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 목적지까지 이동하는 게임
4. 조작 방법: 키보드 4방향키, 스페이스바(점프)

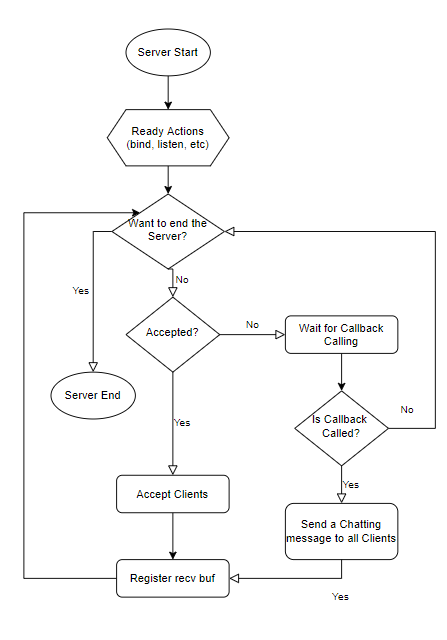
* 게임흐름

|  |  |  |
| --- | --- | --- |
| 화면 | 설명 | 게임 플레이 화면 |
| 로그인 | 유저가 게임 플레이 전에 회원가입과 로그인을 할 수 있는 로그인 화면이 나타난다.  플레이어가 회원가입을 하게 되면 해당 플레이어의 id와 password는 MySQL 서버에 저장이 되고 관리된다. |  |
| 로비 | 유저가 로그인 화면에서 등록한 ID가 플레이어 발 밑에 뜬다.  플레이어 이동이 가능하고 모니터에 마우스를 올리면 조작법이 모니터 화면에 나타난다. |  |
| 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. |  |

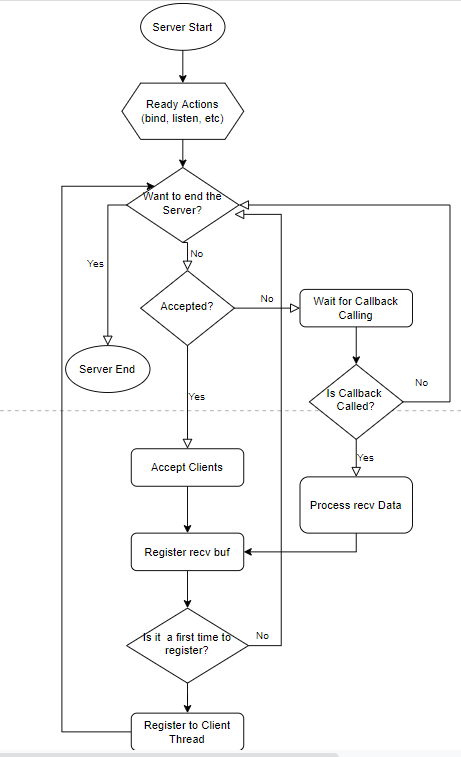
1. High-level 디자인

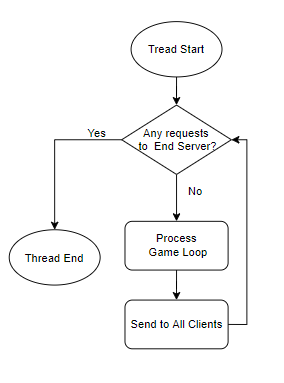
* Flow Chart

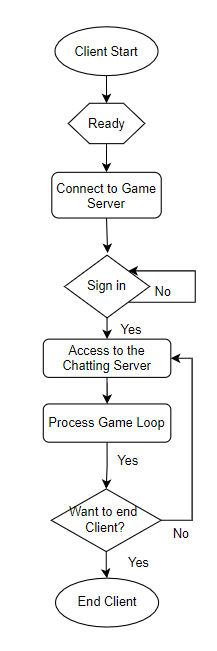
1. 채팅 서버



1. 게임 서버



1. 스레드  
   
2. 클라이언트



* ER Diagram



1. Low-level 디자인

* 애플리케이션 프로토콜(데이터 송수신 패킷)

|  |  |
| --- | --- |
| 패킷 이름 | 설명 |
| **cs\_packet\_login**{  size, type, username[MAX\_NAME\_SIZE], password[MAX\_PW\_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, hp  } | 플레이어의 이동 방향을 가지고 서버에서 이동처리를 해준 다음 그 결과를 클라이언트에게 알려준다. |
| **cs\_packet\_chat**{  size, type, id, msg  } | id번째 클라이언트가 서버로 채팅 패킷을 보냄. |
| **sc\_packet\_chat**{  size, type, id, msg  } | id번째 클라이언트가 서버로 채팅 패킷을 보낸 것을 클라이언트들에게 보냄. |

* 함수 설명(서버)

|  |  |
| --- | --- |
| 함수 이름 | 설명 |
| void CALLBACK **recv\_callback**(DWORD err, DWORD num\_bytes, LPWSAOVERLAPPED recv\_over, DWORD flag) | recv\_over의 h\_event값으로 클라이언트 소켓 id를 얻어온 다음 그 id에 해당되는 클라이언트가 보낸 키 입력으로 플레이어의 state값을 바꿔준다. |
| void CALLBACK **send\_callback**(DWORD err, DWORD num\_bytes, LPWSAOVERLAPPED send\_over, DWORD flag) | send할때 사용했던 자원을 반환한다. (delete) |
| void CLIENT::**do\_send**() | send관련 패킷과 send하기위한 버퍼를 만들고 send한다. |
| void CLIENT::**do\_recv**() | recv를 하고 recv 오류 시 오류 처리를 한다. |
| void **update**() | 플레이어의 이동 처리(CLIENT::update()), 오브젝트의 이동 처리(OBJECT::update()), 플레이어와 오브젝트의 충돌 및 상호작용 처리(adjustPlayer(), collp2o()) 등을 담당하는 함수. 다른 플레이어들에게 바뀐 플레이어 정보 broadcast |
| 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 StageProcessThread(LPVOID arg) | 각 스테이지별로 스레드를 만들어, 스테이지간의 부하를 분산하는 역할을 하는 스레드이다. |

* 함수 설명(채팅 서버)

|  |  |
| --- | --- |
| 함수 이름 | 설명 |
| void CALLBACK **recv\_callback**(DWORD err, DWORD num\_bytes, LPWSAOVERLAPPED recv\_over, DWORD flag) | recv\_over의 h\_event값으로 클라이언트 소켓 id를 얻어온 다음 그 id에 해당되는 클라이언트가 보낸 채팅을 분석하여 처리한다 |
| void CALLBACK **send\_callback**(DWORD err, DWORD num\_bytes, LPWSAOVERLAPPED send\_over, DWORD flag) | send할때 사용했던 자원을 반환한다. (delete) |
| void CLIENT::**do\_send**() | send관련 패킷과 send하기위한 버퍼를 만들고 send한다. |
| void CLIENT::**do\_recv**() | recv를 하고 recv 오류 시 오류 처리를 한다. |
| void processChat(const char\*) | 문자열을 받아 전체채팅인지 귓속말인지 분류해 send 할 대상을 정하는 함수 |

* 함수 설명(클라이언트)

|  |  |
| --- | --- |
| 함수 이름 | 설명 |
| void CALLBACK **recv\_callback**(DWORD err, DWORD num\_bytes, LPWSAOVERLAPPED recv\_over, DWORD flag) | 서버가 보낸 플레이어들의 정보를 받아서 처리한다. |
| void CALLBACK **chat\_recv\_callback**(DWORD err, DWORD num\_bytes, LPWSAOVERLAPPED recv\_over, DWORD flag) | 채팅 서버가 보낸 플레이어들의 메시지를 받는다. |
| void CALLBACK **send\_callback**(DWORD err, DWORD num\_bytes, LPWSAOVERLAPPED send\_over, DWORD flag) | send할때 사용했던 자원을 반환한다. (delete) |
| void CALLBACK **chat\_send\_callback**(DWORD err, DWORD num\_bytes, LPWSAOVERLAPPED send\_over, DWORD flag) | send할때 사용했던 자원을 반환한다. (delete) |
| void CLIENT::**do\_send**() | send관련 패킷과 send하기 위한 버퍼를 만들고 send한다. |
| void CLIENT::**do\_recv**() | recv를 하고 recv 오류 시 오류 처리를 한다. |
| void CLIENT::**do\_chat\_send**() | chatting 관련 패킷을 send하기 위한 버퍼를 만들고 send한다. |
| void CLIENT::**do\_chat\_recv**() | recv를 하고 recv 오류 시 오류 처리를 한다. |

1. 팀원 별 역할 분담

|  |  |  |
| --- | --- | --- |
| 박주용 | 이예준 | 이수민 |
| 네트워크 환경 구축 | 네트워크 환경 구축 | 네트워크 환경 구축 |
| 로그인, 회원가입 구현  (MySQL 연동) | 채팅 서버 구현 | 클라이언트 UI(로그인, 채팅) 및 로직 담당 |
| 클라이언트와 서버 간 키 입력(이동) 연동 |  | 충돌 처리 관련 함수 클라이언트와 서버 간 연동 |
| 플레이어 이동 관련 함수 클라이언트와 서버 간 연동 |  |  |

1. 개발 환경

* 운영체제: Window 10
* 컴파일러: Visual Studio 2019
* 데이터 정렬 방식: 서버/클라이언트 모두 LittleEndian
* 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 | | |
| 11/3 | 게임 로직 파악 | 게임 로직 파악 | ui 설계&구조 바꾸기 |
| 11/4 | 군복무로 인한 개인 사정 | 게임 로직 파악 | 채팅창 ui 리소스 구하기 |
| 11/5 | 캐릭터 움직임 로직 파악 |  |  |
| 11/6 |  |  | 채팅창 ui구조 구현 |
| 11/7 | 로그인, 회원가입 prototype(txt 파일) | Overlaped io에 대해서 공부 |  |
| 11/8 |  | 채팅 콜백 함수 작성 | 채팅창 입력 구현 |
| 11/9 |  | 클라에서 입력을 받으면 do\_chat\_send() 로 서버에 보내줌 |  |
| 11/10 | 로그인, 회원가입 prototype(txt 파일) | 클라에서 입력을 받으면 do\_chat\_send() 로 서버에 보내줌 | 서버 스레드 구조 설계 |
| 11/11 | 로그인, 회원가입 prototype(txt 파일) | do\_chat\_recv() 로 서버에서 받음 | 서버 스레드 구조 설계 |
| 11/12 |  | do\_chat\_recv() 로 서버에서 받음 | 팀원들 도와주기 |
| 11/13 | MySQL, visual studio 연동 |  | 서버 스레드 설계를 기반으로 작성 |
| 11/14 | MySQL, visual studio 연동 |  | 클라 로직 adjustplayer 서버로 옮기기 |
| 11/15 |  | 서버에서 클라로 do\_chat\_send() 채팅을 보내줌 |  |
| 11/16 | 플레이어 움직임 함수(CLIENT::update()) 서버에 적용 | 서버에서 클라로 do\_chat\_send() 채팅을 보내줌 | 클라 로직 adjustplayer서버로 옮기기 |
| 11/17 | 플레이어 움직임 함수(CLIENT::update()) 서버에 적용 | 클라에서 do\_chat\_recv() 채팅을 받음 | 팀원들 도와주기 |
| 11/18 |  | 클라에서 do\_chat\_recv() 채팅을 받음 | 클라 로직 collp2o서버로 옮기기 |
| 11/19 |  |  |  |
| 11/20 | 플레이어 움직임 함수(CLIENT::update()) 서버에 적용 |  | 클라 로직 collp2o서버로 옮기기 |
| 11/21 |  | 귓속말 구현을 위해 string에서 닉네임을 추출 |  |
| 11/22 | 플레이어 움직임 함수(CLIENT::update()) 서버에 적용 | 추출한 닉네임을 이용해 해당 플레이어에게만 메시지 send |  |
| 11/23 |  | 추출한 닉네임을 이용해 해당 플레이어에게만 메시지 send | 팀원들 도와주기 |
| 11/24 | 플레이어 키 입력 서버와 연동(callback) |  | 클라 로직 hurt/spike\_hurttime/stealthtime서버로 옮기기 |
| 11/25 | 플레이어 키 입력 서버와 연동(callback) |  |  |
| 11/26 | 플레이어 키 입력 서버와 연동(callback) | 받은 메시지를 말풍선으로 출력 | 클라 로직 hurt/spike\_hurttime/stealthtime서버로 옮기기 |
| 11/27 | 플레이어 키 입력 서버와 연동(callback) | 받은 메시지를 말풍선으로 출력 | 회원가입 리소스 구하기 |
| 11/28 |  | 받은 메시지를 말풍선으로 출력 | 회원가입 ui버튼 구현 |
| 11/29 | 로그인, 회원가입 클라UI에 접목 |  | 회원가입 입력 로직 구현 |
| 11/30 |  | 여분의 시간 | 팀원들 도와주기 |
| 12/1 | 로그인, 회원가입 클라UI에 접목 | 여분의 시간 | 팀원들 도와주기 |
| 12/2 | 로그인, 회원가입 클라UI에 접목 |  | 팀원들 도와주기 |
| 12/3 | 버그수정 | 버그수정 | 버그수정 |
| 12/4 | 버그수정 | 버그수정 | 버그수정 |
| 12/5 | 버그수정 | 버그수정 | 버그수정 |
| 12/6 | 버그수정 | 버그수정 | 버그수정 |
| 12/7 | 버그수정 | 버그수정 | 버그수정 |
| 12/8 | 최종 마감일 | | |