Network Game Programming

추진 계획서

* Lazenca, Save Us

과목: 네트워크 게임 프로그래밍

교수 명: 김재경 교수님

학번: 2013180049

2016180038

2016182029

이름: 김천기, 장은선, 이소현(PM)

**목차**

1. 애플리케이션 소개
2. High-level 디자인
3. Low-level 디자인

4.동기화 이슈와 스레드 설계

5. 개발환경

6. 팀원 별 역할분담

7. 개발 일정

1. **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.



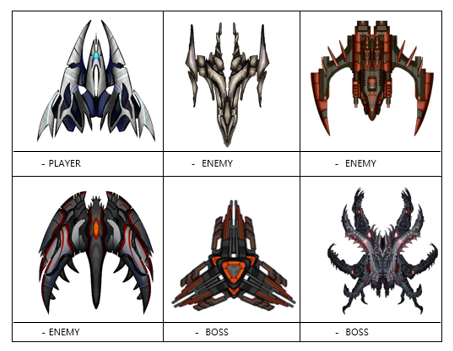
* 게임 실행화면 (예시)
* **조작법**

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

편의를 위해 V에서 M키로 변경하였습니다

* **게임설정**

1. **Object**



,PLAYER2

1. **Skill & Item**



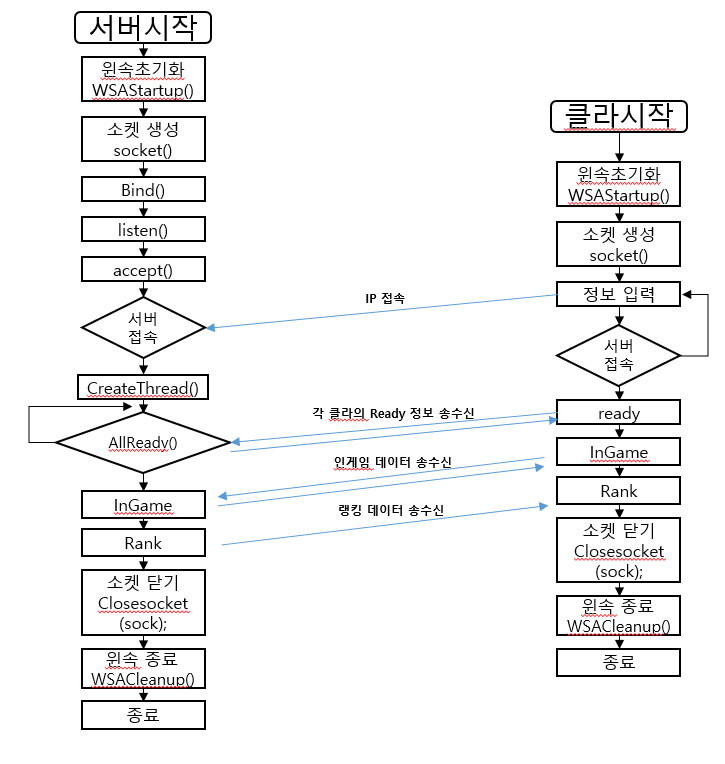
미구현

미구현

미구현

1. **High-Level 디자인**

* **서버 & 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

**- 플레이어의 정보 구조체**

class PlayerInfo{

POINT Pos;

int Hp;

bool Shield;

bool SubWeapon;

bool Power;

int BulletCount;

int Score;

}

**- 총알 정보 구조체**

class BulletInfo {

public:

bool Active;

int index;

POINT Pos; };

**- 적의 정보 구조체**

class EnemyInfo{

int Type;

int Hp;

~~int BoomCount;~~

~~bool IsBoom;~~ 🡪 bool alive;(변수의 용도를 더 잘 설명해주는 이름으로 교체)

<추가된 부분>

int index; 🡪 몇 번째 몬스터인지 접근하기 위함

POINT pos; 🡪 몬스터의 위치를 파악하기 위함

}

**- 클라이언트의 정보 구조체**

class ClientInfo{

int Scene;

bool IsReady;

<추가된 부분>

bool IsDead = false;

int PlayNum; - 몇번째 클라이언트인지 구분하기 위함

}

class ItemInfo { //아이템 정보 구조체

public:

bool alive;

int Type;

POINT pos;

int Index; };

**-Key 정보 구조체**

class KeyIn{

public:

bool Left;

bool Right;

bool Up;

bool Down;

bool Space;

bool Skill; };

**- SCENE 번호**

enum E\_SCENE

{

~~E\_TITLE,~~ 미사용으로 인한 삭제

~~E\_RANK~~ 미사용으로 인한 삭제

E\_MENU,

E\_INGAME,

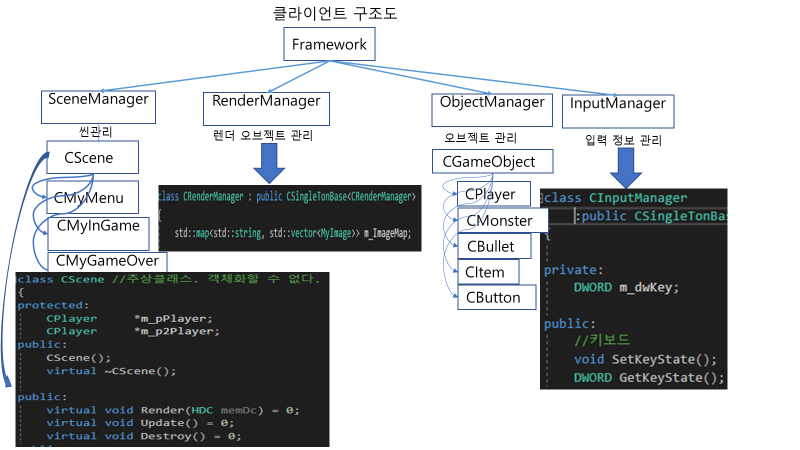
E\_GAMEOVER,

}

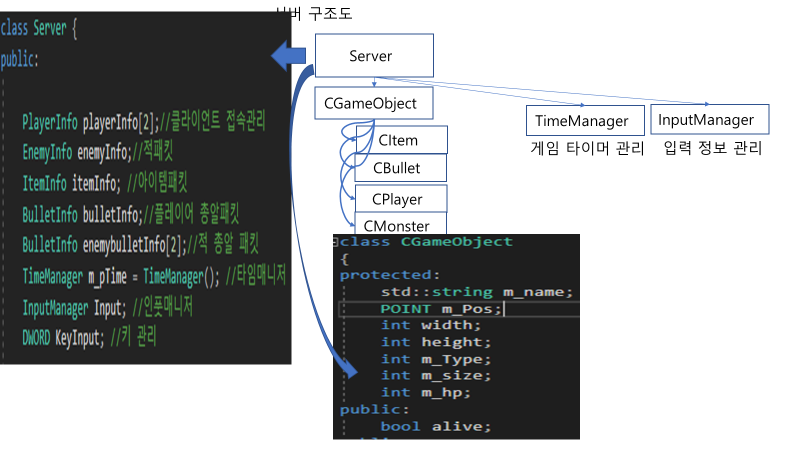
typedef pair<int, string> Score;

|  |  |  |  |
| --- | --- | --- | --- |
| 씬 | 보내는 것 | 주기 | 이동 |
| 메뉴 | 레디정보 | 매 프레임 | Client -> Server |
| 씬 정보 | 매 프레임 | Server -> Client |
| 플레이어 번호 | 씬 정보가 바뀔 때 1회 | Server -> Client |
| 인 게임 | 키 입력 | 매 프레임 | Client -> Server |
| 플레이어 정보 | 매 프레임 | Server -> Client |
| 적 정보 | 매 프레임 | Server -> Client |
| 적 총알 정보 | 매 프레임 | Server -> Client |
| 아이템 정보 | 매 프레임 | Server -> Client |
| 플레이어 총알 | 매 프레임 | Server -> Client |
| 게임오버 | 점수 및 랭크 | 게임이 끝날 경우 1회 | Server -> Client |

* 데이터 전송주기
* 프레임 당 패킷 전송 횟수: 30번
* 클라이언트 클래스 구조도



* 서버 클래스 구조도



**- 윈속 초기화**

Int WSAStartup()

**- 소켓 생성**

socket()

**- 사용자 정보 입력**

ip주소, 닉네임

**- 서버접속**

connect()

**- 레디 정보를 송신**

~~void SendInitData(ClientInfo)~~

🡪 void SendInitData()

메뉴씬에서 ClientInfoToHandle 를 선언하고 이곳에 정보를 송수신하여 정보를 저장하는 방식으로 바꾸어 매개변수를 사용하지 않았다. 플레이어가 레디 상태일 때 통신을 하여 모든 클라이언트가 레디이면 플레이어가 몇 번째 플레이어인지 식별하는 정보를 받고, 인게임 씬으로 씬전환 한다.

**- 게임 실행**

<KeyDown 시>

INPUTMANAGER->GetKeyState() 를 사용하여 메뉴씬에서는 메뉴씬의 업데이트에서, 현재 키상태를 받아오고

인게임에서는 PLAYER의 CHECKKEY함수에서 키 정보를 송수신한다.

<void CMyInGame::Update()>

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

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

🡪 인게임의 업데이트에서 정보 송수신을 처리한다. 서버에서 플레이어 정보, 아이템정보, 적 정보를 연산 후 클라이언트로 보내주면 클라이언트는 멤버변수의 상태에 따라 렌더링한다.

void MakeItem(), void MakeEnemys(); -서버에서 시간마다 적과 아이템을 만들고 만들어진 적과 아이템 정보를 송신하면 클라이언트는 인게임씬의 생성자에서 메모리를 잡아 할당해 두고 인덱스에 따라 alive변수 상태를 변경하여 렌더링하고, 좌표와 사이즈를 받아온다.

🡪적 총알, 플레이어 총알 및 모든 객체를 인게임씬의 업데이트에서 위 방식과 같이 클라이언트에선 미리 정보를 잡아두고 서버에서 만들어졌다는 신호를 alive변수로, 좌표를 수신하여 렌더링한다.

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

🡪 씬 정보를 송수신하여 수신 한 씬 정보가 게임오버 씬이면 씬 정보를 바꾼다.

**- 게임 실행**

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

Player의 GetHp()함수를 사용하여 m\_Hp가 0보다 크면 InGame->Render()에서 렌더링한다

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

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

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

객체의 alive변수가 true면 InGame->Render()에서 렌더링한다

**- 게임종료**

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

MyGameOver 의 생성자에서 정보를 송신받는다

~~void DrawScore()~~ **~~-획득한 점수를 그린다~~.**

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

MyGameOver->Render();에서 서버에서 송신받은 닉네임, 랭킹,점수 정보를 그립니다

- 서버

**- 접속클라이언트 관리**

ClientInfoToHandle clientinfotohandle[2]

client\_count = 0;

**-** **접속**

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

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

**- accept**

**-** **createthread**

~~전역변수 int g\_ClientNum = 0~~; (불필요)

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.**

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.**

서버

Object의 좌표를 얻는 함수

POINT CGameObject::GetPos()

int CGameObject::GetXPos()

int CGameObject::GetYPos()

Object의 Hp를 얻는 함수

int CGameObject::GetHp()

Object의 크기를 얻는 함수

int CGameObject::GetSize()

Object의 Type을 얻는 함수

int CGameObject::GetType()

Object의 Alive값을 얻는 함수

bool CGameObject::GetAlive()

Object의 좌표를 설정하는 함수

void CGameObject::SetPos(int x, int y)

void CGameObject::SetXPos(int x)

void CGameObject::SetYPos(int y)

Object의 Hp를 설정하는 함수

void CGameObject::SetHp(int hp)

Object의 크기를 설정하는 함수

void CGameObject::SetSize(int size)

Object의 Type을 설정하는 함수

void CGameObject::SetType(int type)

Object의 Alive값을 설정하는 함수

void CGameObject::SetAlive(bool life)

Bullet이 생성될 때 좌표와 타입에 따라 다르게 생성

CBullet(POINT pos, int typeNum)

Bullet의 Type값을 얻는 함수

int CBullet::getType()

Bullet과 적이 충돌했는지 체크하는 함수

bool CBullet::IsCrashtoEnemy(CGameObject enemy)

Bullet과 플레이어가 충돌했는지 체크하는 함수

bool CBullet::IsShootPlayer(PlayerInfo player)

Item이 생성될 때 아이템 정보에 따라 다르게 생성

CItem(ItemInfo itemInfo)

Item의 방향을 설정하는 함수

void CItem::SetDir(char c, bool b)

Item의 방향을 얻는 함수

bool CItem::GetDir(char c)

Item의 인덱스를 얻는 함수

int CItem::GetIndex()

Item의 플레이어가 아이템을 얻었는지 체크하는 함수

bool CItem::IsGetItem(PlayerInfo player)

Monster가 생성될 때 적 정보에 따라 다르게 생성

CMonster(EnemyInfo enemyInfo)

Monster의 안의 적이 가지고 있는 총알 벡터

vector<CBullet> m\_EnemyBullet

Monster의 방향을 설정하는 함수

void CMonster::SetDir(char c, bool b)

Monster의 방향을 얻는 함수

bool CMonster::GetDir(char c)

Monster의 인덱스를 얻는 함수

int CMonster::GetIndex()

Monster의 인덱스를 설정하는 함수

void CMonster::SetIndex(int index)

랭크파일을 열어 랭크 벡터에 저장하는 함수.

istream&ReadInputFile(istream& in, vector<Score>& vec)

스레드 함수. 모든 정보 처리를 이 함수에서 함

DWORD WINAPI ProcessClient(LPVOID arg)

들어오는 클라이언트 순서로 플레이어의 정보 초기값을 설정해줌

void Server::SetInitData(PlayerInfo& a, int num)

모든 클라이언트들이 레디했는지 확인하는 함수

bool Server::IsAllClientReady()

플레이어의 좌표 등 바뀌는 정보를 송신

void Server::SendAllPlayerInfo( PlayerInfo P[])

적 생성 함수

void Server::MakeEnemy()

아이템 생성 함수

void Server::MakeItem()

스킬 – 적 간 충돌체크

void Server::SkillCollision(vector<CMonster> &Target)

플레이어 – 아이템 간 충돌체크

void Server::CheckItembyPlayerCollision( vector<CItem>& item, PlayerInfo& player)

플레이어 총알 – 적 간 충돌체크

void Server::CheckEnemybyPlayerBulletCollision(vector<CBullet> &Bullet, vector<CMonster> &Target)

적 총알 – 플레이어 간 충돌체크

void Server::CheckPlayerbyEnemyBulletCollision(vector<CBullet>Bullet, PlayerInfo& player)

이전에 저장된 랭킹에 현재 정보 삽입 및 정렬 함수

void Server::SetRank(ofstream &out, vector<Score>& vec, Score temp)

지난 로우레벨에서 불필요해서 제거한 함수들

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

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

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

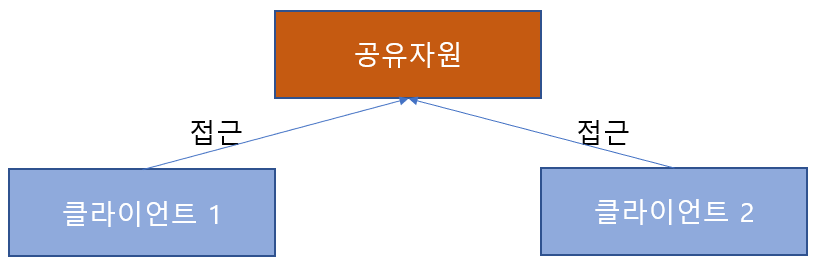
~~void SendInGameData(PlayerInfo, EnemyInfo)~~ **~~-게임 실행 중 필요한 데이터(아이템 획득 여부, 적 죽음 여부, 점수 등) 송신한다.~~**

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

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

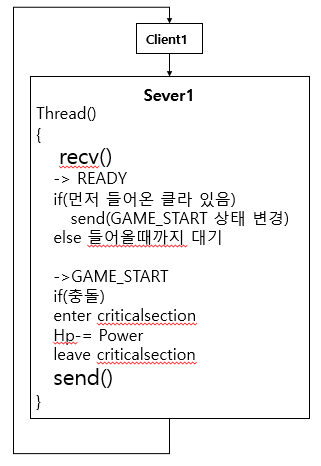
**4. 동기화 이슈와 스레드 설계**

- 동기화 이슈가 발생하는 이유

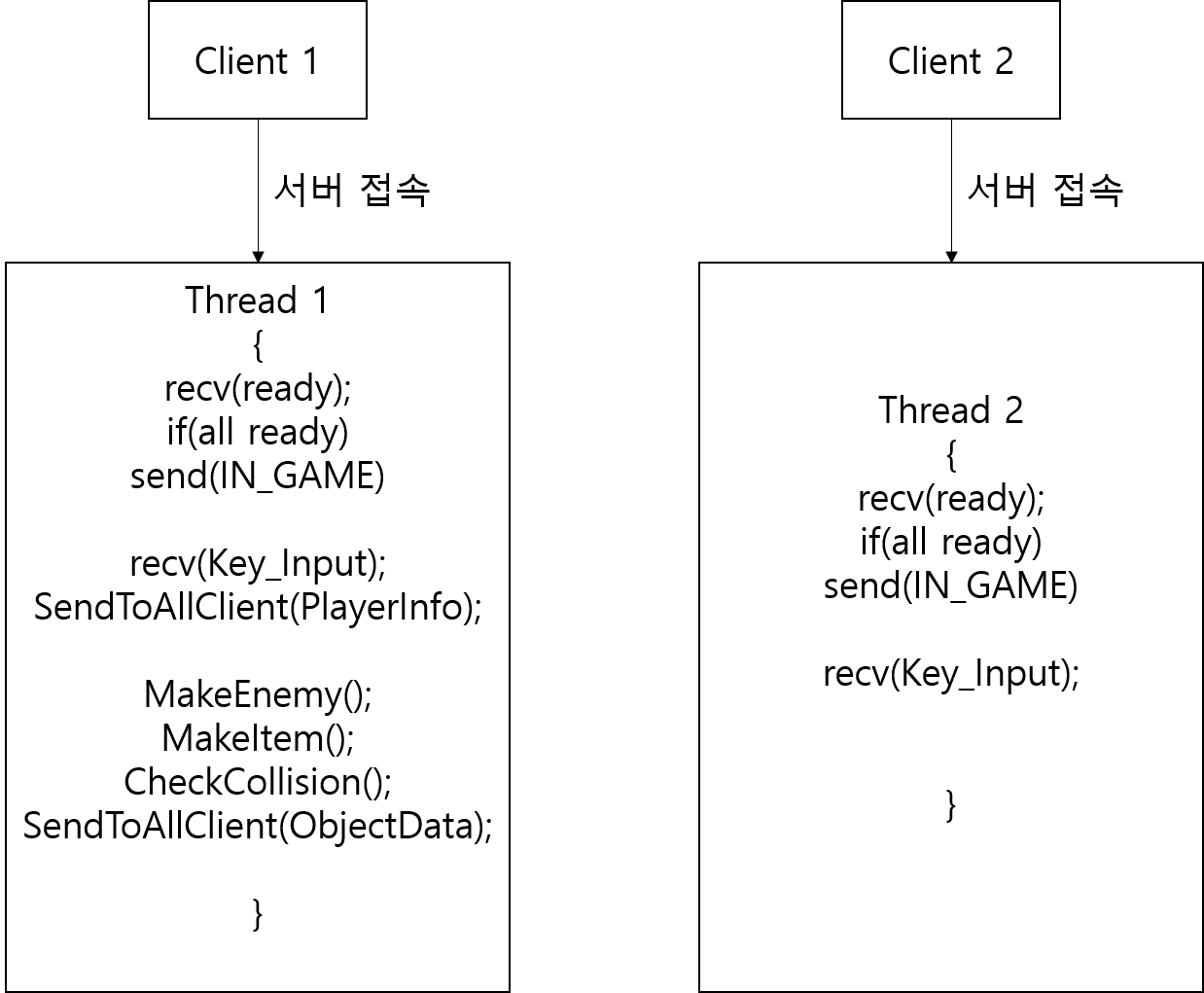


공유자원에 대해 다른 플레이어들이 동시에 정보를 송신하여 공유자원의 정보를 변경하면 정보에 혼란이 생긴다. 따라서 공유자원에 정보가 쓰일 때, 다른 클라이언트에서 읽거나 동시에 쓰는 작업을 수행하지 않아야 한다. 두 스레드에서 공유자원에 접근하는 방식에서 이 문제를 해결하기 위해 한 스레드에서 모든 데이터를 관리하는 방법으로 변경하여 공유자원 문제를 해결했다.

<이전 공유자원 해결 방식>



<최종 공유자원 해결 방식>



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

**5.개발환경**

**6.팀원 별 역할분담**

|  |  |  |
| --- | --- | --- |
| 김천기 | 이소현 | 장은선 |
| 예상 역할분담 | | |
| 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 |
| 실제 역할분담 | | |
| 서버 프레임워크 제작 | 클라이언트 프레임워크 제작 | 서버 프레임워크 제작 |
| 보조 클라이언트 | 메인 클라이언트 | 보조 클라이언트 |
| 레디 송수신 구현 | 레디 송수신 구현 | 레디 송수신 구현 |
| 스레드 함수 설계 및 구현 | 동기화 문제 해결 | 스레드 함수 설계 및 구현 |
| 충돌체크 함수 설계 및 구현 | 충돌체크 함수 및 구현 | 스코어 송수신 구현 |
| 공유자원 설계 | 서버 보조 | 공유자원 설계 |

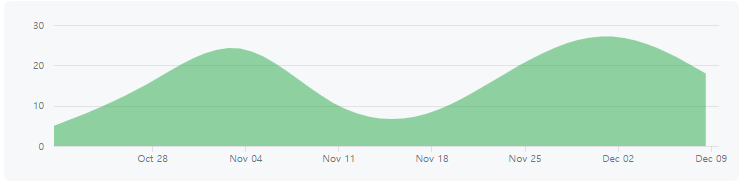
**7.개발일정**

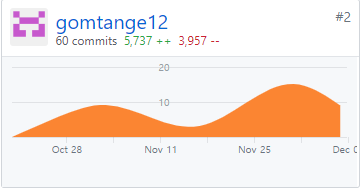
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 김천기 | | |  | 이소현 | | |  | 장은선 | | |
| 내용 | |  | 비고 | 내용 | | 대체 | 비고 | 내용 | | 대체 | 비고 |
| 25~29 | 기획서작성 | | | | | | | | | | | |
| 30~1 | 시험기간 | | | | | | | | | | | |
| 2 일 | 서버 프레임워크 | |  | O | 클라 프레임워크 | |  | X | 클라이언트 보조 | |  | O |
| 3 일 | 서버 프레임워크 | |  | O | 클라 프레임워크 | |  | X | 클라이언트 보조 | |  | O |
| 4 일 | 서버 프레임워크 | |  | O | 클라이언트 제작 | |  | X | 클라이언트 보조 | |  | O |
| 5 일 | \*Ready() 함수 구현  🡪 서버 프레임워크 보완 | |  | O | 클라이언트 제작 | |  | X | 클라이언트 보조 | |  | O |
| 6 일 | \*Ready() 함수 구현 | |  | O | 클라이언트 제작  🡪씬, 렌더매니저, 프레임워크 구현 | |  | O | 클라이언트 보조 | |  | O |
| 7 일 | \*GameData()  게임 실행 중 데이터 통신 함수 설계   * 레디구현 테스트 | |  | O | 클라이언트 데이터 전송 테스트  ->레디구현 테스트 완료 | |  | O | \*GameData()  게임 실행 중 데이터 통신 설계  ->클라이언트 보조, 인게임 데이터 송수신 설계, 레디 구현 테스트 | |  | O |
| 8 일 | 주간 수정 및 보안 | |  |  |  | |  |  |  | |  |  |
| 9 일 | 타임 매니저 설계-> 적 오브젝트 구현 (버그O)  ->다음달 19일 이소현이 수정 | |  | △ | 인풋 매니저, 버튼 구현, 플레이어 이동 수정, 레디 정보 송수신에 따른 클라이언트 수정 | |  | O | 근로장학생으로 일하는 부서에서 주관하는 캠프 참여 | |  |  |
| 10 일 |  | △ | 서버& 클라  테스트 보완 | |  | △ |  |  |
| 11 일 | \*GameData() 게임 실행 중 데이터 통신  -> 클라이언트 보조 | |  |  | SendData() 구현  ->플레이어 총알 생성 | |  | △ |  |  |
| 12 일 | \*GameData() 게임 실행 중 데이터 통신  ->씬전송으로 인한 버그 수정,  방향키 송수신, 씬 송수신수정 | |  |  | SendData() 구현->플레이어 총알 생성(버그O) | |  | △ | 메뉴 업데이트수정(버그O) | |  | O |
| 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 | \*BulletCollision()  설계 | |  |  |  | |  |  | \*BulletCollision()  설계 | |  |  |
| 16 일 | G-STAR 게임 박람회 참관 | |  |  | SendData() 보완 | |  |  | \*BulletCollision()  충돌체크 함수 구현 | |  |  |
| 17 일 |  |  | RecvInitData()에  따른 클라 수정 | |  |  | \*BulletCollision()  충돌체크 함수 구현 | |  |  |
| 18 일 |  |  | 플레이어 총알 구현 완료 | |  | O |  | |  |  |
| 19 일 | 수정 및 보안  ->타임매니저 설계(버그O) | |  | △ | 총알 렌더링 및 버그 수정 | |  | O |  | |  |  |
| 20 일 | \*BulletCollision()  충돌체크 함수 구현  ->시간에 따른 적생성(버그O) | |  | △  △ | IsAllClientReady()  구현(이전에 구현) | |  | O | 수정 및 보완->종료시 랭크 정보 구현 | |  | O |
| 21 일 | 중간 점검  ->시간에 따른 적 생성(버그O)  🡪 타임매니저, 적생성 29일 이소현이 수정 | |  | △ | 적 이동 버그 수정 | |  | O | 중간 점검  ->종료시 랭크정보 구현 | |  | O |
| 22 일 | \*BulletCollision()  충돌체크 함수 구현 | |  | △ | 충돌체크 구현, 버그 수정 | |  | O | 게임종료시  종료, 랭크저장,  랭크전송 함수 | |  | O |
| 23 일 | 적생성 버그 수정 | |  | △ | SendInGameData() 서버&클라 테스트  ->충돌체크->적-플레이어 총알(버그O) | |  | △ | 게임종료시  종료, 랭크저장,  랭크전송 함수 | |  | O |
| 24 일 | 적생성 버그 수정 | |  | △ | SendInGameData() 클라 보완 | |  | △ |  | |  |  |
| 25 일 | ~~게임종료시~~  ~~종료, 랭크저장,~~  ~~랭크전송 함수~~  ->적이동 송수신 | |  | o | 적 생성, 삭제 | |  | o |  | |  |  |
| 26 일 | ~~게임종료시~~  ~~종료, 랭크저장,~~  ~~랭크전송 함수~~ | |  | x | 미흡한 부분 보완   * 부족한 부분 2일 본인이 수정 | |  | O | ~~동기화 문제 해결~~  프레임고정(버그)  ->부족한 부분 2일 이소현이 수정 | |  | △ |
| 27 일 | 테스트 및 보완->PLAYERnum별 송수신 | |  | △ | 미흡한 부분 보완 | |  | O | ~~동기화 문제 해결~~  ~~->~~키입력 수정, 랭크씬 구현중 | |  | O |
| 28 일 | 미흡한 부분 보완  ->플레이어 Num별 송수신 | |  | o | 테스트 | |  |  | ~~동기화 문제 해결~~  ->랭크씬 구현 | |  | △ |
| 29 일 | 주간 수정 및 보완-> 플레이어 총알, 적 충돌체크 | |  | △ | 버그수정-> 서버에서 적이동, 적생성, 플레이어 총알 이동, 생성, 아이템 이동, 생성 | |  | o | ~~미흡한 부분 보완~~  MT참여  적 생성, 이동 송수신 | |  | △ |
| 30 일 | 최종 수정 및 보완->인게임 데이터 송수신 | |  | △ | 최종 테스트&수정  ->랭크 구현버그 수정🡪 8일 장은선이 최종 보완 | |  | O | 최종 점검 및 보완  (28일 랭크씬 구현의 미흡한 부분 본인이 수정) | |  | △ |
| 1 일 | 적생성, 이동 정보 송수신 | |  | △ | 최종 테스트&수정  ->적이동 송수신 | |  | o |  | |  |  |
| 2 일 | 총알 데이터 송수신, 충돌체크(버그) | |  | △ | 프레임별 패킷송수신 횟수 고정, 충돌체크(버그o), ui, 버튼 구현, 서버들과 협업 | |  | ㅇ | 적 이동 송수신, 아이템 이동, 생성 송수신, 아이템-플레이어 충돌체크(버그) | |  | △ |
| 3 일 | 공유자원 설계, 적 이동 송수신, | |  | o | 아이템 업데이트, | |  | o | 필살기 정보 송수신 | |  | △ |
| 4 일 | 협업: 충돌체크 버그 해결, 구조 변경 | |  | △ | 협업: 충돌체크 버그 해결, 구조 변경 | |  | △ | 협업: 충돌체크 버그 해결, 구조 변경 | |  | △ |
| 5 일 | 공유자원 문제 해결, | |  | O | 적총알 생성 및 이동 | |  | O | 공유자원 문제 해결, | |  | O |
| 6 일 | 협업: 인게임 정보 송수신 보완, 충돌체크 보완, 적 총알 송수신-> 실패 | |  | △ | UI작업  협업: 인게임 정보 송수신 보완, 충돌체크 보완, 적 총알 송수신, | |  | △ | 협업: 인게임 정보 송수신 보완, 충돌체크 보완, 적 총알 송수신->실패 | |  | △ |
| 7 일 | 협업: 최종 테스트 및 보완 | |  | O | 적총알 생성 및 이동 방식 변경  협업: 최종 테스트 및 보완 | |  | O | 랭크씬 닉네임 출력 보완  협업: 최종 테스트 및 보완 | |  | O |
| 8 일 | 협업: 최종 테스트 및 보완, 보고서 작성 | |  | O | 협업: 최종 테스트 및 보완, 보고서 작성 | |  | O | 협업: 최종 테스트 및 보완, 보고서 작성 | |  | O |
| 9 일 | 협업: 최종 테스트 및 보완, 보고서 작성 | |  | O | 협업: 최종 테스트 및 보완, 보고서 작성 | |  | O | 협업: 최종 테스트 및 보완, 보고서 작성, 스레드 구조 변경 | |  | O |
| 10 일 | 협업: 최종 테스트 및 보완, 보고서 작성 | |  | O | 협업: 최종 테스트 및 보완, 보고서 작성 | |  | O | 협업: 최종 테스트 및 보완, 보고서 작성 | |  | O |

**비고: 2~5일 메인 클라이언트 이소현의 시험일정 조정으로 초반 클라이언트 작업이 늦어졌고, 13일~17일또한 이소현의 질병으로 인한 부재로 개발이 늦어졌습니다. 1차 중간검사 기준 초기 계획 대비 55%의의 진행률을 완성 하였고, 2차 중간검사 기준 계획 수정 후 수정된 계획 대비 진행률의 65% 달성하였습니다**

**깃 허브 커밋:**

* **일자별 정리**



* **개인별 정리**
* **이소현**
* 
* **김천기**
* 

**장은선 35~40회(계정 변경으로 정확한 횟수 산출X)**