**네트워크 게임 프로그래밍**

**Term Project 보고서**

2014180015 게임공학과 김정현

2014182043 게임공학과 차종원

2016184017 엔터테인먼트 컴퓨팅 박하연

목차

[ 어플리케이션 기획 3](#_Toc531667632)

[• 게임 소개 3](#_Toc531667633)

[• 게임 오브젝트 3](#_Toc531667634)

[• 조작 방법 4](#_Toc531667635)

[• 게임 설명 4](#_Toc531667636)

[• Flow Chart 4](#_Toc531667637)

[ High Level Design 5](#_Toc531667638)

[• Flow Chart 5](#_Toc531667639)

[• Server – Client 통신 7](#_Toc531667640)

[ Low Level Design 8](#_Toc531667641)

[• Server – Client Protocol 8](#_Toc531667642)

[• Server 9](#_Toc531667643)

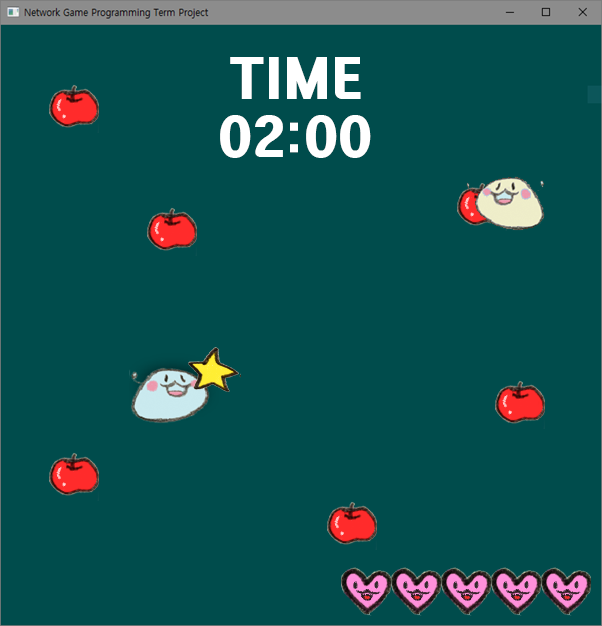
[• Client 11](#_Toc531667644)

[ 역할 분담 12](#_Toc531667645)

[ 개발 환경 12](#_Toc531667646)

[ 개발 일정 13](#_Toc531667647)

[ 개발 보고서 17](#_Toc531667648)

* 어플리케이션 기획

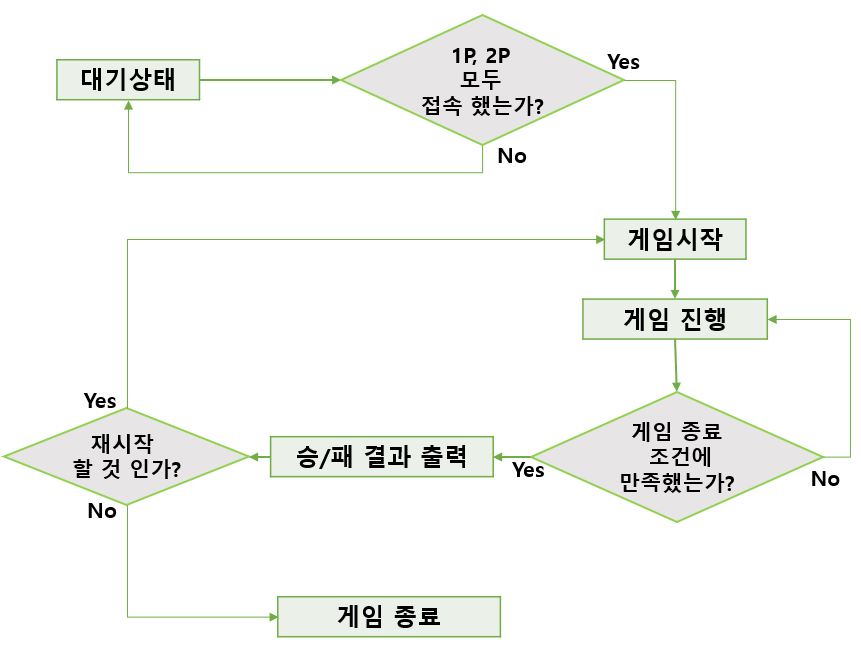
***[ 게임 전체적 모습 ]***

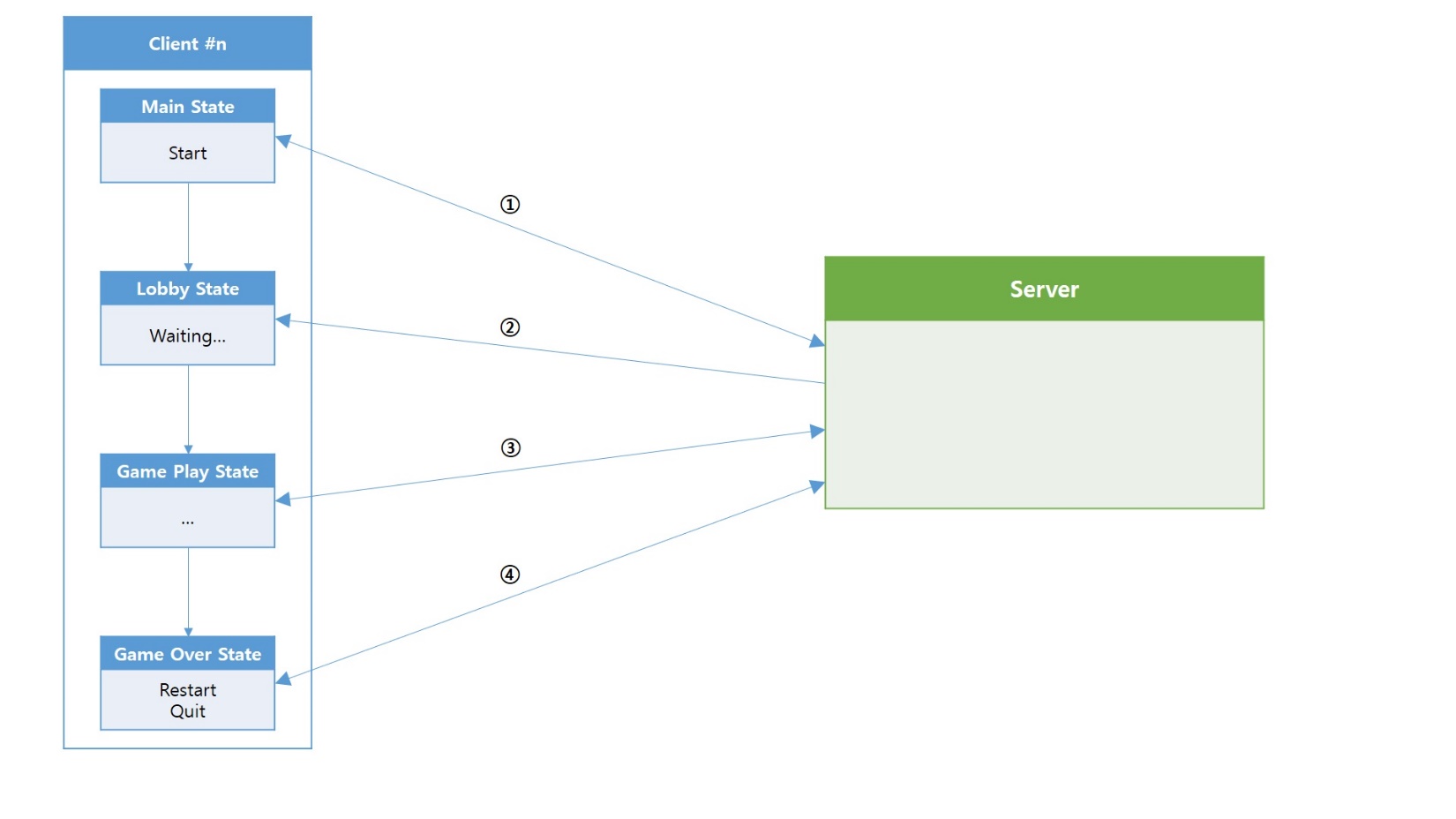
* 게임 소개

OpenGL 위에 2D 스프라이트 리소스를 사용하여 만든 게임이다.

플레이어 2명이 일정 시간마다 무작위 위치에 생성되는 아이템을 먹어 상대방을 공격하는  
 게임이다.   
 게임 시간이 끝나거나 게임 시간 안에 생명이 0이 된 플레이어가 발생하면 게임이 종료된다.

* 게임 오브젝트
* 플레이어: 
* 아이템: 
* 총알: 
* 조작 방법
* 캐릭터 이동 (W/A/S/D) : 캐릭터 좌, 우, 상, 하, 대각선 이동
* 게임 설명
* 이 게임은 2명이 함께 플레이하는 게임이다.
* 캐릭터를 상, 하, 좌, 우로 이동시키며 아이템을 먹거나 상대방의 아이템을 피해야 한다.
* 아이템은 특정 시간마다 맵 안 임의의 위치에 생성된다.
* 아이템을 먹으면 현재 상대방의 위치를 향해 아이템이 발사된다.
* 게임의 제한 시간은 120초, 생명은 5로 시작한다.
* 플레이어와 총알이 충돌하면 플레이어의 생명이 1 감소한다.
* 제한 시간 내에 상대 플레이어의 생명을 0으로 만들면 승리한다.
* 제한 시간 동안 생명이 0이 된 플레이어가 없으면 남은 생명으로 승패를 판단한다.
* Flow Chart



* High Level Design
*  Flow Chart

**[ State별 개괄적 통신 플로우 차트 ]**

1. **Main State – Server 통신**

**클라이언트는 플레이어가 게임을 실행하여 WASD중 하나의 키를 입력하면 그 키 정보를 서버에게 전송한다. 그 후 서버로부터 변경된 자신의 State 정보를 수신한다.**

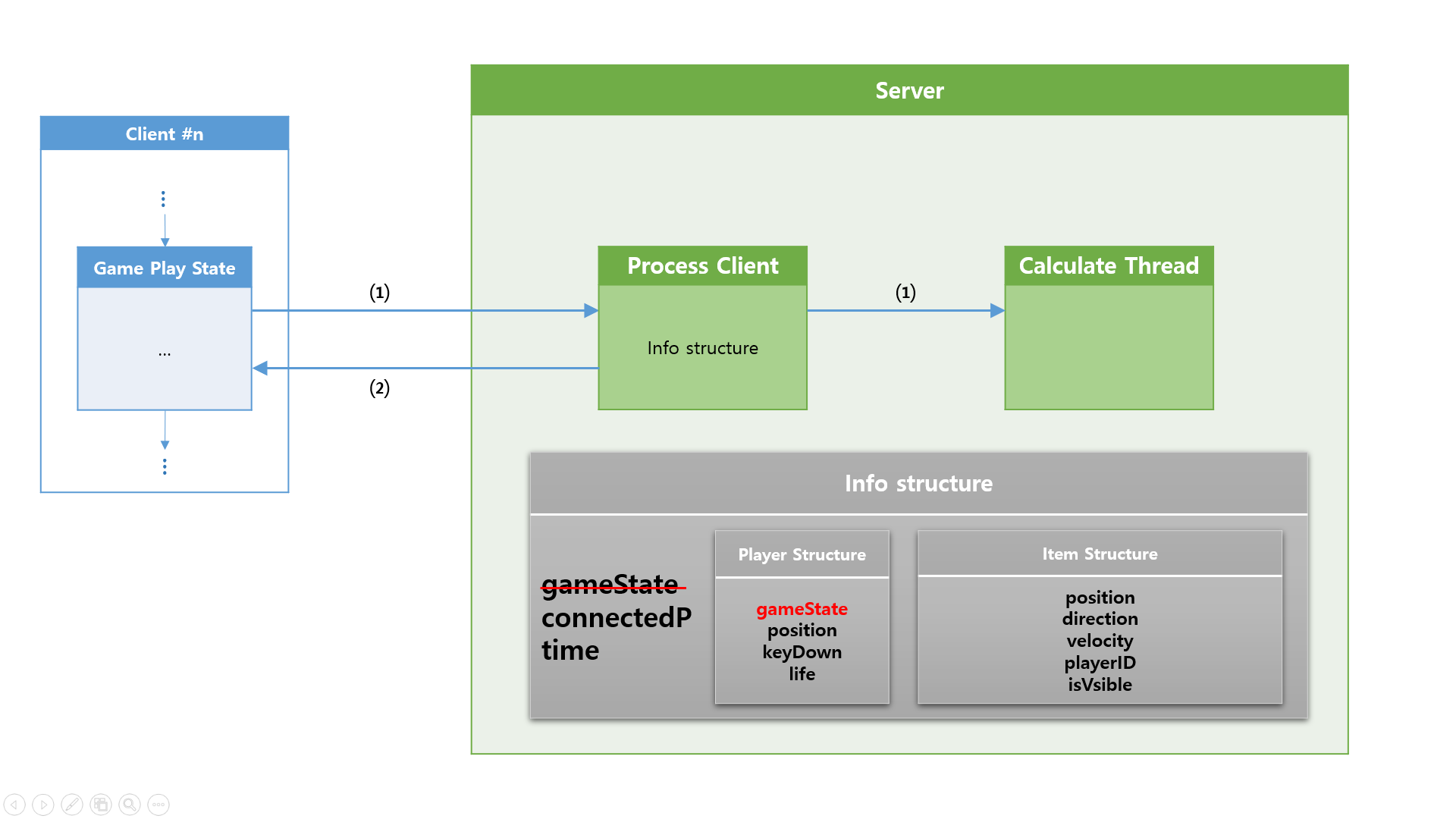
**서버는 받은 키 정보를 토대로 해당 클라이언트의 State 정보를 Lobby State로 변경하여 그 정보를 클라이언트에게 재전송한 후 서버 접속자의 수를 늘린다.**

1. **Lobby State – Server 통신**

**클라이언트는 서버에게 데이터 전송을 하지 않고, 서버로부터 변경된 State 정보를 받을 때까지 계속 대기한다.**

**서버는 서버 접속자의 수가 총 2명이라면 각 클라이언트의 State 정보를 Game Play State로 변경하고 그 정보를 클라이언트에게 전송한다.**

1. **Game Play State – Server 통신**



**[ *Game Play State – Server* 통신 플로우 차트 ]**

⑴ 매 프레임 마다 클라이언트로부터 Process Client 스레드로 정보를 받아와 Info structure의 정보를 갱신한다.

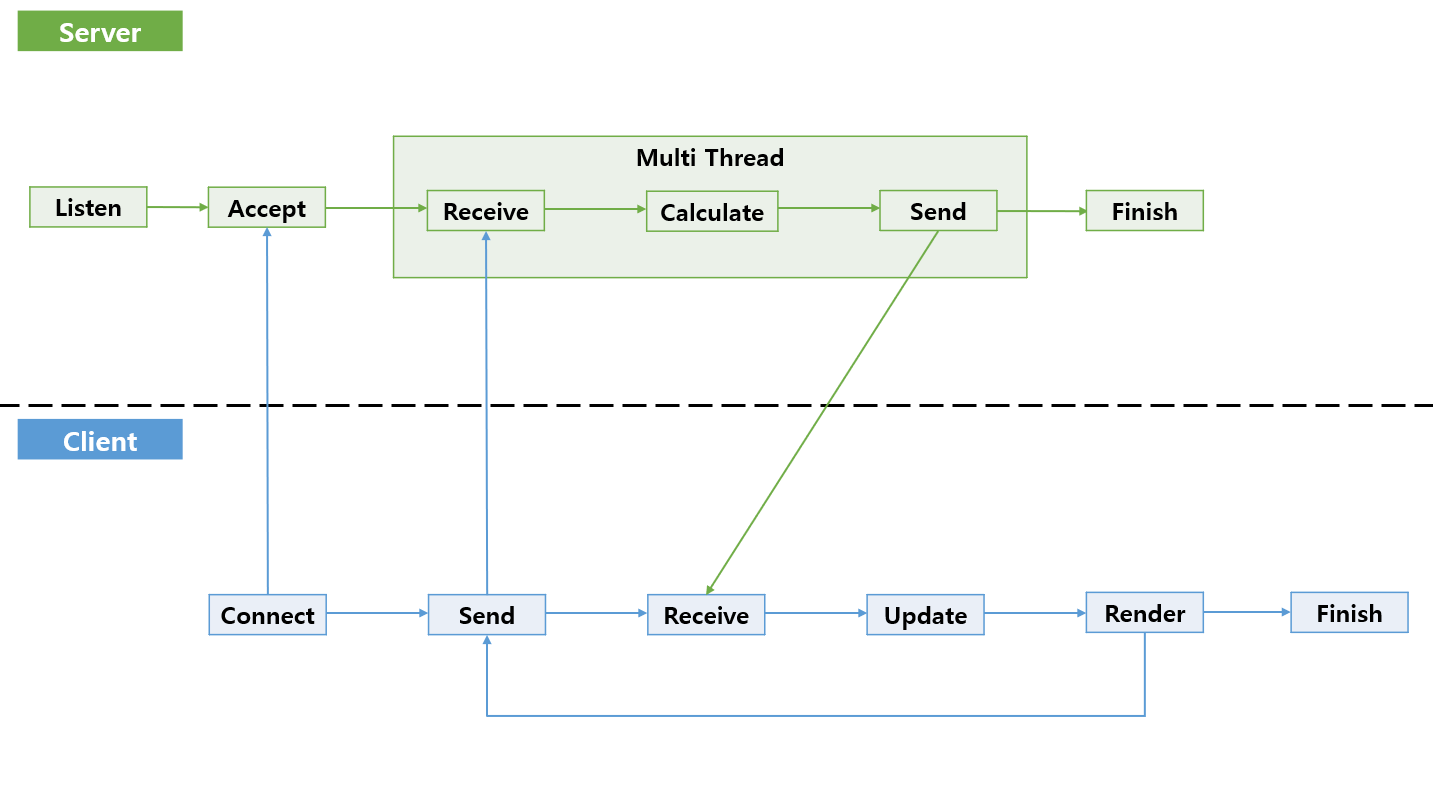
**⑵ 갱신한 정보를 가지고 게임에 필요한 연산(오브젝트 위치 이동, 충돌 체크, 시간 계산 등)을 처리한 후 그 정보를 다시 Process Client에서 해당 클라이언트에게 재전송한다.**

1. **Game Over State – Server 통신**

**클라이언트는 플레이어가 재시작을 위해 “A” 키를 입력하면 그 키 정보를 서버에게 전송한 후 서버로부터 변경된 자신의 State 정보를 수신한다.  
게임 종료를 위해 “D” 키를 누르면 그 키 정보를 서버에게 전송한 후 통신을 중단한다.**

**서버는 받은 키 정보를 토대로 재시작을 원하는 플레이어의 수가 총 2명이라면, 각 클라이언트의 State 정보를 Game Play State로 변경하고 그 정보를 클라이언트에게 재전송한다.  
재시작을 원하는 플레이어의 수가 총 2명이 아니라면, 게임을 종료한 클라이언트와는 통신을 중단하고 재시작을 원하는 클라이언트의 State 정보를 Lobby State로 변경하여 재전송한다.**

* Server – Client 통신



**[ Client *– Server* 통신 플로우 차트 ]**

* 순서

1. **서버는 윈속 초기화, 소켓 생성, bind를 완료한 후 listen을 한다.**
2. **클라이언트는 서버에 connect 요청을 한다.**
3. **서버는 connect 요청이 들어오면 accept하여 ProcessClient 스레드를 생성한다.**
4. **클라이언트는 서버에게 ~~캐릭터, 아이템의 위치정보, 생명,~~ 키 입력 정보를 send한다.**
5. **서버는 클라이언트에게서 데이터를 receive받고 갱신될 위치, 충돌체크 등을 계산한다**
6. **계산이 완료되면 갱신된 데이터를 클라이언트에게 send한다**
7. **클라이언트는 서버로부터 갱신된 데이터를 receive 받는다.**
8. **갱신된 데이터를 update시킨다**
9. **데이터를 이용하여 render한다**
10. **게임 중에 종료조건을 만족하면 서버가 state를 GAME\_OVER로 변경한 뒤 send한다**
11. **클라이언트는 state가 FINISH가 되면 결과를 출력하고 종료 또는 다시 시작한다.**

* Low Level Design
* Server – Client Protocol
* 윈속 버전: 2.2
* 프로토콜: TCP/IP
* 구조체

typedef struct Vec {

float x; float y;

};

// Client -> Server

typedef struct CtoSPacket {

~~Vec pos;~~

bool keyDown[4];

short life;

};

// Server -> Client

typedef struct StoCPacket {  
 short gameState;  
 DWORD time;

Vec p1Pos;

Vec p2Pos;

Vec itemPos[100];

short ~~characterID~~[100];

short **playerID**[100];

bool isVisible[100];

~~short life;~~

};

* Server
* 구조체

~~typedef struct Info {~~

typedef struct **SInfo** {

short connectedP; // 연결된 플레이어의 수

DWORD gameTime; // 게임 시간

~~Player p [2]; // 플레이어 구조체~~

~~ItemObj i[100]; // 아이템 구조체~~

**SPlayer \*players[2];** // 플레이어 구조체

**SItemObj \*items[100];** // 아이템 구조체

};

~~typedef struct Player {~~

typedef struct **SPlayer** {  
 short gameState; // 게임 상태를 나타내는 변수

Vec pos; // 플레이어 위치

bool keyDown[4]; // 클라이언트 키 입력 배열

short life; // 플레이어 생명

};

~~typedef struct ItemObj {~~

typedef struct **SItemObj** {

Vec pos; // 아이템 위치

Vec direction; // 아이템 발사 방향

float velocity; // 아이템 속도

short playerID; // 아이템을 먹은 플레이어의 아이디

bool isVisible; // 화면 표시 여부

};

* 함수
* ~~void~~ RecvFromClient(~~LPVOID arg~~, short PlayerID)

**int** RecvFromClient(**SOCKET client\_sock**, short PlayerID)

설명: 클라이언트로부터 패킷을 받는 함수  
 recvn() 함수를 사용하여 고정 크기의 데이터를 받아 서버의 Info 구조체를 갱신해준다.

인자: 각 스레드 함수에 해당하는 소켓을 함수의 첫 번째 인자로,   
 플레이어의 아이디를 두 번째 인자로 받는다.

* void SendToClient(~~LPVOID arg, short PlayerID~~)

void SendToClient()

설명: 클라이언트로 패킷을 보내는 함수.  
 RecvAndUpdateInfo() 스레드 함수에서 연산이 완료된 후   
 UpdatePackAndSend() 스레드 함수에서 호출하여 패킷 데이터를 클라이언트로 보내준다.

* DWORD WINAPI ~~ProcessClient~~(LPVOID arg)  
  DWORD WINAPI **RecvAndUpdateInfo**(LPVOID arg)  
  설명: 클라이언트로부터 데이터를 받아 연산하는 스레드 함수.  
   충돌 체크(아이템-플레이어, 총알-플레이어), 좌표 갱신(캐릭터, 총알), 게임 종료조건(시간, 생명)을  
   검사하고 계산하여, SInfo 구조체 값을 갱신한다.  
   HANDLE ~~hUpdateEvt~~ **hSendEvt** 이벤트 객체를 WaitForSingleObject() 함수로 신호 체크하여 UpdatePackAndSend() 함수와의 동기화를 한다.   
   플레이어를 구분할 수 있게 플레이어 ID를 부여한다.

* DWORD WINAPI ~~CalculateThread~~(LPVOID arg)  
  DWORD WINAPI **UpdatePackAndSend**(LPVOID arg)  
  설명: RecvAndUpdateInfo()에서 갱신한 값을 클라이언트로 보낼 패킷에 넣어준 후 모든 클라이언트에게  
   전송하는 스레드 함수.  
   HANDLE ~~hRecvEvt~~ **hUpdateInfoEvt** 이벤트 객체를 WaitForSingleObject() 함수로 신호 체크하여 RecvAndUpdateInfo() 함수와의 동기화를 한다.
* Client
* 구조체

~~typedef struct Info {~~

typedef struct **CInfo** {

short gameState; // 게임 상태를 나타내는 변수

DWORD gameTime; // 게임 시간

**short life; // 플레이어 생명**

~~Player p[2]; // 플레이어 구조체~~

Vec playersPos[2]; // 플레이어 좌표들

~~ItemObj i[100]; // 아이템 구조체~~

**CItemObj** i[100]; // 아이템 구조체

};

~~typedef struct ItemObj {~~

typedef struct **CItemObj** {

Vec pos; // 아이템 위치

bool isVisible; // 화면 표시 여부

**short playerID; // 아이템을 먹은 플레이어ID**

};

* 함수
* void SendToServer(SOCKET sock)

설명: 서버에 데이터를 전송하는 함수.   
 서버에 CtoSPacket 구조체를 보낸다.

인자: 서버의 정보를 담은 소켓 구조체를 인자로 받는다.

* void RecvFromServer(SOCKET sock)

설명: 서버에서 데이터를 받아오는 함수  
 서버로부터 Info 구조체를 받아온다.

인자: 서버의 정보를 담은 소켓 구조체를 인자로 받는다.

* 역할 분담
* 김정현: 서버 프레임워크 구현, 서버 내의 클라이언트 처리 스레드 함수 개발

서버 측의 송신 함수 개발, 서버 내의 연산 처리 스레드 함수 구현(좌표 계산)

* 차종원: 클라이언트 프레임워크 구현, 클라이언트 측의 송신 함수 개발

서버 내의 연산 처리 스레드 함수 구현(충돌 체크 및 탄성 구현)

* 박하연: 클라이언트 리소스 제작 및 수정, 클라이언트 측의 수신 함수 개발,

서버 내의 클라이언트 처리 스레드 함수 개발, 서버 측의 수신 함수 개발

서버 내의 연산 처리 스레드 함수 구현(종료 조건)

* 개발 환경
* IDE : Visual Studio 2017
* 운영체제 : Windows10 64bit
* Lib : ws2\_32, OpenGL
* VCS : GitHub (https://github.com/natsnatsmon/AENG-GYUNG)
* 언어 : C++
* 개발 일정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 화 | 수 | 목 | 금 | 토 | 일 |
| **10/23** | **10/24** | **10/25** | **10/26** | **10/27** | **10/28** |
| 박하연 | 계획서 초안 논의 |  |  | 계획서 회의 | High Level   서버-클라 순서도 작성 | Low Level 작성 |
| 차종원 | 계획서 초안 논의 |  |  | 계획서 회의 | 어플리케이션 기획 작성 | Low Level 작성 |
| 김정현 | 계획서 초안 논의 |  |  | 계획서 회의 | High Level  플로우 차트 작성 | Low Level 작성 |

* 1주차

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
| **10/29** | **10/30** | **10/31** | **11/1** | **11/2** | **11/3** | **11/4** |
| 김정현 | 개발 일정 작성, 계획서 마무리 | 계획서  심사 |  | 서버 프레임워크 구현 | |  | 팀 회의  및  검토 |
| 차종원 |  | 클라이언트 프레임워크 구현 (1차) |  | 클라이언트 프레임워크 구현(2차) |
| 박하연 |  |  | 리소스 수정 및 제작 | |

* 2주차

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
| **11/5** | **11/6** | **11/7** | **11/8** | **11/9** | **11/10** | **11/11** |
| 김정현 | UpdatePosition() 구현(1차) | hRecvEvt 생성,  ProcessClient() 내 배치 |  | 개인별 구현 검토 및 보완 | UpdatePosition() RecvFromClient() 테스트 및 보완 |  | 3주차 구현 내용 통합 테스트 및 보완 |
| 차종원 | SendToServer() 구현(1차) | SendToServer() RecvFromClient() 통신 테스트 및 보완 (1차) |  | SendToServer() 구현(2차) |  | SendToServer() RecvFromClient() 통신 테스트 및 보완 (2차) |
| 박하연 | RecvFromClient() 구현(1차) |  |  | UpdatePosition() RecvFromClient() 테스트 및 보완 |

* 3주차

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
| **11/12** | **11/13** | **11/14** | **11/15** | **11/16** | **11/17** | **11/18** |
| 김정현 | SendToClient() 구현 (1차) | Client 통신 thread 구현 |  | UpdatePosition() CollisionCheck() 테스트 및 보완 | SendToClient()  RecvFromServer() 통신 테스트 및 보완(1차) |  | 4주차 구현 내용 통합 테스트 및 보완 |
| 차종원 | CollisionCheck() 구현(1차) | |  |  | 개인별 구현 검토 및 보완 |
| 박하연 | RecvFromServer() 구현 (1차) | 개인별 구현  검토  및 보완 |  |  | SendToClient()  RecvFromServer() 통신 테스트 및 보완(1차) | RecvFromServer() 구현 (2차) |

* 4주차

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
| **11/19** | **11/20** | **11/21** | **11/22** | **11/23** | **11/24** | **11/25** |
| 김정현 | ~~hRecvEvt 생성,~~  ~~Process Client() 내 배치~~ | ~~hUpdateEvt 생성, CalcThread()내 배치~~ |  | 개인별 구현 검토 및 보완 | 소켓옵션 설정 및 테스트 |  | 5주차 구현 내용 통합 테스트 및 보완 |
| 클라이언트, 클라이언트 간의 렌더링 동기화 테스트 및 보완 | |
| 차종원 | ~~충돌 시 탄성력 계산~~  CollisionCheck() 구현(2차) | |  | ~~탄성력 적용~~ 충돌 후처리 |  | 개인별 구현  검토 및 보완 |
| 박하연 | ~~TimeCheck() 구현~~ | ~~개인별 구현 검토 및 보완~~ |  |  | ~~LifeCheck()~~ TimeCheck() 구현 | LifeCheck() GameEndCheck() 구현 |
| 클라이언트, 클라이언트 간의 렌더링 동기화 테스트 및 보완 | |

* 5주차

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
| **11/26** | **11/27** | **11/28** | **11/29** | **11/30** | **12/1** | **12/2** |
| 김정현 | ~~UpdatePosition() 내 탄성력 적용 추가~~ | ~~탄성력 적용한 UpdatePosition() 구현 및 테스트~~ |  | 개인별  구현 검토 및  보완 | ~~SendToClient()~~  ~~RecvFromServer()~~  서버 - 클라이언트 통신 테스트 및 보완(2차) |  | 6주차 구현 내용 통합 테스트 및 보완 |
| 플레이 도중 플레이어가 나갔을 때 gameState 전환 및 BugFix | |
| 차종원 | ~~SendToServer() RecvFromClient()~~ 서버 - 클라이언트 통신 테스트 및 보완 | ~~탄성력 적용한 UpdatePosition() 구현 및 테스트~~  CollisionCheck() BufFix |  | 최종 수정 |  | 개인별  구현 검토 및 보완 |
| 박하연 | 개인별 구현  검토 및 보완 |  |  | ~~SendToClient()~~  ~~RecvFromServer()~~  서버 - 클라이언트 통신 테스트 및 보완(2차) | ~~최종 수정~~  TimeRenderer()구현 |

* 6주차

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
| **12/3** | **12/4** | **12/5** | **12/6** | **12/7** | **12/8** | **12/9** |
| 김정현 | 통합 연동 테스트 및  코드 최적화 | |  | 개인별 구현 검토 및 보완 | 최종 테스트  및  미비점 보완 | |  |
| 차종원 |  |  |
| 박하연 |  |  |

* 7주차

|  |  |  |
| --- | --- | --- |
|  | 월 | 화 |
| **12/10** | **12/11** |
| 김정현 | 최종 심사  준비 | 최종 심사 |
| 차종원 | 최종 심사 |
| 박하연 | 최종 심사 |

* 8주차
* 개발 보고서

1주차

* 목표

1. 계획서 작성

* 내용
* 계획서의 일정부분을 제외한 나머지 부분을 작성하였다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 화 | 수 | 목 | 금 | 토 | 일 |
| **10/23** | **10/24** | **10/25** | **10/26** | **10/27** | **10/28** |
| 박하연 | 계획서 초안 논의 |  |  | 계획서 회의 | High Level   서버-클라 순서도 작성 | Low Level 작성 |
| 차종원 |  |  | 어플리케이션 기획 작성 |
| 김정현 |  |  | High Level  플로우 차트 작성 |

* 일정

( 완료 : 초록색 칸, 미비 : 노란색 칸, 일정 연기 : 빨간색 칸 , 일정 변경 : 빨간색 글씨)

2주차

* 목표

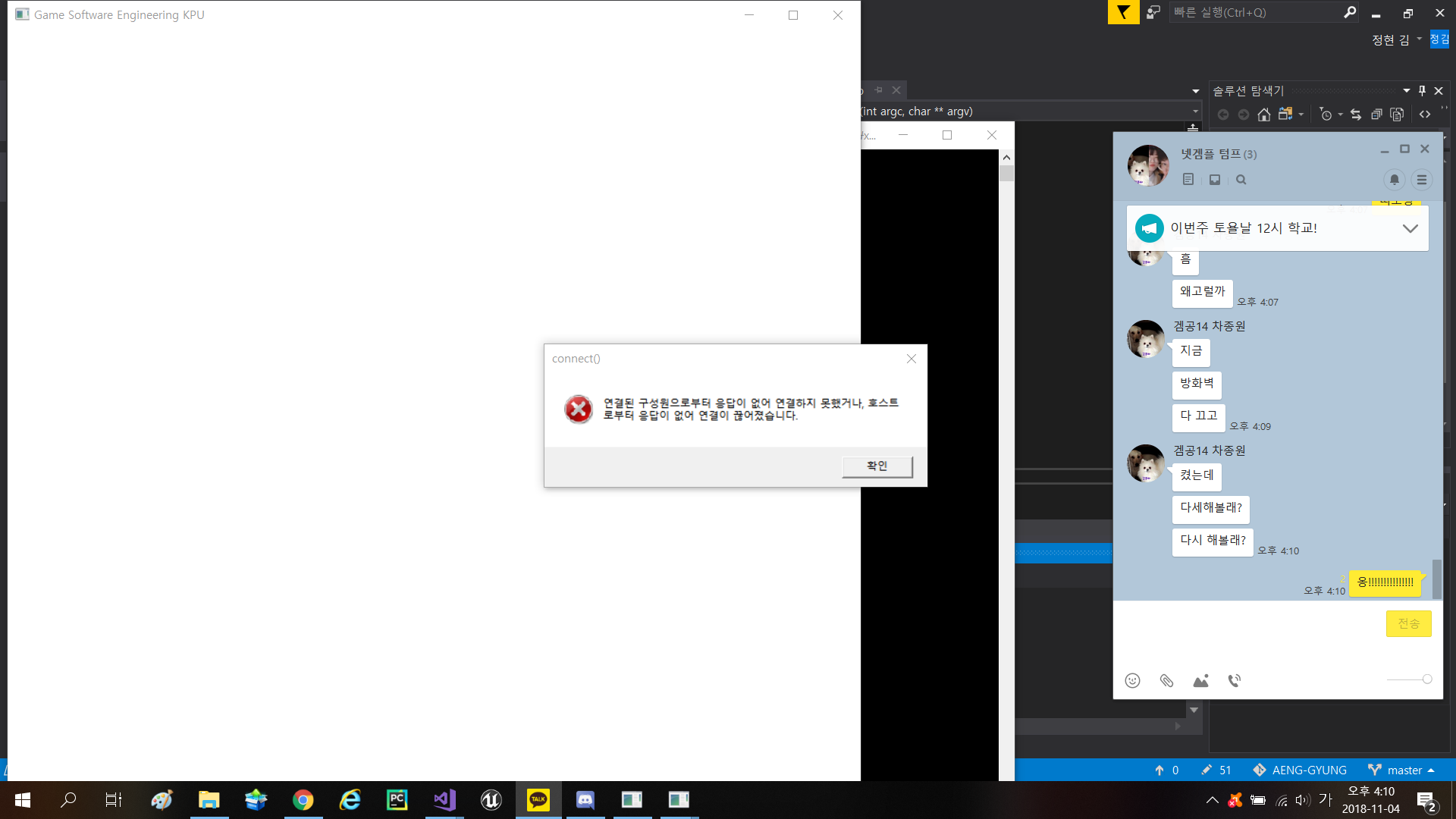
1. 계획서 마무리 및 심사 통과
2. 서버, 클라이언트의 프레임워크 구현 완료
3. 서버 – 클라이언트 접속 확인
4. 리소스 제작 및 적용 완료

* 내용
* **11월 1일**

계획서 마무리 후 심사에 통과하여 프레임워크 구현과 리소스 제작 작업을 진행하였다.

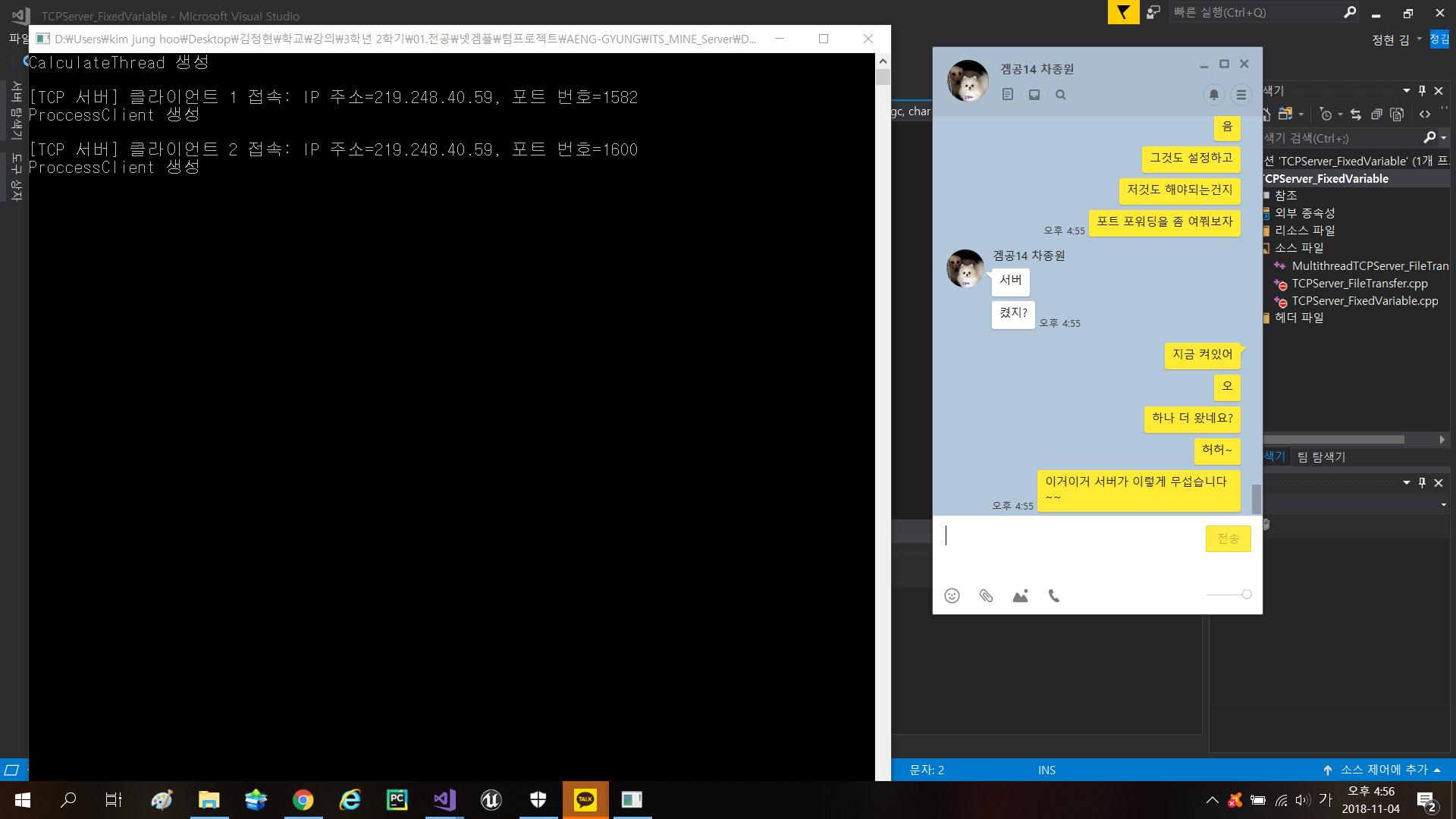
* **11월 4일**

1. 서버와 클라이언트 연결 확인

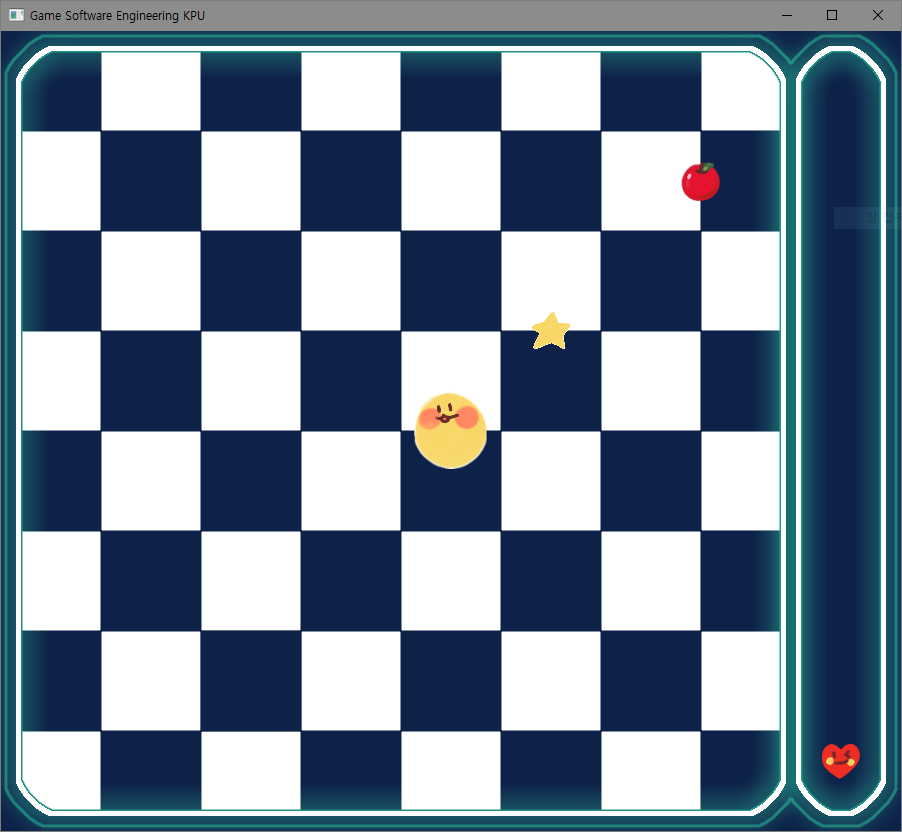


문제점 : 서버와 클라이언트 사이 connect() 문제가 발생하였다.

해결 방법 : 공유기 포트포워딩을 이용하여 해결하였다.



1. 리소스 제작 및 적용 확인



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
| **10/29** | **10/30** | **10/31** | **11/1** | **11/2** | **11/3** | **11/4** |
| 김정현 | 개발 일정 작성, 계획서 마무리 | 계획서  심사 |  | 서버 프레임워크 구현 | |  | 팀 회의  및  검토 |
| 차종원 |  | 클라이언트 프레임워크 구현 (1차) |  | 클라이언트 프레임워크 구현(2차) |
| 박하연 |  |  | 리소스 수정 및 제작 | |

* 일정

( 완료 : 초록색 칸, 미비 : 노란색 칸, 일정 연기 : 빨간색 칸 , 일정 변경 : 빨간색 글씨)

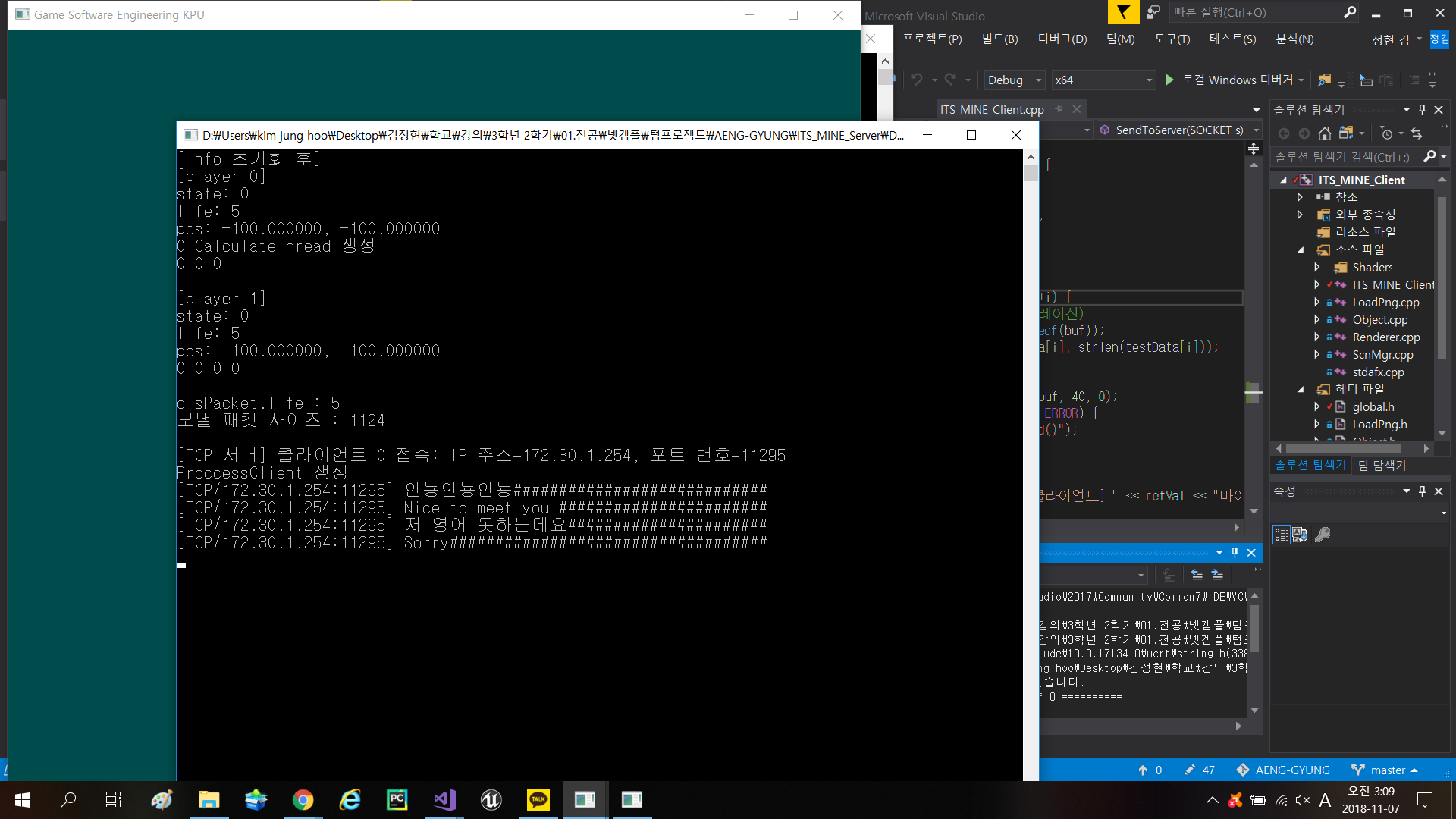
3주차

* 목표

1. UpdatePosition() 구현
2. SendToServer() 구현
3. RecvFromClient() 구현
4. Client -> Server 데이터 통신 확인
5. Receive받은 데이터를 이용해 위치 Update가 잘 되는지 확인

* 내용
* **11월 6일**

SendToServer(), RecvFromClient() 구현 후 임의의 dummy data를 이용한 통신 테스트 진행



[ 서버가 클라이언트로부터 dummy data를 받은 모습 ]

* **11월 8일**

Server <-> Client 간의 원활한 데이터 통신을 위하여 이벤트 객체의 생성과 배치를 일정(11/19,20)보다 앞당겨 진행하였다.

* **11월 9일**

UpdatePosition(), RecvFromClient()의 구현 후 통신 테스트를 진행

테스트 방법 :

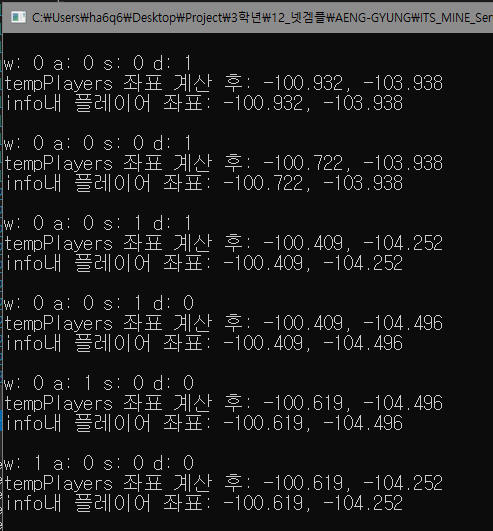
1. 클라이언트가 키 입력 정보를 서버에 보낸다.
2. 서버는 클라이언트가 보낸 정보를 바탕으로 서버에 저장 되어있는 객체들의 좌표를 옮겨준다.
3. 계산 후의 좌표와 서버에 저장 되어있는 객체의 좌표를 비교한다.

* **11월 10일**

Server -> Client 통신 테스트 진행

테스트 방법 : 11월 9일에 진행한 테스트와 동일

에코서버가 아닌 김정현 학우의 집에서 서버를 열고 박하연 학우의 집에서 클라이언트로 접속하여 통신을 진행하였다



[ 좌표 갱신 테스트 ]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
| **11/5** | **11/6** | **11/7** | **11/8** | **11/9** | **11/10** | **11/11** |
| 김정현 | UpdatePosition() 구현 | |  | hRecvEvt 생성,  ProcessClient() 내 배치 | UpdatePosition() RecvFromClient() 테스트 및 보완 |  | 3주차 구현 내용 통합 테스트 및 보완 |
| 차종원 | SendToServer() 구현(1차) | SendToServer() RecvFromClient() 통신 테스트 및 보완 (1차) |  | SendToServer() 구현(2차) |  | SendToServer() RecvFromClient() 통신 테스트 및 보완 (2차) |
| 박하연 | RecvFromClient() 구현 |  |  | UpdatePosition() RecvFromClient() 테스트 및 보완 |

* 일정

( 완료 : 초록색 칸, 미비 : 노란색 칸, 일정 연기 : 빨간색 칸 , 일정 변경 : 빨간색 글씨)

4주차

* 목표

1. SendToClient() 구현
2. Player – Player , Player – Wall CollisionCheck() 구현
3. RecvFromClient() 구현
4. Update된 위치의 충돌체크 확인
5. Server -> Client 데이터 통신 확인

* 내용
* **11월 12일**

SendToClient(), RecvFromServer() 구현 후 임시 통신테스트를 진행

문제점 : 클라이언트에 통신을 위한 thread가 누락되어 실패하였다.

해결 방법 : 일정을 수정하여 11월 13일에 Client 통신 thread 구현을 통해 해결하였다.

* **11월 15일.**

UpdatePosition()을 통해 갱신된 좌표와 P\_P\_CollisionCheck()를 이용한 충돌체크 테스트를 진행



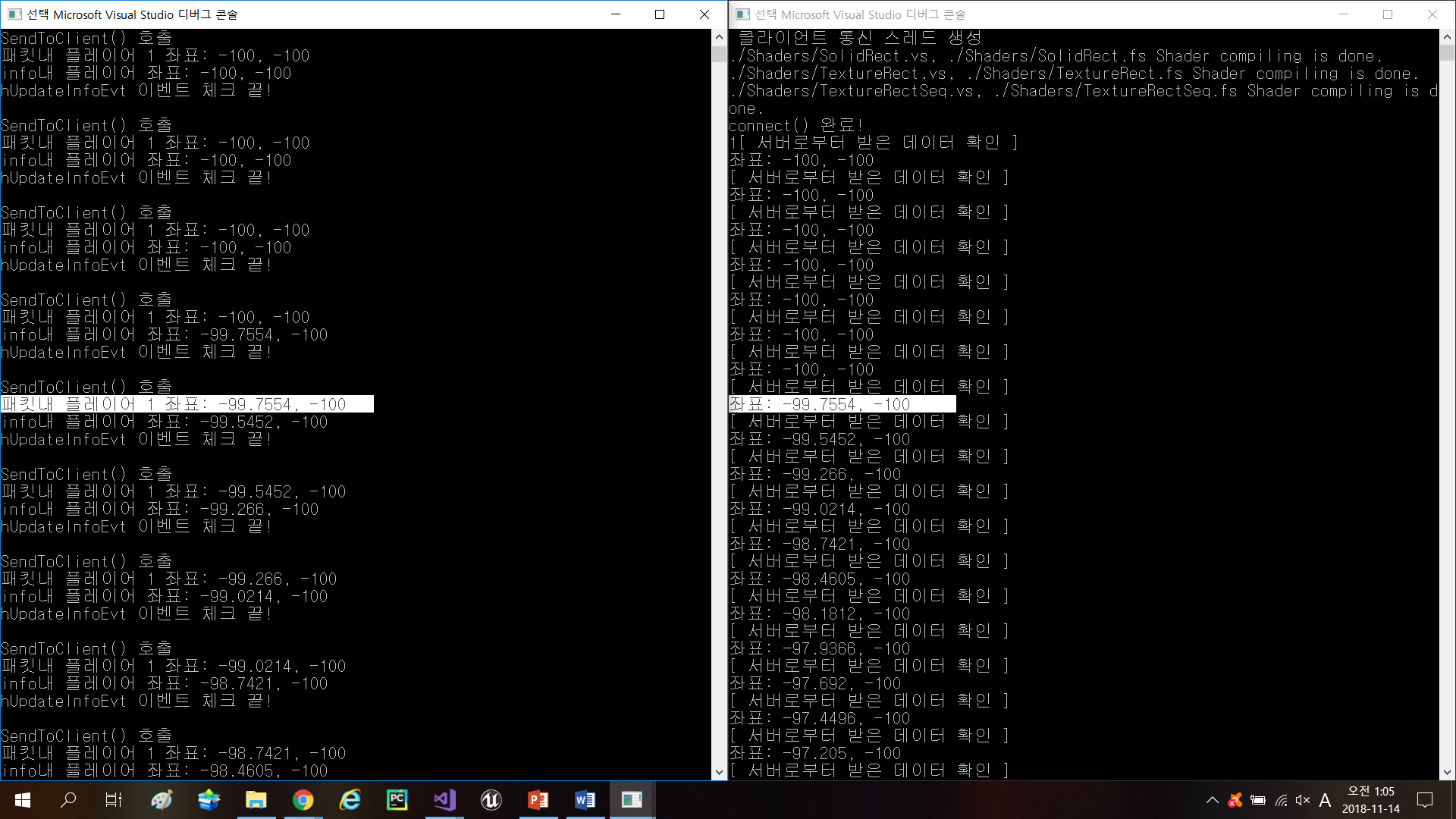
[ 캐릭터 – 캐릭터 간 충돌 체크 확인 ]

* **11월 16일**

Server <-> Client의 데이터 통신 테스트를 진행

테스트 방법 :

1. 클라이언트가 키 입력을 서버에 보낸다.
2. 서버가 클라이언트에게 받은 키 입력을 토대로 좌표를 계산하여 갱신한다.
3. 서버가 갱신된 좌표를 출력하고 클라이언트에게 보낸다.
4. 클라이언트는 서버에게 갱신된 좌표를 받아 출력하여 서버와 비교한다.



[ 서버(왼쪽), 클라이언트(오른쪽)의 출력 결과 ]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
| **11/12** | **11/13** | **11/14** | **11/15** | **11/16** | **11/17** | **11/18** |
| 김정현 | SendToClient() 구현 | Client 통신 thread 구현 |  | UpdatePosition() CollisionCheck() 테스트 및 보완 | SendToClient()  RecvFromServer() 통신 테스트 및 보완 |  | 4주차 구현 내용 통합 테스트 및 보완 |
| 차종원 | CollisionCheck() 구현(1차) | |  |  | 개인별 구현 검토 및 보완 |
| 박하연 | RecvFromServer() 구현 | 개인별 구현  검토  및 보완 |  |  | SendToClient()  RecvFromServer() 통신 테스트 및 보완 | RecvFromServer() 구현 (2차) |

* 일정

( 완료 : 초록색 칸, 미비 : 노란색 칸, 일정 연기 : 빨간색 칸 , 일정 변경 : 빨간색 글씨)

5주차

* 목표

1. 클라이언트 – 클라이언트간의 렌더링 동기화
2. GameEndCheck() 구현
3. Player – Item, Player – Bullet CollisionCheck() 구현 및 후처리

* 내용
* **11월 19일**

서버와 클라이언트 2대의 동기화가 제대로 이루어지지 않는 문제를 발견하였다.

11월 25일까지도 해결하지 못하였다.



[ 11월 23일 동기화 오류 화면 ]

문제점 : High Level에서 계획한 동기화 순서로 진행되지 않음, 클라이언트 간의 렌더링 동기화가 제대로 되지 않음

예상 원인 분석 :

1. WaitFor함수의 대기시간이 INFINITE인 것
2. 이벤트 순서 꼬임
3. 무리한 이벤트 사용 (Critical Section 고려)

해결 방법 : 아직 해결하지 못하여 시도해본 방법을 적음

1. TimeOut 옵션 : 게임 시작 후 약 5초간 플레이어의 이동이 안됨, 이동 속도 매우 느려짐
2. Critical Section : 렌더링이 이전보다 더욱 느려지는 상황 발생

* **11월 22일**

UpdatePosition()을 통해 갱신된 좌표와 P\_P\_CollisionCheck()를 이용한 충돌체크 테스트를 진행



* **11월 24일**

GameEndCheck()를 구현하여 제한 시간(100초)을 초과하였을 때 또는 플레이어의 생명이 어느 한쪽이라도 0이 되었을 경우 gameState를 GameOver 상태로 변경

동기화 문제가 해결되지 않아 한 개의 클라이언트로 TimeCheck()만 테스트하였고 종료되는 것을 확인하였다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
| **11/19** | **11/20** | **11/21** | **11/22** | **11/23** | **11/24** | **11/25** |
| 김정현 | 클라이언트, 클라이언트 간의 렌더링 동기화 테스트 및 보완 | |  | 개인별 구현 검토 및 보완 | 소켓옵션 설정 및 테스트 |  | 5주차 구현 내용 통합 테스트 및 보완 |
| 차종원 | CollisionCheck() 구현(2차) | |  | 충돌 후처리 |  | 개인별 구현  검토 및 보완 |
| 박하연 | 클라이언트, 클라이언트 간의 렌더링 동기화 테스트 및 보완 | |  |  | TimeCheck() 구현 | LifeCheck() GameEndCheck() 구현 |

* 일정

( 완료 : 초록색 칸, 미비 : 노란색 칸, 일정 연기 : 빨간색 칸 , 일정 변경 : 빨간색 글씨)

6주차

* 목표

1. 공백

* 내용

1. 아직 5주차가 오지 않았슴니다..

7주차

* 목표

1. 공백

* 내용

1. 아직 5주차가 오지 않았슴니다..

8주차

* 목표

1. 공백

* 내용

1. 아직 8주차가 오지 않았슴니다..