추진계획서v2

11팀

2018180015 류연우

2018180017 박기정

**목차**

최종 결과물에 따라서 쪽수 다시 입력

1. 애플리케이션 기획 3p

2. High-level 디자인 4p

3. Low-level 디자인

* 1. Class 계층구조 5p
  2. 플레이어 접속 6p
  3. Client Loop 7, 8p
  4. Server Loop 9, 10p
  5. Key 11p
  6. 부록 12 ~ 19p

4. 팀원 별 역할분담 20p

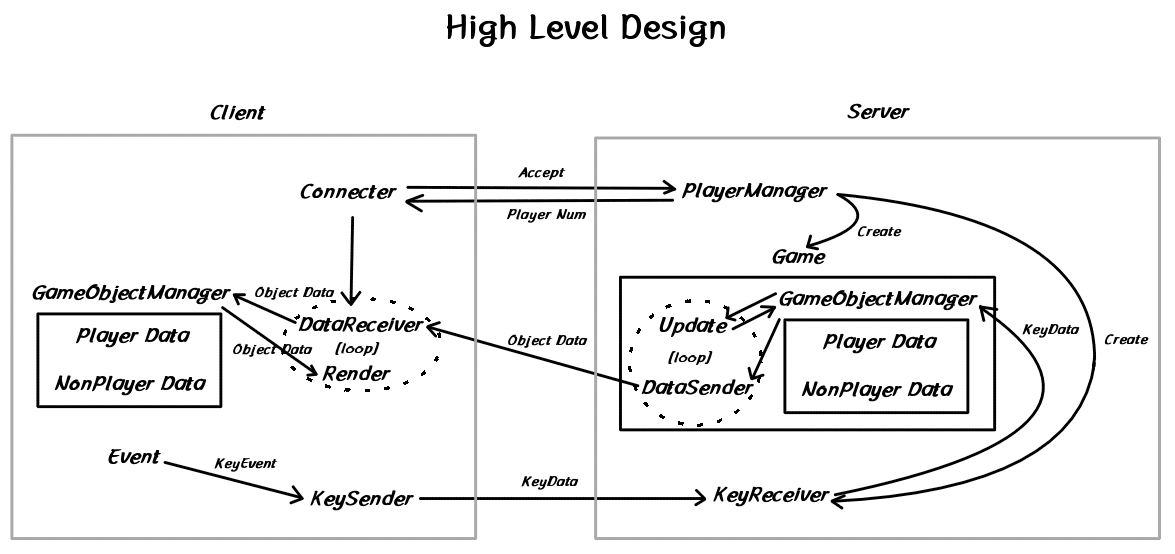
5. 개발환경 21p

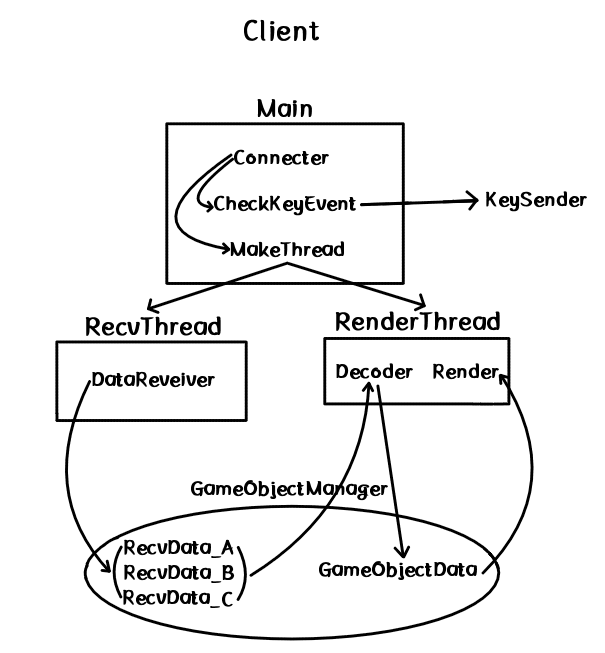
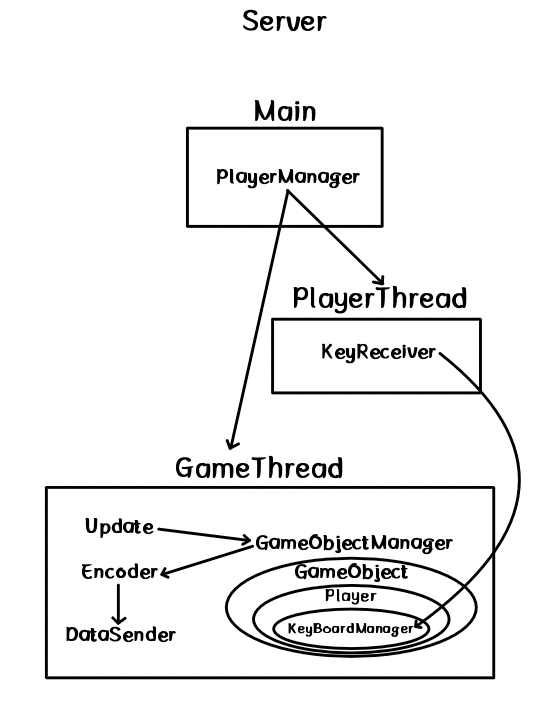
6. 개발일정 21p

**애플리케이션 기획**

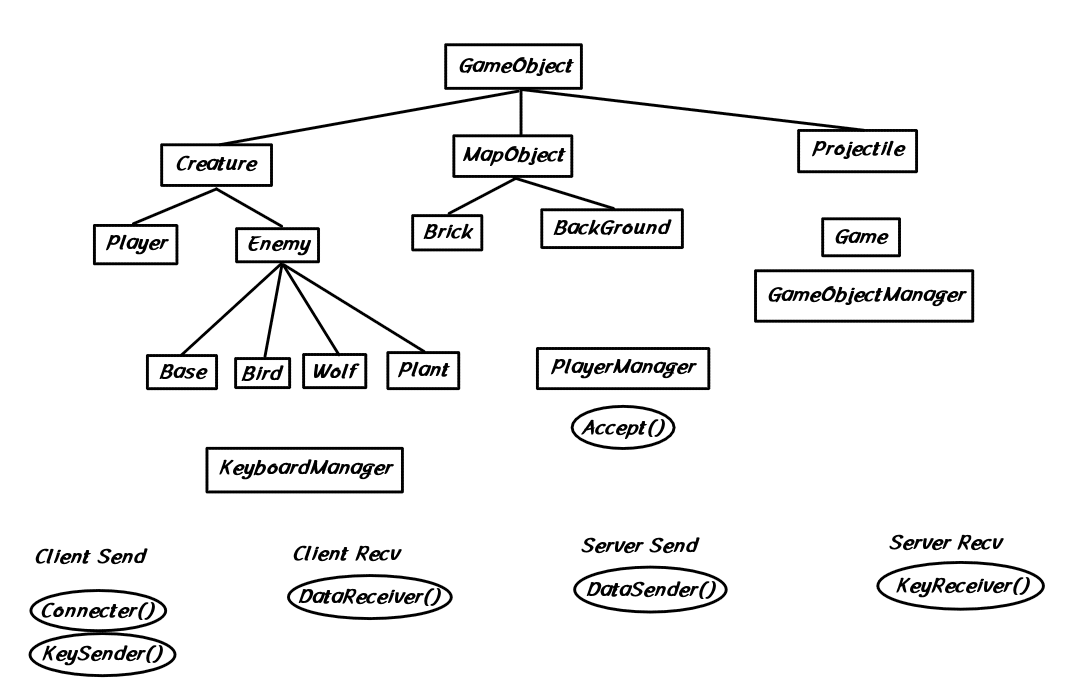
* 박기정이 2학년때 윈도우프로그래밍 수업에서 만든 게임을 사용하여 만드는 온라인 멀티 게임
* 사용 언어 C+
* 사용 API(?) 윈도우 API
* 이미지 .png, .bitmap
* 사운드 .mp3
* 장르 2D횡스크롤 RPG
* 같은 게임을 하고 있는 플레이어간 동기화를 위해 Server에서 업데이트를 하고 출력해야 할 정보를 Client에게 전송한다.
* Server의 Update속도, Client의 Rendering 속도 차이에 따른 문제를 파악하고 예방할 수 있도록 해야 한다.
* 공용자원사용으로 인한 병목현상이 없도록 해야 한다.
* 과연 몇 개까지의 게임을 생성하고 높은 프레임을 유지할 수 있을까?

**High-level 디자인**

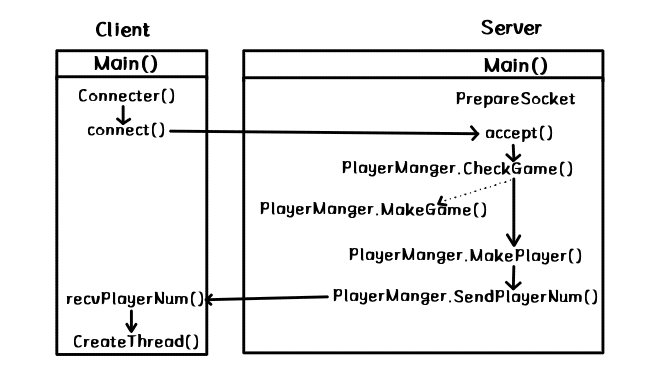




**Low-level 디자인**



게임에서 사용될 오브젝트의 Class 계층구조

**플레이어 접속**

**PrepareSocket**  
윈속 버전 : 2.2  
사용 프로토콜 : IPv4, TCP

**int Connecter(char\*, int)**  
**Return value** : player\_index  
**Parameter** : server\_ip, server\_port  
**Explain** :  
connect()와 RecvPlayerNum()을 통해 서버와 통신하고 CreateThread()를 사용하여 게임 플레이를 위한 쓰레드를 생성한다.

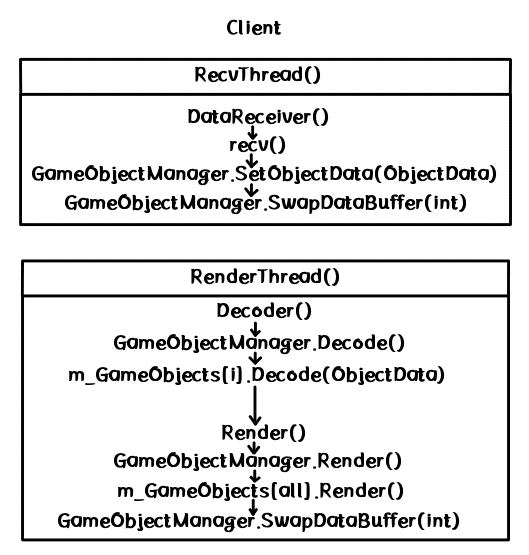
**int CheckGame()  
Return value** : game\_index  
**Parameter** : -  
**Explain** :  
PlayerManger의 게임들이 플레이어를 추가할 수 있는지 확인하고 추가한다. 불가능할 경우 MakeGame()을 부른다.

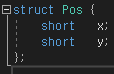
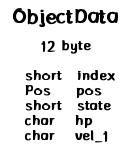
**int MakeGame()  
Return value** : game\_index  
**Parameter** : -  
**Explain** :  
PlayerManger의 게임배열에 새로운 게임을 추가한다.

**int MakePlayer(int)  
Return value** : player\_index  
**Parameter** : game\_index  
**Explain** :  
game\_index번째 게임에 통신자의 플레이어 오브젝트를 추가하고 해당 오브젝트의 인덱스 번호를 반환한다.

**void SendPlayerNum(int, SOCKET)  
Return value** : -  
**Parameter** : player \_index, client\_sock  
**Explain** :  
통신자에게 플레이어 오브젝트의 인덱스 번호를 넘겨준다.

**Client Loop**





**void DataReceiver()   
Return value** : -  
**Parameter** : -  
**Explain** :  
RecvThread()에서 불리는 함수.  
recv()를 통해 서버로부터 ObjectData의 배열을 받는다.

**void SetObjectData(ObjectData\*)  
Return value** : -  
**Parameter** : input\_array  
**Explain** :  
GameObjectManager. p\_Recv에 input\_array를 덮어씌운다.

**void GameObjectManager.SwapDataBuffer(DBufPointer)  
Return value** : -  
**Parameter** : location  
**Explain** :  
인자에 따라서 p\_Recv혹은 p\_Render를 p\_other와 교환한다.  
이 함수는 서로 다른 쓰레드에서 불릴 수 있기 때문에 임계영역으로 동기화한다.

**void Decoder()  
Return value** : -  
**Parameter** : -  
**Explain** :  
GameObjectManager.Decode()를 부른다.

**void GameObjectManager.Decode()  
Return value** : -  
**Parameter** : -  
**Explain** :  
GameObjectManager.p\_Render배열의 값들을 GameObjectManager. m\_GameObjects.Decode()의 원소로 넘겨준다.

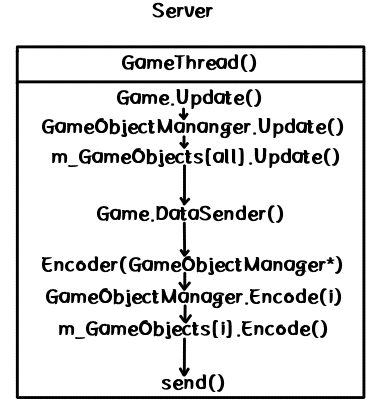
**void GameObject.Decode(ObjectData)  
Return value** : -  
**Parameter** : object\_data  
**Explain** :  
object\_data의 값으로 자신의 멤버변수들을 Update 한다.

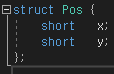
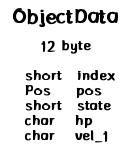
**void Render()**   
**Return value** : -  
**Parameter** : -  
**Explain** :  
RederThread()에서 불리는 함수.  
GameObjectManager.Render()를 부른다.

**void GameObjectManager.Render()**  
**Return value** : -  
**Parameter** : -  
**Explain** :  
GameObjectManager. m\_GameObjects들의 Render()를 부른다.

**void GameObject.Render()**  
**Return value** : -  
**Parameter** : -  
**Explain** :  
자신을 화면에 출력한다.

**Server Loop**





**void Update(float)**   
**Return value** : -  
**Parameter** : eTime  
**Explain** :  
GameThread()에서 불리는 함수.  
GameObjectManager.Updata()를 부른다.

**void GameObjectManager.Update(float)**  
**Return value** : -  
**Parameter** : eTime  
**Explain** :  
GameObjectManager. m\_GameObjects들의 Update()를 부른다.

**void GameObject.Updata(float)**  
**Return value** : -  
**Parameter** : eTime  
**Explain** :  
eTime에 따라 자신의 정보를 갱신한다.

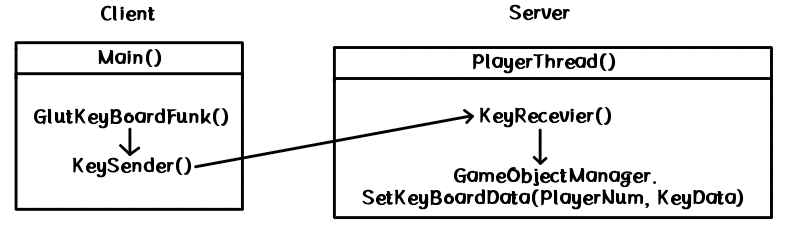
**void Game.DataSender()**   
**Return value** : -  
**Parameter** : -  
**Explain** :  
GameThread()에서 불리는 함수.  
전역함수 Encoder()를 불러 얻은 배열을 game.PD[0], game.PD[1]에게 보낸다.

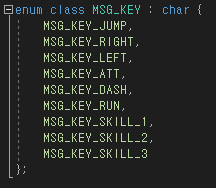
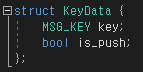
**ObjectData\* Encoder(GameObjectManager\*)**   
**Return value** : object\_data  
**Parameter** : GOMgr  
**Explain** :  
GOMgr.Encode를 불러 얻은 배열을 반환한다.

**ObjectData\* GameObjectManager.Encode(int)**  
**Return value** : object\_data  
**Parameter** : object\_index  
**Explain** :  
GameObjectManager.m\_GameObjects.Encode들의 Encode()를 불러 반환값들로 배열을 만들어 반환한다.

**ObjectData GameObject.Encode()**  
**Return value** : object\_data  
**Parameter** : -  
**Explain** :  
자신의 정보를 ObjectData 형식에 맞게 바꾸어 반환한다.

**Key Event**





**void KeySender(KeyData)**   
**Return value** : -  
**Parameter** : key\_data  
**Explain** :  
콜백함수 안에서 사용될 함수로 입력한 키와 눌림 여부를 서버에 송신한다.

**void KeyRecevier(KeyData)**   
**Return value** : -  
**Parameter** : key\_data  
**Explain** :  
각 플레이어들어게 배정된 PlayerThread()에서 사용될 함수로 Client가 입력한 KeyData를 받아 GameObjectManager.SetKeyBoardData()를 호출한다.

**void GameObjectManager.SetKeyBoardData(int, KeyData)**  
**Return value** : -  
**Parameter** : player\_num, key\_data  
**Explain** :  
입력받은 플레이어의 키 입력상태를 갱신한다.

**부록**

#define CLIENT 0

#define SERVER 1

#define CONSOL SERVER

#define MAX\_OBJECT\_COUNT 1000 // 한 맵에 존재할 수 있는 최대 GameObject 개수 \*\*\*임시값

#define MAX\_GAME\_NUM 1 // 서버가 생성할 수 있는 최대 게임의 개수 \*\*\*임시값

// 키 매핑용 Define

#define KEY\_JUMP ('KEY\_JUMP\_1' || 'KEY\_JUMP\_2') // 미정

#define KEY\_RIGHT

#define KEY\_LEFT

#define KEY\_ATT

#define KEY\_DASH

#define KEY\_RUN

#define KEY\_SKILL\_1

#define KEY\_SKILL\_2

#define KEY\_SKILL\_3

// 통신용 Define

#define OBJECT\_TYPE\_PLAYER 0x0000

#define OBJECT\_TYPE\_BASE 0x1000

#define OBJECT\_TYPE\_BIRD 0x1001

#define OBJECT\_TYPE\_WOLF 0x1002

#define OBJECT\_TYPE\_PLANT 0x1003

#define OBJECT\_TYPE\_BRICK 0x2000

#define OBJECT\_TYPE\_BACKGROUND 0x2001

#define OBJECT\_TYPE\_PLANTPROJECTILE 0x3000

// 통신용 열거형

enum class MSG\_KEY : char {

MSG\_KEY\_JUMP,

MSG\_KEY\_RIGHT,

MSG\_KEY\_LEFT,

MSG\_KEY\_ATT,

MSG\_KEY\_DASH,

MSG\_KEY\_RUN,

MSG\_KEY\_SKILL\_1,

MSG\_KEY\_SKILL\_2,

MSG\_KEY\_SKILL\_3

};

enum class STATE : short{

create\_object,

delete\_object,

idle,

jump,

double\_jump,

left,

right,

att,

dash,

run,

skill\_1,

skill\_2,

skill\_3

};

//-----------------------------------------------------------------------------------

struct Pos {

short x;

short y;

};

struct KeyData {

MSG\_KEY key;

bool is\_push;

};

struct ObjectData {

short index;

Pos pos;

STATE state;

char hp;

char vel\_1; //상태가 생성일 경우 오브젝트 타입이 되고 이외의 경우 점수가 된다.

};

//-----------------------------------------------------------------------------------

class GameObject {

private:

int 위치\_x;

int 위치\_y;

int 크기\_x;

int 크기\_y;

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

//-----------------------------------------------------------------------------------

class Creature :GameObject {

private:

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

};

//-----------------------------------------------------------------------------------

class Player : Creature {

private:

int 점수;

CImage\* 이미지;

KeyBoardManager kbm;

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

void SetKeyBoardData(KeyData keydata);

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

//-----------------------------------------------------------------------------------

class Enemy : Creature {

private:

CImage 이미지;

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

class Base : Enemy {

private:

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

class Bird : Enemy {

private:

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

class Wolf : Enemy {

private:

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

class Plant : Enemy {

private:

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

class MapObject : GameObject {

private:

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

//-----------------------------------------------------------------------------------

class Brick : MapObject {

private:

CImage 이미지;

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

class BackGround : MapObject {

private:

CImage 이미지;

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

//-----------------------------------------------------------------------------------

class Projectile : GameObject {

private:

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

class PlantProjectile : Projectile {

private:

public:

#if CONSOL==CLIENT

public:

virtual void Render();

virtual void Decode(ObjectData object\_data);

#elif CONSOL==SERVER

public:

virtual void Update(float eTime);

virtual ObjectData Encode();

#endif

};

//-----------------------------------------------------------------------------------

class KeyBoardManager {

private:

bool key\_jump;

bool key\_right;

bool key\_left;

bool key\_att;

bool key\_dash;

bool key\_run;

bool key\_skill\_1;

bool key\_skill\_2;

bool key\_skill\_3;

public:

void reset();

void SetKeyBoardData(KeyData keydata);

};

//-----------------------------------------------------------------------------------

class GameObjectManager {

private:

GameObject\* m\_GameObjects[MAX\_OBJECT\_COUNT];

public:

int AddObject(GameObject\* new\_object, int index);

GameObject GetObject(int index);

bool DeleteObject(int index);

#if CONSOL==CLIENT

private:

enum class DBufPointer : int{

Recv,

Render

};

ObjectData RecvData[3][MAX\_OBJECT\_COUNT];

ObjectData\* p\_Recv;

ObjectData\* p\_Render;

ObjectData\* p\_other;

public:

void SetObjectData(ObjectData\* input\_array);

void SwapDataBuffer(DBufPointer location);

void Decode();

void Render();

#elif CONSOL==SERVER

private:

public:

void Update(float eTime);

void SetKeyBoardData(int player\_num, KeyData keydata);

ObjectData\* Encode(int object\_index);

#endif

};

//-----------------------------------------------------------------------------------

#if CONSOL==CLIENT

void DataReceiver();

int Connecter(char\* ServerIp, int PortNum);

int RecvPlayerNum();

void Render();

bool KeySender(KeyData key\_data);

void RenderScene(void);

void MouseInput(int button, int state, int x, int y);

void KeyInput(unsigned char key, int x, int y);

void KeyUpInput(unsigned char key, int x, int y);

void SpecialKeyInput(int key, int x, int y);

void SpecialUpKeyInput(int key, int x, int y);

#elif CONSOL==SERVER

struct PlayerData {

SOCKET sock; // Game.DataSender() 에서 사용

int gamenum; //-플레이어가 속해있는 게임의 번호

int playerIndex; //-플레이어 캐릭터의 오브젝트 인덱스

};

class Game {

GameObjectManager\* GOMgr;

PlayerData\* PD[2];

void Update(float eTime);

bool DataSender();

};

class PlayerManager {

private:

Game\* m\_ppGame[MAX\_GAME\_NUM];

PlayerData\* m\_ppPlayers[MAX\_GAME\_NUM \* 2];

public:

int CheckGame();

int MakeGame();

int MakePlayer(int game\_index);

void SendPlayerNum(int player\_index, SOCKET client\_sock);

};

void DataSender();

void KeyRecevier();

ObjectData\* Encoder(GameObjectManager\* GOMgr);

#endif

**팀원 별 역할 분담**

A : 류연우

B : 박기정

* class GameObject -B
  + class Creature : GameObject -B
    - class Player : Creature -B
    - class Enemy : Creature -A
  + class MapObject : GameObject -B
  + class Projectile : GameObject -A
* 플레이어 접속부분 - A
  + Client Main() -A
  + Server Main() -A
  + class PlayerManager -A
* ObjectData 통신 부분 -B
  + class Game -B
  + class GameObjectManager -B
* KeyData 통신 부분 -A
  + class KeyBoardManager -A

**개발 환경**

류연우

IDE : Visual Studio 2022

CPU : Intel Core i7-11370H

RAM : 40GB

GPU : NVIDIA GeForce RTX 3070 Laptop GPU 8GB

박기정

IDE : Visual Studio 2019

CPU : Ryzen 9 5900HX

RAM : 32GB

GPU : NVIDIA GeForce RTX 3070 Laptop GPU 8GB

**개발 일정**

표표