**네트워크 게임 프로그래밍**

**텀프로젝트**

2013180006 김재환

2013180040 지성인

2013182011 김선필

목차

[**1.** **문서 수정 내역** 3](#_Toc528820274)

[**2.** **어플리케이션 기획** 4](#_Toc528820275)

[A. 개요 4](#_Toc528820276)

[B. 설명 4](#_Toc528820277)

[C. 게임 구성요소 5](#_Toc528820278)

[**3.** **High Level Design** 6](#_Toc528820283)

[A. 프로토콜: TCP 6](#_Toc528820284)

[B. 서버&클라이언트 흐름 6](#_Toc528820285)

[C. 흐름도 7](#_Toc528820289)

[**4.** **Low Level Design** 8](#_Toc528820290)

[A. 프로토콜 8](#_Toc528820291)

[B. 데이터 전송 주기 10](#_Toc528820295)

[C. 데이터 흐름도 10](#_Toc528820296)

[D. 함수 정의 11](#_Toc528820297)

[**5.** **환경 및 세부 개발일정 계획** 14](#_Toc528820302)

[A. 개발 환경 14](#_Toc528820303)

[B. 팀원 별 역할분담 14](#_Toc528820304)

[C. 일정 계획 14](#_Toc528820305)

# **문서 수정 내역**

|  |  |  |
| --- | --- | --- |
| 일자 | 수정 내용 | 작성자 |
| 10/28 | 1차 초안 작성 | 김재환,김선필,지성인 |
| 10/31 | 게임 변경, 개발 일정 수정 | 김선필 |
| 10/31 | 개발 일정 수정, 함수 정의  Collsion() 함수 통일, 역할 변경 | 지성인 |
| 10/31-11/1 | 개발 일정 수정, 일부 사진과 설명문 디테일 수정 | 김재환 |
| 11/1 | 스레드 동기화 부분 추가, 함수 오타 수정 | 김선필 |
| 11/5 | 개발 일정 수정 | 지성인 |
| 11/21 | 함수, 일정 수정 | 김선필 |

# **어플리케이션 기획**

## 개요

**2D Top-View 실시간 네트워크 대전게임**



## 설명

클라이언트 프로그램 실행 시, 처음에 서버의 IP를 입력하여 접속한다. 버 접속이 정상적으로 되었다면 자동으로 게임 화면으로 넘어간다.

게임 화면에 2명 이상의 클라이언트가 접속 시 본 게임이 시작된다.

플레이어는 공격 키를 눌러서 다른 플레이어를 공격할 수 있다.

플레이어의 공격 시점에서 해당 플레이어의 공격 범위에 존재하는 다른 플레이어는 공격을 맞으면 일정량의 피해(HP 감소)를 받고, 공격에 성공한 플레이어는 점수를 얻는다.

HP가 0이 된 플레이어는 사망하며, 다른 플레이어의 HP를 0으로 만든 플레이어는 추가로 일정량의 점수를 획득한다.

서버 시간을 세서 일정 시간이 지나면 서버에서 가장 점수가 많은 유저를 찾아 해당 유저를 승리한 것으로 판단해, 승리한 플레이어의 id를 출력하고 게임을 끝낸다.

게임이 끝나면 접속중인 클라이언트 인원 수를 세고 2명 이상이라면 새 게임을 재개한다. 2명 미만이라면 2명 이상이 될 때까지 다른 클라이언트의 접속을 기다리며 대기한다.

* *조작 방법 -*

|  |  |
| --- | --- |
| 키 입력 | 효과 |
| ↑,↓, ←,→ | 해당 방향으로 플레이어 이동 |
| A | 일반 공격, 플레이어가 바라보고 있는 방향(이전에 이동한 방향)의 앞 10cm의 범위로 공격 판정을 발생시킴 |

## 게임 구성요소

### 플레이어 객체

게임 시작 시 플레이어는 이동 불가 오브젝트가 존재하는 위치를 제외한 맵에서 임의의 좌표에서 시작한다.

플레이어는 다른 플레이어나 이동 불가 오브젝트의 위치로 이동할 수 없다.

플레이어는 플레이어 ID와 2차원 좌표 위치 정보, HP, 점수 정보를 가진다.

### 이동 불가 장애물 오브젝트

처음 맵이 생성될 때 자동으로 임의의 장소에 배치되며, 해당 오브젝트가 존재하는 좌표로는 플레이어가 이동할 수 없다.

모든 장애물 오브젝트는 오브젝트 ID와 2차원 좌표 위치 정보를 가진다.

### User Interface

모든 플레이어는 간단한 사각형 UI를 통해 자신의 HP가 얼마나 남았는지 확인할 수 있으며, 그 박스의 옆에 자신의 점수를 출력한다.

다른 플레이어의 HP나 점수는 확인할 수 없다.

(고득점 플레이어의 의도적 배제나 HP가 낮은 적의 마지막 일격을 훔쳐가는 행위 방지)

### World Map

게임 화면이 일어나는 주 전장이다.

위치 좌표를 통해 관리하며, 충돌 검사는 ~~위치 좌표를 서로 같은지~~

*사각형 간의 충돌로* 판정해서 처리한다.

# **High Level Design**

## 프로토콜: TCP

실시간 대전 게임이므로HP, 점수, 위치 등의 모든 주고 받는 데이터는 소실되어서는 안된다. 따라서 TCP프로토콜을 사용한다.

## 서버&클라이언트 흐름

### 서버 (게임 시작 전)

클라이언트의 접속을 기다린다. (listen)

클라이언트 접속 시 클라이언트와 통신하는 용도로 쓸 스레드를 생성하고 클라이언트 접속 순서대로 플레이어 ID를 부여한다.

접속한 클라이언트는 클라이언트의 게임 화면에 초기화된 정보를 렌더링하고, 클라이언트 접속자가 2명 이상이 될 때까지 본 게임이 시작되지 않는다.

게임 화면에 접속한 클라이언트에서 일어나는 플레이어의 키 입력을 받아와서 서버에서 이동, 공격, 충돌 처리 등의 작업을 전부 수행한다.

### 클라이언트

서버에 연결 요청 후, 정상 접속 되면 플레이어 기본 정보를 초기화한다.

서버에 연결되고 나면 서버에서 관리하는 월드맵 (기본 게임 화면) 상태로 전환하여 플레이어 객체를 띄우고, 키를 입력 받으면 서버로 입력한 키 값을 보낸다.

### 서버 (게임 시작 후)

클라이언트들이 맵에 2명 이상 존재하는 경우 (메인 스레드 제외 생성된 스레드 2개 이상) 게임이 흐르는 시간을 재기 시작한다.

시간을 재기 시작한 시점부터 일정 시간이 흐른 뒤, 가장 점수가 높은 사람을 승리자로 판정한다. (서버에서 최고점자의 정보를 가지고 있다가 해당 최고점에 도달한 사람이 있을 때마다 최고점자의 정보를 갱신한다.)

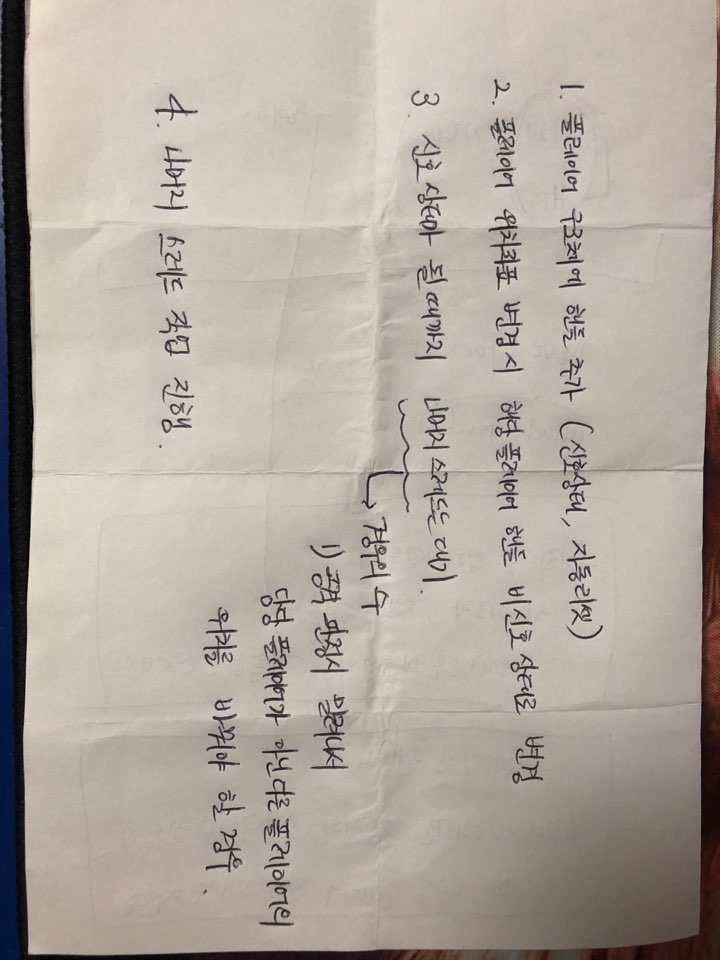
최고점이 동점으로 2명 이상 있는 경우, 가장 나중에 해당 점수에 도달한 사람이 승리자가 된다.

### 스레드 동기화

공유자원으로 플레이어 위치좌표, 체력 등의 정보가 있으며, 위치좌표의 경우 스레드 간의 순서가 중요하다고 판단되므로 임계영역이 아닌 Event를 사용하여 스레드 동기화를 진행한다.

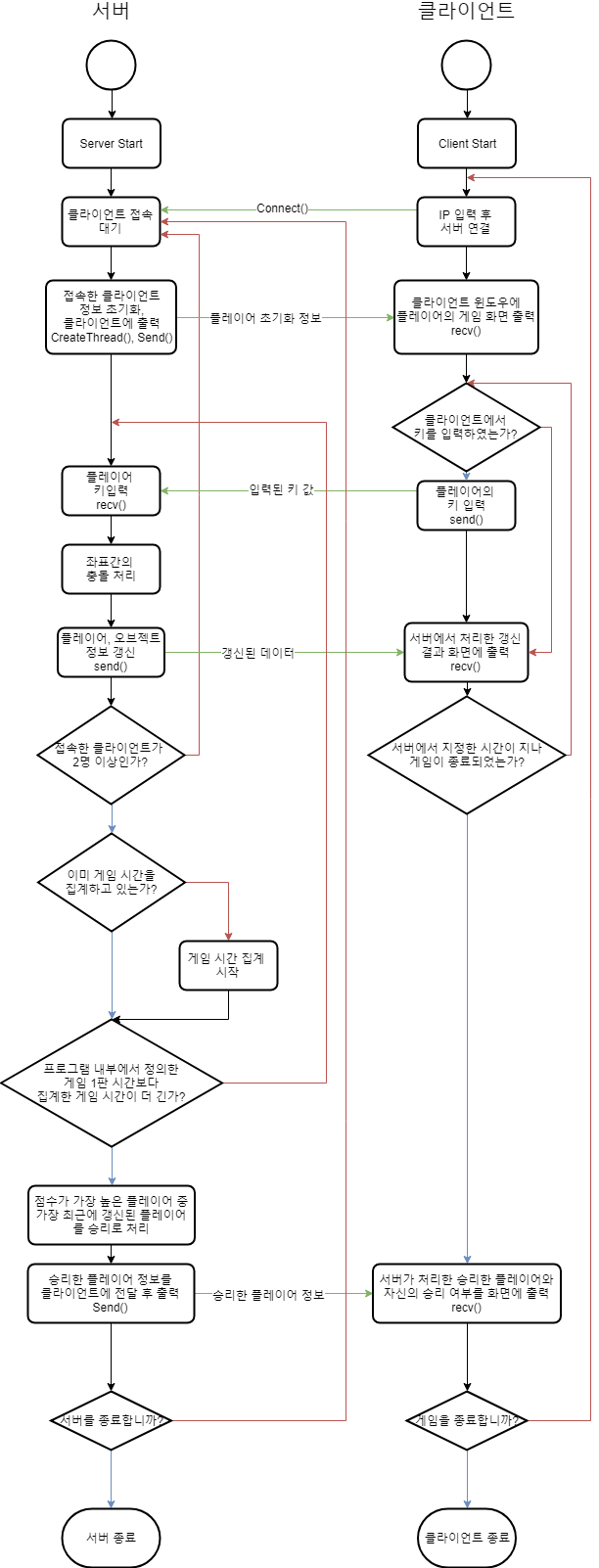
플레이어마다 이벤트 핸들을 하나씩 배당하여 구조체에 포함시키고, 플레이어 정보를 바꾸는 상황이 오면 비 신호 상태로 바꾸고 수정이 완료되면 신호 상태로 바꿔준다.

예상 스레드 동기화 작업 수도코드



중간에 Wait\*()함수를 사용하여 핸들의 상태가 바뀌는 것을 감지한다.(4-D-3 참고)

## 흐름도



# **Low Level Design**

## 프로토콜

### **공용**

**공용 패킷(모든 패킷은 이 패킷을 상속한다.)**

Struct Packet {int type, int size}

//패킷 타입, 패킷 사이즈

**오브젝트 패킷**

Struct Object {int id, int x, y}

//오브젝트의 ID, 위치

**플레이어 패킷**

struct Player {int id; intx,y; int hp; int score;}

//플레이어 ID, 위치, 체력, 점수

**게임 종료 패킷**

struct End {int id;}

//승리한 플레이어 아이디

### **Server**

**초기화 데이터 전송 패킷**

struct StartUp{ intplayerID;}

//서버에서 클라이언트에게 처음 게임 시작을 알려주는 패킷

**동기화 패킷**

struct SendToClient{ intplayerID; int x,y, int hp}

//플레이어 아이디, 플레이어 위치좌표,플레이어 체력

### **Client**

**플레이어 이동패킷**

struct MovePacket {int id, int direction}

//플레이어 아이디, 이동 요청한 방향

**플레이어 공격 패킷**

struct AttackPacket {int id, intattackType}

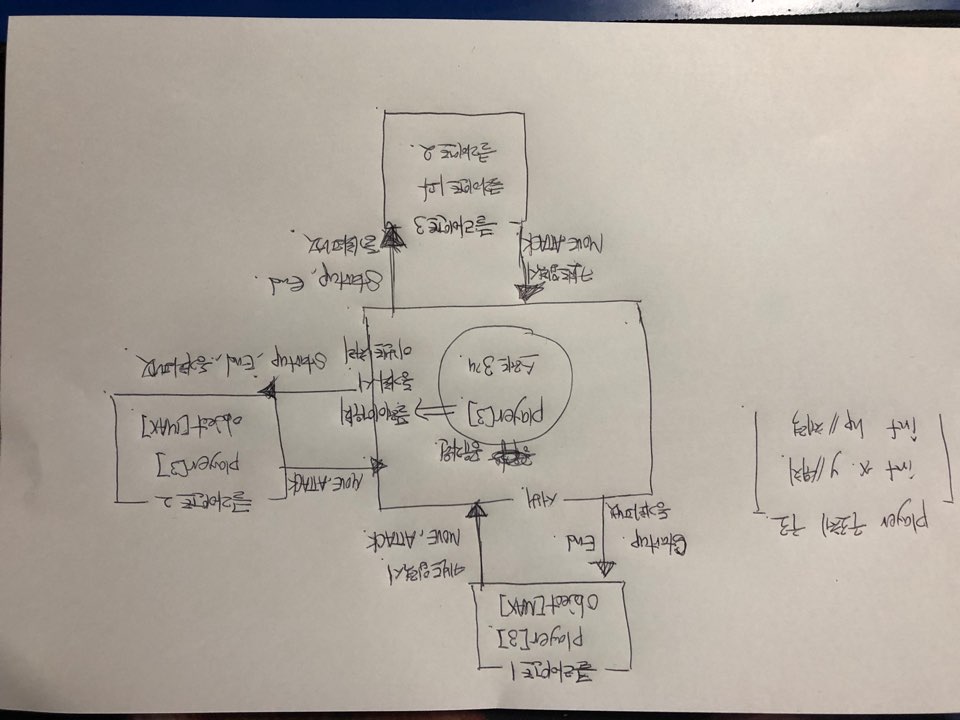
//플레이어 아이디, 공격 종류

|  |  |  |  |
| --- | --- | --- | --- |
| **구조체 이름** | | **주기** | **이동** |
| Startup | | 시작할 때 1회 | Server -> Client |
| Object | | 시작할 때 1회 | Server -> Client |
| SendtoClient | Player | 30ms당 1회 | Server -> Client |
| Object | 30ms당 1회 | Server -> Client |
| Time | 30ms당 1회 | Server -> Client |
| Move, Attack | | 키 입력 발생 시 | Client -> Server |
| End | | 게임이 끝날 경우 1회 | Server -> Client |

## 데이터 전송 주기

\*30프레임으로 가정 / 1초에 약 30번 패킷 송수신

## 데이터 흐름도



## 함수 정의

### 클라이언트

void InitPacketClient(SOCKADDR \*client\_socket, StartUp& startup);

// 서버에서 받아온 초기값으로 게임 초기화 함수

void RealTimeClient(SendtoClient\* sendToClient );

// 서버에서 받아온 데이터를 토대로 실시간 출력 함수

void Update(float ElapsedTime);

// 업데이트 함수

void SendKeyState(SOCKADDR\* client\_socket, Client\_Key\* key);

// 클라이언트에서 입력한 키값을 서버로 넘겨주는 함수.

**CNetwork**

**//소켓 통신 관련 클래스(밑은 내장함수)**

1. Initialize()
   1. 소켓 생성 및 연결
2. Filnalize()
   1. 소켓 닫기
3. SendPacket(char\* packet), RecvPacket()
   1. 서버와의 패킷 통신을 위한 함수
4. ProcessPacket(char\* buffer)
   1. 서버에서 온 패킷 처리 함수

### 서버

void Initialize();

// 게임 데이터들을 초기화 해주는 함수.

void Update(float ElapsedTime);

// 업데이트 함수

void Collision(Player\* player, Object\* object);

// 플레이어와 다른 오브젝트 충돌

// 오브젝트는 플레이어, 이동 불가 오브젝트 등의 모든 오브젝트의 부모 클래스

~~void Collision(Player\* player, Bullet\* bullet);~~

~~// 플레이어와 총알 충돌~~  // Collsion 하나의 함수로 통일

Void Victory(bool win)

// 게임 승패 판정 함수

void InitPacketServer(SOCKADDR\* client\_socket, StartUp\* startup);

//데이터 초기화 패킷 전송

void SendToClientPacket(SOCKADDR\* client\_sock, SendToClient\* packet);

// 클라이언트에게 패킷을 전송하는 함수.

### 서버의 스레드 함수

DWORD WINAPI ClientEventThread (SOCKET arg, KeyState\* key, Player\* player, Object\* object);

// 플레이어의 키입력에 대한 연산 처리, 오브젝트와의 충돌처리 등을 담당하는 연산 처리 스레드 함수, Wait\*(), CreateEvent()를 사용한 동기화 처리

~~DWORD WINAPI ConnectThread(SOCKET arg, StartUp\* stratup);~~

~~//각각의 클라이언트와 통신하는 쓰레드 함수, 초기화도 여기서 해준다.~~

DWORD WINAPI SyncThread(SOCKET arg, Player\* player);

//주기적으로(4-B표 참고) 플레이어의 위치, 체력 정보 모든 클라이언트에게 전송

### 기타 사용할 기본 제공 함수

Int recvn(int s, void \*buf, size\_tlen, int flags);

// 사용자 정의 데이터 수신 함수

int send(int s, const void \*msg, size\_tlen, int flags);

// 데이터 전송 함수

WaitFor\*(HANDLE hHandle, DWORD dwMilliseconds)

//스레드 순서제어를 위해 이벤트와 같이 사용할 종료 대기 함

CreateEvent(lpEventAttributes, bManualReset, bInitialState, lpName)

//이벤트 생성 함수, 자동 리셋

# **환경 및 세부 개발일정 계획**

## 개발 환경

|  |  |
| --- | --- |
| **Operating System** | Windows 10 |
| **Integrated Development Environment** | Visual Studio 2017 Community |
| **Language** | C / C++ |
| **Application Programming Interface** | Window API |
| **Version Control System** | Git hub |

## 팀원 별 역할분담

|  |  |  |  |
| --- | --- | --- | --- |
| 팀원 | **김재환** | **김선필** | **지성인** |
| 역할 분담 | 게임 개발 | | |
| 클라이언트의 Send, Recv구현 | Thread함수 및 동기화 구현 | 서버의 Send, Recv구현 |
| 클라이언트 및 오브젝트 구현 | 서버 및 클라이언트 구현 | 서버 및 플레이어 구현 |

## 일정 계획

|  |  |  |  |
| --- | --- | --- | --- |
|  | **김재환** | **김선필** | **지성인** |
| 10/29 | 계획서검토 | | |
| 10/30 | 계획서 검토 | | |
| 10/31 |
| 11/1 | 계획서 수정/검토 | | |
| 11/2 |
| 11/3 |
| 11/4 |
| 11/5 |
| 11/6 | 클라이언트 프레임워크  (게임 로직) 구현 | UI 제작 | 키보드/마우스입력구현 |
| 11/7 |
| 11/8 | 코드 수정 및 보완 | ~~게임승패판정~~  ~~Victory() 함수구현~~  클라이언트 Network 클래스 구현 및 Initialize() 구현 | 충돌 처리 구현 |
| 11/9 | 게임 테스트/코드 수정 및 보완 | | |
| 11/10 |
| 11/11 |
| 11/12 | 오브젝트 패킷 구현 | ~~공용패킷구현~~  Protocol.h 추가  공용패킷, Put Packet 추가 | 플레이어 패킷 구현 |
| 11/13 |  | 네트워크 클래스 구현 |  |
| 11/14 | 코드 수정 및 보완 | 서버 –클라이언트  프레임워크 작성 | 플레이어이동패킷구현 |
| 11/15 |  | 플레이어공격패킷구현 |
| 11/16 | AsyncSelect 연구  WM\_SOCKET 추가 |
| 11/17 | ~~클라이언트 패킷 송/수신 프레임워크 작성~~ | ~~ClientEventThread()~~  ~~함수 구현~~  클라이언트 패킷 송/수신 프레임워크 작성 | ~~서버 패킷 송/수신~~  ~~프레임워크 작성~~ |
| 11/18 |
| 11/19 |
| 11/20 |  | 서버 패킷 송/수신 프레임워크 작성 |  |
| 11/21 | SendtoServer() 구현 |  | SendtoClient() 구현 |
| 11/22 |
| 11/23 | StartUpServer() 구현 | ~~Initialize() 구현~~ | StartUpClient() 구현 |
| 11/24 |  |  | sendKeyState구현 |
| 11/25 | RealTimeClient() 구현 | Wait\*(), CreateEvent() | 코드 수정 및 보완 |
| 11/26 |  |
| 11/27 |  | | |
| 11/28 | 게임종료패킷구현 | SyncThread() 구현 |  |
| 11/29 | 게임테스트 | 오류 수정 및 보완 |
| 11/30 |
| 12/1 | 서버, 클라이언트 간 전송 확인 및 테스트 | | |
| 12/2 | 사운드 / 추가 리소스 수집 |  | 게임 테스트 |
| 12/3 |
| 12/4 | 오류 수정 및 보완 | 게임테스트 | 오류 수정 및 보완 |
| 12/5 |
| 12/6 |  | 오류 수정 및 보완 |  |
| 12/7 |
| 12/8 | 동기화 작업 및 수정 | | |
| 12/9 |
| 12/10 | 최종 테스트 및 수정, 마무리 | | |
| 12/11 |
| 12/12 | 제출 | | |