Tank Hero

****

2017182016 박해성

2017182020 양원석

2017184004 김기윤

목차

[Tank Hero 1](#_Toc87291204)

[1. 애플리케이션 기획 3](#_Toc87291205)

[① 게임 기본 정보 3](#_Toc87291206)

[② 게임 선정 이유 3](#_Toc87291207)

[③ 플레이 방법 3](#_Toc87291208)

[④ 조작법 4](#_Toc87291209)

[2. High – Level Design 5](#_Toc87291210)

[3. Low – Level Design 7](#_Toc87291211)

[ 클래스 / 구조체 7](#_Toc87291212)

[ 프로토콜 9](#_Toc87291213)

[ 상수 10](#_Toc87291214)

[ 함수 11](#_Toc87291215)

[4. 팀원 별 역할 분담 13](#_Toc87291216)

[5. 개발환경 13](#_Toc87291217)

[6. 개발일정 14](#_Toc87291218)

기획서 변경 ( ~~밑줄~~ )

일정 완료 ( 파란글씨 )

# 애플리케이션 기획

1. 게임 기본 정보 :

텍스트이(가) 표시된 사진

자동 생성된 설명

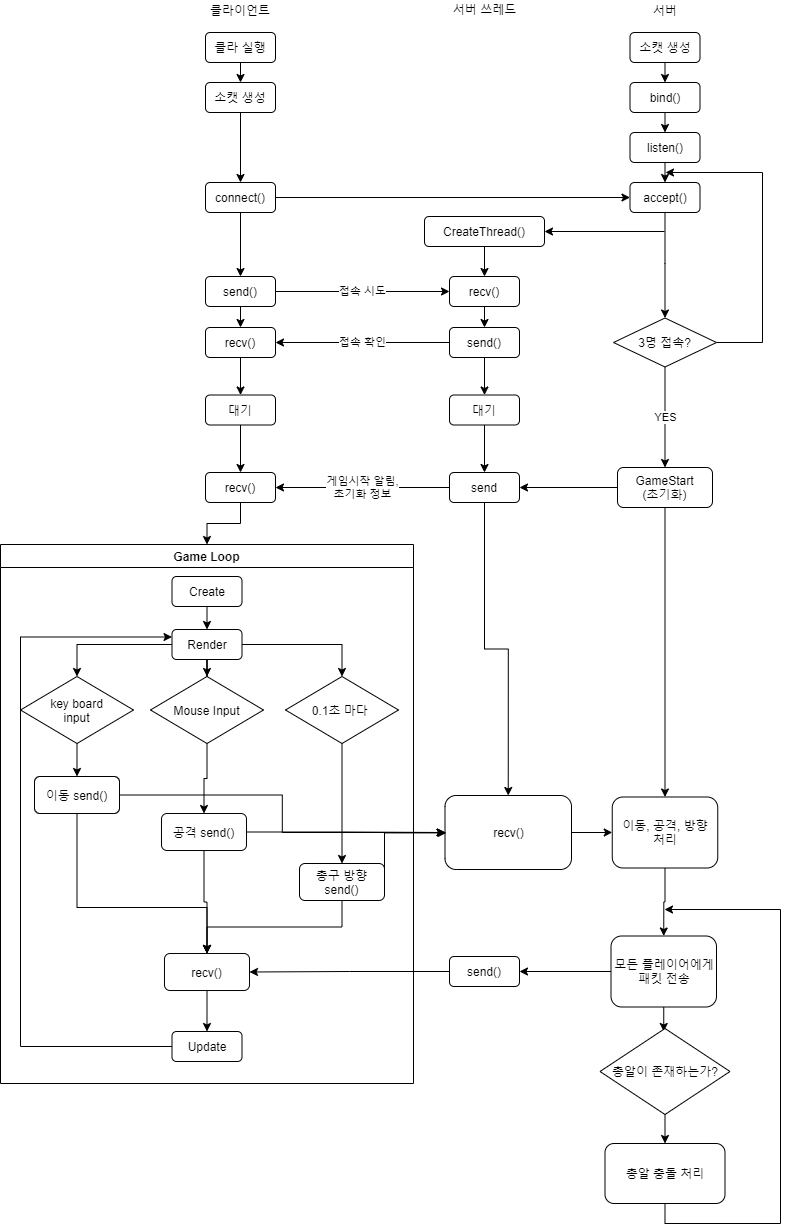
* + - 게임 제목: Tank Hero
    - 게임 컨셉 : 기존에는 AI와 싸워 스테이지를 클리어하는 방식이지만 , AI 대신 다른 플레이어와 싸우는 3인 멀티플레이 게임
    - 게임 소개: 상대방의 포탄을 피하면서 포탄을 발사해 상대방을 맞춰 체력을 전부 깎으면 승리한다.

1. 게임 선정 이유 : 이전에는 네트워크적 요소가 많은 어몽어스를 만들려고 했으나 시야가 플레이어마다 다르고 플레이어 끼리 잘 연결 되었는지 확인하기 어려운 점, 8인 멀티플레이어 게임이기에 테스트 및 검증 부분에서 어려운 점 등을 감안해 비교적 단순한 플레이어 끼리 포탄을 쏴서 맞추는 탱크 게임을 선택하게 되었습니다.
2. 플레이 방법 :
   1. 게임(클라이언트)을 실행하여 서버에 접속한다.
   2. 모든 플레이어(3명)가 입장하면 게임을 시작한다.
   3. 게임이 시작되면 플레이어는 WASD 키로 이동하며 마우스의 커서로 공격 방향을 결정하고 공격키 (마우스 왼쪽)를 누르면 포탄을 발사할 수 있다
   4. 포탄을 맞춰서 상대편의 체력을 전부 깎으면 게임에서 승리한다.
   5. 최후의 승자가 나올 때까지 게임을 진행한다.
3. 조작법 :

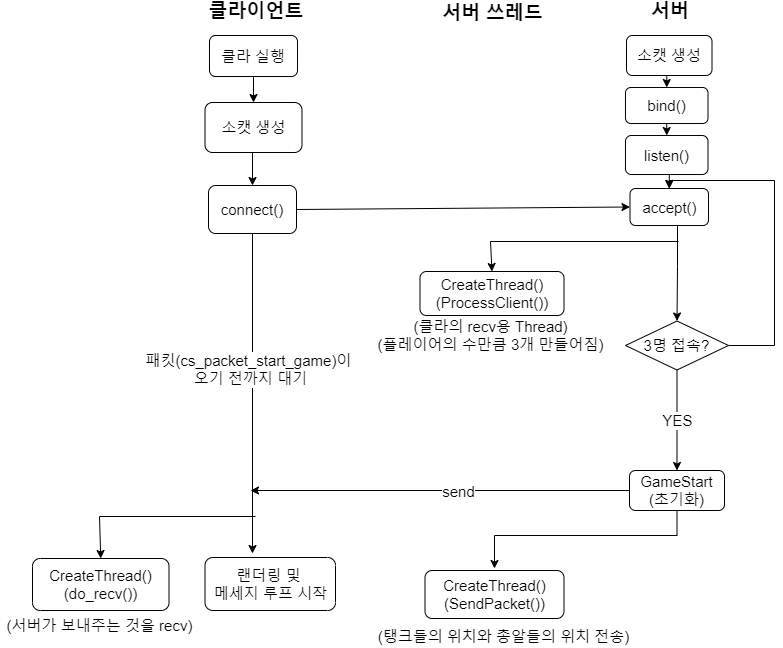
|  |  |
| --- | --- |
| W A S D | 위쪽, 왼쪽, 아래쪽, 오른쪽으로 이동 |
| 마우스 커서 | 탱크의 공격방향을 정함 |
| 마우스 왼쪽 키 | 공격 키 |

# High – Level Design

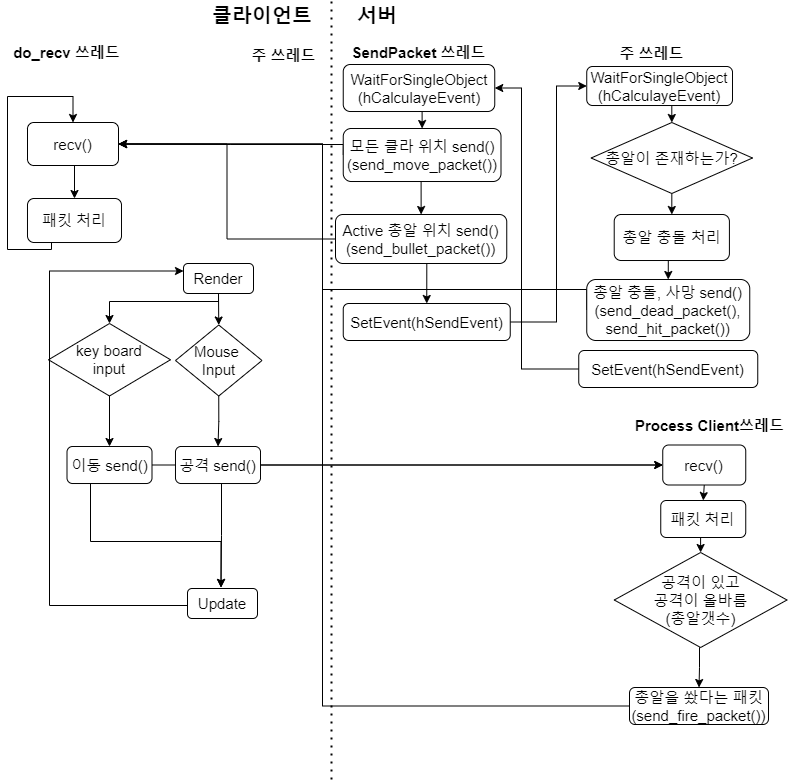
1. ~~Game Flow-Chart~~(초기 기획)



1. Login Flow Chart(현재)



1. Game Flow Chart(현재)

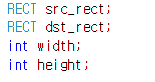


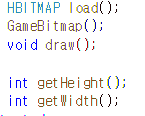
* + 접속
    - 클라이언트를 실행하면 서버에 접속 요청을 하고 서버는 accept가 되면 해당 클라이언트를 위한 쓰레드를 만듭니다
    - 그 후 해당 클라이언트에게 id를 부여하고 id와 함께 접속이 되었다는 의미의 패킷을 보내줍니다
    - 이미 접속한 다른 클라이언트가 있다면 새로 접속한 클라이언트에게 이미 접속해 있는 클라이언트의 id정보를 담은 패킷을 보내고 이미 접속해 있는 클라이언트에게는 새로 접속한 클라이언트의 id정보를 담은 패킷을 보내줍니다
    - 서버는 3명의 클라이언트가 접속하면 게임시작을 위한 정보 초기화(플레이어의 위치)를 해주고 게임이 시작되었다는 패킷을 클라이언트에게 보내줍니다
    - 게임을 시작하면 send를 해주는 쓰레드를 만들고 메인 쓰레드는 총알의 움직임과 충돌 처리를 해줍니다
  + 게임 시작
    - 클라이언트는 게임이 시작되었다는 패킷을 받으면 GUI를 위한 렌더링을 해줍니다
    - 움직임이나 공격을 위한 입력이 있다면 서버에 해당 키를 send()해주고 서버는 알맞은 처리를 해줍니다
    - ~~총구의 방향은 마우스의 커서로 정해지는데 움직임이 있다고 하면 약간만 움직여도 서버에 계속 send를 할 위험이 생길 수 있습니다. 그런 문제점을 막고자 0.1초(?, 수정 가능)마다 서버에 총구 방향을 보내는 것을 생각하였습니다. 또한 움직임과 같이 묶여서 패킷을 전송합니다.~~
    - ~~클라이언트가 총알을 쐈다면 총알을 쐈다는 것을 모든 클라이언트에게 보내줍니다. 총알의 움직임은 서버에서 총알의 위치를 계속 보내주는 것이 아닌 클라이언트에서 처리해줍니다. 다만 서버는 총알이 다른 클라이언트를 맞추는지 아닌지를 계속 확인하며 맞았다면 클라이언트에게 맞았다는 정보를 보내고 hp를 1깎습니다.~~
    - 총알을 쏘고 총알의 움직임을 서버에서 클라이언트에게 ~~0.1초(?)~~1초마다 60번씩 총알의 위치를 보내줍니다. 단, 벽이나 클라이언트(탱크)에 충돌이 일어나면 바로 보내줍니다 (이유 : 네트워크적 요소를 추가하기 위해서)
    - 공유자원 접근에 대해 동기화를 생각하였습니다. 추가적으로 클라이언트의 쓰레드 순서 제어를 하지 않을 경우 불리한 상황이 발생하는 것을 방지하기 위해 동기화 방법으로 이벤트 방식을 사용할 예정입니다.(추가 : 서버간 동기화에서 이벤트 방식을 사용할 시 충돌처리를 하는 부분에서 하는 것이 맞다고 생각하였습니다. 따라서 모든 클라이언트가 한번씩 실행이 되게 하고 한 클라이언트가 중복되서 한번에 여러 번 처리되지 않도록 이벤트 방식을 사용할 예정입니다)

(추가 : 클라이언트에서 send를 하는 경우는 움직이거나 총알을 해줄 때만 보내줍니다. 따라서 recv를 해야지 작동하는 Process Client쓰레드에서 동기화를 하지 않고 총알의 위치수정을 한 후에 send를 해야지 올바르다고 판단해 총알의 위치수정을 해주는 주쓰레드와 SendPacket쓰레드를 이벤트로 동기화 해주었습니다)

# Low – Level Design

* 클래스 / 구조체
  + GameObject Class  
     
  + GameBitmap Class





* + Player Class

텍스트이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명

* + Bullet Class

텍스트이(가) 표시된 사진

자동 생성된 설명



* Wall Class





* Background Class

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* + UI Class

텍스트이(가) 표시된 사진

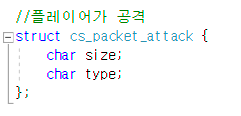
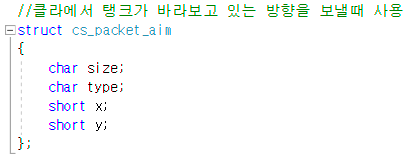
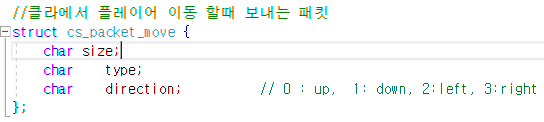
자동 생성된 설명

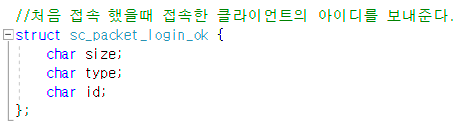
텍스트이(가) 표시된 사진

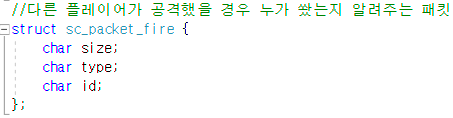
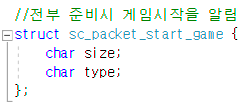
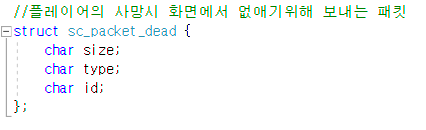
자동 생성된 설명

* 프로토콜

- ~~x, y좌표 short~~ -> float

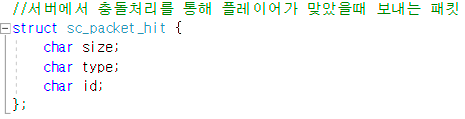


텍스트이(가) 표시된 사진

자동 생성된 설명

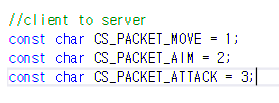
텍스트이(가) 표시된 사진

자동 생성된 설명



* 상수

1. 클라이언트에서 서버로 보내는 패킷 타입



* + 클라이언트에서 서버로 보내는 패킷의 타입을 정의해 서버에서 어떤 패킷을 받았는지 알 수 있게 한다.

1. 서버에서 클라이언트로 보내는 패킷 타입

텍스트이(가) 표시된 사진

자동 생성된 설명

* + 서버에서 클라이언트로 보내는 패킷의 타입을 정의해 클라이언트에서 어떤 패킷을 받았는지 알 수 있게 한다.

1. 기타



- SERVERPORT : 포트 번호를 정의

* 함수

|  |
| --- |
| **클라이언트** |
| * ~~void process\_packet(); : 패킷 재조립을 위한 함수~~ -> do\_recv()쓰레드에서 패킷 처리 * void send\_move\_packet() : 클라이언트가 움직이는 키(W, A, S, D)중 무엇을 눌렀는지 보내주는 함수 * ~~void send\_aim\_packet() : 클라이언트의 대포 방향(총구)를 보내주는 함수~~ * void send\_attack\_packet() : 클라이언트가 마우스 좌측를 눌렀을 때 보내주는 함수 * int netInit() : 서버 접속을 하는 함수 * DWORD WINAPI do\_recv() : 서버가 전송한 패킷을 받고 패킷을 처리하는 쓰레드 함수 |
| **서버** |
| * DWORD WINAPI ProcessClient(LPVOID arg); : ~~클라이언트가 접속하면 해당 클라이언트를 처리해주는 쓰레드 함~~ 클라이언트가 보내주는 패킷을 받는 함수 * void process\_packet(); : 패킷 재조립을 위한 함수 * ~~void bullet\_position() : 총알 충돌처리를 위한 함수~~ * void game\_start() : 게임이 시작하고 정보들을 초기화 하는 함수 * void send\_login\_ok\_packet() : 클라이언트가 접속하면 접속확인과 id를 보내는 함수 * void send\_other\_info\_packet() : 다른 클라이언트의 접속을 보내는 함수 * void send\_start\_game\_packet() : 게임이 시작했다는 정보를 보내는 함수 * void send\_dead\_packet() : 죽은 플레이어가 있으면 보내는 함수 * void send\_move\_packet() : 클라이언트의 움직임, 총구의 방향을 보내는 함수 * void send\_fire\_packet() : 클라이언트가 대포(총알)을 발사하면 보내는 함수 * ~~void send\_aim\_packet() : 클라이언트의 대포 방향(총구)을 보내는 함수~~ * void send\_hit\_packet() : 클라이언트 또는 벽에 대포(총알)에 맞았을 때 보내는 함수 * void send\_bullet\_packet() : 총알의 위치값을 보내주는 함수 * bool collide() : 클라이언트 이동시 탱크의 충돌(이동가능여부)을 판단하는 함수 * bool collide\_bullet() : 총알 이동시 총알의 충돌을 판단하는 함수 * DWORD WINAPI SendPacket(LPVOID arg) : 클라이언트들의 좌표와 총알의 좌표를 지속적(1초에 약 60번) 보내주는 함수 |

# 팀원 별 역할 분담

|  |  |  |
| --- | --- | --- |
| 양원석 | 김기윤 | 박해성 |
| * (서버)패킷 재조립 구현 – ProcessPacket() * (서버) 플레이 관련 전송 - send\_dead\_packet(), send\_move\_packet(), send\_fire\_packet(), ~~send\_aim\_packet()~~, send\_hit\_packet() * 서버 스레드간 동기화~~(mutex)~~   (Event)   * (클라이언트) 맵 구현 * (서버) 플레이어 충돌처리 | * (서버) 클라이언트 연결 – main() * (서버) 쓰레드 생성 – ProcessClient() * (서버) 게임시작과 접속 관련 전송 – GameStart(),   send\_login\_ok\_packet(), send\_other\_info\_packet(), send\_start\_game\_packet()   * (서버)총알 충돌 처리 – ~~Bullet\_Position()~~ * 서버스레드간동기화~~(mutex)~~   (Event)   * (클라이언트) ui * (클라이언트) Process\_packet() | * (클라이언트) 메인 함수 세팅 – winMain() * (클라이언트)플레이어 및 장애물 구현   – Player Class, Wall Class   * (클라이언트)총알 구현   - Bullet Class   * (클라이언트)네트워크 연결 및 전송 함수 - ~~process\_packet(),~~ send\_move\_packet(), send\_aim\_packet(), send\_attack\_packet() |

# 개발환경

* OS : Windows 10
* IDE : Visual Studio 2019
* API : Win32 API / Windows Socket API
* 네트워크 IO모델 : 다중 쓰레드 모델
* 언어 : C / C++

# 개발일정

* 일별 개발 일정

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **박해성** | | **양원석** | | **김기윤** |
| 10/28(목) | 전체 기획 회의 | | | | |
| 10/29(금) | 기획서 작성 및 역할분담, 계획수립 | | | | |
| 10/30(토) |  | |  | |  |
| 10/31(일) | 기획서 최종 점검 | | | | |
| 11/1(월) | 기획서 제출 | | | | |
| 11/2(화) | 기획서 검수 -> 다시 기획(게임 결정) | | | | |
| 11/3(수) | ~~모든 개발 일정 취소(하고자 했던 일정을 뒤엎으므로 개발이라 칭함)~~  High\_Level\_Design 재구성 | | | | |
| 11/4(목) | Low\_Level\_Design 재구성 | | | | |
| 11/5(금) |  | |  | |  |
| 11/6(토) |  | |  | |  |
| 11/7(일) |  | |  | |  |
| 11/8(월) | 개발 일정 재수립 | | | | |
| 11/9(화) | 기획서 재검수 및 검수를 바탕으로 한 기획 회의 | | | | |
| 11/10(수) | 기획 보충 | | | | |
| 11/11(목) |  | |  | |  |
| 11/12(금) | winMain() 작성 | |  | |  |
| 11/13(토) | Player Class 구현 | |  | | 서버 main()작성  클라이언트 접속 시  쓰레드 생성(ProcessClient()) |
| 11/14(일) | Wall Class 구현 | |  | | 전송 패킷 함수  (send\_login\_ok\_packet(),  send\_other\_info\_packet(),  Send\_start\_game\_packet()) |
| 11/15(월) |  | | 서버 플레이어 객체 구현  서버send\_move\_packet()  구현 | |  |
| 11/16(화) | Bullet Class 구현  클라이언트제작  (네트워크X) | | Recv한 데이터를 처리하는  ProcessPacket()  구현 | | 클라 패킷 수신 처리  (process\_packet()) |
| 11/17(수) | ~~서버에서 보내주는~~  ~~패킷 처리~~  ~~(process\_packet())~~  send\_move\_packet() | | 3명 접속 후  게임 시작 처리 (gamestart()) |
| 11/18(목) | (중간 점검1) | | | | |
| 11/19(금) | 이동확인 | | ~~클라이언트~~  ~~맵 구현~~ | | ~~클라이언트 UI 제작~~  서버와 클라이언트  연결 확인 |
| 11/20(토) | send\_attack\_packet() | | 서버 send\_dead\_packet()구현 | | ~~충돌 처리(Bullet\_Position())~~  서버 총알 객체 구현 |
| 11/21(일) | send\_aim\_packet() | | ~~서버 send\_fire\_packet()~~  ~~구현~~  서버 send\_hit\_packet()  구현 | |
| 11/22(월) |  | |  | |  |
| 11/23(화) | 서버 총알 오류 수정  (방향) | | 리소스 수집 및 구현 | | 클라이언트 UI 제작  총알 위치 수신 및 렌더링 |
| 11/24(수) |  | |  | | 충돌처리(주체 : 총알) |
| 11/25(목) | (중간 점검2 - Remote시도) | | | | |
| 11/26(금) |  | 충돌처리(주체 : 플레이어)  ~~동기화~~ | | ~~동기화~~ | |
| 11/27(토) |  |
| 11/28(일) | ~~오류수정~~ | | | | |
| 11/29(월) |  | | | | |
| 11/30(화) |  | | | | |
| 12/1(수) |  | | 동기화(이벤트 처리) | | |
| 12/2(목) |  | | | | |
| 12/3(금) |  | | | | |
| 12/4(토) |  | | | | |
| 12/5(일) | Remote시도 및 오류 수정 | | | | |
| 12/6(월) |  | | | | |
| ~~12/1 ~ 12/7 QA작업 및 최종 점검~~  12/7 ~12/13 : QA작업과 오류수정, 최종 Report작성  **12 / 14일 최종 검수** | | | | | |