Network Game Programming

추진 계획서

* Lazenca, Save Us

과목: 네트워크 게임 프로그래밍

교수 명: 김재경 교수님

학번: 2013180049

2016180038

2016182029

이름: 김천기, 장은선, 이소현

**목차**

1. 애플리케이션 소개
2. High-level 디자인
3. Low-level 디자인
4. 개발환경
5. 팀원 별 역할분담
6. 개발 일정
7. **애플리케이션 소개**

* **게임 이름**: Lazenca, Save Us
* **게임 장르**: 2D 종 스크롤 슈팅게임
* **게임 소개**: Galage를 모작한 2인용 2D 종 스크롤 슈팅 게임이다. 각 플레이어들은 자신의 비행기를 상, 하, 좌, 우로 움직이며 적의 공격을 피해 총알을 발사해 적을 무찌른다.

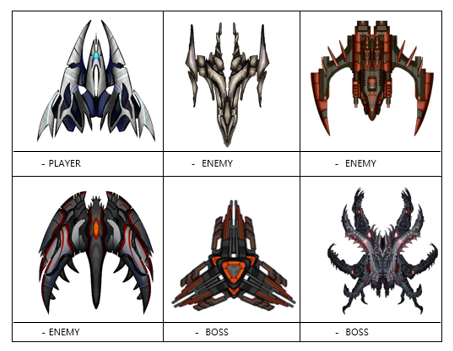


* 게임 실행화면 (예시)
* **조작법**

|  |  |
| --- | --- |
| **키** | **설명** |
| 방향키 | 플레이어 이동 |
| Space bar | 총알 발사 |
| V | 필살 기 |

* **게임설정**

1. **Object**

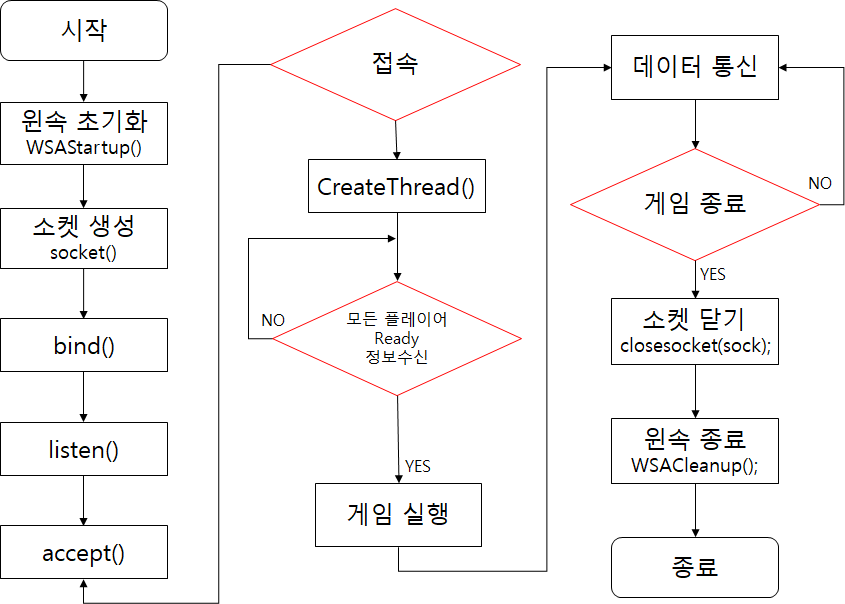


1. **Skill & Item**



1. **High-Level 디자인**

* **서버 순서도**



* **클라이언트 순서도**



**3.Low-level 디자인**

서버 구현 방식: TCP

캐릭터의 움직임이 중요한 게임이기 때문에 데이터 손실이 일어났을 경우, 연속적인 데이터 중간에 데이터 손실로 버림이 일어날 경우에 데이터의 연속성이 끊겨 움직임이 끊어져 보일 수 있기 때문에 모든 데이터를 정확하게 주고받을 수 있는 TCP를 사용한다.

**<송수신 데이터>**

**- 클라이언트 🡪 서버**

플레이어 레디 정보

키 입력 정보

**- 서버 🡪 클라이언트**

플레이어 레디 정보

플레이어 생존 정보

플레이어의 좌표 값

공격의 좌표 값

적의 체력

점수

- 클라이언트

class PlayerInfo{ **- 플레이어의 정보 구조체**

POINT Pos;

int Hp;

bool Shield;

bool SubWeapon;

bool Power;

int BulletCount;

int Score;

}

class EnemyInfo{ **- 적의 정보 구조체**

int Type;

int Hp;

int BoomCount;

bool IsBoom;

}

class ClientInfo{ **- 클라이언트의 정보 구조체**

int Scene;

bool IsReady;

}

enum E\_SCENE **- SCENE 번호**

{

E\_TITLE,

E\_MENU,

E\_INGAME,

E\_GAMEOVER,

E\_RANK

}

**- 윈속 초기화**

Int WSAStartup()

**- 소켓 생성**

socket()

**- 사용자 정보 입력**

ip주소, 닉네임

**- 서버접속**

connect()

**- 레디 정보를 송신**

void SendInitData(ClientInfo)

**- 게임 실행**

<KeyDown 시>

- void SendData(PlayerInfo , EnemyInfo**) - 모든 OBJECT들의 정보 송신**

**- 게임 실행**

Int IsPlayerAlive() **–플레이어의 목숨 개수를 반환하여 플레이어의 생존을 확인하고, 남은 목숨을** **그린다.**

bool IsEnemyAlive() **-살아있는 적을 판단하여 그린다.**

void IsDrawItem(bool) **-활성화된 아이템을 판단하여 그린다.**

void IsDrawBullet(bool) **– 화면 안에서 적 오브젝트와 충돌하지 않은 총알을 그린다.**

void DrawScore() **-획득한 점수를 그린다.**

void RecvInGameData(PlayerInfo, EnemyInfo) **–서버에서 받은 정보로 객체 정보를 업데이트 해준다**

**- 게임종료**

void IsAllPlayerDead() **–게임 종료를 위해 모든 플레이어가 죽었는지 확인 후 서버에서 받은 값을 현재 플레이어의 생존, score정보를 저장하고 Score 페이지로 씬을 넘긴다.**

void RecvRankData() **-10위권 내의 랭킹 정보를 송신받는다.**

void DrawRank() **-서버에서 송신 받은 랭킹정보를 그린다.**

- 서버

class ClientInfoToHandle**{ - 모든 플레이어의 레디 정보와 생존 상태를 확인하고자 만든 구조체**

bool isReady;

bool isDead';

}

**- 접속클라이언트 관리**

ClientInfoToHandle clientinfotohandle[2]

client\_count = 0;

**-** **접속**

void SetReady() **– 플레이어의 ready 정보를 토대로 레디 상태를 설정하는 함수**

bool IsAllClientReady() **- 모든 클라이언트의 레디 상태를 확인하는 함수**

**- accept**

**-** **createthread**

전역변수 int g\_ClientNum = 0;

processClient() **- 접속한 순서대로 g\_ClientNum 할당한다.**

- **레디정보 수신**

void RecvInitData(ClientInfo) **- ready 정보 받고 ClientInfo 의 상태를 바꾼다**

void IsAllClientReady(bool**) - 모든 클라이언트가 레디 상태이면 게임실행 씬으로 넘어간다.**

**- 게임실행**

class PlayerInfo{ **- 플레이어 정보 구조체**

POINT Pos;

int Hp;

int BulletCount;

bool Shield;

bool SubWeapon;

bool Power;

int Score;

}

class EnemyInfo{ **- 적 정보 구조체**

int Type;

int Hp;

int BoomCount;

bool IsBoom;

}

void RecvInGameData(PlayerInfo, EnemyInfo) **-게임 실행 중 필요한 데이터(아이템 획득 여부, 적 죽음 여부, 점수 등) 수신한다.**

void SendInGameData(PlayerInfo, EnemyInfo) **-게임 실행 중 필요한 데이터(아이템 획득 여부, 적 죽음 여부, 점수 등) 송신한다.**

void CheckEnemybyPlayerBulletCollision(playerbullet, EnemyInfo) **-내 총알과 적의 충돌체크 🡪 멤버변수 값을 변경한다.**

void CheckPlayerbyEnemyBulletCollision(enemybullet, PlayerInfo) **-적 총알과 나의 충돌체크 🡪 멤버변수 값을 변경한다.**

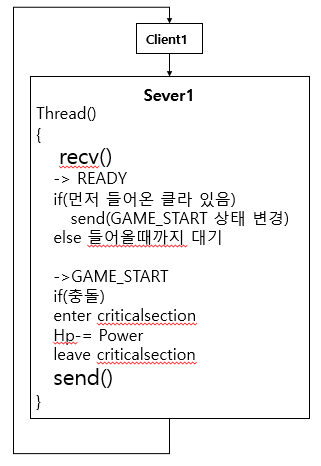
**-** **게임 종료**

bool IsAllPlayerDead(int Client1PHp, int Client2PHp) **-모든 플레이어가 죽으면 게임을 종료한다.**

**- 스코어**

void SetRank() **–이전에 저장된 랭킹정보와 현재 플레이어의 정보를 삽입하여 정렬한다.**

void SendRankData() **-10위권 내의 정보를 클라이언트에게 전송한다.**



- 쓰레드 함수 임계 영역 (수도코드)

**4.개발환경**

|  |  |  |
| --- | --- | --- |
| 김천기 | 이소현 | 장은선 |
| 서버 프레임워크 제작 | 프레임워크 제작 | 클라이언트 보조 |
| 레디 송수신 구현 | 클라이언트 메인 | 동기화문제 해결 |
| 스레드 함수 설계 및 구현 | 클라이언트측 송수신 구현 | 스레드 함수 설계 및 구현 |
| 충돌 체크 함수 설계 및 구현 | 스코어 송수신 구현 | 충돌 체크 함수 설계 및 구현 |

|  |  |  |
| --- | --- | --- |
|  | 클라이언트 | 서버 |
| IDE | MS Visual Studio 2017 | |
| OS | Windows 10 | |
| Language | Visual C++ | |
| Feature | Windows API | TCP |

**5.팀원 별 역할분담**

**6.개발일정**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 김천기 | | 이소현 | | 장은선 | |
| 내용 | 비고 | 내용 | 비고 | 내용 | 비고 |
| 25 일 | 기획서 작성 | | | | |  |
| 26 일 |  |
| 27 일 |  |
| 28 일 |  |
| 29 일 |  |
| 30 일 | 시험기간 | | | | |  |
| 31 일 |  |
| 1 일 |  |
| 2 일 | 서버 프레임워크 |  | 클라 프레임워크 |  | 클라이언트 보조 |  |
| 3 일 | 서버 프레임워크 |  | 클라 프레임워크 |  | 클라이언트 보조 |  |
| 4 일 | 서버 프레임워크 |  | 클라이언트 제작 |  | 클라이언트 보조 |  |
| 5 일 | \*Ready() 함수 구현 |  | 클라이언트 제작 |  | 클라이언트 보조 |  |
| 6 일 | \*Ready() 함수 구현 |  | 클라이언트 제작 |  | 클라이언트 보조 |  |
| 7 일 | \*GameData()  게임 실행 중 데이터 통신 함수 설계 |  | 클라이언트 데이터 전송 테스트 |  | \*GameData()  게임 실행 중 데이터 통신 설계 |  |
| 8 일 | 주간 수정 및 보안 |  |  |  |  |  |
| 9 일 |  |  |  |  | 근로장학생으로 일하는 부서에서 주관하는 캠프 참여 |  |
| 10 일 |  |  | 서버& 클라  테스트 보완 |  |  |
| 11 일 | \*GameData() 게임 실행 중 데이터 통신 |  | SendData() 구현 |  |  |
| 12 일 | \*GameData() 게임 실행 중 데이터 통신 |  | SendData() 구현 |  |  |  |
| 13 일 | \*GameData()  게임 실행 중 데이터 통신 |  |  |  | \*GameData()  게임 실행 중 데이터 통신 |  |
| 14 일 | \*GameData()  게임 실행 중 데이터 통신 |  | SendData() 테스트 |  | \*GameData()  게임 실행 중 데이터 통신 |  |
| 15 일 | 게임종료시  종료,랭크저장,  랭크전송 함수구현 |  |  |  | 게임종료시  종료,랭크저장,  랭크전송 함수구현 |  |
| 16 일 | G-STAR 게임 박람회 참관 |  | SendData() 보완 |  | 게임종료시  종료,랭크저장,  랭크전송 함수구현 |  |
| 17 일 |  | RecvInitData()에  따른 클라 수정 |  | 게임종료시  종료,랭크저장,  랭크전송 함수구현 |  |
| 18 일 |  |  |  |  |  |
| 19 일 | 수정 및 보안 |  |  |  |  |  |
| 20 일 | 수정 및 보안 |  | IsAllClientReady()  구현 |  | 수정 및 보완 |  |
| 21 일 | 중간 점검 |  |  |  | 중간 점검 |  |
| 22 일 | \*BulletCollision()  설계 |  |  |  | \*BulletCollision()  설계 |  |
| 23 일 |  |  | SendInGameData() 서버&클라 테스트 |  | \*BulletCollision()  충돌체크 함수 구현 |  |
| 24 일 |  |  | SendInGameData() 클라 보완 |  |  |  |
| 25 일 | \*BulletCollision()  충돌체크 함수 구현 |  | SendInGameData() 클라 보완 |  |  |  |
| 26 일 | \*BulletCollision()  충돌체크 함수 구현 |  | 미흡한 부분 보완 |  | 동기화 문제 해결 |  |
| 27 일 | \*BulletCollision()  충돌체크 함수 구현 |  | 미흡한 부분 보완 |  | 동기화 문제 해결 |  |
| 28 일 | 미흡한 부분 보완 |  | 테스트 |  | 동기화 문제 해결 |  |
| 29 일 | 주간 수정 및 보완 |  | 버그수정 |  | 동기화 문제 해결 |  |
| 30 일 | 최종 수정 및 보완 |  | 최종 테스트&수정 |  | 미흡한 부분 보완 |  |
| 1 일 |  | 최종 테스트&수정 |  | 최종 점검 및 보완 |  |
| 2 일 |  |  |  |
| 3 일 |  |  |  |
| 4 일 | 최종 점검 및 수정 | | | | |  |
| 5 일 |  |
| 6 일 |  |
| 7 일 |  |
| 8 일 |  |
| 9 일 |  |
| 10 일 |  |