네트워크 프로그래밍 최종 보고서

제출일자 : 2015-12-09

수학과 201010365 최종원

● 개요

1. 분석단계

- 요구사항 분석
- 데이터 분석
- 아키텍처 분석
- 네트워크 토폴로지

2. 설계단계

- 아키텍처 설계
- 애플리케이션 설계
    - 게임 전체 프로세스
    - 게임 전체 프로세스에 필요한 통신요소
    - 매칭 서버
    - 오류 대응
    - 네트워크 토폴로지
    - 게임서버 연결
    - 플레이어 글로벌 ID 및 로컬 ID
    - 캐릭터 이동 디자인
    - 투사체 이동 디자인

3. 구현단계

- 유니티 함수
- 카메라
- 네트워크 라이브러리
- 스레드 사용
- 시리얼라이즈 및 디시리얼라이즈
- 패킷 전달 과정
- 세션 관리
- 클래스 다이어그램
- C# 코딩 기법

4. 참고 문헌

5. 게임 스크립트 목록

6. 차후 진행사항

- 개발환경
- 개선사항

7. 실행화면 캡쳐

1. 분석단계

● 요구사항분석

- 현재 게임 현황 패턴 수집
  - 퍼즐 또는 런(Run) 등 게임 방식이 간단한 순위형 게임
  - 카드나 캐릭터를 수집한 뒤 전투를 하는 게임
  - 자신의 요새나 성을 쌓고 서로 공격하는 게임
  - 자신의 소유물을 경영하는 게임
  - 스포츠 기반 수집 게임

=> 현재 나오는 간단한 게임 패턴이 아닌 온라인 슈팅게임을 선택한다.

- 네트워크 프로그래밍 기술
  - TCP/IP, UDP 프로토콜
  - 타임아웃 프로그래밍
  - 비동기 방식
  - 오류 통지

- 클래스 레퍼런스 테이블
  - 슈팅게임 오픈소스 참고[1)]

한 학기 동안 배운 네트워크 프로그래밍 기술을 이용하여 한 때 유행했던 턴제 슈팅 게임을 기반으로 비동기 방식의 실시간 슈팅 게임을 만든다.

● 승리조건 및 요구사항 정의 :

- 턴제 전략 슈팅 게임을 벗어나 비동기 방식을 채택한다.
- 상대가 컨트롤 하는 모든 캐릭터의 체력을 0으로 만들면 승리한다.
- 캐릭터 및 object는 맵 위에 있으며 중력 엔진의 영향을 받는다.
- 캐릭터는 object의 이벤트로 피해가 발생한다.
- 점프를 최대 두 번까지 할 수 있다.

---

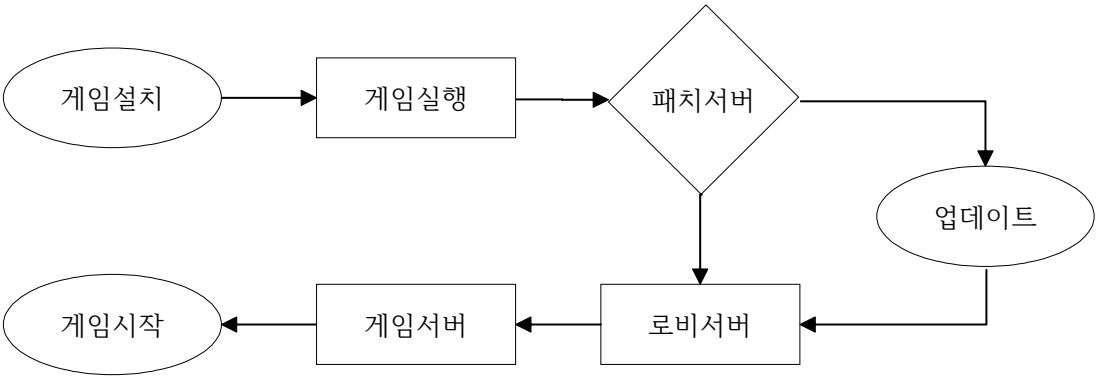
1) http://hedgewars.org/kb/?mode=list

● **데이터 분석**

플레이 패턴
- 평균 5분 내외의 세션 타임
- 비동기 전략 슈팅게임

● **아키텍처 분석**

- 일반적인 온라인 게임 흐름도



● **네트워크 토폴로지**

자료조사 : 게임에서 주로 사용하는 토폴로지

● 스타형
- 장점 : 정보가 서버에 집중되어 일관성을 유지하기 쉽다
- 단점 : 서버를 반드시 경유하므로 통신 지연이 발생한다.
  서버의 부하가 크다
  서버가 끊기면 모든 통신이 두절된다.

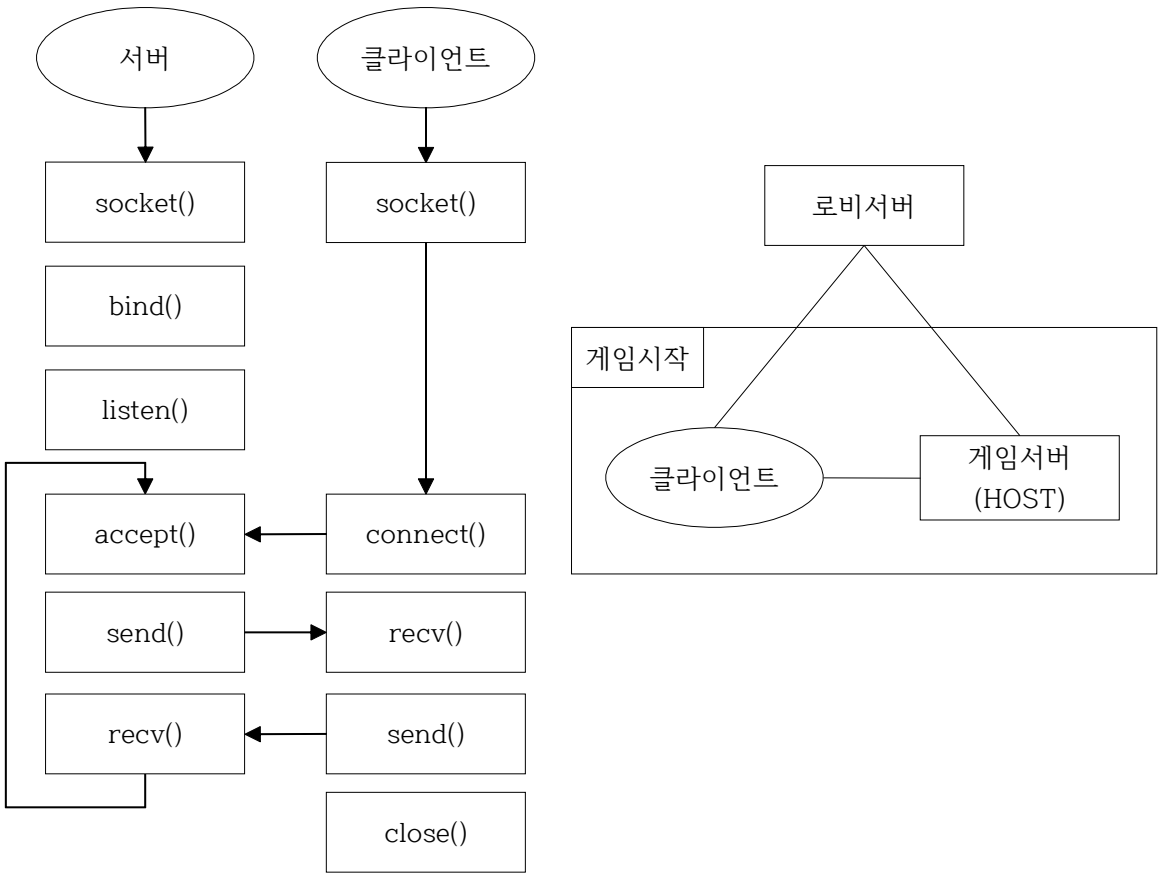
● 메시형
- 장점 : 단절의 내성이 강하다.
- 단점 : 멀리 떨어진 단말의 통신이 지연된다.

● 풀메시형
- 장점 : 모든 노드가 서로 접속하여 동시에 통신할 수 있다.
  단절의 내성이 강하다
- 단점 : 통신량이 증가하고 단말의 관리가 복잡하다.

2. 설계단계

아키텍처 설계

- 기본 틀은 아키텍처 분석의 일반적인 온라인 게임 흐름도를 따른다.
- 로비서버는 접속한 사람들의 리스트를 다룬다.
- 클라이언트 중 하나를 호스트로 만들어 게임서버의 역할을 한다.
- 호스트 안에 로컬 클라이언트를 만들어 로컬 호스트와 통신하게 한다.
- 로비서버와 클라이언트들이 통신 하다가 게임이 시작되면 클라이언트들과 호스트사이 데이터를 교환한다.



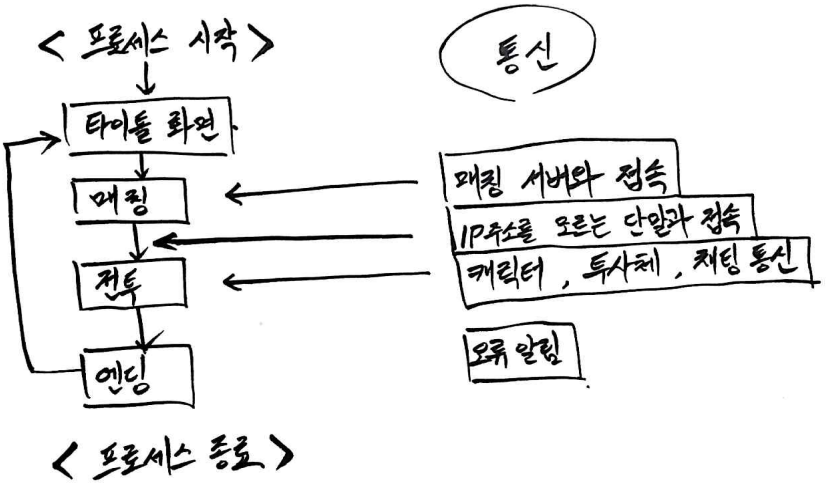
애플리케이션 설계

● **게임 전체의 프로세스**

게임을 시작하면 함께 플레이할 플레이어가 여러 명 접속할 것이기 때문에 매칭 서버를 구현하여 클라이언트가 매칭 서버에 접속하여 매칭할 플레이어를 찾는다.
매칭이 끝나면 함께 플레이 할 플레이어와 접속하고 게임을 시작한다.
게임이 시작되면 플레이어들이 전투를 시작한다.
전투가 끝나면 매칭 서버로 되돌아간다.

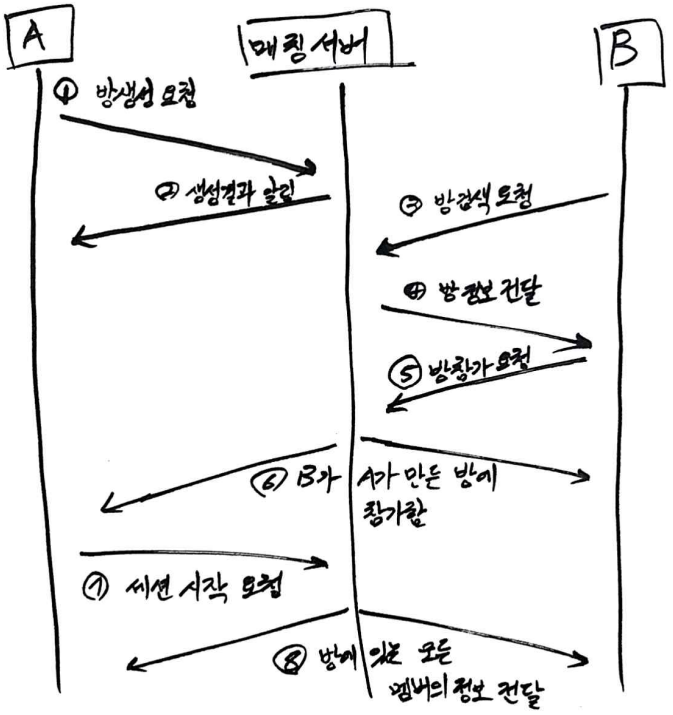
● **게임 전체 프로세스에 필요한 통신요소**

게임이 시작하면 먼저 매칭을 한다. 매칭에서는 다른 플레이어와 접속해야 하므로 통신이 필요하다.
캐릭터가 두 명 이상 한 방에 있으면 게임을 시작할 수 있다.
전투중에 필요한 통신은 캐릭터, 투사체, HP정보 등이있다.
캐릭터 통신은 서로의 위치를 주기적으로 전달할 수 있고, 투사체 정보는 쏘는 위치와 힘, 각도를 쏠 때 마다 전달하는 방법이 있으며, HP정보는 현재 HP를 주기적으로 전달하여 동기화하는 방법이 있다.



● **매칭 서버.**

모르는 사람과 함께 플레이 하려면 서로의 IP를 알아야 하는 서버 클라이언트 1:1 모델로는 설계가 불가능하다. 따라서 매칭 서버의 IP로 접속하는 클라이언트를 만든 다음에 매칭 서버가 들어온 IP를 판별한 후 서로 연결해 주는 방식을 택하면 클라이언트끼리 IP주소를 몰라도 접속할 방법이 생기게 된다.



① 매칭 서버에 방을 만들어 줄 것을 요구한다.

② 매칭 서버는 클라이언트 방 정보를 생성하고 요구 받은 클라이언트에게 방을 만든 결과를 알린다.

③ 또 다른 클라이언트가 매칭 서버에 방이 있는지 요청한다.

④ 매칭 서버는 클라이언트가 만든 방 정보를 요청한 또 다른 클라이언트에게 알려준다.

⑤ 방 목록에서 골라 방에 참가를 요청한다.

⑥ 매칭 서버는 참가한 클라이언트가 참가 했다는 상황을 브로드캐스트 한다.

⑦ 방장(방 생성자)은 매칭 서버에 세션 시작을 요청한다.

⑧ 매칭 서버는 방에 참가한 모든 멤버의 정보를 참가자에게 알려 고유 ID를 부여해준다.

● **매칭 오류 대응**

네트워크 게임을 하는 사용자가 불만을 가지지 않도록 오류가 발생하면 오류가 무엇인지 정확히 알리고 종료하게 한다.

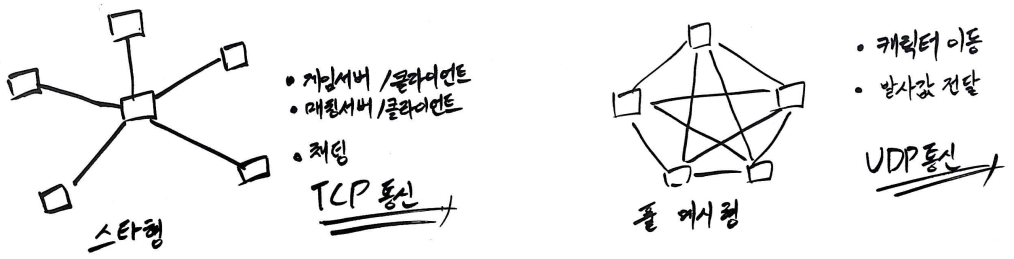
- 방에 참가한 게스트의 접속이 끊김
=> 서버는 접속이 끊긴 게스트의 정보를 방의 멤버에서 삭제해야한다.

- 방을 만든 호스트의 접속이 끊겼다.
=> 호스트 마이그레이션을 하는 방법이 있지만 방을 없애는 방법을 택하여 게스트와 접속을 끊는다.

- 참가할 방이 게임을 시작해버렸다.
=> 방 참가 실패를 알리고 다른 방에 들어갈 수 있게 한다.

● **구현할 네트워크 토폴로지**

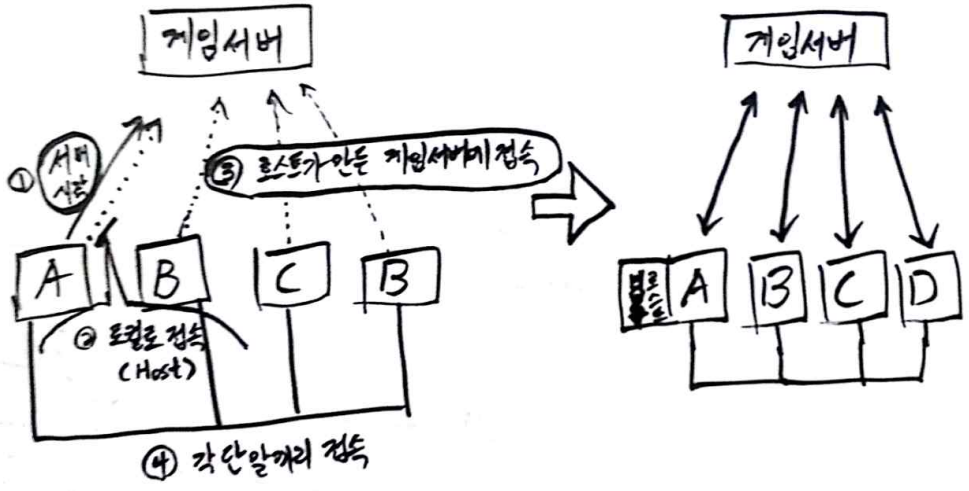
게임에서 통신이 필요한 정보는 캐릭터 좌표, HP정보, 투사체와 채팅 정보가 있으며 캐릭터 이동과 투사체 좌표는 반응이 빨라야 하므로 풀메시형을 채택하고 HP정보나 채팅 정보는 스타형 토폴로지를 채택한다.



스타형 토폴로지는 TCP통신, 풀메시형 토폴로지는 UDP방식을 채택하여 두 통신 방식을 모두 클라이언트에 구현한다.

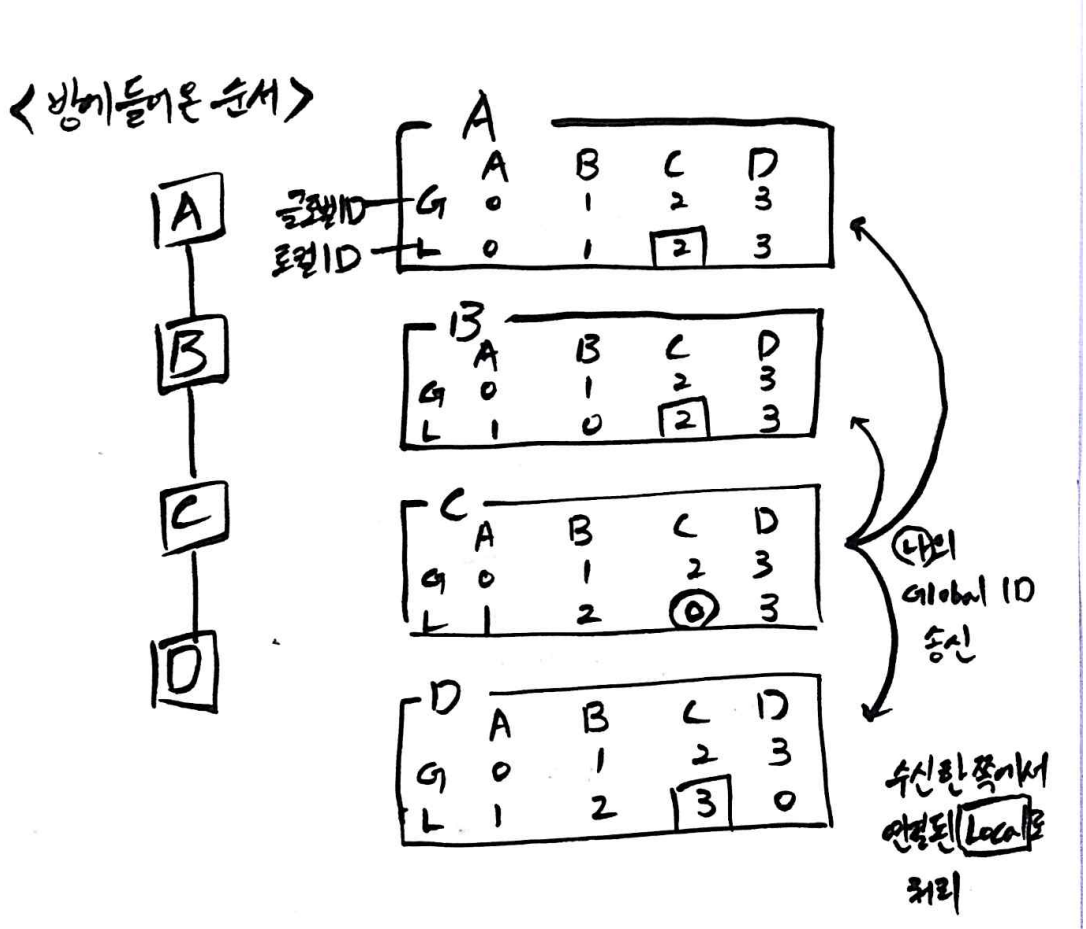
● **게임 서버 연결**

매칭이 완료되면 매칭 서버는 호스트(방을 생성한 클라이언트)에게 게임 서버를 시작하게 한 후 게스트가 게임서버에 참가 한 후 방 안에 있는 클라이언트와의 연결을 끊는다. 게임서버와의 연결이 모두 이루어지면 각 단말 끼리도 접속한다.



- **플레이어 글로벌 ID와 로컬 ID**

- 매칭이 끝나면 플레이어에게 캐릭터를 할당한다.
- 할당방법 : 호스트가 만든 방의 순서대로 글로벌 ID를 할당한다.
- 게임 세션이 시작되면 로컬단말을 0, 네트워크 단말을 순서대로 로컬 ID를 할당해서 각 클라이언트가 네트워크 ID를 판별하게 한다.



게임 시작 전에 캐릭터 생성 위치를 각 클라이언트마다 일치 시킬 난수를 생성하여 위치를 동기화 한 후 게임을 시작한다.

● **캐릭터 이동 디자인**

키 입력을 동기화 하는 방식이 있지만 여러 명의 키 입력 정보를 주고받기 때문에 통신 상대의 키 입력을 수신할 때까지 게임처리가 멈출 수가 있다.
통신 지연 때문에 과거의 위치를 수신하게 되지만 이는 불가피한 사항이므로 쾌적하게 플레이 할 수 있는 모델을 제시한다.

로컬 단말의 현재 캐릭터 좌표를 주기적으로 송신해서 리모트 단말에서 이동시키는 방법을 채용한다.

- **캐릭터 이동의 좌표 보간법**

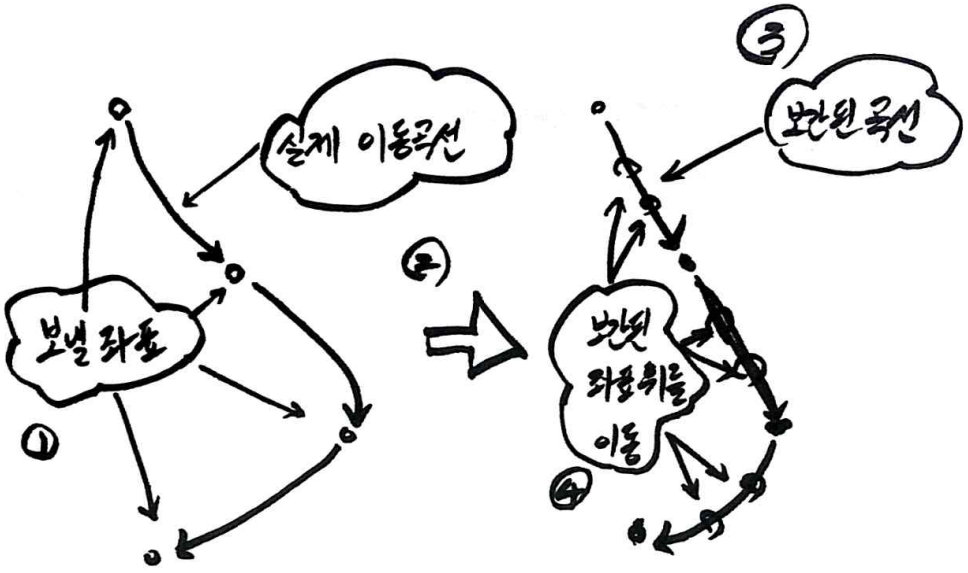
  선형보간(Linear Interpolation)

>> **스플라인보간(Spline Interpolation)**

  라그랑주보간(Lagrange Interpolation)

  에르미트보간(Hermite Interpolation)

스플라인 보간 방법은 점이 n개가 있을 때 n-1차 다항식을 그려 곡선으로 연결하는 것으로 얻은 **곡선의 방정식으로부터 점을 추출**하여 수신된 n개의 점의 공백을 메우는 작업이다. 본 게임에서는 점 4개의 배열을 10 프레임마다 송신하는 것으로 3차 스플라인 곡선 - 3차 다항식으로부터 송신되지 못한 점들을 대입하여 추측하는 방식이다. 스플라인 보간법을 이용한 네트워크 캐릭터의 이동 로직은 다음과 같다.



① 일정 프레임마다 현재 좌표를 추출
② 추출한 좌표를 송신
③ 수신한 단말에서 받은 좌표로 스플라인 곡선을 구하고 보간을 시행
④ 보간된 좌표를 따라 이동

캐릭터 좌표정보는 UDP통신으로 주고받는다. 이때 정보를 한번만 송신하므로 패킷이 유실되면 데이터가 불완전해지기 때문에 패킷 유실 대책으로 지난 점의 좌표를 배열로 중복하여 송신한다.

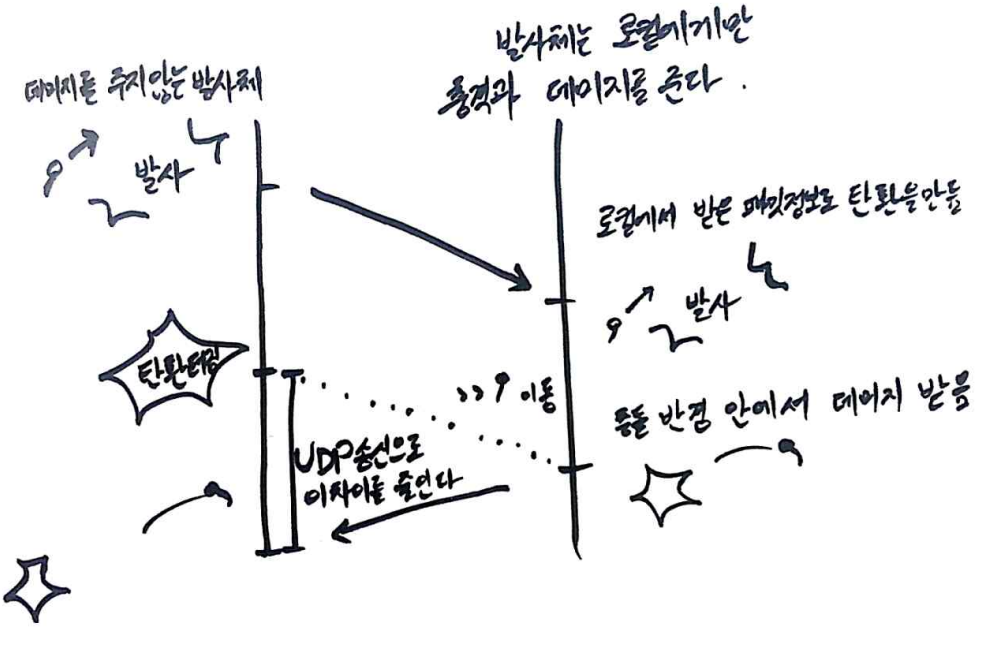
- **예상하는 단점**: 송신한 좌표가 일정하게 나눈 프레임 이상의 차이 + 통신지연 만큼 차이가 난다. 투사체를 맞았을 때 정확히 동기화가 되지 않을 가능성이 있다.

● **투사체 전달 디자인**

- 보간법으로 인한 정확성문제

스플라인 보간법으로 인한 캐릭터의 위치 좌표는 **프레임 차이 + 통신 지연 + 보간 연산**만큼 로컬캐릭터와 네트워크 캐릭터 간에 차이가 발생한다. 네트워크 너머의 상대방을 공격하는 게임인 만큼 프레임별 캐릭터 좌표의 차이가 게임의 영향을 줄 수 있어 치명적인 문제가 될 수 있다. 하지만 네트워크로 데이터를 전달하는 과정에서 통신 지연은 불가피한 문제이므로 어느 정도 차이는 분명히 존재하게 된다. 외관상의 불일치와 타협하면서 게임 성을 검토하는 것이 중요하므로 **사용자 입장에서 쾌적하게 동기화** 하는 것이 좋다. 이에 본 게임에서는 발사체 클래스를 하나로 관리하여 로컬플레이어가 발사 로직을 수행하자마자 패킷으로 발사각도, 발사 힘, 발사 좌표를 송신하게 되면 이 패킷을 수신하는 네트워크 플레이어는 캐릭터가 그 지연시간동안 이동하게 되더라도 정확히 같은 자리에서 같은 힘과 각도로 발사하게 된다.

결론적으로 로컬 단말이 해당위치에서 투사체를 만들어 받는 것이므로 정확하지는 않지만 사용자 입장에서 쾌적하게 동기화되는 것처럼 보일 수 있게 된다.



- 발사 각 구조체 전달

발사각은 3차원 공간의 절대 좌표를 기준으로 오일러 각을 이용해 회전시킨다. 그러나 오일러 각을 이용한 회전 방식은 X, Y, Z 축을 차례대로 회전 시키는 것으로 회전하는 동안 축끼리 겹치는 상황이 발생했을 때 **짐벌락**이라는 잠김 현상이 발생한다. 이러한 문제점을 해결하기 위해 유니티에서는 게임오브젝트의 회전을 Quaternion(사원수)의 데이터 타입으로 처리한다. Quaternion은 4차원 복소수로 이루어져 있기 때문에 이 데이터를 시리얼라이즈 또는 디시리얼라이즈 할 경우 4개의 좌표를 처리해야한다.

- 데미지 계산 방법

투사체가 해당 지역에 폭발하면 폭발 반경 내의 Collider(Unity 물리엔진)를 가져와서 Collider의 위치 벡터와 폭발한 곳의 위치 벡터를 벡터 차로 나타내면 충돌 받는 힘의 벡터를 구할 수 있다. 여기에 일정 힘과 데미지를 곱하여 반경 내에 가까이 있을수록 밀려나는 힘을 많이 받고 데미지도 많이 받는다.

3. 구현 단계

- **유니티 함수 : MonoBehavior 상속**

void Awake()
- 스크립트가 실행될 때 한번만 호출되는 함수
- 주로 게임의 상태 값 또는 변수의 초기화에 사용

void Start()
- Update 함수가 호출되기 전에 한번만 호출
- 스크립트가 활성화 돼 있어야 실행된다.
- 다른 스크립트의 모든 Awake가 모두 다 실행된 이후에 실행된다.

void Update()
- 프레임마다 호출되는 함수로 보통 게임이 30-60fps를 가지므로 1초에 30-60번 실행된다. 시간을 연결시키려면 Time.deltatime을 쓰면 된다.
- 스크립트가 활성화 돼 있어야 실행된다.

void LateUpdate()
- 모든 Update함수가 호출되고 나서 한 번씩 호출된다.
- 순차적으로 실행해야하는 로직에 주로 사용된다.
- 카메라 이동 로직에 사용하였다.

void FixedUpdate()
- 물리엔진을 사용하는 경우 일정 시간 간격으로 힘을 가할 때 사용한다.
- 가속도가 필요한 운동에 사용한다.

void OnGUI()
- GUI관련 함수를 여기서 사용하여 카메라위치에 나타난다.

- **카메라**

카메라 로직은 LookAt함수를 이용하여 타겟을 바라보는 용도로 쓰였다.
디테일한 카메라 작업은 프로젝트를 제출한 다음 차후 수정할 계획이다.

● **네트워크 라이브러리**

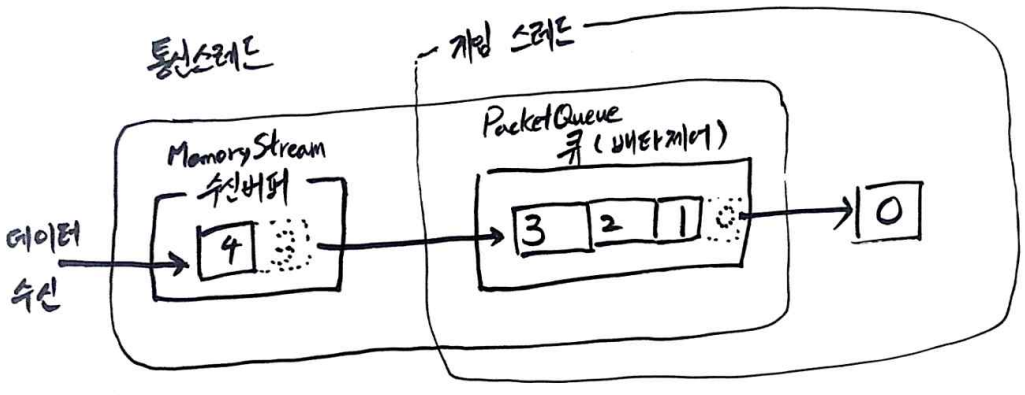
통신의 기본은 접속, 송신, 수신, 접속 종료로 크게 볼 수 있다. 이를 실행하려면 C#에서는 Socket클래스의 인스턴스를 참조해야 하므로 송수신이 일어날 때마다 Socket클래스의 인스턴스를 참조하는 코드를 만들어야한다. 하지만 게임 프로그램에서는 다른 단말과 통신하는 기능만 있으면 되지 Socket클래스의 인스턴스까지 직접 다룰 필요가 없다.

또한 통신의 기본 사항 역시 매번 구현하는 것도 일이기 때문에 네트워크 라이브러리로 만들어두면 게임에서 일어나는 버그에만 신경 쓰게 할 수도 있다.

● **스레드 사용**

게임 애플리케이션의 일부로 통신을 처리하면 게임 자체를 처리하는데 부하가 걸려 통신을 하다가 게임이 버벅거릴 가능성이 있다. 또한 통신 쪽도 부하가 높아진다면 패킷이 유실될 가능성이 있다.
요즘 CPU는 멀티코어로 되어있으므로 통신을 게임과 별도의 스레드로 다른 코어에서 실행하면 게임 쪽 스레드에 영향을 주지 않으면서 통신할 수 있고 또한 통신측도 부하가 적게 걸려 데이터를 계속 수신할 수 있다.

게임 스레드와 통신 스레드는 데이터를 주고받는 버퍼를 공유할 뿐 이 버퍼에 접근할 때만 주의하면 데이터 송수신 타이밍에 신경을 쓰지 않고 게임을 처리할 수 있게된다.



각 스레드에서 비동기로 큐에 접근하게 되면 데이터가 훼손이 될 수 있으므로 버퍼에 접근할 때에는 그림과 같이 실행되기 때문에 각각 스레드에서 배타 제어(세마포어 락)을 해야 한다.

수신한 데이터는 큐 자료구조로 관리하여 통신 스레드에서는 수신 버퍼에 들어온 데이터를 큐로 옮기고, 게임 스레드가 수신한 데이터를 사용할 때에는 큐의 맨 앞 데이터를 사용한다.
송수신할 데이터는 MemoryStream클래스에서 버퍼링한다.
말 그대로 스트림 데이터기 때문에 패킷으론 다룰 수 없다. 그래서 패킷 정보를 구조체로 별도로 관리하여 패킷 정보는 List클래스로 관리하고 큐 자료구조를 이용하여 데이터를 송수신한다.

통신스레드에서는 메인 스레드로부터 송신 데이터가 등록되면 Socket클래스의 Send함수로 데이터를 송신한다. 그다음 Receive함수를 사용하여 수신 데이터를 Socket클래스의 수신 버퍼에서 메인 스레드와 주고받는 큐로 옮긴다. 이 과정을 스레드가 종료할 때까지 반복한다. 하지만 반복하는 과정에서 다른 스레드로의 처리 제어가 돌아가지 않으므로 여기서는 Sleep(3)를 넣어 3ms 마다 다른 스레드로의 제어를 옮겨준다.

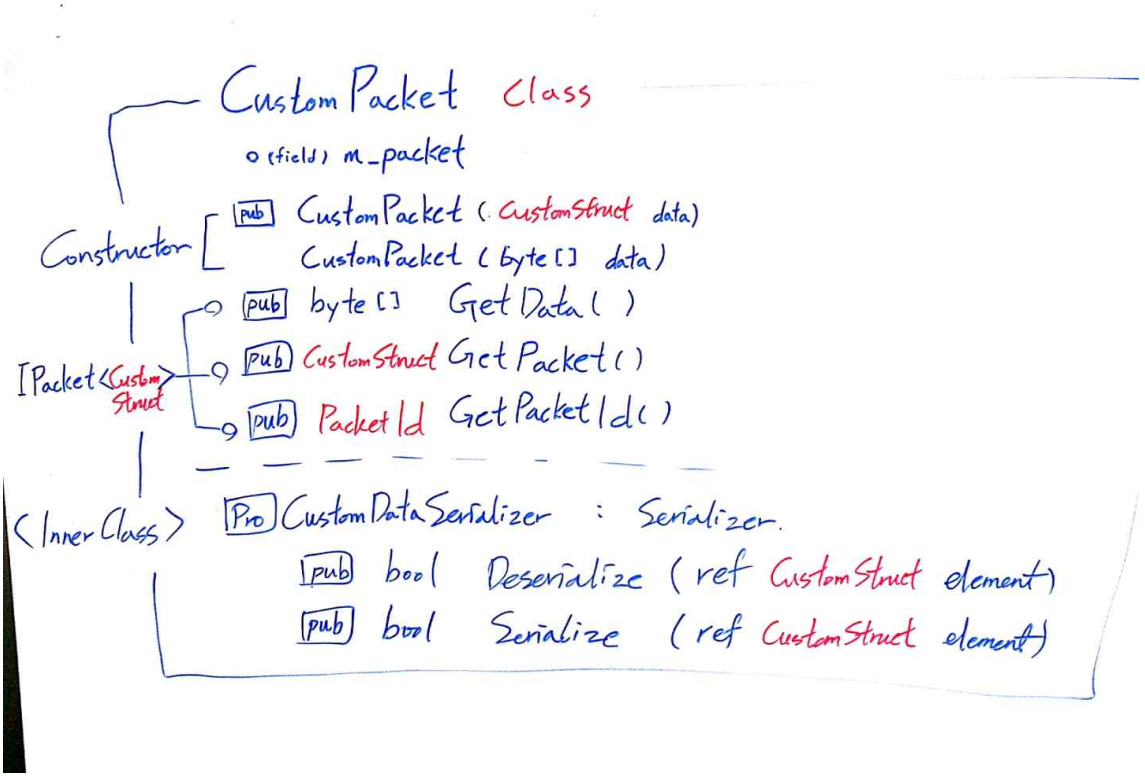
```
//
public virtual void ThreadDispatch()
{

    string str = "ThreadDispatch:" + m_threadLoop.ToString();
    Debug.Log(str);

    while (m_threadLoop) {
        // 접속 요청 감시.
        AcceptClient();

        // 세션 내 노드 송수신 처리.
        Dispatch();

        // 다른 스레드로 처리 위임.
        Thread.Sleep(3);
    }

    Debug.Log("Thread end.");
}
```

- **시리얼라이즈 및 디시리얼라이즈**

패킷에 들어갈 데이터는 여러 변수로 구성되고, 각각 여러 형으로 정의 되어 있다. 하지만 패킷을 구조체로 정의하여 변수 여러 개의 집합체를 모아서 보낼 수는 없다. 구조체의 모든 변수를 byte[]형으로 시리얼라이즈하고, 만들어진 스트림 데이터를 다시 넘겨주면 구조체 데이터로 디시리얼라이즈하는 작업이 필요하다.

- **패킷 전달 과정**

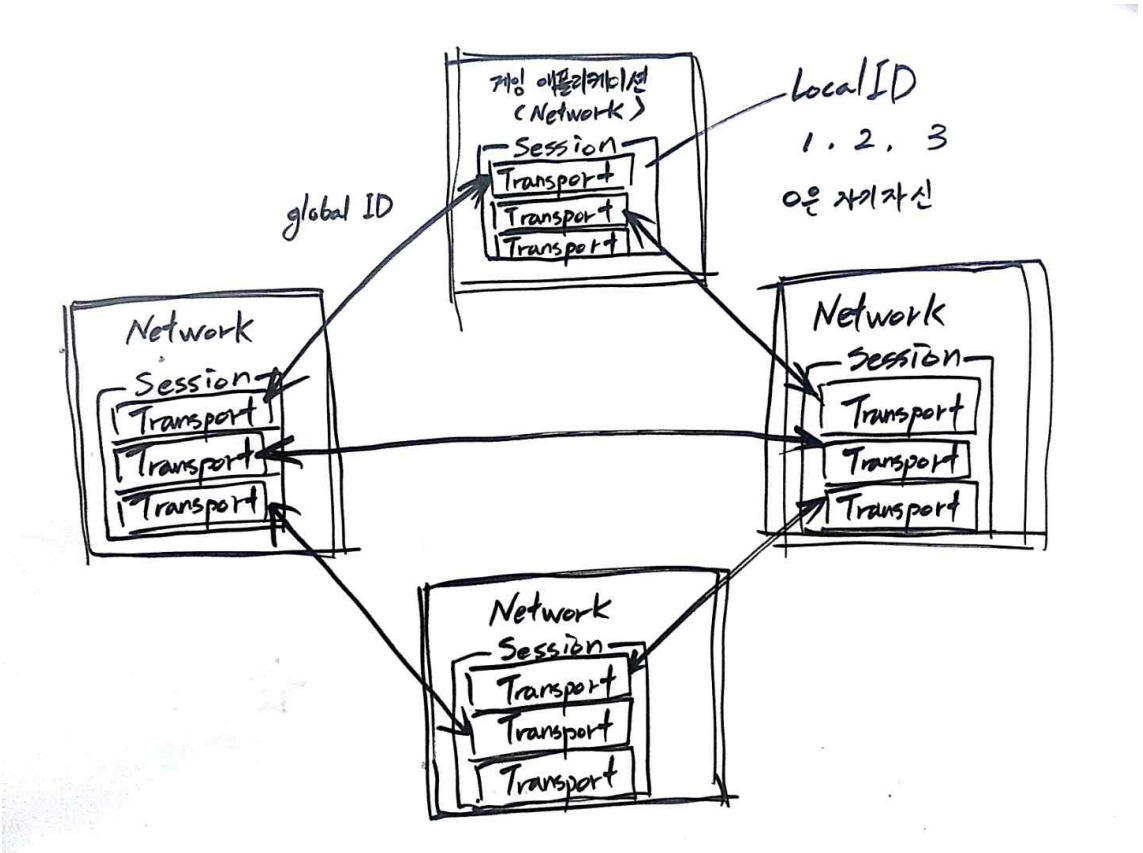
패킷 전달은 네트워크 모듈에 등록된 이벤트를 등록/구독함으로 이루어진다.
PacketStruct.cs에 구조체를 명시하고 PacketId를 구별한 뒤 Packet에 고유 시리얼라이즈와 디시리얼라이즈와 바이트 스트림 정보를 저장한다.
GameServer나 CharacterRoot에 등록된 네트워크 모듈을 통해 이벤트를 전달 받고 각 클라이언트에 패킷을 전달하게 된다.



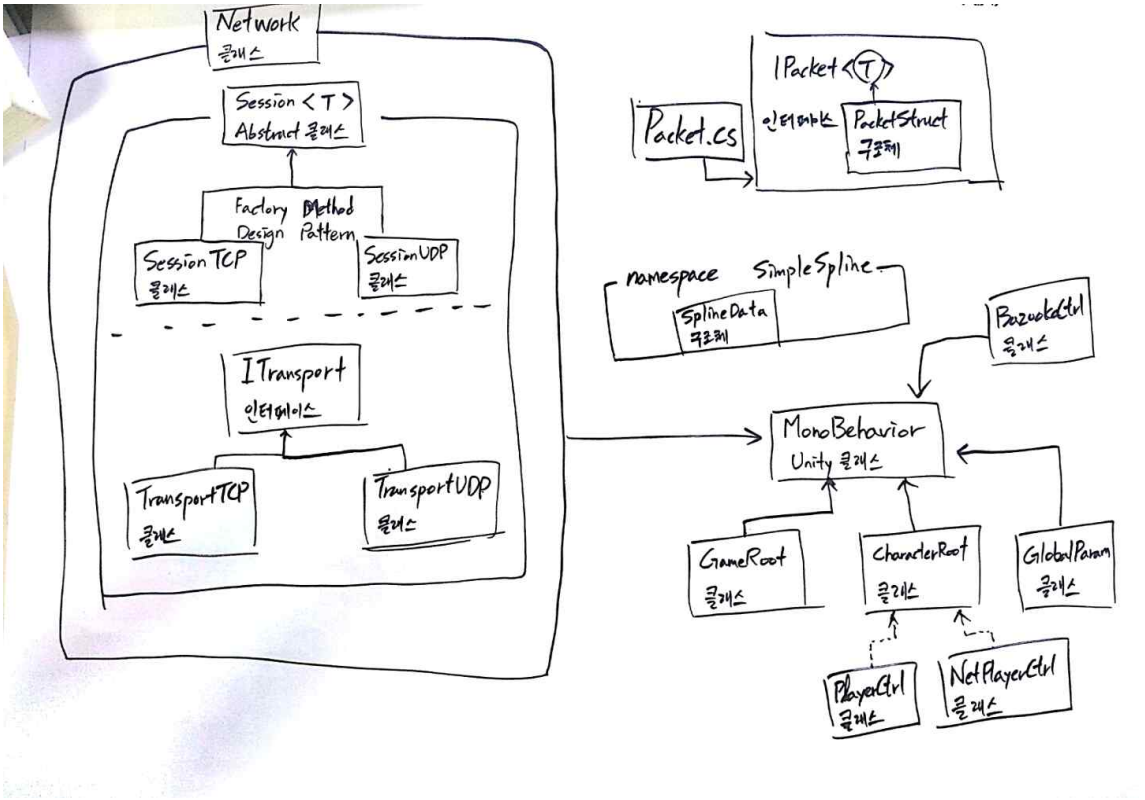
● **세션 관리**

게임을 3명이상 플레이하게 될 때는 1:1 통신을 할 수 없으므로 세션을 관리해야한다. Transport클래스를 묶어 하나의 세션으로 보고 접속할 곳과의 송수신 데이터를 관리하면서 여러 세션과 통신하도록 해야한다.

OSI 7 Layer모델의 Session Transport Layer를 어느정도 본따서 Network(Network Layer는 아니다) 클래스에서 TCP UDP를 구별하거나 패킷의 헤더를 볼 수 있게 하고, Session Layer로 전달한다. delegate 이벤트로 전달받은 Session layer에서 Transport의 인스턴스를 생성하여 네트워크 플레이어를 Transport의 인스턴스로 관리할 수 있게 된다.



- **클래스 다이어그램**



- **C# 코딩 기법**

MonoBehavior클래스는 유니티 엔진으로부터의 클래스이고, 이 클래스는 어느 스크립트나 쉽게 접근할 수 있게 한다.

TransportTCP클래스와 TransportUDP클래스에서는 Socket클래스를 캡슐화하는 역할을 한다.

캡슐화된 Socket클래스를 이용하기 위해 이벤트를 검출해서 게임 프로그램에 알려준다. C#의 Delegate를 Dictionary자료구조에 추가하여 이벤트를 등록/구독하는 방식을 택했다.

4. 참고문헌

- 반다이 남코 현역 프로그래머가 알려주는 유니티 네트워크 프로그래밍
  가와다 마사토시 :: 길벗

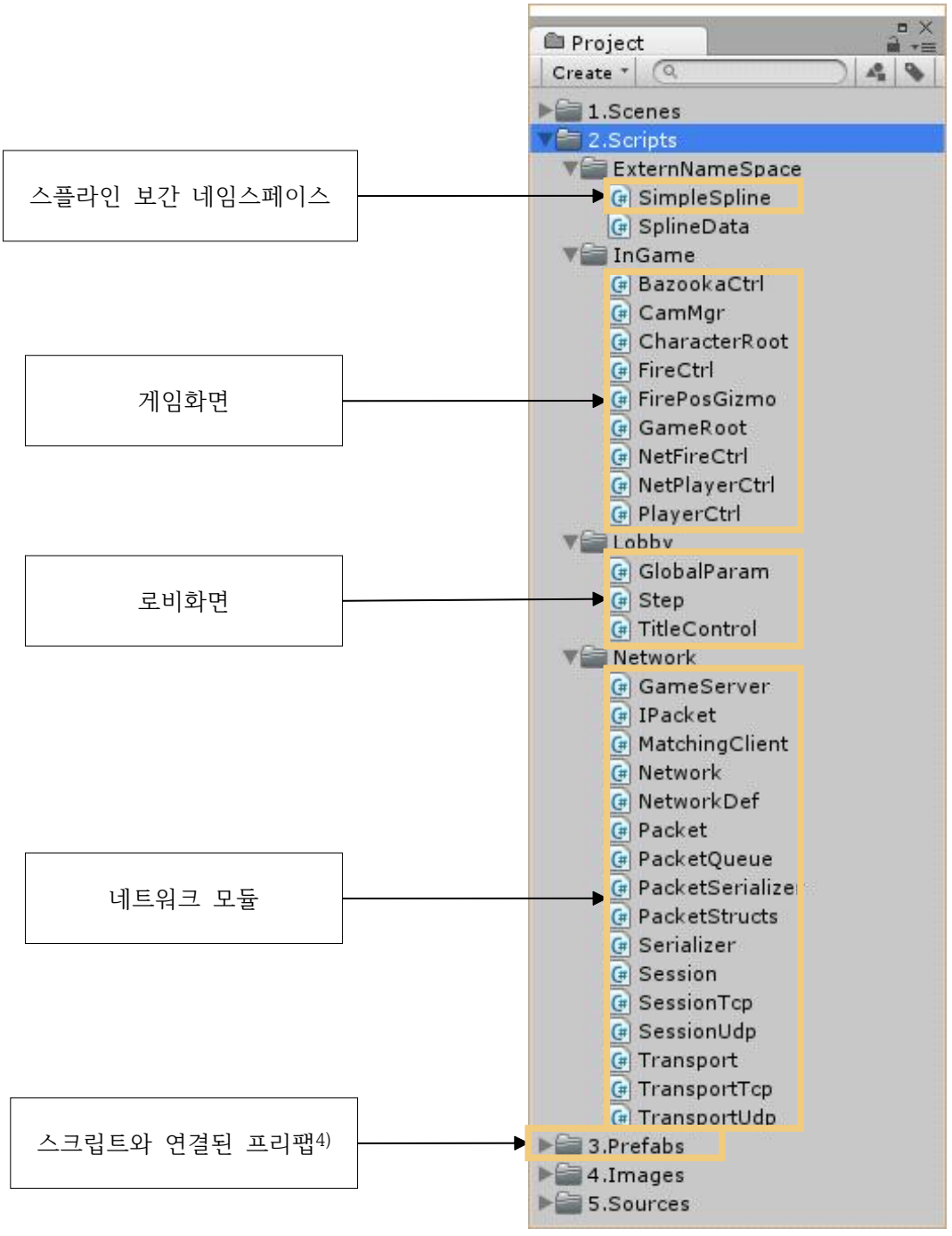
- 절대강좌! 유니티 3D 전문 개발자가 알려주는 효과적인 게임 제작기법
  이재현 :: 위키북스

- MSDN .NET Framework를 이용한 C# 프로그래밍 가이드[2)]

- Unity Scripting API[3)]

2) https://msdn.microsoft.com/ko-kr/library/67ef8sbd.aspx
3) http://docs.unity3d.com/ScriptReference/

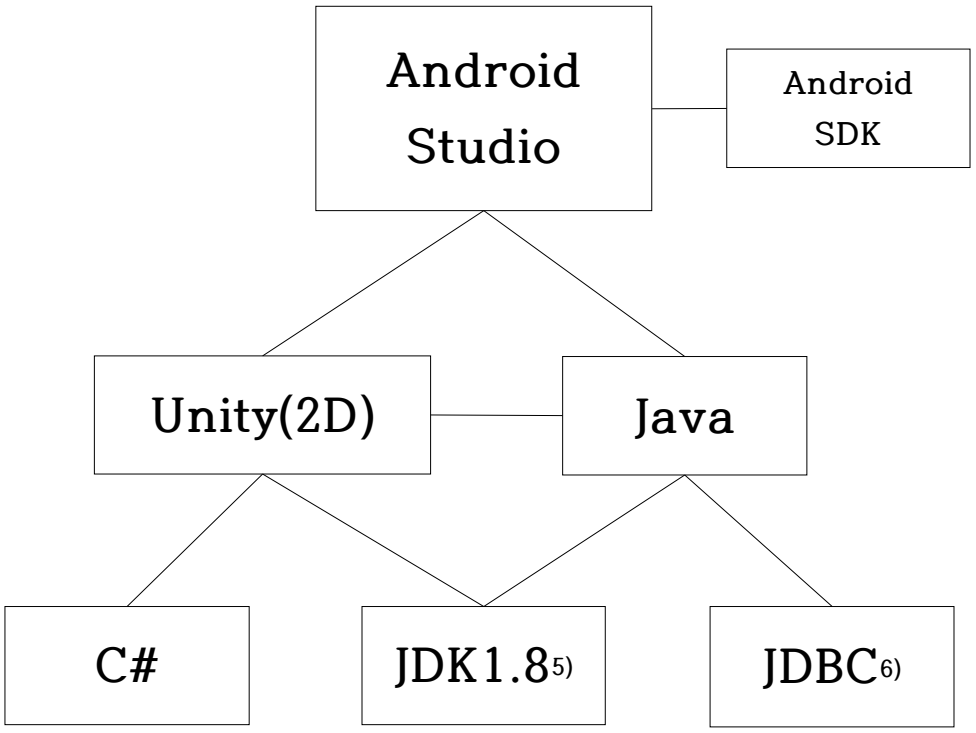
5. 게임 스크립트 목록



4) http://docs.unity3d.com/kr/current/Manual/Prefabs.html

6. 차후 진행 사항

● 개발환경



● 개선사항

- 아이템을 다양화 한다. (공격 아이템 뿐 아니라 이동 아이템도 만든다.)
- 카메라 이동 로직을 추가/확장한다.
- 애니메이션을 좀 더 디테일하게 만든다.
- 최적화 작업을 한다.
- 사용자 인터페이스를 간단하게 만들어 모바일 게임으로 만든다.

5) Unity compiler option: -source 1.6-target 1.6
6) MySQL, SQLite

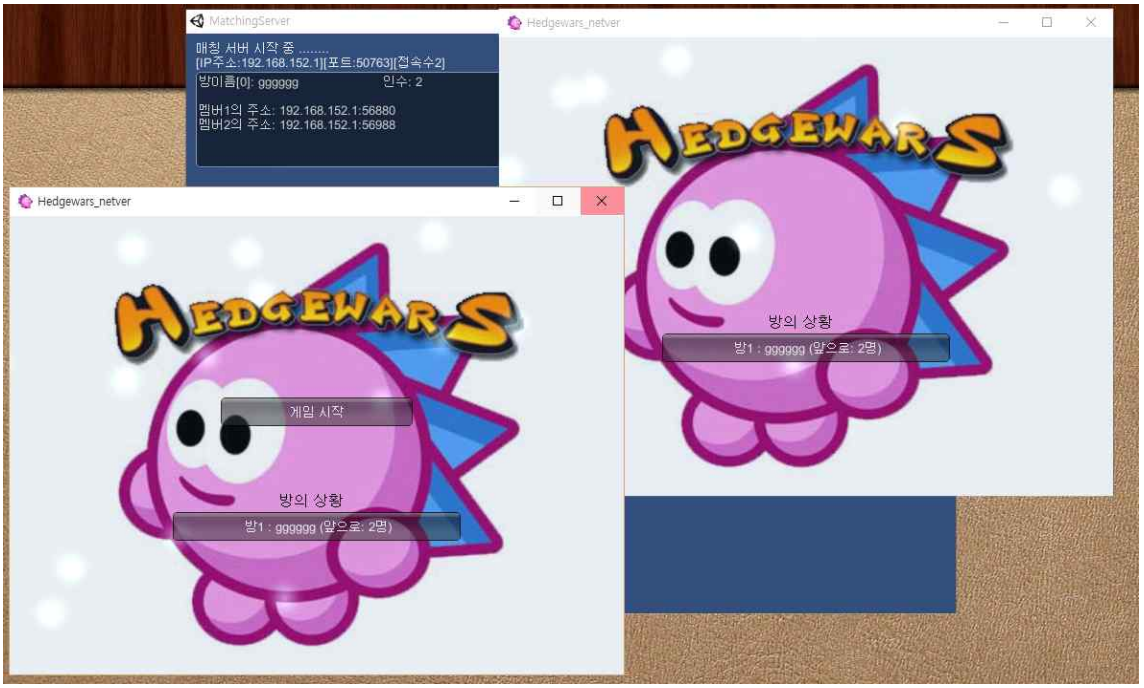
7. 실행 화면 캡쳐

매칭 서버 화면

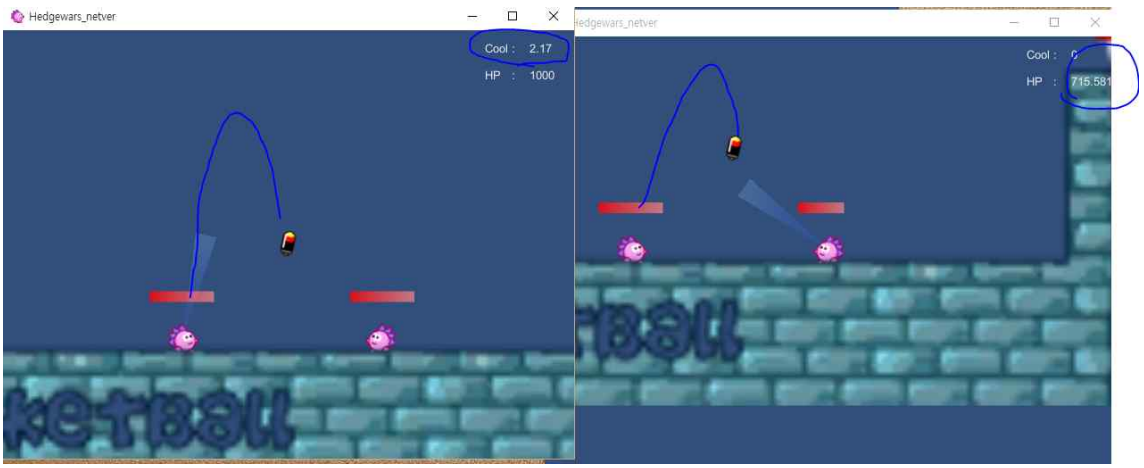
게임을 처음 실행한 화면


Hedgewars_netver
HEDGEWARS
방 이름
gggggg
방 만들기
방 찾기
발견한 방
MatchingServer
매칭 서버 시작 중 ....
[IP주소:192.168.152.1][포트:50763][접속수1]
방이름[0]: gggggg
인수: 1
멤버1의 주소: 192.168.152.1:56880
Hedgewars_netver
HEDGEWARS
게임 시작
방의 상황
방1 : gggggg (앞으로: 3명)

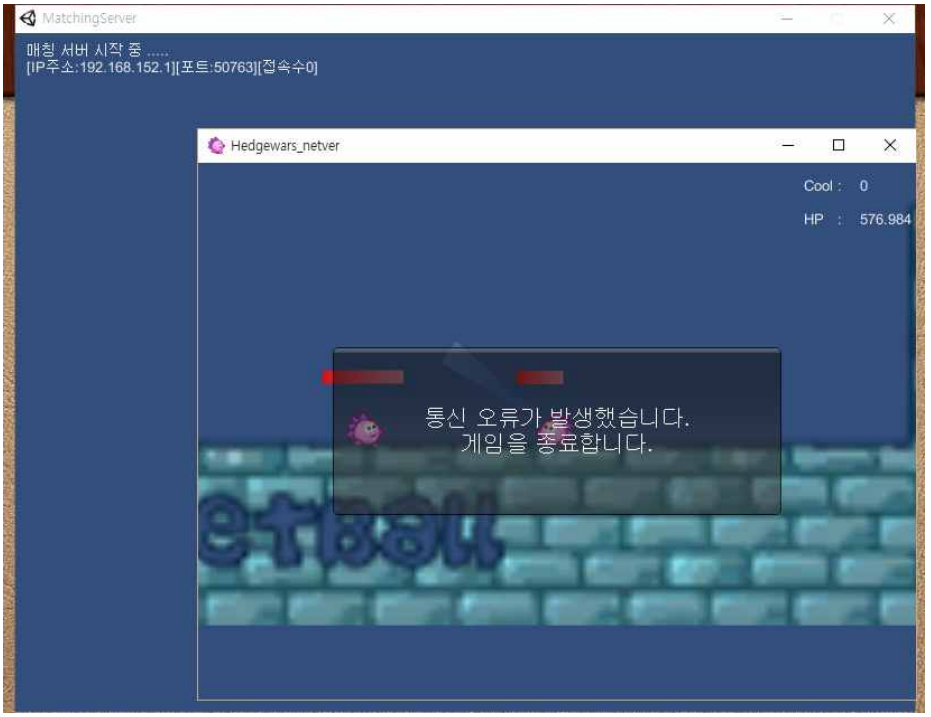
방을 생성한 화면

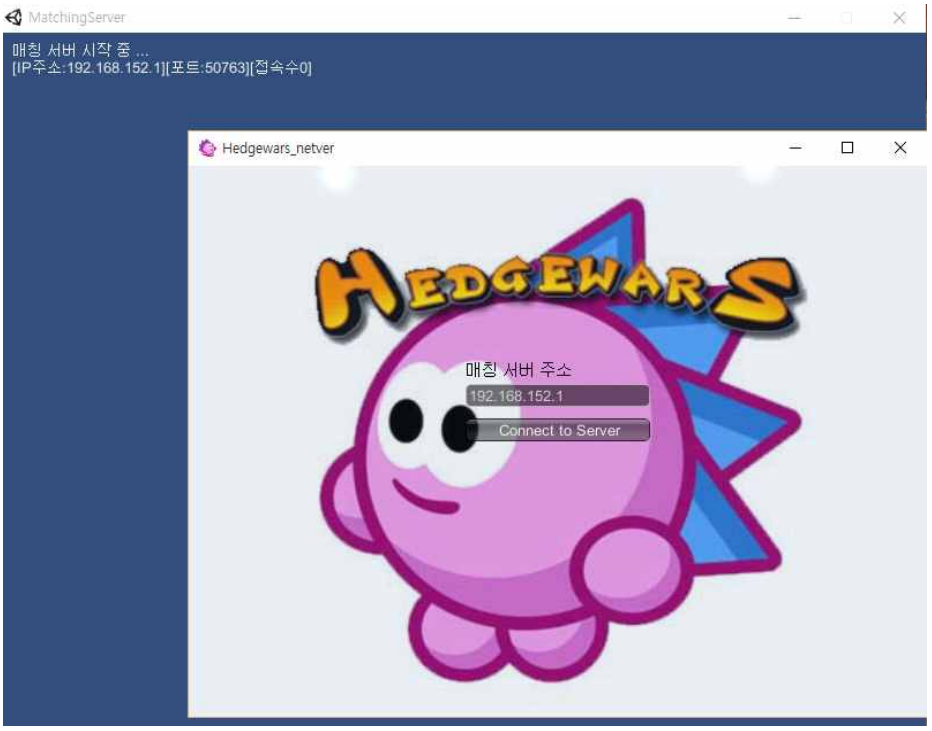


방에 입장한 화면



전투화면(투사체 발사)





호스트가 끊겼을 때 오류화면