

게임서버프로그래밍

2019년 2학기

한국산업기술대학교
게임공학부

정내훈

4장 게임 서버와 클라이언트

- 1 | 패키지 게임에서 게임 서버
- 2 | 온라인 게임에서 게임 서버
- 3 | 서버의 역할
- 4 | 게임 클라이언트와 서버의 상호 작용
- 5 | 게임 서버가 하는 일
- 6 | 게임 서버의 품질
- 7 | 플레이어 정보의 저장
- 8 | 서버 구동 환경
- 9 | 서버 개발 지침
- 10 | 더 읽을 거리

4.1 | 패키지 게임에서 게임 서버

- 패키지 게임
 - 소규모의 플레이어가 같이 모여서 하는 온라인 게임. Offline플레이가 가능한 경우도 많음, MOG(Multiplayer Online Game)이라고도 함
- 서버

플레이어 세 명 이상이 게임을 하려면 그들의 게임 플레이 상태를 저장하는 곳이 필요하고, 이를 위해 플레이어 중 한 명의 컴퓨터가 모든 플레이어의 게임 플레이 상태를 취합해서 유지하는 역할을 해야함 그곳을 서버(server)라고 부름

패키지 게임에서 게임 서버가 하는 역할은 지금 플레이어가 두세 명 혹은 많아도 십여 명 참여하여 게임 플레이를 하는 상태, 즉 세션(session) 처리를 담당하는 것

데디케이트드 서버 (dedicated server) : 클라이언트 프로그램과 같은 엔진을 사용하지만 렌더링과 사용자 입력 처리를 받지 않고, 순전히 클라이언트의 연결을 받는 세션을 처리만 하는 프로그램이 따로 운용하는 경우

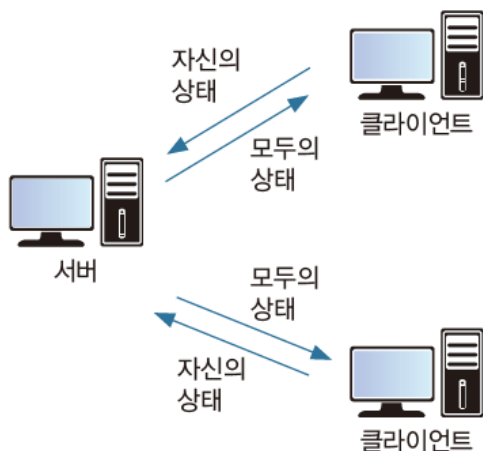


그림 4-2 패키지 게임에서 게임 서버와 클라이언트



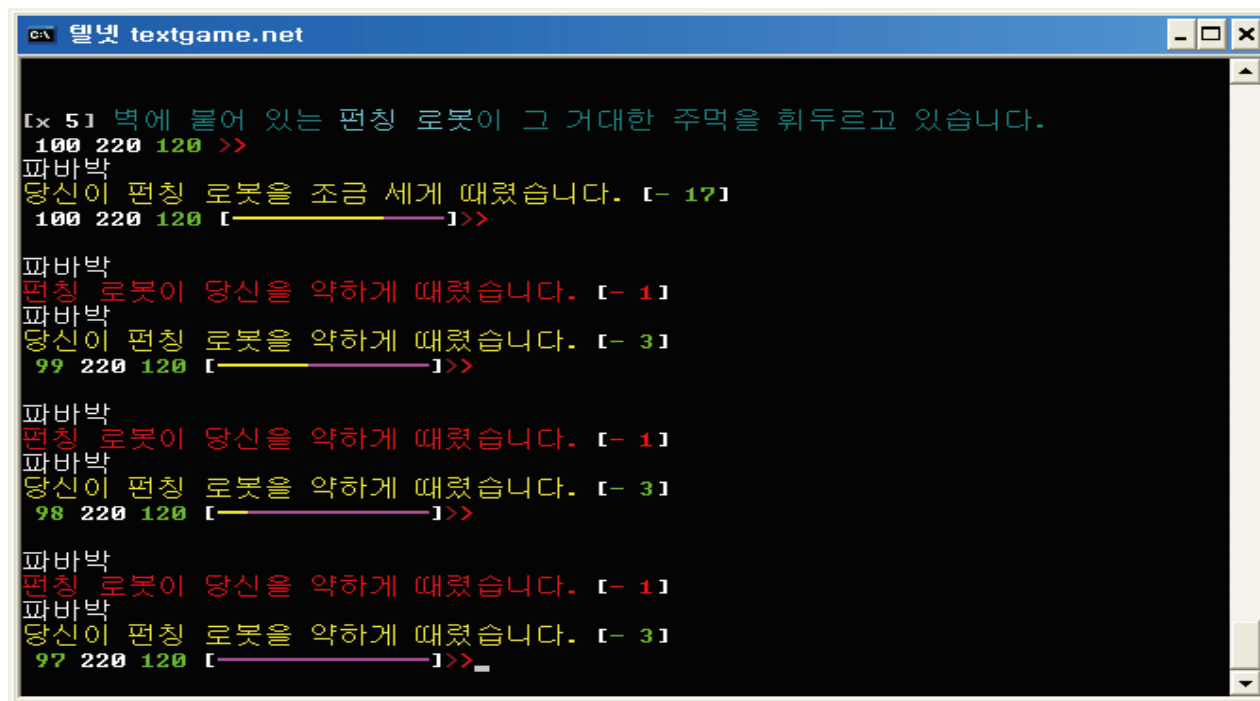
그림 4-3 Quake의 데디케이트드 서버

4.2 | 온라인 게임에서 게임 서버

- MUD(Multi User Dungeon)

여러 사람이 동시에 던전에 들어가서 모험하는 방식의 게임.

이때 MUD 게임은 그래픽 화면을 보여 주는 클라이언트 프로그램 없이, 텍스트 입출력만 받을 수 있는 콘솔(터미널)을 이용해서 게임을 즐기는 방식으로, 온라인 게임에서 본격적인 게임 서버는 이때부터 시작된다.



```
C:\> 텍스트 textgame.net

[ x 5 ] 벽에 붙어 있는 펀칭 로봇이 그 거대한 주먹을 휘두르고 있습니다.
100 220 120 >>
파바박
당신이 펀칭 로봇을 조금 세게 때렸습니다. [- 17]
100 220 120 [—————]>>

파바박
펀칭 로봇이 당신을 약하게 때렸습니다. [- 1]
파바박
당신이 펀칭 로봇을 약하게 때렸습니다. [- 3]
99 220 120 [—————]>>

파바박
펀칭 로봇이 당신을 약하게 때렸습니다. [- 1]
파바박
당신이 펀칭 로봇을 약하게 때렸습니다. [- 3]
98 220 120 [—————]>>

파바박
펀칭 로봇이 당신을 약하게 때렸습니다. [- 1]
파바박
당신이 펀칭 로봇을 약하게 때렸습니다. [- 3]
97 220 120 [—————]>>_
```

그림 4-4 텍스트로만 표시되는 MUD 게임

4.3 | 서버의 역할

- 싱글플레이 게임을 처리하기(게임 루프)

입력받기: 키보드, 마우스, 터치 스크린, 마이크, 카메라 등으로 컴퓨터가 정보를 획득하는 과정.

게임 로직 처리하기: 게임 정보를 담고 있는 상태인 세션은 보통 1초에 60번 상태 변화를 한다. 상태 변화를 하는 과정을 게임 로직 처리라고 한다. 게임 로직 처리 과정 중에는 게임 플레이 판정, 가령 플레이어가 어느 캐릭터에 대미지를 주었는지 혹은 캐릭터가 어느 몬스터 캐릭터에 대미지를 받았는지 등을 계산한다.

렌더링: 변화된 상태를 화면에 표현한다.

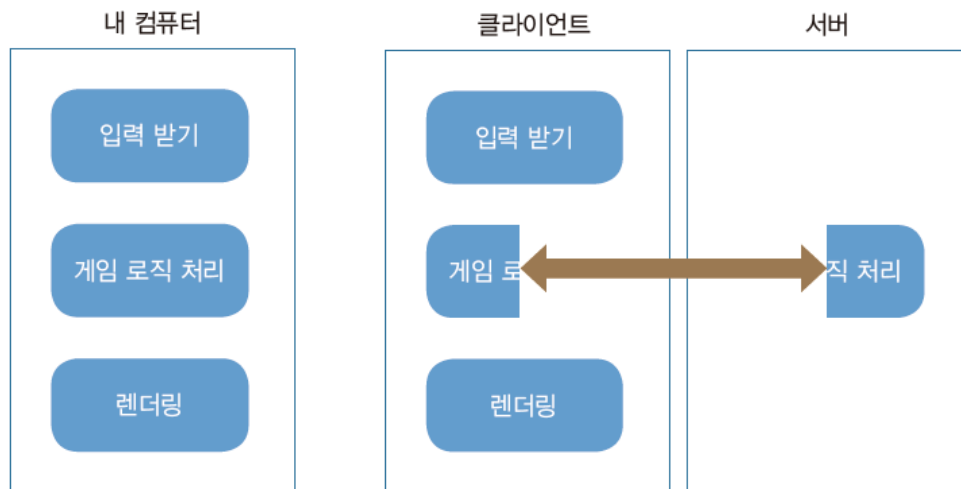


그림 4-5 게임 루프

클라이언트 : 1인용 게임에서 플레이어가 직접 만지는 컴퓨터.

대부분 온라인 게임에서는 클라이언트에서 게임 로직을 처리하는 역할 일부를 떼어 내 서버로 옮긴다. 그리고 클라이언트와 서버 간에는 컴퓨터 네트워크를 통해 서로 데이터를 주고받는다.

그림 4-6 게임 루프의 일부를 서버가 가져간 상태

4.4 | 게임 클라이언트와 서버의 상호 작용

- 게임 클라이언트와 서버의 상호 작용

연결	요청-응답	능동적 통보	연결 해제
<p>최초로 클라이언트가 서버와 데이터를 주고받을 준비를 하는 것</p> <p>서버는 클라이언트의신원을 확인하여 연결을 계속 유지할지 아니면 추방(연결 해제)할지 판단해야 한다</p>	<p>클라이언트는 서버에 메시지를 보내고, 서버는 이를 처리한 후 결과를 응답해 줌</p>	<p>클라이언트에서 요청을 보낸 적도 없는데 서버에서 능동적으로 통보</p>	-

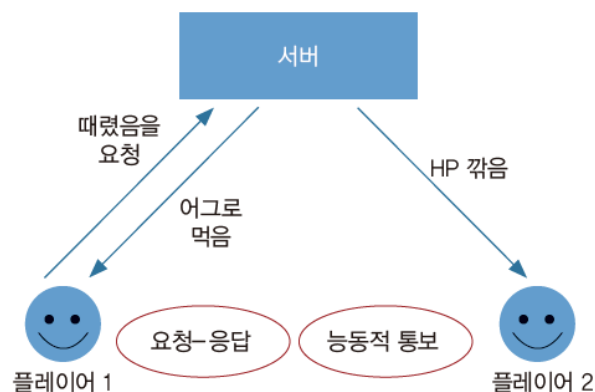


그림 4-7 두 클라이언트와 서버 간 상호 작용

4.5 | 게임 서버가 하는 일

- 게임 서버가 주로 하는 일

여러 사용자와 상호 작용

클라이언트에서 해킹 당하면 안 되는 처리

플레이어의 상태 보관

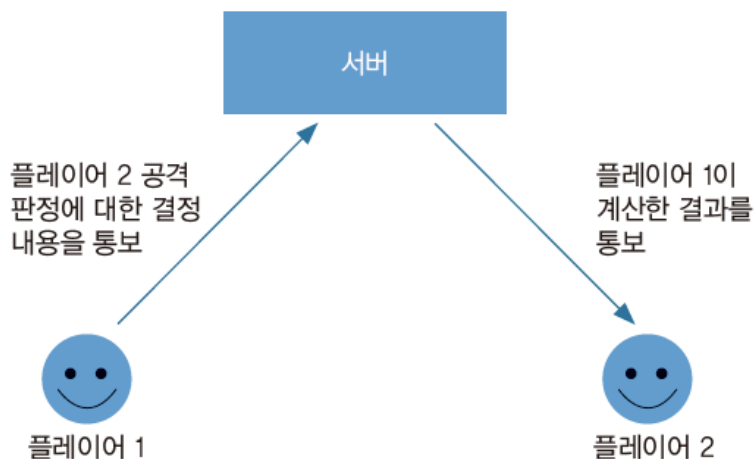


그림 4-8 클라이언트에서 일방적으로 판단하는 경우

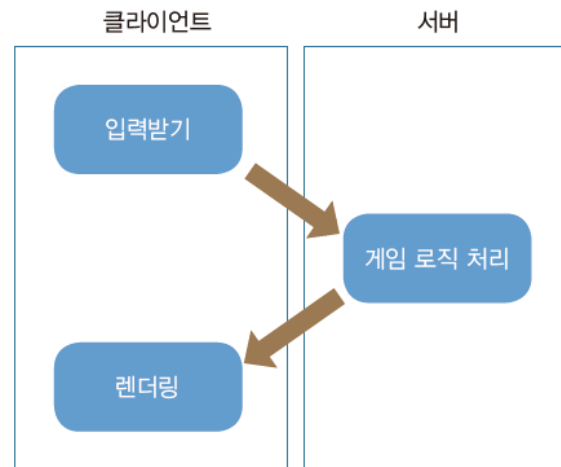


그림 4-9 게임 로직을 서버에서만 처리

4.6 | 게임 서버의 품질

4.6.1 안정성

“게임 서버가 얼마나 죽지 않는가”를 의미.

소프트웨어 측면에서 안정성에 악영향을 주는 주된 요인은 버그임. 구조적으로 설계가 잘못되었거나 사소한 코딩 실수가 서버의 안정성을 위협함.

안정성에는 “게임 서버가 얼마나 오작동을 하지 않는가”도 포함됨.

```
Mob mobs[100];
Player_AttackMob(int mobIndex)
{
    // mobIndex가 범위를 벗어난다면?
    mobs[mobIndex].m_hp -= 10;
}
```

개발 과정에서 안정성을 위해 노력할 것

1. 치밀한 개발과 유닛 테스트	2. 80:20 법칙	3. 1인 이상의 코드 리뷰	4. 가정하지 말고 검증하라
-------------------------	----------------	-----------------------	-----------------------

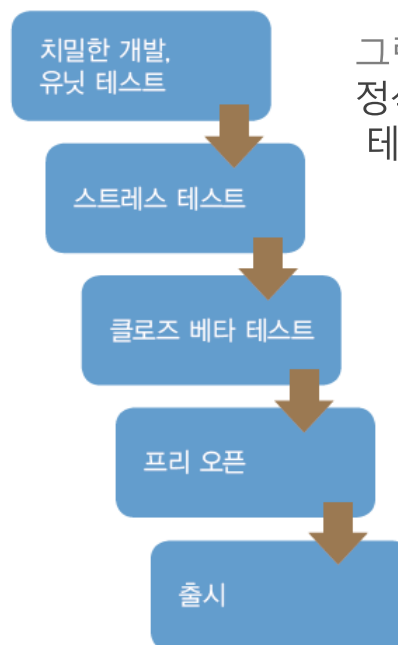


그림 4-10
정식 출시까지 거치는
테스트

4.6 | 게임 서버의 품질

- 서버의 불안정을 극복하는 방법
 1. 서버가 죽더라도 최대한 빨리 다시 살아나게 한다.
 2. 서버는 죽더라도 최대한 적은 서비스만 죽게 한다.
 3. 서버 오작동에 대해서 기록(로그)을 남길 수 있게 한다.

4.6 | 게임 서버의 품질

4.6.2 확장성

서버를 얼마나 많이 설치할 수 있느냐를 의미.

게임 사용자 측면에서 '사용자수가 늘어나더라도 서비스 품질이 떨어지지 않고 유지되느냐'가 곧 확장성을 의미한다.

서버 확장성을 올리는 방법에는 크게 수직적 확장(scale-up)과 수평적 확장(scale-out)이 있다.

구분	수직적 확장	수평적 확장
확장 종류	서버 머신의 부품을 업그레이드 혹은 서버 머신안의 CPU, RAM을 증설한다. 서버 프로그램을 최적화 한다.	서버 머신의 개수를 증설한다.
서버 소프트웨어 설계 비용	HW 확장: 없다. SW 최적화 : 매우 높다.	높다.
확장 설치 비용	높다(기하급수적으로 높아진다).	낮다(선형적으로 높아진다).
과부하 지점	서버 컴퓨터 자체	네트워크 장치
오류 가능성	HW : 낮다 SW : 매우 높다(고도화된 멀티 쓰레드 프로그래밍 필요).	높다(여러 머신에 걸쳐 비동기 프로그래밍 방식으로 작동하므로).
단위 처리 속도 요구량	높다(로컬 컴퓨터의 CPU와 RAM만 사용).	낮다(여러 서버 컴퓨터 간의 메시징이 오가면서 처리하므로).
확장 가능 총량	낮다(서버 컴퓨터 한 대의 성능만 사용하므로).	높다(여러 서버 컴퓨터로 부하가 분산되므로).

표 4-1 수직적 확장과 수평적 확장

4.6 | 게임 서버의 품질

- 4.6.3 성능
성능은 기본적으로 얼마나 빨리 처리하는지를 의미한다.

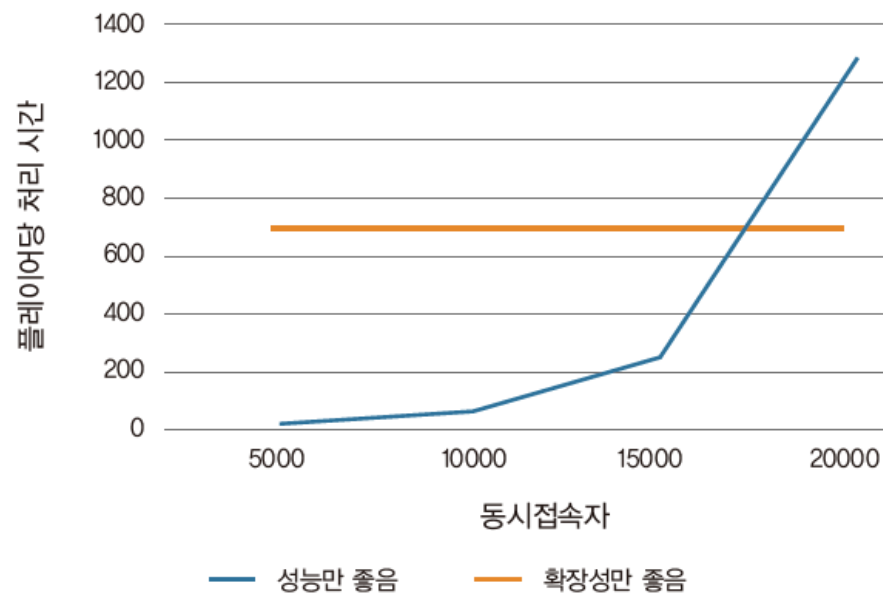
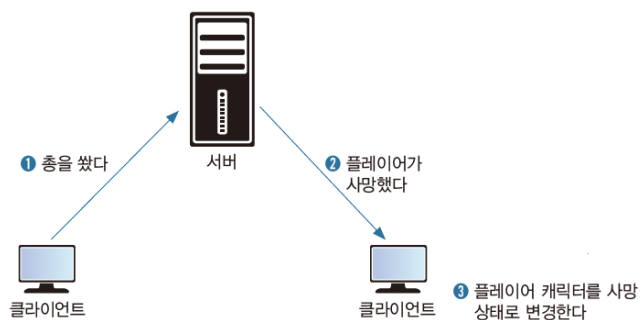


그림 4-12 성능만 좋거나 확장성만 좋을 때 그래프

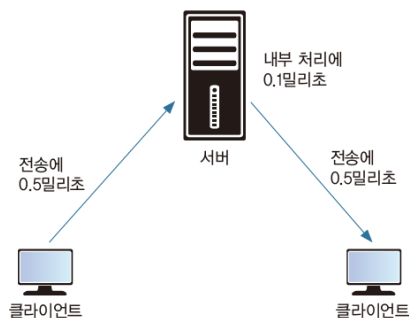
4.6 | 게임 서버의 품질

- FPS 게임에서 처리 과정



- 1 플레이어가 저격총을 쏘는 순간 클라이언트는 서버에 총알을 쏘다는 의미의 메시지를 보낸다.
- 2 서버는 이를 받아서 메시지를 처리, 그리고 "헤드샷 때문에 플레이어가 사망했다."와 같은 메시지 처리 결과를 다른 플레이어에게 보내야 한다.
- 3 클라이언트에서는 이 메시지를 받고 플레이어가 죽는 모습으로 바뀌어야 한다.

그림 4-13 FPS 게임에서 처리 과정



총 걸리는 시간은 1.1밀리초

→서버가 메시지 1개를 처리하는 데 0.1밀리초가 걸린다면, 1초에 최대 처리할 수 있는 메시지 수가 1만 개임을 의미하므로 0.1밀리초는 게임 서버 입장에서는 납득할 수 없는 매우 긴 시간이다.

그림 4-14 순서대로 0.5ms + 0.1ms + 0.5ms가 걸린다고 가정

4.6 | 게임 서버의 품질

- 게임 서버의 성능을 높이는 법 : 서버의 단위 처리 속도를 높이기

서버의 단위 처리 속도를 높이려면, 프로그램이 더 빠르게 실행할 수 있게 코드 최적화나 알고리즘 최적화를 실행

RPG 게임에서는 몬스터나 플레이어 외 캐릭터(Non Player Character, NPC)의 길찾기 알고리즘(path finding)이 소요하는 처리 시간을 $O(1)$ 로 개선하고자 path table 테크닉을 쓰는 것도 최적화 방법 중 하나.

- 게임 서버의 성능을 높이는 법 : 서버의 과부하 영역을 분산

코드 프로파일링(code profiling) 이용하기 : 어떤 함수가 처리 시간을 많이 차지하는지 발견한 후 그것에 집중해서 성능을 개선하함.

함수 A의 처리 속도를 더 높일 수 있는 방법이 더 이상 없고, 그렇다고 함수 A를 실행하는 빈도를 낮출 수 있는 방법도 없다면 분산이 필요하다.

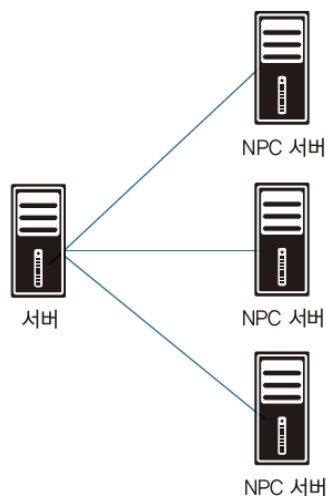


그림 4-15 NPC의 길찾기 알고리즘 분산

4.6 | 게임 서버의 품질

- 게임 서버의 성능을 높이는 법 : 네트워크 프로토콜 최적화

메시지의 양을 줄이기 : 양 자체를 줄이거나 압축하기



그림 4-16 네트워크 최적화 프로토콜 1: 양자화

메시지 교환 횟수 줄이기

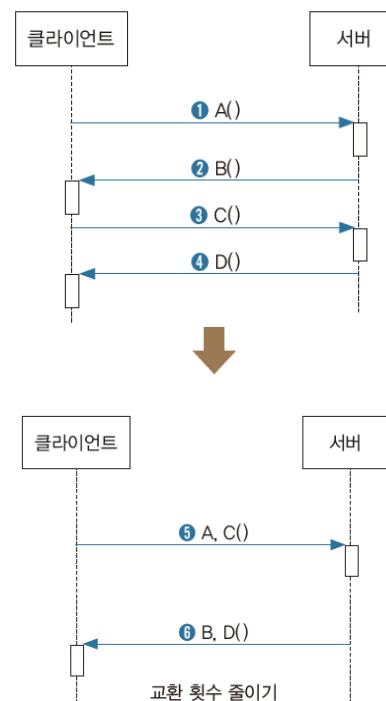


그림 4-16 네트워크 최적화 프로토콜 2
: 메시지 교환 횟수 줄이기

4.6 | 게임 서버의 품질

- 게임 서버의 성능을 높이는 법 : 네트워크 전송 시간 줄이기

고품질 네트워크 회선을 가진 데이터센터에 서버를 설치하기.

지리적으로 가까운 데이터센터에 서버들을 분산해서 설치하기.

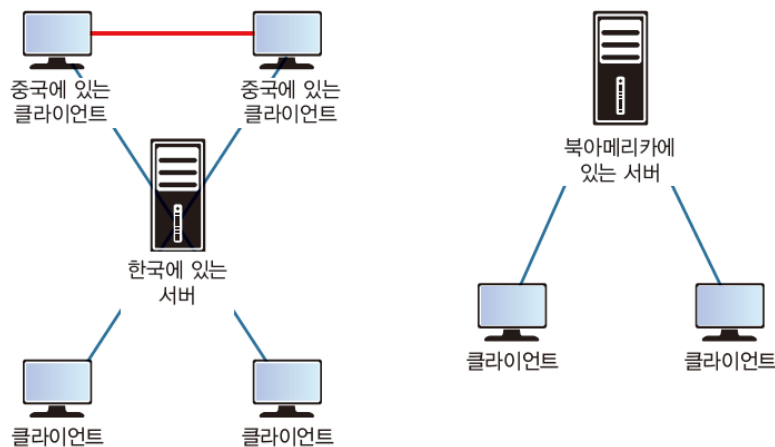


그림 4-18 지리적으로 가까운 데이터센터에 서버 분산

- 게임 서버의 성능을 높이는 법 : 서버를 거치지 않고 클라이언트끼리 직접 통신하게 하기

P2P(peer-to-peer) 네트워킹 : 클라이언트끼리 직접 메시지를 주고받는 것.

P2P 네트워킹은 서버에 걸리는 부담도 줄여 준다는 효과가 있으므로 파일 전송이나 음성 및 화상 채팅처럼 클라이언트끼리 주고받는 데이터의 양이 클 때 더욱 효과적임.

4.6 | 게임 서버의 품질

4.6.4 관리 편의성

데몬(daemon)/서비스(service) : 운영체제에 등록된 백그라운드 프로그램.

데몬은 프로그램 종료 같은 극히 제한적인 종류의 명령만 처리할 수 있으며, 백그라운드로 작동하는 프로그램이기 때문에 GUI는 물론이고 콘솔에 텍스트를 출력하면 화면에 아무것도 나타나지 않는다.

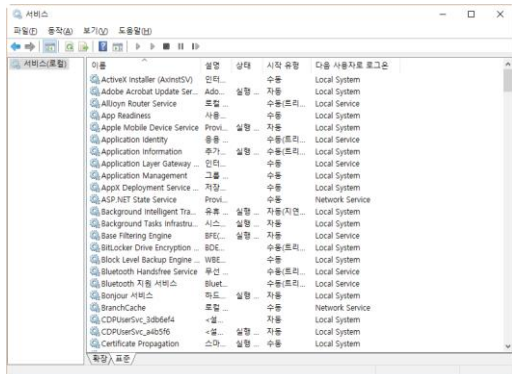


그림 4-19 윈도우에서 서비스들

관리 도구의 역할

서버 켜기/끄기
동시접속자 수 보기
CPU, RAM 사용량 보기



그림 4-20 서버 프로그램과 통신하는 관리 도구

4.7 | 플레이어 정보의 저장

- 온라인 게임에서 플레이어의 데이터 관리 중요성은 점점 커지고 있다.

보통 싱글플레이 게임의 플레이어 정보는 게임을 구동하는 컴퓨터 자체의 디스크에 저장된다.

온라인 게임에서 플레이어 정보를 클라이언트에 저장하지 않는 이유

1. 해킹에 취약하다.
2. 같은 사용자가 다른 기기를 사용할 때 문제가 된다.

그래서 대다수 온라인 게임은 플레이어 정보를 클라이언트가 아니라 서버에 저장한다.

플레이어의 데이터는 게임 서버에 직접 저장하는 경우도 있지만, 게임 서비스의 규모가 큰 경우에는 게임 서버의 디스크에 직접 저장하지 않고 별도의 서버 머신을 쓰기도 한다. 그리고 그 별도의 머신은 데이터베이스라는 소프트웨어 안에서 플레이어 데이터를 관리한다.



그림 4-1 규모가 클 때는 데이터베이스 머신을 따로 둬

4.7 | 플레이어 정보의 저장

• 데이터베이스 소프트웨어를 이용해서 저장하는 이유

1. 데이터 관리와 분석을 빠르게 할 수 있다.
2. 강력한 데이터 복원 기능이 있다.
3. '전부 아니면 전무'로 데이터를 변경할 수 있다.
4. 데이터 일관성을 유지시켜 준다.
5. 처리가 2개 이상 동시에 실행될 때 한 데이터가 동시에 여러 데이터를 액세스하면서 이상한 결과가 나오는 문제를 막아 주는 기능이 있다.
6. 장애에 대한 내성이 강하다.

표 4-2 단순 파일 시스템과 데이터베이스 장단점

구분	단순 파일	데이터베이스	비고
소프트웨어 비용	없다.	없거나 높다.	오픈 소스 제품은 제한적으로 무료다.
저장 및 로딩 속도	빠르다.	느리다	데이터베이스도 결국 파일 시스템을 사용한다.
데이터 관리, 분석 속도	느리다.	빠르다.	데이터베이스는 빠른 검색을 위한 인덱스 기능이 존재한다
데이터 백업, 복원 기능	없다.	있다.	-
원자성(atomicity)	불가능하다.	가능하다.	· 데이터 2개 이상을 all 또는 nothing으로 변경한다. · 데이터베이스의 트랜잭션 기능이다.
일관성(consistency)	없다	있다.	· 잘못된 상태의 데이터를 원천 봉쇄한다. · 데이터베이스의 constraints 기능이다.
고립성(isolation)	없다	있다.	· 경쟁 상태(data race)에서 자유로울 수 있게 하는 기능 이다. · 데이터베이스의 lock 기능이다.
지속성(duration)	없다.	있다.	· 장애 직전 상태로 복구할 수 있는지에 관한 것이다. · 데이터베이스의 로그 버퍼 기능이다

4.8 | 과제 대전 참가

- 10월 16일 수요일 수업은 과제 대전 참가로 대체
- 참가후 사진을 찍어서 email로 제출
 - 제목 [2019 게임서버프로그래밍 과제대전 참가 학번 이름]
 - 10월 17일까지 제출

4.8 | 서버 구동 환경

• 서버의 운영체제와 요구사항

서버를 구동하는 운영체제 : 리눅스와 윈도 서버가 골고루 쓰임

서버를 설치 할 때 요구사항 : 1. 인터넷 품질이 신뢰성이 있어야 한다. 인터넷이 중간에 끊어지거나 느려지지 않아야 한다.
 2. 서버가 쉽게 고장 나지 않아야 한다. 컴퓨터를 365일 24시간 계속 켜 놓는 상황에서 장기간 고장이 없어야 한다.
 3. 도난이나 침입 사고에서 안전해야 한다.
 → 이 점 때문에 규모가 큰 서버는 '데이터센터'라는 특별한 건물에 설치되는 것이 일반적이다.

• 클라우드 서버

클라우드 서버 : 리얼머신 위에 구동하는 가상 머신의 집합. 물리적 머신 한 대 안에서 여러 서버 운영체제가 동시에 작동함.



그림 4-23 온프림(물리적 자체) 서버와 클라우드 서버

4.8 | 서버 구동 환경

- 물리적 서버와 클라우드 서버

표 4-3 물리적 서버와 클라우드 서버 비교

구분	물리적 서버	클라우드 서버
특징	서버 컴퓨터 한 대에 운영체제 하나를 구동한다	서버 컴퓨터 한 대에 여러 운영체제를 가상 머신으로 구동한다.
서버 한 대당 유지 비용	낮다.	높다.
서버 증설 속도	낮다(하루).	높다(몇 분).
서버 철거 속도	낮다(경우에 따라 불가능).	높다(몇 분).
자동 스케일 아웃	불가능하다.	가능하다.
처리 속도의 균일성	높다.	낮다.
장애 처리	느리다.	빠르다.

4.8 | 서버 구동 환경

- 클라우드 서버가 제공하는 기능

IaaS(Infrastructure as a Service) : 가상 머신 자체를 제공하는 서비스. IaaS에서 생성한 가상 머신 안의 운영체제(시스템 레지스트리 등)는 마음대로 제어할 수 있다.

PaaS(Platform as a Service) : IaaS보다는 상위 계층에서 작동. 운영체제뿐만 아니라 운영체제 위에서 어떤 프레임워크 소프트웨어가 이미 구동되고 있으며, 이 프레임워크 위에 서버 코드나 데이터 파일을 업로드해야 한다.

SaaS(Software as a Service) : PaaS보다 상위에 있다. 코딩 자체가 불필요하며, 과금이 나 데이터 분석, 페이스북 로그인 연동 같은 특화된 기능들을 제공함.



IaaS

PaaS

SaaS

그림 4-24 계층에 따른 클라우드 서버 종류

4.9 | 서버 개발 지침

- 언어, 운영체제, 개발 도구 등 어떤 것을 선택하든 자신이 가장 잘 다루는 것을 쓰길 적극 권장함.
- 다양한 문제를 빠르게 해결하면 서버 개발에 사용한 언어, 운영체제, 도구 등이 자신에게 익숙해야 한다.
- 연구는 진보적으로 하고, 필드 적용은 보수적으로 할 것.

4.10 | 더 읽을 거리



- <http://goo.gl/EAbvUF>

5장 게임 네트워킹

- 1 | UML
- 2 | 게임 플레이 네트워킹
- 3 | 레이턴시 마스킹
- 4 | 넓은 월드, 많은 캐릭터 처리
- 5 | 실시간 전략 시뮬레이션 게임에서 네트워크 동기화
- 6 | 실제 레이턴시 줄이기
- 7 | 게임 플레이 이외의 네트워킹
- 8 | 해킹과 보안
- 9 | 요약

5.1 | UML

UML은 단순히 플로차트(어떤 실행의 순서, 즉 프로세스 정의만 다룬 것) 그 이상의 역할을 한다.

UML을 사용하면 데이터 정의와 프로세스 정의가 편해진다.

프로그램 구조 명세를 표현하는 대표적 수단.

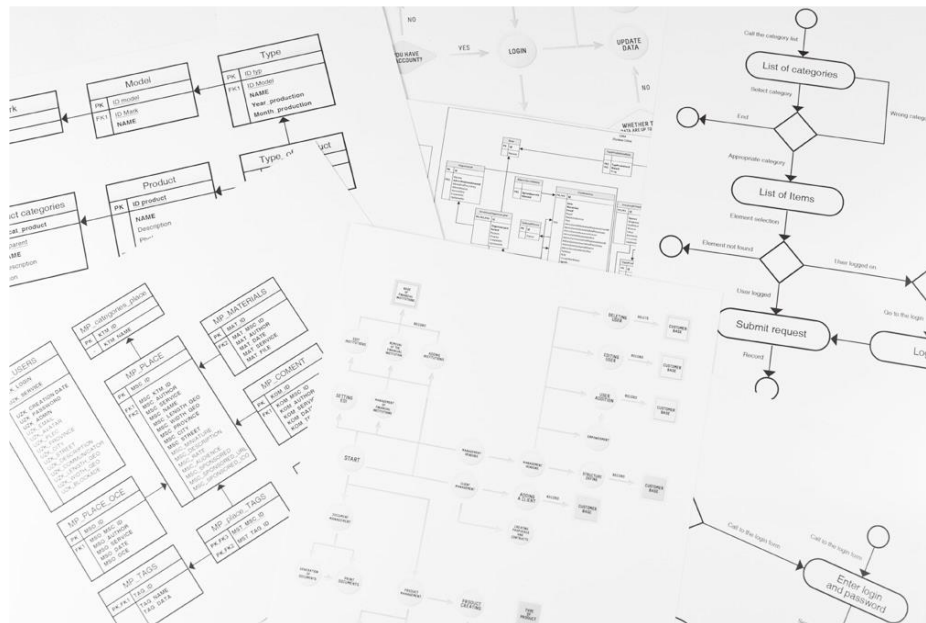


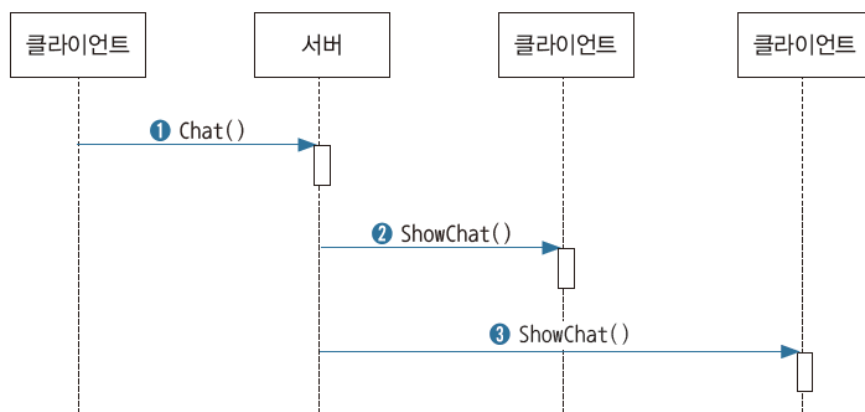
그림 5-1 UML

5.1 | UML

• 5.1.1 UML 시퀀스 다이어그램

UML 시퀀스 다이어그램에서는 객체(object)와 메시지를 사용한다.

시퀀스도의 주요 용도는 객체 간 메시징 흐름을 일목요연하게 표현하는 것.



1. 클라이언트는 서버에 채팅을 보낸다.
2. 서버는 두 클라이언트에 채팅을 보여 주라는 신호를 보낸다.

그림 5-2 시퀀스도 예

5.1 | UML

- 액티비티 다이어그램

시퀀스도에 플로차트를 함께 사용하면, 보다 자세하게 프로그램의 실행 방식을 그림으로 묘사할 수 있다.

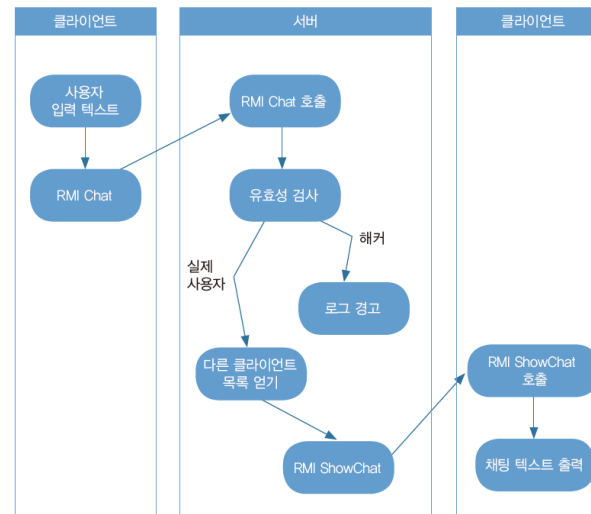


그림 5-3 액티비티 다이어그램

5.2 | 게임 플레이 네트워킹

5.2.1 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)

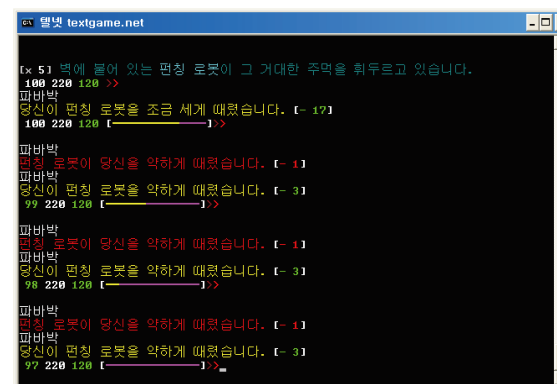
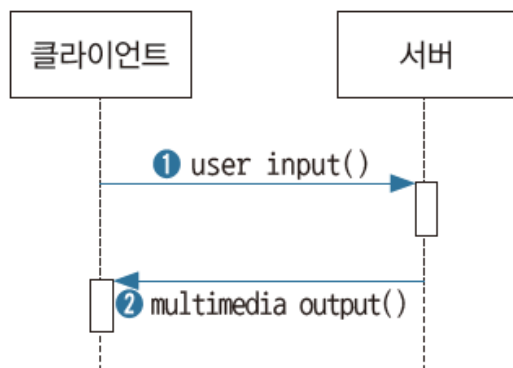


그림 5-4 모든 역할을 서버에서 하는 모델

그림 5-5 터미널은 텍스트 입출력 처리밖에 하지 못함

- 온라인 게임을 방해하는 레이턴시가 길어지는 요인

5.2 | 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.

5.2 | 게임 플레이 네트워킹

• 5.2.2 렌더링은 클라이언트에서 하기

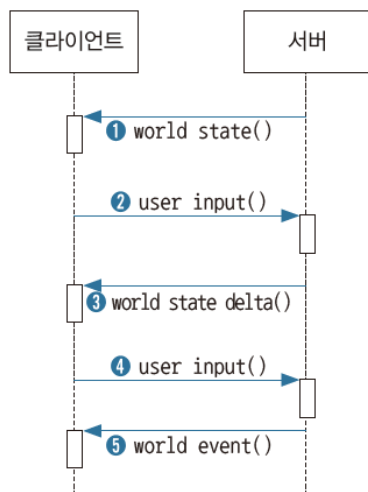


그림 5-6
렌더링은 클라이언트에서 담당하는 형태

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.

5.2 | 게임 플레이 네트워킹

- 클라이언트에서 서버로 보내는 플레이어의 취하는 행동 명령 메시지
 - 채팅 메시지를 입력.
 - 플레이어를 특정 방향으로 이동하라고 명령.
 - 플레이어가 이동을 멈추라고 명령.
 - 특정 아이템을 사용하라고 명령했다.
- 서버에서 클라이언트로 보내는 월드 상태 변화 메시지
 - 캐릭터가 등장 (데이터 추가).
 - 캐릭터가 특정 방향으로 이동 (데이터 변경).
 - 캐릭터가 웃는다(데이터 변경).
 - 캐릭터가 사라진다(데이터 소멸).
- 단발성 이벤트
 - 특정 좌표에서 수류탄이 터짐(단발성 이벤트).
- 단발성 이벤트를 지속성 이벤트로 만들 때
 - 특정 좌표에 수류탄이 생김.
 - 수류탄이 터짐.
 - 수류탄이 사라짐.
 - 파티클 1초 모습.
 - 파티클 2초 모습.
 - 파티클이 사라짐.

5.2 | 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.

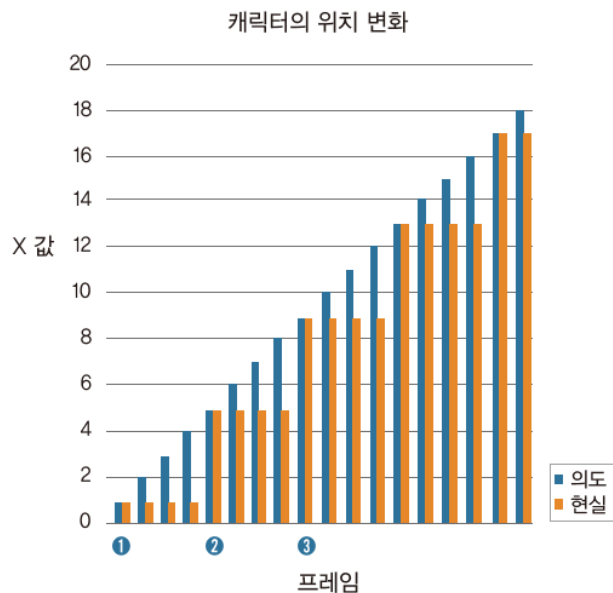


그림 5-7 시간이 흐르면서 서버에서 캐릭터 위치(의도)와 클라이언트에서 캐릭터 위치(현실)

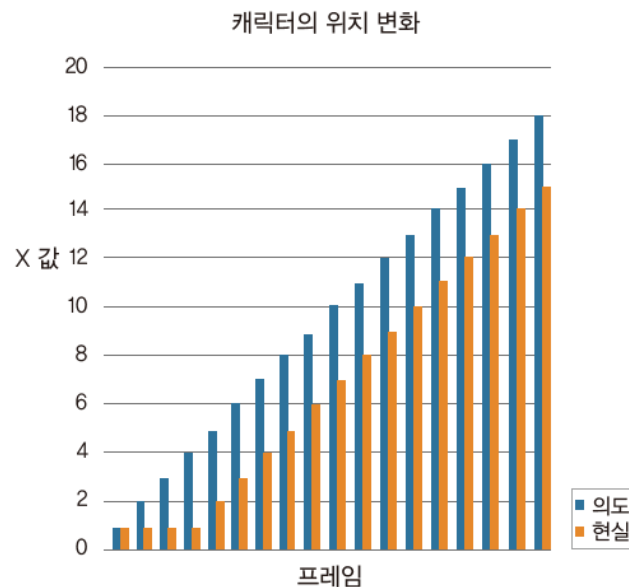


그림 5-8 서버에서 캐릭터 위치(의도)와 클라이언트에서 부드럽게 보여지는 위치(현실)

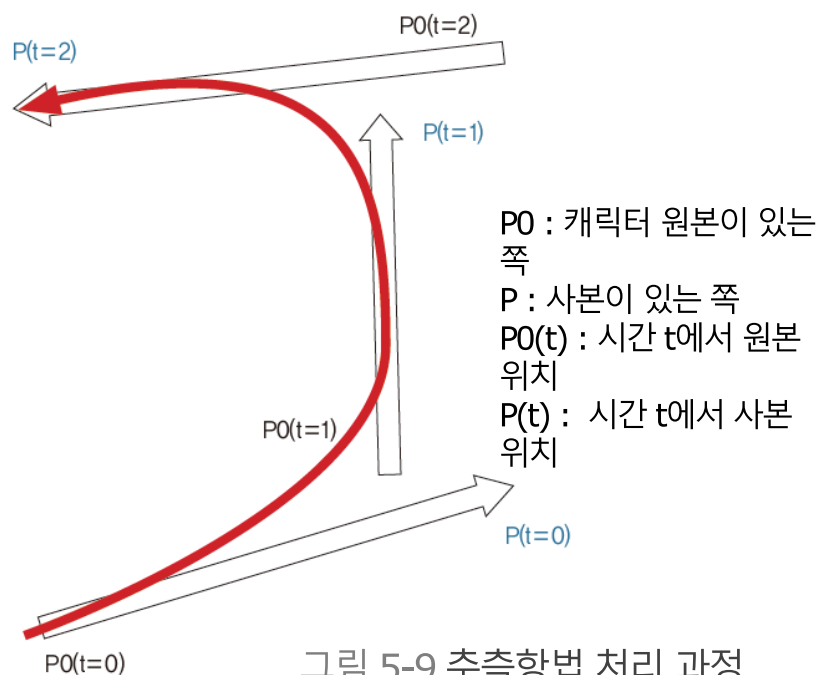
5.2 | 게임 플레이 네트워킹

• 5.2.3 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.



$t = 0$ 일 때 $P0(t = 0)$ 을 보낸다.

마찬가지로 $t = 1$ 일 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.

5.3 | 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 추측방법으로 플레이어 캐릭터 위치를 부드럽게 만든 후 업데이트한다.

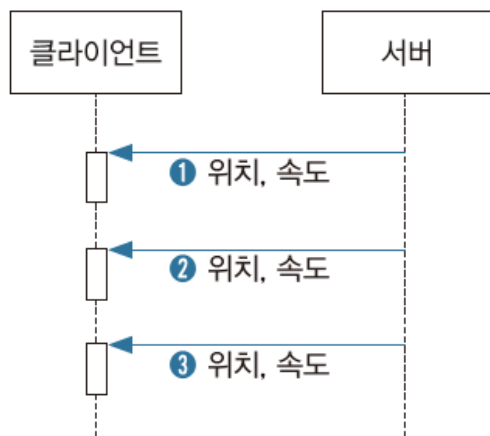


그림 5-10 여러분이 만든 게임이
이렇게 작동 중이라고 가정

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.

동 정보 메시지를 받으면, 앞서 일방적으로 위치를 무시하고 서버에서 메시지에 따라 캐릭터를

움

아

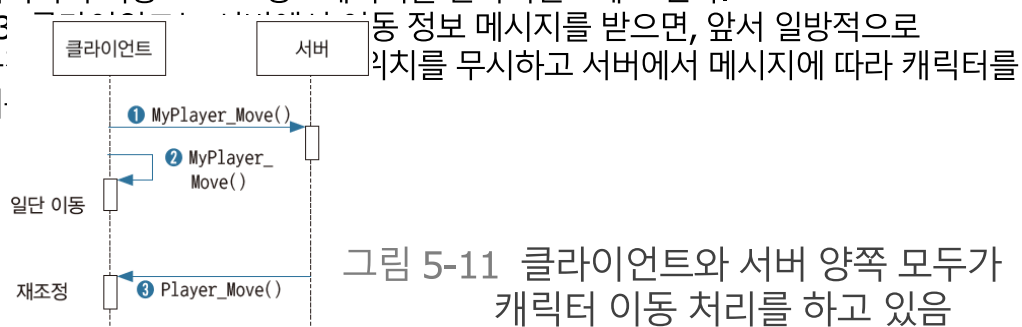


그림 5-11 클라이언트와 서버 양쪽 모두가
캐릭터 이동 처리를 하고 있음

5.3 | 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 클라이언트에 메시지로 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

5.4 | 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.

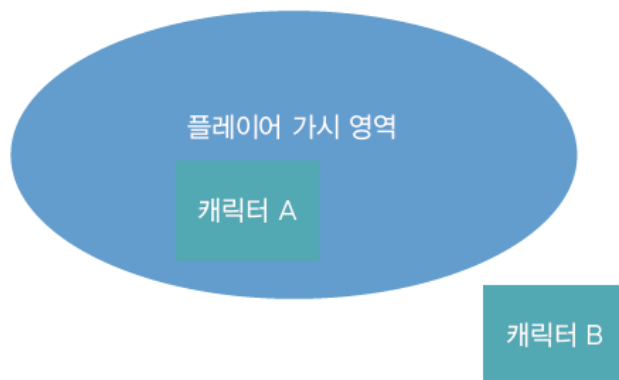


그림 5-15 플레이어의 가시 영역

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

5.5 | 실시간 전략 시뮬레이션 게임에서 네트워크 동기화

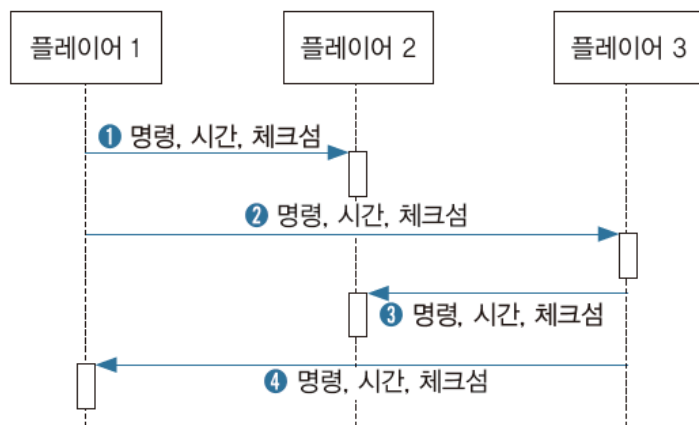
• 락스텝

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.



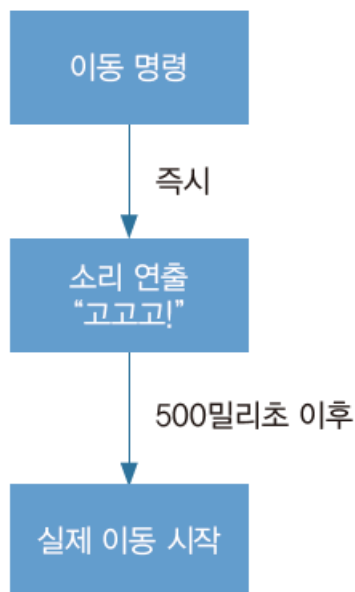
컴퓨터간 통신 레이턴시가 거의 없을 때야
완벽하게 작동
→ 입력명령을 보내되 '언제 실행해야 하는지에
대한
미래 시간'을 같이 보내야 한다.

미래시간 산정법
현재 시간 + 왕복 레이턴시(RTT) / 2 + 임의의
일정 값

그림 5-17 락스텝 동기화 방식에서 플레이어 간에 메시지 전송

5.5 | 실시간 전략 시뮬레이션 게임에서 네트워크 동기화

- 전략 시뮬레이션 게임에서 레이턴시 마스킹
 - 실제로 캐릭터는 입력 명령을 처리하지 못했더라도 명령에 반응하는 연출만 보여 준다. 일단 얼렁뚱땅 보여 주어 “즉시 반응하는구나!”를 느끼게 한다.
 - 이후에 미래 시간에 도달하면 실제로 캐릭터가 움직이게 한다.



락스텝 동기화 알고리즘의 한계

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

그림 5-18
락스텝 동기화 방식에서
레이턴시를
얼렁뚱땅 테크닉으로 감추기

5.6 | 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 네트워킹)을 같이 섞어 쓴다.

5.7 | 게임 플레이 이외의 네트워킹

- 로그인 과정에서 클라이언트와 서버가 대화하는 주요 절차
 - 로그온 요청 메시지를 서버로 전송한다.
 - 서버는 파일이나 데이터베이스에서 해당 유저의 ID와 비밀번호를 받아서 식별한다.
 - 식별 결과, 즉 로그인 처리 결과를 클라이언트에 통보한다.
 - 클라이언트가 이 통보를 받으면(예를 들어 로그인 성공이라는 통보를 받으면) 플레이어 정보를 데이터베이스에서 로딩해서 게임 서버 메모리에 보관한다. 이 과정은 있을 수도 있고 없을 수도 있다.

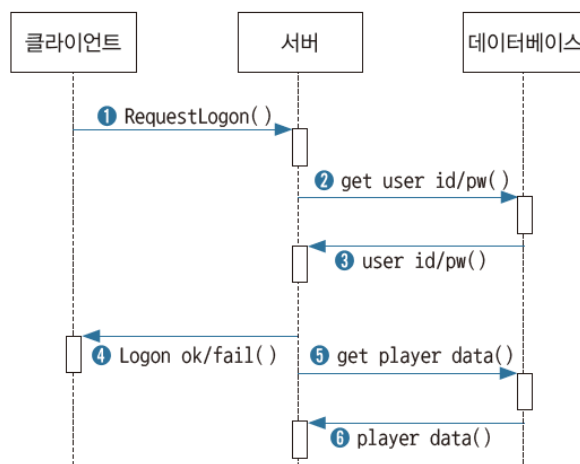
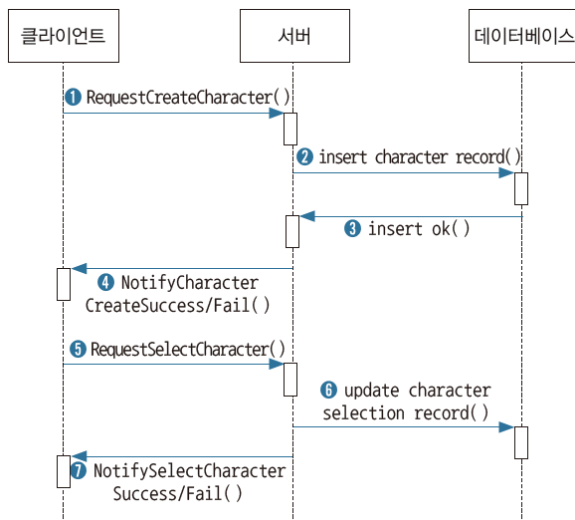


그림 5-19 로그인 과정을 시퀀스 다이어그램으로 표현

5.7 | 게임 플레이 이외의 네트워킹

캐릭터를 만드는 과정



- 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 000를 선택하겠다."라고 요청합니다.
 - 6 서버는 데이터베이스에 "캐릭터 000를 선택했다고 기록하자."라고 요청한다.
 - 7 서버는 클라이언트에 "캐릭터 000를 성공적으로 선택했다."라고 알려 준다.
- 그림 5-20 캐릭터를 만드는 과정을 시퀀스 다이어그램으로 표현

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);
    });
}
  
```

5.7 | 게임 플레이 이외의 네트워킹

- 매치메이킹_플레이어가 수동으로 방을 만들고, 다른 플레이어가 수동으로 방에 들어가는

방식 설명

1. 방을 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

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.

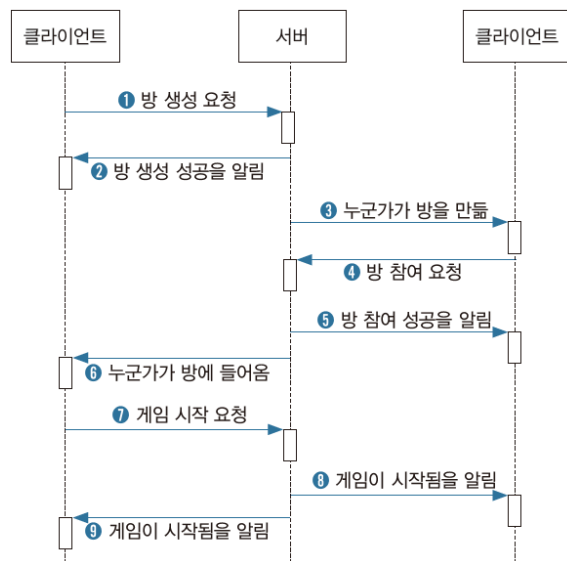


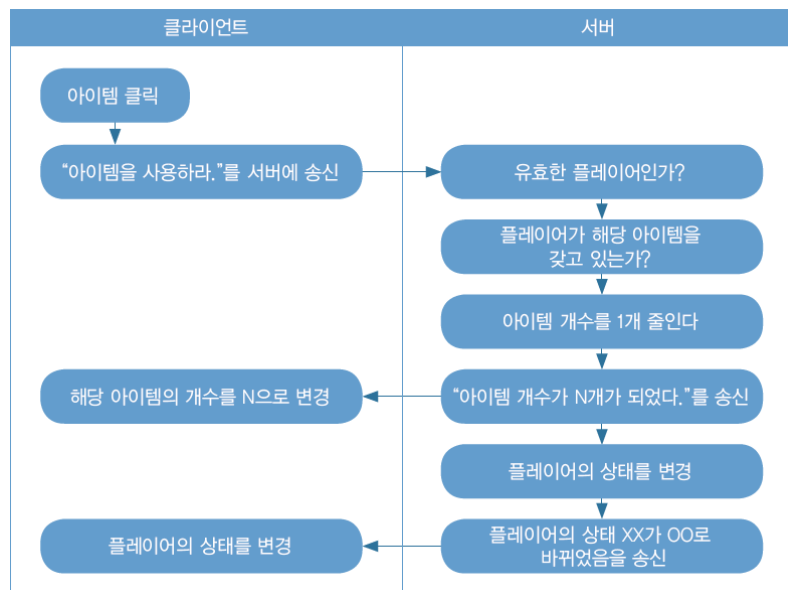
그림 5-21

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 너도 알고 있어야."라고 통보.
- 4 오른쪽 클라이언트의 플레이어는 이 방에 들어가고 싶다. 클라이언트는 서버에 "나는 이방에 들어가겠다."라고 요청.
- 5 서버는 오른쪽 클라이언트에 "방에 성공적으로 들어왔다."라고 응답한다. 동시에 방 정보와 방에 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 양쪽 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.

5.7 | 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.

그림 5-22
아이템 사용 처리를
액티비티 다이어그램으로 표현

5.8 | 해킹과 보안

- 게임 사용자들이 해킹을 하는 유형

1. 크래킹(crack): 게임 외 분야에서 흔히 이야기하는 해킹을 크래킹이라고 한다. 다른 사람의 ID와 비밀번호를 도용하거나 비밀 정보를 보는 것을 의미한다. 서버에 저장된 데이터를 훼손 하거나 훔치는 것도 포함된다.
2. 치트(cheat) 혹은 조작(exploit): 게임에서만 발견되는 형태의 해킹. 내 능력치를 비정상적으로 높이거나 다른 사람의 플레이를 망가뜨리는 것을 의미한다.

- 5.8.1 네트워크 해킹

1. 해커 Alice가 John과 게임 서버 사이에 있는 네트워크 기기에서 John과 게임 서버 간의 데이터 교환을 도청한다.
2. John이 게임 서버에 로그인한다. 이때 John의 클라이언트는 서버에 ID와 비밀번호를 보낸다.
3. Alice는 ID와 비밀번호를 도청하는 데 성공한다.

암호화

평문 + 암호 키 → 암호문

복호화

암호문 + 암호 키 → 평문

5.8 | 해킹과 보안

평문 + 암호 키 1 → 암호문

암호문 + 암호 키 2 → 평문

1. 서버에서는 암호 키 1을 John에게 보낸다.
2. John은 암호 키를 만들어 서버에 보낸다. 그러나 그냥 보내지 않고, 암호 키 1로 암호화해서 보낸다.
3. 서버는 이것을 받아서 암호 키 2로 복호화한다. 그리고 John이 보낸 암호 키를 얻는다.
4. John은 ID와 비밀번호를 암호 키로 암호화하여 서버에 보낸다.
5. 서버는 앞서 받았던 암호 키를 이용해서 이를 복호화한다. 그러면 ID와 비밀번호를 얻을 수 있다.

Alice 가

1. Alice는 암호화 알고리즘 1의 과정에서 암호 키 1을 훔친다.
2. Alice는 암호화 알고리즘 2의 과정에서 암호화된 암호 키를 훔친다. 그러나 암호 키 2가 없으므로 복호화를 할 수 없다.
3. Alice는 암호화 알고리즘 4의 과정에서 암호화된 ID와 비밀번호를 훔친다. 그러나 마찬가지로 암호 키가 없으므로 복호화를 할 수 없다.

암호 키: 대칭 키(symmetric key) 혹은 세션 키(session key)라고 한다.

암호 키 1: 공개 키(public key)라고 한다.

암호 키 2: 개인 키(private key)라고 한다.

5.8 | 해킹과 보안

- 5.8.2 클라이언트 컴퓨터 해킹

이메일이나 스마트폰 문자 메시지로 악성 코드가 들어 있는 첨부 파일이나 인터넷 주소를 보내서 악성 프로그램을 실행시키는 것. 멀웨어나 바이러스 등이 이에 해당

사람들이 사용하는 운영체제나 응용 프로그램의 결함을 이용해서 악성 프로그램을 전파하기도 함.

컴퓨터가 해킹을 당하지 않게 항상 최신 소프트웨어를 유지하고, 불필요한 보안 해제를 하지 말고, 알 수 없는 출처의 프로그램을 실행하지 않아야 한다.

- 5.8.3 서버 컴퓨터 해킹

클라이언트를 해킹할 때와 마찬가지로 서버 쪽 운영체제나 응용 프로그램 결함이나 보안 설정의 구멍을 이용하여 해킹한다.

서버 고유의 역할인 웹 서버나 데이터베이스 서버에서는 클라이언트와는 다른 종류의 해킹을 추가로 예방해야 한다. 질의 구문 인젝션(query injection) 등을 예로 들 수 있다.

게임 서버에는 일반적인 유저가 접속할 수 있는 리스닝 포트를 제외하고 모두 방화벽으로 막아 버려야 한다.

5.8 | 해킹과 보안

• 5.8.4 게임 치트

네트워크 도청 및 조작을 해서 해킹하는 경우

1. 어떤 게임에서는 플레이어 캐릭터가 공격을 할 때마다 “플레이어가 공격 행동을 한다.”라는 메시지를 서버에 보낸다.
2. 해커 Alice는 이 메시지를 갈무리하고 이 메시지를 마구잡이로 복제해서 서버에 보낸다.
3. 서버에서는 이 메시지를 다 처리한다. 그리고 Alice는 엄청난 공격 속도로 다른 플레이어들을 모두 제압한다.

게임 치트 또한 DDOS 공격과 유사하게 할 수 있다.

1. Alice는 정밀하게 조작하지 않아도 되는 플레이어 캐릭터를 가지고 있다.
2. Alice는 게임 서버나 다른 게임 클라이언트에 엄청난 양의 네트워크 데이터를 보낸다. 그러면 멀티플레이어 품질이 매우 크게 떨어진다.
3. 이 상태에서 Alice는 정밀한 컨트롤이 필요한 다른 플레이어들을 상대로 승부 우위를 차지한다.

게임 클라이언트를 해킹하여 치트를 하기도 한다. 클라이언트 내부 데이터를 조작하여 클라이언트에서 처리하는 승부 판정을 조작함.

이러한 종류의 해킹을 차단하려면 이러한 해킹을 진행 중인 프로그램이 있는지 감시해야 한다.

5.9 | 요약

게임 플레이 네트워킹 설계와 구현은 여러분의 온라인 게임 품질에 크게 영향을 준다.

게임 장르뿐만 아니라 게임 기획 내용에 영향을 많이 받는 부분이며, 레이턴시 자체를 없앨 수는 없기 때문에 레이턴시 마스킹 같은 센스와 기발함이 발휘되어야 하는 부분이다.

이 장에서 다루었던 여러 가지 기법을 토대로 여러분은 다양한 시도를 하면서 독창적인 기법을 추가로 만들어 보십시오.

5.9 | 요약

게임 플레이 네트워킹 설계와 구현은 여러분의 온라인 게임 품질에 크게 영향을 준다.

게임 장르뿐만 아니라 게임 기획 내용에 영향을 많이 받는 부분이며, 레이턴시 자체를 없앨 수는 없기 때문에 레이턴시 마스킹 같은 센스와 기발함이 발휘되어야 하는 부분이다.

이 장에서 다루었던 여러 가지 기법을 토대로 여러분은 다양한 시도를 하면서 독창적인 기법을 추가로 만들어 보십시오.