2017학년도 2학기
네트워크 프로그래밍

- 개인 프로젝트 최종보고서 -

과목	네트워크 프로그래밍
담당교수	김기천/진정하 교수님
학과	컴퓨터공학부
학번	201311269
이름	김제헌
제출일	2017-12-13-수

프로젝트 소개

1. 프로젝트 목표 및 결과

가. 최종 목표

○ **서버-클라이언트 1:N기반의 테트리스 게임을 만드는 것**

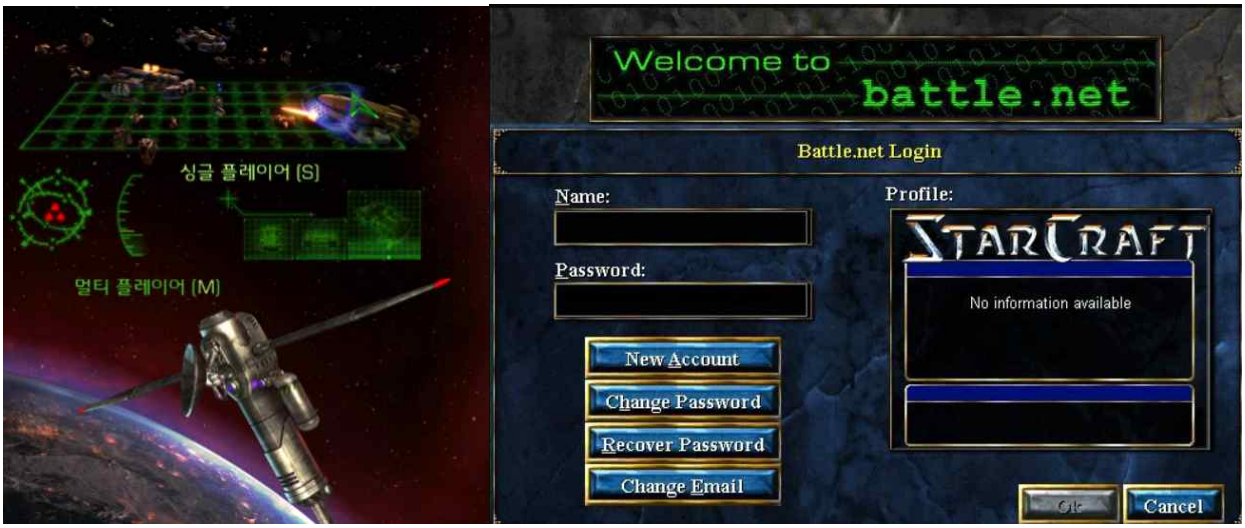
- 프로젝트 제안서에서도 언급했지만, 소켓프로그래밍도 처음 접해서 '어디까지 만들 수 있을까'의 고민과 '마땅한 소재[주제, 콘텐츠]'도 없어 고민하던 도중, 군대가기 전 2학년 2학기 '컴응2'의 마지막 과제인 '테트리스 만들기'가 생각났고, '그 당시 만든 코드를 이용해보면 어떨까?' 했습니다.
- 그 당시 주먹구구식으로 코딩을 했고 1인용이며, 콘솔환경에서 돌아가고, 에러도 있지만 그냥 제출했었는데 이를 에러도 잡고, 객체지향식으로 바꾸고, GUI도 적용하고, 소켓 프로그램을 이용한 1:N 대전[다인용]으로 만들어보려고 계획했습니다.
- 그래도 뼈대 알고리즘인 테트리스 코드가 있다 보니 아무래도 테트리스 자체보다는, 이에 소켓 프로그래밍을 적용해서 어떻게 하면 실시간 문제 등을 처리할 것인가 등을 중점으로 다뤄볼 계획이었습니다.

○ **프로그램의 기능 요구사항 [그 당시 제안서에 작성했던 내용을 그대로 옮겨왔습니다]**

- **싱글 모드와 멀티 모드**
  - 간단하게 혼자서 테트리스를 즐길 수 있는 싱글 모드와 여럿이서 테트리스를 즐길 수 있는 멀티 모드로 나눌 예정이었습니다.
- **서버로의 로그인 및 회원가입 기능**
  - 아이디와 비밀번호를 입력해서 멀티 모드인 서버로 접속하는 로그인 기능을 간단하게 추가할 예정이었습니다.
  - 회원가입 기능도 아주 간단하게 구현할 예정이었습니다.
- **전체 채팅 및 방 생성 및 방 목록 보기 기능**
  - 멀티모드인 서버에 접속하면, 전체적으로 채팅을 할 수도 있고, 새로운 방을 생성하던지, 생성되어 있는 방 목록을 볼 수 있는 기능을 구현할 예정이었습니다.
- **방 목록 중에서의 방 입장 기능**
  - 방 목록을 볼 수 있는 화면으로 넘어오면, 현재 생성되어 있는 방 목록이 보이며, 이에 입장할 수 있는 기능을 구현할 예정이었습니다.
- **각 방에서의 채팅 및 방장의 강퇴 기능, 레디 및 게임 시작 기능**
  - 각 방에서도 채팅이 가능하고, 방장에게 있어 입장 가능한 자리를 열고 닫는다던지 등의 강퇴 기능, 각 참가자들의 레디 기능, 모두 레디가 되었을 때 테트리스 게임을 시작하는 기능을 구현할 예정이었습니다.
- **어떤 방이 시작된 이후 테트리스 게임 시작**
  - 해당 방의 참여자들끼리 개인전 테트리스를 진행하며, 게임을 하는 도중에 채팅도 할 수 있도록 구현할 예정이었습니다.
  - 게임이 끝난 후에는 다시 전체 방 목록을 볼 수 있는 화면으로 돌아가게 할 예정이었습니다.

○ 모티브로 삼은 것들

- 기본적으로 '스타크래프트1' + '한게임 테트리스'를 모티브로 삼았습니다.
- 스타크래프트1
  - 스타크래프트1에서도 싱글모드와 배틀넷 접속 등의 멀티모드가 있고, 배틀넷 접속 [로그인 및 회원가입] 이후에도, 전체 채팅방이 있으며[여기서는 채널개념이지만], 방을 생성하고, 입장하고, 채팅하고, 강퇴하는 등으로 게임 흐름에 있어 전체적인 뼈대는 스타크래프트1을 따라갈 생각이었습니다.



[싱글 모드 및 멀티모드 | 로그인 및 회원가입 기능]

[접속 후 전체채팅 가능, 방 생성 및 방 목록보기 | 방 목록 중 방 입장하기 |
해당 방에서 채팅, 강퇴, 준비 및 게임 시작]

- 한게임 테트리스
  - 이건 게임 자체적인 GUI로써, 게임의 배치나 색상 방식 등을 이런 모습으로 따라 해볼 예정입니다.
  - Hold 명령이나, Next 블록이 어떤 건지에 대해서도 보여주게 할까는 생각 중이었습니다.
  - 음악 변경이나 아이템 사용 등은 일이 커지는 것 같아서 아직까지는 보류 중이었습니다.

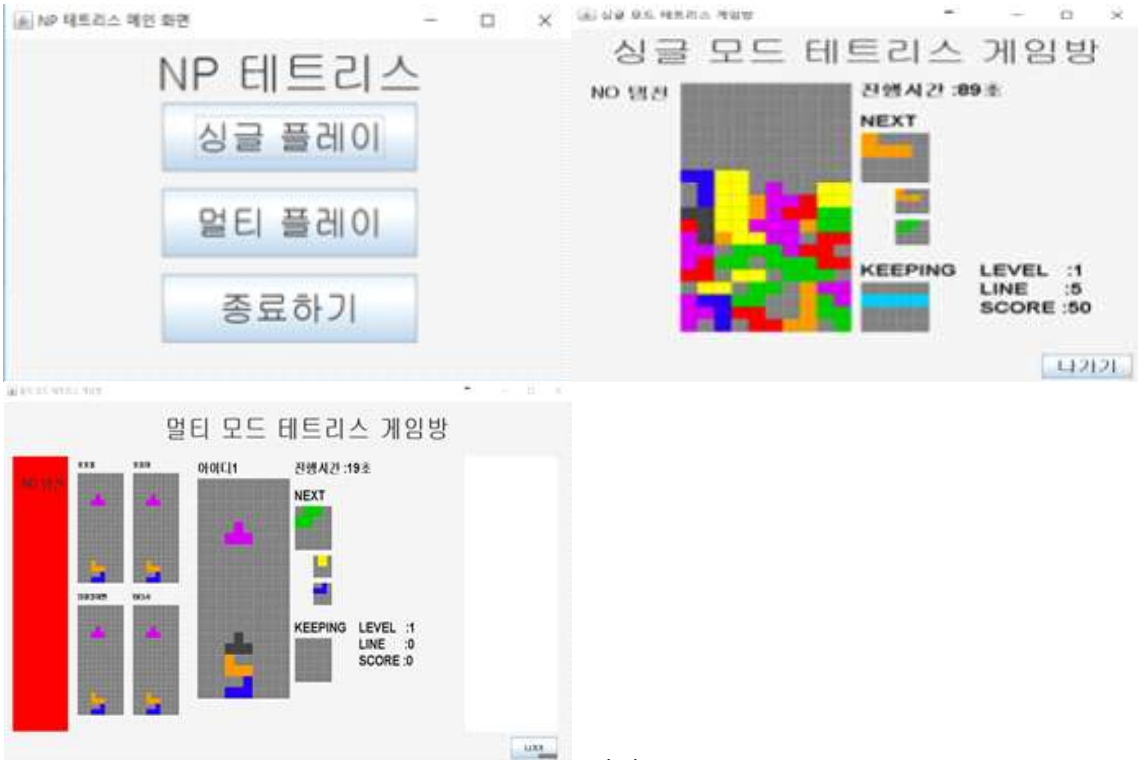
[한게임 테트리스 시작 시, 화면]

나. 프로젝트 결과

○ **프로그램의 기능 비교 [앞의 제안서에서 제시했던 기능들과 하나하나씩 비교해보겠습니다.]**

- **싱글 모드와 멀티 모드**

  • 맨 처음의 메인화면에서 간단하게 혼자서 테트리스를 즐길 수 있는 싱글 모드, 추후 서버 접속 후 여럿이서 테트리스를 즐길 수 있는 멀티 모드로 나뉘어서 개발했습니다.

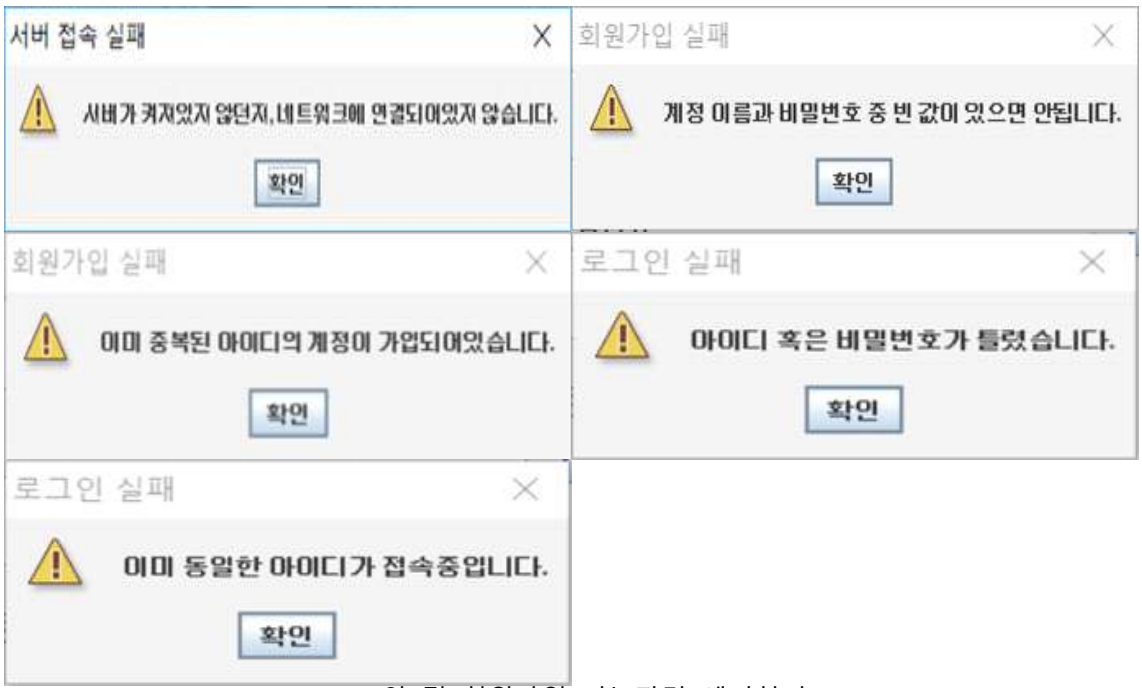


[싱글 모드 및 멀티모드]

- **서버로의 로그인 및 회원가입 기능**
  - 메인화면에서 '멀티 플레이'버튼을 누르면 '서버 로그인 화면'으로 넘어오며, 여기에서 아이디와 비밀번호를 입력해서 멀티 모드인 서버로 접속하는 로그인 기능을 간단하게 구현했습니다.
  - 회원가입 기능도 아주 간단하게 구현했습니다.

[로그인 및 회원가입 기능]

  - 이 부분에서 구현한 예외처리로는,
    - √ 클라이언트가 메인화면에서 '멀티 플레이'버튼을 눌렀을 당시, 서버가 꺼져있어서 접속에 실패할 경우, '서버가 켜져있지 않던지, 네트워크에 연결되어있지 않습니다.'라는 창이 뜨게 됩니다.
    - √ 회원가입 시 계정이름 또는 비밀번호 둘 중 하나라도 빈값이 있으면, '계정 이름과 비밀번호 중 빈 값이 있으면 안됩니다.'라는 창이 뜨게 됩니다.
    - √ 회원가입 시 해당 아이디가 이미 접속돼있는 경우, '이미 중복된 아이디의 계정이 가입되어 있습니다.'라는 창이 뜨게 됩니다.
    - √ 로그인 시 아이디와 비밀번호 값이 해당 계정이 서버에 없을 경우, '아이디 혹은 비밀번호가 틀렸습니다.'라는 창이 뜨게 됩니다.
    - √ 로그인 시 해당 계정이 이미 접속중일 경우, '이미 동일한 아이디가 접속중입니다.'라는 창이 뜨게 됩니다.

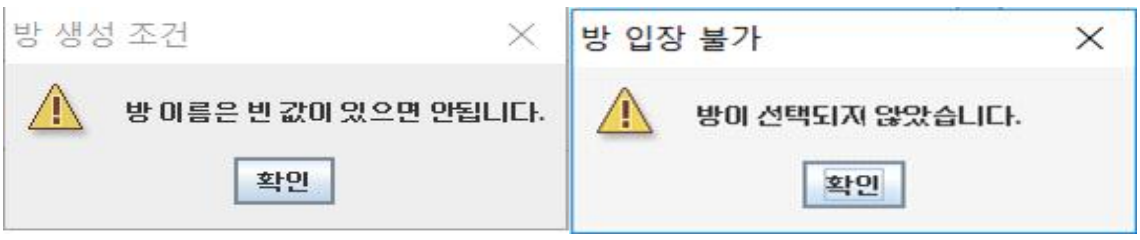


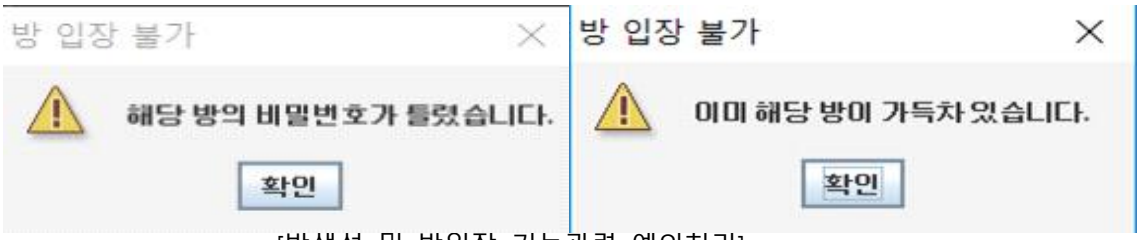
[로그인 및 회원가입 기능관련 예외처리]

- **전체 채팅 및 방 생성 및 방 목록 보기 기능**
  - 멀티모드인 서버에 접속하면, 서버 대기실 화면으로 가지며, 여기서 전체적으로 채팅을 할 수도 있고, 새로운 방을 생성하던지, 생성되어 있는 방 목록을 볼 수 있는 기능을 구현하였습니다.
  - 거기다가 이건 제안서에 없던 내용이었지만, 게임방을 거치지 않고, 일정 수가 되면 바로 게임을 시작할 수 있는 '빠른대전'이란 기능도 추가하여 개발하였습니다.
  - 방 목록에서 주기적으로 방목록을 최신화를 알아서 해주기는 힘들어서 '방 목록 갱신'이라는 버튼을 추가하였고, 이 버튼을 누를 때, 최신화된 게임방의 목록이 뜨도록 구현하였습니다.

[접속 후 전체채팅 가능, 방 생성 및 방 목록보기 | 방 목록 중 방 입장하기 | + 빠른대전 기능]

  - 이 부분에서 구현한 예외처리로는,
    √ 방생성 시 방 이름에 빈값이 있으면 '방 이름은 빈 값이 있으면 안됩니다.'라는 창이 뜨게 됩니다. 방 비밀번호 값이 빈값이면 기본 비밀번호[defaultPwd]값이 전송됩니다.
    √ 방입장 시 방 목록 중에서 선택을 하지 않고 바로 '입장'버튼을 클릭하면, '방이 선택되지 않았습니다.'라는 창이 뜨게 됩니다.
    √ 방입장 시 방에 비밀번호가 있었고 이게 틀리게 될 경우, '해당 방의 비밀번호가 틀렸습니다.'라는 창이 뜨게 됩니다.
    √ 방입장 시 방의 인원수가 모두 차있었다면, '이미 해당 방이 가득 차 있습니다.'라는 창이 뜨게 됩니다.





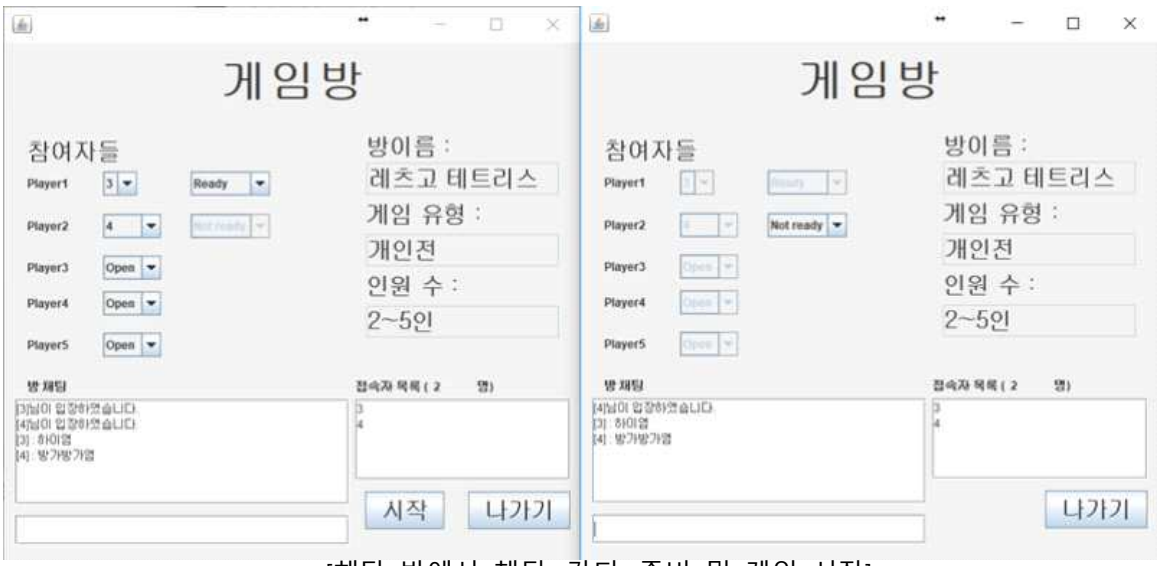
[방생성 및 방입장 기능관련 예외처리]

- **각 방에서의 채팅 및 방장의 강퇴 기능, 레디 및 게임 시작 기능**

• 각 방에서도 채팅이 가능하고, 방장에게 있어 입장 가능한 자리를 열고 닫는다던지 등의 강퇴 기능, 각 참가자들의 레디 기능, 모두 레디가 되었을 때 테트리스 게임을 시작하는 기능을 구현하였습니다.

• 오직 방장만이 입장이 가능한 자리를 열고 닫을 수가 있고 플레이어들을 강퇴할 수가 있으며 [해당 플레이어자리를 Close로 닫으면 강퇴가 됩니다], 게임을 시작할 권한이 주어집니다.

• 그 외의 다른 플레이어들은 채팅 외에 자신의 레디를 준비시키는 것만 할 수가 있습니다.



[해당 방에서 채팅, 강퇴, 준비 및 게임 시작]

• 이 부분에서 구현한 예외처리로는,

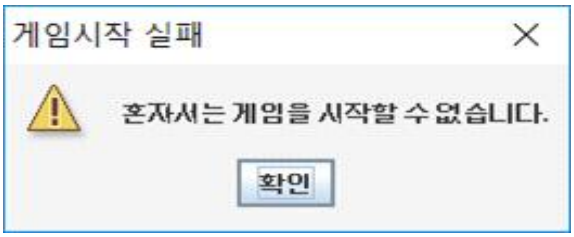
√ 게임방에서 방장이 어떤 클라이언트의 자리를 Close로 바꾸면, 해당 클라이언트 입장에서는 '방장에 의해 강퇴되었습니다.'란 창이 뜨며 대기실 화면으로 나가게 됩니다.

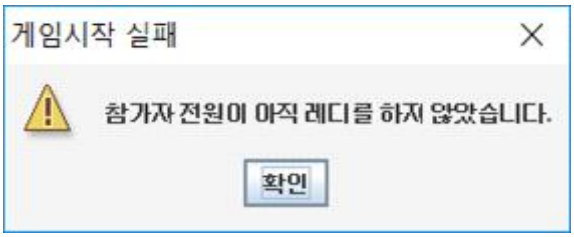
√ 게임방에서 방장이 방을 나갈시, 방이 폭발하게 되며 방장을 제외한, 게임방에 있는 나머지 사람들에게 '방장이 방을 나갔습니다.'라는 창이 뜨며 대기실 화면으로 나가게 됩니다.

√ 게임방에서 방장이 레디를 눌렀을 때 방장 혼자였다면, '혼자서는 게임을 시작할 수 없습니다.'라는 창이 뜨게 됩니다.

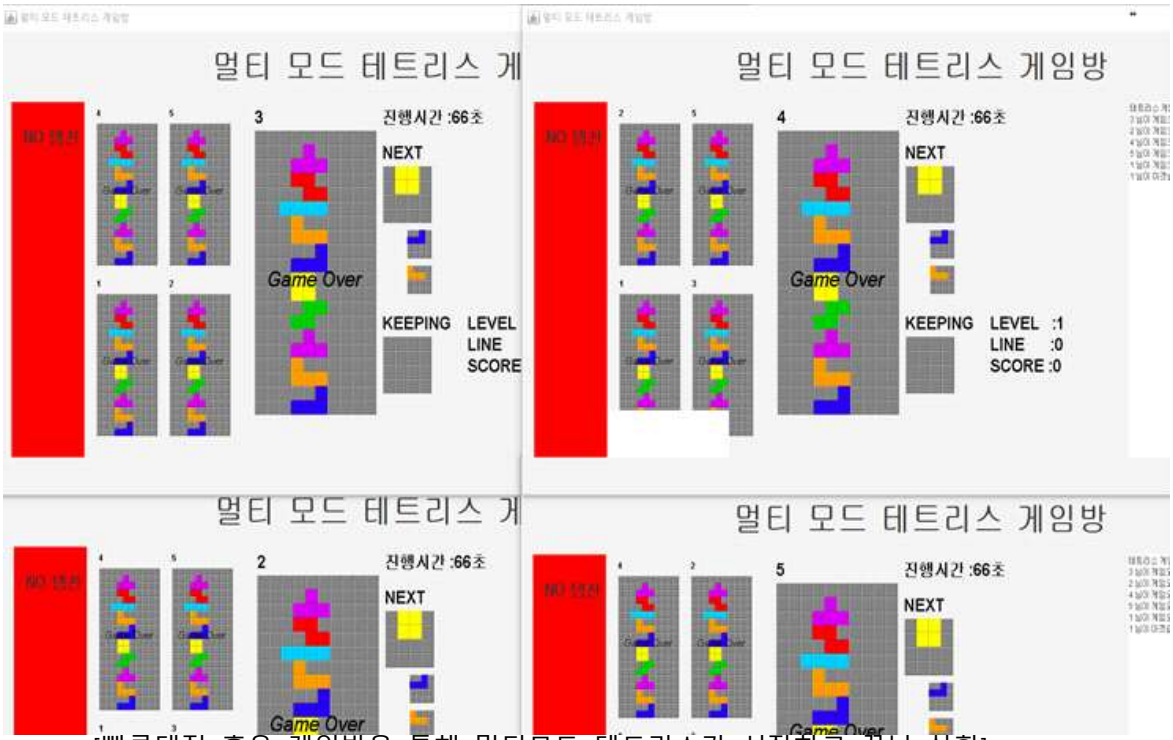
√ 게임방에서 방장이 레디를 눌렀을 때 방장 혼자가 아니더라도 모두가 레디를 하지 않았다면, '참가자 전원이 아직 레디를 하지 않았습니다.'라는 창이 뜨게 됩니다.

[게임방 기능관련 예외처리]





- **어떤 방이 시작된 이후 테트리스 게임 시작**
  - 해당 방의 참여자들끼리 개인전 테트리스를 진행할 수 있도록 구현하였습니다.
  - 본래 게임을 하는 도중에 채팅도 할 수 있도록 구현할 예정이었지만… 테트리스 패널과 제어 관련해서 GUI쪽으로 채팅을 하면 테트리스 패널이 제어를 다시 받지 못하는 문제가 있어서 채팅기능을 빼버렸습니다.
  - 게임이 끝난 후에는 다시 전체 방 목록을 볼 수 있는 화면으로 돌아갈 수 있도록 구현하였습니다.



[빠른대전 혹은 게임방을 통해 멀티모드 테트리스가 시작하고 끝난 상황]

  - 이 부분에서 구현한 예외처리로는,
    √ 테트리스방에서 최종적으로 오래 살아남은 플레이어가 게임이 끝날 때, 방에 남아 있는 모든 사람들에게 'ㅇㅇ님이 이겼습니다.'라는 창이 뜨게 됩니다.
    √ 테트리스방에서 어떤 클라이언트가 방을 나갈 때, 이미 게임오버인 사람이면 그냥 방을 나가는걸로 표시되고, 아직 게임중인 사람이 나갈 경우, 해당 플레이어를 게임오버시키며, 다른 사람들에게 해당 플레이어가 게임을 포기하고 나갔다고 알려주게 됩니다.

2. 프로젝트 수행 내역

가. 프로젝트 진행 일정

○ 제안서 작성 당시의 진행일정은 다음과 같았습니다.

(개발기간 : 2017. 8. 28. ~ 2017. 11. 30.)

주차 / 세부내용	1	2	3	4	5	6	7	8	9	10	11	12	13	14
프로젝트 주제 정하기 및 제안서 쓰기														
중간고사														
1단계 및 2단계 진행하기														
3단계 및 4단계 진행하기														
5단계 진행하기 및 테스팅 등 마무리하기														
최종 발표준비														

○ 프로젝트를 진행하는데 있어 해결해야 할 문제 및 과정들 [제안서 때 작성했던 계획들]

- 1단계. 기존 C++ 테트리스 코드 수정
  - 기존 코드를 재분석하고, 에러를 잡아내고, 객체 지향적인 설계를 적용해서 코드를 수정해야 합니다.
- 2단계. 싱글 모드 기반으로 테트리스 GUI 프로그래밍
  - 기존의 콘솔 모드를 떠나서 QT툴도 익히고, C++ GUI프로그래밍을 익힌 이후에 싱글 모드로 할 수 있는 테트리스 GUI 프로그래밍을 해야 합니다.
- 3단계. 1:N기반의 로그인, 회원가입, 채팅 등등의 채팅서버 구현
  - 우선은 순수하게 '채팅'만 할 수 있는 1:N 채팅서버를 만들어볼 생각입니다.
  - 방을 생성하고, 강퇴하고, 방에서 들어가고, 나오고, 게임을 시작하고 등의 과정이 아직 감이 잘 안 오기 때문에 이와 관련한 큰 틀을 만들어보려고 합니다.
- 4단계. 특정 방 기준으로 1:N 테트리스 게임 구현
  - 2단계까지 만든 [완성된?] 테트리스 게임을 1:N 방식으로 바꿔서 구현해볼 생각입니다.
  - 어떻게 하면 실시간적으로 동시성을 처리할 것인가가 핵심인 것 같습니다.
- 5단계. 3단계와 4단계를 합친다.
  - 마치 네트 FT 기능 구현 과제에서 에코 채팅 서버를 구현하고, FT 기능 서버를 구현한 후에 이 둘을 합친 것처럼 그런 느낌으로 합쳐볼 생각입니다.
- 프로젝트 진행 순서
  - 1단계 및 2단계를 먼저 진행해야 할 것 같고, 그 이후에 3~5단계를 진행하는데, 3단계 및 4단계는 크게 관련은 없지만, 네트워크 프로그래밍이 아직 익숙하지 않기 때문에 3단계를 완성시켜서 감을 잡은 후에 4단계를 진행하는 것이 좋지 않을까 싶습니다. 4단계를 진행할 때는 5단계의 합치는 것까지 염두에 둬서 진행을 해야 할 것 같습니다.

○ 하지만... 현실은 그렇게 되지 않음

- 그렇게 계획을 짜고 중간고사 끝나자마자 바로 기존 C++ 테트리스 코드를 쭉 수정하며 1단계를 진행했고, QT툴도 공부하고 하면서 2단계를 진행하던 도중 GUI의 근본적인 한계[GUI에는 절대좌표의 개념이 없다는 문제]에 부딪히게 되었고, 언어를 자바로 바꾸고 거기에 투자하는 시간이 길었습니다. 이후 버그가 있음에도 3, 4단계를 구현하는데 끊임없는 문제가 발생하면서 그 이후부터 계속 시간은 쏟아 부었지만 진행일정을 맞추지 못했습니다.

나. 프로젝트 개발 환경

◦ Development Environment

- A. 운영체제 : Windows 10 [64비트]
  - 지금 사용하고 있는 노트북 사양입니다.
- B. CPU : Intel(R) Core(TM) i7-6700HQ
  - 지금 사용하고 있는 노트북 사양입니다.
- C. 개발 언어 : 자바
  - 자바인 이유 : 맨 처음 제안서에 개발 언어를 C++로 하겠다는 이유로 별의별 이유를 대며 C++로 개발을 하겠다고 했었지만, GUI로 테트리스를 표현하는데 근본적인 한계[GUI에는 절대좌표의 개념이 없다는 문제]에 부딪히게 되었고, 언어를 자바로 바꾸게 되었습니다. 좀 더 따지면 자바의 애플릿 기능을 이용하기 위해서였습니다.
- D. IDE : 이클립스 및 넷빈즈
  - Visual Studio로 C++ 테트리스 코드를 수정했고, QT를 익히던 도중 역시 비슷한 이유로 자바로 갈아타게 되면서 이클립스로 주로 개발을 하였고, 넷빈즈는 GUI보조 툴로 사용하였습니다.
- E. UML 툴 : StarUML
  - 추후에 Class Diagram, Sequence Diagram 등의 설계과정을 표현하는데 있어 이용하였습니다.

다. 프로젝트 수행 방법

◦ 그저 부딪힘

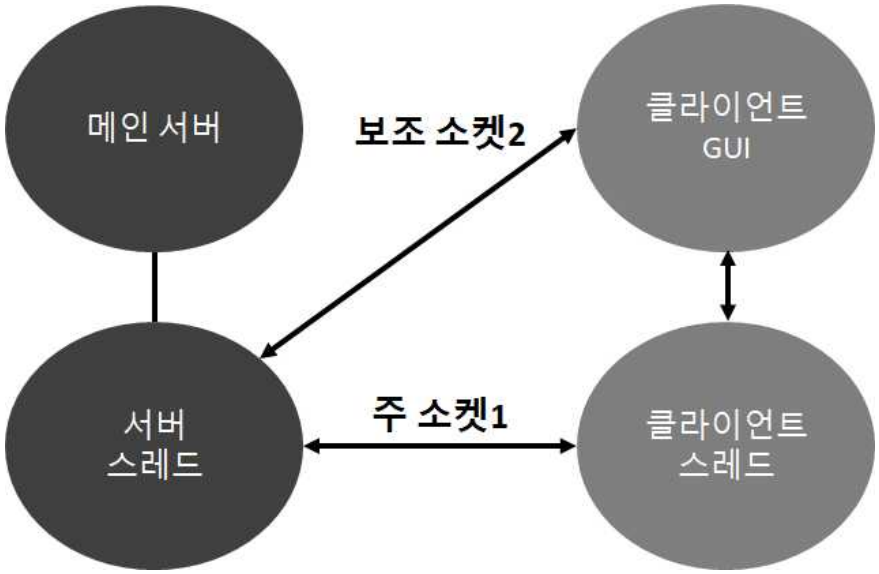
- 수행 방법이라고 거창할 것 까지는 없고, 주구장창 시간을 쏟아 부으며 계속 부딪혔습니다.
  - 막히면 검색하고, 버그뜨면 계속 디버깅하고 검색하고, 치명적인 문제점을 발견해도 어떻게든 해결해 나갔습니다.
  - 하다가 안되자 클래스다이어그램, 시퀀스 다이어그램도 그려보고, 서버쪽 및 클라이언트 쪽의 프로토콜을 정리하는데 힘을 썼습니다.
  - 프로그램이 점점 커지자 조그만 곳에서 돌아가는지 확인하고 갖다붙이는 작업도 자주 했습니다.

3. 알고리즘 및 순서도

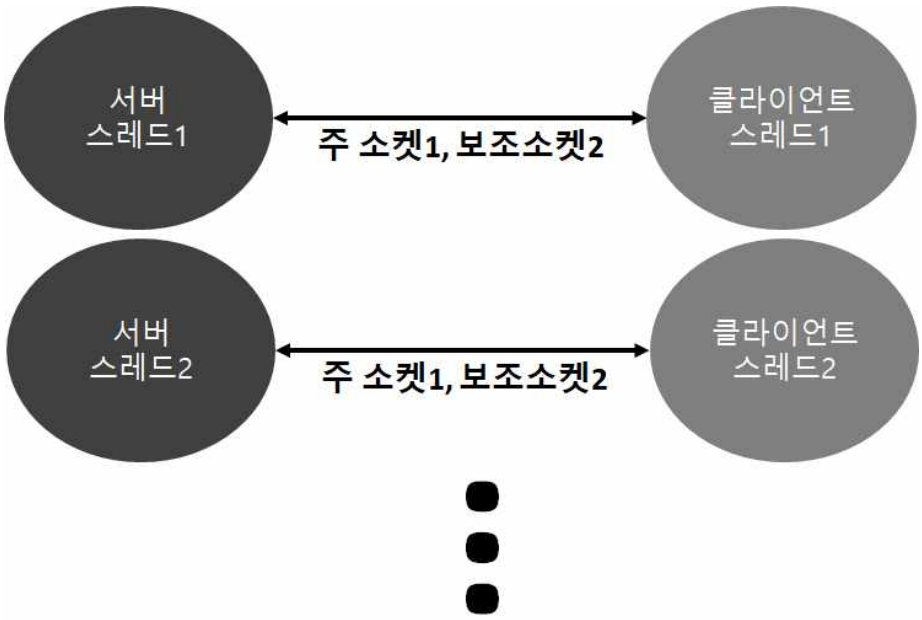
가. 핵심 알고리즘

○ 서버 클라이언트 구조

- 메인서버는 켜지는 순간부터 계속 돌면서 클라이언트의 접속을 기다리다가, 클라이언트가 접속을 하면 서버 스레드를 생성하고, 서버 스레드와 클라이언트 스레드가 1:1로 연결되는 구조입니다.
  - 이 외에 맨 처음에는 항상 클라이언트 GUI쪽에서 서버 스레드 쪽으로 메시지를 보내는 구조인데, 서버 스레드에서 클라이언트 스레드 말고, 다시 클라이언트 GUI쪽으로 신호를 줘야하는 경우도 필요해서 소켓을 2개 연결해서 처리하도록 했습니다.

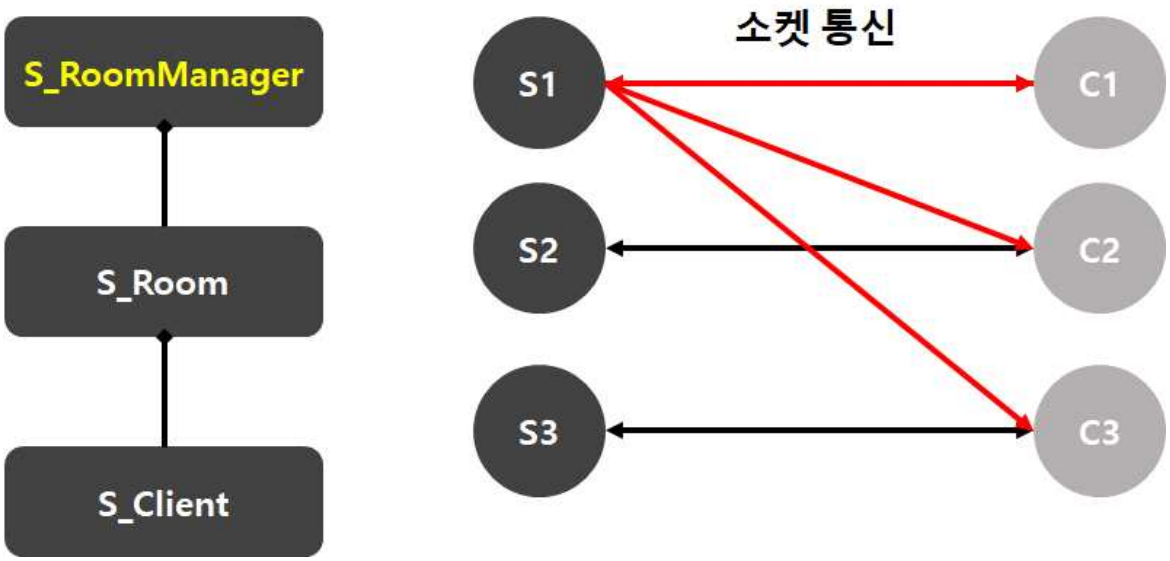


  - 서버 스레드와 클라이언트 스레드는 주소켓과 보조소켓을 통해 1:1로 연결되게 되며, 계속 그러한 구조로 진행됩니다.



○ S_RoomManager

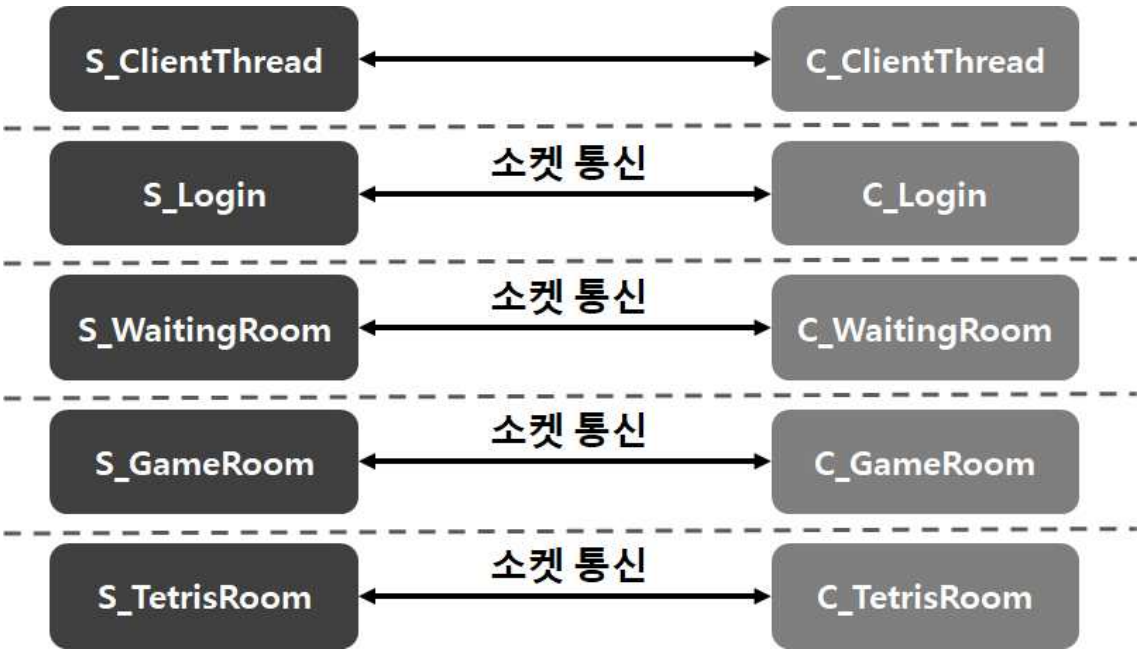
- 기본적으로 서버의 역할은 브로드케스트로 돌아가다 보니 '핵심 알고리즘'이라고 할 만큼 거창한 건 없지만 구조적으로 볼 때 S_RoomManager의 역할이 아주 중요하다고 생각합니다.
- S_RoomManager는 S_Room들을 가지고 있으며, S_Room은 S_Client들을 가지고 있습니다.
  - S_RoomManager는 서버 내에 생성되어있는 방들[대기실 포함]을 모두 관리합니다.
  - S_Room은 해당 방에 입장해 있는 클라이언트들을 모두 관리합니다.
  - S_Client는 해당 클라이언트 스레드와 연결된 서버 스레드입니다.
- 클라이언트 스레드중 하나인 C1에서 자신과 연결된 서버 스레드인 S1으로 메시지를 보내면, S1은 이를 받고, S_RoomManager를 이용해서 적절하게 브로드케스트를 이용하면서 다른 클라이언트 스레드들에게 메시지를 보내게 됩니다.



[S_RoomManager를 이용한, 브로드케스드 기능]

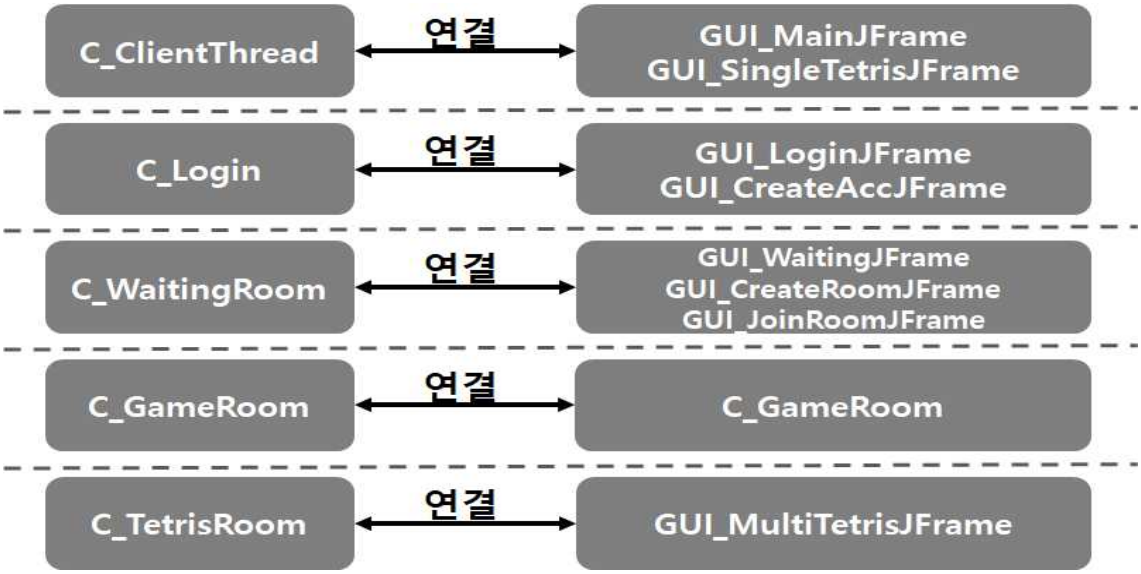
○ 기능에 따른 계층화

- 그 외에 서버의 제어를 함에 있어서, 클라이언트의 기능이 서버 접속, 로그인(회원가입), 대기실, 게임방, 테트리스게임방으로 나뉘어 있는 만큼 서버의 기능도 같게 나눠서 통신 수준을 맞추었습니다.



[기능에 따른 계층화 (서버 - 클라이언트 매칭)]

- 클라이언트 내에서도 View와 Function을 매칭시키는데 있어서도, 계층화시켜서 필요한 기능을 갖다 쓰도록 구현하였습니다.



[기능에 따른 계층화 (클라이언트 내 기능과 GUI 매칭)]

○ 프로토콜

- 구조 외에도 클라이언트와 서버 사이의 제어를 담당하는 프로토콜이 핵심이라고도 볼 수 있는데, 프로토콜은 클라이언트 GUI에서 서버스레드로 메시지를 전송하는 것과, 서버스레드에서 메시지를 받고 적절하게 전송하는 것, 다시 클라이언트 스레드 혹은 GUI에서 이를 받고 적절하게 대처하는 것으로 이루어집니다.
- 클라이언트(GUI) → 서버 스레드 프로토콜 목록

C_ClientThread[로그인전 제어] : 로그인, 회원가입, (강제)접속종료	
로그인	[LOGIN]/id/pwd
회원가입	[JOIN]/id/pwd
(강제)접속종료	[QUIT]
C_WaitingRoom[대기실 제어] : 이름, 방[대기실 등] 입장, 방 생성, 방목록 요청, 채팅[메시지], (강제)접속종료	
클라이언트 이름	[NAME]/userName
방[대기실 등] 입장	[JOIN_ROOM]/roomNumer/roomPwd
게임방 생성	[CREATE_ROOM]/roomName/roomPwd/userName
방목록 요청	[ROOMS]
채팅[메시지]	[MSG]/message
(강제)접속종료	[QUIT]

C_GameRoom[게임방 제어] : 채팅[메시지], 방 나가기, 레디, 자리(열고 닫기), 강퇴, 게임 시작, (강제)접속종료	
채팅[메시지]	[MSG]/message
방 나가기	[EXIT_ROOM]/userName
레디	[READY]/index/Ready or Not ready
자리(열고 닫기)	[PLACE]/index/Open or Close
강퇴	[FIRED]/userName
게임 시작	[GAME_START]
(강제)접속종료	[QUIT]

C_TetrisRoom [테트리스 시작한 방 제어] : 테트리스 명령어, 채팅, 이김, 게임오버[대기] & 게임 종료, 방 나가기, (강제)접속종료	
테트리스 명령어	[TETRIS_CMD]/userName/keyCode
게임오버[대기] & 게임 종료	[GAME_OVER]/userName
방 나가기	[EXIT_ROOM]/userName

- 서버 스레드 → 클라이언트 스레드 및 GUI 프로토콜 목록

S_ClientThread [로그인전 제어] : 로그인, 회원가입, (강제)접속종료	
로그인	writer1 [LOGIN]/SUCCESS/id writer2 [LOGIN]/SUCCESS or FAILURE [LOGIN]/FAILURE/INVAILD_VALUE or ALREADY_CONNECTED
회원가입	writer2 [JOIN]/SUCCESS or FAILURE [JOIN]/FAILURE/ALREADY_JOIN
S_WaitingRoom [대기실 제어] : 이름, 방[대기실 등] 입장, 방 생성, 채팅[메시지], (강제)접속종료	
방[대기실 등] 입장	Writer1 [JOIN_ROOM]/roomNumer/roomName/ roomOwnerName writer2 [JOIN_ROOM]/SUCCESS or FAILURE [JOIN_ROOM]/FAILURE/FULL or WRONGPWD

S_WaitingRoom [대기실 제어] : 이름, 방[대기실 등] 입장, 방 생성, 방 목록 요청, 채팅[메시지], (강제)접속종료	
방[대기실 등] 입장	Writer1 [JOIN_ROOM]/roomNumer/roomName/ roomOwnerName writer2 [JOIN_ROOM]/SUCCESS or FAILURE [JOIN_ROOM]/FAILURE/FULL or WRONGPWD [EXIT]/userName [ENTER]/ userName [PLAYERS]/userName.....
방생성	[CREATE_ROOM]/roomNum/roomName
방목록 요청	[ROOMS]
채팅[메시지]	[MSG]/message
(강제)접속종료	[DISCONNECT]/userName

S_GameRoom[게임방 제어] : 채팅[메시지], 방 나가기, 레디, 자리(열고 닫기), 강퇴, 게임 시작, (강제)접속종료	
채팅[메시지]	[MSG]/message
방 나가기	[ROOM_DESTROYED]
레디	[READY]/index/Ready or Not ready
자리(열고 닫기)	[PLACE]/index/Open or Close
강퇴	[FIRED]/userName
게임 시작	writer1 [GAME_START] writer2 [GAME_START]/SUCCESS [GAME_START]/FAILURE/NOT_START_ALONE or NOT_READY_ALL
(강제)접속종료	[DISCONNECT]/userName

S_TetrisRoom [테트리스 시작한 방 제어] : 테트리스 명령어, 채팅, 이김, 게임오버[대기] & 게임 종료, 방 나가기, (강제)접속종료	
테트리스 명령어	[TETRIS_CMD]/userName/keyCode
게임오버[대기] & 게임 종료	[GAME_OVER]/userName [GAME_END]/winnerName
방 나가기	[GIVE_UP]/userName [EXIT]/userName
(강제)접속종료	[DISCONNECT]/userName

- 본래 이렇게 단순하진 않고 각각 메시지를 받고 어떻게 처리해야하는지도 기술하였는데, 이걸 모두 적자니 보고서가 너무 길어져서 생략하였습니다. [별도로 참고파일로 같이 보냈습니다.]

- 샘플로 하나만 따라가 보자면, 예를 들어 게임방에서 누군가가 방을 나갔을 때의 상황을 따라가 보겠습니다.

• 클라이언트(GUI) → 서버 스레드

C_GameRoom[게임방 제어] 중,

- 방 나가기

GUI_GameRoomJFrame에서 Back_jButtonActionPerformed() 수행시,

- c_GameRoom.exitRoom()

: [EXIT_ROOM]/userName 전송

• 서버 스레드 → 클라이언트 스레드 및 GUI

S_GameRoom[게임방 제어] 중,

- 방 나가기

[EXIT_ROOM]/userName : 해당 클라이언트가 방을 나감

=> // 해당 클라이언크가 방장인지 확인한다.

-> if 방장이 아닐 경우,

// 해당 클라이언트만 방에서 나가게 한다.

sendToRoom. 게임방 사람들에게 자신이 나옴을 알린다. [EXIT]userName

// 사용자의 대기실 방 번호를 0으로 지정한다.

// 대기실로 보낸다.

사용자에게 방에서 나가서 대기실로 감을 알린다. **sendTo** [EXIT_ROOM]

대기실의 다른 사람들에게 자신의 입장을 알린다.

sendToRoom. [ENTER]/userName

사용자에게 대기실에 있는 사용자 이름 리스트를 전송한다.

sendTo [PLAYERS]/userName.....

// 루프를 빠져나온다. 이후는 S_WaitingRoom.controlStart()에서 다시 통제함.

-> else 방장일 경우,

// 방이 터진다. [터지기 전에 모든 사람들을에게 이를 알리고 방에서 나가게 해야한다]

사용자들에게 방이 터졌음을 알린다. **sendToRoomOther** [ROOM_DESTROYED]

// 기존 게임방에 속해있는 각각의 클라이언트들에 대해서 수행한다.

// 해당 방내의 모든 클라이언트를 방에서 나가게 한다.

// 사용자들의 대기실 방 번호 0으로 지정한다.

// 대기실에 사용자들을 넣는다.

대기실의 다른 사람들에게 자신의 입장을 알린다. **sendToRoom.** [ENTER]/userName

사용자에게 새 방에 있는 사용자 이름 리스트를 전송한다. [PLAYERS]/ userName.....

// 방을 터트린다.

// 루프를 빠져나온다. 이후는 S_WaitingRoom.controlStart()에서 다시 통제함.

의 과정을 진행하게 됩니다.

• 그러면 클라이언트 스레드 및 GUI 쪽에서는

[EXIT_ROOM] : 자신이 방에서 나와서 대기실에서 감을 의미함.

=> // 클라이언트의 대기실 방 번호0을 지정하고,

// 클라이언트의 방 이름[대기실]을 지정한 후,

// 루프를 빠져나온다. 이후, 게임방 제어 C_WaitingRoom.controlStart() 에서 통제함.

- 방 터짐

[ROOM_DESTROYED]

// 방장이 나가서 방이 터졌음을 알린다.

// 클라이언트의 대기실 방 번호0을 지정하고,

// 클라이언트의 방 이름[대기실]을 지정한 후,

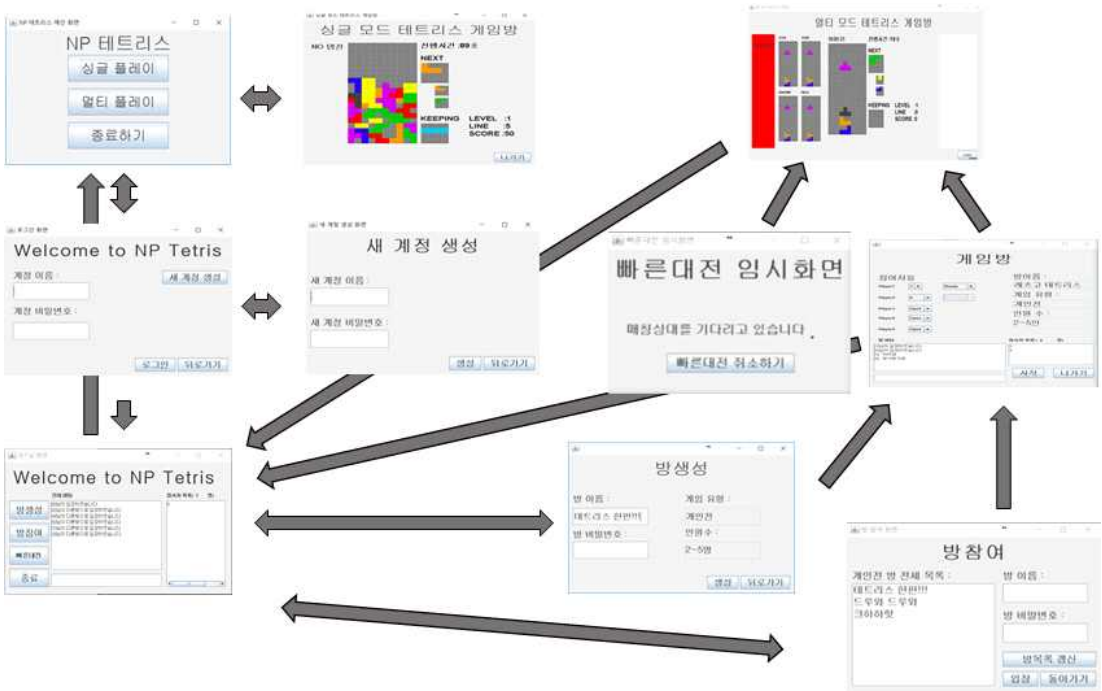
// 본인과 연결된 서버쪽으로 무의미한 메시지를 하나 날린다.

와 같은 과정을 진행하게 됩니다.

나. 순서도

◦ 게임 전체 흐름도

- 알고리즘 및 순서도라고 할만한건 없고, 게임 전체의 흐름도라면 다음과 같습니다.



[게임 전체 흐름도]

4. 소스코드

가. 서버 소스 코드

◦ Server_Start

- 맨 처음에 서버를 구동시키는 클래스입니다.
  - 구동 이후 계속 돌면서 클라이언트의 접속을 받고, 서버 스레드[S_ClientThread]를 만들어서 돌리게 됩니다.

```
// 서버를 실행한다. [얘는 계속 S_ClientThread를 받는 역할만 한다]
public void startServer() {
    try {
        server = new ServerSocket(5000);
        System.out.println("서버소켓[PortNum = 5000]이 생성되었습니다.");
        while (true) {
            // 클라이언트와 연결된 스레드를 얻는다.
            System.out.println("클라이언트를 기다립니다.");
            Socket socket = server.accept(); // 주 소켓
            Socket socket2 = server.accept(); // 클라이언트쪽 메소드용 임시 소켓

            // 스레드를 만들고 실행시킨다.
            S_ClientThread client_Thread = new S_ClientThread(socket, socket2, roomManager);
            client_Thread.start();
        }
    } catch (Exception e) {
        System.out.println(e);
        roomManager.serverDown();
    }
}
```

◦ S_ClientThread

- 클라이언트로부터 접속을 받고 생겨난 스레드로, 클라이언트 C_ClientThread와 1:1로 통신합니다.
- 로그인, 회원가입 등의 기능을 처리하며, 대기실 접속 전까지의 기능을 담당합니다.
- 로그인, 회원가입 등의 기능은 S_Login의 기능을 써서 처리합니다.

```
public void run() {
    isRuuning = true;
    // 수신 메시지와 파싱 메시지 처리하는 변수 선언
    String msg;
    String[] rmsg;

    while (isRuuning) {
        try {
            System.out.println("S_ClientThread While  대기중");
            msg = reader.readUTF();
            System.out.println("S_ClientThread While: 클라이언트로 부터 받은 값 : " + msg);
            rmsg = msg.split("/"); // '/' 구분자를 기준으로 메시지를 문자열 배열로 파싱

            // 로그인 : msg = [LOGIN]/id/pwd
            if (rmsg[0].equals("[LOGIN]")) {
                if(s_Login.login(rmsg[1], rmsg[2])){// 로그인에 성공했을 경우,
                    // 이후는 S_WaitingRoom.controlStart()에서 통제함.
                    isRuuning = s_WaitingRoom.controlStart();
                }
            }

            // 회원가입 : msg = [JOIN]/id/pwd
            else if (rmsg[0].equals("[JOIN]")) {
                s_Login.join(rmsg[1], rmsg[2]);
            }

            // 접속 종료
            else if (rmsg[0].equals("[QUIT]")) {
                isRuuning = false;
            }
        } catch (Exception e) {
            isRuuning = false;
        }
    }
```

○ S_WaitingRoom

- 서버 대기실에 접속한 이후부터 게임방에 들어가기 전까의 기능[방생성, 방참여, 빠른대전 등]을 담당하며, 클라이언트 C_WaitingRoom와 1:1로 통신합니다.
- 대체로 S_RoomManager의 기능을 써서 처리합니다.
- 대충 다음과 같이 구성됩니다. [전체 코드는 길어서 올리지 않겠습니다]

```
// 로그인 성공후, 대기실화면부터 게임제어 서버
public boolean controlStart() {
    boolean isRuuning = true;
    // 대기화면으로 넘어간 이후의 흐름제어
    System.out.println("대기실 화면으로 넘어간 이후의 흐름제어");
    // 수신 메시지와 파싱 메시지 처리하는 변수 선언
    String msg;
    String[] rmsg;

    while (isRuuning) {
        try {
            System.out.println(c.getUserName() + " | S_WatingRoom While 대기중");
            msg = reader.readUTF();
            System.out.println(c.getUserName() + " | S_WatingRoom While : 클라이언트로 부터 받은 값 : "
            rmsg = msg.split("/"); // '/' 구분자를 기준으로 메시지를 문자열 배열로 파싱

            // 클라이언트 이름 : msg = [NAME]/userName
            if (rmsg[0].equals("[NAME]")) {
                c.setUserName(rmsg[1]); // userName를 정한다.
            }

            // 방[대기실 등] 입장 : msg =[JOIN_ROOM]/roomNumber/roomPwd
            else if (rmsg[0].equals("[JOIN_ROOM]")) {
                System.out.println("[JOIN_ROOM]");
                int roomNumber = Integer.parseInt(rmsg[1]);

                if (!roomManager.isFull(roomNumber)) { // 방이 찬 상태가 아니면
                    System.out.println("방이 꽉차지 않음");
                    if (roomManager.isEnter(roomNumber, rmsg[2])) {
                        System.out.println("방 입장에 성공함");
                        if (roomNumber == 0) {// 맨처음 대기실에 입장하는 상황이라면
                            // 대기실에 입장하라고 한다.
                            roomManager.enterWaitingRoom(c);
                        } else {
                            // 게임방에 입장하는 상황이라면
                            // 게임방에 입장하라고 한다.
                            roomManager.enterGameRoom(roomNumber, c);
                            // 사용자에게 방에 성공적으로 입장하였음을 알린다.
                            writer2.writeUTF("[JOIN_ROOM]/SUCCESS");
                            // 이후는 S_GameRoom.controlStart()에서 통제함.
                            s_GameRoom.controlStart();
```

○ S_GameRoom, S_TetrisRoom

- S_GameRoom은 게임방에 들어온 이후부터, 테트리스가 시작하기 전까지를, S_TetrisRoom는 테트리스가 시작한 이후부터 끝나서 대기실로 가기 전까지의 기능을 담당합니다.
- S_WaitingRoom와 유사한 구조로 되어있으며, 역시 S_RoomManager의 기능을 써서 처리합니다. [전체 코드는 너무 길어서 올리지 않겠습니다]

○ S_RoomManager

- 서버에서 단 한 개만 존재하고, S_Room과 같은 전체 방들을 관리하면서 각 서버 스래드들의 요청을 수행하는 클래스입니다.

```java
package Server_Function;

import java.util.Vector;

//게임 내의 방들을 모두 관리하는 클래스 && 메시지를 전달하는 클래스
public class S_RoomManager {

    static private int roomNumberCount;// 방번호는 각 방의 고유 번호로써, 중복되면 안된다.

    private Vector<S_Room> rList; // 전체 방 리스트
    private Vector<S_Client> totalCList; // 접속중인 전체 클라이언트 리스트 [중복 로그인 방지때 사용]
    private Vector<S_Client> quickMatch_waitingCList; // 빠른대전 대기 클라이언트 리스트

    // 단순한 생성자.
    S_RoomManager() {
        roomNumberCount = 0;
        this.rList = new Vector<S_Room>();
        this.totalCList = new Vector<S_Client>();
        this.quickMatch_waitingCList = new Vector<S_Client>();
    }
```

- 생성자 외에도, addRoom, removeRoom, addClient, removeClient, getRoom, disconnectServer, createNewRoomAndEnter, enterRoom, enterWaitingRoom, enterGameRoom, exitRoom, exitRoom, exitGameRoom, exitTetrisRoom, firedUser, tetrisStart, gameOver, sendToRoomMSG, sendToRoomOtherMSG, isFull, isEnter, isAccept, serverDown, getRoomNames, getroomNumberCount, getClientCountInRoom, addClient_QuickMatch_waitingCList, removeClient_QuickMatch_waitingCList과 같은 다양한 메소드들이 존재합니다.
- 전체를 다 올릴 수는 없고... 자주 쓰이는 메소드로 해당 방에 메시지를 뿌리는 sendToRoomMSG, sendToRoomOtherMSG 메소드는 다음과 같습니다.

```java
// 해당 방번호에 msg 메시지를 전송하는 경우,
public void sendToRoomMSG(int roomNumber, String msg) {
    getRoom(roomNumber).sendToRoom(msg);
}

// 해당 방번호에 msg 메시지를 전송하는 경우,
public void sendToRoomOtherMSG(S_Client c, String msg) {
    getRoom(c.getRoomNumber()).sendToRoomOthers(c, msg);
}
```

- 또한, 앞서 프로토콜 중 예시로 들었던 '방나가기 기능의 경우', 서버에서 [EXIT_ROOM]메시지를 받으면 roomManager.exitGameRoom(c);를 실행시키는데, exitGameRoom의 내용은 다음과 같습니다.

```
// 해당 사용자가 roomNumber 게임방에서 나간다.
public void exitGameRoom(S_Client c) {
    int roomNumber = c.getRoomNumber();
    // -> if 방장이 아닐 경우,
    if (!getRoom(roomNumber).getRoomOwnerName().equals(c.getUserName())) {
        System.out.println(c.getUserName() + ". " + roomNumber + "번 게임방 퇴장 ExitGameRoom");
        // 우선, 해당 사용자를 게임방에서 나가게 한다.
        getRoom(roomNumber).removeClient(c);
        c.setInGameRoom(false);

        // 기존 게임방의 사람들에게 해당 사용자의 퇴장을 알린다.
        getRoom(roomNumber).sendToRoom(String.format("[EXIT]/%s", c.getUserName()));

        // 해당 사용자의 방번호를 0으로 설정하고,
        c.setRoomNumber(0);

        // 대기실에 해당 사용자를 넣고,
        getRoom(0).addClient(c);

        // 사용자에게 방에서 나가서 대기실로 감을 알린다. [EXIT_ROOM]
        c.sendTo("[EXIT_ROOM]");

        // 대기실에 있는 다른 사람들에게 해당 사용자의 입장을 알린다.
        // msg = [ENTER]/userName
        getRoom(0).sendToRoom(String.format("[ENTER]/%s", c.getUserName()));

        // 사용자에게 대기실에 있는 사람들의 이름 리스트를 전송한다.
        // : [PLAYERS]/userName 목록 ....
        c.sendTo(getRoom(0).getNamesInRoom());
    } else { // -> else 방장일 경우,
        // 방이 터져야 함. 방이 터지기 전에,
        // 방장에게는 방에서 나갔음을 알리고,
        c.sendTo("[EXIT_ROOM]");

        // 방에 속해있는 나머지 사람들에게 방이 터졌음을 알린다.
        getRoom(roomNumber).sendToRoomOthers(c, "[ROOM_DESTROYED]");

        // 방이 터지기 전에, 해당 방내의 모든 사람들을 방에서 나가야 한다.
        S_Client tempC;
        int size = getRoom(roomNumber).getcList().size();
        for (int i = 0; i < size; i++) {
            // 기존 게임방에 속해있는 각각의 클라이언트들에 대해서 수행한다.
            tempC = getRoom(roomNumber).getcList().get(i);
            // 사용자를 게임방에서 나가게 한다.
            getRoom(roomNumber).removeClient(tempC);
            tempC.setInGameRoom(false);

            // 사용자들의 대기실 방 번호를 0으로 지정한다.
            tempC.setRoomNumber(0);

            // 대기실에 사용자를 넣는다.
            getRoom(0).addClient(tempC);

            // 대기실에 있는 다른 사람들에게 해당 사용자의 입장을 알린다.
            // msg = [ENTER]/userName
            getRoom(0).sendToRoom(String.format("[ENTER]/%s", tempC.getUserName()));

            // 사용자에게 대기실에 있는 사람들의 이름 리스트를 전송한다.
            // : [PLAYERS]/userName 목록 ....
            tempC.sendTo(getRoom(0).getNamesInRoom());
        }
        // 방이 터진다. [없어진다]
        System.out.println("방을 없앤다.");
        removeRoom(getRoom(roomNumber));
    }
}
```

- 나머지도 비슷하게 구성되며, 이 외의 내용은 생략하겠습니다.

나. 클라이언트 소스코드

◦ Client_GUI_Start

- 말 그대로 클라이언트의 메인프레임[GUI_MainJFrame]을 띄우는 클래스입니다.

```
package Client_GUI_Form;

public class Client_GUI_Start {
    static public void main(String args[]){
        GUI_MainJFrame main = new GUI_MainJFrame();
    }
}
```

◦ GUI_MainJFrame

- C_ClientThread를 가지고 있습니다. 싱글 플레이와 멀티 플레이를 했을 때, 각각 해당 화면으로 넘어갑니다. 멀티플레이시, C_ClientThread를 이용해서 서버로 접속을 시도하며, 성공시 '서버 로그인 화면 [GUI_LoginJFrame]'으로 넘어가게 됩니다.

- GUI_MainJFrame의 코드는 다음과 같습니다.

```
public class GUI_MainJFrame extends javax.swing.JFrame {

    C_ClientThread c_ClientThread;
    GUI_SingleTetrisJFrame single;
    GUI_LoginJFrame login;

    public GUI_MainJFrame() {
        initComponents();
        setLocationRelativeTo(null);
        setVisible(true);
        c_ClientThread = new C_ClientThread();
    }

    private void initComponents() {
    // 싱글 플레이 버튼을 눌렀을 때,
    private void siglePlay_jButtonActionPerformed(java.awt.event.ActionEvent evt) {
        single = new GUI_SingleTetrisJFrame(this);
    }
    // 멀티 플레이 버튼을 눌렀을 때,
    private void multiPlay_jButtonActionPerformed(java.awt.event.ActionEvent evt) {
        if(c_ClientThread.connect()){//서버에 접속을 시도한다.
            System.out.println("로그인 화면 이동");
            login = new GUI_LoginJFrame(this, c_ClientThread);
        }else{
            JOptionPane.showMessageDialog(null, "서버가 켜져있지 않던지, 네트워크에 연결되어있지 않습니다.",
        }
    }
    // 종료하기 버튼을 눌렀을 때,
    private void Quit_jButtonActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
        System.exit(0);
    }
```

- 이후에도 비슷하게 다른 GUI프레임도 C_ClientThread를 받아서, 해당 클라이언트 스레드의 기능을 갖다 쓰는 형태로 진행됩니다. [이 외의 GUI_LoginJFrame, GUI_CreateAccJFrame, GUI_WaitingJFrame, GUI_CreateRoomJFrame, GUI_JoinRoomJFrame, GUI_QuickMachJFrame, GUI_GameRoomJFrame, GUI_MultiTetrisJFrame, 등과 같은 코드는 생략하겠습니다.]

○ C_ClientThread

- Server_Start의 서버에 접속해서 S_ClientThread를 생성시키게 하는 스레드로, 서버의 S_ClientThread와 1:1로 통신합니다.
- 로그인, 회원가입 등의 기능을 처리하며, 대기실 접속 전까지의 기능을 담당합니다.
- 코드는 다음과 같이 구성됩니다.

```
// 서버와의 접속 연결하는 메소드
public boolean connect() {
    boolean isConnect = false;
    try {
        System.out.println("서버에 연결을 요청합니다.");
        socket = new Socket("127.0.0.1", 5000);
        socket2 = new Socket("127.0.0.1", 5000);

        System.out.println("---서버 접속 성공--");
        isConnect = true;
        reader = new DataInputStream(socket.getInputStream());
        writer = new DataOutputStream(socket.getOutputStream());

        c_Login = new C_Login(socket, socket2);
        c_WaitingRoom = new C_WaitingRoom(socket, socket2);

        new Thread(this).start(); // 쓰레드를 실행시킨다.
    } catch (Exception e) {
        System.out.println("\n\n서버 연결 실패..\n");
    }
    return isConnect;
}
```

```
// 맨처음 서버와 접속된 이후, 로그인 전까지의 흐름제어
@Override
public void run() {
    isRuuning = true; // 서버로부터의 메시지

    // 수신 메시지와 파싱 메시지 처리하는 변수 선언
    String msg;
    String[] rmsg;

    while (isRuuning) {
        try {
            msg = reader.readUTF();
            System.out.println("C_ClientThread | 서버로 부터 받은 값 : " + msg);
            rmsg = msg.split("/"); // '/' 구분자를 기준으로 메시지를 문자열 배열로 파싱

            //  msg = [LOGIN]/SUCCESS/id
            if (rmsg[0].equals("[LOGIN]")) {
                if (rmsg[1].equals("SUCCESS")) {
                    // 이후는 c_WaitingRoom에서 제어함
                    isRuuning = c_WaitingRoom.controlStart(rmsg[2]);//id 값을 넘겨준다.
                }
            }else if (rmsg[0].equals("[QUIT]")) { // 서버 다운
                JOptionPane.showMessageDialog(null, "알수 없는 이유로 서버가 다운되었습니다.", "서버
                System.out.println("알수 없는 이유로 서버가 종료되었습니다.");
                isRuuning = false;
            }

            // 그냥 돌아가는 중
        } catch (Exception e) {
            System.out.println(e);
            System.out.println("클라이언트 쪽에서 에러가 발생해서, 서버와의 연결이 끊겼습니다.");
            try {
                // 서버쪽에 클라이언트와 접속을 끊으라고 전달함.
                writer.writeUTF("[QUIT]");
            } catch (IOException e1) {
                e1.printStackTrace();
            }
            System.out.println("서버와 접속을 종료함");
            isRuuning = false;
        }
    }
}
```

◦ C_WaitingRoom

- 서버 대기실에 접속한 이후부터 게임방에 들어가기 전까의 기능[방생성, 방참여, 빠른대전 등]을 담당하며, 서버스레이든 S_WaitingRoom와 1:1로 통신합니다.
- 클라이언트쪽은 S_RoomManager같이 통합적으로 관리하는 클래스가 없기 때문에, 각 클래스별로 필요한 기능을 구현해서 처리합니다.
- 대충 다음과 같이 구성됩니다. [전체 코드는 길어서 올리지 않겠습니다]

```
while (isRuuning) {
    try {
        System.out.println(c.getUserName() + " | C_WaitingRoom While 대기중");
        msg = reader.readUTF();
        System.out.println(c.getUserName() + " | C_WaitingRoom While : 서버로 부터 받은
        rmsg = msg.split("/"); // '/' 구분자를 기준으로 메시지를 문자열 배열로 파싱

        // 채팅[메시지] : msg =[MSG]/msg
        if (rmsg[0].equals("[MSG]")) {
            recvTalk(rmsg[1]);
        }

        // 방[대기실 등] 입장함 : msg
        // =[JOIN_ROOM]/roomNumber/roomName/roomOwnerName
        else if (rmsg[0].equals("[JOIN_ROOM]")) { // 방 입장에 성공함
            c.setroomNumber(Integer.parseInt(rmsg[1])); // 방 번호 지정
            c.setRoomName(rmsg[2]);
            if (c.getroomNumber() != 0) // 대기실이 아닐경우,
                // 이후는 c_GameRoom에서 제어함
                c_GameRoom.controlStart(rmsg[3]);
        }

        // 게임방 생성됨 : msg = [CREATE_ROOM]/roomNumber/roomName
        else if (rmsg[0].equals("[CREATE_ROOM]")) { // 새로운 방이 만들어짐을 받음
            // 방 목록에 방을 추가
            rList.add(new C_Room(Integer.parseInt(rmsg[1]), rmsg[2], 5));
            if (joinRoomJFrame != null)
                roomsInfo();// 방목록을 갱신한다.
        }

        // 접속자 목록 : msg = [PLAYERS]/userName1 /t userName2 ...
        else if (rmsg[0].equals("[PLAYERS]")) { // 방에 있는 사용자 명단
            if (!rmsg[1].equals(" ")) {
                // 띄어쓰기 공백 ' '을 없앤후, nameList를 갱신한다.
                resetNameList(rmsg[1].substring(1, rmsg[1].length()));
            }
        }
```

◦ C_GameRoom, C_TetrisRoom

- C_GameRoom은 게임방에 들어온 이후부터, 테트리스가 시작하기 전까지를, C_TetrisRoom는 테트리스가 시작한 이후부터 끝나서 대기실로 가기 전까지의 기능을 담당합니다.
- C_WaitingRoom와 유사한 구조로 되어있습니다. [전체 코드는 올리지 않겠습니다]
- 나머지도 비슷하게 구성되며 이 외의 내용은 너무 길어서 생략하겠습니다.

5. 프로젝트 진행에 따른 문제점 및 개선방안

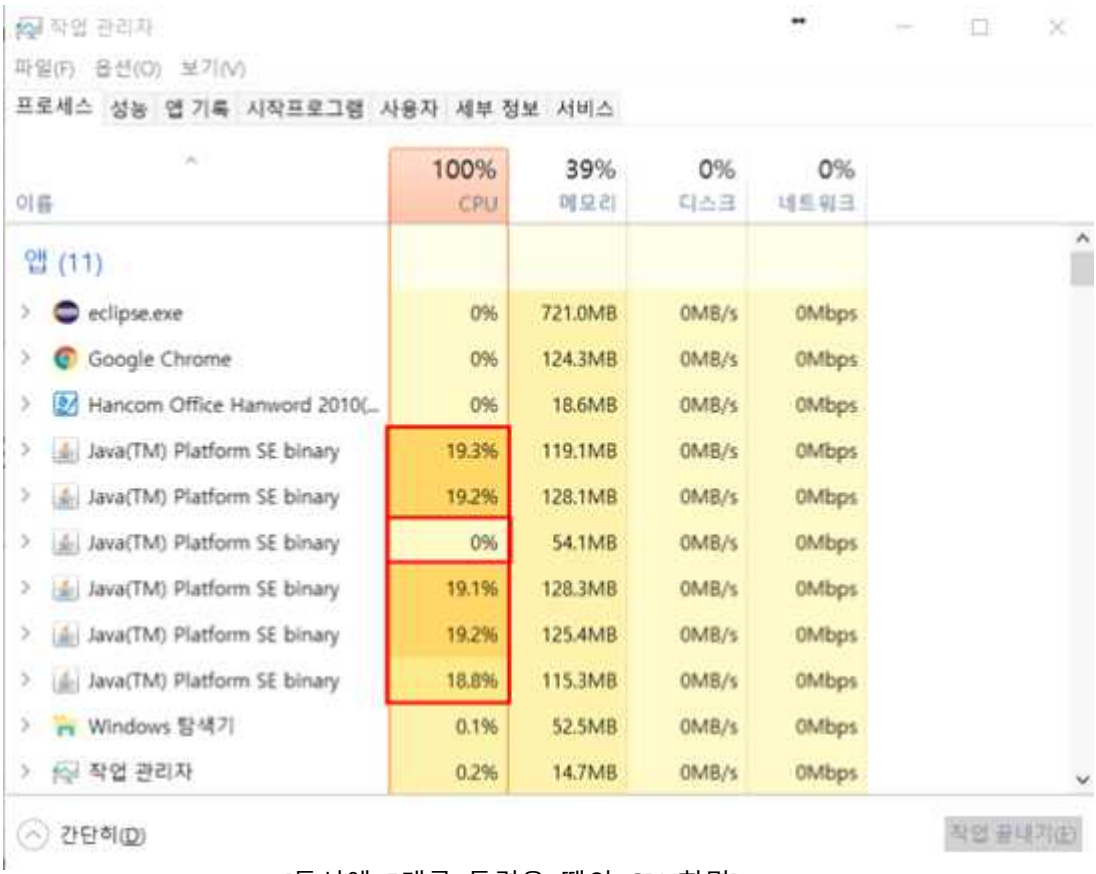
가. 문제점

○ 지금 코드 자체의 문제점만 적으라는 것 같아서 완성상태의 문제점만 적겠습니다.

- 발표 PPT자료에는 지금까지 개발을 하면서 마주쳤던 수많은 문제점들과 시행착오를 담은 '코딩 비하인드 스토리'를 넣었습니다. [본래 보고서에도 꼭 넣으려고 했지만 앞의 이유로 생략함]

○ 완벽한 동기화 불가능

- 제가 코드를 완성시키고 테스트를 해본 결과 노트북 하나에서 모두 돌려보자니 동시에 3개까지는 무리없이 테트리스가 동기화가 되는 것을 확인할 수 있었지만, 그 외에 4~5개가 되는 순간부터 CPU 사용률 100%와 같은 과부하가 오면서 동기화가 되지 않는 문제가 있었습니다.
- 완벽하게 동기화를 구현하려했다면 테트리스 각각의 코드가 서버에서 돌아가고, 클라이언트가 어떤 명령을 보내면 서버가 그걸 받아서 서버내 테트리스에 반영하고, 결과를 클라이언트들에게 보내주는 형식으로 구현을 해야 했지만, 각각의 클라이언트들마다 각각의 테트리스 패널이 돌아가고 있는 형식이라 이게 불가능했습니다.



[동시에 5개를 돌렸을 때의 CPU화면]

○ 예외처리 부족

- 핵심 기능에 대한 구현을 완료한 시점이 최종발표 바로 전날 완료했던터라, 서버 및 클라이언트의 강제 종료에 대한 예외처리가 아직 부족합니다.
- 그 외에도 세부적으로 미처 생각하지 못한 부분이 조금씩 있습니다.

나. 개선방안

○ 소리 및 아이템 추가

- 제안서에는 보류라고 적어놨지만, 맨 처음에 테트리스라면 당연히 테트리스 특유의 BGM도 넣고, 아이템도 넣어야지라는 생각도 해서 미리 준비를 해놨었습니다. [BGM준비 및 아이템 구상]
- 하지만 핵심 기능도 문제가 계속 발생하면서 이를 구현조차 못하게 되었습니다.

추후에 개선을 하게된다면 이러한 부분들을 추가해서 좀 더 게임의 완성도를 높일 수가 있습니다.

[준비 및 구상만 해놓고 구현하지 못했던 BGM 및 아이템]

6. 중요 프로젝트 변경사항

○ 개발 언어 변경

- 개발환경에서도 설명했던 것처럼 맨 처음 제안서에 개발 언어를 C++로 하겠다는 이유로 별의별 이유를 대며 C++로 개발을 하겠다고 했었지만, GUI로 테트리스를 표현하는데 근본적인 한계[GUI에는 절대좌표의 개념이 없다는 문제]에 부딪히게 되었고, 언어를 자바로 바꾸게 되었습니다. 좀 더 따지면 자바의 애플릿 기능을 이용하기 위해서였습니다.
- 그 외에 맨 처음의 프로젝트 제안서에서 제안했던 기능들은 거~의 다 구현했다고 생각합니다. [패널간의 제어문제로 테트리스 방에서 채팅기능을 뺀 것을 제외하면 다 구현함. 오히려 빠른대전 기능도 추가로 구현함]

7. 기타 건의사항

○ 참고할만한 예시를 더 많이 보여주셨으면 좋겠습니다.

- 음... 지금까지 소켓 프로그래밍이란걸 한번도 안했던 사람입장에서, 이번 네프 수업은 의미있는 수업이었습니다. 개인 프로젝트라서 혼자서 죽이되던 주구장창 매달리며 플젝을 완성시키기도 했고, 네트워크 소켓 프로그래밍을 처음해보며 서버가 어떻게 돌아가는지 대~략적인 감을 잡기도 했기 때문이죠.

- 하지만 그것과는 별개로 수업 자체는 프로젝트를 완성시키는데 있어서 크게 도움이 되지 않았(?)던 것 같습니다. 그 이유는 너무 이론적으로 파고들어서 그랬던 것 같습니다.
- 물론 이론 수업이 도움이 안 되었던것도 아니고, 아예 생초짜들에게 이론을 넘기고 예제를 들어가는것도 문제이긴 한데... 이론만 다룬 것 같은 느낌이 듭니다. 소켓이 뭐고 포트가 뭐고 TCP, UDP가 뭐고 bind, accept 등의 과정을 배우긴 했지만 이를 실제로 어떻게 써먹어야 하는지, 전체적인 틀은 어떻게 이루어지는지에 대해서는 배우지 못했습니다.
- 물론 여기서도 네프 과제로써 교수님께서 과제를 내주셨고, 이것도 도움이 많이 된 것은 사실입니다. 하지만 이것과 실제로 진행해야하는 프로젝트 수준과의 차이가 꽤나 컸던 것 같습니다. 그래서 네프 수업은 그냥 '개념적으로 이렇구나'하고 넘어갔고, 실제 코딩에 있어서는 인터넷에서 실제로 어떻게 통신이 이뤄지는지 다양한 예제들을 보고 이를 응용, 변형해서 개발했습니다.

- 결론을 내보자면, 서버, 클라이언트간의 프로토콜이라던지, 실제로 어떻게 통신이 이뤄지는지 참고할만한 예시를 좀 더 많이 보여주셨으면 프로젝트를 진행하는데 있어서 훨씬더 많은 도움이 되지 않았을까 하는 생각이 듭니다.