**네트워크 게임프로그래밍**

**추진계획서**

2015180034 임동주

2015180037 정민수

2015182001 강동균

**목차**

애플리케이션 기획

개발환경

High-Level 디자인

1. 매치메이킹

2. 게임플레이

3. 통신 과정

Low-Level 디자인

팀원 별 역할분담

개발 일정

**애플리케이션 기획**

탑뷰 형식의 2D게임

물풍선을 이용하여 상대를 공격해 모두를 제거하면 승리하는 게임

블록을 부수고 아이템을 얻어 캐릭터의 풍선 개수, 크기, 이동속도를 높일 수 있음

일정 시간마다 플레이어가 담당하는 캐릭터의 변화가 있을 수 있어 전략적 선택 요구



<https://m.blog.naver.com/PostView.nhn?blogId=puck1001&logNo=220429458794&proxyReferer=https%3A%2F%2Fwww.google.com%2F>

**개발환경**

운영체제: Windows10 x64

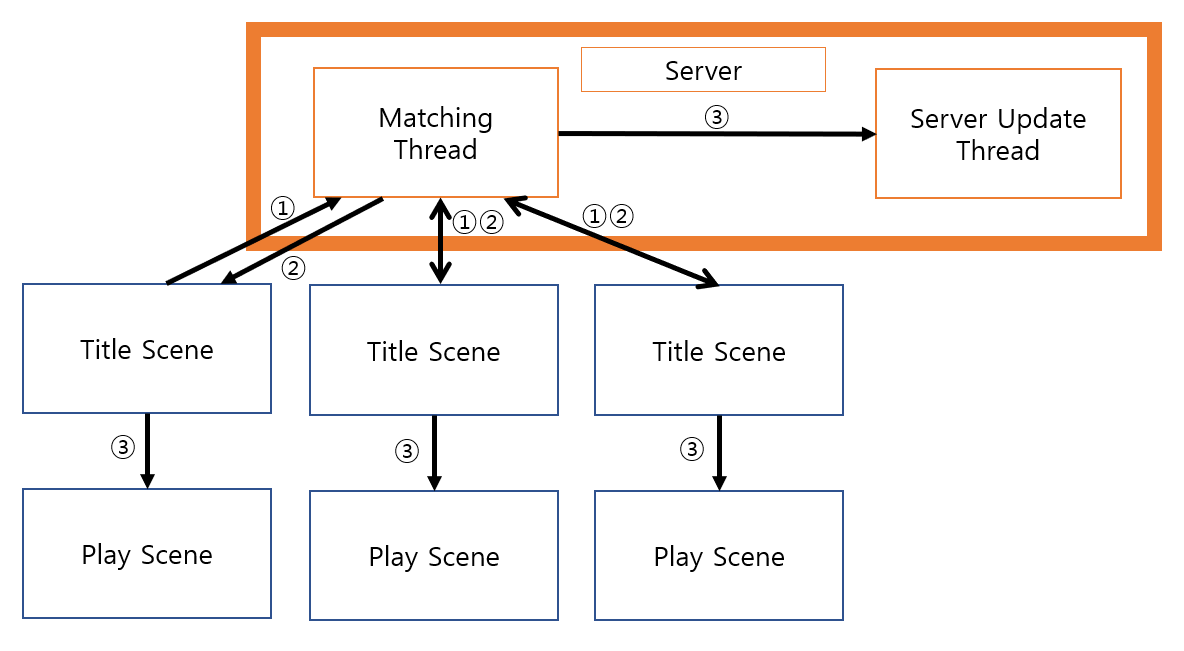
IDE: Visual Studio 2017

언어: C++, OpenGL, ws2\_32

SCM: GitHub

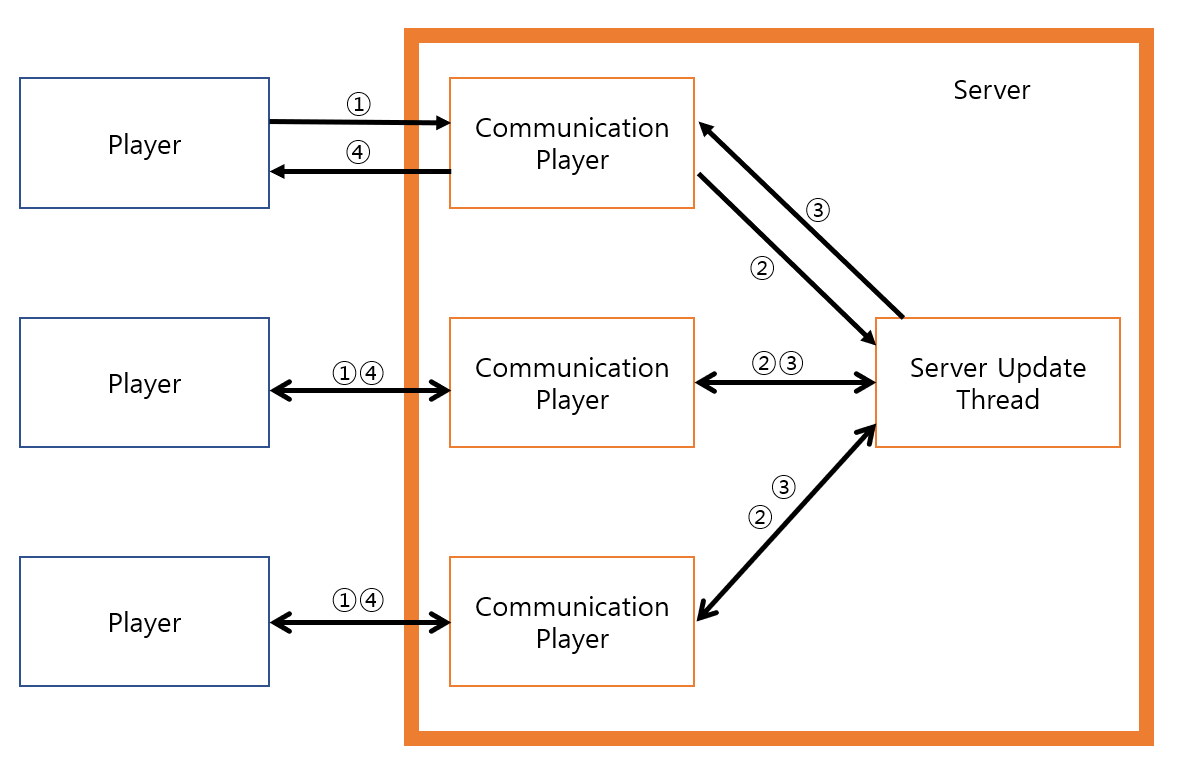
**High-level 디자인**

1. 매치메이킹

1. 최초 타이틀 화면에서 별도의 이동없이 [참가]를 누르면 Matching Thread에 Msg\_Ready를 보낸다. 이후 Mathcing Thread와 지속적으로 통신하며, Msg\_Ready를 보낸다. 참가를 대기하던 중 [참가 취소]를 누르면 Matching Thread에게 Msg\_ReadyCancel를 알려 준다.

2. 클라이언트의 Msg\_Ready 기다리다가, 메시지를 받으면 해당 클라이언트 소켓을 MatchingQueue에 추가한다. MatchingQueue에 추가되어 있는 클라이언트들에게 현재 MatchingQueue에 있는 ClientNum를 보내주며, 3개 이상의 클라이언트가 MatchingQueue에 들어올 때까지 해당 동작을 반복한다. 추가된 클라이언트와 통신 도중, Msg\_ReadyCancel이 들어오면, Msg\_ConfirmReadyCancel을 클라이언트에 보내주고 MatchingQueue에서 해당 클라이언트 소켓을 제거한다.

3. MatchingQueue의 3개 이상의 클라이언트가 대기하는 것이 확인된다면, 각 클라이언트에게 Msg\_PlayGame을 보낸다. 이와 동시에 해당 방을 관리할 서버 업데이트 스레드를 만들어 준 후, 서버 업데이트 스레드에게 3개의 클라이언트 소켓을 넘겨준다. 이후 MatchingQueue에서 방으로 이동한 클라이언트 소켓을 제거한 뒤, Matching Thread는 삭제 없이 클라이언트의 참가 요청을 기다린다.

2.게임플레이

1.. 각 Player 클라이언트는 서버의 Communication Player에게 자신의 KeyInput을 1초의 60번씩 해당 정보를 보낸다.

2. 1에서 들어온 정보를 그대로 Server Update Thread로 넘겨준다.

3. 2번에서 들어온 정보를 게임 로직에 의해 계산하고 변화가 있는 정보들에 대하여 Communication Player에게 돌려준다. 이 때, 변화가 있는 정보만 담을 것이므로, 고정-가변 데이터 통신을 이용한다. 고정데이터에 해당 데이터가 어떻게 담길 지에 대한 것을 4비트를 통해 정의한 후, 가변데이터에 변화가 있는 정보를 담아준다.

변화가 생길 수 있는 정보: CharacterStatus, MapType

4. 고정데이터에 담겨온 4비트를 통해 데이터를 분해한 뒤, 자신의 클라이언트에 반영한다.

3. 통신과정

데이터 구조체 흐름도 작성

서버 – 클라

매칭

플레이

Low-level 디자인

enum MSG\_MatchingSystem //매칭 대기 시스템에서 사용할 메시지

{

Msg\_Ready,

Msg\_ReadyCancel,

Msg\_ConfirmReadyCancel

Msg\_PlayGame

}

unsigned int ClientNum

vector<SOCKADDR\*> MatchingQueue

struct CharacterStatus

{

float x, y;

bool isAlive;

char whoseControl;

char speed;

char power;

char NumOfMaxAttack;

};

struct KeyInput

{

bool Up;

bool Down;

bool Left;

bool Right;

bool Attack;

bool Interact1;

};

enum MapBitComparison

{

EMPTY =0x0000,

WALL =0x0001,

ITEM1 =0x0010,

ITEM2 =0x0011,

ITEM3 =0x0100,

PLAYER1\_ATTACK =0x0101,

PLAYER2\_ATTACK =0x0110,

PLAYER3\_ATTACK =0x0111,

};

struct MapType

{

bool bit1;

bool bit2;

bool bit3;

bool bit4;

};

#define MAX\_PLAYER 3

class RoomGlobalData

{

private:

SOCKET m\_sockets[MAX\_PLAYER];

CharacterStatus m\_stats[MAX\_PLAYER];

KeyInput m\_keyInput[MAX\_PLAYER];

MapType m\_map[WIDTH][HEIGHT];

};

팀원 별 역할분담

임동주: 게임 내 통신부, 멀티스레드 함수, 리소스

정민수: 매칭시스템

강동균: 동기화,

개발일정

5일 작업, 1일 예비일, 1일 휴식

1-4주차

5주차 2인플레이어 -> 3인 플레이어

6주차 최종 발표 준비 및 예비주