**2022 네트워크 게임 프로그래밍(03)**

**Project Progress Report**

**2018182013 박동규**

**2018184005 김정훈**

**2020182044 황석주**

**목차**

**1. 어플리케이션 기획**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3

1-1. 게임 소개\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3

1-2. 게임 진행방식\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3

1-3. 게임 프레임워크\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4

**2. 개발 환경**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_5

**3. High-Level 디자인**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6

2-1. 서버 High-Level\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8

2-2. 클라이언트 High-Level\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10

**4. Low-Level 디자인**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_12

3-1. 서버 Low-Level\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_12

3-2. 클라이언트 Low-Level\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_16

**5. 팀원 별 역할 분담**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_18

**6. 상세 개발 일정**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_19

1. **애플리케이션 기획**

**1-1. 게임소개**

게임 이름: Archery game (2021컴퓨터 그래픽스, 박동규)

플레이어: 각 플레이어는 활과 화살을 갖고 시작한다.

승리 방법: 랜덤 위치에 과녁이 생기고 플레이어들이 과녁들을 맞추면 사라지고 점수를 얻는다.

모든 과녁이 사라졌을 때 점수가 높은 플레이어가 승리한다.

텍스트, 표지판이(가) 표시된 사진

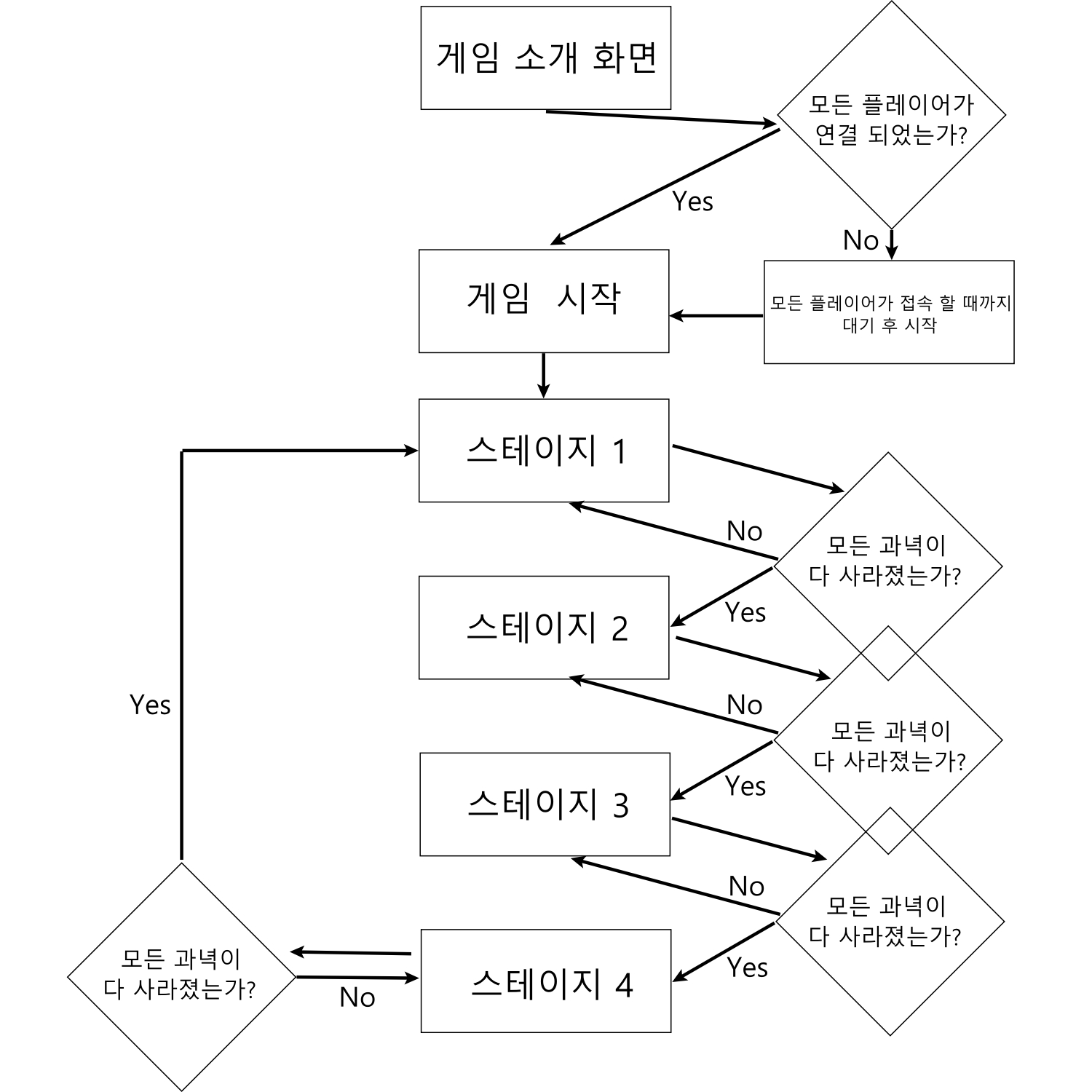
자동 생성된 설명

**1-2. 게임 진행방식**

기존에 1인 플레이 방식이었으나 2인용으로 변경합니다.

1. 실행하면 게임 실행 전 게임소개 화면이 나오게 됩니다.
2. 화면에서 T키를 누르게 되면 본 게임으로 넘어가게 됩니다.
3. 플레이어 두 명은 순서 없이 각자 활을 과녁을 향해 쏘게 됩니다.
4. 모든 과녁이 사라지면 플레이어의 점수가 높은 쪽의 승리 카운트가 증가합니다.
5. 그 후 다음 스테이지로 이동하여 3~4 과정을 계속해서 반복하게 됩니다.

**1-3. 게임 프레임워크**

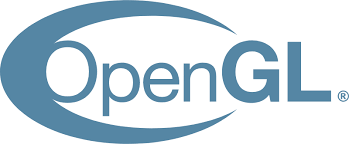


1. **개발 환경**

**2-1. Visual Studio 2022**



**2-2.** OpenGL



**2-3.** GitHub



**3. High-Level 디자인**

**3-1. 서버-클라이언트 간 통신**

**(수정 전)**

서버

클라이언트

Main

Main

Send()

Accept()

Connect()

Recv()

Start

Client

Management

Draw

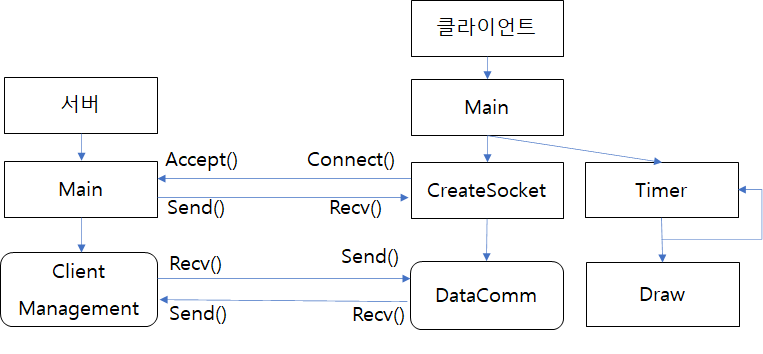
Data Comm

Send()

Recv()

Send()

Recv()

**(수정 후)**

**송수신 데이터**

**클라이언트가 보내는 데이터**

float y\_angle -> 플레이어 카메라의 y각도 값

float x\_angle -> 플레이어 카메라의 x각도 값   
~~float arrow.objectmatrix.position -> 화살의 좌표값~~

~~float arrow.modelmatrix.rotation -> 화살의 회전값~~

glm::vec3 arrow.objectmatrix.position -> 화살의 좌표값

glm::vec3 arrow.modelmatrix.rotation -> 화살의 회전값

**서버가 보내는 데이터**

float y\_angle -> 상대 플레이어 카메라의 y각도 값

float x\_angle -> 상대 플레이어 카메라의 x각도 값

glm::vec3 arrow.objectmatrix.position -> 상대 화살의 좌표값

glm::vec3 arrow.modelmatrix.rotation -> 상대 화살의 회전값

short total\_score -> 전체 플레이어의 점수

short wind\_dir -> 바람의 방향

float wind\_speed -> 바람의 세기

short circleState[CIRCLENUM] -> 과녁의 상태(사라진 과녁, 사라지고 있는 과녁, 남아있는 과녁)

int stage ->현재 스테이지의 번호

short winCount -> 각 플레이어의 승리 횟수

**서버가 클라이언트에게 보낼 초기화 데이터**

~~float circleCenter[25] -> 과녁의 중앙 값~~

~~float player1pos -> 플레이어 1의 위치~~

~~float player2pos -> 플레이어 2의 위치~~

glm::vec3 circleCenter[CIRCLENUM] -> 과녁의 중앙 값

glm::vec3 player1pos -> 플레이어 1의 위치

glm::vec3 player2pos -> 플레이어 2의 위치

**3. High-Level 디자인**

**3-2. 서버 플로우 차트**

서버

클라이언트 ID와

초기화값을 송신한다

다른 플레이어 대기

클라이언트로부터

데이터를 수신한다

다른 클라이언트의

데이터 수신을 기다린다

서버에서 보내주는

데이터를 갱신한다

클라이언트로 데이터를 송신한다

연결을 종료한다

모든 플레이어가

연결 되었는가?

다른 클라이언트가

존재하는지 확인한다

클라이언트와 통신을

확인한다

yes

yes

no

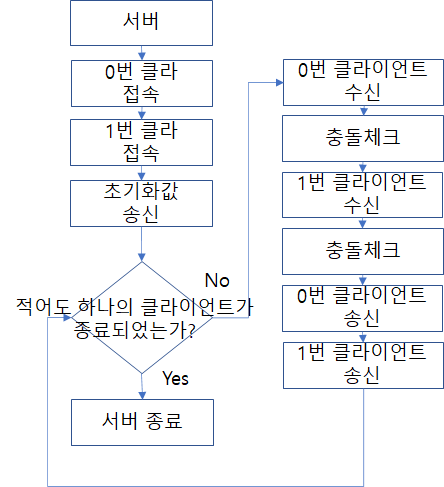
no

no

yes

**(수정 전)**

**(수정 후)**



**3. High-Level 디자인**

**3-3. 클라이언트 플로우 차트**

**(수정 전)**

클라이언트

연결되지 않음

클라이언트의 데이터를 갱신한다

화면에 그린다

서버로 데이터를 송신한다

서버로부터 데이터를 수신한다

연결을 종료한다

서버와 연결을

시도한다

서버로부터 초기화값을

수신해 초기화한다

서버와 통신을

확인한다

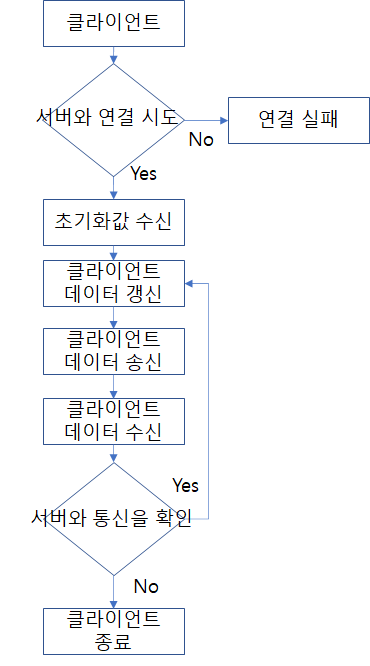
no

no

Yes

Yes

(수정 후)



**4. Low-Level 디자인**

**4-1. 서버 LOW-Level**

#include “Common.h”

* 구형 C 함수 사용 경고 끄기, 구형 소켓 API 사용 경고 끄기
* 윈속2 메인 헤더, 윈속2 확장 헤더, ws2\_32 lib 링크
* 소켓 함수 오류 출력 후 종료, 오류 출력 함수

~~struct Packet~~

* ~~Float x\_angle, y\_angle -> 클라이언트 플레이어의 시야 각도 값~~
* ~~float obecjtmatrix.postion, modelmatrix.rotation 화살의 좌표값, 회전값~~
* ~~short total\_score 클라이언트의 현재 점수~~
* ~~short wind\_dir, float wind\_speed 바람의 방향과 세기~~
* ~~short circleState[25] 과녁의 상태~~

struct Packet

float x\_angle, y\_angle -> 플레이어의 시야 각도 값

glm::vec3 arrowPosition -> 화살의 좌표 값;

glm::vec3 arrowRotation -> 화살의 회전 값;

short total\_score -> 플레이어의 현재 점수

short wind\_dir -> 바람의 방향

float wind\_speed -> 바람의 세기

short circleState[CIRCLENUM] -> 과녁의 상태

int stage -> 현재 스테이지 번호

short winCount=0 각 플레이어의 승리 횟수

~~struct InitPacket~~

* ~~float circleCenter[25] 과녁의 중앙의 위치값~~
* ~~float player1pos 플레이어 1의 위치~~
* ~~float player2pos 플레이어 2의 위치~~

struct InPacket

float x\_angle, y\_angle -> 플레이어의 시야 각도 값

glm::vec3 arrowPosition -> 화살의 좌표 값

glm::vec3 arrowRotation -> 화살의 회전 값

struct InitPacket

glm::vec3 circleCenter[CIRCLENUM]; // 과녁의 중앙의 위치 값

glm::vec3 player1Pos; // 플레이어 1의 위치

glm::vec3 player2Pos; // 플레이어 2의 위치

struct SocketWithIndex

* sock 소켓
* index 몇 번째 클라이언트인지에 대한 값

전역변수

* ~~(int)MAX\_CLIENT 연결 가능한 최대 클라이언트의 수(2)~~
* (short)SERVERPORT 서버 포트번호 지정(9000)
* ~~(HANDLE)hReadEvent[MAX\_CLIENT] 동기화를 위한 이벤트 핸들~~
* ~~(HANDLE)hWirteEvent[MAX\_CLIENT] 동기화를 위한 이벤트 핸들~~
* ~~(Packet)g\_Data[MAX\_CLIENT] 클라이언트와 주고받을 데이터~~
* (HANDLE)hRecvEvent[2] 동기화를 위한 이벤트 핸들
* (HANDLE)hSendEvent[2] 동기화를 위한 이벤트 핸들
* (Packet)g\_Packet[2] 클라이언트와 주고받을 데이터
* InPacket g\_Inpacket -> 클라이언트의 정보를 받을 변수
* HANDLE hThread[2] -> 쓰레드의 핸들 배열

~~int main(int argc, char\* argv[])~~

~~변수~~

* ~~(int)retval 소켓함수의 리턴값을 저장하는 변수~~
* ~~(WSADATA)was 윈속 구현에 관한 정보를 담을 구조체~~
* ~~(sockaddr\_in)serveraddr 서버 주소를 담을 구조체~~
* ~~(SOCKET)listen\_sock 대기 소켓~~
* ~~(SOCKET)client\_sock 클라이언트 소켓~~
* ~~(sockaddr\_in)clientaddr 클라이언트 주소를 담을 구조체~~
* ~~(int)addrlen 주소 길이를 담을 변수~~
* ~~(HANDLE)hThread[MAX\_CLIENT] 스레드 함수를 위한 핸들~~
* ~~(int)client\_index 클라이언트의 현재 개수~~
* ~~(SocketWithIndex)swi 스레드 함수의 인자로 들어가는 소켓과 인덱스값이 들어있는 구조체~~

사용 함수

* WindTimer() -> 바람의 세기와 방향을 정해주는 함수
* InitPacket()-> 초기 데이터를 초기화해주는 함수
* ~~WSAStartup() 윈속 초기화 함수 윈속 버전2.2 사용~~
* ~~socket() IPv4 주소체계를 이용해 TCP통신 소켓을 만들기 위한 함수~~
* ~~hton\*() 네트워크 바이트 정렬을 위한 함수~~
* ~~bind() 대기 소켓에 서버 정보를 넣기 위한 함수~~
* ~~listen() 대기 소켓을 대기 상태로 만드는 함수~~
* ~~accept() 클라이언트의 연결을 받는 함수~~
* ~~send() 클라이언트에게 초기값을 보내주는 함수~~
* ~~CreateThread() 스레드 함수를 만드는 함수~~
* ~~closesocket() 소켓 삭제를 위한 함수~~
* ~~WSACleanup() 윈속 종료를 위한 함수~~

호출 순서

~~윈속 초기화 후 넘겨줄 데이터를 초기화 하고 클라이언트의 연결을 기다린다.~~

~~연결 요청을 받고 모든 클라이언트와 연결이 되었다면 초기화 값을 보내준다.~~

~~연결된 클라이언트 소켓과 client\_idnex로 cwi값을 만들어 인자로 넣어 CreateThread() 함수를 호출하여 ClientMgr스레드 함수를 hThread[client\_index] 핸들에 연결한다.~~

~~CreateEvent()로 자동 리셋 이벤트, 초기 값을 false로 이벤트를 만들어 hWriteEvent[client\_index], hReadEvent[client\_index]에 연결한다.~~

~~SetEvent(hReadEvent[0])로 이벤트를 신호 상태로 바꿔준다.~~

윈속 초기화 후 넘겨줄 데이터를 초기화 하고 클라이언트의 연결을 기다린다.

두 클라이언트가 모두 연결이 되었다면 연결된 클라이언트 소켓과 client\_index로 cwi값을 만들어 인자로 넣고 CreateThread()함수를 호출하여 ClientMgr스레드 함수를 hThread[client\_index] 핸들에 연결한다.

WindTimer를 스레드로 실행시킨다.

각 클라이언트의 송수신이벤트를 CreateEvent함수로 수동 리셋이벤트, 0번 클라이언트의 수신 이벤트를 제외하고는 초기값을 false로 만들어준다.

WaitForMultipleObjects()함수로 ClinetMgr스레드가 하나라도 끝날때 까지 대기한다.

하나라도 끝났다면(연결이 종료되었다면) 이벤트 핸들을 닫고, 소켓을 닫고 윈속을 종료시키고 서버를 끝낸다.

DWORD WINAPI ClientMgr(LPVOIDG arg)

변수

* (int)index 몇 번째 클라이언트인지 확인하기 위한 인덱스
* (SOCKET)client\_sock 클라이언트 소켓
* (int)retval 소켓함수의 리턴값을 저장하는 변수
* (sockaddr\_in)clinetaddr 클라이언트의 주소를 담을 구조체
* (int)addrlen 주소 길이를 담을 변수
* (DWORD)waitval WaitForSingleObject()의 리턴값을 저장하는 변수

사용 함수

* WaitForSingleObject() 이벤트를 기다리기 위한 함수
* SetEvent() 이벤트를 신호 상태로 바꾸기 위한 함수
* ~~getpeername() 클라이언트의 주소 정보를 가져오기 위한 함수~~
* recv() 클라이언트로부터 데이터를 받아오기 위한 함수
* send() 클라이언트로 정보를 보내기 위한 함수
* CircleMgr() 과녁상태와 좌표를 관리하는 함수
* ArrowCheck() -> 화살과 과녁의 충돌체크 함수
* NextStage()-> 스테이지를 관리해주는 함수
* ~~Offset() 과녁 상태와 좌표 관리하는 함수~~

호출 순서

인자로 받은 arg로 client\_sock과 index값을 받는다.

~~getpeername()함수를 호출해 client\_sock에서 clientaddr로 클라이언트 주소 정보를 읽어온다.~~

클라이언트에 초기값을 보내준다~~.~~ 무한 루프를 돌며 클라이언트와 데이터 송수신을 시작한다.

WaitForSingleObject()로 hRecvEvent[index]가 신호상태가 될 때까지 기다린다. 신호상태가 되면 recv()함수로 데이터를 g\_Packet[index]에 수신하고 index가 0이면 SetEvent()로 hRecvEvent[1]을 신호상태로 변경하고 hWirteEvent[0]을 기다리게 하고, 1이라면 hSendEvent[0]을 신호상태로 바꿔주고 hSendEvent[1]을 기다리게 한다.

이 과정에서 ~~Offset()~~ CircleMgr()을 통하여 과녁과의 충돌체크를 하고 과녁을 관리해준다. 실시간으로 먼저 과녁을 맞혔으면 과녁이 사라지고 해당 플레이어가 점수를 획득할 수 있도록 이벤트를 이용해 동기화한다.

hSendEvent[index]가 신호상태가 되면 send()함수로 g\_Packet의 데이터를 송신하고 index가 0이면 hSendEvent[1]을 신호상태로 변경하고 hReadEvent[0]을 기다리고, 1이라면 hReadEvent[0]을 신호상태로 변경하고 hRecvEvent[1]을 기다리게 하면서 이벤트 방식으로 동기화를 하여 두 클라이언트의 데이터를 모두 받아서 같은 데이터를 송신할 수 있도록 한다.

**4. Low-Level 디자인**

**4-2. 클라이언트 LOW-Level**

~~struct Packet~~

* ~~y\_angle -> 현재 플레이어 카메라의 y각도 값~~
* ~~x\_angle -> 현재 플레이어 카메라의 x각도 값~~
* ~~arrow.objectmatrix.position -> 화살의 좌표값~~
* ~~arrow.modelmatrix.rotation -> 화살의 회전값~~

struct Packet

float x\_angle, y\_angle -> 플레이어의 시야 각도 값

glm::vec3 arrowPosition -> 화살의 좌표 값

glm::vec3 arrowRotation -> 화살의 회전 값

struct InPacket

float x\_angle, y\_angle -> 플레이어의 시야 각도 값

glm::vec3 arrowPosition -> 화살의 좌표 값;

glm::vec3 arrowRotation -> 화살의 회전 값;

short total\_score -> 플레이어의 현재 점수

short wind\_dir -> 바람의 방향

float wind\_speed -> 바람의 세기

short circleState[CIRCLENUM] -> 과녁의 상태

int stage -> 현재 스테이지 번호

short winCount=0 각 플레이어의 승리 횟수

struct InitPacket

glm::vec3 circleCenter[CIRCLENUM]; // 과녁의 중앙의 위치 값

glm::vec3 player1Pos; // 플레이어 1의 위치

glm::vec3 player2Pos; // 플레이어 2의 위치

전역 변수

* bool connectState -> 현재 서버와 연결이되었는지 확인하기 위한 변수
* float other\_x\_angle -> 상대 클라 화살 각도
* float other\_y\_angle ---
* float otherCamera\_x -> 상대 클라이언트 카메라 각도
* float otherCamera\_y ---
* float otherCamera\_z ---
* int myScore -> 나의 점수 표시 변수
* int otherScore ->상대 점수 표시 변수
* int myWin -> 나의 승리 카운트 변수
* int otherWin -> 상대 승리 카운트 변수
* int circlewidth -> 과녁의 가로 개수
* int circleheight -> 과녁의 세로 개수
* Circle circles[CIRCLENUM] -> 과녁들의 배열

생성한 함수

* bool CreateSocket() -> 초기에 소켓을 생성하고 connet하는 함수

~~변수~~

* ~~SOCKET sock -> 소켓의 정보를 담을 변수~~
* ~~sockaddr\_in serveraddr -> 서버의 주소를 담을 변수~~
* ~~int retval -> send와 recv의 리턴값을 저장하는 변수~~

~~함수~~

* ~~WSAStartup() -> 윈속 초기화~~
* ~~WSACleanup() -> 윈속 종료~~
* ~~Socket() -> 소켓 생성~~
* ~~closesocket() ->소켓 삭제~~
* ~~htons() -> 네트워크 바이트 정렬을 해줌~~
* ~~memset() -> 메모리를 초기화 시킴~~
* ~~inet\_pton() -> 문자열의 주소를 숫자로 변환함~~
* ~~connect() -> 서버에 연결을 요청하는 함수~~
* ~~recv() -> 서버로부터 초기 offset데이터를 받음~~
* void Data Comm(LPVOID arg) -> 서버의 정보를 recv하고 자신의 정보를 send하는 함수

변수

* ~~int data\_len -> 보낼 데이터의 길이를 담을 변수~~

함수

* send() -> 서버에게 자신의 정보를 보냄
* recv() -> 서버로부터 상대 클라이언트의 정보를 받음

호출 순서

초기 메인로딩 화면이 나온 뒤 플레이어가 ‘T’ 버튼을 누르게 되면 CreateSocket() 호출

서버에서 연결이 될 때까지 대기 후 초기 데이터 수신한다. 또한 Datecomm을 쓰레드 함수로 생성한다.

Datacomm에서는 서버와 정보를 교환하는 send와 recv가 호출된다.

~~if(CreateSocekt()) connectState = true;~~

~~DrawScene()에서 connectState가 true값이라면 DataComm() 호출~~

~~if(connectState) DataComm();~~

~~DataComm()함수 내부에서 다음과 같은 함수들을 호출~~

~~recv() 서버로부터 데이터를 받음~~

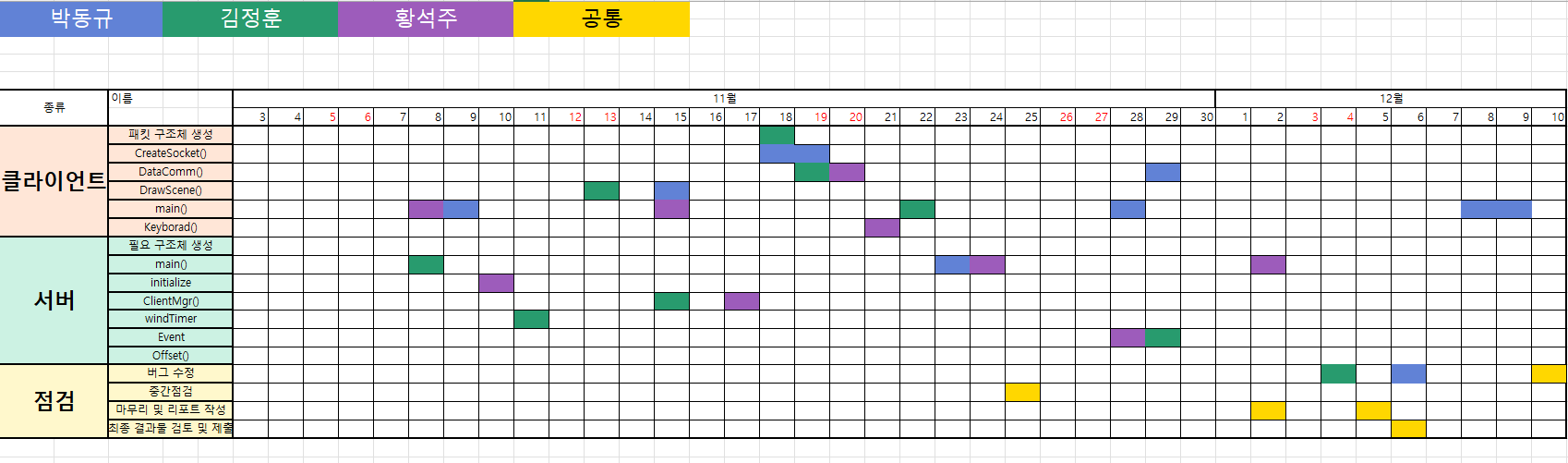
~~send() 패킷의 정보를 담아 서버로 보냄~~

~~데이터를 받았으면 화면을 그리기 위해 DrawScene()으로 돌아감~~

~~받은 상대방의 정보와 자신의 정보를 바탕으로 Scene을 그림~~

**5. 팀원 별 역할 분담**

**역할 분담표**



**6. 상세 개발 일정**

**박동규 일정**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| X | X | X | X | 3 | 4 | 5 |
| 6 | 7 | 8 | 9  클라이언트 main 수정 | 10 | 11 | 12 |
| 13 | 14 | 15  클라이언트 DrawScene 상대 플레이어 그리기  (16일 새벽에 커밋) | 16 | 17 | 18  클라이언트  CreateSocket  연결 부분  (19일 새벽에 커밋) | 19  클라이언트  CreateSocket  송신 부분 |
| 20 | 21 | 22 | 23  서버 과녁 함수 생성 | 24 | 25  공통 중간 점검 | 26 |
| 27 | 28  Draw 함수 고치기(다른 클라이언트 정보 받아와서 그리는 부분) | 29  Windtimer 적용  Circle draw 수정 | 30 | 12/1 | 2 | 3 |
| 4 | 5 | 6  Score표시 처리 | 7 | 8 | 9  클라이언트 화살 충돌처리 수정,  클라이언트 파티클 수정 | 10  승리 카운트처리  , 공통 마무리 |
| 11  최종 검토 제출 | 12 | 13 | X | X | X | X |

**김정훈 일정**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| X | X | X | X | 3 | 4 | 5 |
| 6 | 7 | 8  서버 main  수정, 서버  필요 구조체  생성 | 9 | 10 | 11  Wind Timer  구현 | 12 |
| 13 | 14 | 15  서버  ClientMgr  Index1인 경우  (16일 새벽에 커밋) | 16 | 17 | 18  클라이언트  패킷 구조체  생성 | 19  클라이언트  DataComm  수신 부분 |
| 20 | 21 | 22  클라이언트  과녁 draw | 23 | 24 | 25  공통 중간 점검 | 26 |
| 27 | 28 | 29  Datacomm을 스레드함수로 변경 | 30 | 12/1 | 2 | 3 |
| 4  스테이지 pass 수정, 스코어 draw 수정 | 5 | 6 | 7 | 8 | 9 | 10  공통 마무리 |
| 11  최종 검토 제출 | 12 | 13 | X | X | X | X |

**황석주 일정**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| X | X | X | X | 3 | 4 | 5 |
| 6 | 7 | 8  클라이언트  main  윈속 구현 | 9 | 10  서버  초기화값 설정 | 11 | 12 |
| 13 | 14 | 15  클라이언트  DrawScene  상대배경 그리기 | 16 | 17  서버  ClientMgr  Index2인 경우 | 18 | 19 |
| 20  클라이언트  DataComm  송신 부분 | 21  클라이언트  Keyboard  수정 | 22 | 23 | 24  서버 과녁  충돌체크 | 25  공통 중간 점검 | 26 |
| 27 | 28  서버  ReadEvent 구현  서버 windTimer 스레드 함수로 변경 | 29 | 30 | 12/1 | 2  서버  Event 수정 | 3 |
| 4 | 5 | 6  CircleCenter 초기값 수정 및 Server main 버그 수정 | 7 | 8 | 9 | 10  공통 마무리 |
| 11  최종 검토 제출 | 12 | 13 | X | X | X | X |