게임서버와 클라이언트연동

편집: 김혜영

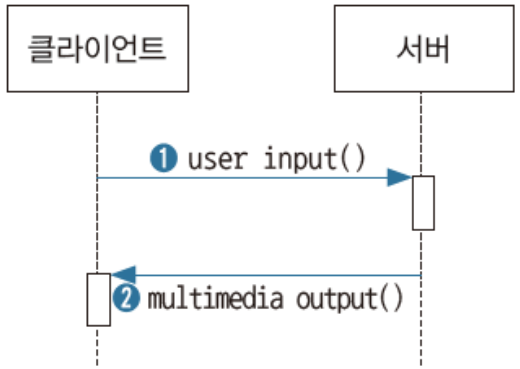
출처: 게임서버프로그래밍교과서, 배현직지음, 길벗

게임 플레이 네트워킹

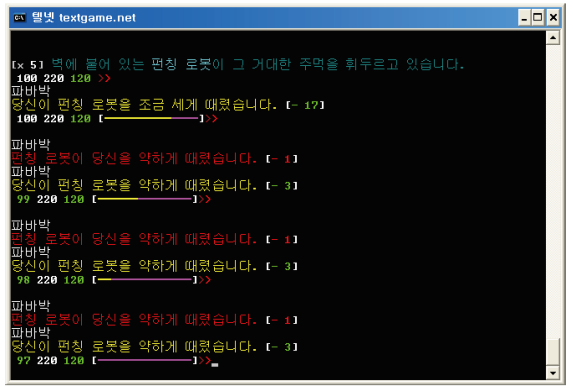
- 모든 역할을 서버에서 하기

  클라이언트가 하는 역할 : 1. 사용자 입력(키 입력, 마우스 좌표) / 2. 화면 출력

  서버에서 하는 역할 : 1. 게임 로직 연산/ 2. 화면 렌더링(그래픽 데이터 보유)/ 3. 화면 송출(비디오 스트리밍)



모든 역할을 서버에서 하는 모델



터미널은 텍스트 입출력 처리밖에 하지 못함

- 온라인 게임을 방해하는 레이턴시가 길어지는 요인

게임 플레이 네트워킹

- 온라인 게임을 방해하는 레이턴시가 길어지는 요인

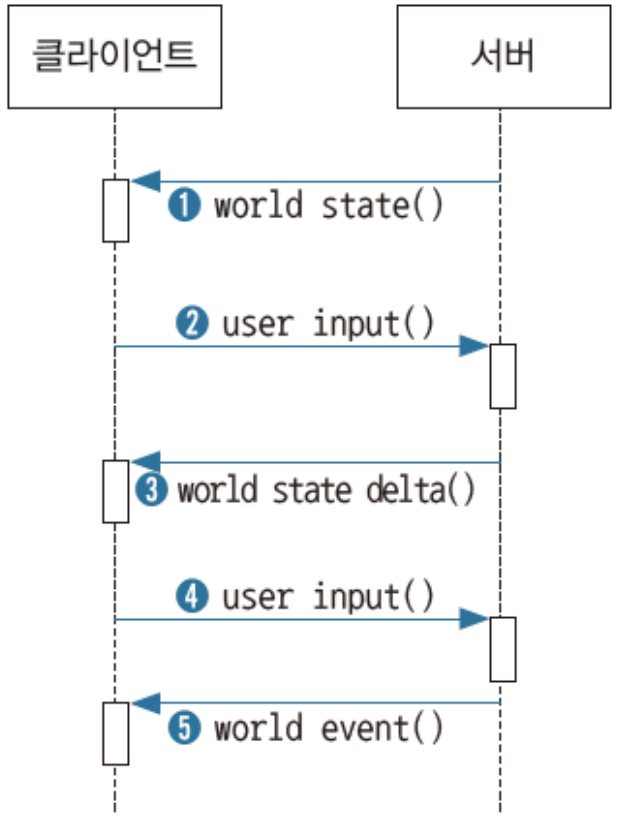
  1. 서버가 멀리 있으면 네트워킹 중에 레이턴시가 추가된다.
  2. 클라우드 서버 안에서 가상 머신은 다른 가상 머신이 CPU 사용량을 잠식하면서 조금씩 지연 시간이 있을 수 있다.
  3. 패킷 드롭으로 인한 재송신은 간헐적인 큰 지연 시간을 일으킨다.
  4. 인구가 낮은 국가에서는 인터넷이 느리다..
  5. 무선 네트워크에서는 레이턴시와 패킷 드롭률이 크게 증가한다.

- 서버 운영의 경제성 문제

  1. 고퀄리티 그래픽을 60프레임으로 렌더링하려면 그래픽카드 하나가 모든 능력을 동원해야 한다.
     서버에서 이것을 하려고 하면 서버에 접속해 있는 사용자 수만큼 그래픽카드를 동원해야 할 것임.
  2. 일반적인 MMORPG 서버 컴퓨터는 플레이어 처리를 2000개에서 2만 개까지 해야 제대로 경제성이 나온다.

게임 플레이 네트워킹

렌더링은
클라이언트서 하기



렌더링은 클라이언트에서 담당하는 형태

서버와 클라이언트가 하는 역할

- 서버는 렌더링을 위한 최소 정보인 게임 월드 상태만 클라이언트에 보낸다.
  월드 상태의 연산(scene update)은 서버에서 한다.
- 렌더링은 클라이언트에서 수행한다.
  이를 위한 그래픽 리소스는 클라이언트에서 보유한다.
- 서버와 클라이언트의 월드 상태(scene, 씬)를 동일하게 유지한다. 즉, 기화합니다.

서버와 클라이언트간의 대화

1. 서버는 월드(world), 즉 씬의 상태 전체 내용을 이미 알고 있지만,
   클라이언트는 이를 전혀 모르므로 서버는 클라이언트에 씬 상태 전체를 전송한다.
2. 게임 플레이어가 채팅이나 캐릭터 이동 등 어떤 행동을 취하면 이를 메시지로 만들어 서버로 보낸다. 그러면 서버는 이를 받아서 자기가 갖고 있는 월드 상태를 변화시키는 데 사용한다.
3. 월드 상태가 변하면 서버는 월드에서 변한 부분만 클라이언트에 보낸다.
   클라이언트는 이메시지를 받아서 자기가 갖고 있는 월드 데이터에 반영한다.
4. 게임 플레이어가 행동을 취할 때마다 각 행동에 대한 메시지를 만들어 서버에 보내 준다.
5. 월드 상태가 변할 때마다 메시지를 만들어 클라이언트에 보내 준다.

게임 플레이 네트워킹

- 클라이언트에서 서버로 보내는 플레이어가 취하는 행동 명령 메시지
  1. 채팅 메시지를 입력.
  2. 플레이어를 특정 방향으로 이동하라고 명령.
  3. 플레이어가 이동을 멈추라고 명령.
  4. 특정 아이템을 사용하라고 명령했다.

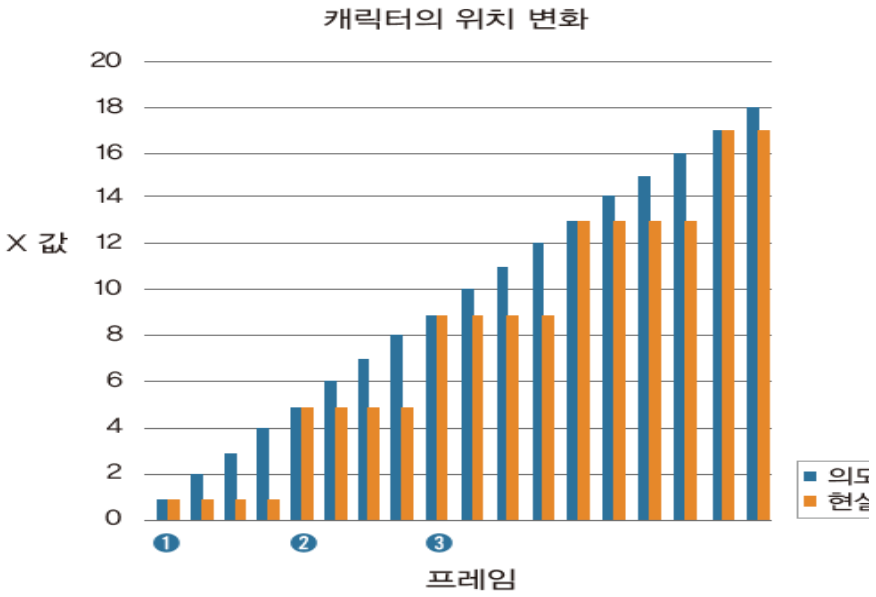
- 서버에서 클라이언트로 보내는 월드 상태 변화 메시지
  1. 캐릭터가 등장 (데이터 추가).
  2. 캐릭터가 특정 방향으로 이동 (데이터 변경).
  3. 캐릭터가 웃는다(데이터 변경).
  4. 캐릭터가 사라진다(데이터 소멸).

- 단발성 이벤트
  1. 특정 좌표에서 수류탄이 터짐(단발성 이벤트).

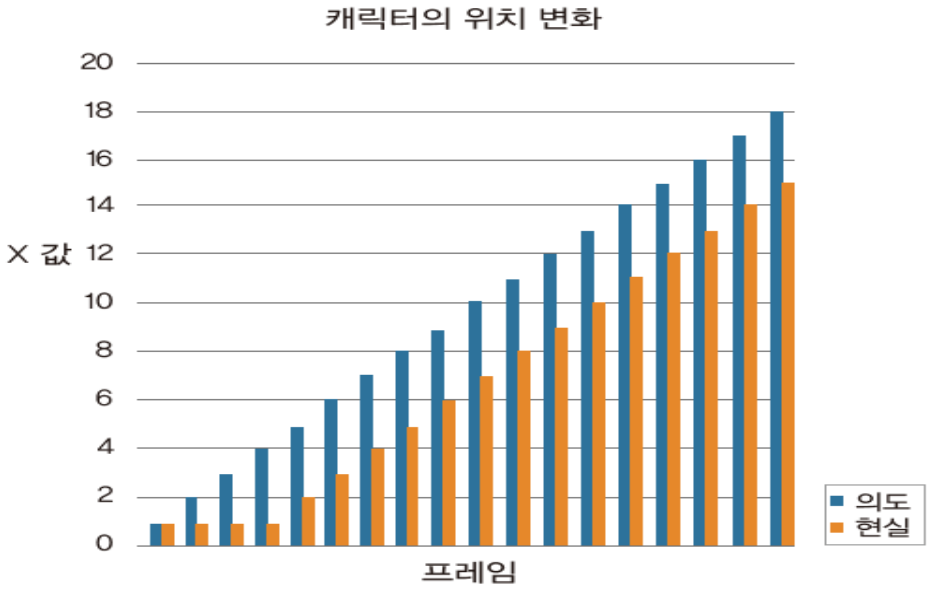
- 단발성 이벤트를 지속성 이벤트로 만들 때
  1. 특정 좌표에 수류탄이 생김.
  2. 수류탄이 터짐.
  3. 수류탄이 사라짐.
  4. 파티클 1초 모습.
  5. 파티클 2초 모습.
  6. 파티클이 사라짐.

게임 플레이 네트워킹

- 해결해야 할 문제
  1. 서버에서는 1/60초마다 월드 상태를 업데이트함.
  2. 서버는 1/60초마다 월드 상태의 변화를 클라이언트에 보냄.
  3. 클라이언트는 이를 지체 없이 받는다.
  4. 클라이언트는 받은 것을 자기의 월드 상태에 반영한다. 그리고 다음 렌더링 프레임에서 이를 그린다.



시간이 흐르면서 서버에서 캐릭터 위치(의도)와
클라이언트에서 캐릭터 위치(현실)



서버에서 캐릭터 위치(의도)와
클라이언트에서 부드럽게 보여지는 위치(현실

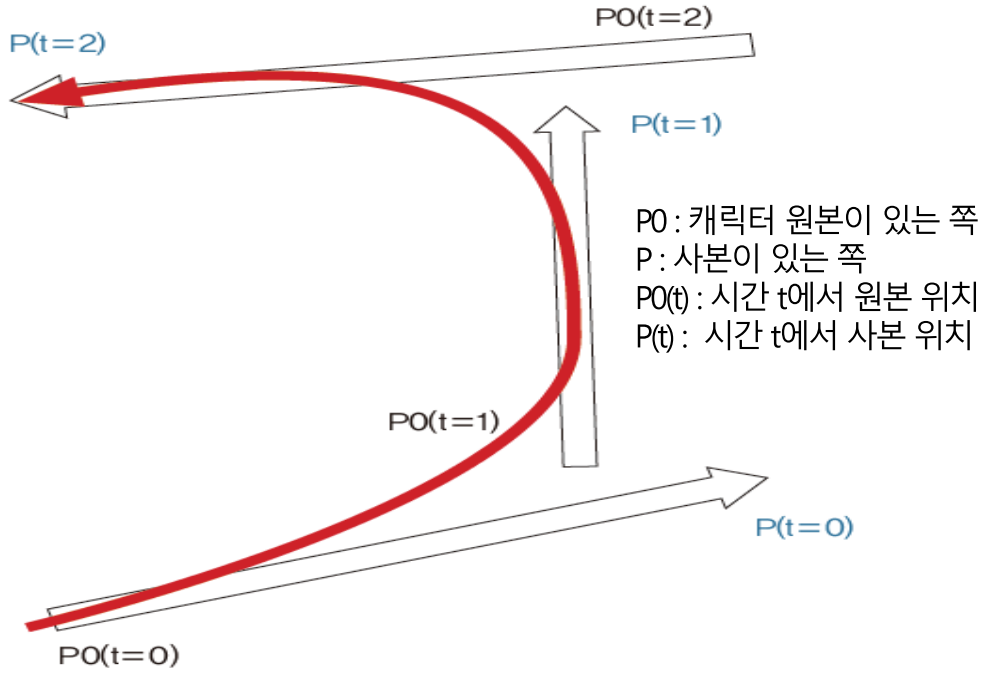
게임 플레이 네트워킹

- 추측항법

상대방 움직임을 어느 정도 예상해서 그 위치로 갈 수 있게 보정시키는 방법

두 기기 간의 레이턴시를 알고 있어야 함. 대표적인 측정방법에는 라운드 트립 레이턴시(round trip latency)가 있음

1. 기기 A에서 기기 B에 패킷을 보냄.
2. 기기 B는 이를 받으면 기기 A에 패킷을 보냄.
3. 기기 A는 1 과정의 시간과 현재 시간의 차이를 구하여 2로 나눔.



추측항법 처리 과정

t = 0일 때 P0(t = 0)을 보낸다.
마찬가지로 t = 1일 때 P0(t = 1)을 보낸다.
→저쪽 기기에서는 P(t = 0 + a), P(t = 1 + a), P(t = 2 + a)를 받는다.(a는 레이턴시)

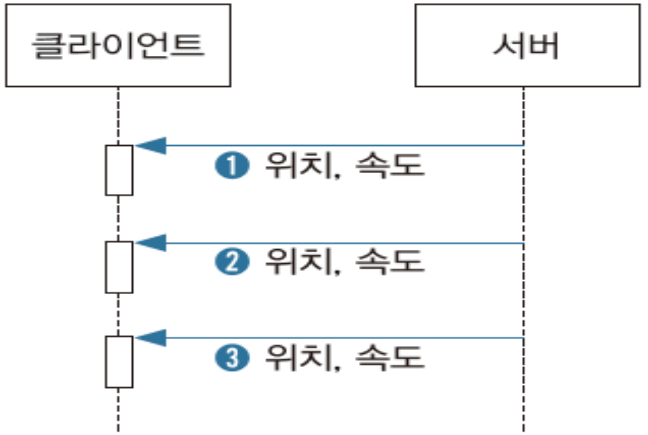
t = 0 + a 시점에서 실제 캐릭터 위치 :
P(t + a) = P0(t) + a * V0(t)

P0(t = 1)을 받으면 :
P(t = 1 + a) = P0(t = 1) + a * V0(t = 1)

캐릭터 위치뿐만 아니라 캐릭터가 바라보고 있는 방향이나 모션 상태 값도 추측항법을 적용하면 더 정확한 행동을 보여 줄 수 있다.

레이턴시 마스킹

1. 클라이언트에서 플레이어 캐릭터를 조종하는 명령을 서버에 보낸다.
2. 서버에서는 플레이어 캐릭터의 이동 연산을 한다.
3. 일정 시간(1/30~1/10초)마다 클라이언트에 이동 정보 메시지를 보낸다.
4. 클라이언트는 이동 정보 메시지를 받으면 추측항법으로 플레이어 캐릭터 위치를 부드럽게 만든 후 업데이트한다.

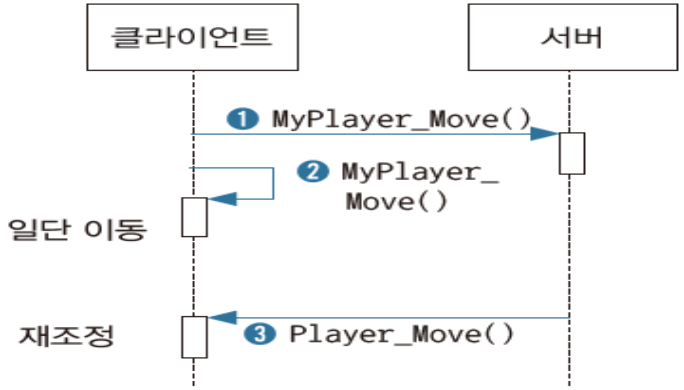


여러분이 만든 게임이 이렇게 작동 중이라고 가정

플레이어 캐릭터가 약간의 시간 지연 후 움직일 때 : 사소한 것들은 클라이언트에서 판단하기

클라이언트가 해킹되어서 플레이어 캐릭터의 이동 속도를 굉장히 빠르게 했다면?
 1. 클라이언트는 플레이어 캐릭터의 이동 정보를 서버에 보낸다.
 2. 서버는 이동 정보를 받아서 정상적인 값 범위에 있는지 검사한다.

또는
 1. 클라이언트는 플레이어 캐릭터의 명령 정보를 서버에 보낸다. 클라이언트에서는 자기 플레이어 캐릭터를 일방적으로 움직이나 캐릭터 이동 정보 메시지는 서버에 보내지 않는다
 2. 서버는 명령 정보에 따라 플레이어 캐릭터를 이동 처리합니다. 플레이어 캐릭터의 이동 정보메시지를 클라이언트에 보낸다.
 3. 클라이언트는 서버에서 이동 정보 메시지를 받으면, 앞서 일방적으로 움직였던 플레이어 캐릭터 위치를 무시하고 서버에서 메시지에 따라 캐릭터를 이동시킨다.



클라이언트와 서버 양쪽 모두가 캐릭터 이동 처리를 하고 있음

레이턴시 마스킹

- 일단 보여주고 나중에 얼렁뚱땅하기

1. 플레이어가 어떤 행동을 하면 행동 명령에 대한 메시지를 서버에 보낸다. 동시에 행동을 연출하는 일부 모습을 클라이언트에서 즉시 시작한다.
2. 서버에서는 행동 명령을 받아 처리하고, 플레이어 캐릭터에 가해야 하는 행동을 클라이언트에메시지로 보낸다.
3. 클라이언트는 이 메시지를 받으면 연출해야 하는 나머지 부분들을 클라이언트에서 보여 주기 시작한다.

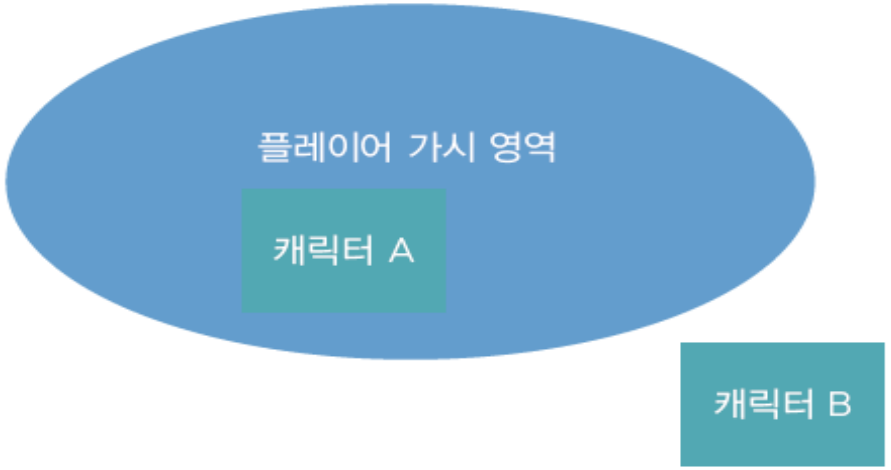
레이턴시가 없을 때	레이턴시가 높을 때
캐릭터가 공격을 할 때 이펙트가 동시에 발생	캐릭터는 그저 공격 모션만 취할 뿐 이펙트가 발생하지 않음. 잠시 후에 연출 이펙트 발생

심리적으로 더 좋아함

넓은 월드, 많은 캐릭터 처리

- 가시 영역 필터링

서버가 가진 월드 전체 상태 중에서 변화 하는 것 모두를 플레이어에게 보내 줄 필요가 없다.
플레이어의 가시 영역에 있는 것들만 보내도 된다.



플레이어의 가시 영역

클라이언트가 가지고 있어야 할 정보

자기 근처에 있는 다른 캐릭터들 정보

서버가 가지고 있어야 할 정보

1. 플레이어 각각에 대해서 각 플레이어가 볼 수 있는 캐릭터 목록
2. 캐릭터 각각에 대해서 자기 자신을 볼 수 있는 플레이어 목록

실시간 전략 시뮬레이션 게임에서 네트워크 동기화

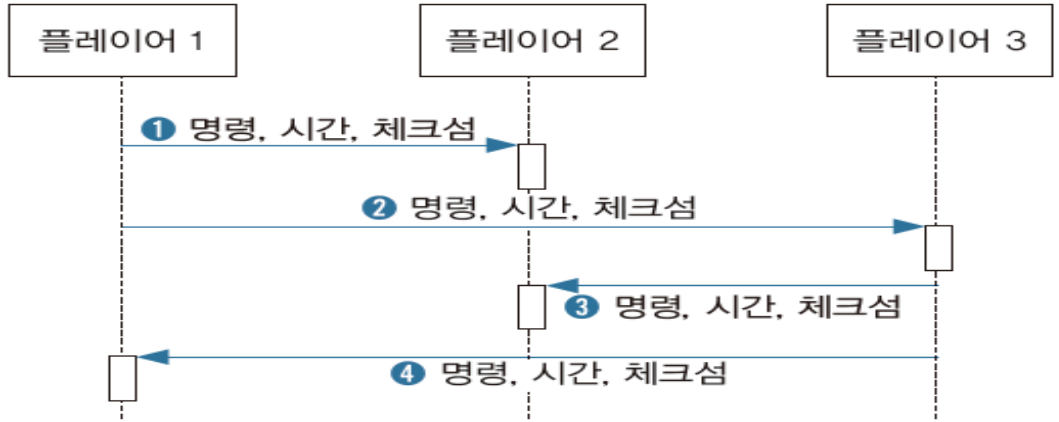
- 락스텝

컴퓨터 프로그램의 같은 상태에 같은 입력을 주면 같은 결과가 나온다는 원리를 응용, 구동원리는 아래와 같다.

1. 각 플레이어는 다른 플레이어들에게 입력 명령을 보낸다.
2. 플레이어의 입력 명령에 따라 모든 클라이언트가 동시에 씬 업데이트를 한다.

락스텝으로 얻는 효과

1. 각 클라이언트 플레이어의 입력 명령만 주고받으며, 씬을 구성하는 캐릭터의 이동 상태를 주고받지 않는다.
2. 입력 명령은 통신량이 상대적으로 매우 적다.



락스텝 동기화 방식에서 플레이어 간에 메시지 전송

컴퓨터간 통신 레이턴시가 거의 없을 때야 완벽하게 작동

→ 입력명령을 보내되 '언제 실행해야 하는지에 대한 미래 시간'을 같이 보내야 한다.

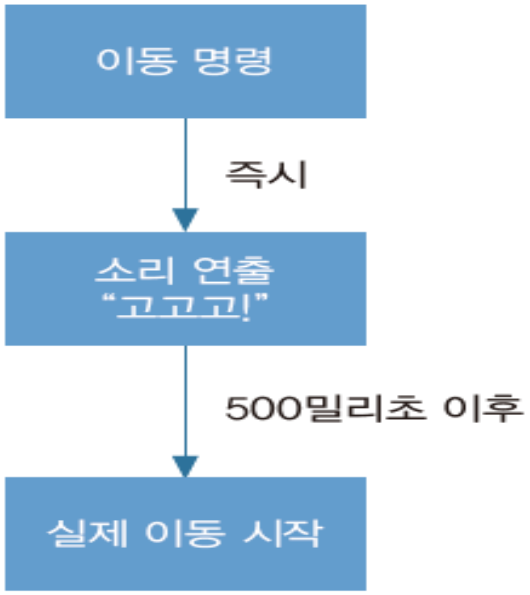
미래시간 산정법

현재 시간 + 왕복 레이턴시(RTT) / 2 + 임의의 일정 값

실시간 전략 시뮬레이션 게임에서 네트워크 동기화

- 전략 시뮬레이션 게임에서 레이턴시 마스킹

1. 실제로 캐릭터는 입력 명령을 처리하지 못했더라도 명령에 반응하는 연출만 보여 준다.
   일단 얼렁뚱땅 보여 주어 "즉시 반응하는구나!"를 느끼게 한다.
2. 이후에 미래 시간에 도달하면 실제로 캐릭터가 움직이게 한다.



락스텝 동기화 방식에서
레이턴시를 얼렁뚱땅 테크닉으로 감추기

락스텝 동기화 알고리즘의 한계

1. 다른 플레이어가 플레이하고 있는 게임 중간에 확 들어오는 것을 만들기 까다롭다.
2. 물리 엔진 등 게임 플레이에 관여하는 연산에 부동소수점을 쓸 수 없다. 개발할 때 가장 까다로운 점.
3. 플레이어 수가 많아지기 어렵다. 통신량이 플레이어 수에 비례해서 증가하기 때문.
4. 씬 업데이트가 일시 정지할 확률이 높다. 원활하게 게임을 플레이하려면 함께 플레이하는 플레이어 중에서 가장 레이턴시가 높은 사람을 기준으로 미래 시간이 결정되어야 한다.
5. 입력 명령의 속도에 민감한 게임에 부적합하다. 락스텝 동기화 알고리즘에서는 필연적으로캐릭터에게 넣은 행동은 미래 시간이 되어야 움직이기 때문.

실제 레이턴시 줄이기

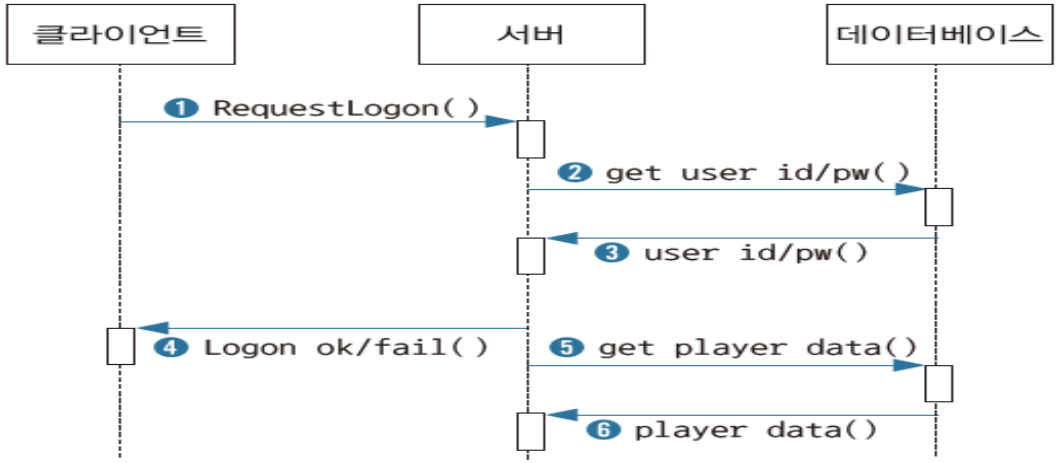
- 실제 레이턴시를 줄이는 방법

TCP 대신 UDP를 사용한다.
똑같은 양의 데이터를 보낸다고 하더라도 가급적 적은 수의 패킷으로 보낸다.
클라이언트와 서버 간 통신(C/S 네트워킹)과 클라이언트끼리 직접 통신하는 것(P2P)을 같이 섞어 쓴다.

게임 플레이 이외의 네트워킹

- 로그인 과정에서 클라이언트와 서버가 대화하는 주요 절차

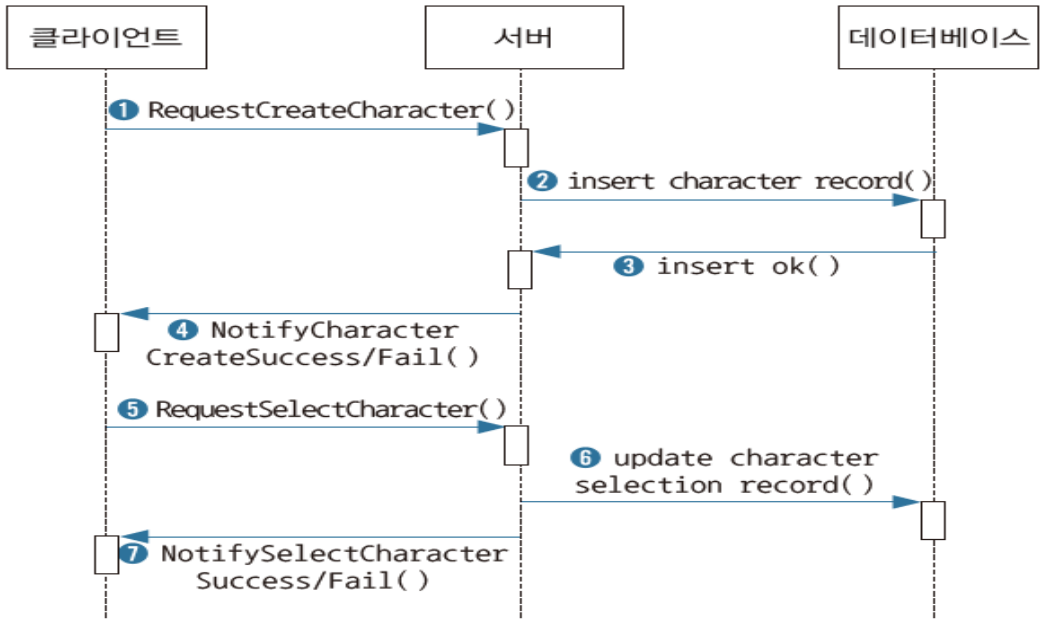
1. 로그온 요청 메시지를 서버로 전송한다.
2. 서버는 파일이나 데이터베이스에서 해당 유저의 ID와 비밀번호를 받아서 식별한다.
3. 식별 결과, 즉 로그온 처리 결과를 클라이언트에 통보한다.
4. 클라이언트가 이 통보를 받으면(예를 들어 로그온 성공이라는 통보를 받으면) 플레이어 정보를 데이터베이스에서 로딩해서 게임 서버 메모리에 보관한다. 이 과정은 있을 수도 있고 없을 수도 있다.



로그온 과정을 시퀀스 다이어그램으로 표현

게임 플레이 이외의 네트워킹

- 캐릭터를 만드는 과정



캐릭터를 만드는 과정을 시퀀스 다이어그램으로 표현

1. 클라이언트가 서버에 "내 캐릭터를 만들어라."라고 요청한다.
2. 서버는 데이터베이스에 "새 캐릭터에 대한 데이터 개체, 즉 레코드를 만들어라."라고 요청한다.
3. 데이터베이스는 이에 대한 응답을 한다. (여기서는 "성공했다."라는 응답을 받았다고 함.)
4. 서버는 클라이언트에 "캐릭터를 만드는 것이 성공했다."라고 알려 준다.
5. 클라이언트는 서버에 "내 캐릭터 중 OOO를 선택하겠다."라고 요청합니다.
6. 서버는 데이터베이스에 "캐릭터 OOO를 선택했다고 기록하자."라고 요청한다.
7. 서버는 클라이언트에 "캐릭터 OOO를 성공적으로 선택했다."라고 알려 준다.

데이터베이스에 기록하되 그 결과를 기다릴 필요가 없을 때는 다음 코드와 같이 데이터베이스에 기록하는 함수를 별도의 스레드나 비동기 함수 호출로 처리해도 된다.

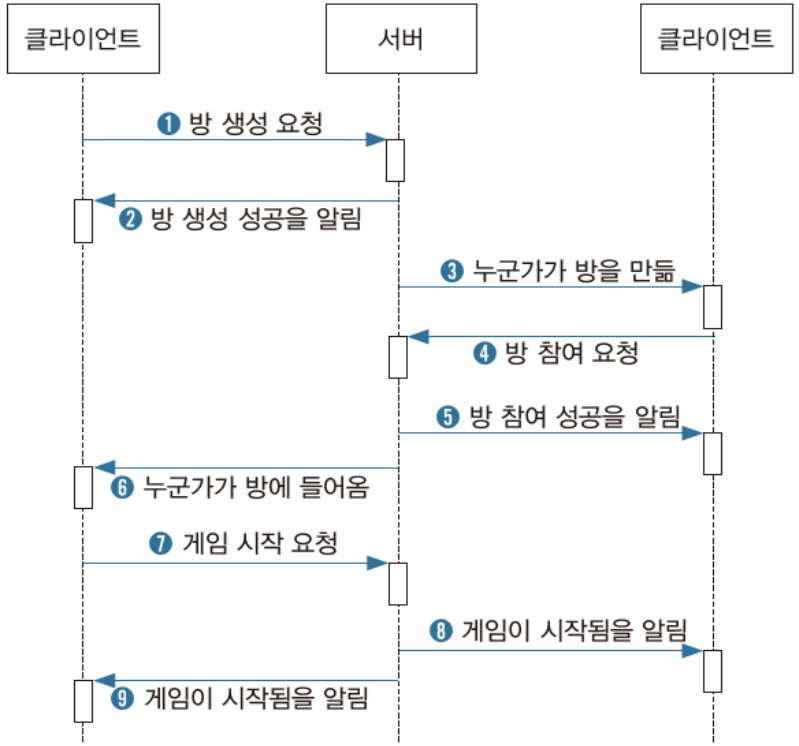
```
func()
{
        data.change(xxx);
        thread.doAsync(() = >
        {
                db.write(xxx);
        });
}
```

게임 플레이 이외의 네트워킹

- 매치메이킹_플레이어가 수동으로 방을 만들고, 다른 플레이어가 수동으로 방에 들어가는 방식 설계

1. 해킹을 방지하고자 방 만들기 혹은 들어가기 정보는 클라이언트에서 판단하지 말고 서버에서 모두 판단할 것.
2. 클라이언트에서는 일방적으로 판단하지 말고 서버에 요청하여 그 결과에 따라서 행동할 것.
3. 방 만들기 혹은 방 들어가기로 서버 내부의 방 목록이나 방 안의 플레이어 목록이 변할 때 클라이언트는 그 변화를 통보받을 것.

서버에서는 방 목록과 각 방에 들어가 있는 플레이어, 즉 방 안의 플레이어 목록을 갖고 있어야 하며,
게임 플레이 중인 방의 상태 데이터도 갖고 있어야 한다.



두 플레이어가 방을 만들고 들어가는 과정의 시퀀스도

1. 왼쪽 클라이언트는 서버에 "방 하나를 만들어라."라고 요청.
2. 서버는 방을 만들고 만든 방을 자신의 메모리에 보관하고 클라이언트에 "방 하나를 만들었다."라고 응답.
3. 이미 서버 접속 상태를 유지하고 있던 오른쪽 클라이언트에 "방 하나가 새로 만들어졌으니 도 알고 있어라."라고 통보.
4. 오른쪽 클라이언트의 플레이어는 이 방에 들어가고 싶다. 클라이언트는 서버에 "나는 이방 들어가겠다."라고 요청.
5. 서버는 오른쪽 클라이언트에 "방에 성공적으로 들어왔다."라고 응답한다. 동시에 방 정보와 에 들어간 왼쪽 클라이언트의 존재도 알려 줌.
6. 오른쪽 클라이언트가 방에 들어갔음을 왼쪽 클라이언트도 이제 알아야 한다.
   따라서 서버는 자체 클라이언트에 "오른쪽 클라이언트가 네 방에 들어왔다."라고 통보. 이제 양
   클라이언트는 서로의 존재를 알고 있으며, 둘 다 방에 들어간 상태.
7. 왼쪽 클라이언트는 서버에 "게임을 시작하자."라고 요청.
8. 서버는 오른쪽 클라이언트에 "게임을 시작하라."라고 통보한다.
9. 서버는 왼쪽 클라이언트에 "게임을 시작하라."라고 통보한다.

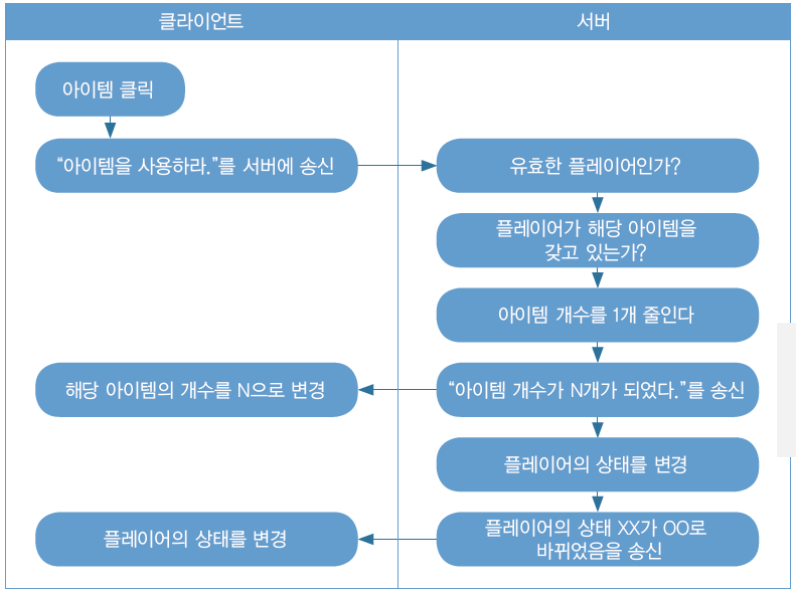
게임 플레이 이외의 네트워킹

- 게임 로직을 개발하는 과정에서 서버와 클라이언트의 대화 규칙

1. 클라이언트에서는 요청을 보낸다.
2. 서버에서는 그 요청을 받아 결과를 판단한다.
3. 요청 결과에 영향을 받을 다른 클라이언트가 서버에 있으면 그 클라이언트에 통보한다.

- 아이템 사용하기 과정 설계하기

1. 클라이언트에서는 사용하는 아이템의 식별자(ID)를 인자로 담은 메시지를 서버에 전송한다.
2. 서버에서는 해당 아이템을 플레이어가 사용할 수 있는 상태인지 판단해서 아이템 사용 결과를 판정한다.
3. 그 결과를 클라이언트에 알려 주어 아이템이 사용됨을 고지한다.



거시적인 흐름을 표현해야 할 때는 시퀀스 다이어그램을 사용하고, 세부적인 로직을 표현해야 할 때는 액티비티 다이어그램을 사용한다.

아이템 사용 처리를 액티비티 다이어그램으로 표현