넷겜플 1팀 Progress Report

임윤수 / 김동재 / 서동우

nongbunongbu@kpu.ac.kr

**목차**

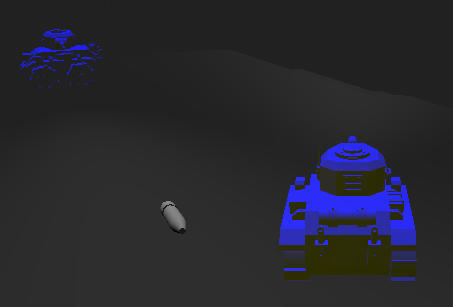
1. 게임 요소
2. 게임 소개
3. 하이 레벨 디자인
4. 로우 레벨 디자인
5. 변동사항
6. 일정표

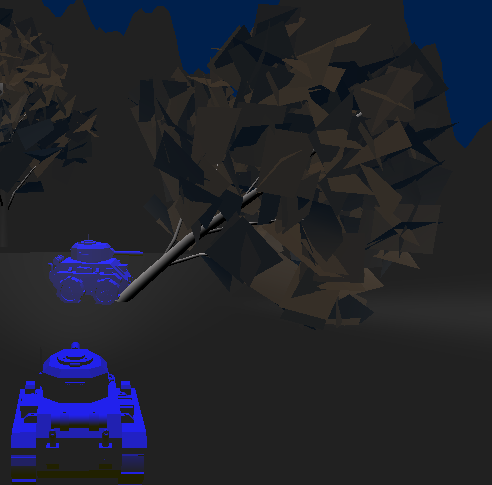
**게임 요소**

1. 장르 : 슈팅게임
2. 컨셉 : 1 vs 1
3. 플레이어 수 : 2인
4. 개발자 : 서동우, 3D게임프로그래밍 과제로 제작.

**게임 소개**

* 규칙 : 두 플레이어는 각자의 탱크를 조종하여 상대의 탱크를 파괴해야 함, 포탄에 피격당할 때마다 체력이 감소하며, 포탄에 10회 맞아 체력이 0이 된 플레이어는 패배하게 됨.
* 조작 : WASD(이동), 마우스(화면, 포탑 회전), SPACE(발사)





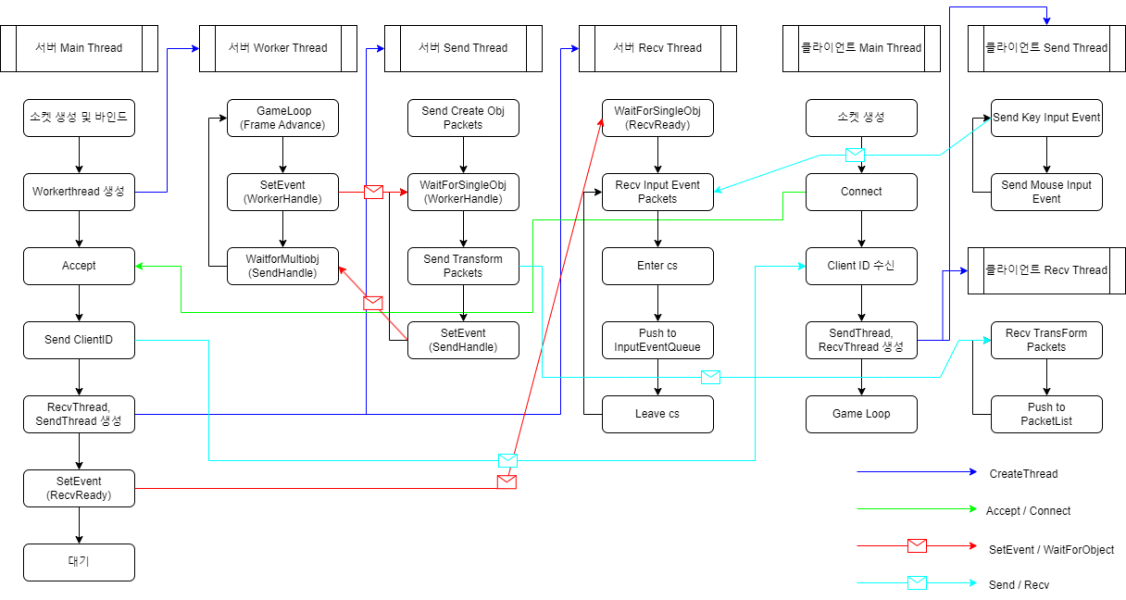
**하이 레벨 디자인**

초기

스크린샷, 텍스트, 그래픽, 도표이(가) 표시된 사진

자동 생성된 설명

최종



**로우** **레벨 디자인**

PORT, BUFSIZE

Object ID

constexpr int PLAYER = 0;

constexpr int BULLET = 1;

constexpr int TERRAIN = 2;

constexpr int TREE = 3;

constexpr int SERVERPORT = 9000;

constexpr int BUFSIZE = 256;

protocol.h에 정의된 상수

(서버와 클라이언트가 공유)

constexpr int SC\_CREATE\_OBJECT = 0;

constexpr int SC\_MOVE\_OBJECT = 1;

constexpr int SC\_DELETE\_OBJECT = 2;

constexpr int SC\_COLLISION\_OBJECT = 3;

constexpr int CS\_PLAYER\_INPUT = 4;

Packet ID

constexpr int KEY\_W = 0;

constexpr int KEY\_S = 1;

constexpr int KEY\_A = 2;

constexpr int KEY\_D = 3;

constexpr int KEY\_SPACE = 4;

constexpr int MOUSE\_LEFT = 10;

constexpr int MOUSE\_RIGHT = 11;

constexpr int RECV\_DONE = 100;

Key ID

struct sc\_object\_transform\_packet {

int object\_id;

XMFLOAT3 position;

XMFLOAT3 rotate;

XMFLOAT3 scale;

XMFLOAT4X4 matrix;

bool isActive;

};

struct sc\_create\_object\_packet {

int object\_type;

XMFLOAT4X4 matrix;

};

protocol.h에 정의된 패킷 구조체.

(서버와 클라이언트가 공유)

typedef struct event {

int client\_id;

int event\_id;

int state;

POINT mouseAxis;

} EVENT;

struct cs\_player\_input\_packet {

int input\_event;

};

struct sc\_collision\_object\_packet {

int object\_id1;

int object\_id2;

}

struct sc\_delete\_object\_packet {

int object\_id;

}

struct sc\_create\_object\_packet {

int object\_type;

XMVECTOR pos;

}

CRITICAL\_SECTION cs;

HANDLE hWorkerEvent[2];

HANDLE hSendEvent[2];

HANDLE hRecvReadyEvent;

SOCKET client\_socket[2];

deque<EVENT> InputEvent;

**서버의 전역 변수**

InputEvent 내부에 클라이언트에서 받은 인풋을 EVENT 형식으로 저장한다.

client\_socket 배열은 접속한 클라이언트의 소켓을 저장한다.

hWorkerEvent 배열은 sendthread가 workerthread의 작업 완료를 대기하기 위해 사용한다.

hSendEvent 배열은 workerthread가 sendthread의 작업 완료를 대기하기 위해 사용한다.

hRecvReadyEvent는 클라이언트가 모두 접속했을 시 recvthread의 작업 시작을 위해 사용한다.

cs는 InputEventQueue를 동기화하기 위해서 사용된다.

void HandleInputEvent(queue<EVENT> q)

{

while(!q.empty()) {

EVENT event = q.pop();

switch(event) {

case KEY\_UP :

break;

case KEY\_DOWN :

break;

.

.

}

}

}

서버에서 InputEventQueue에 저장된 Event를 처리하는 함수.

DWORD WINAPI WorkerThread(LPVOID arg)

{

MSG msg;

int retval;

gGameFramework.BuildObjects();

while (1) {

EnterCriticalSection(&cs);

if (!InputEvent.empty()) {

for (auto input : InputEvent)

Define::Players[input.client\_id]->AddEvent(input);

InputEvent.clear();

}

LeaveCriticalSection(&cs);

gGameFramework.FrameAdvance();

SetEvent(hWorkerEvent[0]);

SetEvent(hWorkerEvent[1]);

WaitForMultipleObjects(2, hSendEvent, true, INFINITE);

{

auto objMgr = Define::SceneManager->GetCurrentScene()->objectManager;

objMgr->GetCreatePack()->clear();

objMgr->GetDeletePack()->clear();

}

}

gGameFramework.OnDestroy();

return((int)msg.wParam);

}

Server Worker Thread

행렬변환 계산을 완료한 뒤, hWorkerEvent를 신호하고 hSendEvent가 신호되는 것을 대기한다.

WorkerThread 게임에서 필요한 행렬 변환이나 충돌 처리를 하며 두 클라이언트에게 계산 결과를 전송한다.

DWORD WINAPI RecvThread(LPVOID arg)

{

int c\_id = \*((int\*)arg);

WaitForSingleObject(hRecvReadyEvent, INFINITE);

while (true) {

EVENT pack;

int retval = recv(Define::sock[c\_id], (char\*)&pack, sizeof(EVENT), 0);

if (retval == 0) return 0;

if (retval == SOCKET\_ERROR) {

if (c\_id == 0) err\_quit("recv()0");

if (c\_id == 1) err\_quit("recv()1");

}

else if (SHOW\_RECV\_DEBUG){

printf("%d : recv Type : %d\n", pack.client\_id, pack.event\_id);

}

EnterCriticalSection(&cs);

InputEvent.push\_back(pack);

LeaveCriticalSection(&cs);

}

}

각 클라이언트에서 recv 받는 쓰레드 함수 이때 클라이언트 구별을 위해 client\_id를 인자로 받았다.

두 클라이언트가 접속하기 전까지 hRecvReady 신호를 기다린다.

InPutEventQueue 동기화를 위해서 임계영역을 사용.

Server Recv Thread

Client에게 오브젝트 생성, 행렬 변환 데이터를 전송하는 Thread

hWorkerEvent를 통해 대기하며, 작업 완료 시 hSendEvent를 신호한다.

DWORD WINAPI SendThread(LPVOID arg)

{

int c\_id = \*((int\*)arg);

int retval;

auto objmgr = Define::SceneManager->GetCurrentScene()->objectManager;

{

auto createPack = objmgr->GetCreatePack();

int createPackSize = createPack->size();

if (SHOW\_SEND\_DEBUG) printf("%d socket : %d EA\n", c\_id, createPackSize);

send(Define::sock[c\_id], (char\*)&createPackSize, sizeof(int), 0);

for (auto pack : \*createPack) {

send(Define::sock[c\_id], (char\*)&pack, sizeof(sc\_create\_object\_packet), 0);

}

}

while (true) {

WaitForSingleObject(hWorkerEvent[c\_id], INFINITE);

{

auto packList = Define::SyncObjectManager->GetAllTransformPack();

int objectSize = packList.size();

if(SHOW\_SEND\_DEBUG) printf("%d socket : %d EA\n", c\_id, objectSize);

for (auto pack : packList)

send(Define::sock[c\_id],(char\*)&pack, sizeof(transform\_packet),0);

}

SetEvent(hSendEvent[c\_id]);

}

return 0;

}

Server Send Thread

DWORD WINAPI RecvThread(LPVOID arg)

{

int reval;

vector<sc\_object\_transform\_packet> packList;

while (true) {

{

for (int i = 0; i < Define::SyncObjectManager->GetSyncList()->size(); i++){

sc\_object\_transform\_packet pack;

reval = recv(Define::sock,(char\*)&pack,sizeof(transform\_packet),0);

if (reval == SOCKET\_ERROR) continue;

packList.emplace\_back(pack);

}

Define::SyncObjectManager->SetTransformPack(packList);

packList.clear();

}

}

}

ClientRecvThread

Client의 Send, Recv를 위한 두 스레드

ClientSendThread

DWORD WINAPI SendThread(LPVOID arg)

{

while (true)

{

KeyControl();

MouseControl();

}

}

void KeyControl()

{

auto recvKey = [](int key, int key\_state) {

EVENT e{ Define::ClientIndex, -1, -1, {-1,-1} };

e.event\_id = key;

e.state = key\_state;

send(Define::sock, (char\*)&e, sizeof(EVENT), 0);

};

if (Define::Input->IsKeyDown())

{

if (Define::Input->GetKeyDown(KeyCode::W))

recvKey(KEY\_W, KEY\_DOWN);

if (Define::Input->GetKeyDown(KeyCode::S))

recvKey(KEY\_S, KEY\_DOWN);

if (Define::Input->GetKeyDown(KeyCode::D))

recvKey(KEY\_D, KEY\_DOWN);

if (Define::Input->GetKeyDown(KeyCode::A))

recvKey(KEY\_A, KEY\_DOWN);

if (Define::Input->GetKeyDown(KeyCode::Space))

recvKey(KEY\_SPACE, KEY\_DOWN);

}

if (Define::Input->IsKeyUp())

{

if (Define::Input->GetKeyUp(KeyCode::W))

recvKey(KEY\_W, KEY\_UP);

if (Define::Input->GetKeyUp(KeyCode::S))

recvKey(KEY\_S, KEY\_UP);

if (Define::Input->GetKeyUp(KeyCode::D))

recvKey(KEY\_D, KEY\_UP);

if (Define::Input->GetKeyUp(KeyCode::A))

recvKey(KEY\_A, KEY\_UP);

if (Define::Input->GetKeyUp(KeyCode::Space))

recvKey(KEY\_SPACE, KEY\_UP);

}

}

KeyControl

Client의 KeyInput 전송을 위한 함수

Key의 Up Down을 서버에게 전송한다.

void MouseControl()

{

if (Define::Input->GetMousePress(MouseButton::Left))

{

POINT mouseAxis = Define::Input->GetMouseAxis();

if (mouseAxis.x != 0)

{

EVENT e{ Define::ClientIndex, MOUSE\_LEFT, MOUSE\_DOWN, mouseAxis };

send(Define::sock, (char\*)&e, sizeof(EVENT), 0);

}

}

else if (Define::Input->GetMouseUp(MouseButton::Left))

{

EVENT e{ Define::ClientIndex, MOUSE\_LEFT, MOUSE\_UP, {0,0} };

send(Define::sock, (char\*)&e, sizeof(EVENT), 0);

}

if (Define::Input->GetMousePress(MouseButton::Right))

{

POINT mouseAxis = Define::Input->GetMouseAxis();

if (mouseAxis.y != 0)

{

EVENT e{ Define::ClientIndex, MOUSE\_RIGHT, MOUSE\_DOWN, mouseAxis };

send(Define::sock, (char\*)&e, sizeof(EVENT), 0);

}

}

else if (Define::Input->GetMouseUp(MouseButton::Right))

{

EVENT e{ Define::ClientIndex, MOUSE\_RIGHT, MOUSE\_UP, {0,0} };

send(Define::sock, (char\*)&e, sizeof(EVENT), 0);

}

}

MouseControl

Client의 MouseInput 전송을 위한 함수

마우스의 이동 방향과 거리를 서버에게 전송한다.

**변동사항**

|  |  |
| --- | --- |
| **수정일** | **수정 사항** |
| **11월 10일** | 서동우 개발 단계 변경으로 17, 18일자 개발 일정표 10, 11일자 내용과 교체 |
| **11월 10일** | 데이터 베이스 시험 날짜 16 -> 23 변경으로 인한 일정표 수정 |
| **11월 16일** | Send Recv 관련 오류 발생으로 인한 일정표 수정 (스레드 동기화 문제로 판명) |
| **11월 18일** | 서버 Sendthread 추가로 인한 로우 레벨 디자인, 하이 레벨 디자인 수정 |
| **11월 24일** | 클라이언트 개발 지연으로 일정표 수정 |
| **11월 28일** | 클라이언트 Sendthread, Recvthread 추가로 인한 로우 레벨 디자인 수정 |
| **12월 2일** | 클라이언트 프레임 드랍 발생 |
| **12월 3일** | 클라이언트 프레임 드랍 문제가 서버 Game Loop에서 동기화 범위를 필요 이상으로 크게 잡은 것이 원인이었음. 해결 |
| **12월 6일** | 서로 다른 호스트에서 서버 연결시 원격으로 연결한 클라이언트가 튕기는 문제 발생. 클라이언트 “transformpacksize”가 잘못된 값을 가지는게 튕기는 원인. |
| **12월 9일** | 호스트 연결 문제 해결을 위해 통신 구조를 변경. 패킷의 개수를 먼저 수신한 뒤 패킷을 전부 수신 받는 형태에서 패킷 개수를 수신하지 않도록 변경. 매 프레임 전송해야하는 패킷의 개수가 동일하기 때문에 문제 없음 |

**일정표**

임윤수 -

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **11/7(화)** | **11/8(수)** | **11/9(목)** | **11/10(금)** | **11/11(토)** | **11/12(일)** | **11/13(월)** |
|  | Protocol.h 헤더파일 완성 | 임시 클라이언트 생성 후 간단한 send recv 테스트 |  | worker thread 내부에서 eventqueue에서 이벤트를 꺼내는 기능 구현 (동기화 고려) | 1차 진행사항 체크 및 피드백 | 겜소공 시험 준비 |
| **11/14** | **11/15** | **11/16** | **11/17** | **11/18** | **11/19** | **11/20** |
| 겜소공 시험 준비 | HandleInput  Event() 함수에서 queue 사용 동기화 구현 | 클라이언트에 소켓 추가 완료 및 send, recv 오류 발생 | 클라이언트에 서버에서 전송한 이벤트 처리 결과  수신 받는 기능 추가 | 오브젝트 생성 패킷을 생성하여 벡터에 삽입하도록 구현 | 2차 진행사항 체크 및 피드백 | 졸작 회의 |
| **11/21** | **11/22** | **11/23** | **11/24** | **11/25** | **11/26** | **11/27** |
| 데베 시험 준비 | 데베 시험 준비 | 충돌 처리를 통해 발생하는 전송해야할 데이터를 벡터에 삽입하도록 구현 | recvthread 마무리  handleinput  event 구현 미흡 | Handleinput  event 구현 완료 | 3차 진행사항 체크 및 피드백 | 졸작 회의 |
| **11/28** | **11/29** | **11/30** | **12/1** | **12/2** | **12/3** | **12/4** |
|  | 클라이언트 수신받은 패킷으로  화면 그리는 기능 구현 |  |  | 1차 플레이 테스트  프레임 드랍 발생 + 카메라 조작 문제 |  | 졸작 회의 |
| **12/5** | **12/6** | **12/7** | **12/8** | **12/9** | **12/10** |  |
| 클라이언트의 workthread와 recvthread의 동기화 구현 | 2차 플레이테스트  카메라 회전 부자연스러움 발생 + 서로 다른 호스트 연결 시 오류 발생 |  |  | 3차 플레이테스트  (타 호스트 연결 시 발생하는 오류 해결하기 위한 회의) | 최종 플레이테스트  (dialog 내용 수정 건의 및 세부 수치 수정) |  |

김동재 -

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **11/7(화)** | **11/8(수)** | **11/9(목)** | **11/10(금)** | **11/11(토)** | **11/12(일)** | **11/13(월)** |
|  | worker thrad와 recvthread 생성 및 대기 구현 | 서버 전역변수 생성 |  | recvthread 에서 패킷을 수신 받아 queue에 삽입하는 기능 구현 (동기화 고려) | 1차 진행사항 체크 및 피드백 | 겜소공 시험 준비 |
| **11/14** | **11/15** | **11/16** | **11/17** | **11/18** | **11/19** | **11/20** |
| 겜소공 시험 준비 | 서버에 패킷을 관리하는 벡터 추가 | workerthread에서 이벤트 처리 후 전송할 패킷 생성 및 list 관리 구현 | 이벤트 처리를 통한 좌표 이동 결과를 클라이언트에게 전송하는 기능 추가  Send 오류 | Sendthread 추가 및 workerthread와 순서 제어 구현 | 2차 진행사항 체크 및 피드백 | 졸작 회의 |
| **11/21** | **11/22** | **11/23** | **11/24** | **11/25** | **11/26** | **11/27** |
| 데베 시험준비 | 좌표를 전송하는 기존 구현에서 변환 행렬을 전송하도록 바꾸고 클라이언트에 적용 테스트 | 충돌 처리를 전송하고 클라이언트에서 수신받아 적용하는지 테스트  테스트 실패 | workerthread 마무리 |  | 3차 진행사항 체크 및 피드백 | 졸작 회의 |
| **11/28** | **11/29** | **11/30** | **12/1** | **12/2** | **12/3** | **12/4** |
|  | 클라이언트 이벤트를 서버에 수신하도록 구현 |  |  | 1차 플레이 테스트  프레임 드랍 발생 + 카메라 조작 문제 | 클라이언트 동기화 수정을 통해 프레임 드랍 해결 | 졸작 회의 |
| **12/5** | **12/6** | **12/7** | **12/8** | **12/9** | **12/10** |  |
| 클라이언트의 workthread와 recvthread의 동기화 구현 | 2차 플레이테스트  카메라 회전 부자연스러움 발생 + 서로 다른 호스트 연결 시 오류 발생 | 클라이언트 마우스 input 정보 전송 최적화 | 서버 workthread와 recvthread 동기화 수정 | 3차 플레이테스트  (타 호스트 연결 시 발생하는 오류 해결하기 위한 회의) | 최종 플레이테스트  (게임 종료 시 출력되는 dialog 수정) |  |

서동우 -

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **11/7(화)** | **11/8(수)** | **11/9(목)** | **11/10(금)** | **11/11(토)** | **11/12(일)** | **11/13(월)** |
|  | 서버에 프레임워크 이식 | 졸작 회의 | 서버의 게임 메니저 클래스 구현 | 서버의 게임 메니저 클래스 완성 | 1차 진행사항 체크 및 피드백 | 겜소공 시험 준비 |
| **11/14** | **11/15** | **11/16** | **11/17** | **11/18** | **11/19** | **11/20** |
| 겜소공 시험 준비 | workerthread 로직 구현 | 졸작 회의 | HandleEvent함수 및  workerthread 로직 구현 | HandleEvent함수 및  workerthread 로직 완성 | 2차 진행사항 체크 및 피드백 | 데베 시험 준비 |
| **11/21** | **11/22** | **11/23** | **11/24** | **11/25** | **11/26** | **11/27** |
| 클라이언트에 송신 받은 데이터 관리할 class 구현  (오브젝트 관리 class 자체 구현) | 클라이언트에 송신받은 데이터 관리할 class 완성  (오브젝트 관리 class 자체 구현) | 졸작 회의 | RecvCreate  Object(), RecvDelete  Object() 구현  Send Recv 오류 발생 | RecvUpdate  Transform()구현 | 3차 진행사항 체크 및 피드백 | SendHandle  InputEvent() 함수 구현 |
| **11/28** | **11/29** | **11/30** | **12/1** | **12/2** | **12/3** | **12/4** |
|  | 클라이언트 수신받은 패킷으로 화면 그리는 기능 구현 완성 | 졸작 회의 | 클라이언트 keyinput up,down만 수신하게 변경 |  | 카메라 문제 해결 및 충돌 처리 수정 |  |
| **12/5** | **12/6** | **12/7** | **12/8** | **12/9** | **12/10** |  |
| 서버의 충돌판정 오류 수정(status component 추가) | 메모리 직렬화 제작 | 메모리 직렬화 제작 완료 | 카메라 이동 부자연스러운 문제 해결 | 3차 플레이테스트  (타 호스트 연결 시 발생하는 오류 해결하기 위한 회의) | 서버 전송 파트 최적화를 통한 문제 해결 |  |