네트워크 게임 프로그래밍

Project Progress Report

Term Project - Agrio

2017180001 고선민
2017180009 남주영
2017180004 김재원

Agrio

목차

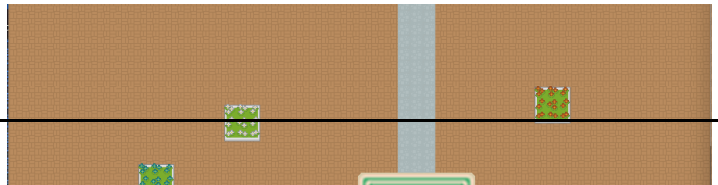
애플리케이션 기획

1. 게임 소개

Agrio	
장르	서바이벌 슈팅 게임
최대 인원	3명
플레이 시간	5분~8분
조작	키보드, 마우스
시점	탑뷰, 2D
개발언어	C++, 윈도우API
플랫폼	윈도우

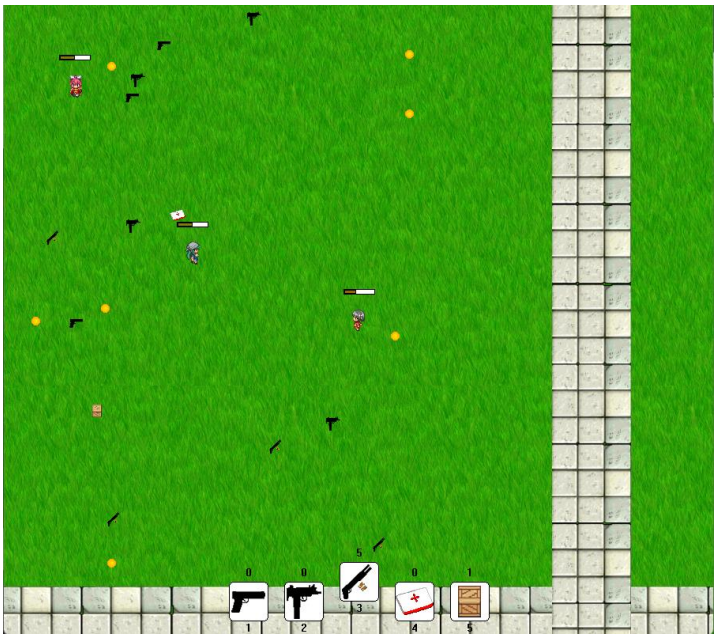
Agrio는 서바이벌 슈팅 멀티플레이 게임입니다.
3명의 플레이어는 맵에서 최후의 1인이 남을 때까지 싸워서 살아남는 게임입니다.

위 스크린샷(좌)은 개발중인 클라이언트로 추후 PVP와 다양한 장애물, 좁아지는 맵 등을 추가할 예정입니다.
게임의 기본적인 구조는 모바일 게임 브롤스타즈(우측 스크린샷)와 유사합니다.



로비 대기 화면

인게임 화면 1

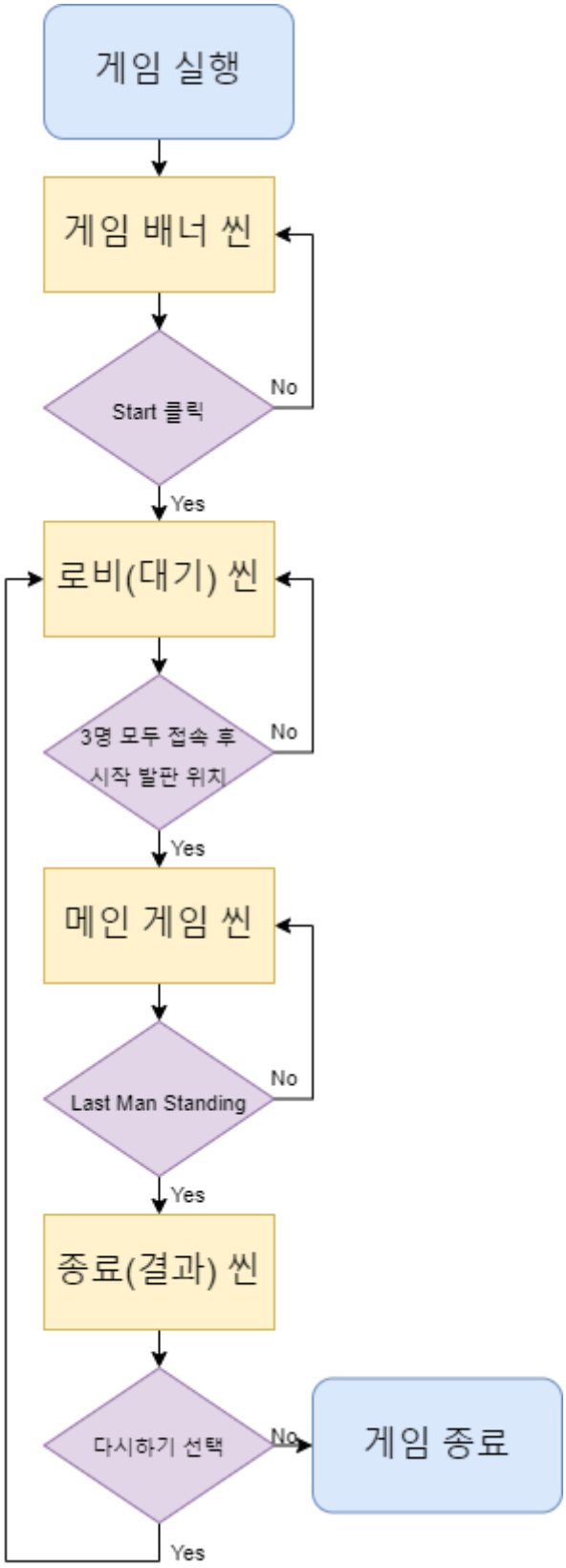


인게임 화면 2

2. 게임 플레이

- 게임을 실행하고 배너를 거쳐 Start 버튼을 눌러 로비로 입장합니다.

- 로비에선 자유롭게 돌아다닐 수 있습니다.
- 3명 모두 로비 특정영역에 있는 시작 발판위로 이동한다면 게임이 시작 됩니다.
- 메인 게임 시작 시 정해진 위치에서 시작합니다.
- 플레이어들끼리 총을 쏘아 맞춰 상대방의 HP를 0이하로 만듭니다.
- ~~플레이어가 한 명씩 탈락할 때 마다~~
  -> 게임 시작부터 조금씩 플레이할 수 있는 맵이 작아집니다.
- 게임 중간중간 사용할 수 있는 아이템(총, 회복 물약, 박스)이 맵 어딘가 스폰 됩니다.
- 최후의 한 명이 남으면 메인 게임을 중단하고 ~~결과창을 보여줍니다~~.
  -> 승자 화면을 보여줍니다.
- 다시하기를 선택하면 로비로 돌아가 대기 합니다.
- 게임종료를 선택하면 그대로 종료합니다.



3. 조작 방법

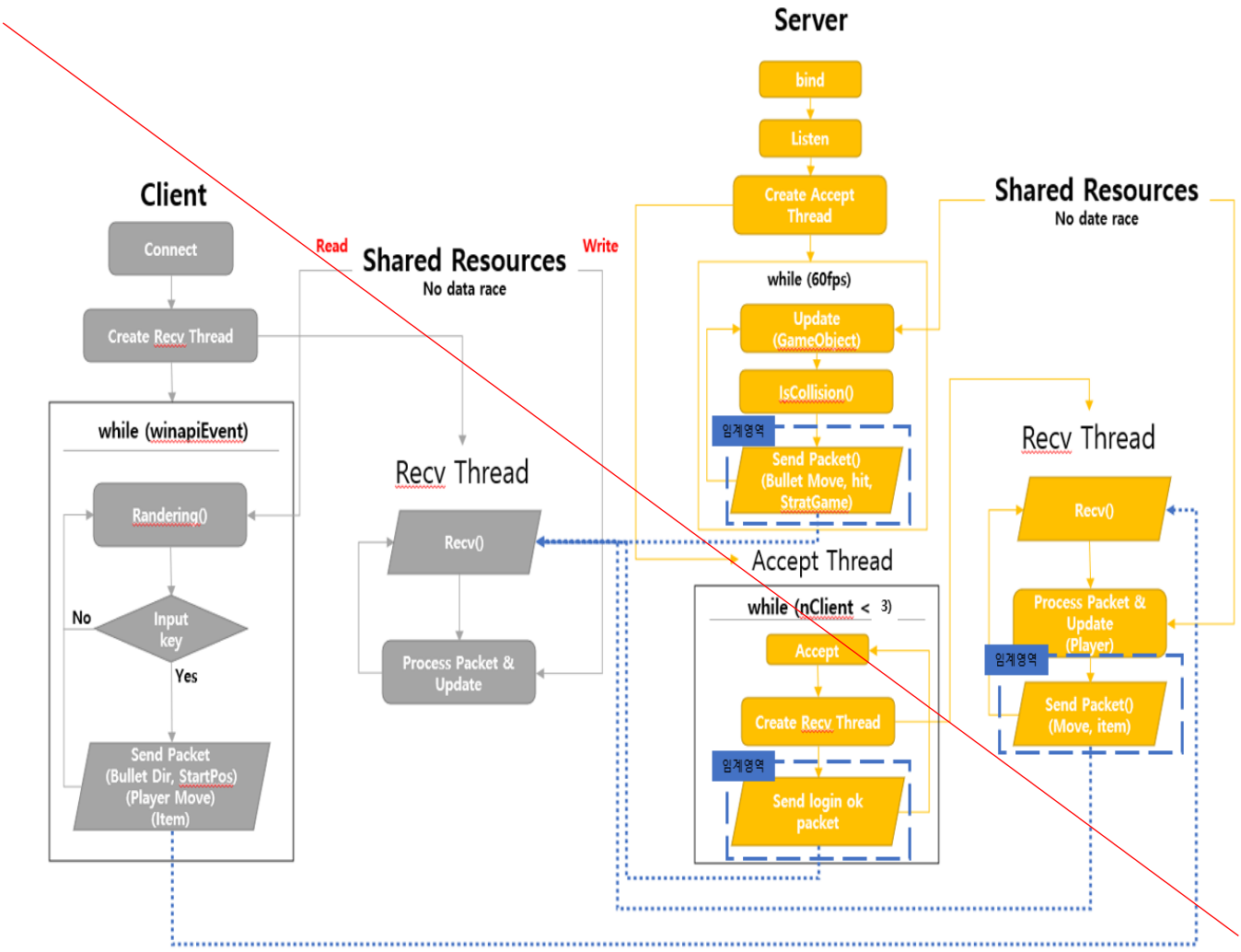
사격 - 스페이스바

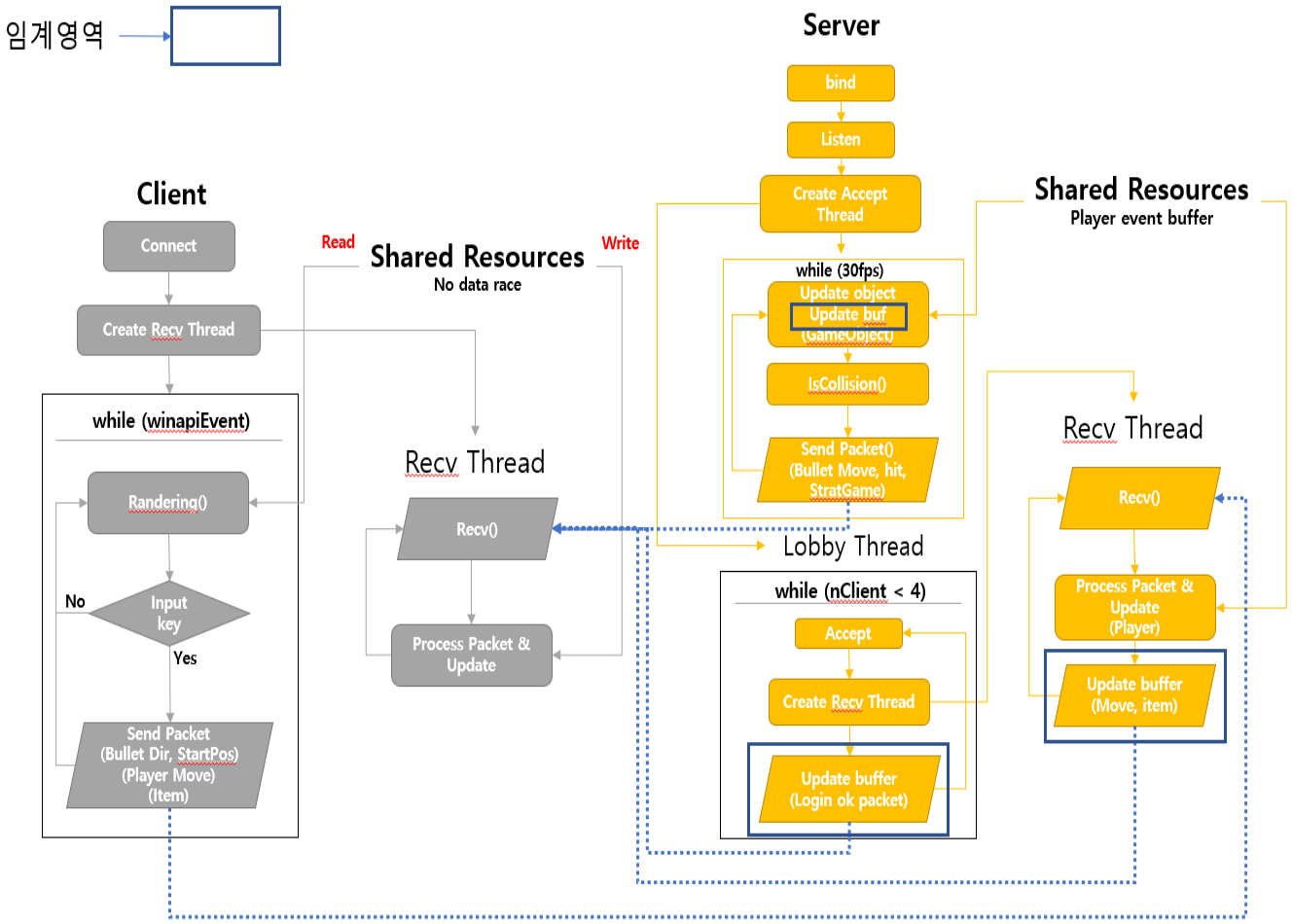
이동 - 방향키

아이템 사용 - 숫자 1~5

클릭 - 마우스

High Level 디자인





- 플레이어의 접속을 계속 기다리면서 먼저 접속한 플레이어들끼리 로비에서 움직이며 대기할 수 있도록 Accept 스레드를 사용합니다.
- 첫번째 로그인시 0번 클라이언트가 Create RecvThread 생성이후에 Login패킷을 보내면 해당 오브젝트에 대한 업데이트를 시작하고 이후 접속하는 클라이언트들은 PutObj로 생성됩니다.
- 파란색 점선은 기본적인 데이터의 흐름입니다.

Low Level 디자인

1. 코딩 규칙(Coding Convention)

Class - 첫번째 문자가 대문자(파스칼 표기법)

Method - 첫번째 문자가 대문자(파스칼 표기법), 동사+명사

Variable & Parameter - 첫번째 문자가 소문자(카멜 표기법)

전역변수 - 모든 문자가 대문자

2. 프로토콜 (Protocol)

- **서버 주소**

  const short SEVER_PORT = 4000;

  const char SEVER_ADDR = "127.0.0.1";

- **최대 접속 가능 클라이언트**

  const char MAX_USER = 3;

- **최대 오브젝트 개수**

  const int MAX_OBJECT = 200;

- **캐릭터의 상태**

enum class STATE { idle, move, attack };

- **총알 발사 방향**

enum class DIR { N, NE, E, SE, S, SW, W, NW};

- **게임의 씬**

enum class SCENE { title, lobby, stage1, gameover, winner };

- **아이템 ID**

enum ITEM { empty, pistol, uzi, shotgun, potion, box };

- **오브젝트 타입**

enum OBJ_TYPE { PLAYER, BOX, BULLET, ITEM, WALL };

- **패킷 타입 (Client -> Server)**

  const char CS_PACKET_LOGIN = 1;

  const char CS_PACKET_PLAYER_MOVE = 2;

  const char CS_PACKET_PLAYER_STATE = 3;

  const char CS_PACKET_SHOOT_BULLET = 4;

  const char CS_PACKET_USED_ITEM = 5;

- **패킷 타입 (Server -> Client)**

```
const char SC_PACKET_LOGIN_OK          = 1;
const char SC_PACKET_CHANGE_SCENE      = 2;
~~const char SC_PACKET_OBJ_MOVE          = 3;~~
const char SC_PACKET_MOVE_OBJ          = 3;
const char SC_PACKET_PLAYER_STATE      = 4;
const char SC_PACKET_PUT_OBJ           = 5;
const char SC_PACKET_REMOVE_OBJ        = 6;
const char SC_PACKET_CHANGE_HP         = 7;
const char SC_PACKET_GET_ITEM          = 8;
~~const char SC_PACKET_ITEM_COUNT        = 9;~~
const char SC_PACKET_CHAGE_WEAPON      = 10;
```

- **패킷 정의(Client -> Server)**

```
struct packet{
        unsigned char packetSize;
        char packetType;
}
struct cs_packet_login : packet{
        char playerSkin;
}
struct cs_packet_player_move: packet{
        char dir;
}
struct cs_packet_player_state: packet{
        char playerState;
}
struct cs_packet_shoot_bullet: packet{
        ~~char shootX, shootY;~~
        ~~char dir;~~
        char playerID;
}
```

```
struct cs_packet_used_item: packet{
        char itemNum;
}
```

- **패킷 정의(Server -> Client)**

```
struct sc_packet_login_ok: packet{
        char playerID;
        char x, y;
        short x, y;
        short width, height;
}
struct sc_packet_change_scene: packet{
        char sceneNum;
}
struct sc_packet_move_obj: packet{
        char objectID;
        char lookDir;
        char x, y;
        short x, y;
}
struct sc_packet_player_state: packet{
        char objectID;
        char playerState;
}
struct sc_packet_put_obj : packet{
        char objectID;
        char sprite;
        char x, y;
        short x, y;
        unsigned char width, height;
}
struct sc_packet_remove_obj : packet{
        char objectID;
}
```

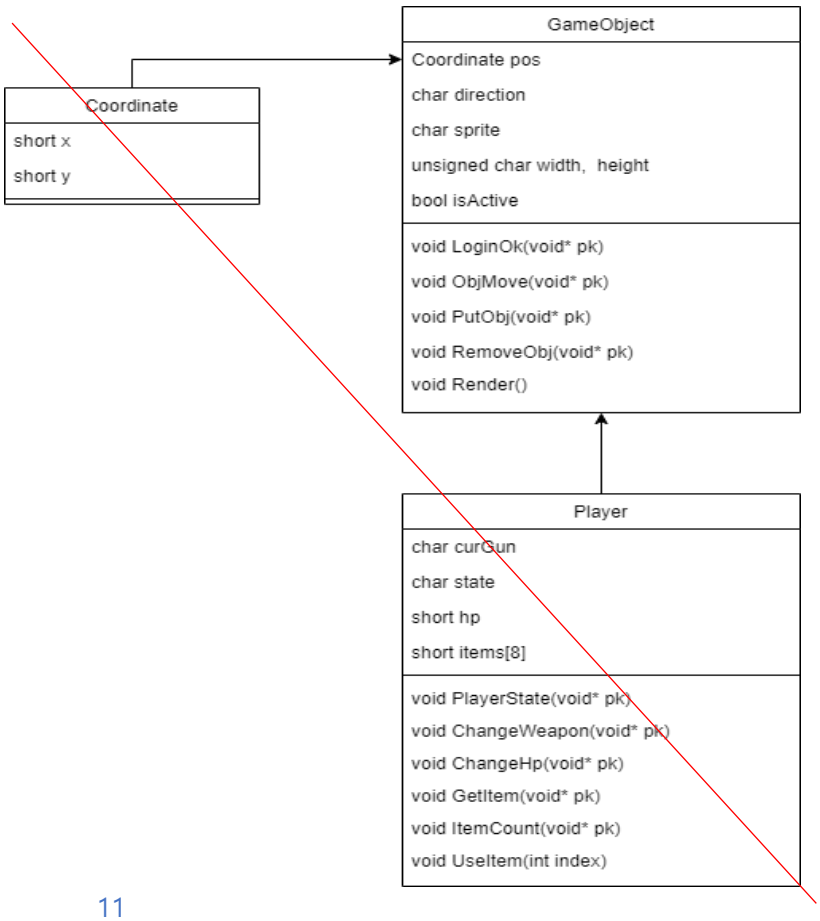
```
struct sc_packet_change_hp : packet{
        char playerID;
        unsigned char hp;
}
struct sc_packet_get_item : packet{
        char playerID;
        char itemID;
        char itemCount;
}

struct sc_packet_item_count : packet{
        char playerID;
        char itemID;
        char itemCount;
}
struct sc_packet_change_weapon : packet{
        char playerID;
        char gunID; // == itemID
}
```

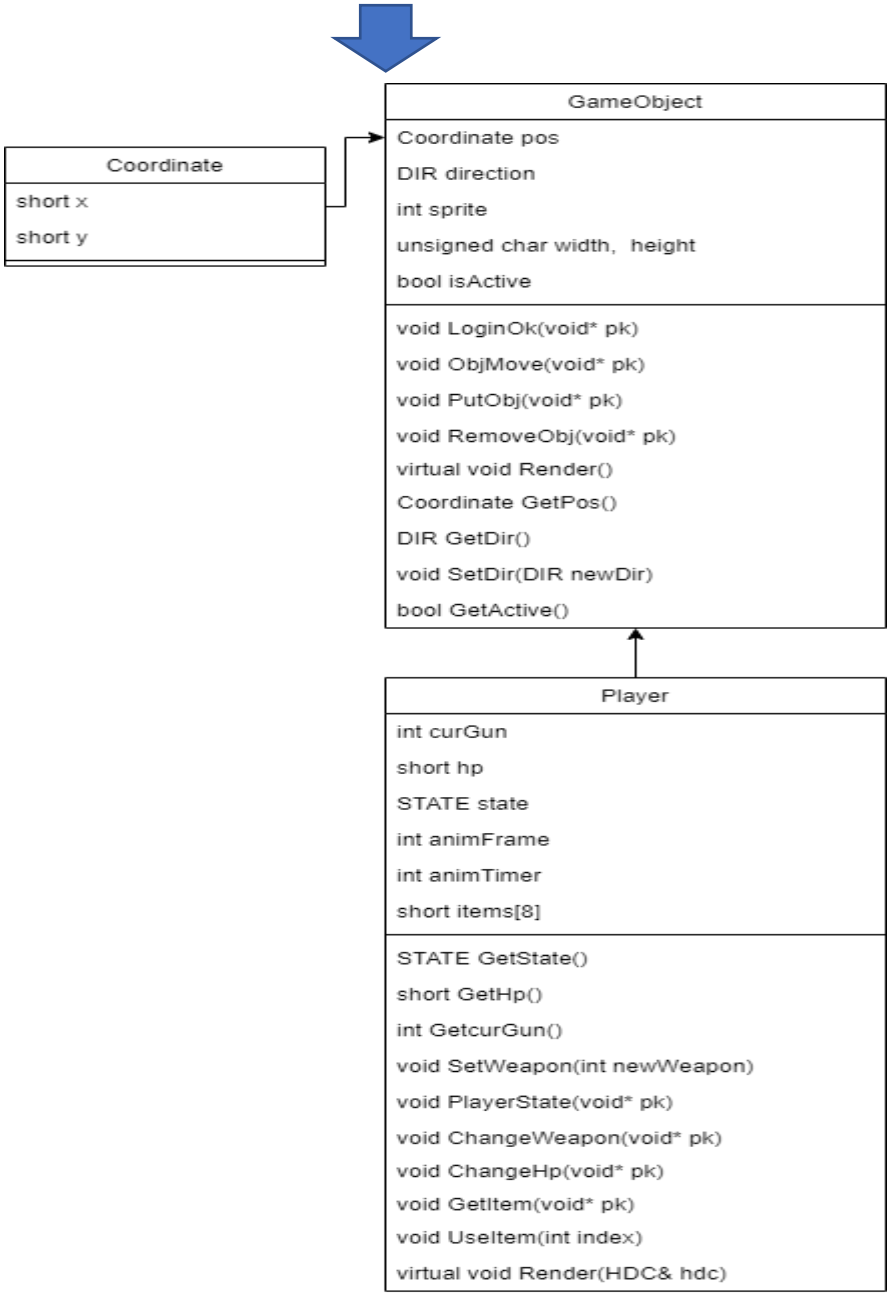
3. 클라이언트 (Client)

Class

변경 전



변경 후



Variable

- **vector<GameObject*> gameObjects;**
- **vector<CImage> sprites;**

Method

- ~~DWORD WINAPI Recv(LPVOID sock)~~ ->

  DWORD WINAPI ProcessClient (LPVOID arg)

  recv를 호출해 packet이 오기를 기다리다 패킷이 오면 앞부분(2바이트) packetSize, packetType을 확인한 후 packetSize – 2 만큼 다시 recv를 받고 packetType에 맞는 행동을 합니다.

  Ex) packType이 SC_PAKCET_PUT_OBJ이면 ObjectID 위치에 있는 오브젝트에 SetActive(), SetPosition(), SetSprite(), SetSize()를 호출해 렌더링 될 수 있도록 한다.

- ~~void Rendering() ->~~

  void GameObject::Render(HDC& hdc) & void Player::Render(HDC& hdc)

  ~~gameObjects의 render를 호출합니다. ->~~ gameObject와 player를 draw합니다.

- ~~Send Packet~~ **-> void Send(void* Packet)**

  입력된 키에 따라 다른 패킷을 서버로 전송합니다.

- **Void GameObject::LoginOk (void* pk)**

  LOGIN_OK 패킷이 오면 호출할 함수입니다. 해당 플레이어 오브젝트를 활성 상태로 바꾸고 시작 위치를 정해줍니다.

- **Void GameObject::ObjMove (void* pk)**

  OBJ_MOVE 패킷이 오면 호출할 함수입니다. 해당 오브젝트의 lookDir, x, y를 수정해줍니다.

- ~~Void GameObject::PlayerState (void* pk)~~

  -> void Player::PlayerState(void* pk)

  PLAYER_STATE 패킷이 오면 호출할 함수입니다. 해당 플레이어의 상태 (이동 중, 공격 중)를 바꿔주어서 적절한 sprite를 사용할 수 있게 해줍니다.

- **Void GameObject::PutObj (void* pk)**

  PUT_OBJ 패킷이 오면 호출할 함수입니다. 해당 오브젝트를 활성화하고 sprite, width, height, x, y를 정해줍니다.

- **Void GameObject::RemoveObj (void* pk)**

  REMOVE_OBJ 패킷이 오면 호출할 함수입니다. 해당 오브젝트를 비활성화 상태로 바꾸어서 화면에 출력하지 않도록 합니다.

- ~~Void GameObject::Render()~~ **// 위에 추가**

  ~~해당 오브젝트를 화면에 출력합니다.~~

- **Void Player::ChangeHp(void* pk)**

  CHANGE_HP 패킷이 오면 호출할 함수입니다. 해당 플레이어의 hp값을 수정합니다.

- **Void Player::GetItem(void* pk)**

  GET_ITEM 패킷이 오면 호출할 함수입니다. ~~플레이어의 인벤토리에서 아이템을 활성화해서 사용할 수 있도록 합니다~~. -> 인벤토리의 아이템 개수를 변경합니다.

- ~~**Void Player::ItemCount(void* pk)**~~

  ~~ITEM_COUNT 패킷이 오면 호출할 함수입니다. 플레이어가 가지고 있는 인벤토리 내의 아이템 보유 수량을 수정해줍니다.~~

- **Void Player::ChangeWeapon(void* pk)**

  CHANGE_WEAPON 패킷이 오면 호출할 함수입니다. 해당 플레이어가 들고 있는 무기를 바꿔줍니다.

- **Void Player::UseItem(int index)**

  ~~플레이어가 들고 있는 총을 발사 혹은 아이템을 사용합니다. 들고 있는 아이템이 총일 경우 cs_packet_shoot_bullet을, 그 외의 아이템은 cs_packet_used_item 패킷을 구성해서 서버에 Send하여 줍니다.~~아이템을 사용하면 인벤토리의 해당 아이템 개수를 줄입니다

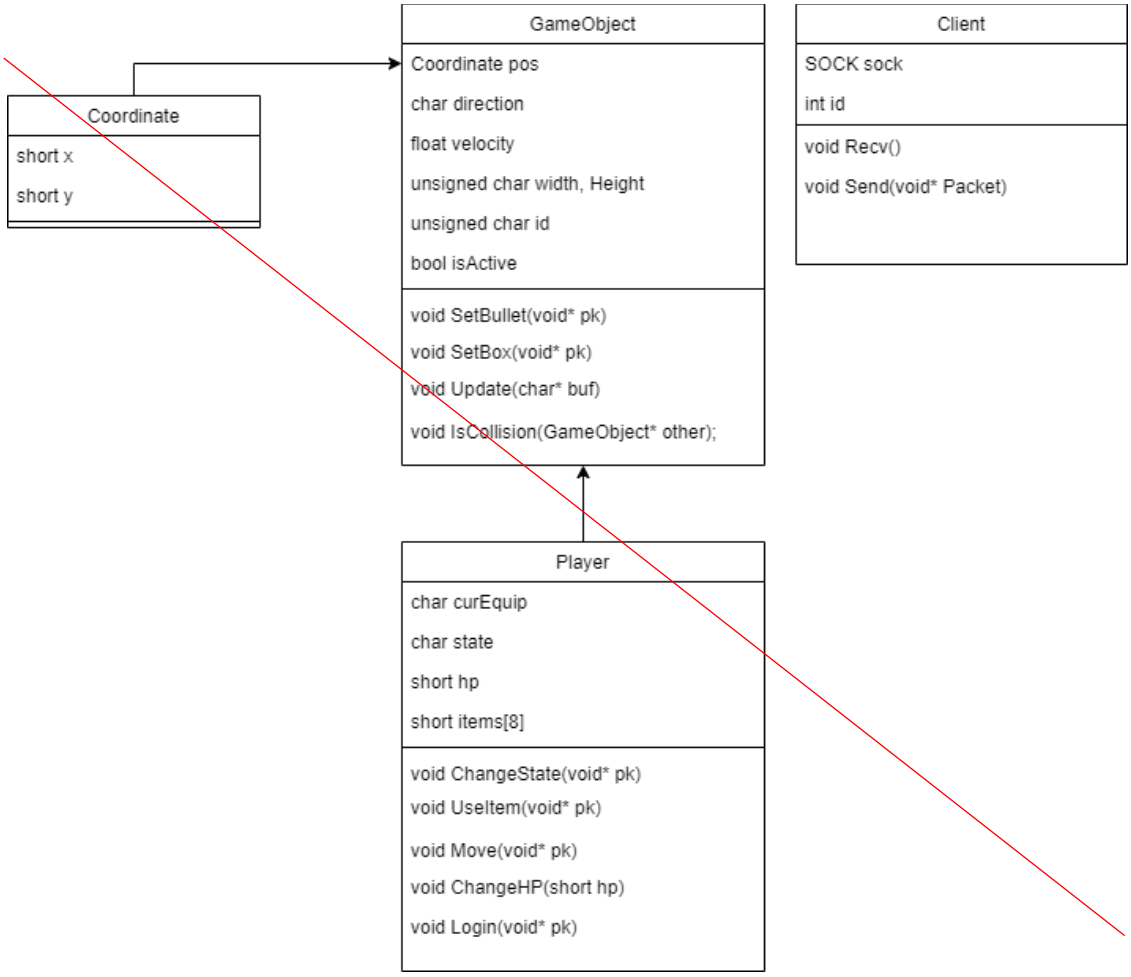
- ~~**Void ChangeScene(void* pk)**~~

  ~~CHANGE_SCENE 패킷이 오면 호출할 함수입니다. 게임을 해당 장면으로 전환합니다.~~

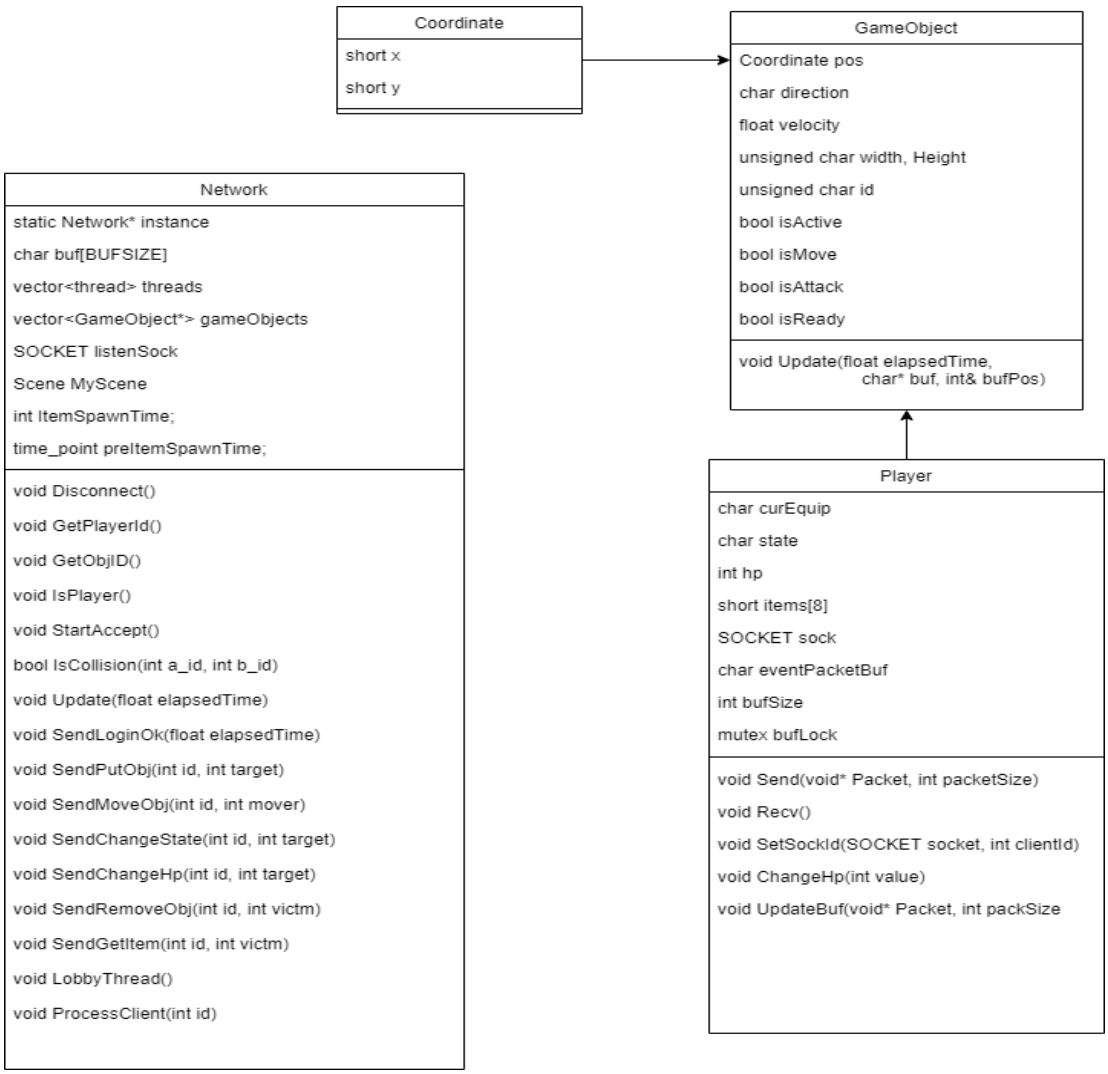
4. 서버 (Server)

Class

변경 전



변경 후



Variable

- ~~vector<GameObject*> GameObjects;~~
- **Network net;**

Method

- ~~void GameObject::SetBullet(void *pk)~~

  -> void Network::SendPutObj(int sender, int target)

  ~~SHOOT_BULLET 패킷이 오면 호출할 함수입니다. 비활성 상태인 오브젝트 중 하나를 활성 상태로 바꾸고 시작 위치 속도 방향을 정한 뒤 sc_packet_put_obj 패킷을 만들어 모든 클라이언트에게 Send하여 줍니다. 이때 클라이언트가 사격키를 연타하여도 최대 공격속도를 넘지 않도록 제한하는 역할도 합니다.~~

● ~~void GameObject::SetBox(void *pk)~~

-> **void Network::SendPutObj(int sender, int target)**

~~USED_ITEM 패킷이 오면 호출할 함수입니다. 비활성 상태인 오브젝트 중 하나를 활성 상태로 바꾸고 위치 방향을 정한 뒤 sc_packet_put_obj 패킷을 만들어 모든 클라이언트에게 Send하여 줍니다.~~

● **void GameObject::Update(~~char* buf~~ >float elapsedTime,**

char* buf, int& bufStart)

활성 상태이면 자신이 가지고 있는 속도와 방향을 이용해 위치를 갱신한 후 충돌 검사를 하고 해당 정보에 따른 알맞은 패킷을 만들어 패킷을 buf에 복사합니다.

● ~~void Player::ChangeState(void* pk)~~

-> **void Network::SendChangeState(int id, int target)**

~~PLAYER_STATE 패킷에 대한 호출 함수입니다. Player.state를 변경한 뒤 sc_packet_player_state 패킷을 만들어 모든 클라이언트에게 Send합니다.~~

● ~~void Player::UseItem(void* pk)~~

-> **void Network::SendChangeState(int id, int target)**

~~USED_ITEM 패킷에 대한 호출 함수입니다. 패킷에서 아이템에 대한 정보를 받아 회복물약이면 sc_packet_change_hp 패킷을 만들어 모든 클라이언트에게 Send | 박스 아이템이면 sc_packet_put_obj 패킷을 만들어 모든 클라이언트에게 Send | 총이면 총 상태를 바꿔주고 sc_packet_change_weapon 패킷을 만들어 모든 클라이언트에게 Send 그 뒤 sc_packet_item_count 패킷을 만들어 사용한 클라이언트에게 Send합니다.~~

● ~~void Player::Move(void* pk)~~

-> **void Network::SendMoveObj(int id, int mover)**

~~PLAYER_MOVE 패킷이 오면 방향을 갱신해 주고 최대 이동속도에 맞춰 좌표를 모든 클라이언트에게 Send합니다.~~

● ~~void Player::ChangeHP(short hp)~~

-> **void Network::SendChangeHp(int id, int mover)**

~~CHANGE_HP 패킷이 오면 해당 플레이어의 체력을 바꾸고 sc_packet_change_hp 패킷을 만들어 모든 클라이언트에게 Send합니다.~~

● ~~void Player::Login(void* pk)~~

-> **void Network::SendLoginOk(int id, int mover)**

~~LOGIN 패킷이 오면 0~2번 중 비활성 상태인 플레이어를 활성상태로 만들고 sc_packet_login_ok 패킷을 만들어 해당 클라이언트에게 보냅니다.~~

~~sc_packet_put_obj 패킷을 만들어 해당 클라이언트에게 맵의 현재 오브젝트 상태를 Send합니다.~~

~~sc_packet_put_obj 패킷을 만들어 다른 클라이언트에게 새로운 플레이어가 생성되었다~~

~~고 알려줍니다.~~

- ~~void GameObject::IsCollision (GameObject* objects)~~

  -> **bool Network::IsCollisoin(int a_id, int b_id)**

  해당 아이디의 오브젝트의 충돌 여부를 반환합니다.

- **void Network::Send*(int receiver, int target)**

  *의 이름에 맞는 target에 대한 정보를 가진 패킷을 만들어 receiver의 UpdateBuf를 호출합니다.

- ~~void Client::Send (void* Packet)~~ -> **void Player::Send(void* buf, int bufSize)**

  버퍼를 받아서 char*로 변환 후 bufSize크기 만큼 eventbuf를 Send 합니다.

  메인 스레드에서만 호출됩니다.

- **void Player::UpdateBuf(void* buf, int bufSize)**

  버퍼를 받아 bufSize크기 만큼 eventBuf에 복사합니다.

- **void Player:: Send(void* buf, int bufSize)**

  버퍼를 받아 플레이어에게 전송합니다.

팀원 별 역할분담

Client - Login 부분	고선민
Client - Rendering 부분	고선민
Client - Send 부분	고선민
Client - Recv 부분	고선민
Client - Input Key 부분	고선민
Client - Process Packet 부분	김재원
Sever – Send 부분	김재원
Sever - Process Packet 부분	김재원
Sever - Update 부분	김재원
Sever - Lobby 부분	김재원
Sever - Login 부분	남주영
Sever - Recv Thread 부분	남주영
Sever - Accept Thread 부분	남주영
Sever - IsCollision 부분	남주영
문서 작성	김재원
일정 관리	고선민
버전 관리	남주영
리소스	고선민

개발 환경

운영체제

- Windows 10

개발 도구

- Visual Studio 2019
- 포토샵CS6
- draw.io
- Word

VCS

- Github

커뮤니케이션

- Discord
- KakaoTalk

사용 언어, 라이브러리

- **C++**
- WinAPI

개발 일정

	월	화	수	목	금	토
	11월 1일	2일	3일	4일	5일	6일
선민	클라이언트 수정		~~클라이언트 Bullet 구현~~		~~클라이언트 Send(), Recv() 제작~~	
주영			프로토콜 작성	~~서버 Recv() 스레드 제작~~	~~Updata()구현~~	~~Accept()스레드 구현~~
재원				~~서버 Send() 제작~~	~~서버 ProcessPacket() 구현~~	~~서버Recv() 스레드 수정~~
리뷰	클라이언트 지속적인 수정 필요		~~계획서와 달라진 프로토콜 수정~~	~~시험 통신 필요~~	~~패킷주고받기 성공~~	
	8일	9일	10일	11일	12일	13일
선민	~~Login() 구현~~	~~ChangeHP() 구현~~	~~ObjMove() 구현~~	~~Render() 수정~~		1주간 미비사항 추가, 수정
주영	~~PutObj() 구현~~	~~LoginOk() 구현~~	~~Move() 구현~~	~~SetBullet() 구현~~		
재원	~~로비 스테이지 구현~~		~~RemoveObj()구현~~	~~ChangeState() 구현~~	~~Update(), Send(), Recv() 수정~~	
리뷰	~~로그인 recv 수정 필요~~	~~ChangeHP() 11월 22일 구현~~	~~Move()는 작동 하나 애니메이션 랜더링 수정 필요~~	~~Bullet 8방향으로 리소스 업데이트~~	~~서버 Network 클래스 추가~~	
	15일	16일	17일	18일	19일	20일
선민	~~클라이언트 오브젝트 추가 구현~~	~~오브젝트 활성화~~	~~UseItem() 구현~~	~~ChangeWeapon() 구현 -> 미구현 총알 수정~~	~~여러 총기 구현~~	1주간 미비사항 추가, 수정
주영	~~Update(), Send(), Recv() 수정~~		~~SetBox() 구현~~	~~아이템 상호작용 디버그~~	아이템 보급 구현 ~~-> 아이템 선택창 구현~~	
재원	~~SetBullet() 구현~~		~~클라 서버 Timing 조정~~	~~Item구현~~	~~Hp Bar 구현~~	
리뷰	Bullet() 애니메이션 수정	~~상자 설치 완료~~	~~Move() 애니메이션 수정완료~~	~~구급상자 리소스 추가 및 탄환 리소스 수정~~	~~아이템 보급은 스테이지 시작 후 구현으로 변경~~	
	22일	23일	24일	25일	26일	27일
선민	~~게임 종료 구현~~	~~리더보드구현-> 종료화면 대체~~	~~씬 변환 수정~~	~~충돌 후 클라이언트 Process 수정~~	~~(18일에서) ChangeWeapon() 구현 클라이언트 최적화~~	기말고사 준비
주영	~~GetItem() 구현~~	~~IsCollision()구현~~		~~충돌 후 서버 Process 구현~~	~~서버 최적화~~	
재원	~~제한구역 구현~~		~~Stage 변화 구현~~	~~ProcessPacket() 수정~~	~~서버 최적화~~	

리뷰	~~제한구역 리소스 추가 필요~~	~~제한구역을 벽의 움직임으로 변환~~	~~리더보드 스테이지 추가~~ -> 게임 오버, 위너 화면으로 변경	~~30프레임 변환~~	~~애니메이션 최적화~~	
	30일	12월 1일	2일	3일	4일	5일
선민	~~게임 씬 최적화~~	~~추가 기능 구현~~		아이템 밸런스 조정	발표 자료 준비, 일정 진척도 종합	
주영	~~비정상적 네트워크상황 대응 서버 디버그~~		~~추가 기능 구현~~		최종 시연 준비, 깃 허브 버전 관리	
재원	~~비정상적 네트워크 상황 대응 클라이언트 디버그~~		~~추가 기능구현 (REPLAY 후 재시작)~~	~~총알 리소스 변경~~	리포트 작성	
리뷰	~~리모트테스트~~	~~리모트테스트~~	~~Replay시 제대로 초기화 하지 않음~~			

개발 일정 변경사항

	일정 변경 사항	내용
1주차	X	
2주차	X	
3주차	ChangeWeapon() 구현 ⇨ 11월 26일로 이동 총알 수정 추가 아이템 보급 구현 ⇨ 아이템 선택창 구현 Update() 수정 ⇨ Hp Bar 구현	앞서 먼저 구현해야 할 작업이 있어서 26일로 이동했습니다. 총알 오브젝트에 수정이 있었습니다. 아이템을 보급하는 방법에 변경이 있어서 구현 사항이 바뀌었습니다. HP를 표시해주는 작업을 진행하였기에 더 정확히 변경했습니다.
4주차	리더보드구현 ⇨ 종료화면 대체 ChangeWeapon() 구현 ⇨ 11월 18일에서 이동	종료 화면 구성에서 변경이 생겨서 수정을 했습니다. 3주차에서 구현을 미루었던 작업을 완료했습니다.
5주차	추가 기능구현 ⇨ REPLAY 후 재시작 총알 리소스 변경 추가	추가 기능으로 게임 종료 후 재시작하는 기능을 추가했습니다. 총알 리소스를 변경했습니다.