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  } | 플레이어들이 있는 stage의 값을 넘겨주는 역할을 한다.  id : 클라이언트의 소켓 번호  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. |

* 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 추가 | 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 | 클라 구조 파악, | 서버의 CLIENT클래스 틀 구현 및 do\_send, do\_recv, processpacket 함수 구현 |
| 11/8 | 서버 프레임 워크 파악, 회의 | 클라 구조 파악.  회의 | 회의 |
| 11/9 | 타 과목 시험준비 | 클라 구조 파악 |  |
| 11/10 | UI함수 파악 및 화면에 text 띄우기 test(클라이언트 addText 함수 이용) | UI함수 파악 및 화면에 UI 띄워보기 |  |
| 11/11 | 클라이언트에서 타이머 구현 후 화면 전환되는지 test |  | 클라이언트 및 서버프레임워크에서 부족한 기능 피드백 이후 추가구현 |
| 11/12 | 서버에서 deltatime 이용해서 타이머 구현 test 및 robby패킷으로 클라와 통신 확인 | 화면에 UI 띄우고 버튼 기능 넣기 | 서버 멀티스레드 함수 GameLogicThread, PlayerInputThread 생성 및 클라이언트와 연동하여 플레이어와 서버간 네트워크 연결 확인 |
| 11/13 | 서버에서 deltatime 이용해서 타이머 구현 test | 버튼 기능 넣고 버튼 누르면 클라이언트 종료 기능. |  |
| 11/14 | 서버에서 deltatime 이용해서 타이머 구현 test |  | Adjustplayer, collp2o로직 서버로 옮기기. |
| 11/15 | 서버에서 3명 클라이언트 들어왔는지 판단 | 클라에서 순위 체크 구현(텍스트 형식으로 | Adjustplayer, collp2o로직 서버로 옮기기 및 1인 플레이어 대상으로 테스트 |
| 11/16 | 서버에서 클라이언트로 카운트다운 통신 test(robby 패킷) | 클라에서 순위 체크 구현(텍스트 형식으로 | Adjustplayer, collp2o로직 서버로 옮기기. 및 다중 플레이어 대상으로 테스트 |
| 11/17 | 서버에서 클라이언트로 카운트다운 통신 test(robby 패킷) | 클라에서 순위 체크 구현(텍스트 형식으로 |  |
| 11/18 | 서버에서 클라이언트로 카운트다운 통신 test(robby 패킷) |  | Spike\_hurttime 로직 서버로 옮기기 및 정상작동 테스트 |
| 11/19 |  |  | Stealthtime() 로직 서버로 옮기기 및 정상작동 테스트 |
| 11/20 | 서버에 플레이어 이동 로직 연동(cs\_packet\_move, sc\_packet\_move\_process 패킷으로 통신, move(), playersetting(), playerwaiting() 서버로 이동) | 클라에서 순위 ui 띄우기 | Hurt()로직 서버로 옮기기 및 정상작동 테스트 |
| 11/21 | 서버에 플레이어 이동 로직 연동(cs\_packet\_move, sc\_packet\_move\_process 패킷으로 통신, move(), playersetting(), playerwaiting() 서버로 이동) | 클라에서 순위 ui 띄우기 |  |
| 11/22 | 서버에 플레이어 이동 로직 연동(cs\_packet\_move, sc\_packet\_move\_process 패킷으로 통신, move(), playersetting(), playerwaiting() 서버로 이동) |  | 공유자원으로 인한 임계영역 사용 최소화작업 |
| 11/23 |  |  | 공유자원으로 인한 임계영역 사용 최소화작업 |
| 11/24 | sc\_packet\_move패킷으로 서버에 정보를 받은 후 클라이언트 draw함수에 적용 | 서버에서 순위 체크 구현 |  |
| 11/25 |  | 서버에서 순위 체크 구현 |  |
| 11/26 | sc\_packet\_move패킷으로 서버에 정보를 받은 후 클라이언트 draw함수에 적용 | 서버에서 순위 체크 구현 |  |
| 11/27 | sc\_packet\_move패킷으로 서버에 정보를 받은 후 클라이언트 draw함수에 적용 |  | 클라이언트 ConnectServer함수 전달인자를 사용자 입력으로 받게끔 수정 |
| 11/28 |  |  |  |
| 11/29 | 코드 정리 | 코드 정리 |  |
| 11/30 | 코드 정리 | 코드 정리 | 코드 정리 |
| 12/1 | 추가 구현 or 정리 | 코드 정리 | 코드 정리 |
| 12/2 |  | 최종 정리 및 추가구현 | 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 | | |