**네트워크 게임 프로그래밍**

**텀프로젝트 추진계획서**

2017180017 송용섭

2018180005 김명훈

2019180050 이민식

**목차**

1. 애플리케이션 소개
2. High-level 디자인
3. Low-level 디자인

4. 개발환경

5. 역할분담 및 개발 일정

**1. 애플리케이션 기획**

크레이지 아케이드

게임 소개

블럭 사이에서 플레이어 두 명이 물풍선을 설치하고 적을 물줄기로 피격 시켜 상대방을 무력화시키는 게임이다. 맵에 배치되어 있는 블럭 중 파괴가능한 블럭에서 플레이어의 스테이터스를 올려주는 아이템을 획득하거나 공격, 방어에 관련된 아이템을 얻어 활용 가능하다.단순한 룰이지만 물풍선을 차거나 던지고 터트리는 등 다양한 전략이 사용 가능하며, 동체시력과 반응속도가 요구된다.

<게임 스크린샷>



<게임 시작>

4명의 플레이어의 연결을 받은 경우 게임이 시작된다.

<게임 종료>

상대 팀의 플레이어가 전원 게임 오버 상태가 되면 게임이 종료된다.  
동시에 모든 플레이어가 게임 오버 상태가 되면 Draw상태로 게임이 종료된다.

<캐릭터, 스테이지>

팀당 플레이어 객체 두 개, 스테이지 한 개로 구성한다.

<기본적인 움직임 및 조작키>

상하좌우 이동 : W, A, S, D

물풍선 설치 : SPACE

1회성 아이템 사용 : T

<물풍선>

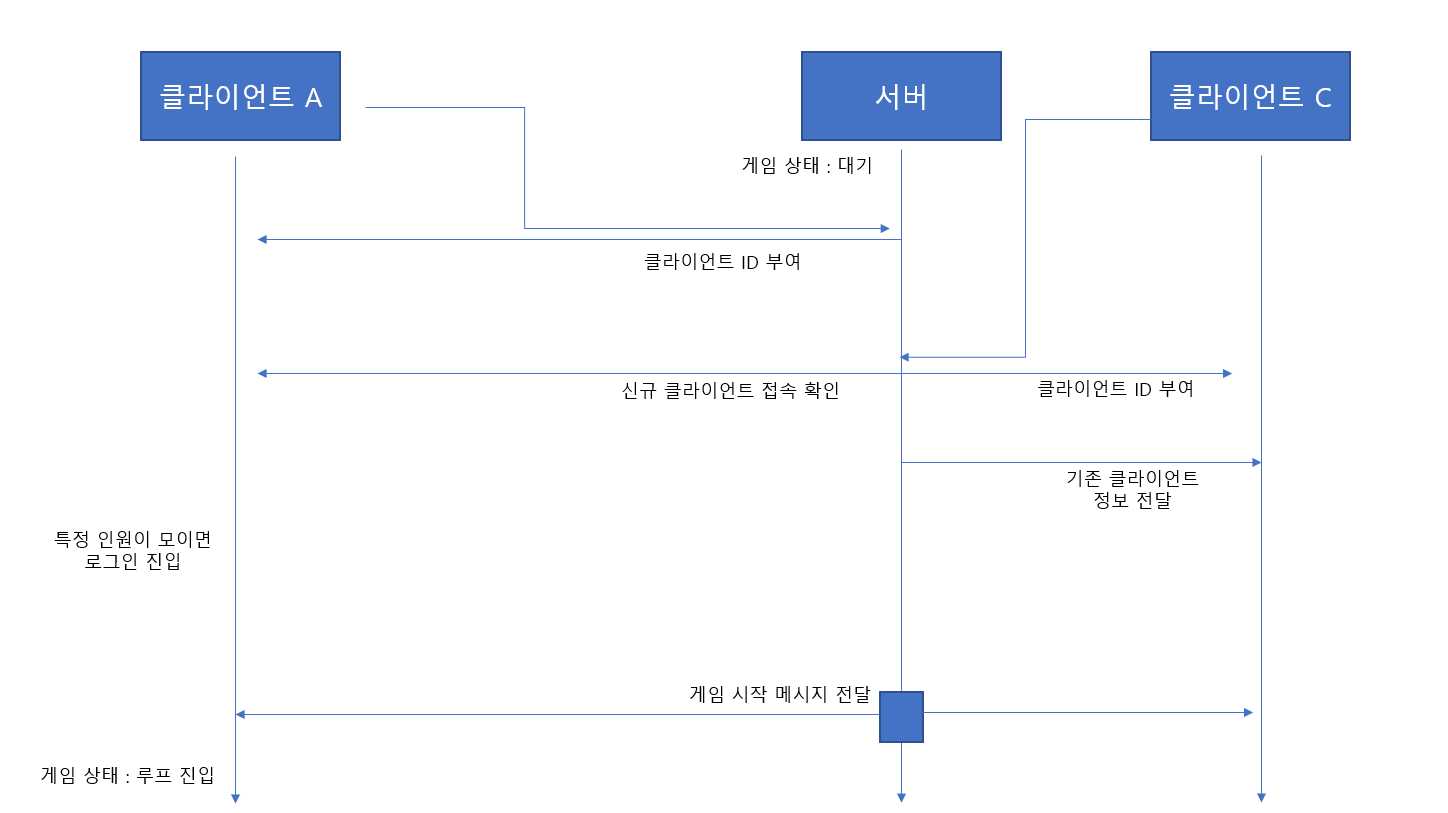
물풍선은 제한된 개수를 한 번에 설치할 수 있다.  
플레이어는 물풍선을 지나칠 수 없다.  
제한된 개수를 모두 설치하면 3초의 쿨타임 이후 다시 설치가 가능하다.  
물풍선에 피격되면 행동불가 상태에 빠진다.  
행동불가 상태에 빠진 후 적과 충돌하거나 일정 시간이 지나면 사망 애니메이션이 출력되며 게임 오버 상태가 되고, 팀원과 충돌하면 행동불가 상태를 벗어나 원래 상태로 돌아온다.  
파괴가 가능한 오브젝트는 물줄기에 피격되면 파괴된다.

<1회성 아이템>

바늘 : 자력으로 행동불가 상태를 탈출한다

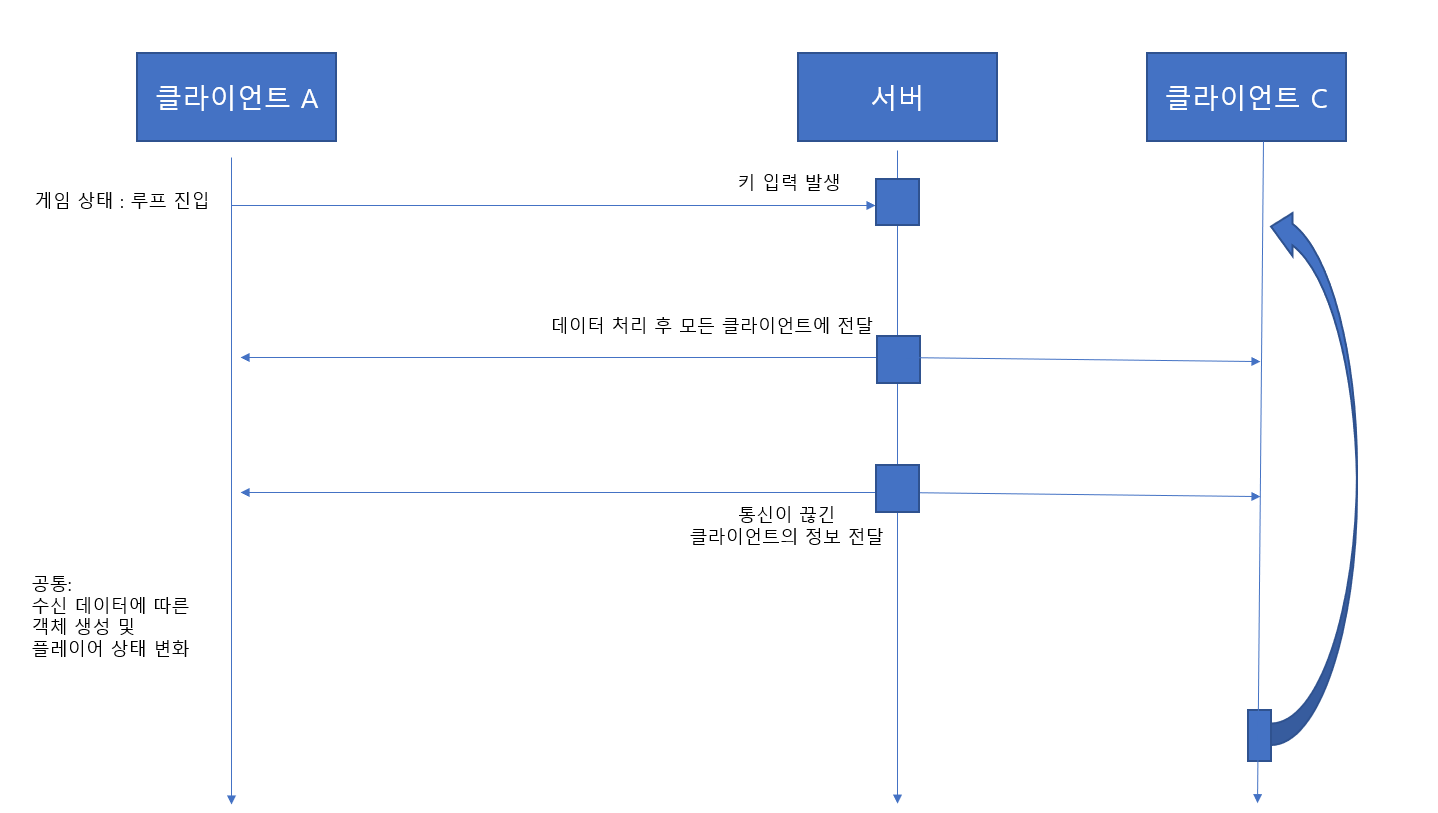
**2. HighLevel 디자인**

**흐름도1) 로그인 ~ 게임 루프 진입**



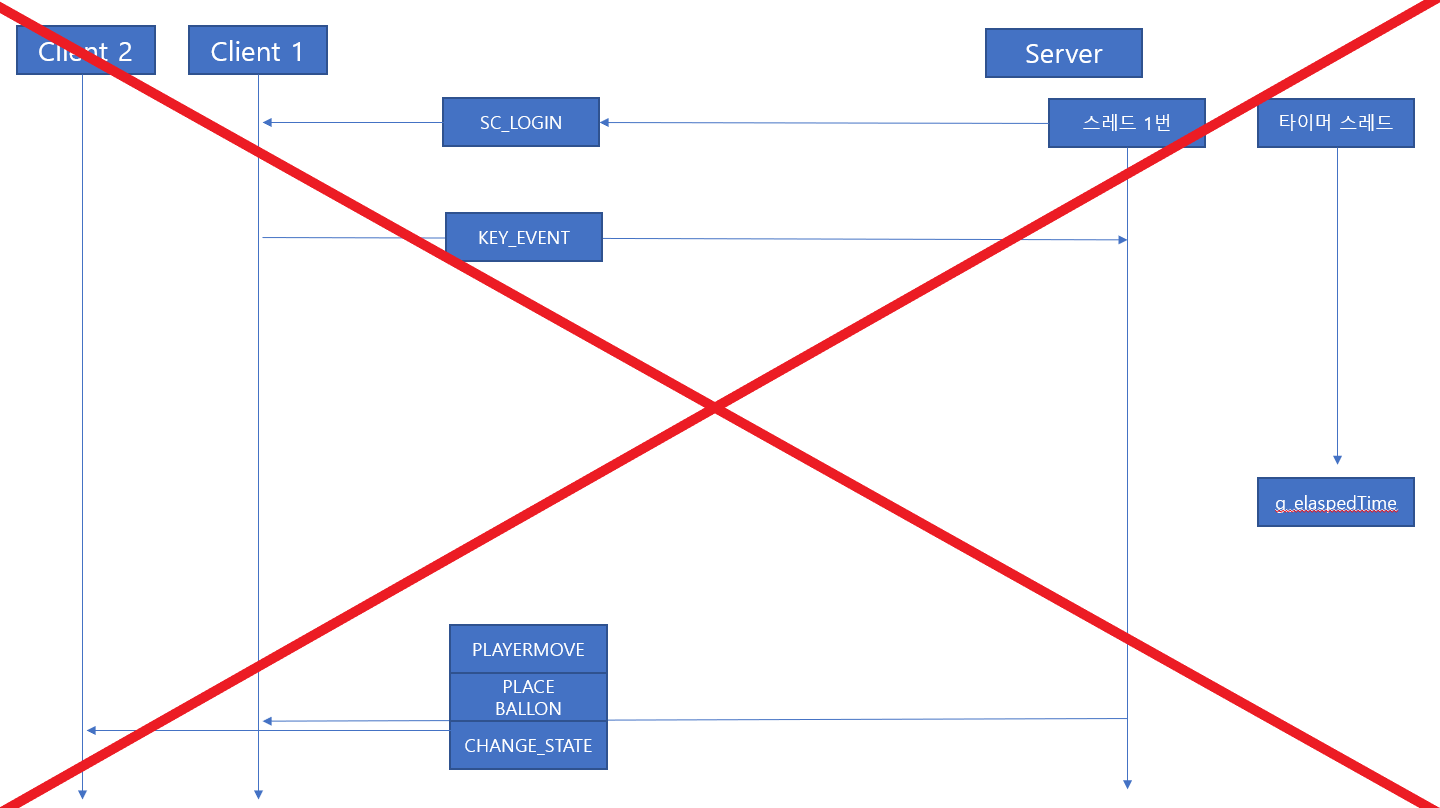
|  |  |
| --- | --- |
| 클라이언트 | 서버 |
| 1. 서버에 로그인 요청  2. 전달받은 캐릭터 정보에 맞춰서 캐릭터 생성  3. 루프 진입. 캐릭터 움직임 해금 | 1. 클라이언트의 로그인 요청 수락  2. 클라이언트에 ID 부여  3. 클라이언트 팀(캐릭터 정보) 설정 후 전달  4. 기존 클라이언트에 신규 접속 클라이언트 정보를 전달한다.  *5. 클라가 모두 연결되면 게임 시작 메시지 전달* |

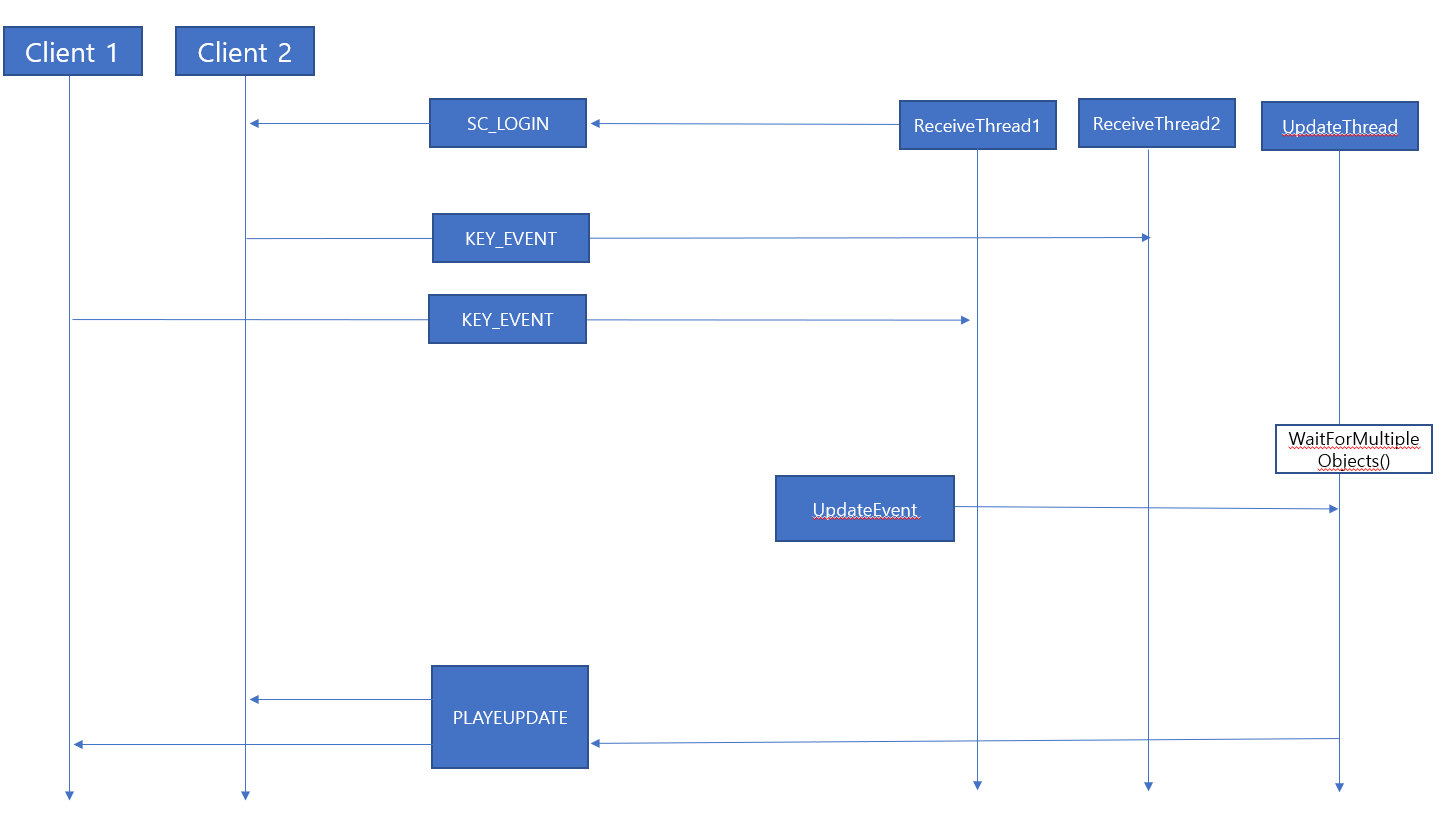
**흐름도2) 게임 루프 진입 ~ 게임 종료**



|  |  |
| --- | --- |
| 클라이언트 | 서버 |
| 1. 게임 루프 진입  2. 키 입력 시 서버에 패킷 전달.  (1회성 아이템 사용, 물풍선 설치, 캐릭터 이동, 애니메이션 프레임)  3. 전달받은 데이터를 바탕으로 객체 생성 및 캐릭터 상태변화.  4. 게임 종료. | 1. 전달받은 데이터 처리 후 모든 클라이언트에 공유  2. 통신이 끊긴 클라이언트가 있다면 해당 클라이언트의 플레이어 객체를 사망 처리. |

**멀티스레드 통신 구조**





클라이언트의 접속 요청을 받으면, 서버는 각 클라이언트에 대응하는 스레드를 생성한다.  
각 클라이언트는 ~~SC\_LOGIN~~ SC\_GAMEINFO를 받아 자신의 클라이언트 인덱스와, 접속 유저 수, 게임 시작 메시지, 팀 정보를 확인한다.  
서버는 전역에 선언된 배열 형태로 클라이언트 정보를 관리한다. ~~서버는 별개의 타이머 스레드를 가지고 있으며, 해당 전역 변수를 사용해 물풍선 객체를 관리.  
클라이언트에서 전달한 키 이벤트에 대응하는 패킷을 모든 클라이언트에 전송한다.~~

<동기화>

키 이벤트 입력에 따라 클라이언트에서 자신의 키타입 정보를 서버에 send()하며, 서버는 스레드에서 해당 키 이벤트 값을 recv()한다. recv()의 값이 모두 수집되면 키 값을 바탕으로 객체를 업데이트 후 모든 클라이언트 플레이어 객체에 대한 정보를 공유한다. 이 때, 서버는 WaitForMultipleObjects()를 사용해 모든 클라이언트의 키 입력을 대기하고, 세 번째 인자 값인 dwMilliseconds를 기준으로 일정 시간동안 키 입력이 없는 클라이언트는 행동이 없다 판단하고 객체 업데이트와 정보전달을 실행한다.

**2. LowLevel 디자인**

|  |  |
| --- | --- |
| **헤더 파일** | |
| #pragma once  ~~constexpr unsigned char PRESS\_LEFT = 0x00;~~  ~~constexpr unsigned char PRESS\_RIGHT = 0x01;~~  ~~constexpr unsigned char PRESS\_UP = 0x02;~~  ~~constexpr unsigned char PRESS\_DOWN = 0x03;~~  ~~constexpr unsigned char PRESS\_SPACE = 0x04;~~  ~~constexpr unsigned char PRESS\_ITEM = 0x05;~~  #define MAX\_BLOCKITEM\_CNT 135  #define VERTBLOCKCNT 15  #define HORZBLOCKCNT 13  #define BLOCKSIZE 20  #define INDEX\_MAPSTART 30  #define INDEX\_MAPEND 165  enum KEYEVENT { PRESS\_LEFT, PRESS\_RIGHT, PRESS\_UP, PRESS\_DOWN, PRESS\_SPACE, PRESS\_ITEM, PRESS\_END };  enum STATE { IDLE, UP, DOWN, LEFT, RIGHT, TRAPPED, SAVED, DIE, DEAD, WIN };  enum TEAM { DAO, BAZZI };  struct CLIENTINFO  {  SOCKET sock;  unsigned short ID;  CPlayer player;  };  #pragma pack(push, 1)  ~~struct FRAME~~  ~~{~~  ~~int StartX;~~  ~~int EndX;~~  ~~int StateY;~~  ~~float DelayTime;~~  ~~float Time;~~  ~~int RepeatCnt;~~  ~~int MaxRepeat;~~  ~~};~~  struct SC\_GAMEINFO  {  Unsigned short ID;  Int currentPlayerCnt;  Int gameStart;  TEAM teamId;  int blockType[MAX\_BLOCKITEM\_CNT];  int itemType[MAX\_BLOCKITEM\_CNT];  }  ~~struct SC\_MAPINFO~~  ~~{~~  ~~int blockType[MAX\_ITEM\_CNT];~~  ~~int itemType[MAX\_ITEM\_CNT];~~  ~~bool enabled;~~  ~~};~~  ~~struct SC\_LOGIN~~  ~~{~~  ~~Unsigned short ID;~~  ~~Int currentPlayerCnt;~~  ~~Int gameStart;~~  ~~TEAM teamId;~~  ~~};~~  ~~struct SC\_FRAME {~~  ~~int StartX;~~  ~~int StateY;~~  ~~float Time;~~  ~~int RepeatCnt;~~  ~~};~~  ~~struct SC\_PLAYERMOVE~~  struct SC\_PLAYERUPDATE  {  Unsigned short ID;  POINT pt;  STATE state;  int ballonLength;  bool keydown;  ~~SC\_FRAME frame;~~  };  ~~struct SC\_PLACEBALLON~~  ~~{~~  ~~Unsigned short ID;~~  ~~int index;~~  ~~int ballonLength;~~  ~~};~~  ~~struct SC\_CHANGESTATE~~  ~~{~~  ~~Unsigned short ID;~~  ~~STATE state;~~  ~~};~~  ~~struct CS\_KEYEVENT~~  struct CS\_EVENT  {  Unsigned short ID;  ~~Unsigned short keyType;~~  bool keyType[PRESS\_END];  Unsigned short Index;  };  ~~struct CS\_BLOCKEVENT~~  ~~{~~  ~~Unsigned short Index;~~  ~~};~~  ~~enum SC\_PACKET {~~  ~~MOVE = sizeof(SC\_MOVE),~~  ~~PLACEBALLON = sizeof(SC\_PLACEBALLON),~~  ~~CHANGESTATE = sizeof(SC\_CHANGESTATE),~~  ~~MAPINFO = sizeof(SC\_MAPINFO) };~~  ~~}~~  ~~enum CS\_PACKET {~~  ~~BLOCKEVENT = sizeof(CS\_BLOCKEVENT),~~  ~~KEYEVENT = sizeof(CS\_KEYEVENT),~~  ~~};~~  #pragma pack(pop) | ~~서버와 클라가 공유하는 키 타입 선언~~  맵에 설치된 최대 아이템 개수  가로 타일 수  세로 타일 수  블록 크기  팀 사이 설치되어 있는 장애물 시작위치  팀 사이 설치되어 있는 장애물 끝위치  서버와 클라가 공유하는 키 타입 선언  플레이어의 상태  소속 팀 정보  서버에서 관리하는 클라이언트 정보  Sock = 연결된 소켓  ID = 클라이언트 ID  CPlayer = 플레이어 객체  \*14p 참고  ~~클라이언트의 애니메이션 출력을 위한 구조체~~  ~~StartX , EndX: BitBlit출력을 위한 X좌표~~  ~~StartY : BitBlit출력을 위한 Y좌표~~  ~~DelayTime : 객체의 애니메이션 재생속도~~  ~~Time : 객체의 애니메이션 진행시간~~  ~~RepeatCnt : 애니메이션 총 출력횟수~~  ~~MaxRepeat : 애니메이션 반복횟수~~  로그인한 플레이어에게 최초 1회 보내는  초기 게임 정보  // ID, 팀 정보   * ID : 부여받은 플레이어 ID * currentPlayerCnt : 현재 접속자수 * gameStart : 게임 시작 여부 * teamId : 소속 팀 정보 * blockType : 블록 종류 * itemType : 아이템 종류   // 맵 정보   * ~~blockType : 블록 종류~~ * ~~itemType : 아이템 종류~~ * ~~enabled : 아이템 활성화 여부~~   ~~플레이어 로그인 화면을 위한 정보~~   * ~~ID : 부여받은 플레이어 ID~~ * ~~currentPlayerCnt : 현재 접속자수~~ * ~~gameStart : 게임 시작 여부~~ * ~~teamId : 소속 팀 정보~~   ~~애니메이션 재생을 위한 패킷.~~  ~~클라이언트가 필요로 하는 정보들로만 구성해 간소화시킴.~~  변화된 플레이어 정보를 담은 패킷   * ID : 플레이어 ID * pt : 플레이어 위치 * state : 플레이어 상태 정보 * ballonLength : 물줄기 길이 * keydown: 키다운 여부   ~~물풍선 설치 패킷~~   * ~~ID : 플레이어 ID~~ * ~~Index : 풍선이 놓일 타일 위치~~ * ~~ballonLength : 물줄기 길이~~   ~~플레이어 상태 전환 패킷~~   * ~~ID : 플레이어 ID~~ * ~~state : 플레이어 상태~~   클라이언트가 서버에 보내는 이벤트 패킷   * ID : 플레이어 ID * keyType : 입력 키 정보 * Index: 파괴된 블록의 index   ~~파괴된 블록 정보 패킷~~   * ~~Index 블록의 index 위치~~ |

**<서버 함수>**

~~DWORD WINAPI ProcessClient(LPVOID arg) : 클라이언트가 전송한 키 이벤트를 받고, 객체를 업데이트해 모든 클라이언트에게 결과값을 전달.~~

DWORD WINAPI RecvThread(LPVOID arg) : 클라이언트가 전송한 키 이벤트를 받고, 객체를 업데이트한다. 모든 객체가 recv()를 완료하면 SetEvent()를 호출하여 UpdateThread로 모든 클라이언트에 뿌려준다.

Void Map\_Init() : 맵 초기화, 아이템 종류 및 위치 지정

Void Player\_Init() : 플레이어의 아이디, 위치, 팀 부여

Void Player\_Collision() : 플레이어간 충돌체크

Void Move() : 플레이어 이동

Void Map\_Update() : 지형지물 변화 체크

Void Use\_Needle() : 특수 아이템 사용

Void Set\_Ballon() : 풍선 설치

DWORD WINAPI ~~Object\_Timer~~ UpdateThread (LPVOID arg) : ~~서버 객체 업데이트 타이머~~ 모든 데이터를 클라이언트에 뿌려준다

**<클라이언트 함수>**

DWORD WINAPI RecvData(LPVOID arg)

{

Int retval;

While(1)

{

SC\_PLAYERUPDATE UPDATE{};

Retval = recv(sock, (char\*)&UPDATE, sizeof(SC\_PLAYERUPDATE), MSG\_WAITALL);

If (retval == SOCKET\_EROR)

err\_display("recv()");

}

~~PACKET packetType{};~~

~~retval = recv(sock, (char\*)&packetType, sizeof(int), MSG\_WAITALL);~~

~~if (retval == SOCKET\_ERROR) {~~

~~err\_display("recv()");~~

~~}~~

~~Switch(packetType)~~

~~{~~

~~case MOVE~~

~~{~~

~~SC\_PLAYERMOVE Move{};~~

~~retval = recv(sock, (char\*)&playerMove, sizeof(packetType), MSG\_WAITALL);~~

~~if (retval == SOCKET\_ERROR) {~~

~~err\_display("recv()");~~

~~}~~

~~players[Move.ID].setPosition(pt);~~

~~players[Move.ID].changeState(state);~~

~~players[Move.ID].setFrame(frame);~~

~~}~~

~~break;~~

~~case PLACEBALLON:~~

~~{~~

~~SC\_PLACEBALLON placeBallon{};~~

~~retval = recv(sock, (char\*)&placeBallon, sizeof(packetType), MSG\_WAITALL);~~

~~if (retval == SOCKET\_ERROR) {~~

~~err\_display("recv()");~~

~~}~~

~~ballons.emplace\_back(map[index].GetPos(), map[index].GetSize(), index, ballonLength, clientNum, &players[clinetNum]);~~

~~}~~

~~break;~~

~~case CHANGESTATE:~~

~~{~~

~~SC\_CHANGESTATE changeState{};~~

~~retval = recv(sock, (char\*)&changeState, sizeof(packetType), MSG\_WAITALL);~~

~~if (retval == SOCKET\_ERROR) {~~

~~err\_display("recv()");~~

~~}~~

~~players[changeState.ID].ChangeState(changeState.state);~~

~~}~~

~~break;~~

~~case MAPINFO:~~

~~{~~

~~SC\_MAPINFO mapInfo{};~~

~~retval = recv(sock, (char\*)&mapInfo, sizeof(packetType), MSG\_WAITALL);~~

~~if (retval == SOCKET\_ERROR) {~~

~~err\_display("recv()");~~

~~}~~

~~for (int i = 30; i < 165; ++i)~~

~~obstacles.emplace\_back(map[i].GetPos(), map[i].GetIndex(), map, mapInfo.blockType);~~

~~}~~

~~break;~~

~~}~~

~~}~~

설명 :

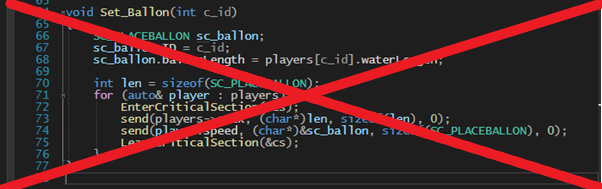
서버함수는 클라이언트 키 입력값을 받아 객체값에 변화를 주고 클라이언트에 전달한다.  
클라이언트 함수는 서버에 키 입력을 전달한다. 서버에서 전달받은 값을 바탕으로 객체들을 업데이트한다. ~~고정길이로 데이터 패킷에 대한 정보 및 크기를 파악한다.~~

~~SC\_MOVE 경우 플레이어의 움직임을 제어하고, 애니메이션에 대한 제어를 받는다.  
PLACE\_BALLON은 풍선의 위치 인덱스를 받고, 관리 컨테이너에 물풍선 객체를 추가한다.  
CHANGE\_STATE는 플레이어 상태 변환에 필요한 데이터를 받아 적용시키고  
MAP\_INFO는 게임 초기화 단계에서 블록 배치 정보와 아이템 정보를 받아 관리 컨테이너에 객체를 추가한다.~~

**<샘플 코드>**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

Player객체

텍스트이(가) 표시된 사진

자동 생성된 설명

**<개발 환경>**

IDE : VS 2019  
사용 라이브러리 : WINAPI  
윈도우 버전 : Window 11Pro / Window 10 / Window 10  
사용 버전 관리 시스템 : GitHub

**<역할 분담 및 일정>**

**명훈**

|  |  |
| --- | --- |
| 11 / 15 | 맵 생성 함수 구현  (완료) |
| 11 / 16 |
| 11 / 17 | 병합 작업 및 정기 회의 |
| 11 / 18 | ~~recvThread() - SC\_MAPINFO 패킷 전송 구현~~  SendClientEvent() - CS\_EVENT 패킷 전송 구현  (완료) |
| 11 / 19 |
| 11 / 20 |  |
| 11 / 21 |  |
| 11 / 22 | ~~SendClientEvent() - CS\_EVENT 패킷 전송 구현~~  UpdateThread() – 서버 내부 객체 업데이트 코드 작성(완료) |
| 11 / 23 |
| 11 / 24 | 병합/동기화 작업 및 정기 회의 |
| 11 / 25 | ~~UpdateThread() 함수 구현 보조, 패킷 송수신 결과 확인~~  UpdateThread() – SC\_PLAYERUPDATE 송신 구현(완료) |
| 11 / 26 |  |
| 11 / 27 |  |
| 11 / 28 | 클라이언트와 물줄기 오브젝트의 충돌 처리 구현  ~~및 클라이언트 객체 Trapped 상태 변화 구현~~ |
| 11 / 29 |
| 11 / 30 |
| 12 / 1 | 병합/동기화 작업 및 정기 회의 |
| 12 / 2 | ~~Use\_Needle() - 특수 아이템 사용 구현과 그로 인한 상태 변화 구현~~  클라이언트 객체 Trapped 상태 변화 구현 |
| 12 / 3 |
| 12 / 4 |  |
| 12 / 5 |  |
| 12 / 6 | 최종 테스트 및 수정, 마무리 |
| 12 / 7 |
| 12 / 8 | 제출 |

**용섭**

|  |  |
| --- | --- |
| 11 / 15 | 객체 생성 함수 InitGameInfo() 구현 후, 클라이언트에 초기 정보 전달  (완료) |
| 11 / 16 |
| 11 / 17 | 병합 작업 및 정기 회의 |
| 11 / 18 |  |
| 11 / 19 |
| 11 / 20 |  |
| 11 / 21 |  |
| 11 / 22 | 서버 객체 업데이트 함수 구현 및 클라 공유  SetUpBallon(), PlayerMove()(완료) |
| 11 / 23 |
| 11 / 24 | 병합/동기화 작업 및 정기 회의 |
| 11 / 25 | 충돌함수 점검(완료) |
| 11 / 26 |  |
| 11 / 27 |  |
| 11 / 28 | CS\_EVENT값 변경 후 게임 적용 확인.  CollisionPlayer() 구현 및 동기화 |
| 11 / 29 |
| 11 / 30 |
| 12 / 1 | 병합/동기화 작업 및 정기 회의 |
| 12 / 2 | 나머지 객체간 충돌함수 구현 및 동기화 |
| 12 / 3 |
| 12 / 4 |  |
| 12 / 5 |  |
| 12 / 6 | 최종 테스트 및 수정, 마무리 |
| 12 / 7 |
| 12 / 8 | 제출 |

**민식**

|  |  |
| --- | --- |
| 11 / 09 | 플레이어 생성 함수 구현  (완료) |
| 11 / 10 |
| 11 / 11 |
| 11 / 12 |
| 11 / 13 |
| 11 / 14 |
| 11 / 15 |
| 11 / 16 |
| 11 / 17 | 병합 작업 및 정기 회의 |
| 11 / 18 | recvThread() - SC\_GAMEINFO 패킷 전송 구현  (완료) |
| 11 / 19 |
| 11 / 20 |  |
| 11 / 21 | ~~SendClientEvent() - CS\_EVENT 패킷 전송 구현~~  UpdateThread() 프레임 워크 구현  (완료) |
| 11 / 22 |
| 11 / 23 |
| 11 / 24 | 병합/동기화 작업 및 정기 회의 |
| 11 / 25 | ~~recvThread() 함수 구현 클라이언트에게 받은 데이터 CLIENT\_INFO에 저장~~  RecvData() – SC\_PLAYERUPDATE 수신 구현(완료) |
| 11 / 26 |
| 11 / 27 |  |
| 11 / 28 | 클라이언트로 부터 전달받은 파괴 블록 인덱스 제거,  Player\_Collision()함수 구현 |
| 11 / 29 |
| 11 / 30 |
| 12 / 1 | 병합/동기화 작업 및 정기 회의 |
| 12 / 2 | Use\_Needle() - 플레이어와 아이템 충돌 확인 후 충돌 시 클라이언트 객체에 아이템 보유 여부 업데이트 |
| 12 / 3 |
| 12 / 4 |  |
| 12 / 5 |  |
| 12 / 6 | 최종 테스트 및 수정, 마무리 |
| 12 / 7 |
| 12 / 8 | 제출 |