네트워크 게임프로그래밍
추진계획서

2015180034 임동주
2015180037 정민수
2015182001 강동균

목차

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

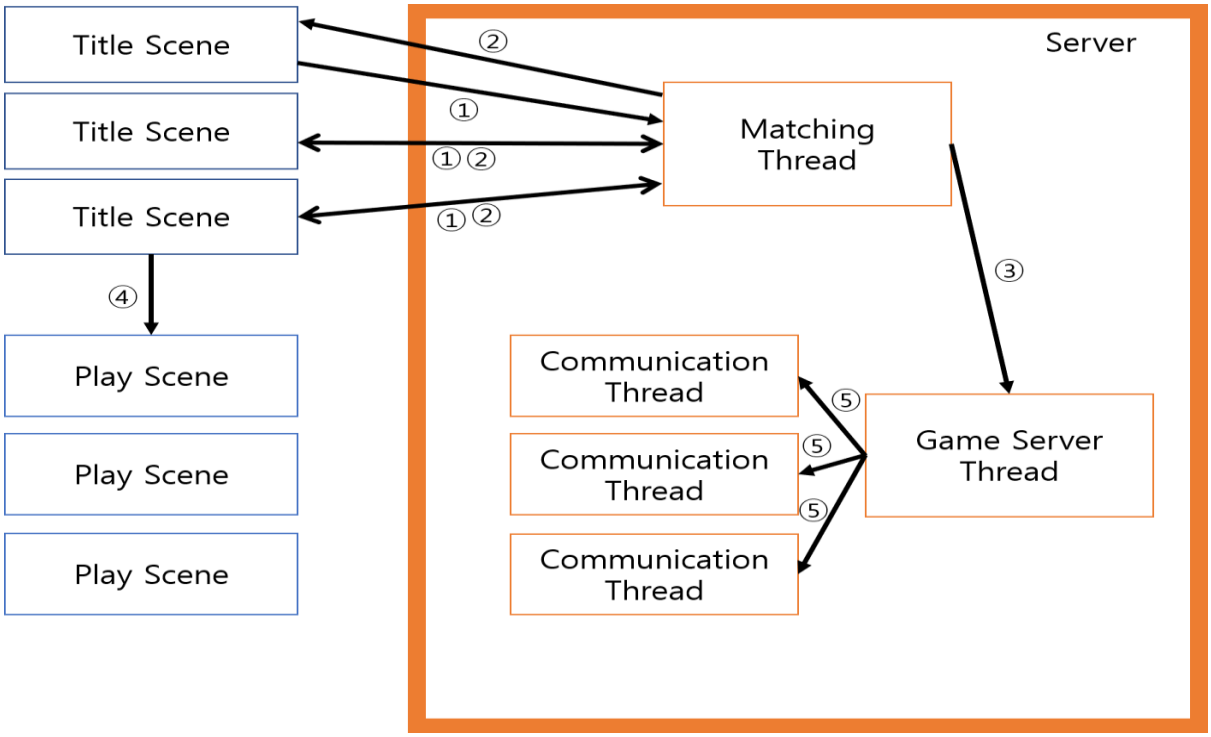
IDE: Visual Studio 2017

언어: C++, OpenGL, ws2_32

SCM: GitHub

High-level 디자인

1. 매치메이킹 [TCP 통신]



1. 타이틀 화면에서 [참가]를 누르면 서버에 Connect를 한 뒤, 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.

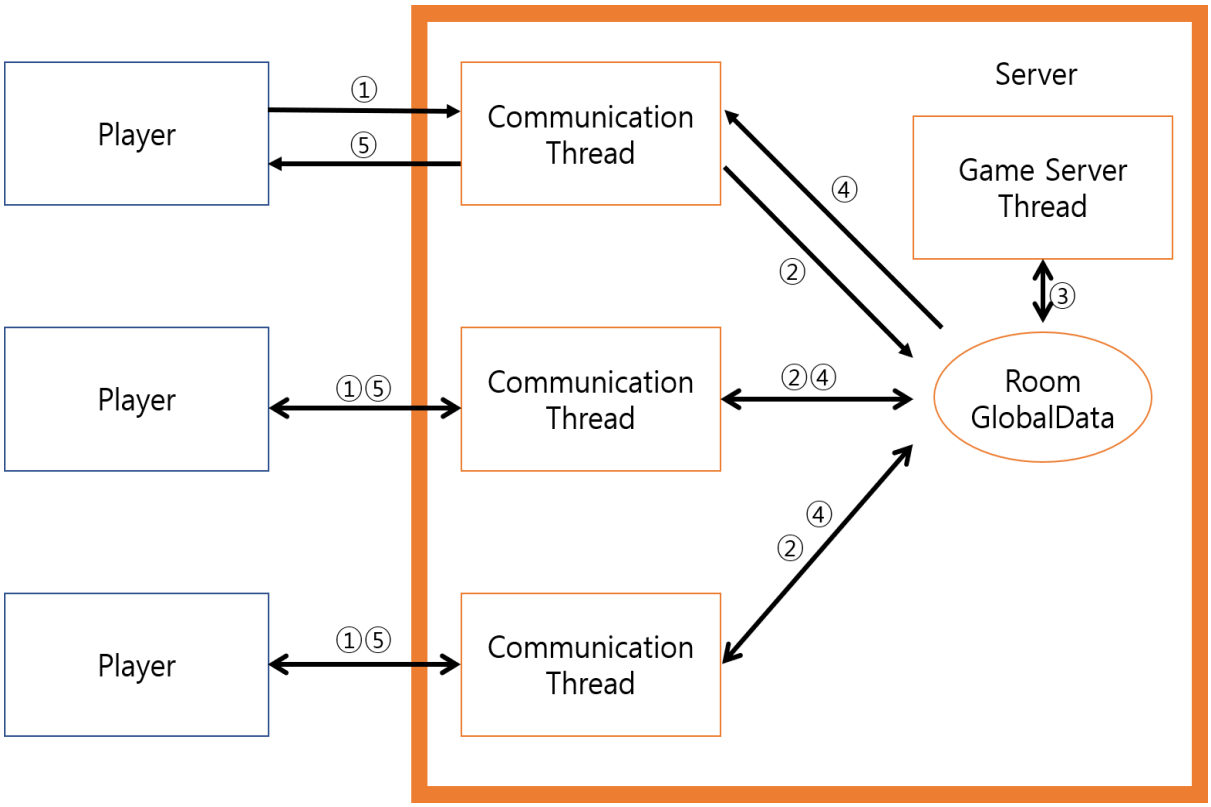
2. 클라이언트의 accpet를 기다리다가, 연결이 이루어지면 해당 클라이언트 소켓을 MatchingQueue에 추가한다. MatchingQueue에 추가되어 있는 클라이언트들에게 현재 MatchingQueue에 있는 ClientNum를 보내주며, 3개 이상의 클라이언트가 MatchingQueue에 들어올 때까지 해당 동작을 반복한다. 추가된 클라이언트와 통신 도중, Msg_ReadyCancel이 들어오면, Msg_ConfirmReadyCancel을 클라이언트에 보내주고 MatchingQueue에서 해당 클라이언트 소켓을 제거한다.

3. Mathcing Thread는 주기적으로 MatchingQueue에 3개 이상의 클라이언트가 있는지 확인한다. MatchingQueue의 3개 이상의 클라이언트가 대기하는 것이 확인된다면, 각 클라이언트에게 Msg_PlayGame을 보낸다. 이와 동시에 Game Server Thread를 만든다.

4. Msg_PlayGame을 받으면, Play Scene로 바꾼다.

5. Game Server Thread가 생성될 때 클라이언트와 통신할 Communication Thread도 만든다.

2.게임플레이 [TCP 통신]



1.. 각 Player 클라이언트는 서버의 Communication Thread에게 자신의 KeyInput을 보낸다.

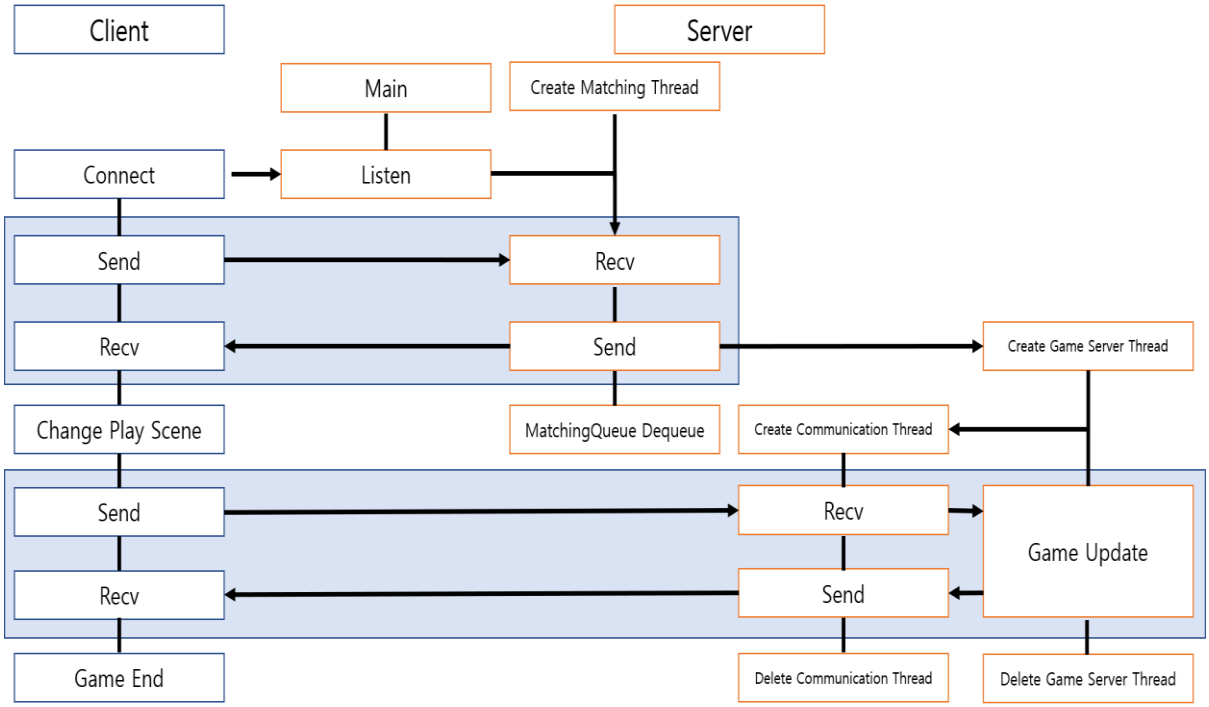
2. 1에서 들어온 정보를 RoomGlobalData에 있는 KeyInput에 각 플레이어에 맞게 갱신한다.

3. KeyInput을 이용하여 맵 및 캐릭터 정보를 연산한다. 이후 연산된 정보를 RoomGlobalData에 채워넣는다.

4. RoomGlobalData를 Communication Thread에서 읽어간다.

5. RoomGlobalData와 맵 정보 변화에 따라 MapType을 넣거나 제외한다. 맵 정보에 따라 보낼 데이터의 양이 가변적이므로, 고정부와 가변부로 나눠보낸다.

3. 통신과정



1. 서버가 업 앤 런 상태가 되면 Main과 Machting Thread 가 활성화된다

2. main: Client의 Connect 요청 Listen

3. MatchingThread: MatchingQueue에 main이 accept한 소켓 추가
   Client: MatchingThread에 Msg_Ready 전송

4. MatchingQueue가 게임 실행 인원 수가 되기 전까지 MatchingQueue에 ClientSocket 추가, 게임 실행 인원 수가 됐다면 Game Server Thread생성 후 MatchingQueue의 소켓들을 전달한 뒤, Client에게 Msg_PlayGame 전송.

5. Client: Play Scene 로드
   Matching Thread: MatchingQueue를 비우고 Game Server Thread에서 Client와 통신하기 위한 Communication Thread 생성

6. Client의 키입력 전송

7. 받은 데이터로 게임 업데이트 한 뒤, 게임 정보를 Client 에게 보내준다.

8. Client: Title Scene 로드
   Communication Thread와 Game Server Thread를 삭제

Low-level 디자인

Matching System [서버]

```
DWORD WINAPI MatchingThread();          //매칭시스템 관리 스레드로, 최초 프로세스 시작 시 서버와 함께 실행된다.
vector<SOCKADDR> MatchingQueue          //대기 중인 클라이언트 소켓을 저장하는 전역 변수
enum MSG_MatchingSystem                 //매칭 대기 시스템에서 사용할 메시지
{
    Msg_Ready,                          //클라이언트가 서버에게 대기중임을 알림
    Msg_ReadyCancel,                    //클라이언트가 서버에게 대기 취소를 알림
    Msg_ConfirmReadyCancel,             //서버가 클라이언트에게 대기 취소 받았음을 돌려줌
    Msg_PlayGame                        //서버가 클라이언트에게 게임이 시작 됐음을 알려줌
}
Unsigned char ClientNum                 //서버가 클라이언트에게 현재 MatchingQueue에서 대기중인 클라이언트 개수를 알려줌
bool isMatchingQueueFull(int MatchingQueueCount)
        //현재 MatchingQueue에서 대기하는 클라이언트의 개수가 3이상인지 파악
        3이상인 경우(true) MatchingQueue에 추가되어 있는 순서대로 3개의 클라이언트에게 Msg_PlayGame을 보내주고 CreateGameServerThread와 MatchingQueueDeQueue를 실행
        3미만인 경우(false) MatchingQueue에 추가되어 있는 클라이언트들에게 현재 대기 중인 클라이언트의 수(ClientNum)를 보냄
void CreateGameServerThread()
        //GameServerThread를 생성하며, MatchingQueue의 소켓들을 전달
void MatchingQueueDeQueue()
        //GameServerThread로 넘어간 MatchingQueue에 클라이언트 소켓을 제거
```

```
Game Server [서버]
DWORD WINAPI GameServerThread(LPVOID* arg)
        //Msg_PlayGame 을 받은 후 실행, Game Server Thread 생성
enum status
{
    dead, live
};
struct Player
{
   SOCKET clientSocket;
    KeyInput m_Key[MAX_PLAYER];          //클라이언트의 키입력 저장
    CharacterStatus m_stats;             //캐릭터들의 상태 저장
};
struct GameServerThreadData
{
    std::vector<SOCKET*> pClients;      //MatchingThread 에 넘겨받은 클라이언트 소켓
    std::vector<Player*> pPlayers;      //소켓 프로그래밍과 무관한 플레이어 개개인의 상태
    char        m_nCurrentPlayerAmount;  //현재 접속한 플레이어의 수
    bool        m_bMapChanged;           //맵이 바뀌었는 지 여부 알려줌
    CGameTimer    m_timer;               //업데이트에서 프레임시간 연산 시 사용
    MapType GAMEMAP[8][16]               //게임 연산에 쓰일 맵 정보

    void MakeCommunicationThread(void); //클라이언트와 통신할 스레드 생성
};
DWORD WINAPI ClientCommunicationThread(LPVOID* arg)
        //GameServerThread 를 생성 후 클라이언트와 통신할 스레드 생성
```

[클라이언트]

```
class GameTCPClient
{
private:
    WSADATA wsa;
    SOCKET sock;
    KeyInput m_keys;                                    //클라이언트의 키입력 저장
    CharacterStatus m_stats[MAX_PLAYER];                //캐릭터들의 상태 저장
    MapType          m_map[WIDTH][HEIGHT];              //게임 맵의 상태 저장
private:        //에러 발생 시 해당 내용 출력
    void err_quit(char* msg);
    void err_display(char* msg);
public:
    GameTCPClient();        //생성 시 윈속초기화와 소켓 생성, connect 를 수행
    ~GameTCPClient ();      //소멸 시 closesocket 과 윈속종료 수행
public:
    void TitleSceneSendData(enum MSG_MathcingSystem);
        //Msg_Ready 나 Msg_ReadyCancel 를 MatchingThread 에 전송
    int TitleSceneRecvData(int *GetClientNum);
        //MatchingThread 로부터 다음 메시지를 수령
        ClientNum: 매칭 대기 중인 상태일 때, 현재 MatchingQueue 에 대기 클라이언트 수를
        GetClientNum 의 저장
        Msg_ConfirmReadyCancel: Msg_ReadyCancel 전송 후, 서버 쪽에서 수신이 확인됐음을
        확인하고 연결 종료
        Msg_PlayGame: PlayScene 으로 전환

    void PlaySceneSendData();       //KeyInput m_keys 의 값을 Communication Thread 에 전송
    void PlaySceneRecvData();       //Game Server 에서 연산된 값을 Communication Thread 로부터
                                    고정-가변 데이터 형태로 수신, 고정은 char 형태이며, 가변은
                                    CharacterStatus m_stats, MapType  m_map 을 수신
    void UpdateClientData(CharacterStatus* recvCSData);
    void UpdateClientData(CharacterStatus* recvCSData, MapType* recvMTData);
         //서버에서 받은 데이터로 현재 클라이언트의 데이터를 업데이트
    void SetKeyInput(KeyInput* keyinput)
        //보내야 할 클라이언트의 키 값을 받아서 저장
};
```

공통 사용 구조체

```
const int MAX_PLAYER 3                    //최대 플레이어는 3
class RoomGlobalData     //CommunicationThread와 클라이언트 간의 소켓, 게임 업데이트의
                         입력 및 출력에 쓰이는 데이터
{
private:
    SOCKET          m_sockets[MAX_PLAYER];      //통신하고 있는 소켓들을 저장
    CharacterStatus m_stats[MAX_PLAYER];        //캐릭터 상태 저장
    KeyInput        m_keyInput[MAX_PLAYER];     //각 클라이언트 키 입력 저장
    MapType         m_map[WIDTH][HEIGHT];       //맵 상태 저장
};
struct CharacterStatus                    //캐릭터 정보
{
    float x, y;                           //위치
    bool isAlive;                         //생존여부
    char whoseControl;                    //담당 플레이어 표시
    char speed;                           //캐릭터 속도
    char power;                           //물풍선의 줄기 크기
    char NumOfMaxAttack;                  //물풍선을 놓을 수 있는 개수
};
struct KeyInput
{
    bool Up;                              //↑
    bool Down;                            //↓
    bool Left;                            //←
    bool Right;                           //→
    bool Attack;                          //폭탄 놓는 키 space
};
enum MapBitComparison
{
    EMPTY          =0x0000,               //비어있는 곳
    WALL           =0x0001,               //벽
    ITEM1          =0x0010,               //물풍선 개수 증가
    ITEM2          =0x0011,               //물풍선 줄기 크기 증가
    ITEM3          =0x0100,               //캐릭터 속도 증가
    PLAYER1_ATTACK =0x0101,               //1P 캐릭터의 물풍선
```

```
    PLAYER2_ATTACK    =0x0110,              //2P 캐릭터의 물풍선
    PLAYER3_ATTACK    =0x0111,                    //3P 캐릭터의 물풍선
};

struct MapType   //4비트 2개로 2칸을 표시, 맵 크기는 16x16이지만, 배열 크기는 8x16 사용
{
    bool bit1;
    bool bit2;
    bool bit3;
    bool bit4;
};
```

팀원 별 역할분담

임동주: Communication Thread 및 Game Server Thread,

정민수: Server main, Matching Thread

강동균: Client Title, Play Scene 소켓 프로그래밍, Client Rendering

개발일정

<table>
<tr><td></td><td>11/4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td></tr>
<tr><td>임동주</td><td colspan="4">리소스 획득</td><td rowspan="3">휴식</td><td rowspan="3">예비일</td><td rowspan="3">예비일</td></tr>
<tr><td>정민수</td><td colspan="4">서버 main 작성</td></tr>
<tr><td>강동균</td><td colspan="4">Client Title, Play Scene 제작</td></tr>
</table>

<table>
<tr><td></td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td></tr>
<tr><td>임동주</td><td colspan="4">서버 Update 통신부 제작</td><td rowspan="3">휴식</td><td rowspan="3">예비일</td><td rowspan="3">예비일</td></tr>
<tr><td>정민수</td><td colspan="4">서버 MathcingSystem 통신부 제작</td></tr>
<tr><td>강동균</td><td colspan="4">Client Title Scene 통신부 제작</td></tr>
</table>

<table>
<tr><td></td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td></tr>
<tr><td>임동주</td><td colspan="4" rowspan="2">Game Logic 작성</td><td rowspan="3">휴식</td><td rowspan="3">예비일</td><td rowspan="3">예비일</td></tr>
<tr><td>정민수</td></tr>
<tr><td>강동균</td><td colspan="4">Client Play Scene 통신부 제작</td></tr>
</table>

<table>
<tr><td></td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>12/1</td></tr>
<tr><td>임동주</td><td rowspan="3">중간회의</td><td colspan="3" rowspan="3">중간회의 이후 변경사항 적용</td><td rowspan="3">휴식</td><td rowspan="3">예비일</td><td rowspan="3">예비일</td></tr>
<tr><td>정민수</td></tr>
<tr><td>강동균</td></tr>
</table>

<table>
<tr><td></td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr><td>임동주</td><td colspan="4" rowspan="3">플레이어 수 증가 적용(2인 → 3인)</td><td rowspan="3">휴식</td><td rowspan="3">예비일</td><td rowspan="3">예비일</td></tr>
<tr><td>정민수</td></tr>
<tr><td>강동균</td></tr>
</table>

<table>
<tr><td></td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td>임동주</td><td rowspan="3">예비일</td><td rowspan="3">예비일</td><td colspan="5" rowspan="3">프로젝트 마감</td></tr>
<tr><td>정민수</td></tr>
<tr><td>강동균</td></tr>
</table>