네트워크 게임 프로그래밍 Term Project

계획서

게임공학과

2017180012 박주용

2017180029 이예준

2017182030 이수민(팀장)

목차

1. 애플리케이션 기획

* 게임 소개
* 게임 흐름

1. High-level 디자인

* Flow Chart
* ER Diagram
* 서버 구현 내용
* 클라이언트 구현 내용

1. Low-level 디자인

* 애플리케이션 프로토콜
* 함수 설명

1. 팀원 별 역할분담
2. 개발환경
3. 개발일정
4. 애플리케이션 기획

* 게임소개

1. 게임 이름: Just Jump
2. 게임 장르: 2D 플랫포머 게임
3. 게임 컨셉: 3단계 스테이지로 구성되어 있으며 장애물들을 피해 플랫폼들을 밟고 점프하며 마지막 스테이지까지 먼저 가는 사람이 승리하는 게임.
4. 조작 방법: 키보드 4방향키, 스페이스바(점프)

* 게임흐름

|  |  |  |
| --- | --- | --- |
| 화면 | 설명 | 게임 플레이 화면 |
| 로그인 | 유저가 게임 플레이 전에 회원가입과 로그인을 할 수 있는 로그인 화면이 나타난다.  ID를 입력하고 로그인을 누르면 해당 아이디로 플레이어 네임이 정해진다. |  |
| 로비 | 3명의 유저가 로비에 모두 모이면 10초 후 3명의 유저가 동시에 인게임으로 넘어간다. |  |
| 인 게임 | 게임이 시작되면 플레이어가 시작 위치와 HP를 받게 되고 점프, 이동을 통해 플랫폼을 받고 올라가며 장애물을 피하며 맨 위 포탈(목적지)까지 간다.  포탈에서 윗 방향 키를 누르면 다음 스테이지로 넘어간다.  장애물에 피격될 때, 장애물로 인해서 혹은 플랫폼을 밟지 못해서 낙하했을 때 HP가 깎이게 된다.  HP가 0이 되면 죽게 되고 사망 메시지가 화면에 뜬다. 확인을 누르면 HP가 리셋되고 해당 스테이지 초기 위치에서 다시 시작한다. |  |
| 게임  클리어 | 마지막 스테이지에서 포탈로 이동하면 게임 클리어 화면이 나온다.  모든 플레이어가 클리어 스테이지에 도착하면 게임종료 버튼을 눌러 게임을 종료한다.. |  |

1. High-level 디자인

* Flow Chart

1. 게임 서버



1. 클라이언트



* ER Diagram



1. Low-level 디자인

* 애플리케이션 프로토콜(데이터 송수신 패킷)

|  |  |
| --- | --- |
| 패킷 이름 | 설명 |
| **cs\_packet\_login**{  size,  type,  username[MAX\_NAME\_SIZE],  } | size만큼의 패킷을 받아서 type을 확인하고 cs\_packet\_login이면 username을 서버에 있는 클라이언트에 할당한다.  이때 서버는 클라이언트의 id, 즉 c\_id를 할당 받아서 클라이언트에게 할당을 하고 sc\_packet\_login\_ok를 클라이언트에게 보낸다. |
| **cs\_packet\_move**{  size, type, vk\_key  } | 플레이어가 ~~dir 방향으로 움직이고 싶다는 패킷을 서버가 받는다~~. vk\_key를 누른것을 서버로 보낸다. |
| cs\_packet\_keyup{  size,type,vk\_key  } | 플레이어가 vk\_key를 뗏다는걸 서버로 보낸다. |
| cs\_packet\_robby{  size,type  } | ?? 주용 검증필요 |
| cs\_packet\_die\_ok{  size,type  } | ?? 주용 검증필요 |

|  |  |
| --- | --- |
| 패킷 이름 | 설명 |
|  |  |
| **sc\_packet\_login\_ok**{  size, type, id, x, y, stage  } | size만큼의 패킷을 받아서 type을 확인하고, 클라이언트의 id와(name아님), 클라이언트(플레이어)의 시작 x, y좌표를 보내준다. 또한 player가 속할 stage 넘버도 보내준다. |
| **sc\_packet\_put\_object**{  size, type, id, x, y,  username[MAX\_NAME\_SIZE], state,  w, h, hp, dir, stealth  } | 클라이언트가 어떠한 정보를 가지고 사용자들을 표현해야 하는지 위치정보, username, 배열구조에서 사용할 클라이언트 id, 플레이어 상태, 캐릭터 이미지의 너비/높이, 플레이어 무적 유무, 플레이어 hp, 플레이어의 방향을 보내서 캐릭터를 설정한다. |
| **sc\_packet\_logout\_object**{  size, type, id  } | 클라이언트 id가 사라졌다는 패킷을 보내 다른 사람들의 화면에서 해당 id를 가진 플레이어를 없애 주는 역할이다. |
|  |  |
| **sc\_packet\_move\_process{**  size, type, id, x, y, state, h, dir, stealth, hp, rank  } | 플레이어의 이동 방향을 가지고 서버에서 이동처리를 해준 다음 그 결과를 클라이언트에게 알려준다.  Id: 클라이언트의 소켓 번호  x,y: 플레이어의 위치정보  state: 플레이어의 상태정보  h : 플레이어의 캐릭터 높이  dir : 플레이어가 보는 방향  stealth : 플레이어 무적 시간  Rank: 플레이어들의 좌표값을 토대로 순위를 매겨서 전송  hp : 플레이어의 hp |
| **~~sc\_packet\_stagenum~~**~~{~~  ~~size, type, id, stagenum~~  ~~}~~  **sc\_packet\_portal**{  size, type, stagenum  } | ~~플레이어들이 있는 stage의 값을 넘겨주는 역할을 한다.~~  ~~id : 클라이언트의 소켓 번호~~  플레이어가 포탈과 상호작용을 했을 때, 다음 스테이지로 넘어가기 위한 트리거 역할 패킷  stagenum : 플레이어가 속한 스테이지 넘버 |
| **sc\_packet\_robby{**  size, type, player\_cnt, countdown  } | player\_cnt: 로비에 플레이어가 몇 명이 모였는지 센다.  countdown: 10,9, …, 1 까지 카운트다운 |
|  |  |
| **sc\_packet\_object\_sync**{  size,type,objnum,mx,my,index  } | 서버에서 움직이는 장애물들의 좌표나, 공격타이밍을 클라이언트에게 동기화해주는 패킷  objnum 장애물이 몇 번째 장애물인지 숫자.  mx,my : gear장애물의 위치좌표 동기화  index : steam장애물의 index프레임 동기화 |
|  |  |
| **sc\_packet\_gamestart{**  size, type, stage, x, y, h, state, dir, stealth, COMMAND\_die  **}** | 주용이형 검증 필요. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

* 함수 설명(서버)

|  |  |
| --- | --- |
| 함수 이름 | 설명 |
| void CLIENT::**do\_send**(void\* packet, int bytes) | send를 하기 위해 buf를 만들고 buf에 packet을 담아서 send를 해주는 함수.  packet : send할 패킷이 담긴 packet내용을 가리키는 포인터  bytes : 몇 size의 packet을 보낼지의 크기 |
| void CLIENT::**do\_recv**() | 먼저 recv를 받고, recv된 데이터들을 모아서 패킷 재조합을 통해 하나의 패킷을 만든 다음 패킷을 처리하는 함수. |
| void CLIENT::ProcessPacket(unsigned char\* p) | 패킷을 받아서 그 패킷의 type을 분석하여 패킷을 처리하는 함수.  p : packet의 시작번지를 가리키는 포인터 |
| void **update**() | 게임로직 스레드에 들어가는 함수로, 플레이어의 이동 처리(CLIENT::update()), 오브젝트의 이동 처리(OBJECT::update()), 플레이어와 오브젝트의 충돌 및 상호작용 처리(adjustPlayer(), collp2o()) 등을 담당하는 함수. |
| void CLIENT::update() | 플레이어의 움직임 및 상태를 처리하는 함수이다. |
| void OBJECT::update() | 오브젝트의 움직임을 처리하는 함수이다. |
| void adjustPlayer(client, object, ocount) | 플레이어와 오브젝트 간의 상호 작용을 처리하는 함수이다. |
| void collp2o(client, object) | 플레이어와 오브젝트 간의 충돌 체크를 하는 함수이다. |
| void CLIENT::spike\_hurttime() | 플레이어가 가시 장애물에 부딪혔을 때 발생하는 로직을 처리하는 함수이다. |
| void CLIENT::stealthtime() | 플레이어가 장애물에 부딪혀서 일정 시간무적 상태가 되는 것을 처리하는 함수이다. |
| void CLIENT::hurt() | 플레이어가 장애물에 부딪혀서 hp가 감소하는 것을 처리하는 함수이다. |
| DWORD WINAPI ClientInputThread(LPVOID arg) | 플레이어의 키 입력을 받는 스레드이다.  Arg로 플레이어 넘버와 소켓을 받는다.  플레이어 3명 모두가 이 함수로 독자적인 스레드를 생성한다. |
| DWORD WINAPI GameLogicThread(LPVOID arg) | 게임 로직이 돌아가는 스레드이다.  다른 플레이어들에게 update()된 정보 broadcast |
| void InitClientInfo(int clientNum) | 클라와 연결이 되면 클라이언트의 정보를 초기화해준다. |
|  |  |

* 함수 및 클래스 설명(클라이언트)

|  |  |
| --- | --- |
| 클래스 이름 | 설명 |
| Network{  SOCKET s\_socket;  SOCKADDR\_IN server\_addr;  WSADATA WSAData;  Player\* mPlayer;  static Network\* mNetwork;  int prev\_size;  unsigned char\* buf[MAX\_BUF\_SIZE]  } | 네트워크에 필요한 기능들을 모아놓은 클래스.  server\_addr : 서버와 연결을 하기 위해 서버의 정보를 저장하기 위한 소켓주소구조체  s\_socket : 서버와 연결을 하기 위해 생성한 소켓  WSAData : 윈속을 시작하기 위해 필요한 변수.  mPlayer : 플레이어를 가리키는 포인터를 받아 네트워크 클래스에서 플레이어를 처리할 수 있게 해주는 포인터 변수  mNetwork : 싱글턴 기법을 사용하기 위한 변수.  buf : 클라이언트가 recv할 때 사용할 버퍼  prev\_size : 패킷 재조합시, 이전에 받았던 데이터 크기가 얼마나 되는지 확인하기 위한 변수. |

|  |  |
| --- | --- |
| 함수 이름 | 설명 |
| Network::Network() | 네트워크 클래스의 생성자로 멤버변수를 초기화 해주고, 윈속을 사용하기 위한 작업을(WSAStartup , socket)처리한다. |
| Network::~Network() | 네트워크를 끝마치는 곳으로 사용했던 자원을 반환하는(closesocket, WSACleanup, delete 등) 역할을 한다. |
| void Network::ConnectServer(const char\* server\_ip) | 서버의 ip주소를 받아 server\_addr를 초기화 해주고 connect함수를 통해 서버와 연결을 하는 함수. |
| int Network::C\_Recv() | 먼저 recv를 받고, recv된 데이터들을 모아서 패킷 재조합을 통해 하나의 패킷을 만든 다음 패킷을 처리하는 함수. |
| int Network::C\_Send(void\* packet, int bytes) | send를 하기 위해 buf를 만들고 buf에 packet을 담아서 send를 해주는 함수.  packet : send할 패킷이 담긴 packet내용을 가리키는 포인터  bytes : 몇 size의 packet을 보낼지의 크기 |
| void Network::ProcessPacket(unsigned char\* p) | 패킷을 받아서 그 패킷의 type을 분석하여 패킷을 처리하는 함수.  p : packet의 시작번지를 가리키는 포인터 |

1. 팀원 별 역할 분담

|  |  |  |
| --- | --- | --- |
| 박주용 | 이예준 | 이수민 |
| 서버로 이동 로직 옮기기 | 클라- 순위매기기 UI추가 | 서버로 충돌처리 옮기기 |
| 클라 - 로비에 모인 플레이어수와 플레이어3명이 모두 모이면 카운트다운 | 서버에서 순위로직 만들기 | 서버 프레임워크 구현 |
| 서버에서 로비에 몇 명 들어왔는지 판단하고 클라와 통신 | 게임종료 버튼. | 클라이언트ui 프레임워크구현 |

1. 개발 환경

* 운영체제: Window 10
* 컴파일러: Visual Studio 2019
* 데이터 정렬 방식: 서버/클라이언트 모두 LittleEndian
* 클라이언트 사용 라이브러리: Win32 API
* 사용 언어: C/C++
* 통신 프로토콜: TCP/IP
* 주소 체계: IPv4
* 버전 관리 시스템: GitHub

1. 개발 일정

|  |  |  |  |
| --- | --- | --- | --- |
| 날짜 | 박주용 | 이예준 | 이수민(팀장) |
| 11/2 이전 | 계획서 작성 | | |
| 11/5 | 전체적인 클라이언트 구조 파악 |  | 클라이언트 Network클래스 생성자 소멸자 구현 |
| 11/6 | 전체적인 클라이언트 구조 파악 |  | Network클래스의 ConnectServer, C\_Recv, C\_Send,ProcessPacket구현 |
| 11/7 | 클라이언트 구조 파악,  이동 관련 함수(move(), playersetting(), playerwaiting())집중 파악 | 전체적인 클라 구조 파악, | 서버의 CLIENT클래스 틀 구현 및 do\_send, do\_recv, processpacket 함수 구현 |
| 11/8 | 서버 프레임 워크 파악, 회의 | UI관련 구조체, 클래스, 동작원리 파악  회의 | 회의 |
| 11/9 | 타 과목 시험준비 | UI관련 구조체, 클래스, 동작원리 파악 |  |
| 11/10 | UI함수 파악 및 화면에 text 띄우기 test(클라이언트 addText 함수 이용) | 게임 종료 UI 리소스 찾고 Final Stage 화면에 띄워보기 |  |
| 11/11 | 클라이언트에서 타이머 구현 후 화면 전환되는지 test |  | 클라이언트 및 서버프레임워크에서 부족한 기능 피드백 이후 추가구현 |
| 11/12 | +서버에 플레이어 이동 로직 연동(cs\_packet\_move, sc\_packet\_move\_process 패킷으로 통신, move(), playersetting(), playerwaiting() 서버로 이동) | 화면에 UI 띄우고 버튼 기능 넣기 | 서버 멀티스레드 함수 GameLogicThread, PlayerInputThread 생성 및 클라이언트와 연동하여 플레이어와 서버간 네트워크 연결 확인 |
| 11/13 | 서버에 플레이어 이동 로직 연동(cs\_packet\_move, sc\_packet\_move\_process 패킷으로 통신, move(), playersetting(), playerwaiting() 서버로 이동) | 버튼 기능 넣고 버튼 누르면 클라이언트 종료 기능: main.cpp->CALLBACK WndProc |  |
| 11/14 | 서버에 플레이어 이동 로직 연동(cs\_packet\_move, sc\_packet\_move\_process 패킷으로 통신, move(), playersetting(), playerwaiting() 서버로 이동) | 클라에서 왼쪽 상단에 플레이어 랭킹을 나타내는 UI 추가 | Adjustplayer, collp2o로직 서버로 옮기기. |
| 11/15 | ~~서버에 플레이어 이동 로직 연동(cs\_packet\_move, sc\_packet\_move\_process 패킷으로 통신, move(), playersetting(), playerwaiting() 서버로 이동)~~  sc\_packet\_moveprocess패킷으로 서버에게 정보를 받은 후 클라이언트에 캐릭터 draw함수에 적용  + 애니메이션도 적용 | Test로 stageNum 의 값을 변수로 저장해 출력해본다 | Adjustplayer, collp2o로직 서버로 옮기기 및 1인 플레이어 대상으로 테스트  서버에서의 클라 관리를 클래스 상속관계로 수정 AdjustPlayer, Collp2o로직을 옮기기위해 Object와 Object를 관리하는 Map을 서버로 이동. 상속으로 관리하는 클래스들간의 정보공유 (SceneChanger) 초석 코딩 |
| 11/16 | ~~Sc\_packet\_moveprocess패킷으로 서버에게 정보를 받은 후 클라이언트에 캐릭터 draw함수에 적용~~  서버에서 deltatime을 이용해서 로비 카운트다운 구현 test 및 robby 패킷으로 클라이언트와 통신 확인 | 클라이언트 UI의 버튼구조 수정 ( UI 부모클래스 관리 구조 -> 자식클래스 각자 관리 구조로 변경) | Adjustplayer, collp2o로직 서버로 옮기기. 및 다중 플레이어 대상으로 테스트 |
| 11/17 | 서버에서 deltatime을 이용해서 로비 카운트다운 구현(sc\_packet\_robby로 서버에서 1초에 한 번씩 클라이언트에 카운트다운 전송) |  | SceneChanger 완성 ChangeLoginToRobby(), ChangeRobbyToGame(), 이를위한 이벤트 동기화 설계 |
| 11/18 | ~~sc\_packet\_moveprocess패킷으로 서버에게 정보를 받은 후 클라이언트 draw함수에 적용~~  로비에서 10초가 지나면 인 게임 화면으로 화면 전환하기 | ~~순위 체크 구현(텍스트로): y값 비교는 나중에 하고 먼저 Stage 번호가 클수록 순위가 바뀌는 것 구현.adjustPlayer에서 201(포탈번호) 가 불리면 순위변수의 값 ++~~ | ~~Spike\_hurttime 로직 서버로 옮기기 및 정상작동 테스트~~ |
| 11/19 | ~~sc\_packet\_moveprocess패킷으로 서버에게 정보를 받은 후 클라이언트 draw함수에 적용~~  다중접속에 따른 애니메이션 및 로비에서의 화면전환 수정,  로비에서 10초가 지나면 인게임으로 플레이어들 동시 입장구현,  서버에서 n명 클라이언트 들어왔는지 판단 | ~~순위 체크 구현(텍스트로): y값 비교는 나중에 하고 먼저 Stage 번호가 클수록 순위가 바뀌는 것 구현.adjustPlayer에서 201(포탈번호) 가 불리면 순위변수의 값 ++~~  테스트를 위한 치트키 동작 서버와 연동 | ~~Stealthtime() 로직 서버로 옮기기 및 정상작동 테스트~~  win32api의 keyinput 고질적인 문제 수정 -> keyinput시 패킷보내는게 아니라, keyboard변수를 만들어, 거기에 변화를 저장. 이후 변수를 보고 update에서 패킷 송신, SC\_PACKET\_PUT\_OBJECT패킷을 통한 다중접속 이후 서로 보이게끔 Player others로 관리. |
| 11/20 |  | ~~순위 체크한 것을 sc\_packet\_stagenum 로 클라에게 send~~  서버에서 클라이언트들의 좌표를 비교해 순위를 계산하는  rank로직 구현 | ~~Hurt()로직 서버로 옮기기 및 정상작동 테스트~~  포탈상호작용 로직 서버와 연동  Stealthtime(), spike\_hurttime(),  Hurt()로직 서버와 연동 |
| 11/21 | ~~서버에서 deltatime 이용해서 타이머 구현 test 및 robby패킷으로 클라와 통신 확인~~  팀원들 작업에 맞춰 패킷, 애니메이션 등 기존의 오류 수정 및 클라이언트(로비) UI 수정 | 서버에서 클라이언트들의 좌표를 비교해 순위를 계산하는  rank로직 구현 및 랭킹을 movepacket에 넣어 클라이언트로 send후 확인 |  |
| 11/22 | ~~서버에서 deltatime 이용해서 타이머 구현 test~~ |  | ~~공유자원으로 인한 임계영역 사용 최소화작업~~  Object 클래스 구조를 Obstacle 클래스의 상속을 받아 MoveObstacle, AttackObstacle,  PlatformObstacle로 변경 이후 오브젝트 위치, 공격타이밍을 클라이언트에 동기화 |
| 11/23 | ~~서버에서 deltatime 이용해서 타이머 구현 test~~  로비 timer동기화 버그(카운트다운이 중복돼서 보이는 버그)수정,  로그아웃 시 상대방 화면에서 해당 로그아웃 된 플레이어 사라지는 것 구현 및 수정,  1차 remote 테스트 | ~~서버로 옮겨진 플레이어 이동 로직을 통해~~  ~~stageNum이 같을 경우는 플레이어들의 y값을 비교하여 1,2,3(순위)으로 변수를 설정한 후 클라이언트에 send~~  StageNum이 다를 경우 순위를 비교하여 rank를 정한다. | ~~공유자원으로 인한 임계영역 사용 최소화작업~~  동기화된 정보를 바탕으로 Player Hurt관련 함수들 동작하는지 확인,  1차 remote 테스트 |
| 11/24 | ~~서버에서 3명 클라이언트 들어왔는지 판단~~  로비에서 시간이 지날수록 프레임 드랍이 되는 오류 발견-> 수정 | 위와같음 |  |
| 11/25 | ~~서버에서 클라이언트로 카운트다운 통신 robby 패킷)~~  ~~로비 timer동기화 버그수정~~  구조가 바뀐 뒤  로비에서 인게임으로 넘어가면 플레이어가 움직이지 않는 오류, ChangeRobbyToGame함수가 플레이어 수\*3번 불리는 오류, 로비에서 인게임으로 넘어갈 때 플레이어들의 상태 값(mCss, mSn)이 바뀌지 않는 오류 발견->수정,  2차 remote 테스트 및 버그 분석 | ~~위와같음~~  서버로부터 받은 순위판정 값을 플레이어 이름과 함께 UI로 띄워줌,  2차 remote 테스트 및 버그 분석 | 2차 remote 테스트 및 버그 분석 |
| 11/26 | ~~서버에서 클라이언트로 카운트다운 통신 (robby 패킷)~~  플레이어의 state가 순간적으로 바뀔 때(점프, 줄 이동 시) 잠깐 깜빡하는 버그 일부 수정 | 서버로부터 받은 순위판정 값을 플레이어 이름과 함께 UI로 띄워줌 |  |
| 11/27 | ~~서버에서 클라이언트로 카운트다운 통신 (robby 패킷)~~  기존에 구현했던 클라이언트에서의 로비 UI 구현 방법을 수정하여 프레임 드랍 없이 최적화 | 서버로부터 받은 순위판정 값을 플레이어 이름과 함께 UI로 띄워줌 | 클라이언트 ConnectServer함수 전달인자를 사용자 입력으로 받게끔 수정 |
| 11/28 | 이전 수정 작업(11/27)에서 빼먹은 부분 추가 수정 | ~~서버로부터 받은 순위판정 값을 플레이어 이름과 함께 UI로 띄워줌~~ |  |
| 11/29 | 코드 정리 | 팀장과 코드리뷰 및 추가작업 |  |
| 11/30 | 코드 정리 | 팀장과 코드리뷰 및 추가작업 | 코드 정리 |
| 12/1 | 추가 구현 or 정리 |  | 코드 정리 |
| 12/2 |  |  | 코드 정리 |
| 12/3 | 추가 구현 or 정리 | 최종 정리 및 추가구현 | 최종 정리 및 추가구현 |
| 12/4 | 추가 구현 or 정리 | 최종 정리 및 추가구현 | 최종 정리 및 추가구현 |
| 12/5 | 최종 정리 | 최종 정리 및 추가구현 | 최종 정리 및 추가구현 |
| 12/6 |  | 최종 정리 및 추가구현 | 최종 정리 및 추가구현 |
| 12/7 | 최종 정리 |  | 최종 정리 및 추가구현 |
| 12/8 | 최종 마감일 | | |

|  |  |
| --- | --- |
| 박주용 | |
| 11/2 이전 | 계획서 작성 |
| 11/3 |  |
| 11/4 |  |
| 11/5 |  |
| 11/6 |  |
| 11/7 |  |
| 11/8 |  |
| 11/9 |  |
| 11/10 |  |
| 11/11 |  |
| 11/12 |  |
| 11/13 |  |
| 11/14 |  |
| 11/15 |  |
| 11/16 |  |
| 11/17 |  |
| 11/18 |  |
| 11/19 |  |
| 11/20 |  |
| 11/21 |  |
| 11/22 |  |
| 11/23 |  |
| 11/24 |  |
| 11/25 |  |
| 11/26 |  |
| 11/27 |  |
| 11/28 |  |
| 11/29 |  |
| 11/30 |  |
| 12/1 |  |
| 12/2 |  |
| 12/3 |  |
| 12/4 |  |

|  |  |
| --- | --- |
| 이예준 | |
| 11/2 이전 | 계획서 작성 |
| 11/3 |  |
| 11/4 |  |
| 11/5 |  |
| 11/6 |  |
| 11/7 | 팀장이 구현해 놓은 클라이언트 코드의 전체적인 구조 파악 |
| 11/8 | 내가 해야 할 ui 작업을 위해 클라에서 ui 관련 함수와 로직, 동작원리 파악 |
| 11/9 | <종료버튼 ui>  종료버튼 리소스 제작.  종료버튼을 화면에 띄워본다  =>이미지가 잘 출력되는지 보기 위해 일단은 로비에 버튼을 띄워봄 |
| 11/10 |  |
| 11/11 |  |
| 11/12 |  |
| 11/13 | <종료버튼 ui>  본격적으로 종료 버튼 만들기 시작.  addButton 함수를 이용해 버튼을 누르면 버튼이 잘 동작함  맵의 final stage 에만 종료 버튼을 띄워야 하나 캐릭터의 map number 가 13(final stage) 일때만 종료 버튼이 뜨게 하는 것이 잘 되지 않음 |
| 11/14 | 현재 클라이언트에 있는 화면에 ui 를 그려주는 draw()함수를 이용해서는 마지막 스테이지에만 종료 버튼을 띄우는 작업이 안 될것이라고 생각해 UI클래스에 drawExit() 함수를 따로 제작하여 UI클래스를 상속받는 GameHUD 클래스에서 drawExit함수를 불러 종료버튼을 띄움  ->문제점 발생: 마지막 스테이지에 종료 버튼이 뜨기는 하나 내가 입력한 좌표값과 버튼의 위치가 다르게 나옴 |
| 11/15 |  |
| 11/16 | <종료버튼 ui>  14일에 생긴 문제점을 고침  ->맵의 고정된 위치에 버튼을 생성하는 것이 아닌 카메라를 기준으로 버튼 생성  종료버튼 완성. |
| 11/17 |  |
| 11/18 |  |
| 11/19 | 서버에서 캐릭터가 속해있는 스테이지의 넘버를 클라이언트에게 넘겨주는 gamestagenum 패킷 생성 |
| 11/20 | <치트키 만들기>  게임 테스트를 위해 end 키를 누르면 다음 맵으로 넘어가는 포탈이 있는 곳으로 바로 이동할 수 있는 치트키를 만듬.  순위 작업 시작  ->move\_process\_packet 에 랭크에 대한 변수 추가.  GameClient의 move 함수 안에서 클라이언트의 순위를 결정하는 알고리즘 짬. 콘솔창으로 서버 내에서 클라이언트들의 순위가 바뀌는 동작 확인. |
| 11/21 | 서버에서 판단한 클라이언트들의 랭킹 값을 move\_process\_packet을 통해 클라에게 보냄. 클라에서는 본인의 순위를 콘솔창을 통해 확인 가능 |
| 11/22 |  |
| 11/23 | <순위 기능>  스테이지 넘버가 상대방 보다 크면 상대보다 y 좌표가 작더라도 순위는 높아야 한다. 현재 구현한 순위 알고리즘은 랭킹의 초기값이 1로서 나의 y좌표가 다른 플레이어의 y좌표보다 높으면 다른 플레이어의 랭킹이 +1 되는 식이다. 그러나 다음 맵으로 넘어가면 순위가 1로 초기화가 된다. 그래서 1등이면 랭킹이 3, 2등이면2, 3등이면 1 이런식으로 하고 다음 스테이지로 넘어가면 해당 스테이지 에서 랭킹의 초기값이 1이 아니라 1에서 2를 더한 값(3) 으로 하여 임의로 해당 맵에 대한 초기값을 정해 내가 다음 맵으로 넘어가 랭킹이 초기화가 되더라도 초기값이 3이기 때문에 이전 맵에 있는 플레이어들 보다 랭킹 값이 커서 랭킹이 높은 것을 나타냄  ->문제점: 다른 스테이지에서 y좌표가 서로 같아지는 순간 순위도 같아짐 |
| 11/24 | <순위 기능>  클라이언트 3명 모두가 스테이지 1, 2, 3 있을 때는 스테이지와 y값에 따라 순위가 결정된다 그러나 한명이 마지막 스테이지(게임 완료 스테이지)로 이동을 하면 1등은 게임을 클리어해 사라진 것이기 때문에 스테이지에 아직 남아 있는 플레이어 중 2등은 1등으로, 3등은 2등으로 된다. 모든 플레이어가 게임종료 스테이지에 모이게 되면 그때 다시 순위가 알맞게 매겨진다. |
| 11/25 |  |
| 11/26 | <순위 기능>  클라이언트에서 플레이어(나) 의 이름과 서버로부터 넘겨 받은 순위를 ui로 띄움  원래는 서버에서 위치를 비교해 1등의 숫자를 크게 나오게 하여 이 숫자를 클라로 보내면 클라에서는 숫자가 큰 순서대로 UI상 1등,2등,3등이 나오게 하려고 했는데 이 변환하는 작업이 생각보다 잘 안되고 비효율적인 것 같아 처음부터 서버에서 순위대로 숫자를 정해 클라로 보내주는 것이 낫다고 판단하여 순위 알고리즘을 수정.  문제점: 만약 1등이 다음맵으로 혼자 넘어간 경우, 1등 클라의 ui에는 1,2,3등이 잘 나온다. 그러나, 2,3등의 경우 둘의 순위만 비교해 2등이 1등이 되고 2등이 3등이 되는 현상이 나타남. 그러나 1등이 있는 맵으로 넘어갈 경우 다시 순위가 제대로 매겨짐. |
| 11/27 | <순위 기능>  내 이름과 순위, 상대방의 순위와 이름을 화면에 띄움  문제점1: 상대방의 이름, 순위의 배치 좌표가 클라이언트마다 다르다.  문제점2: 26일의 문제점과 같음 |
| 11/28 |  |
| 11/29 |  |
| 11/30 | <순위 기능>  이제까지는 main에서 send\_move\_process를 통해 rank 값을 클라에게 보내줬다. 그러나 계속 26일의 문제점이 해결되지 않아 GameClient의 move 함수에서 순위 계산을 하면 계산 이후에 패킷을 보내는 방식으로 수정했다. 그랬더니 문제가 해결됨  팀장에게 코드를 보여줌. Move 함수 안에 순위 로직을 짜 놓으면 send\_move\_process에서 다 send 를 해주기 때문에 move함수 안에 또 패킷을 보내는 코드를 짤 필요가 없었다. 따라서 move 함수 내에서 짠 패킷 보내는 코드 전부 수정 -> 수정 해도 제대로 잘 동작함  기존에는 순위가 변경되면 숫자가 변경되는 식이었으나 팀장의 요청으로 순위가 바뀌면 닉네임의 배치가 바뀌는 식으로 변경. |
| 12/1 | <순위 기능>  Final stage 넘어온 클라이언트 순서대로 랭킹이 정해짐. 그리고 final stage의 화면에 자신의 랭킹을 출력.  불필요한 코드들 지우고 ui 를 좀더 꾸미고 최종 마무리 작업 |
| 12/2 |  |
| 12/3 |  |
| 12/4 |  |

|  |  |
| --- | --- |
| 이수민 | |
| 11/2 이전 | 10.3 클라 소리조정.  10.3 로그인화면 추가.  10.4 닉네임 입력기능 추가  10.4 닉네임 플레이어 아래에 뜨게 추가.  10.31 Ui 구조 변경 - Ui클래스를 상속받는 각각의 HUD로 구조 변경,  Ui위에 그려주기 위해 필요한 Text클래스, Button클래스 생성  10.31 기존의 메시지큐에서 타이머를 통해 윈도우프로시저 콜백으로 render처리하던것을  PeekMessage함수를 통해서 콜백이 안불러질때 render 하는 방식으로 구조 변경  10.31 Screen고정버튼 방식에서 UserByButton방식 추가. ui를 넣는 구조에서 frame마다 넣어서 벡터에 매우많이 ui가 쌓여서 프레임드랍이 일어나는 버그 수정.  11.1 한글 입력하는 방식 추가.  11.1 Text클래스에서 한글을 띄워주기 위해 wstring간 변환방식 추가. |
| 11/3 |  |
| 11/4 |  |
| 11/5 | 11.5 넷겜플 팀플 개발일정 작성. |
| 11/6 | 서버,클라 기초적인 네트워크 프레임워크 작성. 기본 클라이언트에서 사용하던 Player.cpp,h  프로토콜 기획제안서에 맞춰 입력,  클라이언트 접속시도마다 스레드 하나씩 할당해주는것 테스트. |
| 11/7 | 본격적인 네트워크 프레임워크 작성.  클라에서 Network.cpp를 통해 recv, send를 할 수 있도록 추가.  서버에서 각 Player마다 recv,send를 해야하므로 Player에 recv,send를 하는 기능 추가. |
| 11/8 | 서버 main.cpp의 클라이언트의 id를 받아주는곳 추가.  서버 프레임워크의 Player를 상속화로 변경 Player->Client 명칭 변경  Client부모 클래스를 상속받는 LoginClient, LobbyClient, GameClient 3가지방식으로 변경 Client에는 소켓관련 데이터들이 존재하고, 각 Login,Lobby,Game은 그 자리에서 가져야하는 데이터, Login (x) Lobby(Player x,y 데이터), Game(player x,y,hp,savey등등)을 가짐.  Client를 관리하는 배열 컨테이너가 존재하고, 포인터형으로 자식 컨테이너들을 할당받아서 Client컨테이너에 넣어 관리하는 구조.  그리고 Login -> Lobby로 가야할때는 Lobby를 새롭게 new 할당하고, Login을 Lobby클래스로 강제형변환 한다음 넣어준다. 그럼 부모클래스는 맞춰지지만, 각 클래스마다 가지고있는 변수들은 이상한 값으로 채워지게되는데, 이건 다시 초기화를 해줌으로써 SceneChange를 해준다.  클라이언트 나중에 ip주소를 사용자입력으로 받을 것에 대비하여 ConnectServer에 server\_ip 주소를 받도록 함수 변경. 클라이언트가 blocking 소켓이라 서버에서 send를 안해주면 클라가 멈춰버리기 때문에 임시용으로 empty패킷을 만들어서  클라가 원활하게 작동될 수 있도록했다. |
| 11/9 |  |
| 11/10 |  |
| 11/11 |  |
| 11/12 |  |
| 11/13 |  |
| 11/14 |  |
| 11/15 | 11.15 AdjustPlayer, Collp2o를 옮기기 위해 사전작업이 필요하다는것을 인지했다.  -> Object가 없어서 Player와 충돌체크를 할 수 조차 없음.  그래서 Object를 클라에서 서버로 옮겨주었다. 그랬더니 또다른 문제가 발생.  Object는 옮겨왔는데 Objects를 관리하는 곳이 없다.  그래서 Map클래스도 새롭게 만들었다. Map클래스는 Object를 관리하는 벡터가 존재한다. 클라에있는 map.txt들을 가져와서 불러와 읽은다음 Object를 new로 할당받아 Map클래스에 Object포인터를 보관하는식으로 코딩.  Object는 나중에 Object를 상속받는 상속구조를 만들기 위해서 클래스로 제작.  그리고 Network를 어디에서든 참조할수 있게 싱글톤 디자인패턴을 가져다가 썼는데, 소멸자에서 전역변수를 한번 더 해제하는바람에 소멸자에서 무한루프를 도는 버그가 발생했고, 이 버그를 해결.  또한 기존의 SceneChange 방식은. Client 부모클래스부분말고 다른부분도 메모리를 침범하는것이 문제였는데, 단순 int,char변수들만 있을때는 문제가 되지 않았으나, 포인터형식과 vector가 멤버변수로 들어가게되면서 잘못된 메모리 쓰기 버그가 일어나는것을 확인.  그래서 SceneChange방식을 변경  바뀐 SceneChange방식 -> login과 lobby둘다 client라는 클래스를 상속받고있으므로 client로의 업캐스팅(적법) 그리고 업캐스팅된 것에서의 값 복사 -> 둘다 client로 업캐스팅됐으므로 client만큼만 메모리복사.  이후 새롭게 할당된것의 멤버변수들은 따로 초기화 해주고 컨테이너에 원래있던것은 delete해주고, 새롭게 만든걸 CLIENTS 컨테이너에 넣어줌으로써 SceneChange 구조를 변경.  이 구조변경이 끝난 후 AdjustPlayer를 테스트하기위해 Move를 간단한 로직을 넣어 테스트. 잘 되는것을 확인. |
| 11/16 |  |
| 11/17 | 기존의 SceneChanger 방식을 또한번 변경  기존 -> 3초뒤에 Login->Lobby로 변하도록(임시)  변경 -> 클라에서 Login 버튼을 눌러 Login패킷을 보내면 SceneChange  Login패킷이 왔을때 Scene을 변경하기위해 mCss(ClientSceneState를 Dead로설정)->씬을 변경할때가 되었다는 뜻, mSn(SceneName) 어떤 씬으로 변경해야한다는 뜻. 을 추가. 씬 변경은 InputThread에서 받고, 씬 변경은 LogicThread에서 하는데, 이후 씬 변경할동안 다른 데이터를 받으면 문제가 생길것같아서 이벤트를 만들고 (SceneChangeTrigger, SceneChangeIsDone)  그리고 mCss와 mSn을 변경하는데있어서 두개의 변경이 함께 적용되어야하는데, 두개를 임계영역으로 감싸고 logic에서 항상 임계영역을 검사하면 오버헤드가 클것같아서 mCss는 한번 그냥 읽고 바꿔줘야하면  WaitForSingleObject로 멈춰준다. (inputThread에서 mCss를 쓰고 mSn을 쓰기 전에 로직이 mSn을 읽으려고해도 멈춰주기때문에 오류 방지 LogicThread에서 SceneChange가 끝나면 SceneChangeIsDone을 켜주어, inputThread도 무사하게 나가도록 설정. |
| 11/18 |  |
| 11/19 | - 기존 윈플코드의 key input은 프로시저에서 해주는데 키보드를 꾹 누를 경우.  콜백프로시저에서 키보드를 처리하는곳에서 약간의 딜레이가 생긴다.(처음에 한번 들어오고 잠시뒤에 계속 들어오게된다.)  이를 해결하기위해 Keyboard라는 배열을 만든다음. KeyDown과 Keyup에서는 배열의 값을 0,1로 설정만 해주고, 실제 KeyInput으로 인한 동작은 GameLogic에서 Keyboard배열에서 1인지 0인지 보고 작업하는 방식으로 변경하여 딜레이를 없앴다.  기존의 empty패킷을 주고받는 방식(임시) 에서 유의미한 데이터를 가지는 move\_process패킷을 주고받는 방식으로 변경.  클라->서버로 로직을 옮기면서 생겼던 버그들 수정 ROWSPEED = 3 -> 1을 3분의 1로 이해하여 ROWSPEED = ROWSPEED/ 3 으로 서버로 옮겨왔는데 매 프레임마다 3분의 1로 나눠버려서 움직임이 멈추는 버그 발생. 이를 수정 ROWSPEED = 50 고정값으로 바꿔주어 오류를 수정했다.  클라이언트에서 다른 플레이어를 보여주기 위한 작업을 시작.  클라이언트는 others라는 player객체를 가지고있다.  그래서 서버가 부여해주는 id값으로 others를 구분가능하게 했다.  서버에서 player들의 update가 끝난 다음 모든플레이어에게 바뀐 상태를 한번에 업데이트하도록 해야하므로 client멤버함수 안에서 해주던것을 밖으로 빼주었다.  또한 그동안 Recv의 치명적인 버그가 있었는데 여태 무시되다가 패킷이 쌓이기 시작하면서 버그가 발생 unsigned char 자료형인 buf를 포인터형식으로 저장해버려서 1바이트->4바이트가 되어버리는 (32비트기준) 치명적버그가 발생.  또한 recv이후 읽는것 buf의 첫 부분을 읽어야하는데 버퍼의 중간을 읽도록 하여 이상하게 읽어버리는 버그 또한 존재. 이러한부분을 모두 수정. |
| 11/20 | 포탈 상호작용 구현  - 포탈패킷, logout\_object패킷 활성화  - 포탈패킷은 클라 맵 가려주는 용도이며,  - logout으로 서버에서 클라들간의 상태를 바꿔주는 역할을 한다.  ---클라---  - 기존의 move/AdjustPlayer를 서버 로직으로 빼줬기때문에 이제 클라에선 사용하지 않으므로 주석처리  - portal패킷, logout\_object 패킷 설정. sethp(5) 같은 경우는 나중에 hp관련 처리 하면서 꼭 수정하도록 하자. 임의로 넣어둔 값.  ---서버---  - 클라이언트를 전역으로 관리하는걸 Lobby에서 Network으로 이동 (구조상 Network가 좀 더 나아보임)  - Cnt\_Player 또한 같은 이유로 Network으로 빼줌  - hurt 함수 활성화 But hp변화량은 현재 패킷으로 관리가 안되기때문에 클라에서 hp가 감소되어보이진 않음. 서버에서는 정상작동되는걸 확인.  - 또한 서버에서 update 이후 do\_send부분에서 같은 stage에 있지 않다면, 패킷보내는걸 스킵함. 패킷이 많이 보내지는것을 방지.  StealthTime(), Spike\_Hurttime()로직 클라에서 서버로 이동.  하지만 Object들의 이동처리,인덱스처리가 아직 진행되지않아 확인불가.  컨베이어벨트는 확인가능 <- 잘 작동 된다. |
| 11/21 |  |
| 11/22 | Player제외 Object들을 Object 상속을 받아 좀더 세밀하게 구성, Obstacle과 Interact로 분류했으며 Obstacle은 Attack,Move,Platform으로 나뉘며 Interact는 Protal과 Rope로 분류. Interact같은 경우는 Object와 동일하다고 볼 수 있지만 후의 보수작업을 생각하여 미리 분류.  - 이후 서버에서 바뀐 좌표와 index를 클라에 동기화. 이를위해 object\_sync 패킷을 추가. mx,my는 gear를 위해 쓰이고, index는 steam을 위해 사용된다. gear와 steam 패킷을 따로 분류하지않고, 한번에 묶어서 사용하도록 했다.  ---서버---  - Attack과 Move Obstacle은 update내에서 자체적으로 자신만의 로직 수행.  - Main에서 Map의 모든 Object들을 Update해준다.  - 동기화를 위해서 GameClient Update에서 매 프레임마다 바뀐 Object들의 정보들을 클라에게 패킷전송  - robby에는 장애물이 없으므로, 쓸대없는 adjustplayer의 로직 삭제. collp2o 또한 마찬가지 이유로 로직들 삭제.  --- 클라 ---  - 서버에서 보낸 패킷들을 받아서, object들의 정보를 업데이트 한다.  - 기존의 main에서 object들을 update해주던 로직을 주석처리. |
| 11/23 | 기존의 방식의 치명적인 버그가 존재하여 수정.  - 기존 로직은 Accept를 받으면 is\_active를 true로 바꿔주었는데 이 것 때문에 두번째 클라를 켜고, 첫번째 클라가 로그인을 누르기전에 두번째 클라가 먼저 들어가면 SceneChanger에서 첫번째클라가 is\_active = true 상태라 putObject가 된다. 막아주기위해 로그인 버튼을 눌렀을 때 비로소 active가 true가 되도록 바꿔주었다. |
| 11/24 |  |
| 11/25 | 서버에서 클라로 보내주는 패킷수를 60 -> 30으로 줄이기위한 작업 시작  30으로 그냥 줄여버리니 클라가 30프레임에 맞춰 멈춰버리는 (blocking 소켓) 현상 발생.  그래서 클라이언트를 멀티스레드로 분리  멈춰버리는 현상은 사라졌으나, 움직임이 30프레임으로 와서 뚝뚝 끊겨보임  이것을 해결하기위해 데드레커닝방식을 사용.  등가속도법칙을 사용하여 다음프레임의 위치를 보간하는 방식  S= V0+1/2at^2 을 사용하고 , a는 게임로직내에서 정했으므로 상수, t는 deltatime이고 모르는것이 V0인데,  V0는 거리 = 시간\*속도 공식을 사용하여 속도 = 거리/시간 을통해 구할 수 있다.  거리는 OldX와 새롭게 패킷으로 받은 X를 비교하여 거리를 구할 수 있고,  시간은 ServerSendTime패킷을 추가하여 여기에 서버가 한 프레임에 걸린 시간을 보내서 얻을 수 있다. 이렇게 하여 인자들을 모두 구하고, S를 구해서  패킷이 오기전에 한번 더 logic이 실행되면 S를 더하여 자연스럽게 움직이도록 수정해주었다.  ingame에서 한글 입력시 튕기는 현상 수정  맵 이동시, logoutObject패킷을 다른사람에게 안보내줘서 다른사람한테 내가 계속 남아있는 버그 수정. |
| 11/26 |  |
| 11/27 | id or ip를 다 지운다음 한글을 입력하려고 할 때 changewChar에서 mText[-1]을 접근하여 발생하는 문제 해결  UI.cpp에서 mUiCount로 예외사항 debug log가 정상작동이 안되는것을 확인하고 이를 수정.  - 기존 id와 password를 입력하는 구조에서id와 ip를 입력하는 구조로 기획 변경--> ip를 입력하여 로그인하는 방식으로 변경 -> 사용자입력으로 원하는 ip에 접속하도록 변경 (여기서부터 LoginClient가 쓸모가없어짐 바로 LobbyClient로 이동)  - 로그인실패시 서버연결실패 ui를 띄워줌. 이때 ui가 띄워지면 로그인버튼이 다시 눌리면 안되므로 LoginHUD에 추가하는것이 아닌 OkHUD를 mUI에 push\_back하는 구조로 생성.  - 새로운 ui를 위한 bmp 이미지 구해오기. connectServer의 return값을 바꿔주어 실패했음을 알 수 있도록 변경.  - ip입력란을 최대 16글자로 변경. 기존 main의 LButtonDown부분을 LoginHUD의 ProcessInput으로 이동.  dieHUD를 버튼하나로 퉁쳤었는데 이를 bitmap과 button으로 분리.  서버에서 hurt시 state만 바꿔주고 다음프레임의 move에서 stealth time을 정해주었는데, 다음 프레임에 들어오기 전에 inputThread에서 state를 바꿔주어 생기는 버그를 수정. hurt시에 바로 stealthtime을 정해주어 state와 상관없게 해줌. |
| 11/28 |  |
| 11/29 | 데드레커닝 방식 제거, Nagle알고리즘을 Off하고 30프레임으로 보내도 여전히 끊기는 문제 잔재. 30프레임 ->60프레임으로 바꿔줌. ->해결  -구조의 치명적인 문제 발견  Login은 클라에서 요청을 보내고, 받으면 inputthread는 멈추고, 로직을수행하고 delete를 해주는데, LobbyToGame은 서버에서 일방적으로 보내는구조라  카운트다운이 끝나고 inputThread가 recv를 받지못하게 통제를 못해서 delete된 포인터가 recv를 걸어두고 삭제됨.  - 이후 recv가 되면 delete된 주소에서 recv 이후의 코드들이 수행됨. 이는 패킷의 유실과도 문제가 되지만, 왠진모르게 CLIENTS컨테이너의 다음 주소에 있는 소켓이 받아온것처럼 말도안되는 버그가 발생. 그런데 이게 딱 한번만 버그가 생기는것이라 클라에서 서버로 send할때 자신의 id를 보내주어 오류처리하는식으로 해결. |
| 11/30 | die HUD안그려지는 버그 수정. |
| 12/1 | 서버에서의 클라들간의 입력 순서 처리.  교수님이 말씀하신 thread당 cpu할당 순서가 랜덤이기때문에 기아현상발생으로 인해 rank의 부정확함을 이벤트방식으로 해결.  0은 1or2를 1은 2or0을 2는 0or 1을 이벤트를 켜주는 방식으로 순서 동기화를 시켜주었다.  그리고 이는 클라에서 send를 안하면 멈춰버리는 불상사가 생기기때문에, 클라이언트에서 키를 아무것도 안누를때는 아무것도 안하는 공패킷을 보내도록 해주었다.  1129에서 구조적으로 애초에 잘못돼서 꼬여버린 코드를 차마 그냥 바라볼수가 없어서 구조를 변경  기존의 client를 상속받는 구조 그리고 scenechange로 컨테이너에서 포인터를 delete하는 구조가 얼마나 위험한지 깨달았고, 왜 로비서버와 인게임서버를 나누는지에 대해서 납득하게 됨.  우리 게임은 lobby와 ingame이 사실상 똑같은 ingame이고,  loginclient는 클라에서 ip입력후 로그인 하는 방식에서 사실상 쓸모가 없어졌으므로  분리되었던것을 하나의 client로 묶어준다음, 포인터를 delete하는 방식을 사용하지 않는 구조로 변경.  client내의 코드에 lobby와 ingame이 합쳐지면서 client코드는 굉장히 길어졌지만  이상한 오류들이 이제 생기지 않고, 이를 해결하기 위해 땜빵용으로 이상한 패킷들을 보내고 이상한 오류검사를 하던 코드들을 다 제거해줌. |
| 12/2 | dieui bitmap 안뜨는 버그, Die했을 때 자동이동버그, Player 비트맵 깜빡임 버그 수정. 두번 죽으면 dieui 안뜨는 버그수정 UIState::Active로 바꿔주는 코드 추가. |
| 12/3 |  |
| 12/4 |  |