**네트워크 게임프로그래밍**

Project Progress Report

**2019184004 김성준**

**2021180006 김진선**

**2021184016 백하빈**

목차

* 어플리케이션 기획
  + 게임 특징
  + 개발환경
* High level Design
  + 전체 흐름도
  + 로그인 흐름도
  + 인 게임 흐름도
* Low Level Design
  + 상수
  + client low-level
  + server low-level
  + 스레드
  + -패킷
* 팀원 별 역할분담
* 개발일정

* 노랑: 수정 사항
* 빨강: 삭제된 사항

1. **어플리케이션 기획**

**-** 소개

* 1. 게임 개요
     1. 개발 소스 : 컴퓨터 그래픽스 텀프로젝트(김진선)
     2. 장르: 3d 그래픽 pvp
     3. 게임컨셉 : 서로 탱크로 공격하여 먼저 죽이는 사람이 이기는 게임
     4. 플레이어 수: 2인
  2. 게임 요소
     1. 게임 진행 : 상대의 hp를 0으로 만들 시 승리
     2. 조작법
        + - w/a/s/d : 이동
          - f: Bullet 재장전
          - space bar: 총알 발사
          - h: 게임시작
          - r: 레디
          - 1: 탑뷰/1인칭 변경
          - 마우스: 카메라 움직임

우클릭: 카메라 인칭 변경

* + 1. 아이템: 맵에 놓인 아이템을 먹을 시 특수효과 발동   
       (자신에게 도움이 되거나, 상대를 방해)
       - * hp회복

줄어든 체력을 일부 회복

체력이 최대치일 시 적용되지 않음

* + - * + 속도 증가

본인의 이동 속도 증가

* + - * + 상대 freeze

상대방 움직임 일정시간동안 제안

아이템 획득 후 처음 발포 되는 총알이 얼음 총알로 발포 되며, 해당 총알 피격시 상대방 이동 속도 감소

1. **High Level Design**

1)전체 흐름도

텍스트, 친필, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

2)로그인 흐름도

텍스트, 친필, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

3)인게임 흐름도

텍스트, 친필, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

1. **Low Level Design**

1) 상수

enum DATA\_TYPE { SEND, RECV };

-작업자스레드에서 처리할 데이터를 구분짓기 위해서 사용

=>삭제, send,recv는 각각의 스레드에서 처리하기때문에 enum구조체 필요없음

enum COMP\_TYPE { OP\_ATTACK, OP\_STATE\_CHANGE, OP\_MOVE };

-패킷처리를 할때 어떤 패킷의 내용을 처리할지 구분하기위해서 사용

=> protocol 구조체에서 packet type으로 어떤 명령인지 구분하기 때문에 필요없음

enum ITEM { HEAL, SPEEDUP, FREEZE };

- 어떤 아이템을 처리할지 구분하기위해 사용

constexpr int MAX\_USER = 2;

-동접자 수

constexpr int NAME\_SIZE = 20;

-아이디

constexpr char CS\_LOGIN = 0;

constexpr char CS\_MOVE = 1;

constexpr char CS\_ATTACK = 2;

constexpr char CS\_ITEM = 3;

constexpr char CS\_LOGOUT = 4;

=>수정

constexpr char CS\_LOGIN = 0;

constexpr char CS\_READY = 1;

constexpr char CS\_MOVE = 2;

constexpr char CS\_ATTACK = 3;

constexpr char CS\_ITEM = 4;

constexpr char CS\_LOGOUT = 5;

constexpr char CS\_YAW = 6;

constexpr char CS\_RELOAD = 7;

constexpr char CS\_HIT = 8

constexpr char SC\_LOGIN\_INFO = 0;

constexpr char SC\_LOGIN\_OK = 1;

constexpr char SC\_LOGIIN\_FAIL = 2;

constexpr char SC\_MOVE\_OBJECT = 3;

constexpr char SC\_DIE\_OBJECT = 4;

constexpr char SC\_STATE\_CHANGE = 5;

constexpr char SC\_ATTACK = 6;

constexpr char SC\_LOGOUT = 7;

constexpr char SC\_GAMESTART = 8;

constexpr char SC\_PLAYING\_TIME = 9;

constexpr char SC\_RELOAD\_GUN=10;

=>수정

constexpr char SC\_LOGIN\_INFO = 0;

constexpr char SC\_LOGIN\_OK = 1;

constexpr char SC\_LOGIIN\_FAIL = 2;

constexpr char SC\_READY\_OK = 3;

constexpr char SC\_MOVE\_OBJECT = 4;

constexpr char SC\_DIE\_OBJECT = 5;

constexpr char SC\_STATE\_CHANGE = 6;

constexpr char SC\_ATTACK = 7;

constexpr char SC\_LOGOUT = 8;

constexpr char SC\_GAMESTART = 9;

constexpr char SC\_PLAYING\_TIME = 10;

constexpr char SC\_UPDATE = 11;

constexpr char SC\_SET\_ITEM = 12;

constexpr char SC\_ALL\_ITEM\_SET = 13;

constexpr char SC\_RELOAD = 14;

2) client low level

텍스트, 도표, 평행, 라인이(가) 표시된 사진

자동 생성된 설명

이전에 기획한 클라이언트 low level

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

최종 클라이언트 low level

client에서 socket의 생성 및 초기화는 메인에서 진행한다.

필요 함수 : void sendPakcet() : 서버로 패킷을 전송한다.

      void RecvPacket() : 클라이언트에서 패킷을 전송한다.’

=> 삭제, 멤버함수로 처리하지 않고 main에서 recv스레드를 할당하여 처리한다.

클라이언트에서 스레드를 하나 더 추가한다.

Timer스레드로 재장전할때 시간을 계산하여 3초뒤에 장전이 완료 될 수 있도록 설계하였다.

이때 스레드가 블록되어 실행이 멈추면 안되므로 furture라이브러리에 있는 비동기함수인 async를 이용하여 구현하였다.

3)server low-level

텍스트, 친필, 폰트, 서예이(가) 표시된 사진

자동 생성된 설명

수정 전 서버 low level

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

최종 서버 low level

Socketinfo\* socketinfo : 특정 클라이언트의 아이디와 소켓정보.

char\* m\_name: 멤버변수 이름

float m\_x,m\_y: 클라이언트로부터 받는 탱크의 좌표값

short m\_hp: 플레이어 체력

float m\_speed: 탱크 속도

bool m\_online: 플레이어가 접속중인지 판단 전부 접속하면 게임 시작패킷 보냄

접속중이 플레이어에게만 패킷을 전송한다.

short m\_bullet\_cnt : 클라이언트 총알 개수

bool m\_accept\_player: accept()함수를 통해 성공적으로 소켓이 리턴되었는지 판단

bool m\_ready\_player:  접속이 완료된 플레이어에 대해서 적용되는 레디값

float m\_yaw : 탱크 head의 yaw값

float m\_bodyyaw: 탱크 body의 yaw값

bool m\_collision : 탱크-탱크 충돌 여부

bool m\_wallCollision : 탱크-벽 충돌 여부

추가된 함수

bool collision\_Chk(float aL, float aR, float aT, float aB, float bL, float bR, float bT, float bB)

객체간의 충돌을 체크해주는 bool 함수

void tank\_collid(std::array<Session, MAX\_USER>& players)

탱크간의 충돌을 확인하고, collision 맴버 변수에 넘겨주는 함수

void wall\_collid(std::array<Session, MAX\_USER>& players,short id)

탱크와 벽 간의 충돌을 확인하고 ,collision멤버변수에 넘겨주는 함수

4)스레드

스케치, 그림, 도표, 텍스트이(가) 표시된 사진

자동 생성된 설명

수정 전 스레드

수정, 변경사항은 후술

1)Main Thread

소켓을 생성하고 초기화 함 listen상태로 계속해서 클라이언트의 접속을 받기 위해 대기함

2.Client Thread

accept()를 성공적으로 받으면 스레드를 할당한 뒤 클라이언트로부터 데이터를 받음

3.WokerThread

클라이언트로부터 전송받은 패킷의 세부정보를 해석하고 처리하는 스레드

WokerThread를 따로 할당하지 않고 ClientThread에서 데이터를 받음고 함께 패킷의 세부정보를 해석하고 처리한다.

4.do\_send

해석된 데이터를 클라이언트로 전송하는 스레드

초당패킷을 이용하여 1초에 송신 횟수를 제한한다.

ex) 클라이언트로부터 접속을 받은 서버는 클라이언트1스레드를 할당한다. 만약 클라이언트로부터 데이터를 수신받았다면 클라이언트 내부 루프에서 switch문을 통해 DATA\_TYPE::RECV: 로 이동 후WokerThread를 통해 데이터를 처리한다,  처리가 끝난 데이터들에 대해서 DATA\_TYPE::SEND: 로 이동 후 do\_send스레드를 통해 데이터를 클라이언트에게 송신한다.

ex)클라이언트로부터 접속을 받은 서버는 클라이언트1스레드를 할당한다. 만약 클라이언트로부터 데이터를 수신받았다면 ClientThread에서 packet\_type에 해당하는 연산을 수행한다. 로그인 및 레디 및 플레이어 사망 정보는 즉각적으로 클라이언트에게 뿌려주고  좌표는 do\_send()에서 초당패킷으로 계속 클라이언트에게 송신한다.

5)패킷

struct CS\_LOGIN\_PACKET {

char type;

char name[NAME\_SIZE];

};

로그인시 클라이언트에서 정보를 받아오는 패킷

struct CS\_READY\_PACKET {

char type;

char name[NAME\_SIZE];

};

클라이언트 레디시 정보를 받아오는 패킷

struct CS\_MOVE\_PACKET {

char type;

char direction;

float bodyYaw;

};

클라이언트의 ROTATE와 MOVEMENT정보를 받는 패킷

struct CS\_ATTACK\_PACKET {

char tpye;

float now\_yaw;

float x;

float z;

bool isshoot;

};

클라이언트로 받은 Bullet의 정보를 받아오는 패킷

struct CS\_ITEM\_PACKET {

char type;

char item;

bool exist;

int num;

};

아이템의 정보를 담은 패킷

struct CS\_YAW\_PACKET {

char type;

float yaw;

float bodyYaw;

};

클라이언트로 부터 탱크의 HEAD의 YAW 정보를 받아오는 패킷

struct CS\_HIT\_PACKET {

char type;

int id;

int hp;

bool freeze\_bullet;

};

클라이언트의 아이템이나 충격을 받았을때의 HP정보를 받아오는 패킷

struct SC\_LOGIN\_OK\_PACKET {

char type;

char name[NAME\_SIZE];

int id;

int hp;

float x;

float y;

float z;

float speed;

short bullet\_cnt;

};

클라이언트의 로그인 여부의 정보를 받고 보내는 패킷

struct SC\_READY\_OK\_PACKET {

char type;

char name[NAME\_SIZE];

};

클라이언트의 레디 정보를 받고 보내는 패킷

struct SC\_DIE\_PACKET {

char type;

int id;

};

클라이언트의 사망정보를 전송하는 패킷

struct SC\_GAMESTART\_PACKET {

char tpye;

};

게임 플레이 시작 여부를 전송하는 패킷

struct SC\_UPDATE\_PACKET {

char type;

short id;

short hp;

float x;

float y;

float z;

float speed;

short bullet\_cnt;

float yaw;

float bodyYaw;

};

초당 패킷으로 클라에게 계속 정보를 전송하는 패킷

struct SC\_SET\_ITEM\_PACKET {

char type;

char item\_type;

bool exist[3];

float x[3];

float z[3];

};

아이템 정보를 보내주는 패킷

struct SC\_ALL\_ITEM\_SET\_PACKET {

char type;

};

아이템 세팅 여부정보를 보내주는 패킷

struct SC\_RELOAD\_PACKET {

char type;

int bullet\_num;

};

클라이언트 Bullet정보를 보내주는 패킷

std::array<Session,MAX\_USER> g\_players;

동접수가 많지 않으므로 플레이어 컨테이너는 array를 사용 session에서 정의한 구조체를 바탕으로 글로벌플레이어 변수 선언

공유자원인 g\_players로 인한 임계영역 사용

하나의 프로세스에 대해서만 임계영역을 설정할 필요가 있기 때문에 무거운 커널영역에 있는 객체인 mutex를 사용하기 보다는 criticalSection을 사용할예정

임계영역을 사용하였지만 mutex객체도 사용하였는데 lock\_guard를 사용하였다. 일일이 LeaveCriticalSection()함수를 호출하는것이 번거로웠기 때문이다.

플레이어 사망 시 접속중인 클라이언트 모두에게 사망 패킷을 보내고 서버도 정상적으로 종료되게 구현하였다.

1. **팀원간 역할분담**

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **구분 내용** | **구현 내용** | **담당자** |
| **서버** | **Session** | **Session class** | **김진선** |
| **로그인,레디** | **서버ClientThread()**  **로그인 및 레디 처리** | **김성준** |
| **움직임** | **서버ClientThread()**  **움직임 처리** | **김성준** |
| **아이템** | **서버ClientThread()**  **클라 RecvThread()**  **아이템 세팅 처리** | **김성준** |
| **재장전** | **서버ClientThread()**  **클라 TimerThread()**  **재장전 처리** | **김성준** |
| **사망처리,게임종료** | **서버ClientThread()**  **사망,게임종료처리** |
| **초당패킷send** | **do\_send()** | **김성준** |
| **충돌처리** | **collision()** | **백하빈,김진선** |
| **클라이언트** | **서버로부터 받은 데이터 처리** | **RecvThread()** | **김성준** |
| **서버로 데이터 송신** | **sendPacket()** | **김성준** |
| **클라이언트 연결 구현** | **connect()까지** | **김성준** |
| **재장전 쿨타임구현** | **TimerThread()**  **재장전 쿨타임 처리** |

1. **개발일정**

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명