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. |
| **sc\_packet\_login\_ok**{  size, type, id, x, y  } | size만큼의 패킷을 받아서 type을 확인하고, 클라이언트의 id와(name아님), 클라이언트(플레이어)의 시작 x, y좌표를 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. |
| **cs\_packet\_move**{  size, type, dir  } | 플레이어가 dir 방향으로 움직이고 싶다는 패킷을 서버가 받는다. |
| **sc\_packet\_move\_process{**  size, type, id, x, y, state, h, dir, stealth, 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 |
| **~~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, count, countdownTime  } | Count: 로비에 플레이어가 몇 명이 모였는지 센다.  countdownTime: 10,9, …, 1 까지 카운트다운 |
| **sc\_packet\_empty**{  size,type  } | 서버에서 처리할 보내줘야 하는 패킷이 없을 때 blocking 소켓으로 구현된 클라이언트가 멈추지 않도록 보내주는 패킷. |
| **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 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

* 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 리소스 제작.  종료버튼을 화면에 띄워본다  =>이미지가 잘 출력되는지 보기 위해 일단은 로비에 버튼을 띄워봄 |
| 11/10 |  |
| 11/11 |  |
| 11/12 |  |
| 11/13 | 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 | 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 | 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 | 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 |  |
| 11/25 |  |
| 11/26 |  |
| 11/27 |  |
| 11/28 |  |
| 11/29 |  |
| 11/30 |  |
| 12/1 |  |
| 12/2 |  |
| 12/3 |  |
| 12/4 |  |

|  |  |
| --- | --- |
| 이수민 | |
| 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 |  |