|  |
| --- |
| 네트워크 게임 프로그래밍 |
| 개구리 : 비상 |
| [문서 부제 입력] |

|  |
| --- |
|  |

목차

1. 애플리케이션 기획
   1. 게임 컨셉
   2. 플레이 방식
   3. 게임 소개
   4. 플레이 방법
2. High-Level 디자인

2-1. Login

2-2. Lobby

2-3. Ingame

1. Low-level 디자인

3-1. server

3-2. client

3-3. protocol

1. 역할 분담
2. 개발 일정
3. 개발 환경

애플리케이션 기획

게임 컨셉: 발판을 이용해 적들의 공격을 피해가며 두 명의 플레이어가 모두 버튼을 밟아 스테이지를 클리어 하는 협동 게임

플레이 방식: 2인 협동 멀티 플레이 게임

게임소개: 맵의 특정 지역에 버튼 2개가 배치되어 있다. 맵에는 원거리 공격 몬스터와 지정된 형식으로 돌아다니는 몬스터가 있다. 플레이어는 마우스 클릭을 통해 원하는 위치에 발판을 생성할 수 있으며 생성 가능 개수에 제한이 있다.

생성된 발판과 기존 발판을 이용하여 2개의 버튼에 도달하여 누르고 있을 시 스테이지가 클리어 된다. 스테이지는 3개 있으며 진행할수록 난이도가 상승한다. 플레이어가 몬스터와 충돌하거나 몬스터의 공격에 맞을 시 생성된 발판은 초기화되며 모든 플레이어가 출발 지점으로 돌아간다.



플레이 방법:

1. 게임 접속

2. 아이디 입력을 통한 로그인

3. 로그인 후 로비에서 대기

4. 두 명의 플레이어가 모이면 게임 시작

5. 방향키와 스페이스바를 이용해 움직인다.

6. 마우스 클릭을 통해 발판을 생성한다.

7. 두 명의 플레이어가 모두 발판을 밟을 시 스테이지 클리어 or 게임 승리

8. 플레이어가 몬스터와 혹은 공격에 충돌 시 발판 및 플레이어 위치 초기화

High-Level디자인

TCP를 이용한 멀티 스레드

Login true

<server> <client> false

InitClient()

InitServer()

SendID()

RecvID()

RecvID()

ChekID()

CheckID()

SaveID()

.

GoToLobby()

GoToLobby()

InitServer() 와InitClient()로 소켓을 생성하고 연결한다.

클라이언트가 서버에 접속을 요청하고 서버는 요청을 수락한다.

각 클라이언트는 자신이 사용할 ID를 입력하고 서버에 전송한다.

서버는 클라이언트가 입력한 ID를 받아서 이미 존재하는 ID인지를 파악해 그 결과를 클라이언트에 전송하고, 존재하지 않는 ID라면 저장하고 로비로 넘어간다.

클라이언트에서는 서버에서 받은 ID의 중복여부를 가지고 이미 존재하는 ID라면 ID입력 후 전송을 반복한다.

Lobby

<server> <client>

GoToLobby()

GoToLobby()

RecvReady()

SendReady()

RecvReady()

ReadyCheck()

GameStart()

GameStart()

클라이언트가 준비되었다는 상태를 서버에 전송해준다.

서버는 클라이언트의 레디 상태를 받고 모든 클라이언트가 준비되었는지 판단한 후 준비가 되었다면 클라이언트에 준비가 완료되었다고 전송하고 게임을 실행한다.

Ingame

<server> <client>

GameStart()

GameStart()

InitPlayerInfo()

InitPlayerInfo()

InitObjectInfo()

InitObjectInfo()

<server> <client>

GameStart()

GameStart()

SendKey()

RecvKey()

InputKey()

RecvPlayerMove()

Render()

SendPlayerPos()

RecvPlayerPos()

<server> <client>

GameStart()

GameStart()

Inputmouse()

CheckObject

Collision()

RecvSetObject()

SendObjectInfo()

RecvObjectInfo()

SendObjectInfo()

saveblock()

Render()

게임을 시작하면 서버로부터 오브젝트와 플레이어의 기존 위치 정보를 받는다.

클라이언트는 키보드 입력으로 맵을 이동하고 마우스 클릭으로 블록을 생성한다.

클라이언트의 키보드 값을 서버에 보내주면 서버가 위치를 계산하고 충돌체크를 진행하여 다시 서버로 보내주고, 클라이언트는 그 정보를 이용해 플레이어를 렌더링 한다.

클라이언트의 마우스가 클릭되면 먼저 클라이언트에서 기존의 블록과 위치가 겹치는지 검사한다. 위치가 겹치지 않는다면 서버로 좌표를 넘겨주고 그 좌표를 모든 클라이언트로 전송해 각 클라이언트가 해당 위치에 블록을 렌더링 한다.

또한 몬스터 혹은 몬스터의 공격이 플레이어와 충돌했을 경우에도 서버에서 충돌 검사를 수행하여 그 결과를 클라이언트에게 전송하면 모든 블록과 클라이언트의 위치가 초기화 된다.

Low-Level 디자인

- Server

~~void InitServer() : 클라이언트와의 소켓 연결~~

~~void RecvID(): 클라이언트로부터 id정보를 받는다.~~

~~bool CheckID(const char\* ID) : 클라이언트의 아이디 중복 검사를 한다.~~

~~void SaveID() : 클라이언트의 아이디를 저장~~

~~void GoToLobby(): 로그인이 완료되면 로비로 들어간다.~~

~~Void RecvReady(): 플레이어의 레디 상태를 받는다.~~

~~void ReadyCheck(int Ready) : 클라이언트들이 레디 상태인지 확인한다.~~

~~void InitPlayerInfo(): 게임 시작 시 플레이어에 대한 정보를 클라이언트로 전송~~

~~void InitObjectInfo(): 게임 시작 시 오브젝트에 대한 정보를 클라이언트로 전송~~

void InitGameObject(): 게임 시작 시 발판, 몬스터, 장애물의 위치를 저장

~~void SendPlayerPos(float x, float y) : 플레이어의 위치를 전송~~

~~void RecvPlayerMove() : 플레이어의 이동 계산~~

~~void SendPlayerInfo(Player player) : 플레이어의 정보를 전송~~

~~void RecvPlayerInfo() : 플레이어의 정보를 받는다.~~

~~void RecvSetObject() : 설치한 오브젝트의 위치를 받는다.~~

~~void SendObjectInfo(Object obj) : 오브젝트의 정보를 보낸다.~~

~~void CollisionObject(Object pos) : 캐릭터의 이동, 몬스터의 공격에 대한 충돌 처리를 한다.~~

~~void CollisionBullet(Bullet pos) : 총알과 캐릭터의 충돌 검사를 한다.~~

~~void BulletPos(Bullet bullet) : 총알의 위치를 계산한다.~~

~~DWORD WINAPI ProcessClient() : 클라이언트 접속을 관리한다.~~

DWORD WINAPI Client\_Thread() : 클라이언트 접속을 관리한다.

~~DWORD WINAPI FrameThread() : 프레임과 타이머를 갱신한다.~~

~~class Server~~

~~{~~

~~public:~~

~~Server();~~

~~~Server();~~

~~public:~~

~~void err\_display(int err\_num);~~

~~void err\_quit(const char\* msg);~~

~~int recvn(SOCKET s, char\* buf, int len, int flags);~~

~~int InitServer();~~

~~void LoginServer();~~

~~void LobbyServer();~~

~~void InGame();~~

~~void UpdateStatus();~~

~~void UpdateCollision();~~

~~};~~

* Client

~~void InitClient(): 윈속 초기화, 소켓 생성, 서버와 소켓 연결~~

~~void SendID(): id를 입력 받아 서버에 전송~~

~~void RecvID(): 서버로부터 id가 중복인지 정보를 받음~~

~~void CheckID(bool ServerCheckID): 서버로부터 받은 id체크 여부를 가지고 다시 입력받을 것인지, 게임을 시작할 것인지 검사~~

~~void GoToLobby(): 로그인이 완료되면 로비로 들어간다.~~

~~void SendReady(): 서버에게 레디 상태인 것을 알림~~

~~void RecvReady(): 서버에게서 2명의 플레이어가 모였다는 정보를 받음~~

~~void GameStart(): 게임 화면으로 이동~~

~~void InitPlayerInfo(): 게임 시작 시 플레이어에 대한 정보를 서버에서 받아 저장~~

~~void InitObjectInfo(): 게임 시작 시 오브젝트에 대한 정보를 서버에서 받아서 저장~~

~~void InputKey(): 게임 화면에서 플레이어가 키보드를 눌렀으면 어떤 키를 눌렀는지 판단~~

~~void SendKey(): 키보드에서 눌려진 키를 서버로 전송~~

~~void RecvPlayerPos(): 서버에서 전송받은 플레이어의 위치를 저장~~

~~void InputMouse(): 마우스 클릭 시 마우스의 좌표 저장~~

~~bool CheckObjectCollision(float x, float y): 클릭한 마우스의 좌표를 가지고 그곳에 이미 오브젝트가 설치되어 있는지 검사~~

~~void SendObjectInfo(): CheckObjectCollision이 true(겹치지 않는다)면 마우스의 좌표를 서버로 전송~~

~~void RecvObjectInfo(): 다른 플레이어가 설치한 오브젝트의 정보를 서버로부터 받는다~~

~~void RecvPlayerInfo(): 다른 플레이어의 정보를 서버로부터 받는다.~~

~~void RecvHit(): 플레이어가 몬스터 혹은 총알에 충돌했는지를 서버가 검사하고 초기화된 위치를 서버로부터 받는다.~~

~~void Render(): 객체들의 정보를 이용해 화면을 그린다.~~

DWORD WINAPI Recv\_Thread(LPVOID arg): 클라이언트 동작

**- Protocol**  
#pragma once  
~~const short SERVER\_PORT = 4000;~~  
const short SERVER\_PORT = 9000;

~~const int MAX\_NAME\_SIZE = 20;  
const int MAX\_USER = 10;  
const char CS\_PACKET\_LOGIN = 1;  
const char CS\_MOVE\_INPUT = 2;  
const char CS\_MOUSE\_INPUT = 3;  
const char CS\_CLOSE\_ANIMATION = 4;  
const char SC\_PACKET\_LOGIN\_OK = 1;  
const char SC\_PACKET\_PUT\_OBJECT = 2;  
const char SC\_PACKET\_REMOVE\_OBJECT = 3;  
const char SC\_SET\_ANIMATION = 4;  
const char SC\_UPDATE = 5;~~

~~const char를 이용해 패킷을 구별 할 수 있도록 한다.~~

~~struct cs\_packet\_login {  
 unsigned char size;  
 char packet\_type;  
 char name[MAX\_NAME\_SIZE];  
}; 클라이언트에서 로그인을 하기위해 서버로 보내는 패킷  
struct sc\_login\_ok {  
 unsigned char size;  
 char packet\_type;  
 int id;  
 short x, y;  
}; 서버에서 로그인 확인을 클라이언트로 보내는 패킷~~

~~struct cs\_move\_input {  
 unsigned char size;  
 char packet\_type;  
 int type;   
}; 클라이언트에서 움직임을 서버에 보내는 패킷  
struct cs\_mouse\_input{  
 unsigned char size;  
 char packet\_type;  
 float x,y;  
}; 클라이언트에서 마우스 좌표값을 서버로 보내는 패킷~~

~~struct cs\_update{   
unsigned char size;  
char packet\_type;  
bool isPush;  
bool isColl;  
float x,y;  
int anim\_type;~~

~~}; 서버에서의 연산으로 바뀐 게임의 전체적인 변경사항을 클라이언트로 받아주는 패킷~~

struct cs\_send\_keyinfo {

short id;

bool left = false;

bool right = false;

bool jump = false;

bool isClick = false;

float x = 0;

float y = 0;

}; 클라이언트에서 사용자가 입력한 키보드나 마우스의 정보를 서버로 보내주는 패킷

~~struct sc\_put\_object {  
 unsigned char size;  
 char packet\_type;  
 float x,y;  
 int type;~~

~~}; 서버에서 생성될 객체의 타입과 위치값을 클라이언트로 보내는 패킷~~

~~struct sc\_remove\_object {  
unsigned char size;  
char packet\_type;  
int type;  
int index;~~

~~}; 서버에서 제거될 객체의 타입과 해당 객체의 인덱스를 클라이언트로 보내는 패킷~~

~~struct sc\_set\_animation{  
unsigned char size;  
char packet\_type;  
int type;   
int animCount = 0;   
int maxCount;~~

~~}; 서버에서 객체의 애니메이션을 변경하기 위해 클라이언트로 보내는 패킷~~

~~struct sc\_update {   
 unsigned char size;  
 char packet\_type;  
 bool isColl;  
 bool isPush;  
 float x,y;  
 int anim\_type;~~

~~}; 서버에서 업데이트에 의해 연산된 값들을 클라이언트로 보내주는 패킷~~

struct sc\_send\_player\_id {

short id;

}; 서버에 몇 번째로 연결된 클라이언트인지를 클라이언트에게 보내는 패킷

struct sc\_send\_player {

short id;

int state;

float x;

float y;

int jumpCount;

int dir;

}; 서버에서 연산된 플레이어의 정보를 다시 클라이언트로 보내주는 패킷

struct sc\_obstacle {

float x;

float y;

}; 장애물의 움직임을 연산해서 서버에서 클라이언트로 보내주는 패킷

struct sc\_bullet {

float x;

float y;

bool isColl = false;

int type = 0;

int imageCount;

int imageSizeX;

int imageSizeY;

int anim;

bool isStart = false;

}; 총알의 모든 정보를 연산한 결과를 서버에서 클라이언트로 보내주는 패킷

struct sc\_update{

float x, y;

int type;

bool isClick;

bool isPush[2];

int AttackMonsterId;

bool clear = false;

int Stage;

}; 서버에서 연산한 값들을 한 번에 보낼 수 있는 것끼리 묶어서 클라이언트로 보내주는 패킷(발판 설치, 버튼 on/off, 공격한 몬스터의 id, 플레이어 충돌결과, 현재 스테이지)

struct sc\_start\_game {

bool gamestart;

int stage;

}; 2명의 클라이언트가 들어오고 게임 시작버튼을 눌렀을 때, 서버에서 클라이언트로 게임시작을 알려주는 패킷

**- 동기화**

DWORD WINAPI Client\_Thread ()에서 클라이언트의 접속을 관리하고 각 클라이언트의 정보를 각 클라이언트로 보내줄 때 ~~mutex~~ 임계영역을 사용하여 클라이언트 동기화를 한다. ~~또한 DWORD WINAPI FrameThread()에서 각 클라이언트의 이동속도와 총알 정보에 대해서 총알이 발사 될 때, 총알에 맞았을 때, 이 두 경우에 mutex를 사용하여서 동기화한다.~~

역할 분담

강상준 : 인 게임 클라이언트 제작 및 로비화면, 클라이언트에서 전달받은 데이터로 서버에서 연산 실행 후 클라이언트로 전송

유영준 : 서버 프레임워크 제작 및 서버 로그인, 서버에서 연산 후 클라이언트로 데이터를 보냄으로써 클라이언트와 통신

오성훈 : 로그인 화면 제작 및 클라이언트 로그인, 클라이언트에서 서버로 데이터를 보냄으로써 서버와 통신, 서버에서 데이터를 받아 클라이언트에 적용

개발 일정

|  |  |  |  |
| --- | --- | --- | --- |
|  | 강상준 | 유영준 | 오성훈 |
| 10/28 목 | 게임 선정 및 추진 계획서 작성 | | |
| 29 금 |
| 30 토 |
| 31 일 |
| 11/1 월 |
| 2 화 | 추진 계획서 제출 | | |
| 3 수 | 인 게임 제작 및 수정 | Class Server | 맵 제작 및 오브젝트 배치 |
| 4 목 | InGame() |
| 5 금 | InitServer() | ProcessClient() | InitClient() |
| 6 토 | 프로토콜 구현 | | |
| 7일 |
| 8 월 | 전체 피드백 및 진행사항 확인, 다음 주차 일정 수립 | | |
| 9화 | 계획서 검수 및 피드백 | | |
| 10 수 |  |  |  |
| 11 목 | 로비  SendReady() | 서버 로그인  RecvID() | 클라이언트 로그인  SendID() |
| 12 금 | RecvReady() | CheckID() | RecvID() |
| 13 토 | ReadyCheck() | SaveID() | CheckID() |
| 14 일 | RecvReady() |  |  |
| 15 월 | 전체 피드백 및 진행사항 확인, 다음 주차 일정 수립 | | |
| 16 화 | 클라이언트  InitPlayerInfo()  InitObjectInfo() | 서버  InitPlayerInfo()  InitObjectInfo() | InputKey()  SendKey() |
| 17 수 | SendPlayerPos() | SendPlayerInfo() | RecvPlayerPos() |
| 18 목 | RecvPlayerMove() | RecvPlayerInfo() | InputMouse()  CheckObjectCollision() |
| 19 금 |  |  | SendObjectInfo() |
| 20 토 | SendObjectInfo() | RecvSetObject() | RecvObjectInfo() |
| 21 일 |  |  |  |
| 22 월 | 전체 피드백 및 진행사항 확인, 다음 주차 일정 수립 | | |
| 23 화 | FrameThread() | | |
| 24 수 |
| 25 목 |
| 26 금 | BulletPos() | CollisionObject() |  |
| 27 토 | RecvHit() | CollisionBullet() | RecvPlayerInfo() |
| 28 일 |  |  |  |
| 29 월 | 전체 피드백 및 진행사항 확인 | | |
| 30 화 | 미 구현 부분 구현 | | |
| 12/1 수 |
| 2 목 | 최종 테스트 및 오류 검수 | | |
| 3 금 |
| 4 토 |
| 5 일 | 발견된 오류 수정 | | |
| 6 월 |
| 7 화 |
| 8 수 | 최종 본 제출 | | |

함수의 내용은 구현하였으나 코드들을 함수로 분리하지 않고, 각 함수에 해당하는 내용들을 역할에 맞게 구현하였다.

개발 환경

OS: Windows 10

IDE: Visual Studio 2019

API: Windows Socket API(WSA)

개발 언어: C/C++