**컴퓨터공학특론2 개인 프로젝트 보고서**

**60182203 이하윤**

**1. 서론  
강의 실습시간, 과제 등에서 공부해온 소켓 프로그래밍, producer-consumer 구조 등을 최대한 이용하기 위해서 실습 등에서 사용한 github의 dkmoon/mju2022backend6 저장소의 코드를 기본적인 틀로 잡고 개발하였다.   
프로젝트를 설계할 때 코드가 길어지고 복잡해질 것으로 생각했기 때문에 코드를 헤더 파일과 소스 파일로 분리하고 각 기능별로 class 혹은 namespace로 구분하여 정리했다.  
Visual Studio 2019(v142)와 ISO C++ 14 Standard에서 개발하였다.**

**2. Server  
2.1. 헤더파일 (.h)  
- class Client  
서버와 연결된 유저 클라이언트를 나타내는 클래스이다. Client 클래스를 설계할 때 게임 로직에서 사용하는 데이터들(ex. Redis에 저장되는 데이터들)을 최대한 저장하지 않도록 하였고, 기본 데이터 값, 공격 범위 등 클라이언트에게만 적용되는 고정된 데이터들은 심볼릭 상수로 선언하였다. 몬스터에게 공격받았을 때의 로직을 처리하는 OnAttack 함수가 있다.  
- namespace Json  
JSON 문자열 관련 변수, 함수들이 정의된 네임스페이스이다. JSON 문자열은 서버, 클라이언트 모두 동일한 형식, 구조를 사용하도록 설계하였다. JSON 문자열의 첫번째 키는 TEXT, 이후 키들은 PARAM1, PARAM2이다. 서버에서 클라이언트로 보내는 메시지의 타입을 나타내는 enum class M\_Type이 존재한다. 메시지 타입은 유저가 체력이 다해 죽었음을 나타내는 E\_DIE, 다른 유저가 중복 로그인했음을 알리는 E\_DUP\_CONNECTION, 유저가 로그인에 성공했음을 알리는 E\_LOGIN\_SUCCESS, 다른 유저로부터 온 귓속말임을 나타내는 E\_CHAT으로 구성된다. 메시지 타입을 명시해야하는 메시지의 경우 JSON 문자열의 두번째 키인 PARAM1의 value로 제공된다. JSON 문자열을 만들기 위해 필요한 심볼릭 상수들이 존재하고, 특정 상황, 받은 명령어에 대한 응답을 위한 메시지를 JSON 문자열 형식으로 변환하여 반환하는 Get~Json 함수들이 존재한다. Logic namespace의 Process~ 함수들이 메시지를 반환할 때와 같이 JSON 문자열이 필요할 때 이 함수들을 호출한다.  
- namespace Redis  
Redis 관련 변수, 함수들이 정의된 네임스페이스이다. hiredis를 위한 redisContext 변수와 이에 대한 mutex가 존재하고, Redis의 명령어를 구성할 때 필요한 심볼릭 상수들이 존재한다. Redis에 저장하는 게임 데이터들에 대한 flushall, get~, set~, expire, persist 명령어를 hiredis로 구현한 함수들과 서버와 연결된 유저의 로그인, 등록 과정을 수행하는 RegisterUser 함수가 존재한다. 클라이언트는 서버와 연결되면 로그인을 수행하고, Redis에 등록되고 부터는 Redis namespace의 함수들을 통해 유저 본인의 데이터들에 접근한다.  
- class Slime  
슬라임 몬스터를 나타내는 클래스이다. 슬라임 클래스에는 게임 로직에서 사용할 데이터들을 저장하도록 설계하였다. 또한 몇번째 슬라임인지 나타내는 static 인덱스 변수와 고정된 기본값들을 위한 심볼릭 상수가 존재한다. Client 클래스와 마찬가지로 유저에게 공격받았을 때의 로직을 처리하는 OnAttack 함수가 있다.  
- namespace Server  
서버 구동 관련 변수, 함수 등이 정의된 네임스페이스이다. 주로 여러 위치에서 사용하는 서버 주소, 서버 포트, 연결된 클라이언트 맵, 작업 큐 등 서버 관련 변수, 함수가 존재한다. TerminateRemainUser 함수를 통해 파라미터로 전달한 ID와 동일한 ID를 가지는 클라이언트 들을 모두 연결해제 시키는 기능이 있다. 이 함수는 중복 로그인 문제가 발생했을 때 사용할 수 있다.  
- namespace Logic  
전반적인 게임 로직에 관련된 변수, 함수들이 정의된 네임스페이스이다. 게임 로직에 필요한 변수들과 각 명령어를 처리하는 Process~ 함수들, 슬라임을 생성하는 SpawnSlime 함수, Str 포션의 효과를 등록하는 StrPotion 함수, 슬라임의 공격을 체크하는 SlimeAttackCheck 함수 등이 존재한다. Process~ 함수들은 Handler와 Handler 맵을 통해 클라이언트에게 받은 명령어로 바로 등록된 명령어 처리 함수가 실행된다. 각 Process~ 함수는 명령어에 따른 적절한 액션을 수행하고 클라이언트가 기대하는 JSON 메시지를 반환하는 것으로 기능을 모두 수행한다.  
shouldSendPackets 맵을 통해 각 클라이언트의 소켓 key에 문자열들을 저장한다. 이 맵을 통해 서버로 보내야하는 메시지를 예약하고 main 함수에서 클라이언트의 send, recv가 끝나 아무것도 수행하지 않을 때 이 맵에 예약된 메시지를 보내도록 설정한다. SpawnSlime을 통해 슬라임을 생성할 때 SlimeAttackCheck 함수를 실행함수로 하는 쓰레드를 슬라임 객체를 파라미터로 넘겨줘 실행하여 공격을 주기마다 체크한다.  
- namespace Rand  
게임 로직에서 사용할 난수 관련 변수, 함수들이 정의된 네임스페이스이다. uniform distribution과 Mersenne\_twister\_engine을 이용해 난수를 생성한다.** **로직에서 사용하는 여러 난수 범위를 나타내는 uniform\_int\_distribution 변수를 생성해두고, GetRandom~ 함수들을 통해 필요한 범위의 난수를 반환받는 방식으로 사용한다.**

**2.2. 소스파일 (.cpp)  
소스 파일에는 기초적인 소켓 프로그래밍과 producer-consumer 구조의 뼈대가 되는createPassiveSocket(), processClient(), workerThreadProc(), main() 함수들이 존재한다.  
- processClient()  
processClient 함수를 설계할 때 중요하게 생각한 점은 Worker Thread를 통해 blocking되지 않도록 한번씩 작업을 처리하면서 데이터의 recv, send를 처리하는 것이다. 기존의 recv를 처리하는 코드의 구조를 차용해 recv 후 필요한 작업까지 모두 처리가 됐을 때 다시 클라이언트로 보내야하는 데이터가 있다면 Client::sendTurn 변수를 true로 세팅하고 main 함수에서 select를 통한 읽기 이벤트가 없더라도 작업 큐에 넣어 다시 recv와 동일하게 Worker Thread가 send를 수행하도록 했다.   
- workerThreadProc()  
작업 큐에서 작업을 빼고 worker thread의 processClient 함수를 통해 작업을 처리한다. 오류 등으로 작업 처리가 실패하면 등록된 예약 메시지를 제거하고 해당 클라이언트를 삭제시키고 소켓을 닫는다.  
- SlimeGenerateThread()  
1분마다 슬라임의 수가 최대 마리 수만큼 존재하도록 생성하는 함수이다. main 함수에서 생성한 쓰레드가 실행하는 함수이다.  
- main()  
프로그램 초기화, 소켓 연결 관리, producer의 역할 등을 수행한다. 기존 역할에 추가적으로 남아있는 active socket들에 대해서 set에 넣어줄 때 작업 중이 아니고 보내야하는 메시지가 존재한다면 그 클라이언트가 메시지를 보내도록 설정한다.**

**3. Client  
클라이언트 측은 send, recv를 할 때 모든 데이터를 보내고 받을 때까지 계속 잡고 있는 blocking 방식으로 구현하였다. 이때 main 쓰레드에선 명령어를 입력받고 send를 보내고, 새로운 쓰레드를 생성하여 recv를 전담하여 받도록 구현하였다.  
3.1. 헤더파일 (.h)  
- namespace Client  
- namespace Logic  
- namespace Json  
- namespace Bot  
- namespace Rand**

**3.2. 소스파일 (.cpp)  
- RecvThreadProc()  
- main()**