**네트워크 게임 프로그래밍**

**Term Project 추진계획서**

2016184018 박형순

2016150019 박재홍

**목차**

1. 게임 소개

2. 요구사항

3. High Level Design

4. Low Level Design

5. 역할 분담

6. 개발환경

7. 개발일정

**1. 게임 소개**

**게임 이름:** TANK

**게임 장르:** 슈팅, 경쟁

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

<게임 예시>

**게임 설명**

* 각 플레이어가 서로를 맞춰서 상대방의 최종 HP가 0이 되면 승리하는 2인용 게임입니다.
* 총 HP는 3개이며, 한 번 맞을 때마다 하나씩 감소합니다.
* 한 번 죽을 때마다 처음 시작했던 위치에서 리스폰됩니다.
* 한 번 공격하고 나면 2초가 지난 후에 공격이 가능합니다.

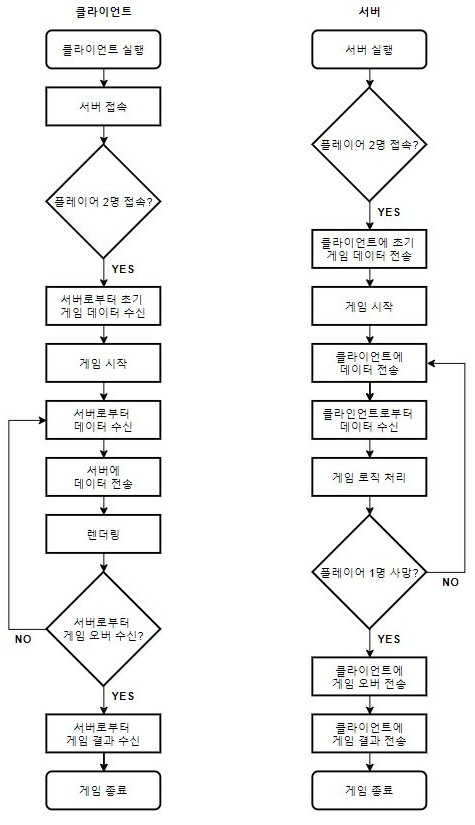
**게임 조작법**

* 이동: 방향키
* 공격: 스페이스바

**2. 요구사항**

* 네트워크 방식은 TCP를 사용한다.
* 클라이언트를 실행하면 서버의 IP 주소를 입력하여 접속한다.
* 서버는 최대 2개의 클라이언트를 사용한다.
* 서버와 클라이언트는 30프레임을 기준으로 매 프레임마다 데이터를 송수신한다.
* 서버에는 클라이언트와 통신하는 스레드와 게임 로직을 처리하는 스레드가 있는데, 이를 동기화하기 위해서 Event를 사용한다.
* 서버를 실행하면 자동 리셋 이벤트 (mEvent)를 생성한다.
* 충돌 체크 전에는 mEvent가 신호상태가 되기를 무한히 기다린다.
* 충돌 체크 후에는 mEvent를 신호상태로 바꾼다.

**3. High Level Design**



**4. Low Level Design**

**프로토콜**

struct ~~Player~~ Object {

int type; // 0이면 총알, 1이면 플레이어

float x, y; // 플레이어 x, y 좌표

int ID; // 먼저 접속한 플레이어 ID는 1, 나중에 접속하면 2)

int HP; // 플레이어 HP

float time; // 공격 재사용 대기시간

int state; // 플레이어 상태 (다른 플레이어를 기다리면 0, 게임 중이면 1)

float b\_x, b\_y; // 총알의 x, y 좌표

int b\_state; // 총알 상태 (날아가는 중이면 0, 어딘가에 부딫히면 1)

} ~~PLAYER~~ OBJECT

~~struct Bullet {~~

~~float x, y; // 총알 x, y 좌표~~

~~int ID; // 플레이어 1이 발사했으면 ID는 1, 플레이어 2가 발사했으면 2~~

~~int state; // 총알 상태 (날아가는 중이면 0, 어딘가에 부딪히면 1)~~

~~} BULLET~~

**클라이언트**

void sendKey(int keypress) // 플레이어가 입력한 키 값을 서버에 전송

void sendPlayerAccess(int ID) // 플레이어의 접속을 서버에 전송

void recvGameData(Player p, Bullet b) // 서버로부터 게임 데이터를 수신

void recvGameResult(Player p) // 서버로부터 게임 결과를 수신

class Scene // Scnen 클래스를 상속받아 다양한 Scene 클래스

{ // (GameScene, GameResultScene) 제작

protected:

vector<Image> images

vector<Font> fonts

public:

Scene() // 게임에 사용할 리소스 추가

~Scene() // 리소스 할당 해제

void update(Player\* p, Bullet\* b) // 서버로부터 게임 데이터를 받아 업데이트

void render() // 화면에 업데이트된 데이터들을 렌더링

}

**서버**

void recvKey(int keypress) // 플레이어가 입력한 키 값을 수신

void recvPlayerAccess(int ID) // 플레이어의 접속을 수신

void sendGameData(PLAYER p, BULLET b) // 플레이어와 총알의 정보를 클라이언트에게 전송

void sendGameResult(Player p) // 클라이언트에게 게임 결과를 전송

void checkCollision() // 플레이어와 총알 간의 충돌 체크

DWORD WorkerThread(LPVOID arg) // 각종 연산, 충돌 체크 등을 처리하는 스레드 함수

DWORD CommunicateThread(SOCKET client\_sock)

// 각 클라이언트와 통신하는 스레드 함수

// recvKey() 함수로 키 값을 수신. WorkerThread() 함수에서

// 업데이트된 게임 데이터들을 클라이언트에 전송

**5. 역할 분담**

박형순: 게임 내부 전반적인 세계와 프레임워크 제작, CommunicateThread() 구현

박재홍: 서버와 클라이언트 간의 데이터 송수신 구현, WorkerThread() 구현

**6. 개발환경**

* Visual Studio 2019
* SDL 2.0.16
* GitHub

**7. 개발일정**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **월** | **화** | **수** | **목** | **금** | **토** | **일** |
|  |  |  | **11월 3일** | **11월 4일** | **11월 5일** | **11월 6일** | **11월 7일** |
| **박형순** |  |  | 리소스 수집 | 기획서 수정 및 검토 | | | |
| **박재홍** |  |  | SDL 튜토리얼 |
|  | **11월 8일** | **11월 9일** | **11월 10일** | **11월 11일** | **11월 12일** | **11월 13일** | **11월 14일** |
| **박형순** | 기획서 수정 및 검토 | 기획서 검수 | 클라이언트 프레임워크 제작 | | |  | 코드 리뷰 및  수정 |
| **박재홍** | 프로토콜 제작 | 기본 TCP  서버 제작 |  | 데이터 송수신 구현  (이동, 공격) |
|  | **11월 15일** | **11월 16일** | **11월 17일** | **11월 18일** | **11월 19일** | **11월 20일** | **11월 21일** |
| **박형순** | GameScene  클래스 제작 | GameResult  Scene  클래스 제작 |  | 코드 리뷰 및 수정 |  | CommunicateThread() 제작  (패킷 수신과 동기화) | |
| **박재홍** | checkCollision() 제작 | | 데이터 송수신 구현  (게임 결과) |  | WorkerThread() 제작  (게임 로직 업데이트) | |
|  | **11월 22일** | **11월 23일** | **11월 24일** | **11월 25일** | **11월 26일** | **11월 27일** | **11월 28일** |
| **박형순** | CommunicateThread() 제작 | 코드 리뷰 및 수정 | 동기화 추가 및 개선 | |  | 디버깅 | 사운드 추가 구현 |
| **박재홍** | Worker  Thread() 제작 | 동기화 추가 및 개선 | |  | 강제 종료  여부 구현 |
|  | **11월 29일** | **11월 30일** | **12월 1일** | **12월 2일** | **12월 3일** | **12월 4일** | **12월 5일** |
| **박형순** |  | 코드 리뷰 및 수정 |  |  |  | 최종 테스트 및 마무리 작업 | |
| **박재홍** |  |  |  |  |
|  | **12월 6일** | **12월 7일** |  |  |  |  |  |
| **박형순** | 최종 테스트 및  마무리 작업 | 최종 검수 |  |  |  |  |  |
| **박재홍** |  |  |  |  |  |