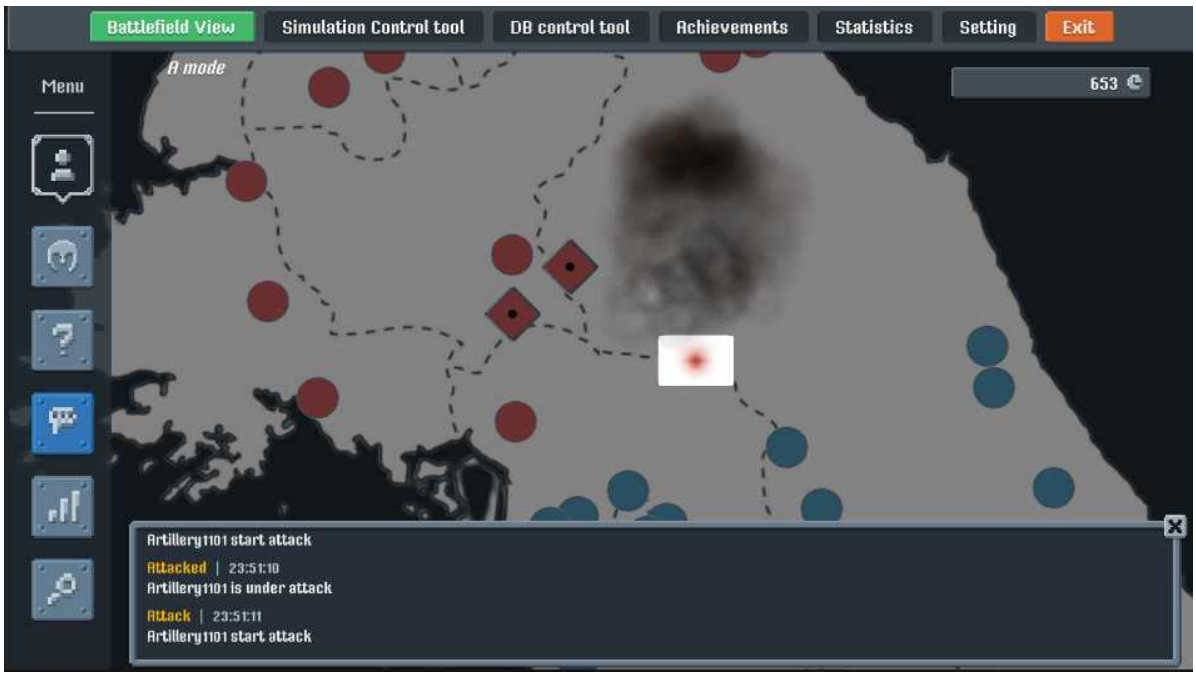
■ [BattleField View] tab에서 각 부대에 명령을 내릴 수 있게 된다.



[그림 7] 클라이언트 전장 상황도 화면

■ [BattleField View] 부대에 출격 명령, 자주포에 공격명령을 내린다.

[그림 8] 클라이언트 전장 상황도 유닛 명령 화면



[그림 9] 클라이언트 전장 상황도 유닛 피격 화면

2.2 서버

2.2.1 데이터 전달 처리 서버

■ 서버 ip, port, key 설정

• cd /bin/cfg 후 vim sysconfig.cfg에서 상황에 맞게 값을 변경한다.

```
#DataTrsferModel systemconfig
[DATATRANSFER_INTERNAL_INFO]
IP_ADDRESS=0.0.0.0
PORT=7400
UDP_PORT=7401

[DATATRANSFER_EXTERNAL_INFO]
UPPER_IP_ADDRESS=0.0.0.0
UPPER_PORT=7400
UPPER_UDP_PORT=7401
EXTERNAL_IP_ADDRESS=0.0.0.0
EXTERNAL_PORT=12345
EXTERNAL_UDP_PORT=12346

#DataTrsfer upperserver connect key
[DATATRANSFER_KEY_INFO]
KEY=1234
~
~
~
~
~
~
"sysconfig.cfg" 17L, 335B                                    1,1           모두
```

[그림 10] sysconfig.cfg vim화면

■ Database 설정

• 마찬가지로 mysql server를 설치후 database.sql을 실행하고 database.cfg를 수정.

```
[DATABASE_ADMIN]
IP_ADDRESS=127.0.0.1
PORT=3306
ID=admin
PW=1234
DATABASE_NAME=community
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"database.cfg" 6L, 89B                                       1,1           모두
```

[그림 11] database.cfg vim화면

■ 컴파일, 링킹, 빌드

- makefile에서 Home 변수를 사용자의 환경에 맞게 수정한다.
- mysql conn/cpp lib 의존성이 있으니 설치 후 makefile을 입맛에 맞게 수정한다.
- /DataTrsfer 경로에서 ./r 파일을 실행한다.

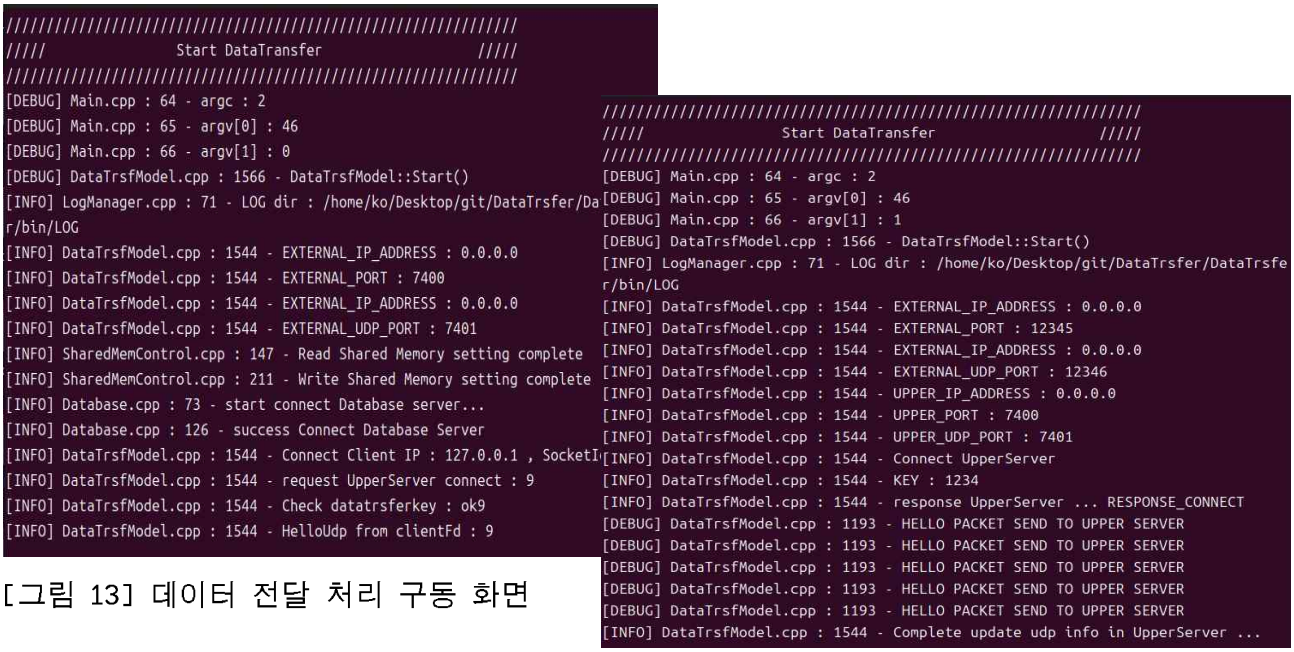
[그림 12] 데이터전달처리 실행화면

■ 실행모드

- 3번 메뉴를 실행시 internal server, 4번은 external server
- 5번 메뉴는 디버깅용 gdb tool을 사용할 수 있는 모드이다.

■ 실행 순서

- 반드시 internal server를 실행 후, external server를 실행하도록 한다.
- 이는 external server 초기에 internal server와 연결하도록 만들어졌기 때문인데, 추후 internal server가 모종의 이유로 종료되더라도 다시 연결이 되도록 로직을 개선할 여지가 남아있다.
- 또한 공유메모리의 생성과 제거의 책임이 internal server에 있기 때문에 시뮬레이션 서버를 열기 전에 internal server를 먼저 실행한다.



[그림 13] 데이터 전달 처리 구동 화면

■ 종료

- ctrl + c를 입력하여 프로그램 종료 시그널을 보낸다.
- 종료시, 여러쓰레드, 공유메모리, 클라이언트 fd등을 자동으로 모두 close 하도록 설계하였다.

[그림 14] SIGINT 시그널 처리 화면

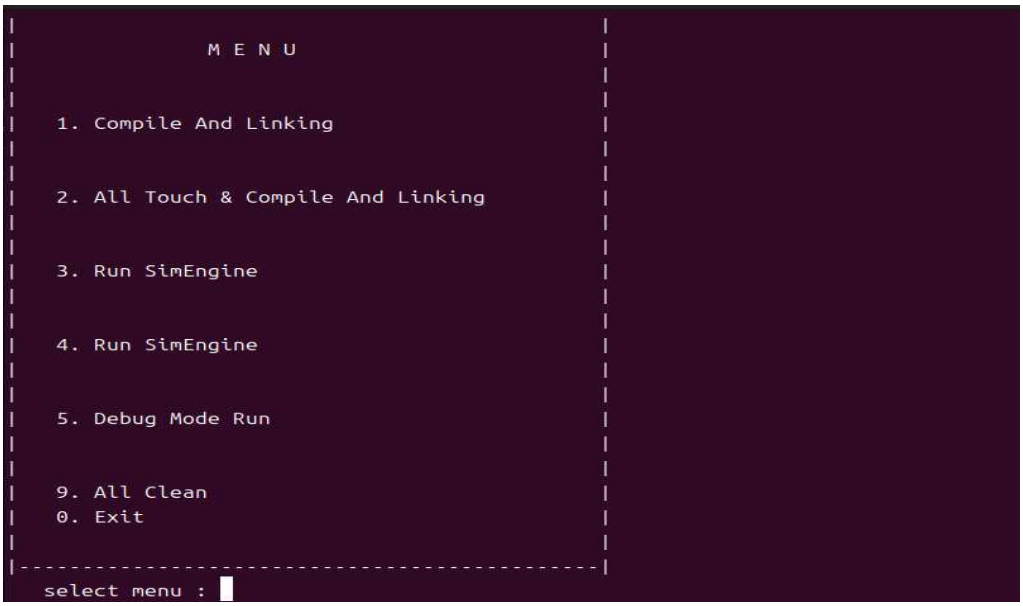
2.2.2 시뮬레이션 엔진 서버

■ 컴파일, 링킹, 빌드

- makefile에서 Home 변수를 사용자의 환경에 맞게 수정한다.

■ 실행모드

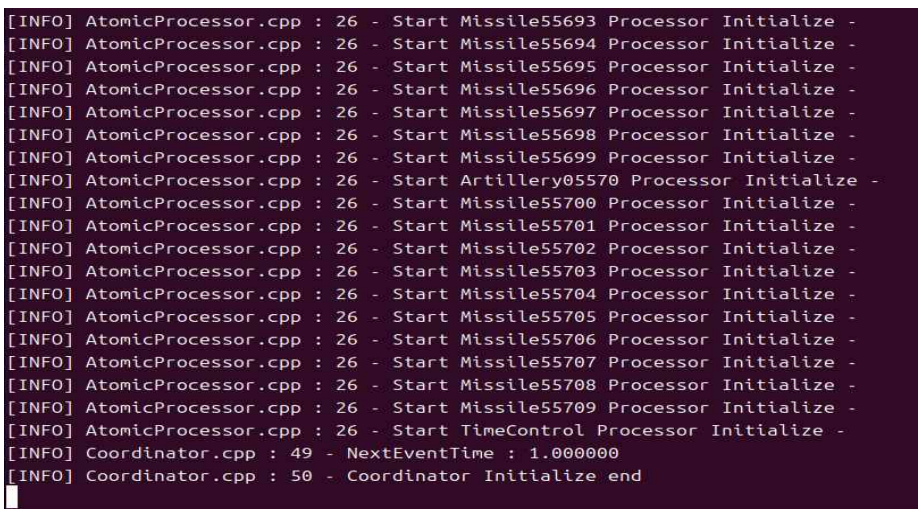
- 3번 메뉴를 실행시 simulatior server가 실행된다.
- 5번 메뉴는 디버깅용 gdb tool을 사용할 수 있는 모드이다.



[그림 15] 시뮬레이션 서버 실행 화면

■ 시나리오 업데이트

- 시뮬레이터를 켜면 internal server에서 시나리오(부대정보)를 받아온다.
- 모두 받으면 받은 정보에 따라 모델들을 생성하고 시뮬레이션을 초기화한다.



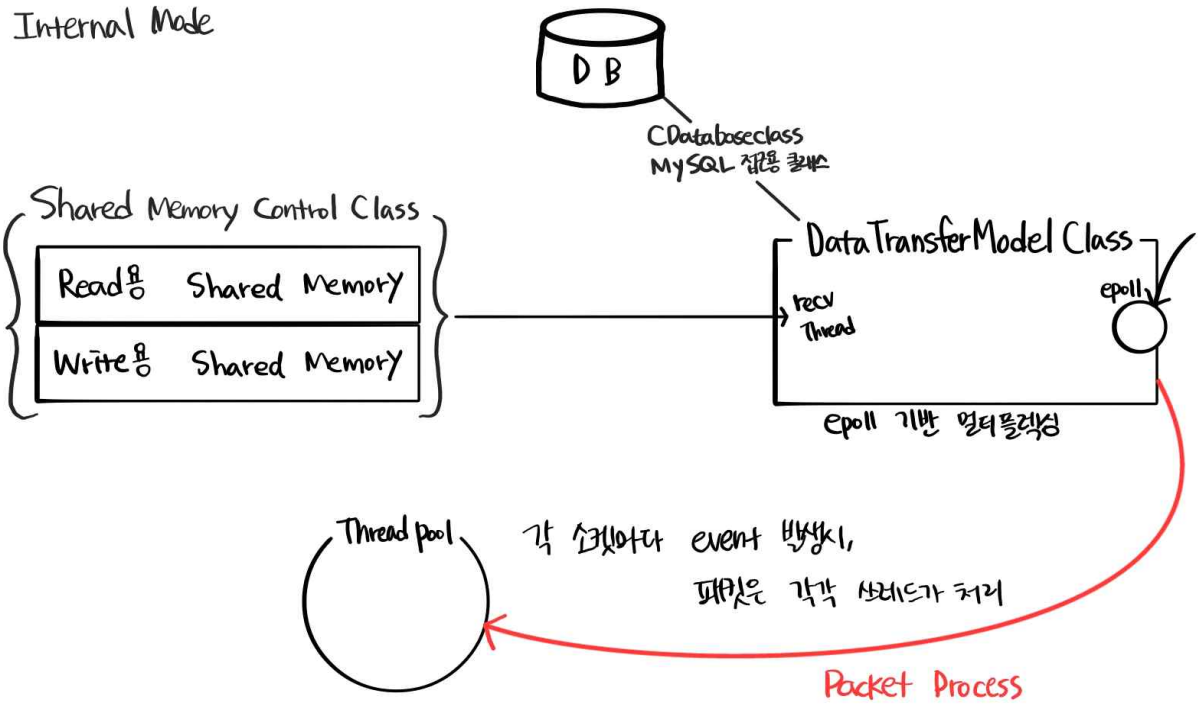
[그림 16] 시뮬레이터 초기화 화면

3. 데이터 전달 처리 서버

3.1 구조

■ 환경

- 운영체제: linux (ubuntu 22.04)
- 사용언어: C++

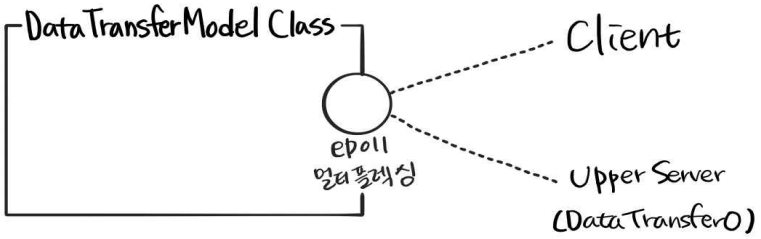


[그림 17] 데이터 전달 처리 서버(internal) 구조

■ Internal mode

- 시뮬레이션 엔진 서버와 IPC(inter process communication)로 통신.
- 하위 데이터 전달 처리서버와 tcp, udp 소켓 기반으로 통신
- 데이터베이스 서버와 통신

추가에 Upper Server (Internal mode) 와 연결 (TCP, UDP 정보전달)



[그림 18] 데이터 전달 처리 서버(external) 구조

■ external mode

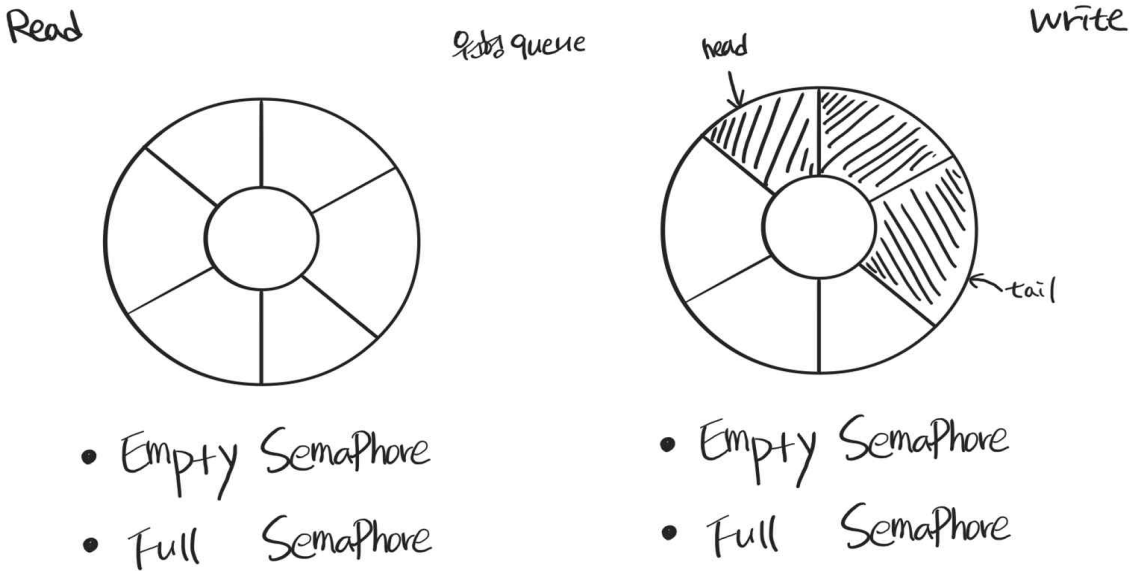
- 상위 데이터 전달 처리 서버, 클라이언트와 소켓 통신
- 멀티플렉싱(epoll) 구조로 효율적인 쓰레드 관리

3.2 상세 기능

3.2.1 IPC (POSIX shared memory)

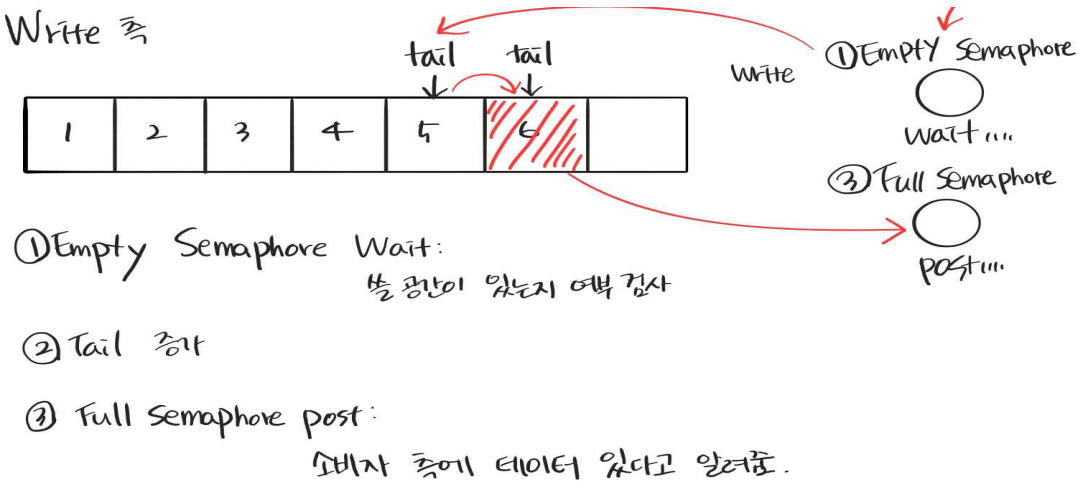
■ 공유메모리

- 시뮬레이션 엔진과 통신하기 위해 공유메모리를 선택
- 멀티플렉싱(epoll) 구조로 효율적인 쓰레드 관리
- 각 프로세스가 read, write memory(원형 큐)를 가져 독립적으로 운영.
- Full Semaphore : 읽을 데이터가 있는지 여부를 검사.
- Empty Semaphore : 쓸 공간이 있는지 여부를 검사.



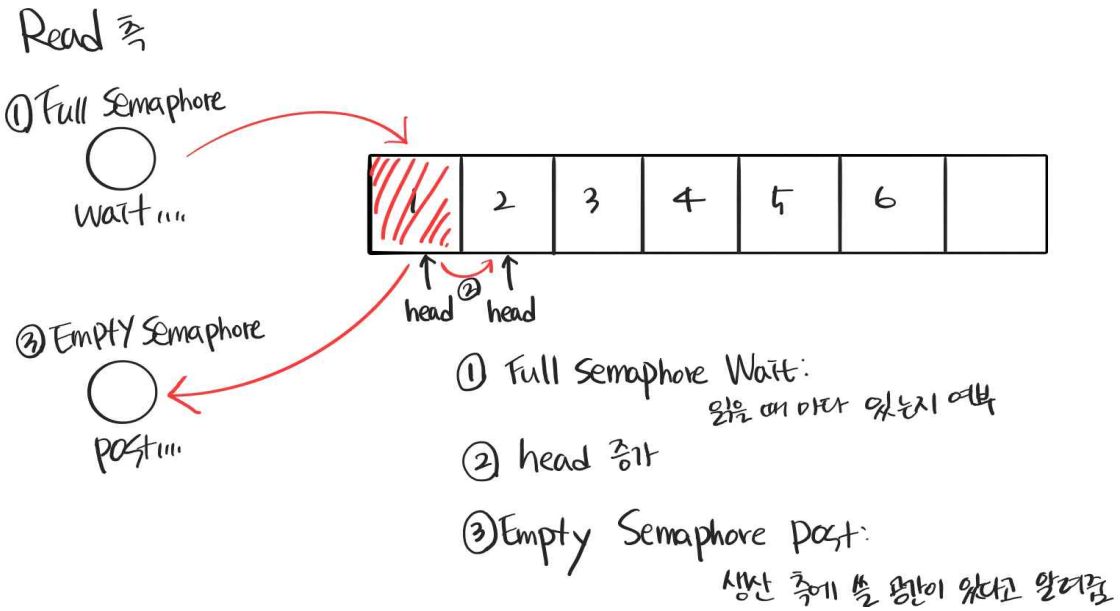
[그림 19] 공유 메모리 구조

■ 공유메모리에 데이터를 쓰는 상황



[그림 20] 공유 메모리 데이터 쓰기

■ 공유메모리에 데이터를 읽는 상황



[그림 21] 공유 메모리 데이터 읽기

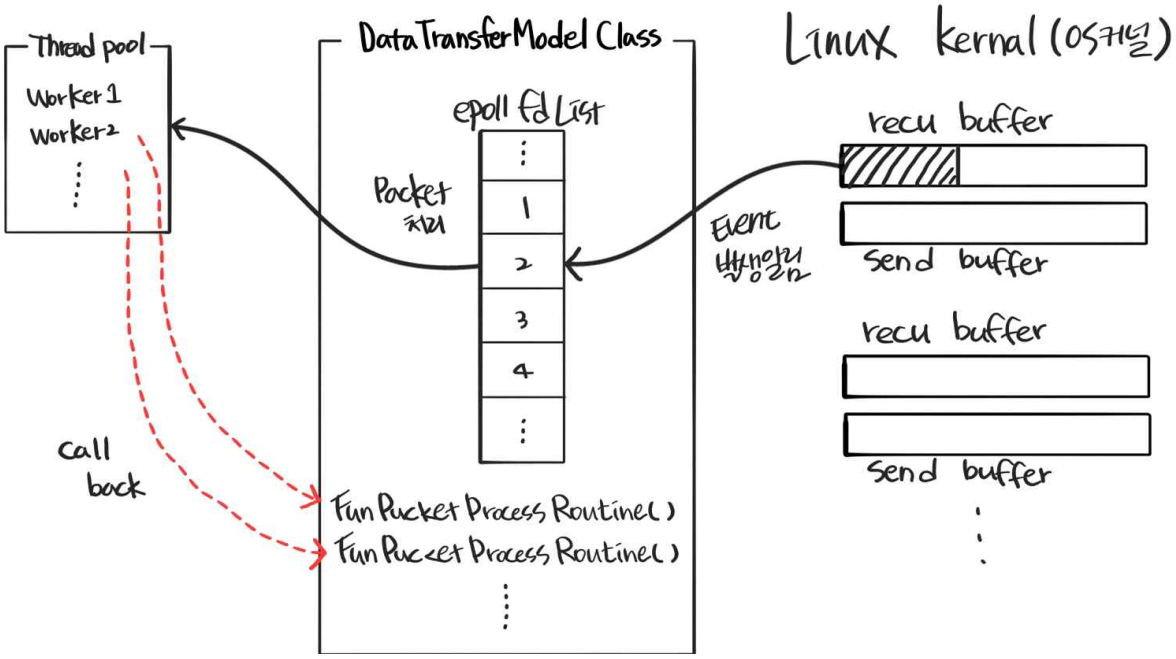
3.2.2 패킷 처리

■ I/O 멀티 플렉싱 구조

- 여러 개의 파일 디스크립터(fd, socket)를 감시하며, 어떤 fd에서 이벤트가 발생했는지 알 수 있다.
- 하나의 쓰레드를 가지고도 수천 수만개의 소켓을 처리할 수 있음.

■ Thread Pool

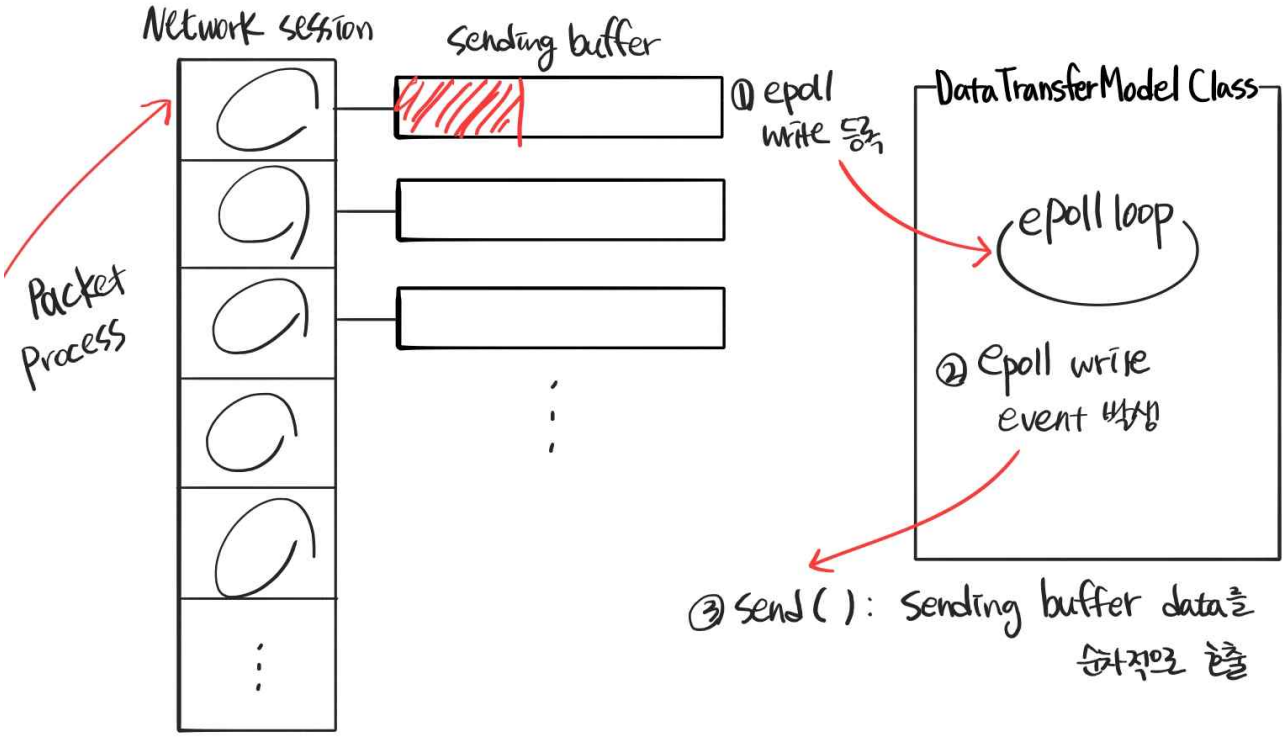
- 각 패킷마다 워커 쓰레드를 할당하여 독립적으로 처리될 수 있도록 쓰레드 풀 도입
- 이벤트를 받는 epoll loop와 packet 처리는 개별적으로 수행된다.



[그림 22] I/O 멀티 플렉싱과 패킷 처리 구조

■ Sending Queue

- tcp는 스트림 형식으로 보냄. 때문에 상대 수신 버퍼가 꽉 찼거나 보낼 수 없을 때 패킷이 잘려서 보내는 경우가 생길 수 있음. 따라서 네트워크 세션당 하나의 송신 큐를 둬서 다 보내지 못한 스트림 bytes를 저장하여 보낼 수 있을 때 먼저 보내게 된다.
- tcp의 경우 보낼 데이터를 sending queue에 먼저 넣고 해당 fd에 epoll write event를 등록. 실질적으로 데이터를 보낼 때는 epoll write event가 발생할 때 이뤄진다.



[그림 23] 전송 큐와 socket send의 관계

■ 엣지 트리거(Edge Trigger)

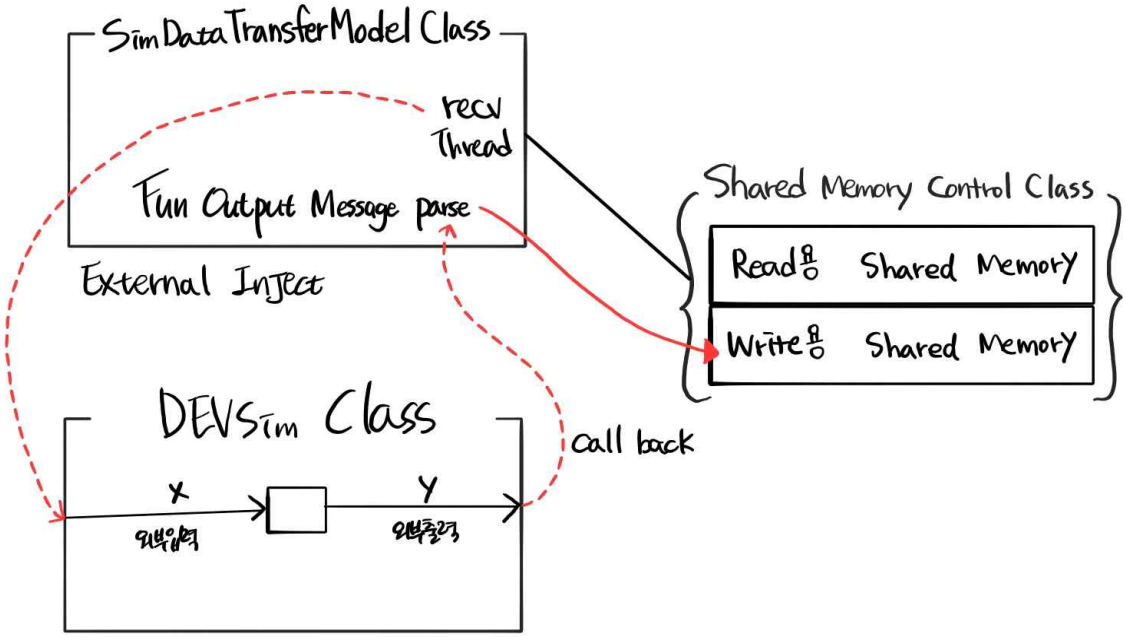
- 상태 변화가 일어날 때만 이벤트 알림이 오도록 설정하여 많은 이벤트가 동시에 수신되는 상황을 줄임. 예를 들면 수신 버퍼에 데이터가 도착한 딱 한 번만 이벤트가 발생한다.
- 위와 같은 상황이므로 한번 이벤트가 발생하면 수신버퍼를 모두 처리해서 다음 이벤트를 받을 수 있도록 while문으로 recv를 처리함.

4. 시뮬레이션 엔진 서버

4.1 구조

■ 환경

- 운영체제: linux (ubuntu 22.04)
- 사용언어: C++



[그림 24] 시뮬레이션 서버 구조

■ packet처리

- internal server와 공유메모리로 통신.
- DEVS simulator가 보내주는 출력메세지를 callback함수가 공유메모리에 전달한다.
- 출격명령, 미사일 발사 등 사용자가 실시간으로 주입해야 하는 이벤트 역시 공유메모리로 받아 시뮬레이터에 주입하여 반영한다. (External inject)

■ DEVS simulator

- 처음 프로그램을 실행시 DB서버에서 시나리오 정보(부대 정보)를 받아 모델을 생성하고 초기화한다.
- 각 모델당 processor가 생성되어 processor들끼리 상호작용하면서 시뮬레이션이 동작하게 된다.

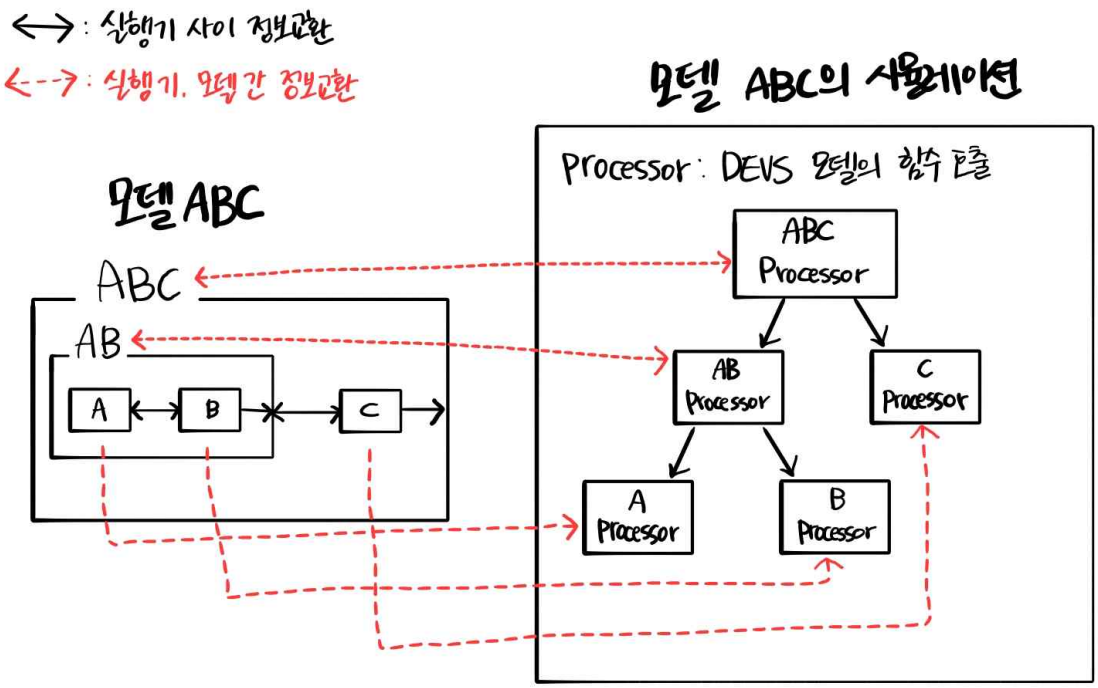
4.2 DEVS

4.2.1 DEVS 이론

■ DEVS(**Discrete Event System Specification**)

- 이산 사건 모델링에 대한 수학적이고 계층적인 명세로 1976~1984년 Bemard P. Zeigler 교수에 의해 고안.
- Modeling & Simulation 학술적 연구 및 국방, 교통, 재난 등 다양한 분야에서 실용적인 적용이 이루어지고 있다.

■ 모델과 시뮬레이션 알고리즘



[그림 25] DEVS 모델과 시뮬레이터 구조

- 모델과 시뮬레이션 알고리즘은 분리하여 구현된다.
- 원자 모델(atomic) : 분해가 불가능한 최소 단위 모델을 의미한다. (model A, B, C)
- 결합 모델(coupled): 원자 혹은 결합 모델들의 결합체를 의미한다. (model AB, ABC)
- devs 환경에서 모델링을 하는 사용자는 모델의 수학적인 정의에 따라 모델을 정의하게 된다.

■ 원자 DEVS 모델 수학적 표현

$M = <X, Y, Q, \delta_{ext}, \delta_{int}, ta, \lambda>$

- X: 입력 사건 집합
- Y: 출력 사건 집합
- Q: 상태 집합 $Q_t=\{(q,e)|q\in Q, 0 \leq e \leq ta(q)\}$
- δ Int: 내부 상태천이 함수 $Q_t \rightarrow Q$ 어떠한 상태에서 t시간이 흐르면 다음 상태를 출력
- δ Ext: 외부 상태천이 함수 $Q_t * X \rightarrow Q$ 어떠한 상태에서 t시간, 어떠한 사건(입력)을 매개로 다음 상태 q를 결정한다. 즉, 아직 $t \leq Ta$, 이고 새로운 사건x가 온다면 상태를 천이하게 된다.
- Ta: Q →T 시간 전진 함수. 현 상태에서 얼마나 머무를지 결정하는 함수
- λ: 출력 함수. Q * T →Y 내부 상태천이 발생 시(ta만큼 시간이 가면) 다음 출력을 결정한다.

■ 결합 DEVS 모델의 수학적 표현

$DN = <X, Y, M, EIC, EOC, IC, select>$

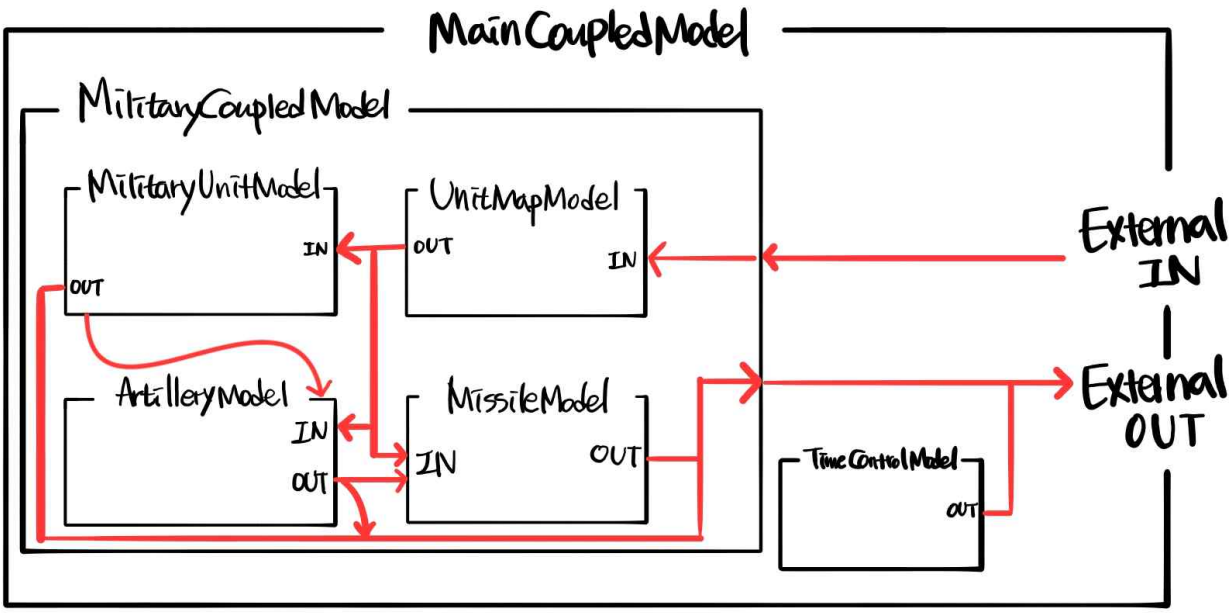
- X, Y: 입출력 사건 집합
- M: DN내부에 모든 하위 모델들의 집합
- EIC(External Input Coupling): 외부 입력 연결관계 EIC $\subseteq$DN.X * M.X
- EOC(External Output Coupling): 외부 출력 연결관계 EOC $\subseteq$M.Y * DN.Y
- IC(Internal Coupling): 내부 연결 관계IC $\subseteq M_i.X_i * M_j.X$ i번째 하위모델의 출력이j번째 하위 모델의 입력으로 들어간다는 뜻으로, 내부 연결을 정의 해놓은 집합이다.
- Select: $2^M-\Phi$ → M 모델들의 이벤트 발생의 우선순위를 결정하는 함수. 즉, M의 부분집합의 집합(멱집합)은 동시에 사건이 발생한 것이고, 그중 우선순위 모델을 출력하는 것이다.

4.2.2 DEVS 구현

■ 모델 실행기(processor)의 입출력 정의

- (x, t) : 외부 입력 x가 시각 t에 도착
- (y, t) : 외부 출력 y가 시각 t에 발생
- (*, t) : 타임 아웃(내부 상태 천이)가 시각 t에 발생
- (done, t_n) : 이벤트 처리가 완료되고 다음 이벤트 발생 시각은 tn

■ 시뮬레이션 서버 DEVS 모델 설계

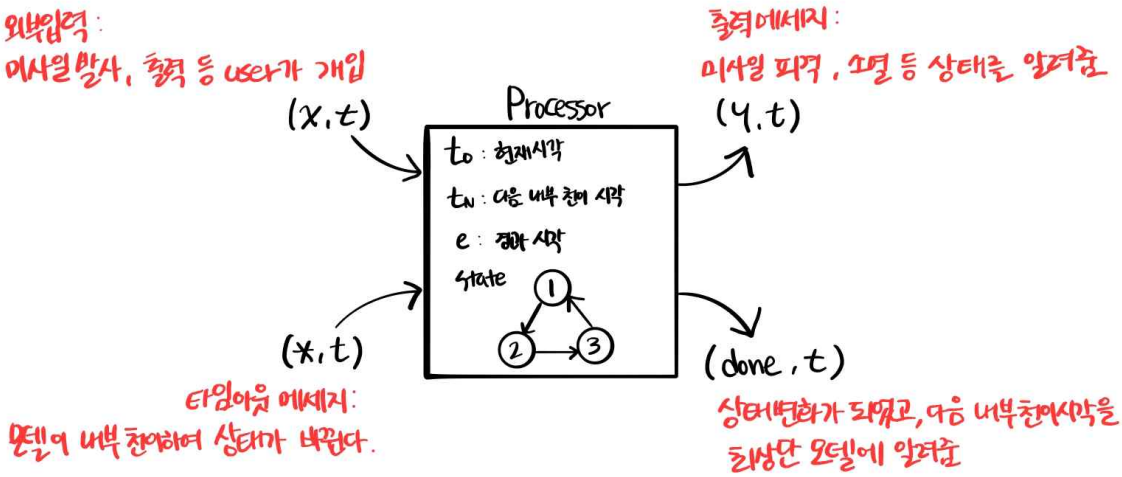


[그림 26] 시뮬레이션 서버 모델 설계도

- MainCoupledModel : 모든 모델의 최상단 결합모델. 실질적으로 이 모델에 도착하는 메시지와 이벤트 시간으로 시뮬레이션이 동작하게 된다.
- TimeControlModel : 시뮬레이션 시간을 관리하는 원자 모델 1초에 한 번씩 시간을 mainCoupledModel에 보내주는 역할을 한다.
- MilitaryCoupledModel : 자주포, 부대, 미사일 모델들을 결합한 모델. 하위 모델에서 발생한 이벤트들을 정리하여 mainModel에 보내준다.
- MilitaryUnitModel : 부대 객체의 상태를 나타내는 원자 모델
- ArtilleryModel : 자주포 객체의 상태를 나타내는 원자 모델
- MissileModel : 미사일 객체의 상태를 나타내는 원자 모델
- UnitMapModel : 모든 객체의 id를 map으로 저장하고 있으며, 보내진 (x, t), (*, t)를 알맞은 모델로 보내는 역할을 수행한다.

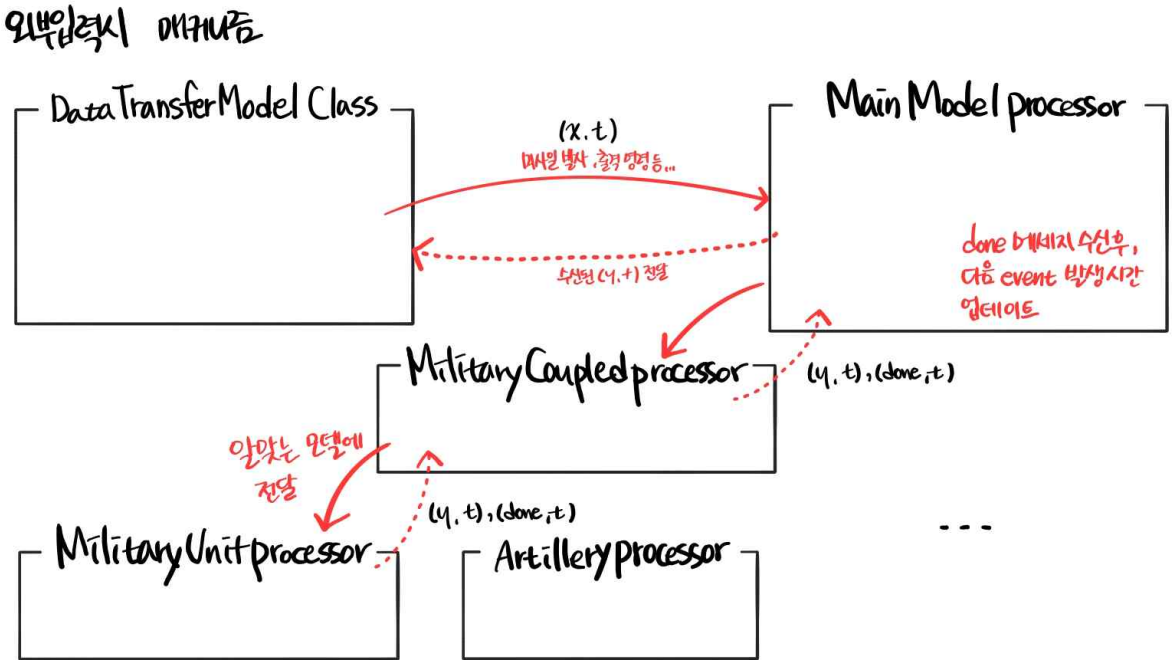
■ 모델 processor의 동작

- model과 1대1 대응되는 processor들이 아래와 같은 기준으로 동작.
- 계층적인 구조를 가져 서로 상호작용 하면서 시뮬레이션이 흘러가게 된다.



[그림 27] 모델 실행기의 동작

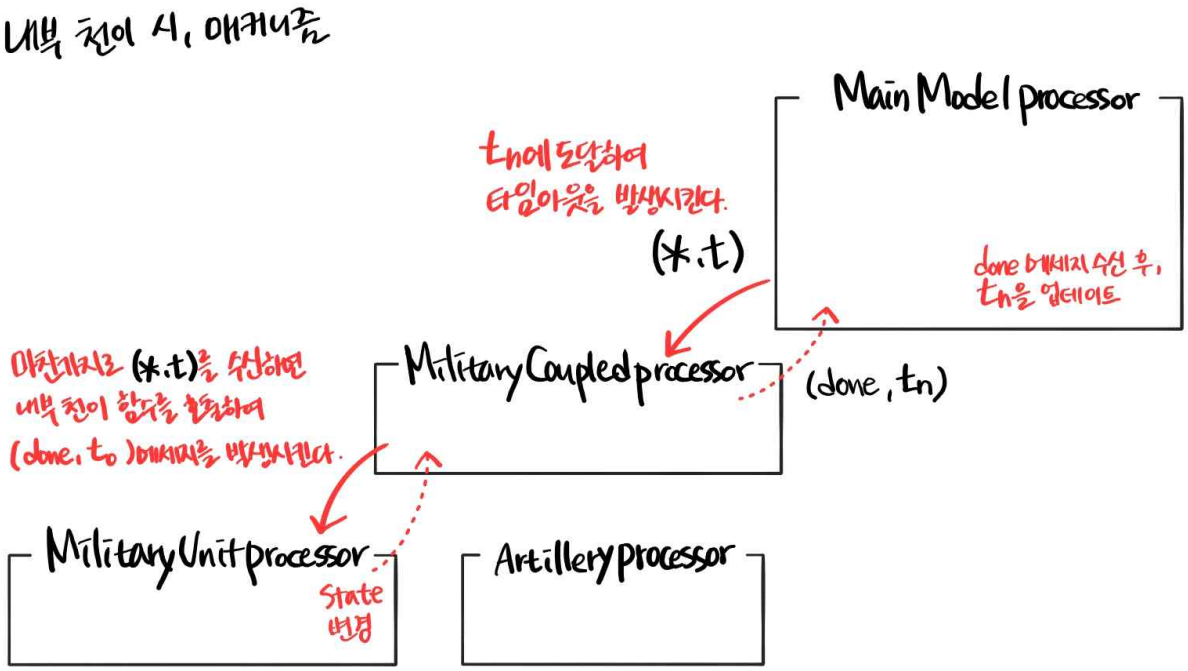
■ 외부 이벤트가 발생했을 때



[그림 28] 외부 이벤트 발생 시 DES의 동작

- 외부 이벤트(x, t)가 클라이언트로부터 도착한다면 모델 프로세서들이 계층적으로 메시지를 주고받으면서 각 모델의 상태를 업데이트한다.
- 최상단의 메인프로세서가 모든 출력 메시지(y, t)를 받고 SimDataTrsfModel class의 call back 함수를 호출하여 클라이언트로 패킷을 보내게 된다.

■ 내부 타임아웃이 발생하여 내부 상태 변화가 일어날 경우



[그림 29] 내부 타임아웃 발생 시 DES의 동작

- MainModel에 현재 시뮬레이션 시각이 다음 시뮬레이션 시각 Tn과 같아진다면, 타임아웃 이벤트(*, t)가 발생한다. 그럼 mainModel은 하위 모델들에게 메시지를 전달하게 되고, 모델별로 다음 이벤트 시각이 (*, t)과 같다면(같거나 작다면) 내부 상태천이 함수를 호출하여 모델의 상태를 바꾸게 된다.
- 내부 상태천이를 하면서 발생한 다음 이벤트 발생 시각(다음 내부 상태천이가 일어날 시각)을 상위 모델 실행기에 전달하면서 다음 타임아웃 시각을 업데이트 하게 된다.
- 여기서 수신된 여러 개의 Tn 중에서 가장 적은 값으로 업데이트하여 다음 (*, t)메세지를 보낼지 결정하게 된다.

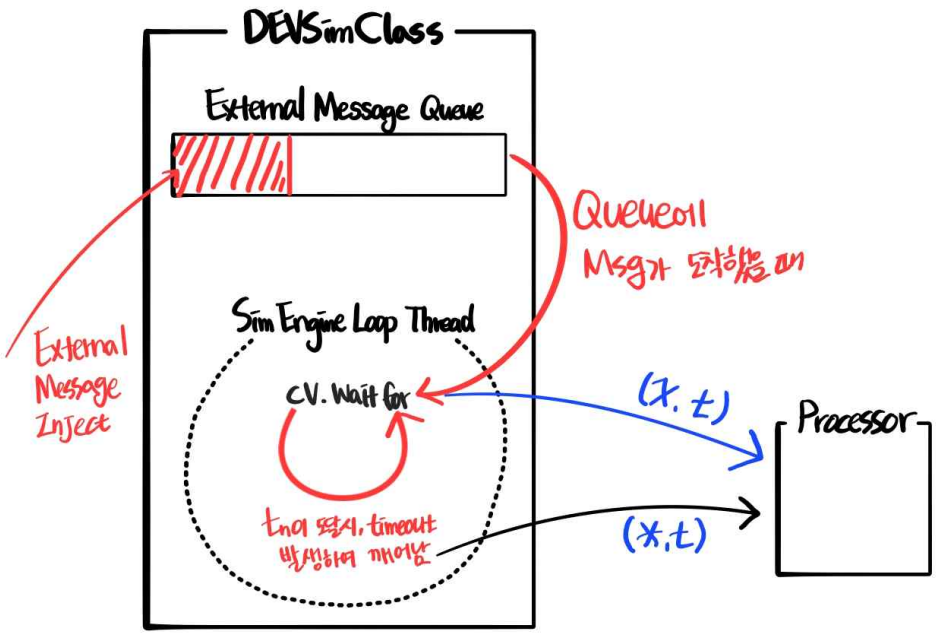
4.2.3 이산 사건 시스템의 실시간 연동

■ 이산 사건 시스템과 시간

- 이산 사건 모델 시뮬레이션은 시간이 경과함에 따라 시뮬레이션이 진행되는 연속시간 시뮬레이션과 다르게 시스템 외부 혹은 내부에서 사건(event)가 발생했을 때 모델을 실행시키게 된다.
- 따라서 초기 모델을 정의하고 실행한다면 순식간에 시뮬레이션이 구동되어 결과를 얻을 수 있다.
- 하지만 이 프로젝트에서 필요한 부분은 시뮬레이션이 흘러감에 따라 외부에서 미사일 발사, 출격 명령을 내리는 등 외부 입력이 실시간으로 주입되기에 실제 시간과 시뮬레이션 상 시간이 일치(혹은 배속으로)시켜야 할 필요성이 있다.

■ Conditional_Variable

- 시간 동기화를 위해서 conditional_variable을 사용하였다.
- ExternalMessageQueue에 메시지(x, t)가 들어올 때 통지하여 깨어나도록 하여 외부 입력에 대한 시간 동기화를 할 수 있다.
- 또한 conditional_variable의 wait할 수 있는 시간은 아무런 입력이 들어오지 않을 경우 다음 이벤트(Tn) 시각에 깨어나도록 timeout을 설정하여 내부 이벤트 (*, t)를 발생 시키는 방식으로 실시간 시간 연동을 처리하였다.



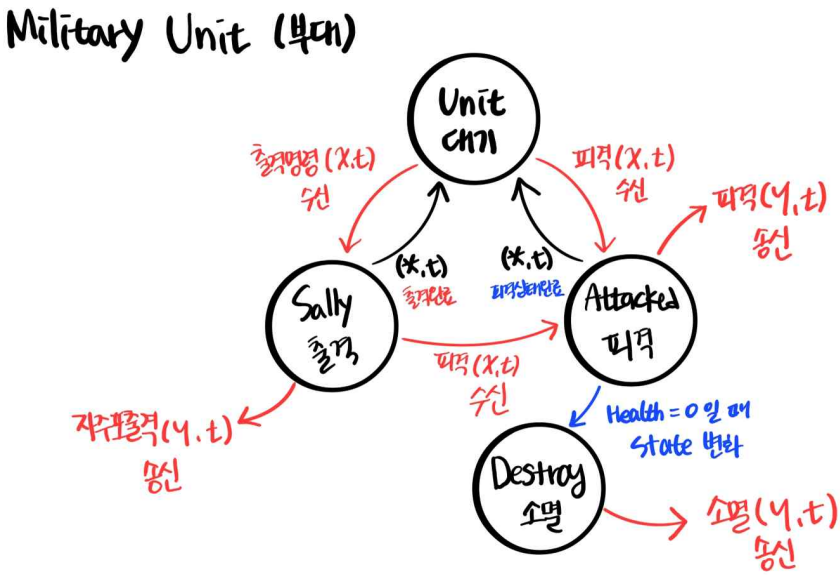
[그림 30] 이산 사건 시스템의 실시간 연동

4.2.4 모델 상태전이

■ 모델의 모의 논리

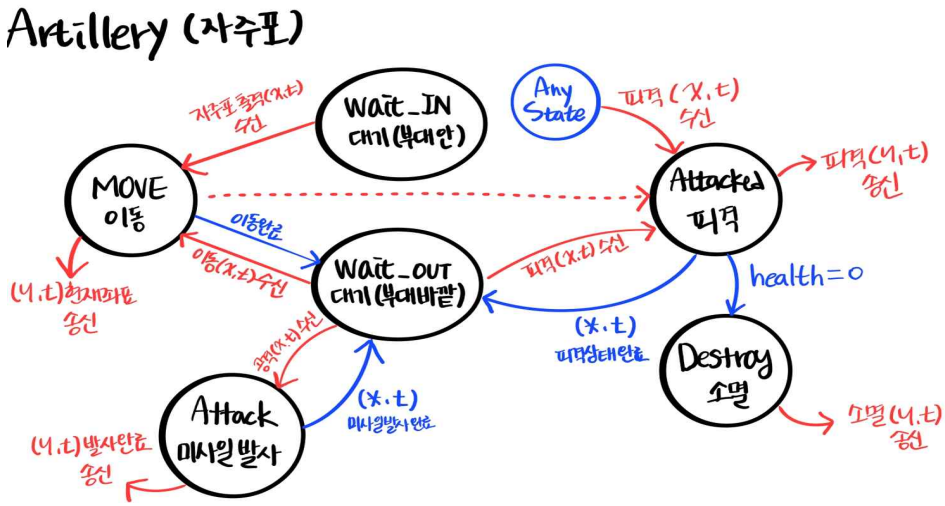
- M&S DEVS환경에서 모델의 상태나 변수 정의는 해당 분야 도메인 전문가가 주로 담당하게 되는데, 해당 모델의 상태전이나 변수 설정에 따라 시뮬레이션의 정확도가 달라지게 된다.
- 예를 들어 미사일 발사 시 얼마나 시간이 소요되고 명령을 수행 중에 다른 명령이 들어오면 실제 상황에서는 어떤 식으로 행동하는 매뉴얼이 있는지 등 해당 분야에 심도 있는 지식이 필요하기에 이 프로그램에서는 간단하게 모델 상태전이와 변수를 설정하였다.

■ Military Unit atomic Model 상태전이



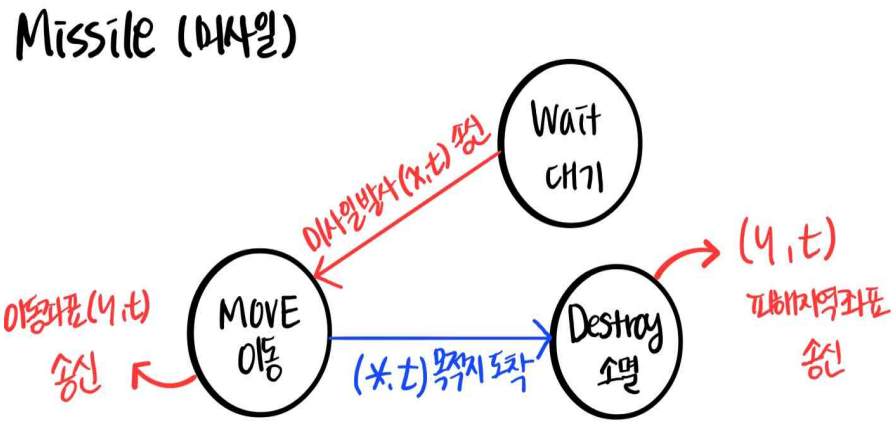
[그림 31] 부대 유닛의 상태전이 그림

■ Artillery atomic Model 상태전이



[그림 32] 자주포 유닛의 상태전이 그림

■ Missile atomic Model 상태전이



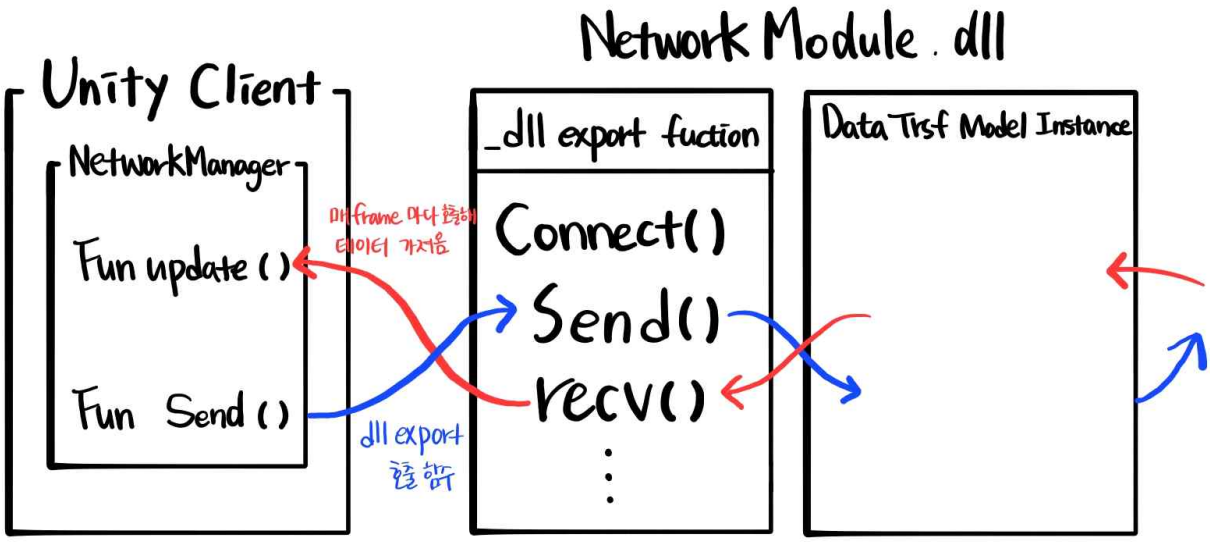
[그림 33] 미사일 유닛의 상태전이 그림

5. 클라이언트

5.1 구조

■ 환경

- 운영체제 : windows os
- 사용언어 : C# (Unity), C++(NetworkModule.dll)



[그림 34] 클라이언트 전체 구조도

■ 유니티 클라이언트

- NetworkManager class에서 매 프레임 update() 호출 시 네트워크 모듈 내부의 recv queue에서 수신된 데이터를 전부 꺼내온다.
- 패킷 송신 시 라이브러리에 dllexport된 함수를 호출하여 네트워크 모듈에 넘겨준다. 여기서 필요한 최소한의 값만 넘겨주고 패킷을 구성하는 역할은 모두 네트워크 모듈이 담당하게 된다.

■ 네트워크 모듈 (dynamic linking library)

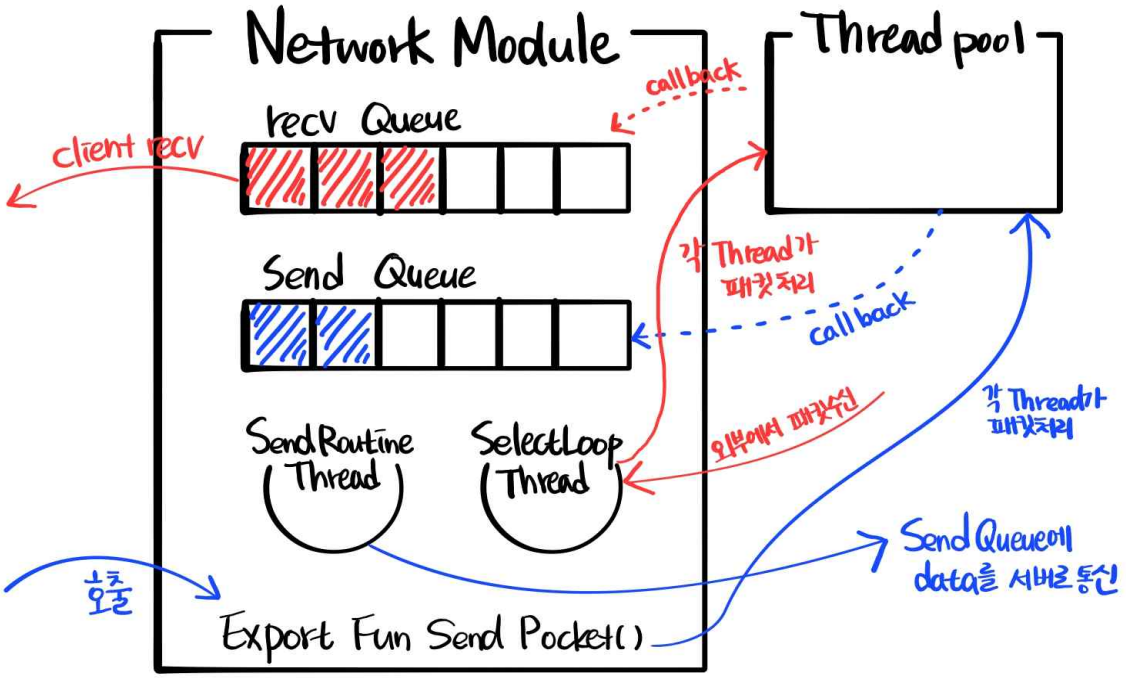
- 유니티는 싱글 쓰레드로 동작하기 때문에 네트워크 패킷을 직접 처리하기엔 부족하다. 따라서 별도로 네트워크를 처리하는 모듈이 필요하다.
- 동적 링킹 라이브러리 : Runtime에 동적으로 라이브러리를 불러오도록 dll파일로 작성하였다. 여기서는 클라이언트 프로그램이 하나로 뭉쳐져 있지만, 실제 시뮬레이터에서는 모의 통제 도구, DB 통제 도구, 전장 상황도 등 여러 개로 분리되어 있기 때문에 여러 클라이언트 들이 메모리에 올라온 dll 하나로 네트워크 통신을 하는 것이 메모리 효율성에 좋다.

5.2 네트워크 모듈

5.2.1 상세 동작

■ packet process

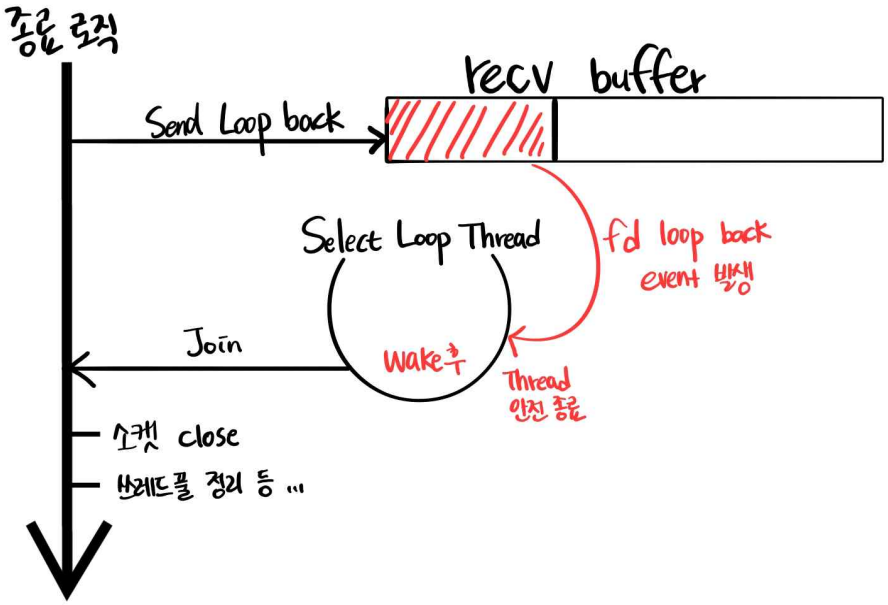
- select 방식으로 이벤트가 수신될 때만 동작하기에 수신 쓰레드의 cpu점유율을 낮출 수 있다.
- 데이터 전달 처리와 마찬가지로 Thread Pool을 도입해 패킷 수신 송신 하나하나를 각 쓰레드가 처리하게 하였으며, 전송 큐를 둬서 서버 네트워크 상황에 대응할 수 있도록 구성하였다.



[그림 35] 네트워크 모듈 패킷 처리 과정

■ loopback socket

- select loop thread는 이벤트가 발생할 때까지 blocking된 상태이다. 그렇기에 만약 프로그램이 종료된다고 할 때 모든 쓰레드를 join하고 정리 후에 메인 쓰레드가 종료되는 것이 이상적이라고 할 수 있는데, join을 한다고 해도 자고 있는 쓰레드를 깨울 수는 없기 때문에 종료시에 강제로 select이벤트를 발생시켜 thread를 종료시켜야 한다.
- 따라서 종료시에 select이벤트를 발생시키기 위해 loopback socket을 만들어 FD_SET에 등록하게 되고, 종료 시그널이 발생할 때 패킷을 전달하여 select blocking을 해제할 수 있다.



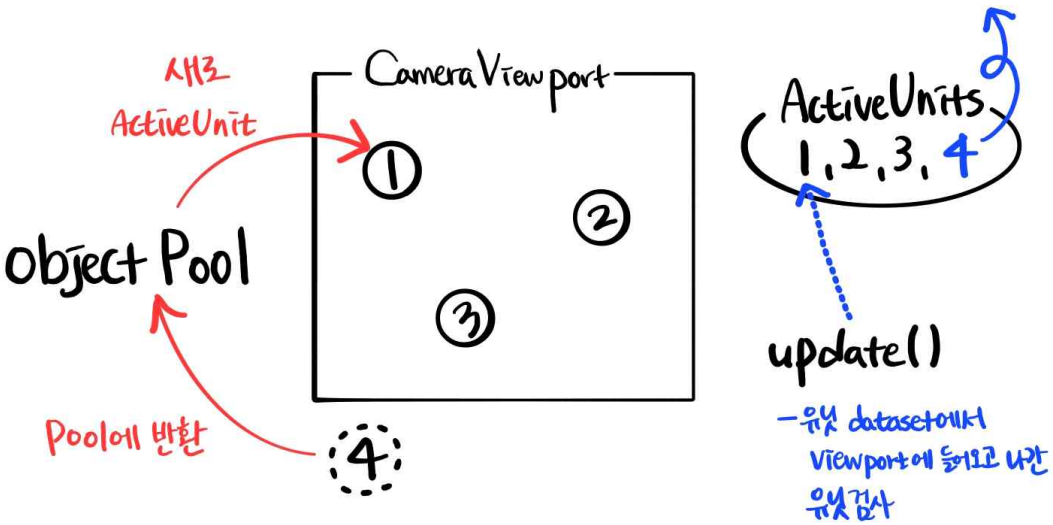
[그림 36] Loopback socket 이벤트 발생 과정

5.3 클라이언트 구현

5.3.1 렌더링 최적화

■ 화면 객체 최적화

- 모든 유닛은 데이터로 패킷을 받으면 화면에 업데이트하는 것이 아닌 유닛 데이터 셋을 업데이트한다.
- 매 frame update()마다 현재 화면에 존재하는 activeUnits 셋과 카메라 viewport에 걸리는 유닛들을 비교하여 필요하면 오브젝트 풀에서 GameObject를 받아 렌더링하고, 없다면 풀에 다시 반환하는 방식으로 처리된다.

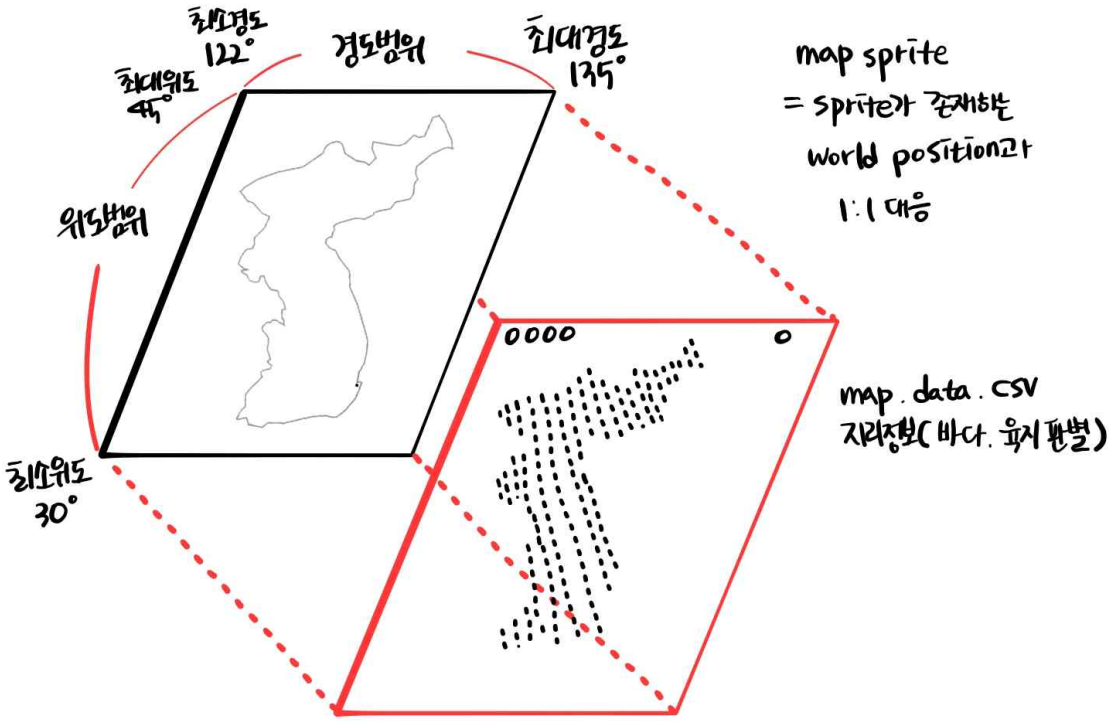


[그림 37] 클라이언트 object 렌더링 과정

5.3.2 위경도 지도 매핑

■ 지도 화면 매핑

- 여기서는 지구가 둥글다는 사실을 배제하고 선형 매핑(Linear Mapping)방식으로 유니티 월드 좌표와 위도 경도를 1대1 매핑 하였다.
- map sprite에 크기와 같은 csv 파일을 작성하여 지리적 정보(1이면 육지, 0이면 바다)를 기록하고 로그인 직후 로딩중에 불러온다.
- 산과 평지를 구분하기 위해 고도화된 csv파일을 만들 수도 있으나 여기서는 단순히 0과 1로 바다와 육지만 구분하였다.



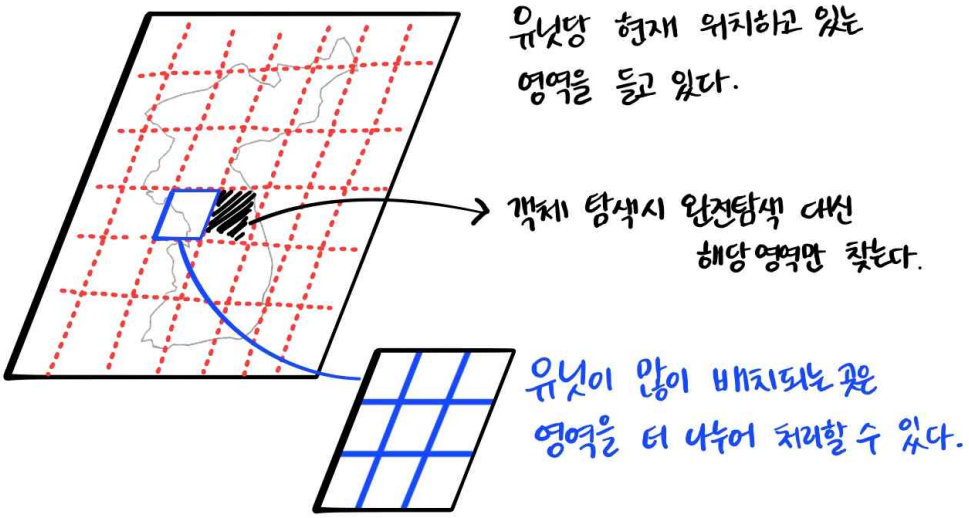
[그림 38] 맵 스프라이트와 위경도, 지리정보 매핑

6. 결론

6.1 보완할 점

■ 객체 탐색 시간

- 시뮬레이션 서버에서 미사일이 떨어지면 그 좌표를 (y, t)출력 해줍니다. 여기서 지금은 모든 객체를 탐색하며 그 좌표의 미사일 피격 범위에 걸리는 객체를 찾아 피격 이벤트를 (x, t) 발생시키는 구조입니다. 탐색 시간 객체 수가 N개일 때 O(N)
- 좌표 범위를 여러 개의 범위로 쪼개서 영역을 저장하고 미사일이 떨어진 영역의 범위만 검사하는 방식으로 바꾸면 훨씬 효율적인 탐색이 가능할 것입니다. 이 방식으로 바꿨을 때 영역을 A개로 나눴고 영역을 찾는데 O(logA) (map 자료구조에 저장했을 경우) 각 데이터가 영역에 균등 분배되어 있을 때 O(logA*N/A) 시간이 걸릴 것이고, 최악의 경우 O(logA*N), 최선의 경우 O(logA) 이 될 것입니다.
- 최악의 경우를 피하려면 주로 유닛이 자주 배치되는 대도시(서울)나 시나리오상 중요한 전략적 요충지에 계층구조(트리)로 2차 3차 영역을 배치한다면 문제가 조금 나아지지 않을까 라는 생각도 듭니다.



[그림 39] 객체 탐색의 방안

■ 최적의 경로로 이동

- 현재는 출격 명령을 내리면 유닛이 해당 좌표에 직선으로 이동합니다. 물론 지금은 육지는 모두 1이기 때문에(이동하는 cost가 모두 같기에) 바다를 제외하고 육지로 이동한다면 이것이 최적의 동선일 것입니다만 실제 map은 산과 강, 도로 등 타일마다 cost가 다릅니다. 때문에 출격 명령이 도착하면 미리 해당 좌표에 최적의 경로를 계산하고 그 동선으로 이동하는 로직이 필요하다고 볼 수 있겠습니다.

■ 여러 기능의 부재

- 시뮬레이션 중단, 배속 기능 등 시뮬레이션 엔진에 기능이 모두 다 구현되지 않았습니다.
- 시뮬레이션이 시작되고 몇 분에 한 번씩 현재 시뮬레이션 상태(객체의 상황 등)를 데이터베이스에 저장하여 문제가 생겼을 때 복구가 가능한 기능이 필요할 것입니다.
- 데이터 전달 처리 서버에서 현재는 요청한 클라이언트에게만 전달되는 상태입니다. 그러나 이 유닛이 보이는 가시거리에 있는 모든 클라이언트에게 자주포 유닛, 미사일 유닛의 좌표를 보여줘야 하기에 이 부분이 구현되지 못한 점이 아쉽다고 느낍니다. 좀 더 모델링에 신경 써서 가시거리까지 되었다면 좋았을 것입니다.

■ 자주포 외의 유닛의 부재

- 여기선 단순히 움직이고 공격할 수 있는 유닛이 자주포 하나에 불과합니다.
- 더 섬세하고 완성도 높은 모의 논리를 통해 다양한 유닛을 모델링하고 다양한 상황에 대응되는 실제와 비슷한 시뮬레이션을 하기 위해서 갈 길이 멉니다.

6.2 참고 문헌 및 미디어

-1994 "DEVSim++ : C++ Environment for Modeling/Simulation of DEVS Models" 한국시뮬레이션학회
저자 : 김탁곤/Tag-Gon Kim

-2013 “복합체계 개념에 기반한 국방체계 모델링 시뮬레이션 방법론” 한국과학기술원(KAIST)
저자 : 김탁곤 /Tag-Gon Kim 1, 권세중2, 강봉구/Bong Gu Kang 3

-「KAIST 온라인 공개 강좌」 시스템 모델링 시뮬레이션 개론 김탁곤 교수
url : https://www.youtube.com/watch?v=DeTJUBa84IY