네트워크 게임 프로그래밍
프로젝트 추진 계획서

2022182035 임채은(팀장)
2022180039 임수영
2022184031 최미나

목차

1. 애플리케이션 기획

A. 게임 소개

이름	대초원의 모험
작업자	최미나, 임수영 (윈도우 프로그래밍)
장르	2D 탑뷰 아케이드 게임
참고한 게임	스타듀밸리 대초원 왕의 모험

B. 수정 및 추가할 내용

- LobbyScene, GameNetwork 클래스 추가
- ESC 입력 시 이전 Scene으로 돌아가기 기능 추가

C. 조작 Key

WASD	상하좌우 이동
방향키	총알 발사 방향
Space Bar	아이템 사용
Shift	폭탄 발로 차기

D. Monster와 Item

[Monster]

- 상하좌우 4방향에서 랜덤하게 생성
- 처치 시 효과가 나오고 확률적으로 아이템을 떨어뜨림
- 플레이어와 충돌시 플레이어의 목숨 -1 후 플레이어가 화면 중앙에서 부활함
- 몬스터 종류 및 특징

종류	특징	사망 조건
일반 몬스터	X	총알 1번 맞음
부활 몬스터	총알을 한 번 맞으면 기절하고 부활함	부활 후 총알 1번 맞음
탱커 몬스터	덩치가 큼	총알 5번 맞음
설치형 몬스터	일정 시간이 지나면 장애	장애물 변신 전 1번,

	물로 변함	장애물 변신 후 3번 맞음
폭탄 몬스터	시간 간격을 두고 폭탄 설치, 설치 시 정지함	총알 1번 맞음

[Item]

- 몬스터 처치 시 얻는 아이템

아이템	능력(일시)
캐릭터 얼굴	목숨 +1
탄창	공속 증가
벼락	게임 화면 내 모든 몬스터 처치
물레바퀴	모든 방향으로 총알 발사
커피	이동 속도 증가
총(샷건)	한 방향으로 총알 3개씩 발사
시계	몬스터 정지 시키기

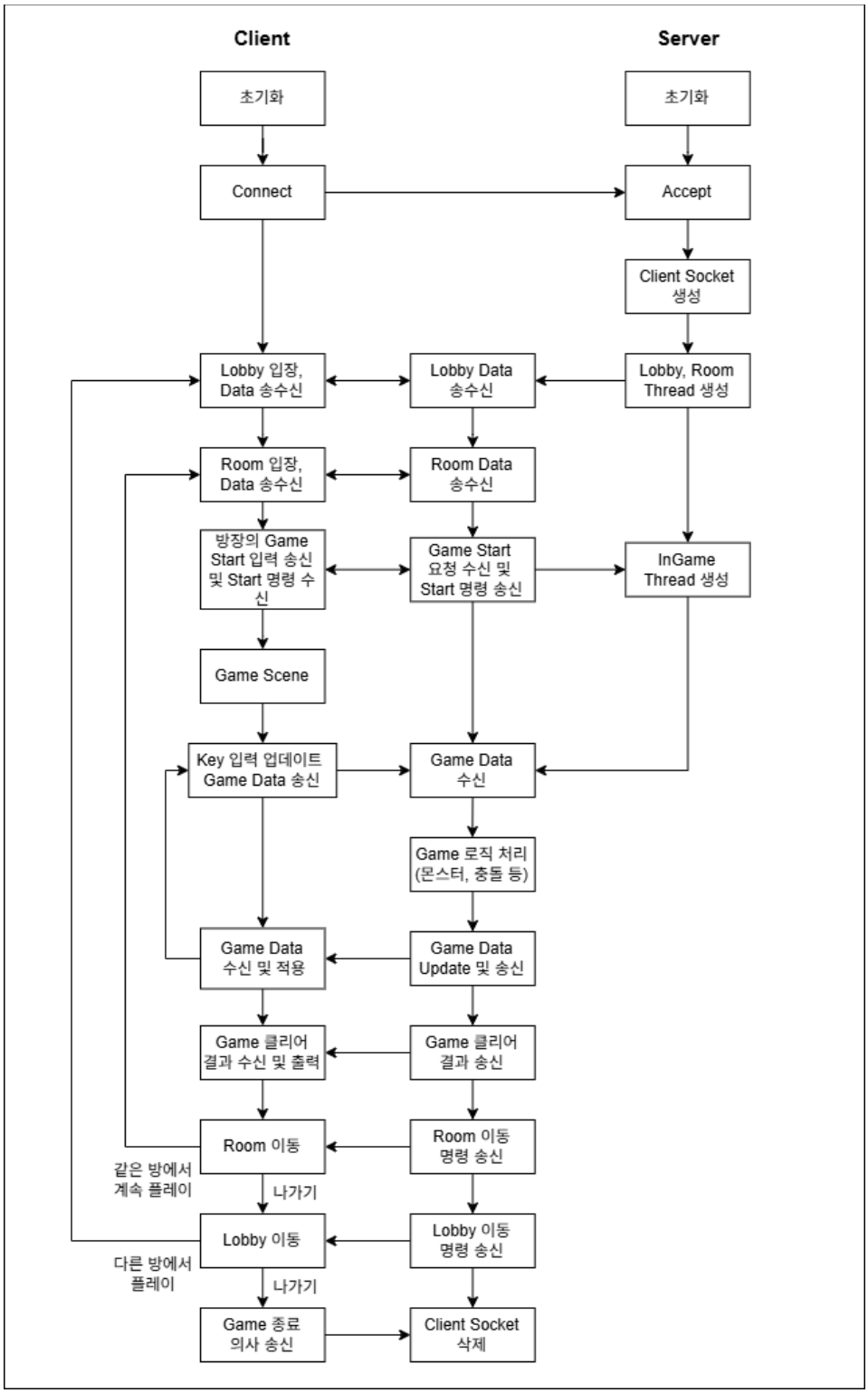
E. 스테이지

- 총 4개의 스테이지
- 몬스터를 다 잡지 못하면 다음 스테이지로 넘어갈 수 없음
- 스테이지마다 각각 다른 장애물 존재
- 장애물: 나무 장애물, 설치형 몬스터, 굴러다니는 선인장(플레이어와 충돌 시 플레이어 목숨 -1)

스테이지 1	스테이지 2
	
스테이지 3	**스테이지 4**
	

2. High-level Design

A. 플로우 차트



B. 서버와 클라이언트 흐름

1) 서버와 클라이언트의 연결

[서버]

번호	흐름	설명
1	Winsock 초기화	WSAStartup()을 호출해 소켓 라이브러리 초기화
2	Socket 생성	대기 소켓 생성
3	bind()	서버의 IP 주소와 포트를 socket에 연결
4	listen()	대기 소켓을 생성하고 대기 상태로 전환
5	accept()	클라이언트가 connect 요청하면, 새로운 전용 socket을 생성

[클라이언트]

번호	흐름	설명
1	Winsock 초기화	WSAStartup()을 호출
2	서버 연결	• 서버의 IP 주소와 포트 번호 입력 • connect()를 호출해 서버에 접속 요청을 보냄

2) Lobby와 Room

[서버]

번호	흐름	설명
1	Room List 관리	존재하는 Room의 정보를 관리(Room ID, 인원, 방장의 ID)
2	Room List 송신	클라이언트가 로비에 입장하면, Room List를 보냄
3	Room 입장	클라이언트가 입장할 Room ID를 보내면, 해당 Room에 Player를 추가 (한 Room의 최대 인원은 3명)
4	Room에서 게임 시작 송신	방장이 Start 버튼을 누르면, 서버는 해당 Room의 모든 클라이언트에게 게임 시작 메세지를 전송

[클라이언트]

번호	흐름	설명
1	Lobby 입장	서버로부터 Room List를 받고, 화면 출력
2	Room 선택	입장할 Room을 선택하여 서버에 Room Id 전송
3	Room 입장	서버로부터 확인 메시지를 받고 Room Scene으로 전환
4	시작 대기	방장이 Start를 누를 때까지 기다림

3) Game Scene

[서버]

번호	흐름	설명
1	플레이어 데이터 수신	• 각 클라이언트의 정보를 수신(Object Type, 위치, 속도 등) • 게임 데이터를 업데이트
2	오브젝트 동기화	서버는 클라이언트에게 주기적으로 게임 데이터를 보내 동기화
3	몬스터 AI	몬스터 로직 처리 후 클라이언트에게 송신
4	아이템 및 충돌 처리	아이템 사용, 충돌 확인 등 이벤트 발생 시 데이터를 송신
5	스테이지 클리어	몬스터를 다 처치하면, 다음 스테이지로 전환하라는 메시지를 보냄

[클라이언트]

번호	흐름	설명
1	플레이어 조작	키 입력에 따른 위치, 방향, 상태 등을 서버로 보냄
2	오브젝트 동기화	서버에서 받은 정보로 다른 플레이어 및 오브젝트 동기화
3	스테이지 클리어	스테이지 클리어 여부 수신

4) 결과 및 Room 복귀

[서버]

번호	흐름	설명
1	게임 클리어	모든 스테이지 완료 시 메시지 전송
2	플레이어가 나갔을 때	• 어떤 클라이언트가 종료했는지 id를 보냄 • 남은 플레이어에게 떠난 플레이어를 알려줌

[클라이언트]

번호	흐름	설명
1	게임 클리어 수신	클리어 메시지를 받으면, 결과 출력 후 기존 Room으로 복귀
2	떠난 플레이어 반영	떠난 플레이어 정보를 받으면 해당 Room에서 해당 플레이어 삭제
3	이어하기 or Lobby로 나가기	자동으로 Room으로 복귀 후 Lobby로 나가거나 같은 Room에서 다시 플레이

3. Low-level Design

A. 패킷

• 클라이언트 패킷

```
C_EnterRoom
{
	int roomID;
}

C_LeaveRoom
{
	int roomID;
}

C_StartGame
{
	int roomID;
}

C_EndGame
{
	int roomID;
}

C_UpdateObjectState
{
	int objectID;
	ObjectType type;
	ObjectState state;
}

C_UpdateDir
{
	int objectID;
	ObjectType type;
	Dir dir;
}

C_Move
{
	int objectID;
	ObjectType type;
	Vertex pos;
}

C_AddRoom
{
	int roomID;
}

C_RemoveRoom
{
	int roomID;
}

C_Collision
{
	CollisionType collisionType;
	int objectID1;
```

```
        ObjectType type1;
        Vertex pos1;
        int objectID2;
        ObjectType type2;
        Vertex pos2;
}

C_UseItem
{
        int objectID;
        ObjectType itemType;
}
```

- **서버 패킷**

```
S_SendRoomList
{
        int roomID[roomCount];
        std::string roomTitle[roomCount];
        RoomState state[roomCount];
        int playerNum;
}

S_EnterRoomResult
{
        bool result;
}

S_AddObject
{
        int objectID;
        ObjectType type;
        Vertex pos;
}

S_RemoveObject
{
        int objectID;
        ObjectType type;
}

S_UpdateRoomState
{
        int roomID;
        RoomState state;
}

S_UpdateObjectState
{
        int objectID;
        ObjectType type;
        ObjectState state;
}

S_UpdateDir
{
        int objectID;
        ObjectType type;
        Dir dir;
}
```

```
S_Move
{
	int objectID;
	ObjectType type;
	Vertex pos;
}

S_ChangeNextStage
{
	int stageNum;
}

S_AddRoom
{
	int roomID;
	RoomState state;
}

S_RemoveRoom
{
	int roomID;
}

S_CollisionResult
{
	bool result;
}

S_MonsterDamaged
{
	int objectID;
	ObjectType type;
	int monsterHP;
}

S_ItemUseResult
{
	bool result;
}
```

B. 서버

```
class ServerFramework
{
public:
	ServerFramework();
	~ServerFramework();

public:
	void Update();
	void ProcessSend();
	void ProcessRecv();

public:
	void AddRoom();
	void RemoveRoom();

public:
	template <class T>
	std::vector<char*> CreatePacket();
};
```

```
class Room
{
public:
		Room();
		~Room();

public:
		void Update();

public:
		void EnterRoom();
		void LeaveRoom();

private:
		void AddObject();
		void RemoveObject();

public:
		void SetID();
		int GetID();
		void SetState();
		RoomState GetState();
};

class Player : public GameObject {
public:
		Player();
		virtual void Update();
}
```

```
class State {
public:
		virtual void Enter(GameObject* object) = 0;
		virtual void Exit(GameObject* object) = 0;
		virtual void Doing(GameObject* object) = 0;
};

class MoveToTarget : public State {
public:
		virtual void Enter(GameObject* object) override;
		virtual void Exit(GameObject* object) override;
		virtual void Doing(GameObject* object) override;

};

class SetTarget : public State {
public:
		virtual void Enter(GameObject* object) override;
		virtual void Exit(GameObject* object) override;
		virtual void Doing(GameObject* object) override;
};

class Bomb : public State {
public:
		virtual void Enter(GameObject* object) override;
		virtual void Exit(GameObject* object) override;
		virtual void Doing(GameObject* object) override;
};

class Dead : public State {
public:
		virtual void Enter(GameObject* object) override;
		virtual void Exit(GameObject* object) override;
		virtual void Doing(GameObject* object) override;
```

```
};

class UseItem : public State {
public:
	virtual void Enter(GameObject* object) override;
	virtual void Exit(GameObject* object) override;
	virtual void Doing(GameObject* object) override;
};

class StateMachine {
public:
	StateMachine(GameObject* object);
	~StateMachine();
	void Start();
	void Update();

	void ChangeState(State* state);
private:
	GameObject* _object{};
	State* _curState{};
};

enum class ObjectType
{
	Player,
	Monster,
	Button,
	Item,
};

struct Vertex
{
	int x, y;
};

class GameObject {
public:
	GameObject();
	~GameObject();
	virtual void Update();
	virtual void Move();
	virtual void FindTarget();

	void SetObjectType(ObjectType type);
	ObjectType GetObjectType();
	void SetPos(Vertex pos);

	StateMachine* GetStateMachine();
	Vertex GetTargetPos();
	void SetTargetPos(Vertex target);
	Vertex GetPos();
}

class Monster : public GameObject {
public:
	Monster();
	void Move() override;
	void Update() override;
};

class TankMonster : public Monster {
public:
	TankMonster();
	void FindTarget() override;
};
```

```
class RespawnMonster : public Monster {
public:
          RespawnMonster();
          void FindTarget() override;
};

class ObstacleMonster : public Monster {
public:
          ObstacleMonster();
          void FindTarget() override;
};

class NormalMonster : public Monster {
public:
          NormalMonster();
          void FindTarget() override;
};

class BomberMonster : public Monster {
public:
          BomberMonster();
          void FindTarget() override;
};

enum class ItemType
{
          Life,
          Magazine,
          Lightning,
          Waterwheel,
          Coffee,
          Shotgun,
          Hourglass,
};

class Item : public GameObject {
public:
          Item(ItemType);
          virtual void Update();
};

class Bomb : public GameObject {
public:
          virtual void Update();
};

class Projectile : public GameObject {
public:
          virtual void Update();
          virtual void Move();
};
```

C. 클라이언트

```
class GameNetwork
{
public:
          GameNetwork();
          ~GameNetwork();

public:
```

```
        void Update();

private:
        void ProcessSend();
        void ProcessRecv();

public:
        template <class T>
        std::vector<char*> CreatePacket();
};

class LobbyScene : public Scene
{
public:
        LobbyScene();
        virtual ~LobbyScene();

public:
        virtual void Update();
        virtual void Render();

public:
        void AddRoom();
        void RemoveRoom();
        void ProcessInput() override;
};

class GameObject {
public:
        void SyncData(); // 변경된 정보를 GameNetwork 에 알림
};
```

D. 멀티 스레드 함수

```
DWORD WIMAPI InGameThread()
DWORD WINAPI AcceptThread()
```

4. 팀원 간 역할 분담

임채은	• 클라이언트/서버 연동을 위한 클라이언트 코드 추가 GameNetwork::GameNetwork() GameNetwork::~GameNetwork() GameNetwork::Update() GameNetwork::ProcessSend() GameNetwork::ProcessRecv() LobbyScene::LobbyScene() LobbyScene::~LobbyScene() LobbyScene::Update() LobbyScene::Render() LobbyScene::AddRoom() LobbyScene::RemoveRoom() LobbyScene::ProcessInput() GameObject::SyncData()
최미나	• 멀티 스레드 동기화 TankMonter::FindTarget() NormalMonter:: FindTarget() ObstacleMonter:: FindTarget() RespawnMonter:: FindTarget() BomberMonter:: FindTarget() MoveToTarget::Enter, Exit, Doing SetTarget:: Enter, Exit, Doing Bomb:: Enter, Exit, Doing Dead:: Enter, Exit, Doing UseItem:: Enter, Exit, Doing StateMachine::StateMachine(GameObject* object); StateMachine::~StateMachine(); StateMachine::Start(); StateMachine::Update(); StateMachine::ChangeState(State* state); GameObject::GameObject(); GameObject::Move(); GameObject::FindTarget(); Player::Player(); Player::Update(); DWORD WIMAPI InGameThread() Item::Item(ItemType)

	Item::Update(); Bomb::Update() Projectile::Update() Projectile::Move()
임수영	• 서버/클라 연동 및 ServerFramework, Room 구현 ServerFramework::ServerFramework() ServerFramework::~ServerFramework() ServerFramework::Update() ServerFramework::ProcessSend() ServerFramework::ProcessRecv() ServerFramework::AddRoom() ServerFramework::RemoveRoom() ServerFramework::CreatePacket() Room::Room() Room::~Room() Room::Update() Room::EnterRoom() Room::LeaveRoom() Room::AddObject() Room::RemoveObject() Room::SetID Room::GetID Room::SetState() Room::GetState() DWORD WINAPI AcceptThread()

5. 개발 환경

작업 IDE	Visual Studio 2022
버전 관리	Github
개발 언어	C++
네트워크 프로토콜 및 API	TCP/IP, Winsock
입출력(I/O) 모델	Select
코딩 컨벤션	• 변수 이름: camelCase, 멤버 변수 앞에 _ 붙이기 • 함수 이름: PascalCase • 클래스 이름: PascalCase

6. 개발 일정 ■: 임수영 ■: 최미나 ■: 임채은

2025년 11월

월	화	수	목	금	토	일
					1	2
						DWORD WINAPI AcceptThread()
					TankMonter, NormalMonter, ObstacleMonter, RespawnMonter, BomberMonter 클래스의 FindTarget(),	[GameObject] GameObject(), Move(), FindTarget(), [Projectile] Update(), Move()
					[GameNetwork] GameNetwork() – connect 연결 요청 ~GameNetwork(), Update() [LobbyScene] LobbyScene(), ~LobbyScene()	[GameNetwork] GameNetwork() – connect 연결 요청 ~GameNetwork(), Update() [LobbyScene] LobbyScene(), ~LobbyScene()

3	4	5	6	7	8	9
	[ServerFramework] ServerFramework(), ~ServerFramework(), Update() [Room] Room(), ~Room					[ServerFramework] Update(),ProcessSend(), CreatePacket()
	MoveToTarget, SetTarget, Bomb, Dead, UseItem class			[Player] Player(), Update();	Player::Update();	Player::Update();
	[GameNetwork] ProcessSend(), Update(), [LobbyScene] Render(),Update()				[GameNetwork] ProcessSend(), Update(), [LobbyScene] Render(),Update()	[GameNetwork] ProcessSend(), Update(), [LobbyScene] Render(),Update()
10	11	12	13	14	15	16
	[Room] SetState(), GetState(), GetID(),SetID() [ServerFramework] AddRoom(), RemoveRoom(),					
[StateMachine] StateMachine(GameObject* object), ~StateMachine(), Start(),Update(), :ChangeState(State* state)	[Item] Item(ItemType), Update()					
	[GameNetwork] ProcessSend(), Update(), [LobbyScene] Render(),Update()					LobbyScene] AddRoom(), RemoveRoom(), ProcessInput()

17	18	19	20	21	22	23
					[ServerFramework] Update(), ProcessRecv(), ProcessSend() [Room] EnterRoom(), AddObject(), Update()	[Room] LeaveRoom(), RemoveObject(), Update()
					InGameThread()	InGameThread()
[LobbyScene] AddRoom(), RemoveRoom(), ProcessInput()	GameObject::SyncData() [GameNetwork] ProcessRecv(), Send()				GameObject::SyncData() [GameNetwork] ProcessRecv(), Send()	GameObject::SyncData() [GameNetwork] ProcessRecv(), Send()
24	25	26	27	28	29	30
	[ServerFramework] ProcessRecv(), ProcessSend()				[ServerFramework] ProcessRecv(), ProcessSend() [Room] Update()	[ServerFramework] ProcessRecv(), ProcessSend() [Room] Update()
InGameThread()	InGameThread()				InGameThread() 동기화	InGameThread() 동기화
GameObject::SyncData() [GameNetwork] ProcessRecv(), Send()		GameObject::SyncData() [GameNetwork] ProcessRecv(),			GameObject::SyncData() [GameNetwork] ProcessRecv(),	

		Send()			Send()	

2025년 12월

월	화	수	목	금	토	일
1	2	3	4	5	6	7
					테스트 및 버그 수정	테스트 및 버그 수정
					테스트 및 버그 수정	테스트 및 버그 수정
					테스트 및 버그 수정	테스트 및 버그 수정
8	9	10				
	최종 테스트					
	최종 테스트					
	최종 테스트					