Project Progress Report

제출 일자 : 12월 14일
과 목 명 : 네트워크 게임 프로그래밍
담당 교수 : 김재경 교수님
학 과 : 게임공학과
팀 원 : 2017182034 임보배
2019182050 이정선
2020180051 임성규

목차

1. 애플리케이션 기획
2. 개발환경
3. HIGH LEVEL 디자인
4. LOW LEVEL 디자인
5. 역할 분담
6. 개발 일정
7. 최종 결과물

1. 애플리케이션 기획

개요
임보배 학생이 2019년 1학기 윈도우 프로그래밍 과목에서 2인 팀프로젝트 기말과제로 제작한 1인 전략 시뮬레이션 게임입니다.

게임 이름
SHAOS

장르
실시간 전략시뮬레이션(RTS)

개발환경
- 언어 : C++, Window API
- IDE : Visual Studio

게임 진행

- 적 기지와 플레이어 기지 중앙에는 타워가 있고, 기지 사이는 길로 연결되어 있습니다.
- 플레이어의 타워가 파괴되기 전에 적의 타워를 파괴하면 승리합니다.
- 플레이어를 조작하여 게임을 진행합니다.
- 길에는 적의 포탑 3개가 놓여있습니다.
- 타워에서는 유닛이 생성되고, 유닛은 상대 기지의 타워를 파괴하러 나아갑니다.

맵
그림과 같은 형태의 전체 맵이 존재하고, 실제 게임을 진행할 때 전체 맵의 일부만 화면에 출력합니다.

시점
화면에 출력되는 시점은 플레이어를 기준으로 이동합니다.

플레이어

육각형 모양. 중심에 있는 원에서 공격이 나갑니다. (3인 게임으로 개발)

이동 키

WASD키보드를 이용하여 이동합니다.

공격 방법

플레이어의 공격 범위 내의 적을 클릭하면 타겟팅이 됩니다.
타겟팅이 된 적에게는 일정 시간마다 자동으로 공격이 나갑니다.

스킬

공격 스킬 : 공격의 방향을 조준해서 데미지가 큰 공격을 합니다.
방어 : 플레이어 주변에 방어막이 생성되어 공격 피해를 입지 않습니다.
광역기 : 플레이어로부터 일정 거리에 있는 적들이 모두 피해를 입습니다.
귀환 : 기지로 이동합니다.

포탑

총 3개의 포탑이 존재하고, 적 기지에 가까운 포탑일수록 공략이 어렵습니다.
1단계 : 일정 시간마다 범위 내의 적에게 총알을 발사합니다.
2단계 : 공격 속도가 1단계의 2배입니다.
3단계 : 공격 속도는 1단계와 같고, 폭탄을 발사합니다.

특징

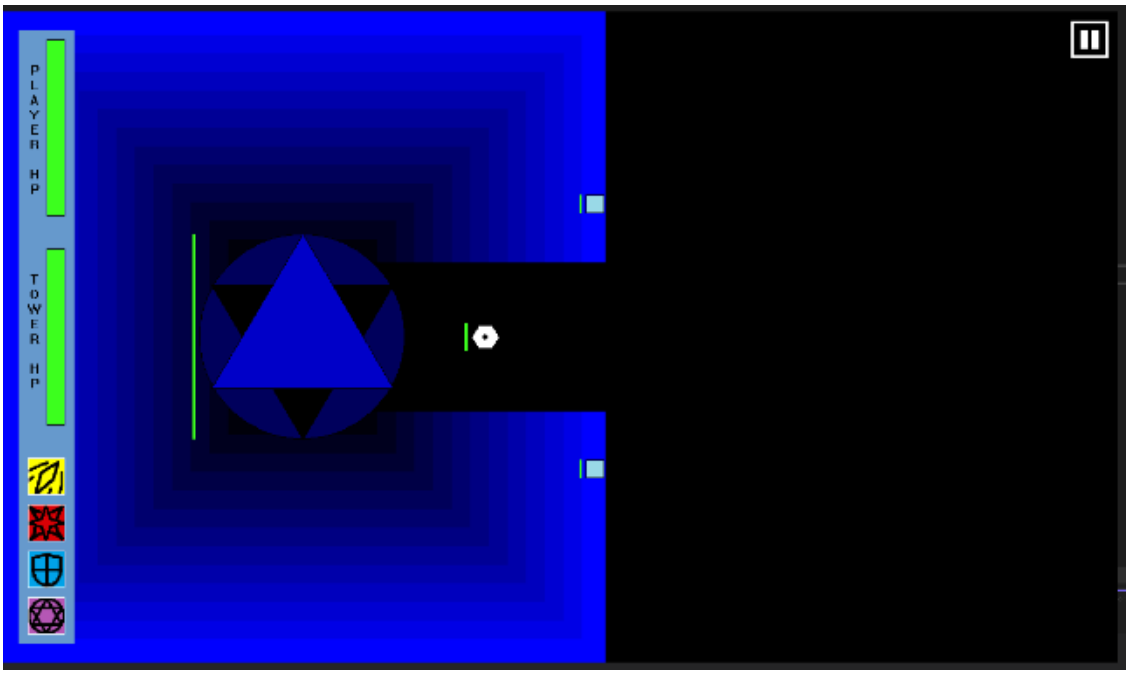
유닛에게 들어가는 데미지는 적지만, 플레이어에게 들어가는 데미지는 큽니다.

유닛

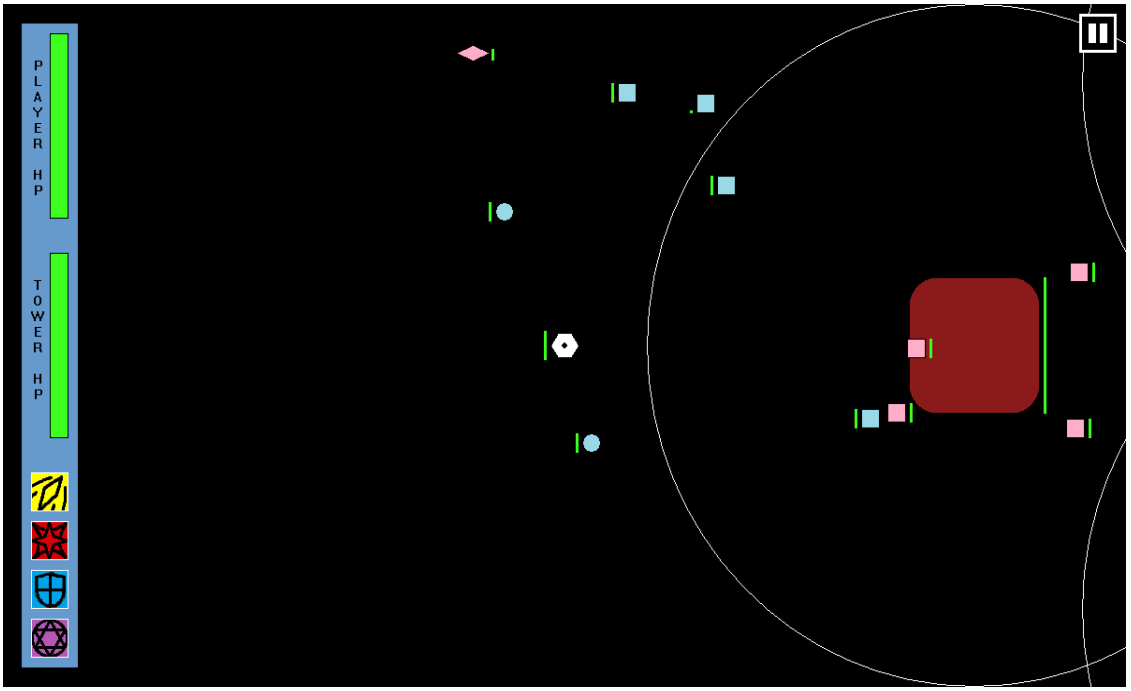
각 기지의 타워 옆에서 생성되며, 상대편 타워를 향해 이동합니다.
유닛의 종류는 4가지가 있습니다.
Rect : 근거리 공격 유닛
Ellip : 원거리 공격 유닛
Dia : 타워 공격 전용 유닛
Tri : 플레이어 공격 전용 유닛

게임 플레이 이미지

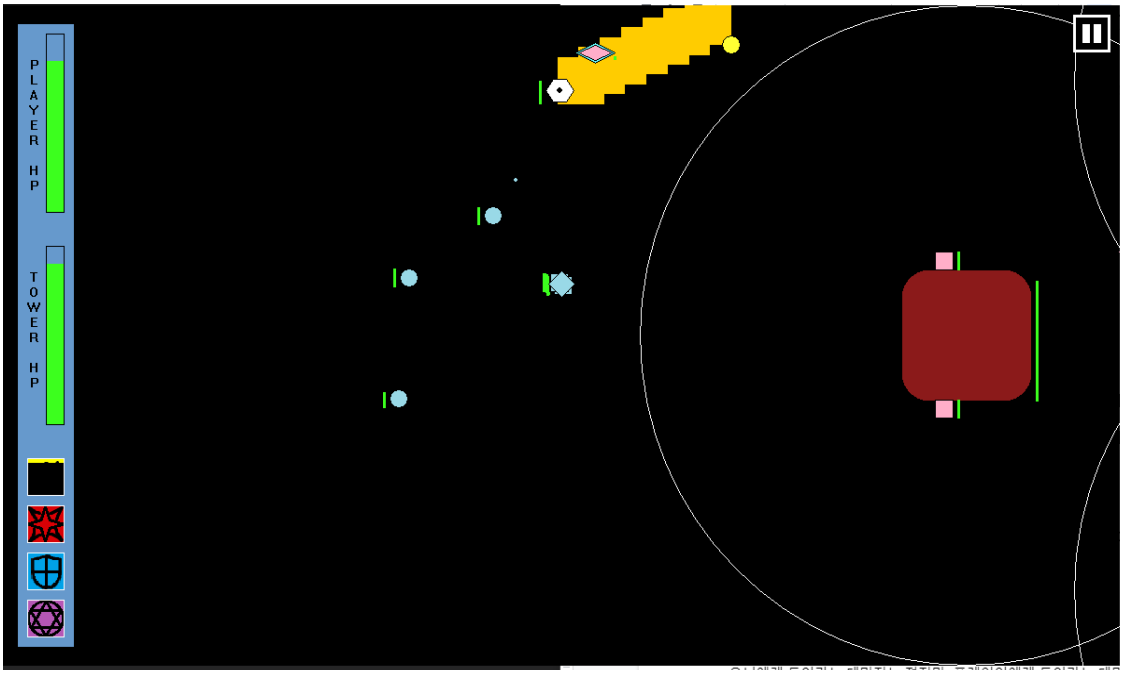
- 게임 시작 시 플레이어의 리스폰(타워)
아군은 파랑, 적은 빨강으로 표현됩니다.



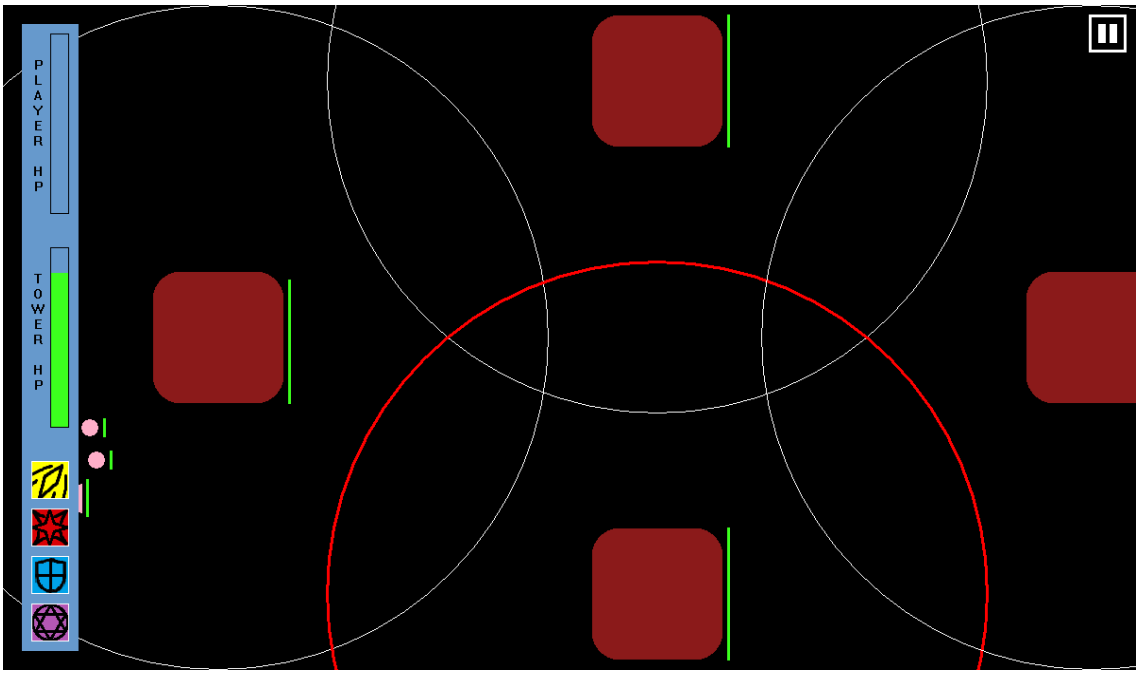
- 아군유닛과 상대유닛



- 플레이어의 스킬공격



- 포탑들에게 타켓팅 되어 죽은 플레이어



2. 개발환경

(1) C/C++

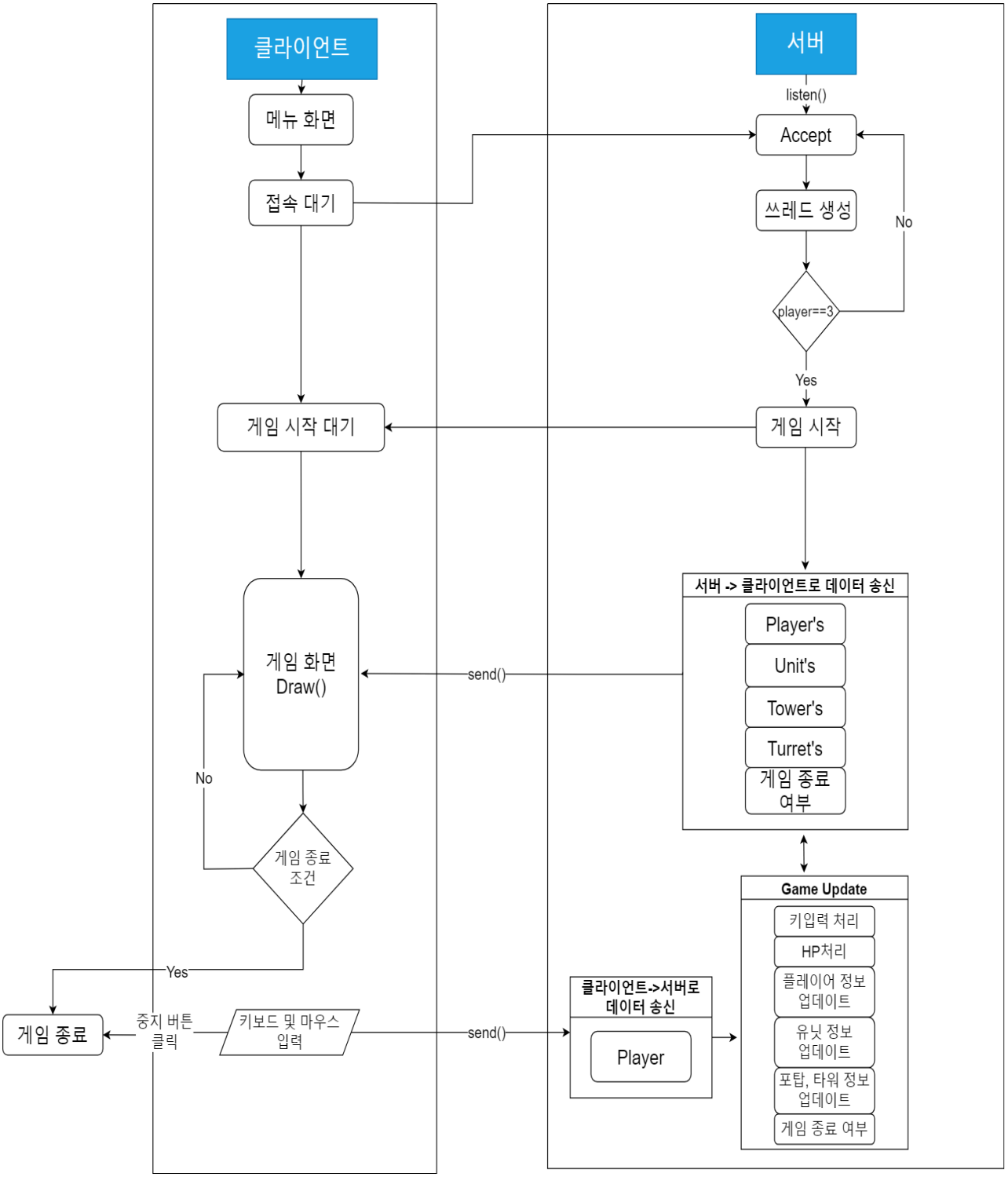
(2) Windows API

(3) Visual Studio

(4) GitHub

3. High-level 디자인

-클라이언트-서버 간 단계별 관계 흐름도



-설명

1. 클라이언트가 게임 플레이 버튼을 누르면 서버에 접속합니다.

2. 서버에서 3명의 플레이어가 모두 접속하기 전까지 클라이언트는 게임 화면에서 대기합니다.

3. 서버는 클라이언트가 3명 접속하게 되면, 게임 월드 객체들에 대한 Update연산을 시작합니다.

4. 서버는 각 클라이언트에게 갱신된 플레이어들의 정보, 유닛 정보, 타워 정보, 포탑 정보를 전송합니다.

5. 클라이언트는 서버에서 받은 정보들을 포맷한 후 화면을 렌더링 합니다.

6. 클라이언트는 서버에게 현재 입력된 키보드와 마우스 입력을 서버에게 전송합니다.

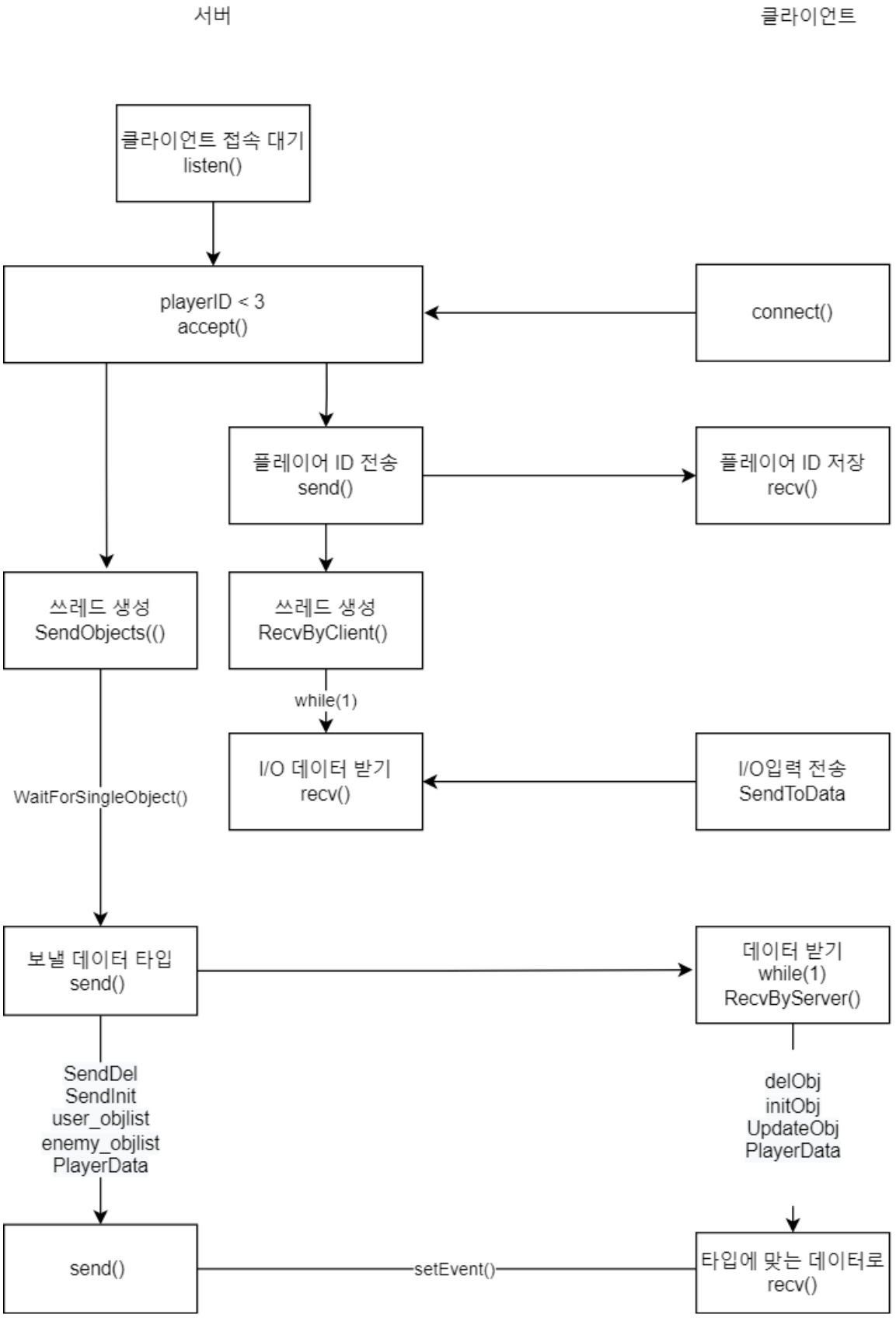
7. 서버는 클라이언트에게 받은 정보에 따라 플레이어의 기본 데이터를 변경하고, 이를 반영하여 게임 월드 전체의 Update() 연산을 합니다.

8. 이후 서버는 게임 종료 조건을 판단하고, 종료 조건에 해당될 시 클라이언트에게도 게임이 종료되었음을 알려준 업데이트를 멈추게 됩니다.

9. 4. 부터 반복합니다.

10. 게임이 종료될 시 클라이언트는 승리 화면을 그려줍니다.

-클라이언트-서버 간 네트워크 구조



4. Low-level 디자인

네트워크 송수신 구현

서버 수신 방식 **:** TCP/IP 고정 길이 전송방식
서버 송신 방식 **:** TCP/IP 고정 길이 + 가변길이 전송방식
고정 데이터 **:** 보낼 데이터의 크기

1. 클라이언트의 네트워크 구조
1) 송수신 데이터

```
struct SendData {
        int playerID;    // 플레이어 아이디

        // 키입력
        UINT message;
        WPARAM wParam;
        LPARAM lParam;

        // 플레이어 시점의 화면
        int iViewX;
};

enum RecvType {
        initObj, delObj, UpdateObj, PlayerData        // 데이터 타입
};

struct RecvInitObj {
        int key;                // 해당 오브젝트의 key
        int objTAG;             // 해당 오브젝트의 type
/*
-#include “framework.h”
#define         OPLAYER         0
#define         OTOWER          1
#define         OTURRET         2
#define         ODIA            3
#define         OELLIP          4
#define         ORECT           5
#define         OTRI            6
*/
        TEAM team;              // 해당 오브젝트가 존재하는 팀
        POINTFLOAT mptpos;      // 해당 오브젝트의 생성 위치
};

struct RecvDelObj {
        int key;                // 해당  오브젝트의 key
        TEAM team;              // 해당 오브젝트가 존재하는 팀
```

```
};

struct RecvUpdateObj {              // 오브젝트의 기본 변수들
        int key;
        TEAM team;

        POINTFLOAT          mptpos;

        RECT                mrcRng;

        CHp                 mhp;
        RECT                mrchpbar;
        BOOL                mdeath;

        INT iattackcooltime;
        INT ideatheffecttime;
};

struct addRecvPlayerInfo { // 오브젝트의 변수들 중 플레이어가  추가로 들고 있는 변수
        BOOL R_On, L_On, U_On, D_On;
        BOOL pressQ, pressSft, onshield;


        UINT effecttime_AoE;
        UINT effecttime_Shoot;
        UINT effecttime_Return;

        UINT castingtime_return;
        UINT activetime_shield;

        UINT cooltime_Shoot;
        UINT cooltime_AoE;
        UINT cooltime_Shield;
        UINT cooltime_Return;
        UINT cooltime_Death;

        int       ptarget_key;

        POINT worldmousepos;

        BOOL immotal;

        POINT triangle1[3];
        POINT triangle2[3];
        POINT shootattackpt[7];
        RECT shootattackrange[7];
};
```

2) CNetwork 클래스
:멤버변수
map<int, CGameObject*>& user_objlist;
map<int, CGameObject*>& enemy_objlist;

:멤버**, static** 함수
CNetwork(map<int, CGameObject*>& user_objlist, map<int, CGameObject*>& enemy_objlist);
~CNetwork();
void SendToData(UINT, WPARAM, LPARAM, int); // 클라이언트의 키입력 전송
static DWORD WINAPI RecvByServer(LPVOID arg); // 서버의 오브젝트 데이터들 수신

2. 서버의 네트워크 구조

1) 송수신 데이터

```
struct RecvData {
        int playerID;    // 플레이어 아이디

        // 키입력
        UINT message;
        WPARAM wParam;
        LPARAM lParam;

        // 플레이어 시점의 화면
        int iViewX;
};

enum SendType {     // 송신 데이터의 타입
        initObj, delObj, UpdateObj, PlayerData
};

struct SendInitObj { // 생성할 데이터 전송
        int key;
        int objTAG;     // 객체 구분을 위한 일련 번호
/*
-#include “framework.h”
#define         OPLAYER         0
#define         OTOWER          1
#define         OTURRET         2
#define         ODIA            3
#define         OELLIP          4
#define         ORECT           5
#define         OTRI            6
*/
        TEAM team;
        POINTFLOAT mptpos;
};

struct SendDelObj { // 삭제할 데이터 전송
        int key;
        TEAM team;
};

struct SendUpdateObj {      // 기본 오브젝트의 데이터 전송
        int key;
        TEAM team;

        POINTFLOAT          mptpos;

        RECT                mrcRng;
```

```
        CHp             mhp;
        RECT            mrchpbar;
        BOOL            mdeath;

        INT iattackcooltime;
        INT ideatheffecttime;
};

struct addSendPlayerInfo { // 기본 오브젝트를 상속받은 플레이어의 추가 데이터 전송
        // 키 입력
        BOOL R_On, L_On, U_On, D_On;
        BOOL pressQ, pressSft, onshield;
        UINT effecttime_AoE;
        UINT effecttime_Shoot;
        UINT effecttime_Return;

        UINT castingtime_return;
        UINT activetime_shield;

        UINT cooltime_Shoot;
        UINT cooltime_AoE;
        UINT cooltime_Shield;
        UINT cooltime_Return;
        UINT cooltime_Death;

        int     ptarget_key;

        POINT worldmousepos;

        BOOL immotal;

        // 유닛
        POINT triangle1[3];
        POINT triangle2[3];
        POINT shootattackpt[7];
        RECT shootattackrange[7];
};
```

2) CNetwork 클래스
:멤버**, static** 변수

```
map<int, CGameObject*>& user_objlist;
map<int, CGameObject*>& enemy_objlist;

static SOCKET listen_sock;          // 서버 소켓
static SOCKET client_sock[3];       // 세 개의 클라 소켓
static HANDLE hSendEvent;           // 이벤트
```

:멤버**, static** 함수

```
CNetwork(map<int, CGameObject*>& user_objlist, map<int, CGameObject*>&
enemy_objlist);          // 생성자
~CNetwork();             // 소멸자

static DWORD WINAPI RecvByClient(LPVOID arg);          // 클라이언트의 키 입력을
받는다
void SendDel(SendDelObj data);                         // 클라이언트에게 삭제 명령
void SendInit(SendInitObj data);                       // 클라이언트에게 생성 명령
static DWORD WINAPI SendObjects(LPVOID arg);           // 오브젝트 데이터 전송
```

(3) 함수

// 서버의 Main쓰레드 역할을 하는 함수이자 네트워크 클래스 생성자이다.

```
CNetwork::CNetwork(map<int, CGameObject*>& user_objlist, map<int,
CGameObject*>& enemy_objlist)
{
        …
        hSendEvent = CreateEvent(NULL, FALSE, TRUE, NULL);  // 이벤트 생성
        …
        for (int playerID = 0; playerID < 3; ++playerID) {      // 클라이언트 3개 접속
                // accept()
                addrlen = sizeof(clientaddr);
                client_sock[playerID] = accept(listen_sock, (struct sockaddr*)&clientaddr,
                &addrlen);
                if (client_sock[playerID] == INVALID_SOCKET)
                        err_display("accept()");
                …
        // 플레이어 아이디 전송
        retval = send(client_sock[playerID], (char*)&playerID, sizeof(playerID), 0);
                if (retval == SOCKET_ERROR)
                        err_display("send()");

        // 키입력 받는 스레드 생성
        hThread = CreateThread(NULL, 0, RecvByClient, (LPVOID)client_sock[playerID], 0,
        NULL);
        …
}
```

// 네트워크 객체의 소멸자

- **CNetwork::~CNetwork()**

```
{
        closesocket(listen_sock); // 소켓 닫기
        WSACleanup(); // 윈속 종료
}
```

// 클라이언트 데이터를 수신 및 처리 하는 함수

- **DWORD WINAPI CNetwork::RecvByClient(LPVOID arg)**

```
{
	RecvData tempData;
	while (true) {
	// 데이터 받기(파일 이름)
	retval = recv(client_sock, (char*)&tempData, sizeof(tempData), MSG_WAITALL);
		if (retval == SOCKET_ERROR)
			err_display("recv()");

		// 메시지 입력을 받으면 플레이어의 아이디와 키, 마우스 정보를 받는다
		switch (tempData.message) {
		case WM_KEYDOWN:
		case WM_KEYUP:
			g_GameFrameWork.MSG_Key(tempData.playerID,
tempData.message, tempData.wParam, tempData.lParam);
			break;
		case WM_LBUTTONDOWN:
		case WM_LBUTTONUP:
		case WM_RBUTTONDOWN:
		case WM_MOUSEMOVE:
			g_GameFrameWork.MSG_Mouse(tempData.playerID,
tempData.message, tempData.wParam, tempData.lParam, tempData.iViewX);
			break;
		}
	}
}

//삭제된 객체가 있을 때 바로 전송하는 함수
- void CNetwork::SendDel(SendDelObj data)
{
	// 클라에게 삭제 명령을 하는 데이터 타입을 보낸다.
	int type = SendType::delObj;

	for (int i = 0; i < 3; ++i) {
	int retval = send(client_sock[i], (char*)&type, sizeof(type), 0);
	if (retval == SOCKET_ERROR)
	err_display("send()");

	// 클라에게 삭제 명령을 하는 데이터를 보낸다.
	retval = send(client_sock[i], (char*)&data, sizeof(data), 0);
	if (retval == SOCKET_ERROR)
		err_display("send()");
	}
}

// 새로 생성된 객체가 생길 때 바로 송신하는 함수
- void CNetwork::SendInit(SendInitObj data)
{
	int type = SendType::initObj;
```

```
// 클라의 생성 데이터 타입을 보낸다.
	for (int i = 0; i < 3; ++i) {
		int retval = send(client_sock[i], (char*)&type, sizeof(type), 0);
		if (retval == SOCKET_ERROR)
			err_display("send()");

	// 클라의 생성 데이터를 보낸다.
	retval = send(client_sock[i], (char*)&data, sizeof(data), 0);
		if (retval == SOCKET_ERROR)
			err_display("send()");
	}
}

//매 프레임 마다 클라이언트로 객체들을 전송하는 송신 쓰레드
- DWORD WINAPI CNetwork::SendObjects(LPVOID arg)
{
	WaitForSingleObject(hSendEvent, INFINITE);	// 전송 완료 대기
	…
	// 유저 리스트의 타입을 보낸다
	int type = SendType::UpdateObj;
	retval = send(client_sock, (char*)&type, sizeof(type), 0);
	if (retval == SOCKET_ERROR)
		err_display("send()");

	// 유저의 데이터 크기를 보낸다
	int userDataSize = Network->user_objlist.size();
	retval = send(client_sock, (char*)&userDataSize, sizeof(userDataSize), 0);
	if (retval == SOCKET_ERROR)
		err_display("send()");

	// 유저리스트를 보낸다
	for (auto tmp : Network->user_objlist) {
		SendUpdateObj temp;
		…
	retval = send(client_sock, (char*)&temp, sizeof(temp), 0);
	if (retval == SOCKET_ERROR)
		err_display("send()");
	}
}
	type = SendType::UpdateObj;
	retval = send(client_sock, (char*)&type, sizeof(type), 0);
	if (retval == SOCKET_ERROR)
		err_display("send()");

	userDataSize = Network->enemy_objlist.size();
	retval = send(client_sock, (char*)&userDataSize, sizeof(userDataSize), 0);
	if (retval == SOCKET_ERROR)
```

```
		err_display("send()");

// 적팀리스트를 보낸다
for (auto tmp : Network->enemy_objlist) {
		type = SendType::UpdateObj;
			…
retval = send(client_sock, (char*)&temp, sizeof(temp), 0);
		if (retval == SOCKET_ERROR)
			err_display("send()");
}

// 플레이어 데이터 전송
type = SendType::PlayerData;

retval = send(client_sock, (char*)&type, sizeof(type), 0);
if (retval == SOCKET_ERROR) {
		err_display("send()");
		//break;
}
for (int j = 0; j < 3; ++j) {
		addSendPlayerInfo temp;
		…
// 임시 객체로 저장한 플레이어 데이터 전송
retval = send(client_sock, (char*)&temp, sizeof(temp), 0);
if (retval == SOCKET_ERROR)
		err_display("send()");
}
```

3.멀티스레드 기능
(1) 스레드 핸들
HANDLE hThread;

(2) 이벤트 핸들
static HANDLE hSendEvent;

(3) 스레드 생성
// 키 입력 받는 스레드
CreateThread(NULL, 0, RecvByClient, (LPVOID)client_sock[playerID], 0, NULL);

(4) 스레드 동기화
```
DWORD WINAPI CNetwork::SendObjects(LPVOID arg) {
        WaitForSingleObject(hSendEvent, INFINITE);        // 전송 완료 대기
        …
        유저 리스트, 적팀리스트, 플레이어 데이터를 전송
        …
        SetEvent(hSendEvent);        // 전송 완료 알림
}
```

4.추가 항목
-클라이언트 기존 내용 수정
1.클래스 CTeam의 멤버 변수 CGameObject* p_myobjlist{} 와 CGameObject* p_opponentobjlist{} 의 연결리스트를 변경했다. map<int, CGameObject*> user_objlist와 map<int, CGameObject*> enemy_objlist로 컨테이너를 사용하여 CWorld의 멤버 변수로 수정했다.
2. 플레이어를 3명으로 바꾸면서 user_objlist의 1~3번 키를 플레이어에게 고정시켜 놓았다.
3. 키 입력이 발생하면 서버로 키 입력을 넘겨준다.

5. 팀원 별 역할분담

팀원 명	역할	관련 함수

임성규	서버 송수신 멀티쓰레드 생성 및 종료
	서버 송신 데이터 적용
	서버 수신 데이터 적용
	클라이언트 구조 변경(list->map)
	기존 클라이언트 Update() 수정
		CNetwork();
		static DWORD WINAPI RecvByServer(LPVOID arg);
		static DWORD WINAPI RecvByClient(LPVOID arg);
		static DWORD WINAPI SendObjects(LPVOID arg);

이정선	서버 연결 함수
	서버 송신 데이터 정의
	서버 송신 데이터 적용
	서버 draw() 수정
	서버에 Shaos 클래스들 추가
		CNetwork();
		void SendInit(SendInitObj data);
		void SendDel(SendInitObj data);
		static DWORD WINAPI SendObjects(LPVOID arg);

임보배	클라이언트 송수신 함수 구현
	클라이언트 송신 데이터 적용
	클라이언트 수신 데이터 적용
	서버 main.cpp 테스트 버전
		클라이언트 class CNetwork{};
		void SendToData(UINT, WPARAM, LPARAM, int);
		static DWORD WINAPI RecvByServer(LPVOID arg);
		Shaos.cpp 텍스트 출력

6.개발일정

(일별**/**개인별 계획 수립**,** 달력 형태로 작성**)**

11월

	일	월	화	수	목	금	토
			1	2	3	4	5
임성규							
이정선							
임보배							계획서 수정
	6	7	8	9	10	11	12
임성규		계획서 수정			계획서 수정		
이정선		계획서 수정			계획서 수정		
임보배	계획서 수정				계획서 수정		
	13	14	15	16	17	18	19
임성규		기존 클라이언트 Update() 수정				기존 클라이언트 연산자 오버로딩 적용	
이정선	서버 draw() 수정, 클래스 추가						서버 연결함수 구현
임보배		Network.h 선언		git-fetch 후 서버 pull		Network.cpp 작성	git push
	20	21	22	23	24	25	26
임성규	static DWORD WINAPI RecvByClient(LPVOID arg); 서버로 수신된 키 입력 임시 데이터 적용						
이정선	서버 연결함수 ServerToClient Connect()						
임보배	Network 객체 생성 및 소멸(connect() & close())	네트워크 송신 함수 구현		데이터 매핑 문제 해결 방안 자료조사	테스트 서버 생성 및 임시 데이터 적용	테스트 서버에서 네트워킹 테스트 (임시 데이터)	git push 회의 - 송수신 데이터 추출 문제
	27	28	29	30			
임성규	static DWORD WINAPI SendObjects(LPVOID arg); 서버 멀티쓰레드 함수 구현						
이정선			CNetwork() 수정				
임보배	회의 - 추진 계획서 수정	클라이언트 쓰레드 생성 및 CNetwork 정리	클라이언트 ver2 생성 - 키 입력 메세지 송신 및 Update()적용	네트워크 송신 테스트			

12월

	일	월	화	수	목	금	토
					1	2	3
임성규						서버/클라이언트 데이터 송수신 테스트	
이정선						서버 송신 데이터 함수	
임보배						추가 송수신 데이터 추출 및 데이터 송수신 적용	윈도우 메세지로 전송 시도
	4	5	6	7	8	9	10
임성규	클라이언트 구조 변경 list->map		포인터 변수들을 일반 변수들로 변경	타켓을 키로 관리	구조 변경한 클라이언트를 서버에도 적용	RecvByClient 함수 재정의	SendObjects 함수 재정의
이정선	서버 송신 데이터 함수					서버 송신 데이터 함수	
임보배	client ver3 : 윈도우 메세지로 키 입력데이터 send()	마우스 메세지 송신 구현 및 클라이언트 수신 테스트 서버 생성	GUI WinMain(테스트 서버) 생성 후 수신 데이터 적용	생성한 서버에서 윈도우 메세지 전송 테스트 최종 확인	send용Thread와 Recv용 Thread 분리	테스트 서버에서 테스트	기존 클라이언트 변경 내용과 쓰레드 분리 내용 merge
	11	12	13				
임성규	서버를 옵저버로 활용	오류 디버깅	디버깅 및 보고서 작성				
이정선	서버 송신 데이터 함수		보고서 작성, 디버깅				
임보배	수신 데이터 적용 및 Frame - 갱신 시간 맞추기		release Test 및 디버깅 , 보고서 작성				

7. 최종 결과물