목차

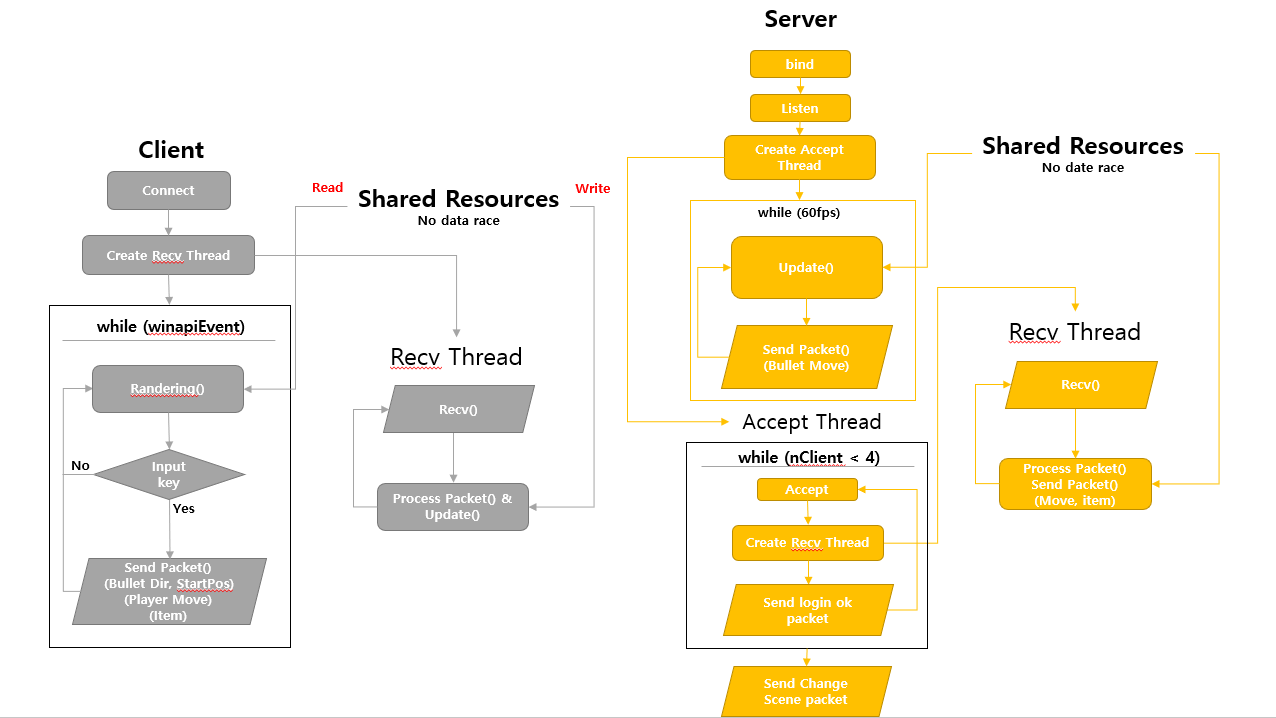
1. 애플리케이션 기획

* 게임 설명

1. High Level Design
2. Low Level Design

* 패킷

**High Level Design**

****

**Low Level Design**

1. **Protocol**

* **서버 주소**

const short SEVER\_PORT = 4000;

const char SEVER\_ADDR = “127.0.0.1”;

* **최대 접속 가능 클라이언트**

const char MAX\_USER = 4;

* **총알 발사 방향**

enum DIR {

UP, DOWN, LEFTUP, RIGHTUP, LEFTDOWN, RIGHTDOWN, LEFT, RIGHT

}

* **아이템 ID**

enum ITEM { // 8개 채우자

pistol, uzi, shotgun, potion, wall

}

* **패킷 타입 (Client -> Server)**

const char CS\_PACKET\_LOGIN = 1;

const char CS\_PACKET\_PLAYER\_MOVE = 2;

const char CS\_PACKET\_PALYER\_SPRITE = 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\_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\_CHAGE\_WEAPON = 9;

* **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;

}

struct cs\_packet\_used\_item : packet{

char itemNum;

}

* **패킷 정의(Server -> Client)**

struct sc\_packet\_login\_ok: packet{

char playerID;

char x, y;

}

struct sc\_packet\_change\_scene: packet{

char sceneNum;

}

struct sc\_packet\_obj\_move: packet{

char objectID;

char lookDir;

char 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;

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;

}

struct sc\_packet\_change\_weapon : packet{

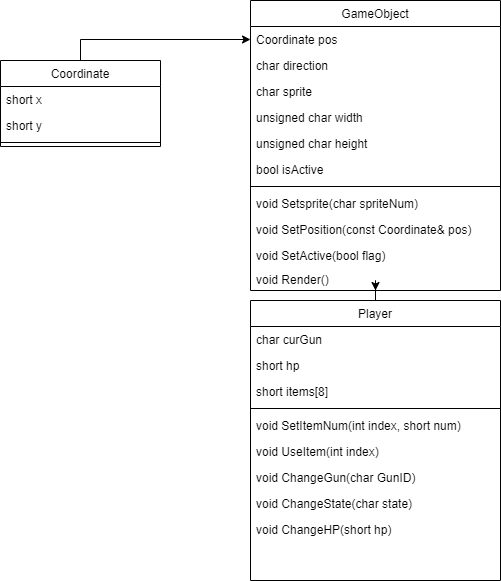
char playerID;

char gunID; // == itemID

}

1. **Client Method**

**Class**

****

**Variable**

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

**Method**

* **DWORD WINAPI Recv(LPVOID sock)**

recv를 호출해 packet이 오기를 기다리다 패킷이 오면 앞 부분(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()**

gameObjecs의 render를 호출합니다.

* **Send Packet**

입력된 키에 따라 다른 패킷을 서버로 전송합니다.

* **void GameObject::SetSprite (char spriteNum)**

Sprite의 값을 spriteNum으로 변경합니다.

* **void GameObject::SetSize (char w, char h)**

width, height의 값을 w, h로 변경합니다.

* **void GameObject::SetPosition(const Coordinate& pos)**

Pos의 값을 pos로 변경합니다.

* **void GameObject::SetActive(bool flag)**

오브젝트를 활성|비활성 상태로 변경합니다.

* **Player::SetItemNum(int index, short num)**

인덱스로 items배열에 접근해 배열 값(아이템 개수)을 수정합니다

* **void GameObject::Render()**

해당 오브젝트가 활성 상태라면 sprite,curGun를 인덱스로 리소스 배열에 접근해 해당 리소스를 그립니다.

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

해당 인덱스의 아이템의 개수를 1개 줄이고 해당 되는 행동을 합니다.

* **Player::ChangeGun(char GunID)**

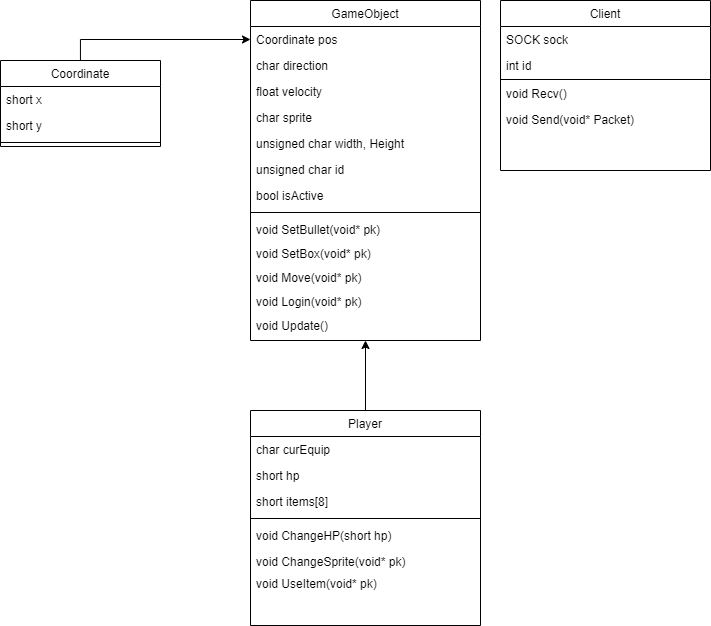
curGun의 값을 GunID로 변경합니다.

* **Player::ChangeState(char state)// state 왜 있는 거지 sprite로 하면 될 거 같은데**
* **Player::ChangeHp(short hp)**

hp값을 변경합니다.

1. **Server**

**Class**

****

**Variable**

* **vector<GameObject> GameObjects;**
* **vector <client> clients;**

**Method**

* **void GameObject::SetBullet(void \*pk)**

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.

* **void GameObject::SetBox(void \*pk)**

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)**

활성 상태이면 자신이 가지고 있는 속도와 방향을 이용해 위치를 갱신한 후 해당 정보를 패킷으로 만들어 패킷을 buf에 복사합니다.

* **void Player::ChangeState(void\* pk)**

PLAYER\_STATE 패킷이 오면 호출할 함수입니다. 플레이어의 상태를 나타내는 변수를 변경한 뒤 sc\_packet\_player\_state 패킷을 만들어 모든 클라이언트에게 Send하여 줍니다.

* **void Player::UseItem(void\* pk)**

USED\_ITEM 패킷이 오면 호출할 함수입니다. 패킷에서 아이템에 대한 정보를 받아 회복 물약이면 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)**

**PLAYER\_MOVE 패킷이 오면 방향을 갱신해 주고 속도에 맞춰**

* **void Player::ChangeHP(short hp)**

**CHANGE\_HP 패킷이 오면 해당 플레이어의 체력을 바꾸고 sc\_packet\_change\_hp패킷을 만들어 모든 클라이언트에게 Send합니다.**

* **void Player::Login(void\* pk)**

**LOGIN 패킷이 오면 0~4번 중 비활성 상태인 플레이어를 활성상태로 만들고 sc\_packet\_login\_ok 패킷을 만들어 해당 클라이언트에게 보냅니다.**

**sc\_packet\_put\_obj 패킷을 만들어 해당 클라이언트에게 맵의 현제 오브젝트 상태를 Send합니다.**

**sc\_packet\_put\_obj 패킷을 만들어 다른 클라이언트에게 새로운 플레이어가 생성되었다고 알려줍니다.**

**충돌 체크**

* **CheckCollision ()**

**플레이어끼리 충돌검사**

**오브젝트끼리 충돌검사, 총알과 다른 오브젝트 충돌 시 SC\_PACKET\_REMOVE\_OBJ 패킷 송신**

**패킷 송신(Send는 패킷의 종류만큼 많아 지나?)**

* **Send (Object\* objects)**

**충돌체크는 매번 플레이어가 움직일 때, 총알이 움직이고 있는 동안 작동해야 한다.**

**총알이 날라가는 중엔 서버가 매번 총알이 1프레임씩 움직일 때마다 보내나?**

**패킷이 조금 과하게 나눠진 것 같은데**

**Coding Convention**

**Class - 첫번째 문자가 대문자(파스칼 표기법)**

**Method - 첫번째 문자가 대문자(파스칼 표기법), 동사+명사**

**Value & Parameter - 첫번째 문자가 소문자(카멜 표기법)**