# 네트워크 게임 프로그래밍 기말 프로젝트 v2

2022184049 함민규

2022182050 최제연

2022180047 김준호

## 목차



## 1. 개요

**게임**: 포트리스 (모작)

**제작자**: 함민규

**제작 과목**: 원도우 프로그래밍

**장르**: 2D 포물선 슈팅/대전형 액션

**플랫폼**: PC

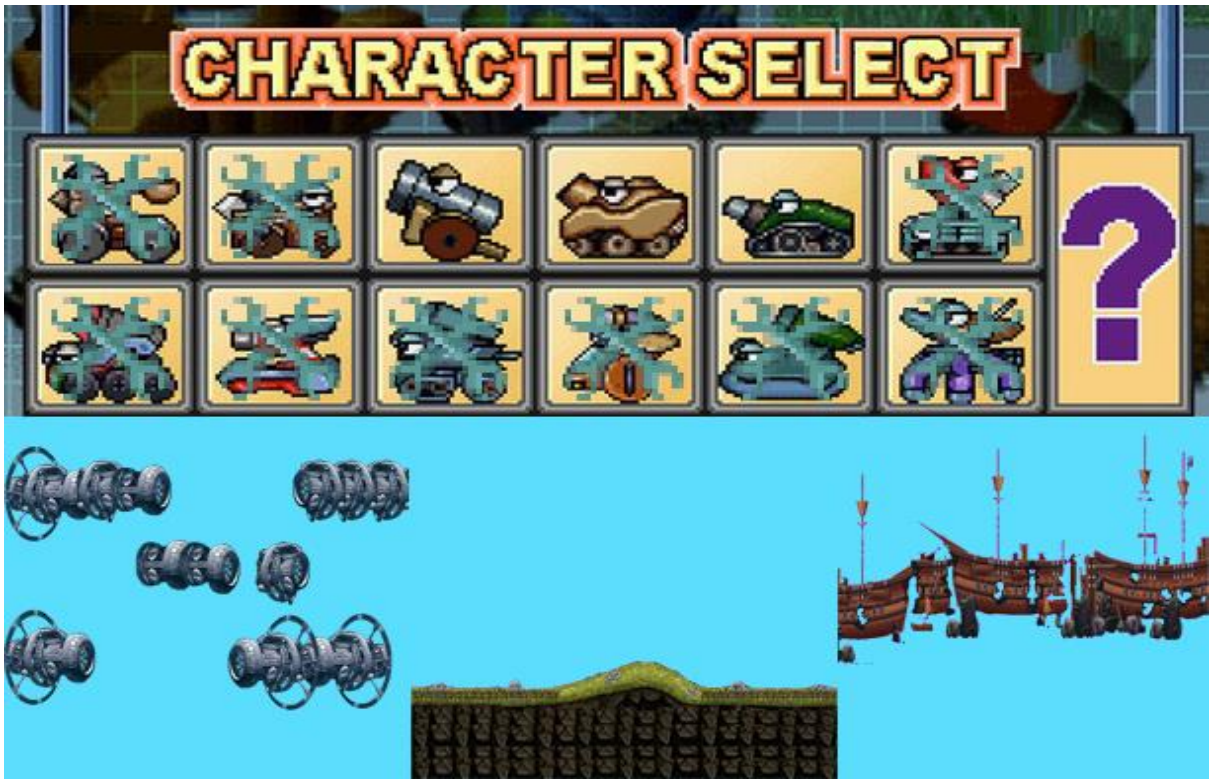
**네트워크**: TCP 소켓을 이용한 클라이언트 – 서버 구조

**개발 목표**: 포트리스의 기본 전투 시스템을 구현하고, 서버 클라이언트 통신을 통해 1:1 대전이 가능한 구조를 구현.

## 2. 애플리케이션 기획

**월드**

기본 전투 맵 3종 (지형 파괴 가능)




지형은 Tile map 기반으로 구성

플레이어, 몬스터, 아이템 존재

**플레이어**

이동: 좌우 이동

공격: 포물선 각도 조절 및 파워 조절 후 발사

체력: HP가 0이되면 패배

인벤토리 및 아이템 장착 가능

**플레이어 이동**

A, D 키: 좌우 이동

W, S 키: 각도 조절

Space: 발사

F1: 스킬 1

F2: 스킬 2

F3: 스킬 3

E: 카메라 자유시점

**몬스터 (AI)**

맵에 자신만의 턴을 가지는 AI가 등장함.

**아이템**

강화탄, 이동탄, 수리 3가지의 아이템이 존재.

**인벤토리**

스킬의 사용가능 여부 표시.

# 3. High – Level 디자인

## 3 – 1 게임 루프 구조

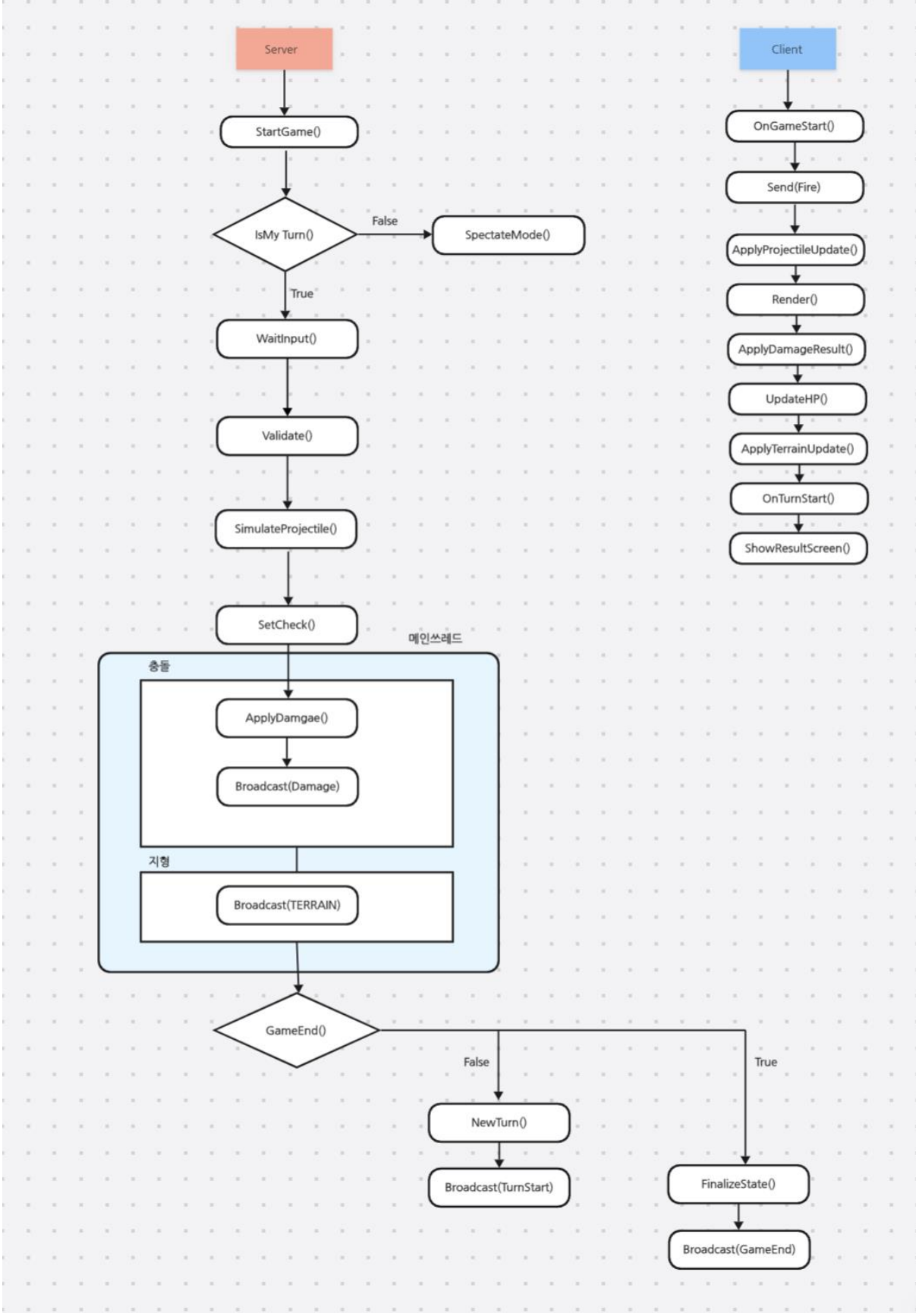
1. **로비 접속**
   A. 플레이어는 서버에 접속 후 대기실에 입장.
   B. 서버는 각 방에 대해 고유 ID를 부여, 참가자 수를 관리.
2. **방 생성 및 준비**
   A. 한 명이 방을 생성하고 나머지 플레이어가 참가.
   B. 모든 인원이 준비 완료 시 서버가 게임 세션을 생성
3. **게임 시작/초기화**
   A. 서버는 랜덤으로 턴 순서를 결정, 첫 번째 플레이어에게 공격 권한 부여.
   B. 클라이언트는 자신의 탱크 위치 및 환경을 수신.
4. **공격 단계**
   A. 공격자는 각도와 파워를 설정하여 발사 버튼 클릭

B. 해당 데이터를 서버로 송신
C. 서버는 궤적 계산 및 충돌 감지를 처리한 뒤, 포탄 이동 데이터를 UDP로 전체 브로드 캐스팅
D. 충돌이 감지되면 피해량과 파괴 좌표를 전송

5. **피해 판정 및 상태 갱신**
   A. 서버는 각 플레이어의 HP 및 지형 데이터를 업데이트
   B. 클라이언트는 이를 받아 UI 및 맵 상태를 갱신
6. **턴 전환**
   A. 다음 플레이어로 턴 이동
   B. 서버는 현재 턴 정보를 브로드 캐스트
7. **승패 판정/게임 종료**
   A. 한 쪽의 플레이어의 HP가 모두 소진되면 게임 종료
   B. 서버는 게임 결과를 모든 클라이언트에 전송 후 세션 종료

## 3 – 2 High-Level 흐름도

**흐름도**




4. Low– Level 디자인

## 4 – 1 server

### 1. 인 게임

#### IsMyTurn()

- 현재 턴이 해당 플레이어의 차례인지 검사.

#### WaitInput()

- 현재 턴인 플레이어의 입력을 기다리는 함수.

#### Validate()

- 플레이어가 보낸 입력 데이터가 올바른 범위인지 검사.

#### SimulateProjectile()

- 포탄의 궤적을 계산하기 충돌감지를 수행.

#### ApplyDamage()

- 피격 대상의 체력을 감소시킨다.
- 데미지 계산 후 남은 HP를 클라이언트에게 전송.

#### GameEnd()

- 게임 종료 조건 검사.

### 2. 스레드

#### StartUpdateThread()

- 인 게임 상태를 주기적으로 갱신

#### StartCollisionThread()

- 투사체 충돌 판정 전용 스레드 시작

#### StartTerrainThread()

- 지형 파괴 작업 처리 및 패킷 생성 스레드

## 4 – 2 Cilent

### 1. 인 게임

#### SendFire()

- 발사 입력을 서버로 전송

#### SendData()

- 갱신되는 데이터를 서버로 전송.

### 2. 스레드

#### StartClientUpdateThread()

- 인게임 오브젝트/상태를 주기적으로 갱신

#### StopClientThreads()

- 클라이언트 스레드 종료 및 리소스 정리

## 4 – 3 Protocol

**Global**

Enum PacketType: 패킷의 종류

Enum PlayerState: 플레이어 상태

#define MAX_ID_SIZE 32: ID 최대 길이

#define MAX_NAME_SIZE 32: 표시명 최대 길이

#define MAX_PLAYERS 2: 방 최대 인원

#define ROOM_ID_NONE 0: 미배정 방 아이디

#define TRUE 1/FALSE 0: BOOL 치환 상수

### 1. 로그인

**패킷 구분**

#define LOGIN_REQ 1

#define LOGIN_OK 2

#define LOGIN_FAIL 3

#define SERVER_STATUS 4

**구조체**

**클라이언트**

**Struct LoginReqPacket {}: 로그인 요청**

Char size

Char type

Char playerID[MAX_ID_SIZE]: 로그인하려는 사용자 ID

Char password[MAX_ID_SIZE]: 비밀번호 문자열

**서버**

**Struct LoginOkPacket {}: 로그인 성공**

Char size

Char type

unsigned int sessionId: 재접속 식별

**Struct LoginFailPacket {}: 로그인 실패**

Char size

Char type

Char reason[MAX_ID_SIZE]: 로그인 실패 사유

**Struct ServerStatusPacket {}: 서버 상태**

Char size

Char type

unsigned int userCount: 서버에 접속중인 사용자 수

unsigned int roomCount: 현재 존재하는 방의 개수

## 2. 로비

### 패킷 구분

```
#define ENTER_LOBBY_REQ 10
#define ENTER_LOBBY_ACK 11
#define CLIENT_READY 12
#define CLIENT_NOTREADY 13
#define GAME_START 14
```

### 구조체

#### 클라이언트

**Struct EnterLobbyReqPacket {}: 로비 입장 요청**

Char size

Char type

**Struct ReadyClientToServer {}: 준비/준비해제**

Char size

Char type

Char id[MAX_ID_SIZE]: 준비/해제 요청을 보낸 플레이어 ID

#### 서버

**Struct EnterLobbyAckPacket {}: 로비 입장 승인**

Char size

Char type

unsigned int roomId: 입장한 방의 고유 번호

char seatIndex: 방 안에 플레이어 자리 인덱스

char readyCount: 현재 준비 인원 수

**Struct GameStartPacket {}: 게임 시작 전송**

Char size

Char type

unsigned int roomId

unsigned int seed: 게임 시작 시 랜덤 시드

char playerCount: 게임에 참여 중인 플레이어 인원

unsigned int playerIds[MAX_PLAYERS]: 참가자들 고유 ID

## 3. 인게임

### 패킷 구분

```
#define TURN_START 20
#define FIRE_CMD 21
#define PROJECTILE_UPDATE 22
#define DAMAGE_RESULT 23
#define TERRAIN_UPDATE 24
#define GAME_RESULT 25
```

### 구조체

**클라이언트**

**Struct FireCmdPacket {}: 발사**

Char size

Char type

unsigned int roomId: 게임이 진행중인 방의 고유 ID

float angleDeg: 포탄 발사 각도

float power: 포탄 발사 세기

### 서버

**Struct TurnStartPacket {}: 턴 시작**

Char size

Char type

unsigned int roomId

unsigned int currentPlayerId: 현재 턴을 가진 플레이어 ID

unsigned int turnMs: 한 턴의 제한 시간

**Struct ProjectileUpdatePacket {}: 투사체 위치 갱신**

char size

char type

unsigned int roomId

unsigned int projId: 투사체의 고유 식별 번호

float x, y: 투사체의 현재 좌표

float vx, vy: 투사체의 속도 벡터

**Struct DamageResultPacket {}: 데미지 결과**

char size

char type

unsigned int roomId

unsigned int targetId: 공격을 받은 플레이어 ID

int damage: 계산된 데미지

int hp: 데미지 적용 후 남은 체력

**Struct TerrainUpdatePacket {}: 지형 파괴**

char size

char type

unsigned int roomId

unsigned short x: 지형이 파괴된 중심 좌표

unsigned short y: 지형이 파괴된 중심 좌표

unsigned char radius: 지형 파괴 범위 반경

**Struct GameResultPacket {}: 게임 결과**

char size

char type

unsigned int roomId

unsigned int winnerTeam: 승리한 팀의 식별 번호

## 4 – 4 동기화

### 4 – 4 – 1 턴 동기화

1. 턴 시작 시점에 서버가 Player ID를 세팅 뮤텍스로 잠금
2. 플레이어의 입력을 기다림
3. 입력이 오면 Validate()로 검사, 포탄 발사.
4. 시뮬레이션 결과를 같은 락 안에서 처리.
5. 턴 종료 시 뮤텍스 해제 후 다음 턴으로 이동.

### 4 – 4 – 2 충돌/지형 스레드

1. 충돌과 지형은 개별 뮤텍스를 사용 동시 접근 관리.
2. TerrainThread는 큐에서 작업을 꺼내 처리.

**4 – 4 – 3 클라이언트 동기화**

1. 서버 패킷 수신 -> 오브젝트 상태 업데이트
2. 오브젝트 좌표 읽어 화면 렌더링
3. 입력 중복 방지

# 5. 개발 환경

**언어**: C++

**IDE**: Visual Studio 2022

**OS**: Windows 11

**협업 애플리케이션**: git

**통신 프로토콜**: TCP/IP 소켓 기반

# 6. 팀원간 역할 분담

| 구분 | 내용 | 인원 |
|---|---|---|
| Server | Server Framework | 최제연, 김준호 |
| | **In Game (핵심 로직)**<br>(IsMyTurn/WaitInput/Validate) | 최제연 |
| | **InGame (전투 시뮬레이션)**<br>(/SimulateProjectile/ApplyDamage/GameEnd) | 김준호 |
| | 동기화 | All |
| | Client Thread | 함민규 |
| | Update Thread | 최제연 |
| | Collision Thread | 최제연 |
| | 지형 Thread | 함민규 |
| 구분 | 내용 | 인원 |
| Client | Client Framework | 함민규, 김준호 |
| | **InGame·Collision**<br>(SendFire/SendData) | 함민규, 김준호 |

# 7. 개발 일정

## 김준호

| 10/27 | 10/28 | 10/29 | 10/30 | 10/31 | 11/1 | 11/2 |
|---|---|---|---|---|---|---|
| 기획서 제출 및 피드백 | | 피드백 수집 기획서 수정 완료 | 클라이언트 코드 분석 | Git 생성 | 메인 스레드 회의 | 서버 전투 시뮬레이션 설계 (구조 설계 및 함수 흐름도 작성) |
| **11/3** | **11/4** | **11/5** | **11/6** | **11/7** | **11/8** | **11/9** |
| | 회의 | 기획서 최종 수정 | 클라이언트 프레임워크 (기본 루프,렌더링 정의) | 클라이언트 Recb 구조 파악 | 서버 Recv/Send 처리 모듈 설계 | 버퍼 관리 로직 구현 |
| **11/10** | **11/11** | **11/12** | **11/13** | **11/14** | **11/15** | **11/16** |
| | 회의 | Server Framework 협업 | Server Famework (초기 구조 테스트) | Server Framework 점검 및 수정 | Update Thread/Simulation 구조 검토 | Simulation 흐름 정의 및 전투 모듈 설계 |
| **11/17** | **11/18** | **11/19** | **11/20** | **11/21** | **11/22** | **11/23** |
| InGame Simulation 구조 초안 완성 | | SimulateProjectile 설계 및 제작 | ApplyDamage 설계 및 테스트 코드 작성 | 통합 테스트 및 디버깅 | SimulateProjectile/Apply Danage 통합 | GameEnd 처리 설계 및 예외 처리 |
| **11/24** | **11/25** | **11/26** | **11/27** | **11/28** | **11/29** | **11/30** |
| 전투 모듈 통합 테스트 완료 | 회의 | 동기화 구조 설계 협업 | 전투 로직 단위 테스트 및 디버깅 | 충돌 감지 및 전투 시스템 통합 | Server Update Thread 설계 및 제작 | Server 동기화 테스트 완료 |
| **12/1** | **12/2** | **12/3** | **12/4** | **12/5** | **12/6** | |
| 최종 디버깅 | 리펙토링 | 리펙토링 | | | | |

## 함민규

| 10/27 | 10/28 | 10/29 | 10/30 | 10/31 | 11/1 | 11/2 |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| 기획서 제출 및 피드백 | | 피드백 수집, 기획서 수정 완료 | 클라이언트 코드 분석 | Git 생성 | 메인 스레드 회의 | 스레드 함수 정의 |
| **11/3** | **11/4** | **11/5** | **11/6** | **11/7** | **11/8** | **11/9** |
| | 회의 | 기획서 최종 수정 | | 포물선 각도/ 파워 UI 입력 연결 | 키 입력 처리 | 서버 송신 테스트 |
| **11/10** | **11/11** | **11/12** | **11/13** | **11/14** | **11/15** | **11/16** |
| | 회의 | 더미 서버로 데이터 송신 테스트 | Recv ThreadProc() 작성 | Recv ThreadProc() 작성 | PROJECTILE_UPDATE 수신 처리 | |
| **11/17** | **11/18** | **11/19** | **11/20** | **11/21** | **11/22** | **11/23** |
| 스레드 간 동기화 적용 | 지형 스레드 작성 | 서버에서 오는 TERRAIN_UPDATE 처리 | | 지형 데이터 수정 확인 | 화면상 지형 파괴 시각화 구현 | |
| **11/24** | **11/25** | **11/26** | **11/27** | **11/28** | **11/29** | **11/30** |
| | 회의 | 클라이언트 전체 루프 확인 | 서버의 턴 전환, 피해 판정 연동 확인 | 서버 패킷 시각화 | 버그 수정 | |
| **12/1** | **12/2** | **12/3** | **12/4** | **12/5** | **12/6** | |
| 최종 디버깅 | 리펙토링 | 리펙토링 | | | | |

# 최제연

| **10/27** | **10/28** | **10/29** | **10/30** | **10/31** | **11/1** | **11/2** |
|---|---|---|---|---|---|---|
| 기획서 제출 및 피드백 | | 피드백 수집, 기획서 수정 완료 | 클라이언트 코드 분석 | Git 생성 | 메인 스레드 회의 | 스레드 함수 정의 |
| **11/3** | **11/4** | **11/5** | **11/6** | **11/7** | **11/8** | **11/9** |
| 회의 | 회의 | 기획서 최종 수정 | Server Framework설계 및 (클레스 구축) | Server Framework(네트워크 구축 | Server Framework초기구조 제작 | |
| **11/10** | **11/11** | **11/12** | **11/13** | **11/14** | **11/15** | **11/16** |
| 지형파괴 작업 처리 및 패킷 생성구현 | 회의 | Server Framework 협업 | Server Framework ( 초기 구조 테스트) | Server Framework 점검 및 수정 | In Game핵심 로직 설계 | IsMyTurn 설계 및 제작 |
| **11/17** | **11/18** | **11/19** | **11/20** | **11/21** | **11/22** | **11/23** |

| | | | | | | |
|---|---|---|---|---|---|---|
| | WaitInput 설계 및 제작 | NextTurn | Broadcast TurnStart | | 입력값으로 턴 반복 확인 | Validate설계 및 제작 |
| **11/24** | **11/25** | **11/26** | **11/27** | **11/28** | **11/29** | **11/30** |
| 동기화 구현 | 회의 | 지형 Thread와 연동 | 충돌 및 지형 시스템 통합 | UpdateThread 설계 및 제작 | UpdateThread 점검 및 수정 | |
| **12/1** | **12/2** | **12/3** | **12/4** | **12/5** | **12/6** | |
| 최종 디버깅 | 리펙토링 | 리펙토링 | | | | |