**네트워크 게임 프로그래밍 추진 계획서**

|  |  |
| --- | --- |
| 학번 | 이름 |
| 2013182001 | 강수빈 |
| 2013182010 | 김병진 |
| 2013182030 | 이상기 |

목차

내용

[애플리케이션 기획 3](#_Toc497956189)

[개요 3](#_Toc497956190)

[네트워크 기능 6](#_Toc497956191)

[네트워크 구조 개요도 6](#_Toc497956192)

[서버 7](#_Toc497956193)

[클라이언트 7](#_Toc497956194)

[High-Level Design 10](#_Toc497956195)

[Low-Level Design 16](#_Toc497956196)

[팀원 별 역할 분담 24](#_Toc497956197)

[개발 환경 24](#_Toc497956198)

[테스트 환경 24](#_Toc497956199)

[개발 일정 25](#_Toc497956200)

[11/9~11/15 25](#_Toc497956201)

[11/16~11/22 26](#_Toc497956202)

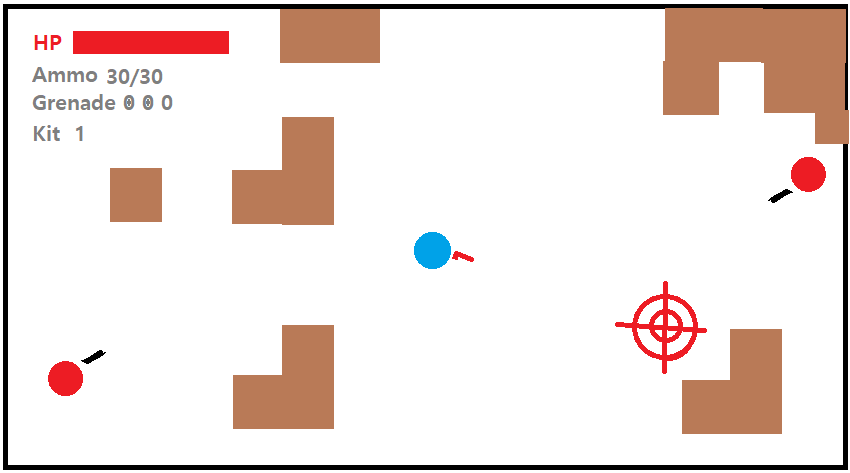
[11/23~11/29 27](#_Toc497956203)

[11/30~12/6 28](#_Toc497956204)

[12/7~11/13 29](#_Toc497956205)

# 애플리케이션 기획

### 개요



제목 :: Network Shooting Game Simulator

개요 :: 로비에서 네트워크 접속을 통해 방에 접속하여 플레이하는 2D 탑 뷰 슈팅게임이다.

이 게임은 키보드 WASD 키로 움직이며 플레이어가 마우스를 사용하여 적을 조준하여 사격하며 진행하는 게임으로서 게임 시작 이후 해당 룸에 플레이어가 단 한 명만 남게 되면 게임이 종료된다.

조작법

|  |  |
| --- | --- |
| 마우스 이동 | 조준점 이동 |
| 마우스 왼쪽 클릭 | 사격 |
| W | Y축 양의 방향 ( 위 방향 ) 으로 이동 |
| S | Y축 음의 방향 ( 아래 방향 ) 으로 이동 |
| A | X축 음의 방향 ( 왼쪽 방향 ) 으로 이동 |
| D | X축 양의 방향 ( 오른쪽 방향 ) 으로 이동 |
| G | 수류탄 투척 |
| R | 재장전 |

* 플레이어

플레이어는 서버에 접속한 클라이언트가 조작하는 에이전트다.

플레이어는 게임 시작 시 서버에서 정의한 좌표에 생성된다.

플레이어는 30발 탄알집을 사용해서 사격을 할 수 있다.

사격을 한번 시행할 시마다 탄알집에서 탄환을 하나 소비한다.

탄알집의 모든 탄환을 소비하면 재장전 커맨드를 입력하여 탄환을 보충할 수 있다.

각 플레이어는 3발의 수류탄, 그리고 하나의 터렛 설치 키트 하나를 지급받는다.

플레이어는 클라이언트에서 키보드와 마우스를 사용하여 에이전트를 조작할 수 있다.

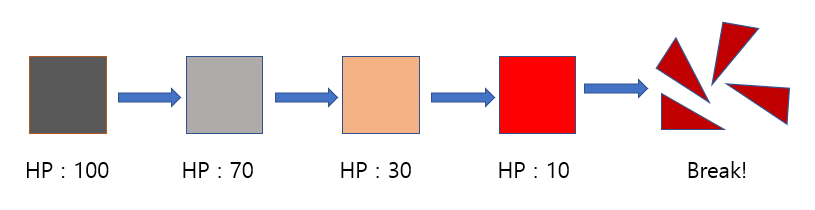
플레이어는 G키를 입력하여 바라보는 방향으로 수류탄을 투척할 수 있다.

수류탄은 투척된 직후 3초 이후 터지며 수류탄 반경 내에 있는 모든 오브젝트에게 손상을 준다.

플레이어는 터렛 설치 키트를 사용해 플레이어의 터렛을 설치할 수 있다.

플레이어의 체력이 0이 되면 사망한다.

* 벽



벽은 서버 호스트에 저장되어 있는 맵 데이터를 읽어 데이터에 따라 벽 오브젝트를 배치한다.

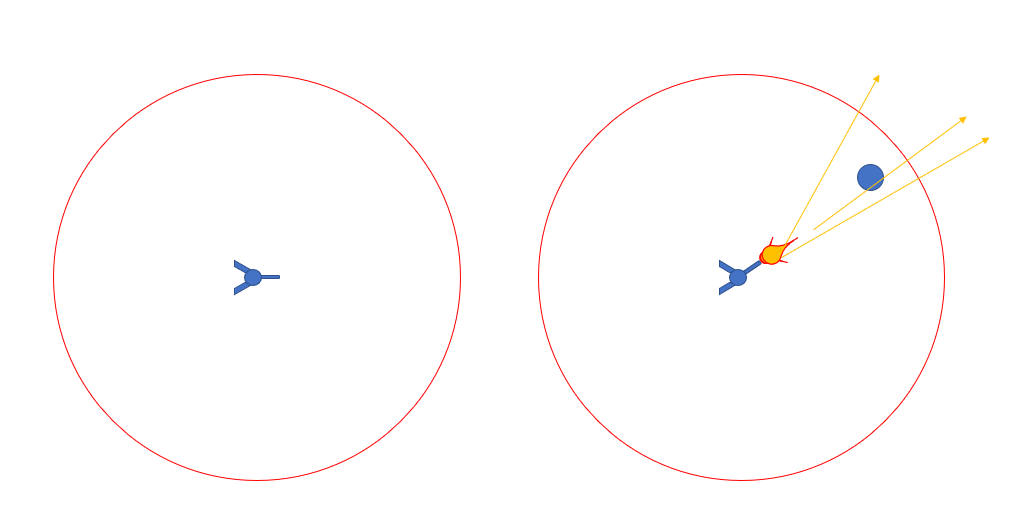
벽은 플레이어와 투사체와 충돌판정을 하며 플레이어가 이동하는 것을 막는 장애물의 역할과 투사체의 진행을 가로막는 엄폐물의 역할을 한다.

벽은 고유한 체력을 가지고 있다. 체력이 줄어들면 줄어들수록 손상되는 이미지를 출력하며 벽이 가진 체력 수치가 0이 되면 파괴된다.

* 에이전트

AI는 플레이어가 따로 컨트롤하지 않는 에이전트로서 서버에서 미리 정의된 상태기계에 따라 움직이거나 사격하는 행위를 한다.

1. 터렛



* 터렛의 사격 도식 :: 터렛은 고유의 사거리를 지니고, 사격을 할 수 있다.

터렛은 맵 상에 무작위로 배치되거나, 플레이어가 아이템을 사용해서 설치했을 때 생성되는 에이전트이다.

다른 플레이어가 설치한 터렛의 경우 자신이 어느 플레이어에게 설치되었는지를 기억한다.

터렛은 자신이 설치한 플레이어를 제외한 모든 에이전트를 적으로 간주한다.

터렛은 각자가 고유의 사정거리를 지니고 있다. 사거리 내부에 적이 들어오면 해당 적의 방향으로 사격을 개시한다.

1. 자동으로 움직이는 에이전트 ( 이하 AI로 지칭한다. )

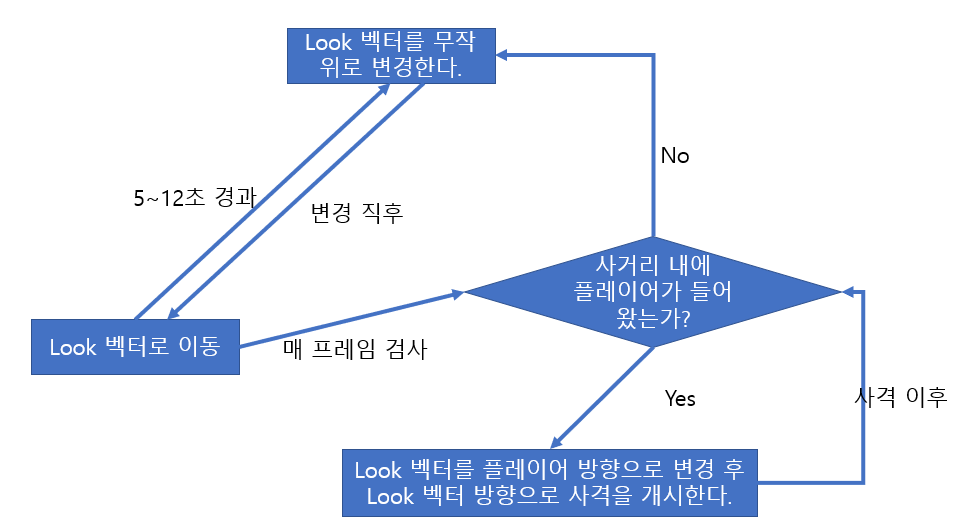
AI는 5 + rand()%7초 동안 바라보고 있는 방향으로 이동한다.

AI 는 이동 상태에서 5 + rand()%7초가 지나면 무작위로 바라보는 방향을 변경한다.

AI는 매 프레임마다 AI의 사거리 내에 다른 플레이어가 있는지 판정을 한다.

플레이어가 사거리 내에 있을 시 플레이어 방향으로 사격을 개시한다.

사격을 행하는 것은 플레이어가 AI의 사거리 밖으로 벗어날 때 까지 진행한다.



* AI의 유한상태기계

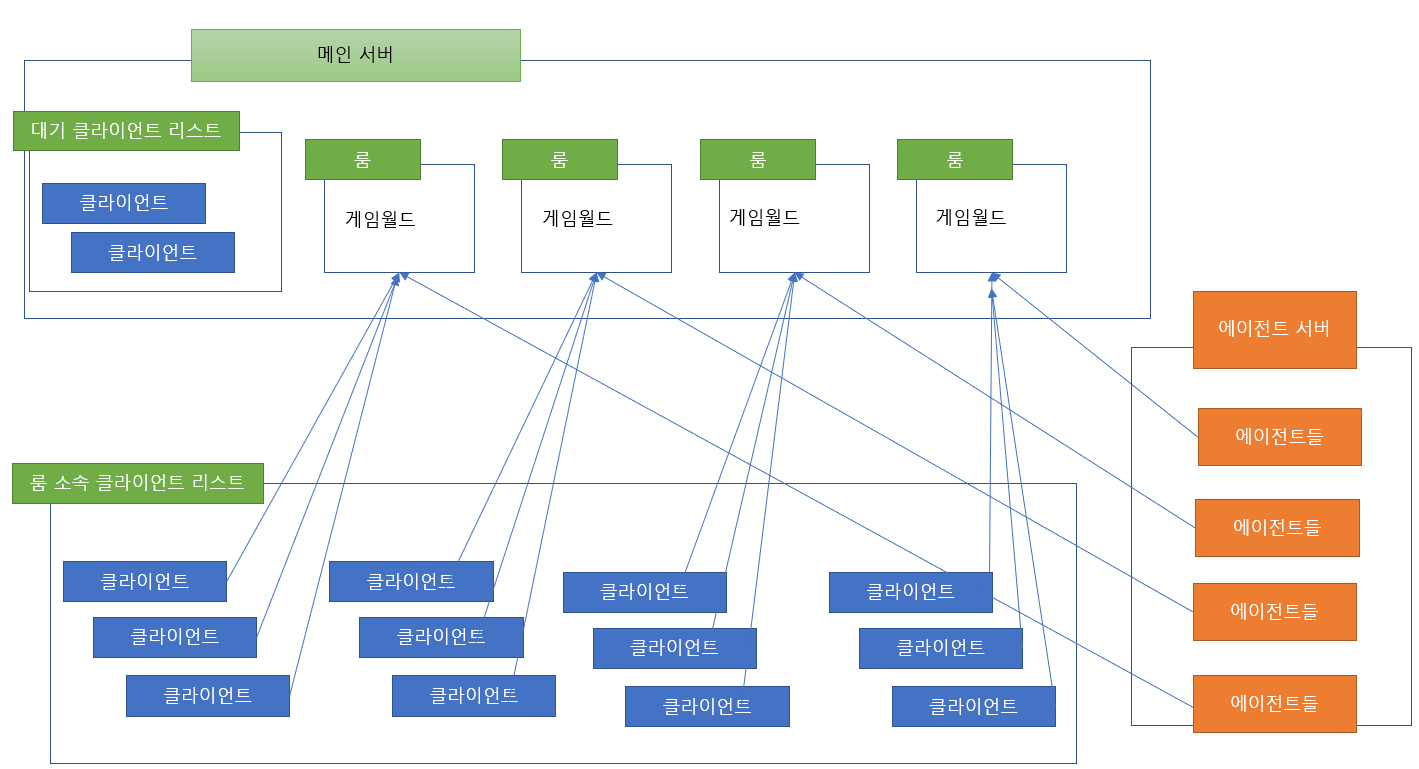
## 네트워크 기능

클라이언트와 서버간 데이터들은 반드시 다음 사항을 보장한다.

네트워크 통신을 통해 주고 받는 데이터들은 유실이 없어야 한다.

따라서 서버와 클라이언트 간에 모든 데이터 통신은 **TCP 프로토콜**을 사용한다.

### 네트워크 구조 개요도



### 서버

#### 메인 서버

메인 서버는 게임에 접속한 클라이언트들이 룸에 접속하여 게임을 진행할 때 발생하는 모든 상호작용을 컨트롤하고 상호작용의 결과들을 클라이언트들에게 다시 송신하는 역할을 한다.

###### 룸의 리스트

룸은 게임이 진행되고 있는 월드를 지칭한다.

생성된 룸은 서버에서 연산한 정보들을 룸에 소속된 클라이언트 들에게 데이터를 보내기 위해 해당 룸에 소속된 클라이언트들의 정보를 가진다.

###### 클라이언트 리스트

클라이언트 리스트는 메인 서버에 접속한 클라이언트들의 정보를 저장하는 자료구조다. 클라이언트 리스트는 대기 클라이언트 리스트와 룸 소속 클라이언트 리스트로 나뉜다.

대기 클라이언트 리스트

대기 클라이언트 리스트는 서버에 접속한 후 아직 게임을 진행하지 않고 대기하는 클라이언트들을 관리한다.

룸 소속 클라이언트 리스트

하나의 룸에 소속되어 게임을 진행하고 있는 클라이언트들을 관리한다.

메인 서버는 클라이언트 리스트와 생성된 룸의 리스트들을 관리한다.

#### 에이전트 서버

에이전트 서버는 메인 서버에서 게임이 진행될 때 메인 서버의 룸에서 행동하는 에이전트들의 상태기계와 상태의 전환들을 관리하는 역할을 한다.

###### 룸의 리스트

룸의 리스트는 메인 서버가 가지고 있는 룸의 리스트와 동기화 한다.

### 클라이언트

클라이언트는 서버가 생성한 룸에서 다음과 같은 정보를 가진다.

클라이언트 캐릭터의 체력, 탄약, 수류탄, 터렛 킷 개수

다른 에이전트 오브젝트의 정보

클라이언트에서는 메인 서버가 보낸 월드 정보를 수신을 받고 그 정보를 바탕으로 게임 화면을 렌더링한다.

##### 클라이언트의 메인 서버로의 접속

클라이언트는 프로그램을 실행한 후 메인 서버에게 접속을 요청한다.

클라이언트가 서버에 접속을 성공하면 게임을 진행할 룸이 생성될 때까지 대기한다.

메인 서버가 룸을 생성하면 대기하던 클라이언트들은 생성된 룸으로 들어가 게임플레이를 진행한다.

게임이 끝나면 다시 대기한다.

##### 메시지 리시버

메시지 리시버는 클라이언트와 메인 서버, 그리고 에이전트 서버가 공통적으로 사용하는 스레드 함수이다.

메시지 리시버에서는 호스트에게 전송된 메시지들을 수신하는 역할을 하며, 수신된 모든 메시지를 메시지 큐에 저장하는 역할을 수행한다.

##### 클라이언트와 메인 서버간 네트워크 통신을 위한 메시지 큐

메시지 큐는 메시지 리시버가 받아온 메시지들을 저장하는 자료구조다.

##### 액션 메시지

액션 메시지는 클라이언트 또는 에이전트가 메인 서버에게 보내는 메시지를 지칭한다.

액션 메시지는 클라이언트 또는 에이전트의 변경할 행동을 나타내는 메시지다.

클라이언트는 조작 커맨드를 입력할 시 서버에게 액션 메시지를 송신한다.

에이전트의 경우 상태 전환이 발생할 시 서버에게 액션 메시지를 송신한다.

액션 메시지는 이동, 사격, 터렛 설치, 재장전 타입을 가지고 행동을 갱신할 오브젝트의 ID와 변경 정보를 가진다.

그 다음 클라이언트는 응답이 올 때까지 가장 최근에 했던 행동을 유지한다.

서버는 클라이언트와 에이전트의 액션 메시지를 받아 메시지 큐에 추가한다.

##### 허용 가능한 메시지

예를 들어 이미 클라이언트가 사망하였는데 사망한 플레이어가 보낸 액션 메시지가 메인 서버의 메시지 큐에 남아 있을 수 있다.

이를 위해 메시지 큐에서 메시지를 처리하기 전에 해당 메시지가 허용 가능한 메시지인지에 대한 판별을 우선적으로 시행한다.

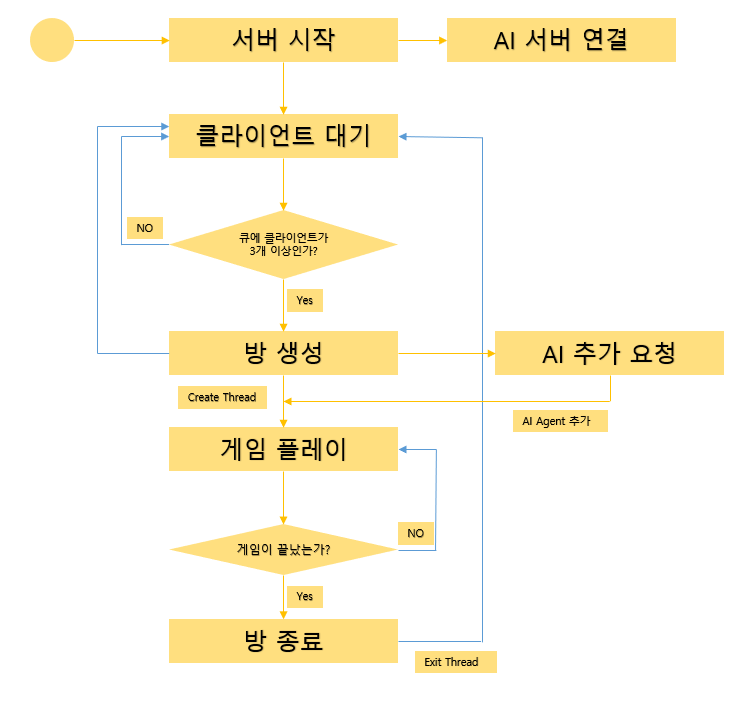
만약 해당 메시지가 허용 가능한 메시지가 아닐 경우 해당 메시지를 처리하지 않는다.

##### 메인 서버가 클라이언트와 에이전트에게 보내는 메시지

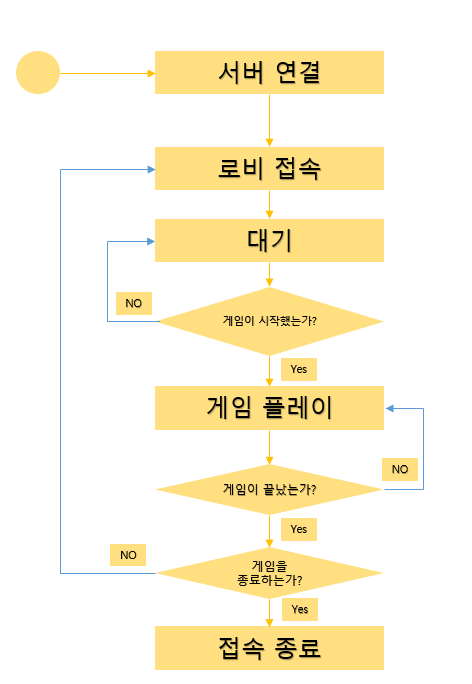
메인 서버는 메인 서버가 처리한 월드의 결과값들을 클라이언트와 에이전트들에게 송신한다.

# High-Level Design

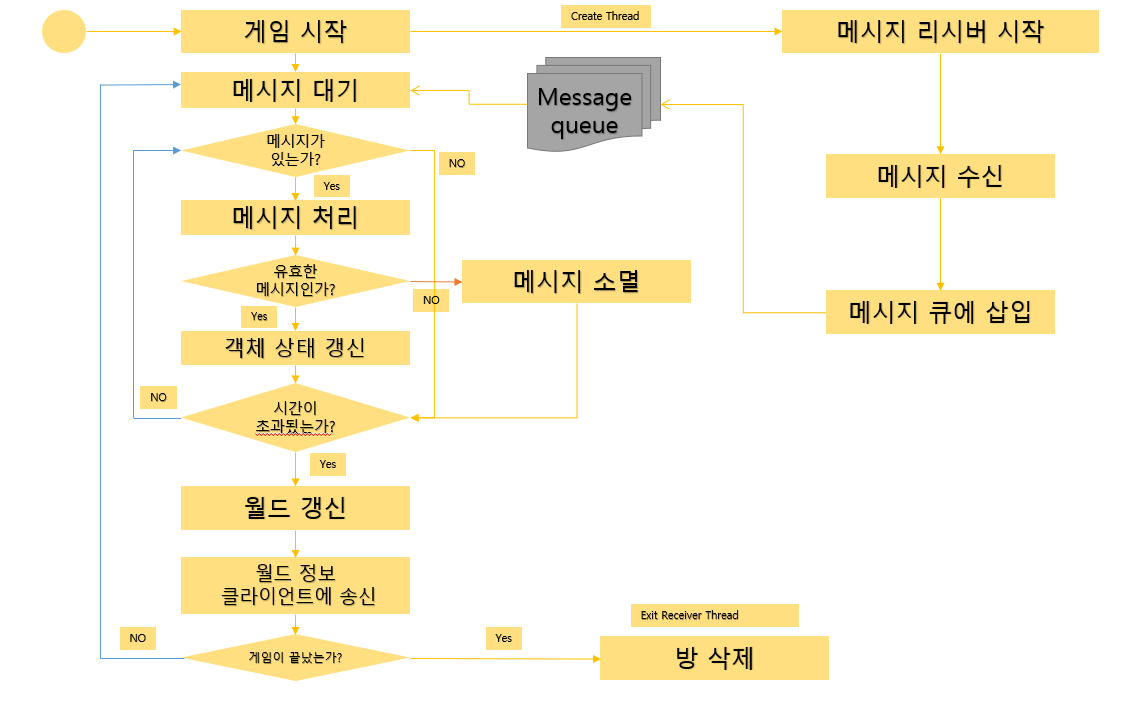
Main Server



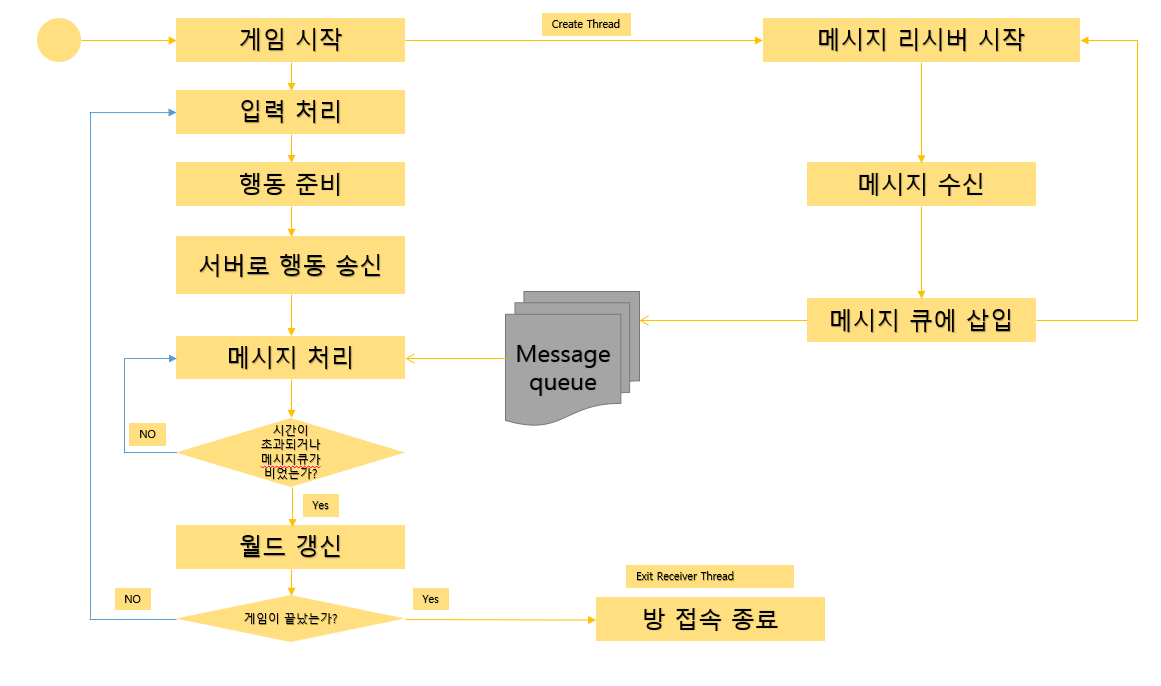
Client



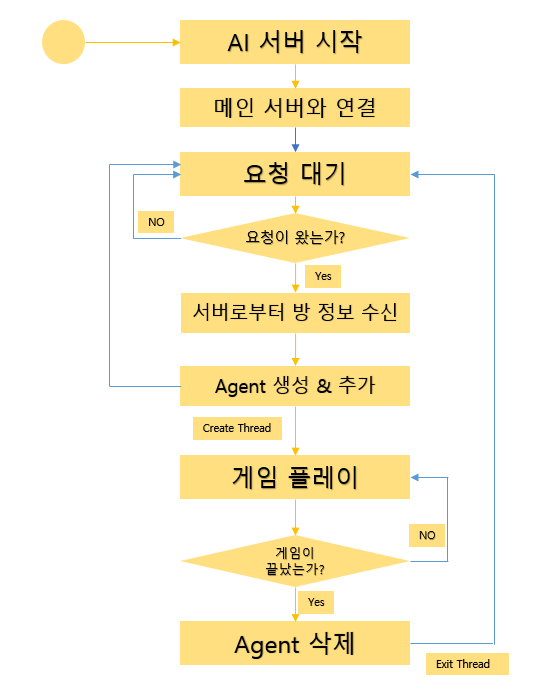
Server In Game



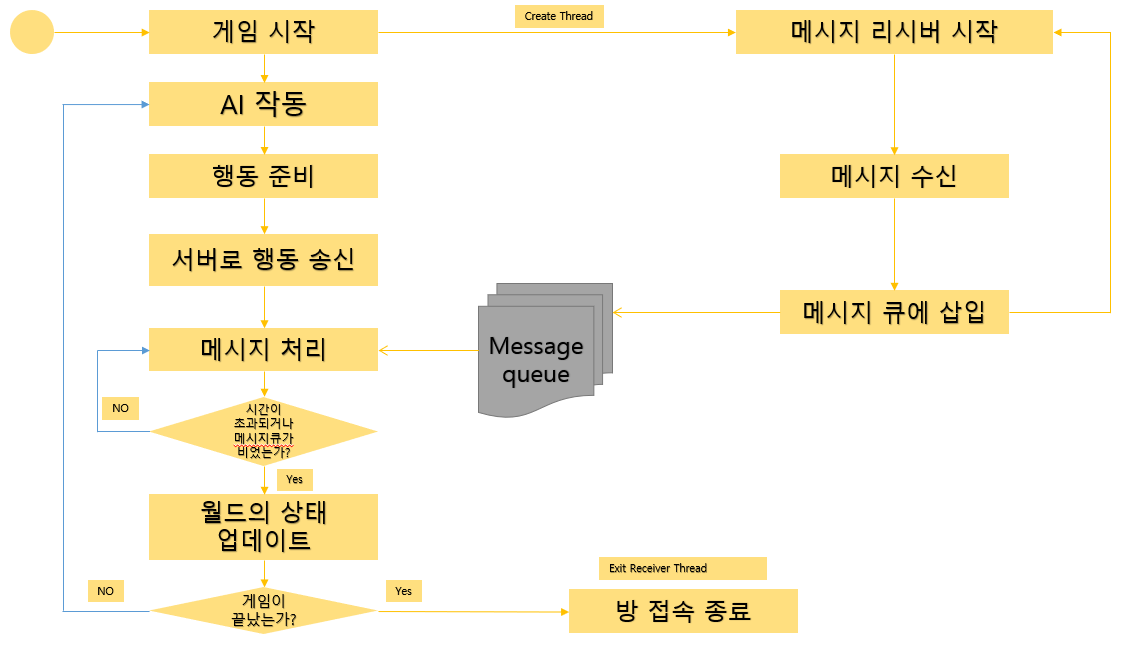
Client In Game



AI Server



AI Agent In Game



# Low-Level Design

|  |
| --- |
| Message Struct |
| Message Header |
| Detail Information |

|  |  |  |
| --- | --- | --- |
| TYPE (enum) | State | Room Creation |
| Client Game Over |
| Client Ready |
| AI Creation Request |
| AI Agent Info |
|  |  |
| Action | Move |
| Shoot |
| Build Turret |
| Reload |
|  |  |
| Update | Adjust Position |
| Create Object |
| Delete Object |
| Update Object State |

|  |  |
| --- | --- |
| Action Information | |
| Vector 2D<float> | Direction |
| unsigned int | Object ID |

|  |  |
| --- | --- |
| Object Information | |
| unsigned char | Object ID |
| Vector 2D<float> | Position |
| Vector 2D<float> | Direction |
| unsigned char | HP |
| struct | Ammunition |

|  |  |
| --- | --- |
| Ammunition | |
| unsigned char | Granade |
| unsigned char | Gun Ammo |
| unsigned char | Turret |

* 통신에 사용되는 메시지는 최상단의 표로 표현된 구조체의 형태를 따르게 된다. 메시지의 타입을 비롯한 정보를 가지고 있는 메시지 헤더와 세부정보 두 부분으로 나뉜다. 수신된 메시지 헤더부분을 읽고 해당 메시지의 세부정보를 얼마나 읽을 것인지를 정하고 나머지 메시지를 수신한다.
* 메시지의 타입은 상태(State), 행동(Action), 갱신(Update)의 분류로 나뉘게 되며 각 타입에 따라서 세부 정보에 붙는 내용이 달라지게 된다. 수신자는 이 타입에 따라서 메시지를 분류하여 처리하게 된다.
* 메시지의 헤더는 8byte로 이루어지고 각 2byte를 메시지 타입, 세부정보 크기, 방 번호, 오브젝트 번호로 구성하여 비트연산을 이용하여 정보를 저장하고 읽는다.

|  |  |  |  |
| --- | --- | --- | --- |
| 메시지 타입 | 세부정보 크기 | 방 번호 | 오브젝트 번호 |

* 메시지의 수신시 타입에 따라서 세부정보를 어떻게 읽을 것인지 정해져 있고 이 방식에 따라서 세부정보를 읽고 해당 메시지의 정보를 분할하여 각 데이터에 저장한다.

|  |  |  |
| --- | --- | --- |
|  | Type | Detail Information |
| State | Room Creation | Object Information |
| Client Game Over | Bool |
| Client Ready | Bool |
| AI Creation Request | Object Information |
| AI Agent Info | Object Information |
| Action | Move | Action Information |
| Shoot | Action Information |
| Build Turret | Action, Object Information |
| Reload | Action, Object Information |
| Update | Adjust Position | Object Information |
| Create Object |
| Delete Object |
| Update Object State |

RoomInfo

+UINT id

+ConnectedServerInfo server\_info

+std::list<ClientInfo\*> client\_list

+std::list<Agent\*> agent\_list

+GameWorld gameworld

+MsgQueue msg\_queue

+HANDLE hGameWorld

+void UpdateWorld(void)

+void SendMsgs(void)

생성된 룸의 정보를 저장하는 구조체. 룸에서 진행되는 게임에 참여중인 클라이언트, 에이전트 등 게임 진행에 필수적인 정보들을 가지고 있다. hGameWorld에는 스레드를 생성하여 RunGameWorld() 함수를 연결해 게임 메인 스레드로 이용한다.

void RoomInfo::UpdateWorld(void)

Msg\_queue에 쌓은 메시지를 처리하고 게임 월드를 업데이트 한다.

void RoomInfo::SendMsgs(void)

UpdateWorld()를 통해 생성된 새로운 정보를 룸에 소속된 객체들에게 전달한다.

ConnectedServerInfo

+SOCKET sock

+SOCKADDR\_IN addr

+HANDLE hReceiver

+MsgQueue \*pMsgQueue

서버에 연결된 클라이언트의 정보를 저장하는 구조체. hReceiver에는 새로운 스레드를 생성하여RecvMassage() 함수를 연결해 메시지를 받는 용도로 사용한다.

Agent

+UINT id

+HANDLE hReceiver

+MsgQueue \*pMsgQueue

게임에 참여중인 Agent에 사용되는 정보를 저장한 구조체. hReceiver에는 새로운 스레드를 생성하여 RecvMassage() 함수를 연결해 메시지를 받는 용도로 사용한다.

GameWorld

실제로 게임을 실행하는 게임 씬 객체. 게임을 진행하는데 필요한 모든 오브젝트들을 가지고 있으며 게임 루프를 실행하면서 로직 처리 및 상태 갱신 부분과 그리기 부분을 반복하면서 게임을 진행시킨다.

Agent와 클라이언트는 로직 처리와 상태 갱신을 MainServer에 위탁하며 클라이언트를 제외한 곳에서는 그리기 부분이 처리되지 않는다.

공용 스레드 함수

Static DWORD WINAPI RecvMassage(LPVOID arg)

CreateThread()로 생성된 스레드가 실행하는, 메시지를 받는 함수. 함수 인자 arg를 통해 전달되는 info구조체를 통해 연결된 대상으로부터 받는 메시지를 info구조체에 연결된 공유 메시지 큐에 저장한다. (스레드 동기화 필요, RoomInfo구조체 제외)

static DWORD WINAPI RunGameWorld (LPVOID arg)

CreateThread()로 생성된 스레드가World의 게임 루프를 실행하여 게임을 진행한다. 함수 인자 arg에는 Gameworld 객체가 포함되어 있어야한다.

Server

#WSADATA m\_Wsa

#SOCKET m\_Listen\_sock

#SOCKADDR\_IN m\_Serveraddr

#MsgQueue m\_MsgQueue

+void Initialize(void)

+void Release(void)

+virtual void Run(void) = 0

서버의 가장 기본적인 구성요소들과 메서드들을 가지고 있는 서버 인터페이스 객체.

이 객체를 상속받는 자식 서버 객체들의 기본적인 틀을 정해준다.

void Server::Initialize(void)

WSAStartup(), socket(), bind(), listen() 함수들을 이용하여 멤버변수들을 사용 가능한 상태로 만들어 accept()를 호출할 수 있도록 준비한다.

void Server::Release(void)

closesocket(), WSACleanup() 함수들을 호출하여 멤버변수 listen 소켓을 닫고 윈속을 종료한다.

virtual void Server::Run(void)=0

서버가 서비스를 시작하게 하는 완전 추상화 가상 함수이다.

상속받는 자식 서버객체에서 정의된다.

MsgQueue

서버-서버 또는 서버-클라이언트 간의 통신을 할 때 전달되는 메시지들을 도착한 순서대로 enqueue하고 메시지를 처리할 때는 가장 먼저 enqueue된 메시지부터 dequeue하여 처리하는 용도로 사용되는 컨테이너이다.

상속 : public Server

MainServer

-std ::list<RoomInfo\*> m\_Room\_list

-std ::list<ClientInfo\*> m\_WaitingClient\_list

-ConnectedServerInfo m\_AgentServer

+virtual void Run(void) override

+void ConnectAgentServer(void)

+void RequestAddAgentsToRoom(void)

+void CreateRoom(void)

+void DeleteRoom(void)

virtual void MainServer::Run(void) override

서버가 서비스하기 시작하는 함수. High level design에 정의 되어있는 순서도를 따라 로직이 수행된다.

void MainServer::ConnectAgentServer(void)

콘솔 창에서 AgentServer의 IP주소를 입력 받아 ConnectedServerInfo구조체멤버 변수를 이용하여 MainServer와 AgentServer를 연결하는 일을 수행한다.

void MainServer::RequestAddAgentsToRoom(void)

AgentServer에 게임을 플레이하는데 필요한 Agent들을 생성하여 룸에 추가하는 것을 요청한다.

void MainServer::CreateRoom(void)

게임을 시작할 준비가 된 클라이언트의 수가 3 이상이 되었을 때 게임을 플레이할 룸을 생성한다. 이 때 게임을 플레이할 클라이언트를 대기 리스트에서 룸의 참가자 리스트로 이동시킨다.

void MainServer::DeleteRoom(void)

플레이중이던 게임이 끝나면 호출되는 함수.

룸에 소속되 있었던 클라이언트들을 대기리스트로 옮긴 후 게임에 사용된 모든 자원을 사용 해제한다.

상속 : public Server

AgentServer

-ConnectedServerInfo m\_MainServer

-std::list<RoomInfo\*> m\_Room\_list

+virtual void Run(void) override

+void AcceptMainServer(void)

+void CreateAgentsToRoom(void)

+void DeleteAgentsFromRoom(void)

virtual void AgentServer::Run(void) override

서버가 서비스하기 시작하는 함수. High level design에 정의 되어있는 순서도를 따라 로직이 수행된다.

void AgentServer::AcceptMainServer(void)

Run()함수의 초반부에서 호출되며 메인 서버의 연결 요청을 기다린다. 메인 서버가 연결되면 recv 스레드가 생성된다.

void AgentServer::CreateAgentsToRoom(void)

룸의 생성을 알리고 에이전트의 생성을 요청하는 메시지를 받으면 생성된 룸의 정보를 바탕으로 룸 리스트에 룸을 추가하고 에이전트들을 생성하여 정보를 메인 서버에 전달한다.

void AgentServer::DeleteAgentsFromRoom(void)

게임이 종료된 룸이 있으면 호출되는 함수. 게임 종료를 알리는 메시지를 받아 게임이 종료된 룸에 사용되던 시스템 자원을 회수하고 룸을 삭제한다.

Client

-UINT m\_Local\_id

-SOCKET m\_Local\_sock

-SOCKADDR\_IN m\_Local\_addr

-MsgQueue m\_MsgQueue

-ConnectedServerInfo m\_MainServer

+void Initialize(void)

+void Release(void)

+void ConnectServer(void)

+void SendMsgs(void)

void Client::Initialize(void)

WSAStartup(), socket() 함수들을 이용하여 멤버변수들을 사용 가능한 상태로 만들어 connect()를 호출할 수 있도록 준비한다.

void Client::Release(void)

closesocket(), WSACleanup() 함수들을 호출하여 멤버변수 m\_Local\_sock을 닫고 윈속을 종료한다.

void Client::ConnectServer(void)

함수가 호출되면 메인 서버에 연결 요청을 하며, 연결이 성공하면 recv 스레드가 생성된다.

void Client::SendMsgs(void)

연결되어있는 서버에 생성된 메시지를 전송한다.

# 팀원 별 역할 분담

강수빈 – 메인 서버, PM

김병진 – AI 서버, 에이전트, 리소스

이상기 – 클라이언트 전반

# 개발 환경

|  |  |
| --- | --- |
| 개발 PC 운영체제 | Microsoft Windows 10 |
| 개발 컴파일러 | Visual Studio Community 2017 |
| 개발 언어 | C / C++ |
| 개발 그래픽 라이브러리 | Direct2D |
| 형상관리 | GitHub  https://github.com/Lyricia/2017-Network-Game-Programming |

## 테스트 환경

강수빈

|  |  |
| --- | --- |
| OS | Windows 10 |
| CPU | Intel I7 - 6700K |
| GPU | GTX 1080TI |
| RAM | 16 GB |

김병진

|  |  |
| --- | --- |
| OS | Windows 10 |
| CPU | Intel I7 - 6820HK |
| GPU | GTX 980M |
| RAM | 16 GB |

이상기

|  |  |
| --- | --- |
| OS | Windows 10 |
| CPU | Intel I7 - 6700HQ |
| GPU | GTX 960M |
| RAM | 16 GB |

## 개발 일정

## 11/9~11/15

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11/9~11/15 | | 11/9 | 11/10 | 11/11 | 11/12 | 11/13 | 11/14 | 11/15 |
| 강수빈 | |  |  |  |  |  |  |  |
| 김병진 | |  |  |  |  |  |  |  |
| 이상기 | |  |  |  |  |  |  |  |
| 11/9 | 강수빈 | 서버 기본 구조 구현 | | | | | | |
| 김병진 | 게임에 필요한 그래픽 리소스 [ 오브젝트 ] 제작 | | | | | | |
| 이상기 |  | | | | | | |
| 11/10 | 강수빈 | 서버 기본 구조 구현 | | | | | | |
| 김병진 | 에이전트 상태기계 테스트를 위한 에이전트 서버 프레임워크 개발 | | | | | | |
| 이상기 | 게임 어플리케이션 프레임워크 및 필요한 오브젝트 클래스 작성 | | | | | | |
| 11/11 | 강수빈 | 로비, 접속 대기열 구현 | | | | | | |
| 김병진 | 에이전트 클래스 구현 | | | | | | |
| 이상기 | 플레이어 input processing 부분 작성 및 오브젝트 상호작용 구현 | | | | | | |
| 11/12 | 강수빈 | 메시지 큐 구조체 제작 | | | | | | |
| 김병진 | 상태기계 구동 테스트를 위한 단순 패턴 에이전트 구현 | | | | | | |
| 이상기 | RoomInfo 구조체에 들어갈 GameWorld객체 구현 | | | | | | |
| 11/13 | 강수빈 | 메시지 큐 제작 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 | MsgQueue 제작 | | | | | | |
| 11/14 | 강수빈 | 스레드 이용 다중 메시지 큐 적용 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 |  | | | | | | |
| 11/15 | 강수빈 | 메시지 송신기 구현 | | | | | | |
| 김병진 | 에이전트 서버 메시지 큐 제작 | | | | | | |
| 이상기 |  | | | | | | |

## 11/16~11/22

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11/16~11/22 | | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 | 11/21 | 11/22 |
| 강수빈 | |  |  |  |  |  |  |  |
| 김병진 | |  |  |  |  |  |  |  |
| 이상기 | |  |  |  |  |  |  |  |
| 11/16 | 강수빈 | 메시지 송신기 구현 | | | | | | |
| 김병진 | 에이전트 서버 메시지 구조체 제작 | | | | | | |
| 이상기 |  | | | | | | |
| 11/17 | 강수빈 |  | | | | | | |
| 김병진 | 메시지 리시버 테스트를 위한 간단한 오브젝트 클래스 구현 | | | | | | |
| 이상기 | Client 객체 구현 | | | | | | |
| 11/18 | 강수빈 | 메시지 리시버 구현 및 연결 | | | | | | |
| 김병진 | 메시지 리시버 구현 및 메인 서버와 연결 | | | | | | |
| 이상기 | Static DWORD WINAPI RecvMassage(LPVOID arg) 구현 및 서버와 연결 | | | | | | |
| 11/19 | 강수빈 | 룸 구조체 구현 | | | | | | |
| 김병진 | 터렛 상태기계와 자동 에이전트 상태기계 구현 | | | | | | |
| 이상기 | 기본적인 게임 어플리케이션 구현 완료. | | | | | | |
| 11/20 | 강수빈 | 멀티스레딩 적용 다중 방 생성 및 작동 확인 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 | Client객체에 MsgQueue 적용 | | | | | | |
| 11/21 | 강수빈 | 멀티스레딩 적용 다중 메시지 리시버 생성 및 작동 확인 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 |  | | | | | | |
| 11/22 | 강수빈 | 방 구조체 및 메시지 리시버 통합 | | | | | | |
| 김병진 | 메인서버로부터 받은 요청을 통한 에이전트 룸 생성 구현 | | | | | | |
| 이상기 |  | | | | | | |

## 11/23~11/29

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11/23~11/29 | | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 | 11/28 | 11/29 |
| 강수빈 | |  |  |  |  |  |  |  |
| 김병진 | |  |  |  |  |  |  |  |
| 이상기 | |  |  |  |  |  |  |  |
| 11/23 | 강수빈 | 방 구조체 및 메시지 리시버 통합 | | | | | | |
| 김병진 | 에이전트 서버와 메인 서버간 동기화 테스트 후 문제점 보완 | | | | | | |
| 이상기 |  | | | | | | |
| 11/24 | 강수빈 | 클라이언트와 연결, 메시지 통신테스트, 디버그 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 | MainServer와 Client의 메시지 송수신 테스트 | | | | | | |
| 11/25 | 강수빈 |  | | | | | | |
| 김병진 | 메인 서버에서 월드 정보를 받아오기 위한 월드 오브젝트 구현 | | | | | | |
| 이상기 | Client객체에 적용된 MsgQueue 기능 테스트 | | | | | | |
| 11/26 | 강수빈 | AI 서버와 연결, 메시지 통신 테스트, 디버그 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 | MsgQueue 스레드 동기화 완료 | | | | | | |
| 11/27 | 강수빈 | AI 서버와 연결, 메시지 통신 테스트, 디버그 | | | | | | |
| 김병진 | 메인 서버에서 전송 받은 메시지 처리 구현 | | | | | | |
| 이상기 | 여러 개의 Client를 멀티 스레드로 구동 테스트 | | | | | | |
| 11/28 | 강수빈 | 클라이언트, AI서버, 메인서버간 메시지 송수신 체크 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 |  | | | | | | |
| 11/29 | 강수빈 | 클라이언트, AI서버, 메인서버간 메시지 송수신 체크 | | | | | | |
| 김병진 | 룸스레드 생성 삭제 반복 테스트를 통한 버그 탐색 | | | | | | |
| 이상기 |  | | | | | | |

## 11/30~12/6

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11/30~12/6 | | 11/30 | 12/1 | 11/2 | 11/3 | 11/4 | 11/5 | 11/6 |
| 강수빈 | |  |  |  |  |  |  |  |
| 김병진 | |  |  |  |  |  |  |  |
| 이상기 | |  |  |  |  |  |  |  |
| 11/30 | 강수빈 | 클라이언트, AI서버, 메인서버간 메시지 송수신 체크 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 |  | | | | | | |
| 12/1 | 강수빈 | 물리처리 구현 | | | | | | |
| 김병진 | 게임에 필요한 그래픽 리소스 [ UI ] 제작 | | | | | | |
| 이상기 | Agent 객체 및 추가 리소스를 GameWorld에 추가 | | | | | | |
| 12/2 | 강수빈 | 물리처리 구현 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 | 에이전트를 포함한 모든 오브젝트 상태 갱신 테스트 | | | | | | |
| 12/3 | 강수빈 |  | | | | | | |
| 김병진 | 룸 스레드 함수에서 다수의 에이전트 생성 및 구동 구현 | | | | | | |
| 이상기 | 여러 개의 Client객체 구동으로 MainServer의 RoomInfo List 테스트 | | | | | | |
| 12/4 | 강수빈 | 클라이언트와 연결, 작동테스트, 디버그 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 | MainServer - Client의 로컬 통신 기능 테스트 | | | | | | |
| 12/5 | 강수빈 | AI 서버와 연결, 작동테스트, 디버그 | | | | | | |
| 김병진 | 메인 서버 전송 데이터를 통한 에이전트 월드 갱신 최적화 | | | | | | |
| 이상기 |  | | | | | | |
| 12/6 | 강수빈 | 클라이언트와 AI서버 연결, 작동테스트, 디버그 | | | | | | |
| 김병진 |  | | | | | | |
| 이상기 |  | | | | | | |

## 12/7~11/13

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12/7~12/13 | | 12/7 | 12/8 | 12/9 | 12/10 | 12/11 | 12/12 | 12/13 |
| 강수빈 | |  |  |  |  |  |  |  |
| 김병진 | |  |  |  |  |  |  |  |
| 이상기 | |  |  |  |  |  |  |  |
| 12/7 | 강수빈 | 클라이언트와 AgentServer 연결, 작동테스트, 디버그 | | | | | | |
| 김병진 |
| 이상기 |
| 12/8 | 강수빈 | 클라이언트와 AgentServer 연결, 작동테스트, 디버그 | | | | | | |
| 김병진 |
| 이상기 |
| 12/9 | 강수빈 | 로컬이 아닌 상태에서 Client와 AgentServer연결, 작동테스트, 디버그 | | | | | | |
| 김병진 |
| 이상기 |
| 12/10 | 강수빈 | 로컬이 아닌 상태에서 Client와 AgentServer연결, 작동테스트, 디버그 | | | | | | |
| 김병진 |
| 이상기 |
| 12/11 | 강수빈 | 최종 점검 | | | | | | |
| 김병진 |
| 이상기 |
| 12/12 | 강수빈 | 최종 점검 | | | | | | |
| 김병진 |
| 이상기 |
| 12/13 | 강수빈 | 최종 점검 | | | | | | |
| 김병진 |
| 이상기 |