네트워크 게임 프로그래밍
Term Project 추진 계획서

2017182028 윤혜림

2017182045 황신필

1. 애플리케이션 기획

1-1 개발 환경

- Window 10
- Visual Studio 2019
- Window Socket API
- Direct2D
- GitHub

1-2 네트워크 구현

- 유저 채팅 기능
- 실시간 멀티플레이(2~4명)

1-3 게임 소개

이름 : 붐렉트(BombRect)

장르 : 아케이드 대전 게임

플레이어 수 : 2 ~ 4인

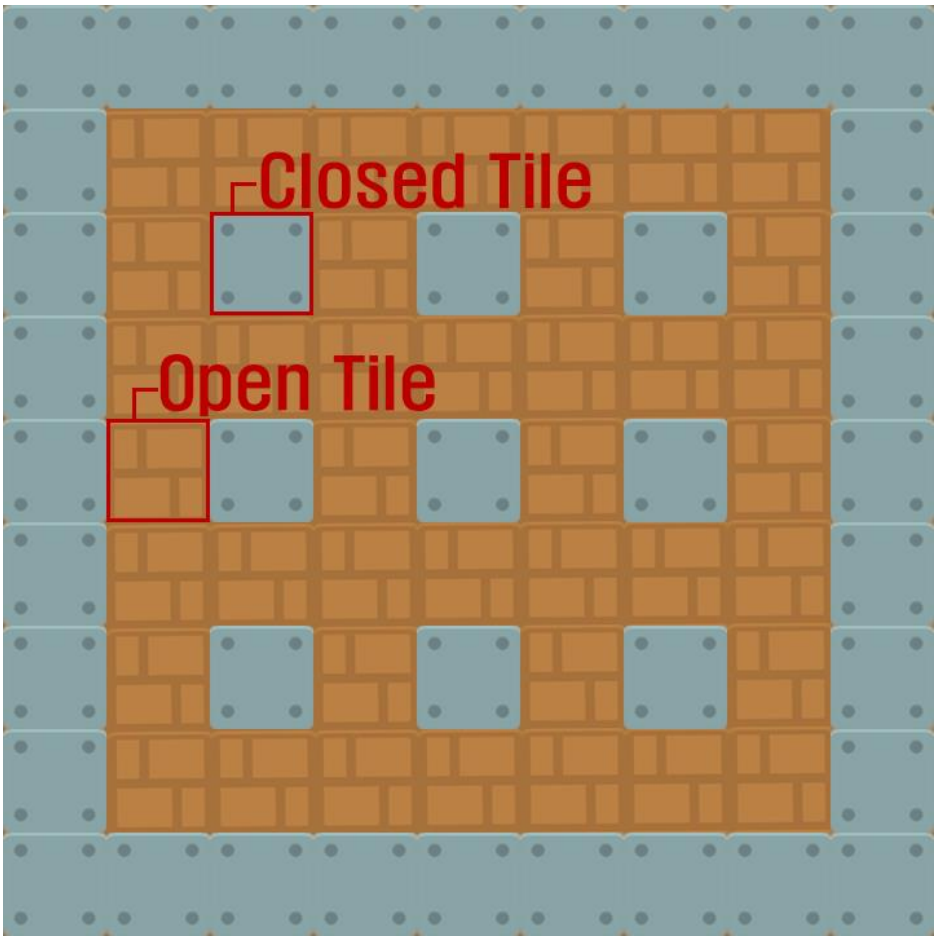
플랫폼 : Window PC 게임

벤치마킹 게임 : 봄버맨(Bomberman)

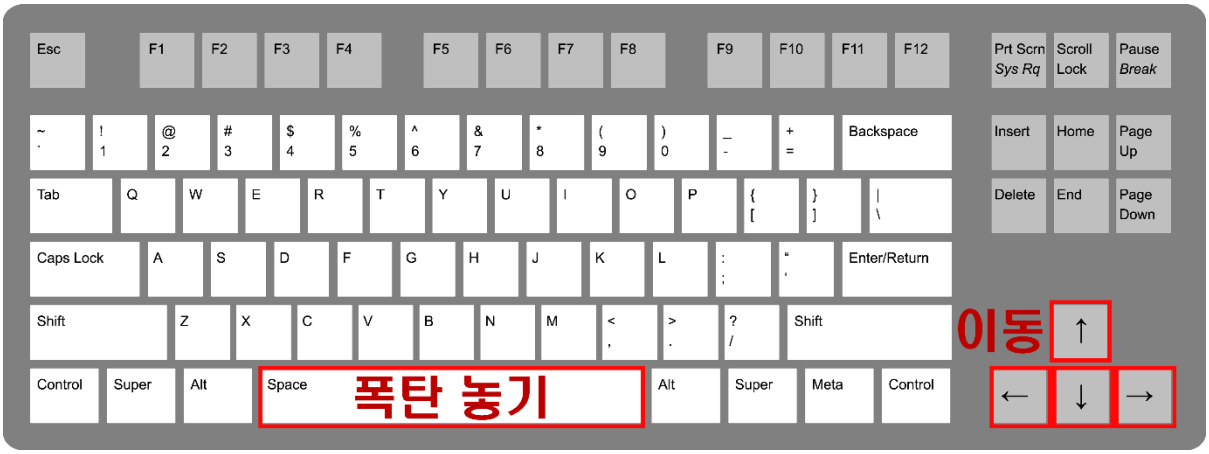
[출처 - https://namu.wiki/w/%EB%B4%84%EB%B2%84%EB%A7%A8%20%EC%8B%9C%EB%A6%AC%EC%A6%88]

1-4 게임 플레이

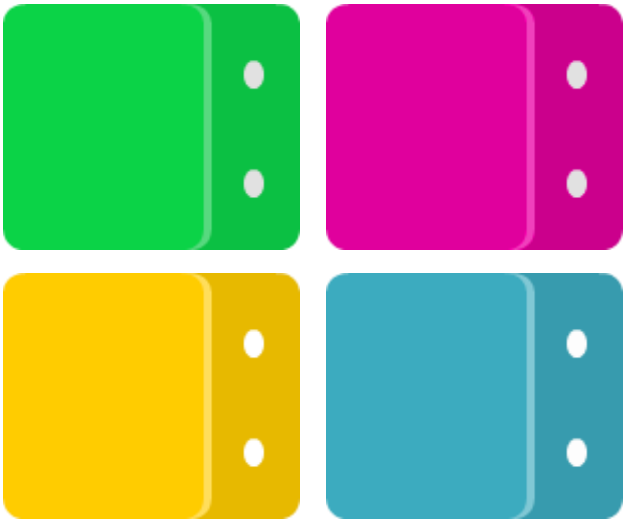
■ 맵



■ 조작키



■ 플레이어



플레이어는 최대 3개의 폭탄을 동시에 맵에 놓을 수 있다.
폭탄으로부터 생성되는 데미지 영역에 3번 충돌하면 패배한다.

■ 폭탄

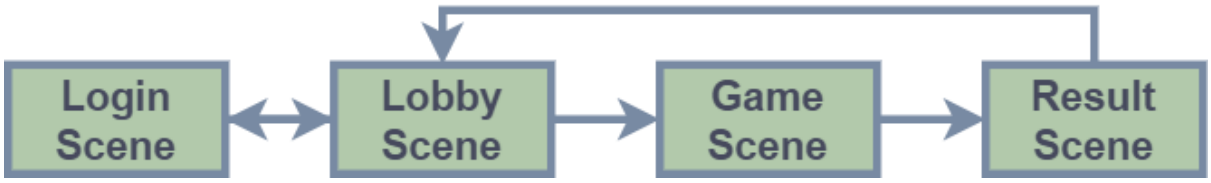
플레이어가 둔 폭탄은 일정 시간후에 터진다.
폭탄이 터진 후 데미지 영역이 생성된다.

■ 데미지 영역

폭탄이 터진 위치에서 상하좌우로 연속된 Open Tile이
데미지 영역의 범위가 된다.
데미지 박스는 일정 시간후에 사라진다.

2. High-Level 디자인

Scene 흐름도



■ Login Scene

유저가 사용할 닉네임을 설정하고, IP 주소를 입력해 서버에 접속을 시도한다.

■ Lobby Scene

접속된 유저를 보여주고, 채팅 기능을 통해 소통할 수 있다. 모든 유저가 Ready 사인을 보내면 게임을 시작한다.

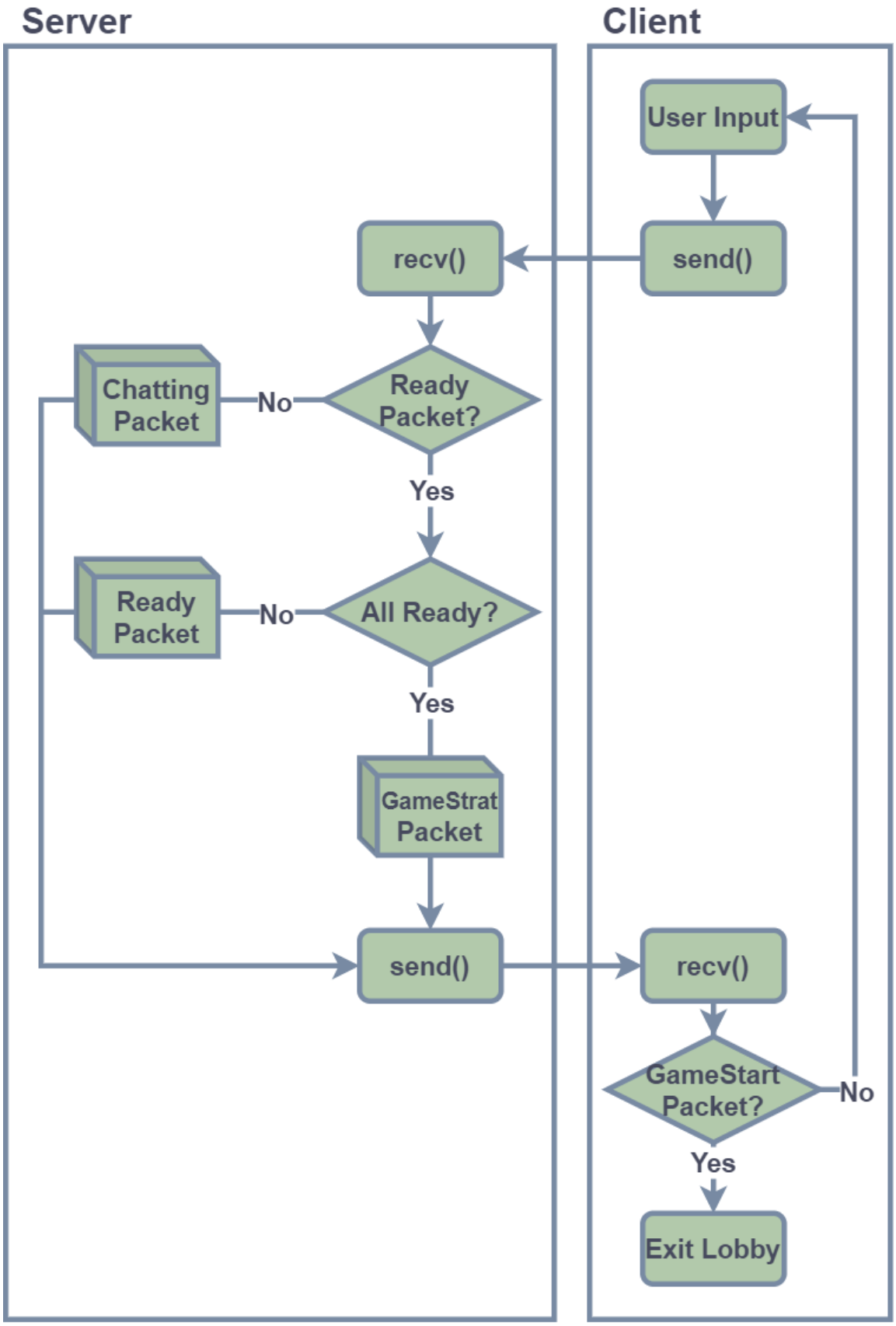
■ Game Scene

게임을 진행한다. 패배했지만 아직 게임이 끝나지 않은 경우, 관전 모드로 전환되어 게임이 끝날 때까지 기다린다. 생존한 유저가 한 명이라면 게임을 종료한다.

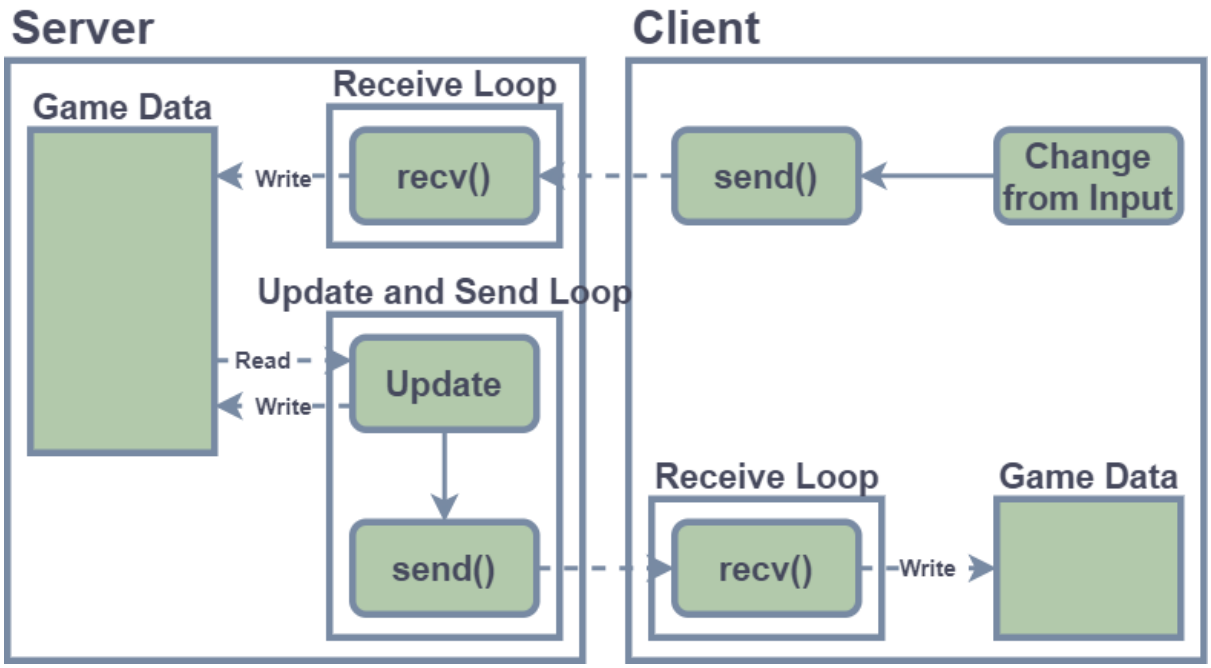
■ Result Scene

오래 생존한 순서에 따라 순위를 정해 보여준다. 일정 시간이 지난 후 Lobby Scene 으로 전환된다.

Lobby Scene Flow Chart

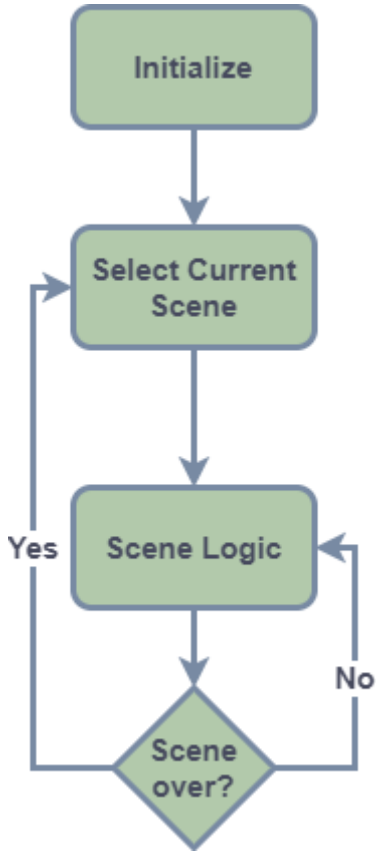


Game Scene Data Flow



3. Low-Level 디자인

3-1 서버 디자인



1. Initialize

서버가 생성될 때 초기 값을 정한다.

2. Check Scene

Scene 의 정보를 받아와 대응되는 Scene 함수를 호출한다.

3. Scene Loop

Scene 의 Logic 을 반복한다.

4. Scene Over

Scene 의 Loop 탈출 여부를 판단한다. Scene 이 종료되면, Scene 의 값을 바꾼다.

함수 설계

1. ClientThread

```
// 클라이언트의 Scene 상태를 저장한 후 알아서 판단
DWORD WINAPI ClientThread(LPVOID lpParam) {
    int id = (int)lpParam;
    void(*curr_comunication)(int);
    curr_comunication = LobbyCommunicate;

    while (true) {
        curr_comunication(id);

        switch (scene_state) {
        case Scene::LOBBY:
            curr_comunication = LobbyCommunicate;
            break;
        case Scene::GAME:
            curr_comunication = GameCommunicate;
            break;
        case Scene::RESULT:
            curr_comunication = ResultCommunicate;
            break;
        }
    }
}
```

2. LobbyCommunicate

```
void LobbyCommunicate(int id) {
    while (true) {
        // recv();

        // recv한 정보를 토대로 String과 Ready를 판단한다.
        // if (String) 데이터를 모든 클라이언트에게 보낸다.
        // else
        //      if (AllReady) 게임시작 패킷을 모든 클라이언트에게 보낸다.
        //      else (!AllReady) 모든 클라이언트에게 레디 정보를 보낸다.


        // scene_state = Scene::GAME;
        // break;
    }
}
```

3. GameCommunicate

```
void GameCommunicate(int id) {
    // recv();
    // GaemPacket을 받는다.
    // Player의 상태를 Write한다.
    // if (폭탄 패킷) Player가 소지하고 있는 Bomb의 개수를 감소 후에 Write 한다.
    //
}
```

4. ResultCommunicate

```
void ResultCommunicate(int id) {

    // 일정 시간이 지나면 로비씬으로 전환
    // Scene 전환 패킷을 모든 클라이언트에게 보낸다.

}
```

5. UpdateAndSend

```
DWORD WINAPI UpdateAndSend(LPVOID lpParam) {
    // lock()
    // Game World의 모든 정보를 읽어온다.
    // unlock()
    //
    //  Player의 위치를 업데이트 해준다.
    //
    //  Bomb의 카운트다운을 갱신한다.
    //  if(bomb의 카운트다운 == 0)
    //      Bomb의 위치를 통해 Explosion의 위치를 판단후 이차원 배열에 저장한다.
    //      Player의 Bomb의 소지 수를 늘린다.
    //
    //  if (Explosion !=0 )
    //      Explosion 시간을 갱신한다.
    //
    //   플레이어의 충돌처리 연산을 한다.
    //   if (closed tile) 진행불가
    //   else 진행 가능
    //
    //   if (Explosion) Hp감소
    //
    //   lock()
    //   전역 메모리 위치에 업데이트 정보를 복사한다.
    //   unlock()

    // send(모든 클라이언트)

}
```

1. ClientThread: Scene 을 판단해 호출한다.

2. LobbyCommunicate: 로비의 패킷 처리 Logic 을 담당한다. 로비는 Chatting 과 Ready 패킷을 판단하고 처리한다. 모두가 Ready 상태가 되면 게임시작 패킷을 보내고 리턴한다.

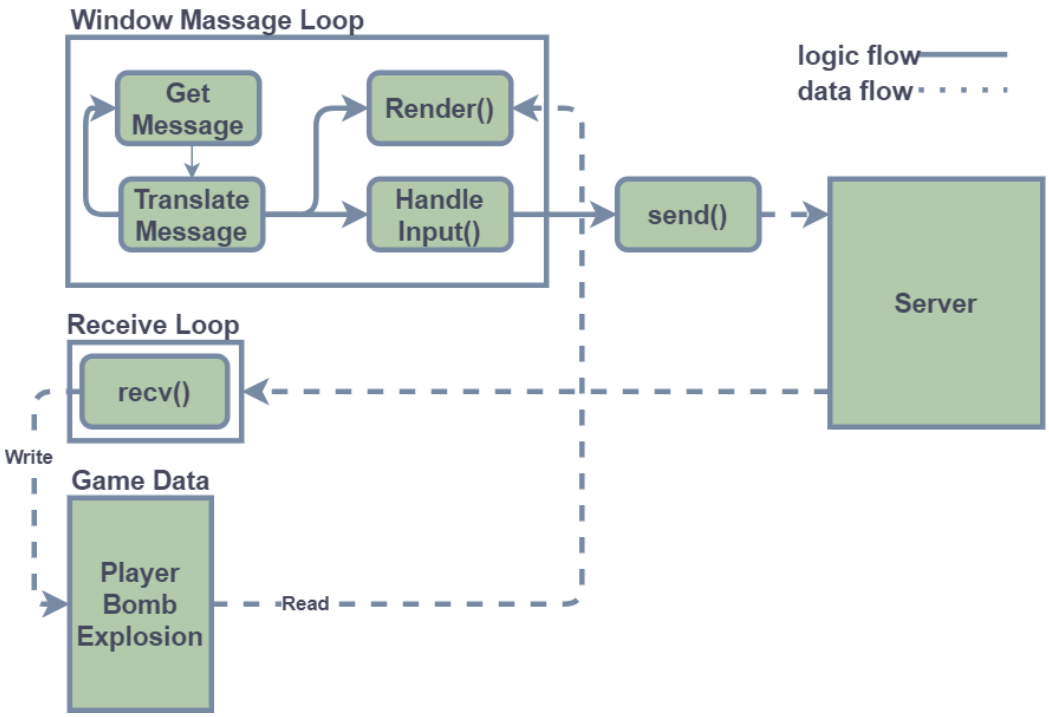
3. GameCommunicate: 게임안에서 플레이어의 상태, 폭탄의 상태를 Write 한다.

4. ResultCommunicate: 결과창을 보여준다. 결과창의 Scene 정보를 가지고 있는다. 결과창에서 로비로 돌아가는 타이머를 갱신하다. 타이머가 0 이되면 Scene 전환 패킷을 보낸다.

5. UpdateAndSend: 최신의 Game World 정보를 읽어와 계산하고, 정보를 갱신한 후 모든 클라이언트한테 보내준다. 전역 데이터에 접근할 때 동기화 기법을 사용한다.

3-2 클라이언트 디자인

Game Scene Flow



함수 설계

```
// 메세지 루프 안에서 실행된다.
void GameScene::Render()
{
    // 게임 월드 데이터를 읽어와
    // Graphic API를 호출한다.
}

// 메세지 루프 안에서 실행된다.
void GameScene::HandleInput()
{
    if (input == arrow_key) {
        // apply input to player
        if (player_state_has_changed) {
            // send player_state_packet
        }
    }

    if (input == space_key) {
        // send bomb_packet
    }
}
```

```
void NetworkCommunicator::RecieveGameScenePacket()
{
    while (true) {
        // recv();

        if (player_count == 1) {
            // RusultScene으로 전환
            break;
        }

        // lock();
        // 서버로부터 받은 모든 게임 월드 데이터를 전역 변수에 Write
        // unlock();
    }
}
```

3-3 객체 모델링

```
class TilePos {
    int r;
    int c;
};

enum class TileType {
    OPEN,
    CLOSED,
};

class Tile {
    TilePos pos;
    TileType type;
};

class Bomb {
    TilePos pos;
    int count_down;
};
```

```
enum class PlayerState {
    IDLE,
    UP,
    DOWN,
    LIGHT,
    LEFT,
    DEAD,
};

class Player {
    int id;
    PlayerState state;
    Vector2 pos;
    int life_count;
    int bomb_count;
    int no_damage_time;
};

class Explosive {
    TilePos bomb_pos;
    int count_down;
};
```

3-4 패킷 모델링

```
namespace login_packet {
#pragma pack(1)
    struct CS_Nickname {
        unsigned short size;
        char buf[16];
    };
#pragma pack()
}
namespace lobby_packet {
    enum class PacketType: char {
        READY,
        CHATING,
        GAME_START
    };

#pragma pack(1)
    struct Ready {
        PacketType type;
        unsigned short size;
    };

    struct Chatting {
        PacketType type;
        unsigned short size;
        char string[256];
    };

    struct SC_GameStart {
        PacketType type;
        unsigned short size;
    };
#pragma pack()
}
```

```
namespace game_packet {
    enum class PacketType {
        PlayerState,
        Bomb,
    };
#pragma pack(1)
    struct CS_PlayerState {
        PacketType type;
        PlayerState state;
    };

    struct CS_Bomb {
        PacketType type;
    };

    struct SC_WorldState {
        unsigned short player;
        unsigned short bomb;
        unsigned short explosive;
        char buf[1024];
    };
#pragma pack()
}
namespace result_packet {
#pragma pack(1)
    struct TimeOver {
        bool time_over;
    };
#pragma pack()
}
```

4. 팀원 별 역할분담

		윤혜림	황신필
서버 설계		O	O
클라이언트 설계		O	X
기획서 작성		O	O
객체 모델링		O	O
패킷 모델링		O	O
Server	GameCommunication	X	O
	LobbyCommunication	X	O
	ResultCommunication	X	O
	UpdateAndSend	X	O
Client	GameFramework	O	X
	Scene	O	X
	Renderer	O	X

5. 개발 일정

11 월 첫째 주	11/02(월)	11/03(화)	11/04(수)	11/05(목)	11/06(금)	11/07(토)	11/08(일)
윤혜림					설계 및 모델링	Framework 및 Scene 구현	
황신필			설계 및 모델링				

11 월 둘째 주	11/09(월)	11/10(화)	11/11(수)	11/12(목)	11/13(금)	11/14(토)	11/15(일)
윤혜림	Renderer 구현		Renderer 구현			Renderer 구현	Renderer 구현
황신필	패킷 전달 구현	패킷 전달 구현	패킷 전달 테스트		패킷 전달 테스트 및 수정		

11 월 셋째 주	11/16(월)	11/17(화)	11/18(수)	11/19(목)	11/20(금)	11/21(토)	11/22(일)
윤혜림	Renderer 구현		Chatting 클라이언트 구현		Chatting 클라이언트 구현	Login Scene 구현	
황신필	충돌 체크 함수 prototype 제작	충돌 체크 함수 prototype 제작	chatting 서버 제작		chatting 서버 제작		

11 월 넷째 주	11/23(월)	11/24(화)	11/25(수)	11/26(목)	11/27(금)	11/28(토)	11/29(일)
윤혜림	Lobby Scene 구현	Game Scene 구현				Game Scene 구현	Game Scene 구현
황신필	Update 구현	Update 구현	동기화 최적화		동기화 최적화		

12 월 첫째 주	11/30(월)	12/01(화)	12/02(수)	12/03(목)	12/04(금)	12/05(토)	12/06(일)
윤혜림	Result Scene 구현				최적화 및 디버그	최적화 및 디버그	최적화 및 디버그
황신필	동기화 최적화				최적화 및 디버그	최적화 및 디버그	최적화 및 디버그