최종 보고서 (Sophist)

2013112130 정재엽 2014112039 박태수
2014112074 손정인 2016112129 김태건

개발 환경

OS : OS X 및 Ubuntu 16.04
IDE : 이클립스
사용언어 : JAVA
사용한 협업 툴 : 깃허브, 레드마인, 소스트리, 깃 크라켄
효과음 제작 오픈소스 툴 주소 : https://github.com/sfbtom/as3sfxr

프로젝트의 목적 및 구현 사항

1. 기존 테트리스의 밋밋한 점을 개선하여 더 나은 UX를 제공하고, 게임으로서의 가치를 더하기 위함
2. 아이템 기능, 멀티 화면, 사운드
3. 코드 길이 TetrisBoard.java(주요 기능 구현)
   초기 : 약 750줄
   완료 후 : 약 1800줄

프로젝트의 선택 이유

1. 왜 테트리스 여야만 했는가?
'테트리스' 라는 게임은 모든 사람들이 게임의 구성에 대해 알고 있고, 쉽게 즐길 수 있으며 접근하기 쉬운 게임이었습니다. 이러한 이유에서 이미 여러 테트리스 게임들이 시장에 나와 성공을 이루기도 했습니다. 그만큼 많은 사람들이 이해하기 쉬우며, 단순히게 즐기기도 쉽고, 충분한 재미가 보장되어 있는 프로그램이었기 때문에 저희 팀의 이목을 사로잡았습니다.

2. 왜 jalhaebojo의 프로그램을 선택하였는가?
테트리스 게임의 여러 오픈소스들을 확인해 본 결과 팀원 모두가 익숙한 JAVA 언어로 작성되어있으며, 타 오픈소스 프로그램보다 좀 더 많은 기능을 제공하고, 가장 많은 확장 가능성을 보였기 때문에 이 프로그램을 선택하였습니다.
jalhaebojo의 테트리스 프로그램은 아래의 기능을 포함하고 있습니다.
- 기본적인 테트리스의 UI 및 게임 진행구성
- 다음 블록이 무엇인지 보여주는 NEXT와 블록 리스트
- 블록 하나를 붙잡아 저장하는 HOLD기능
- 다른 사용자와의 채팅 기능
- 소켓 프로그래밍을 통한 멀티 플레이 기능

여기서 Sophist가 주목하였던 기능은 다음과 같습니다.
- 기본적인 테트리스의 UI 및 게임 진행구성
- 소켓 프로그래밍을 통한 멀티 플레이 기능

3. 왜 해당 기능에 주목하였는가?

저희 팀은 본 테트리스 프로그램과 시장에서 성공을 거두었던 테트리스 게임의 차이가 무엇인지에 주목하였습니다. 여기서 우리는 사용자가 느끼는 재미의 차이가 게임의 성공과 실패를 결정한다는 결론에 도달하였습니다.

따라서, 저희 팀은 사용자가 좀 더 게임에 흥미를 느끼고, 사용자의 경쟁심리를 자극하며, 때로는 원치 않는 공격을 받아 좌절감을 느낄 수 있도록 프로그램을 확장하기로 하였습니다. 여기서 나온 아이디어가 바로 테트리스 게임의 '**아이템 추가**' 와 '**승/패 기준의 다양화(Mode의 다양화)**', 그리고 '**멀티스크린 구현**' 입니다.

진행 중 발생한 이슈

1. 깃허브 병합 충돌

팀원 모두 이번 프로젝트 이전에 깃허브 사용법을 제대로 숙지하지 못하였습니다. 프로젝트 초기 깃허브 사용법에 대한 공부를 하여 깃허브를 통한 프로젝트 관리를 익히긴 했습니다만, 브랜치에 대한 이해가 부족한 상황이었습니다. 그로 인해 초중반 개별 과제 진행시 브랜치 사용을 하지 않고 master 브랜치 만을 이용하여 진행하였습니다. 이로 인하여 프로젝트 중간 중간 pull시도 중 오류가 발생하거나 commit 에러가 발생하기도 하였습니다. 충돌이 발생하는 경우 커밋을 하기 위해서는 pull을 한 뒤에 commit을 해야 한다는 정보를 이용하여 진행하였으나 충돌된 부분의 코드는 사용자가 직접 수정을 하도록 표시가 되어있어 진행 중 직접 수정하는 작업을 추가적으로 거쳐야 했습니다.

```
public TetrisBlock(int x, int y, Color color, Color ghostColor) {
    this.color = color;
    <<<<<<<<<<<<<<<<<<<<<<< HEAD
    Color item = new Color(255,255,50);//Yellow
    Color item2 = new Color(255, 0, 255);//Pink, 아이템 변수 추가
    int random = (int)(Math.random()*150);//아이템이 나올 확률을 조정.
    =======================
    for(int i=0 ; i<colBlock.length ; i++){
        if(MODE_NUM == 0) {//아이템을 사용할 때
            if(i == random) {
            colBlock[i] = new Block(0,0,item,ghostColor);
            }
            else {
                if(random >= 40 && random <= 43) {//random 변수가 50-53사이
                    colBlock[i] = new Block(0, 0, item2, ghostColor);
                    random += 5; //한 블록에 아이템이 하나만 나오게 하기 위해
                }
                else {
                    colBlock[i] = new Block(0,0,color,ghostColor);
                }
            }
        }
        else if(MODE_NUM == 1){//아이템을 사용하지 않을 때
            colBlock[i] = new Block(0,0,color,ghostColor);
        }
    }
    >>>>>>>>>>>>>>>>>>>>>>> master
    this.rotation(ROTATION_0); //기본 회전모양 : 0도
    this.setPosX(x);
    this.setPosY(y);
}
```

2. 깃허브 사용 미숙으로 인한 이전 버전의 삭제

깃허브를 이용하여 진행 중 이전 인원이 수정하여 올린 코드에 대하여 다른 인원이 수정한 코드를 올리는 바람에 중간에 사운드 및 아이템 구현하였던 버전이 지워지고 사운드 및 아이템이 구현되지 않은 버전으로 코드가 올라간 적이 있습니다. 당시 백업도 없이 올라온 코드들을 pull하여 작업을 진행하였던 터라 상당히 당황하였으나 인터넷 검색을 통한 깃허브 사용법 강의를 통해 이전 버전의 코드도 다시 pull을 통하여 가져와 사용할 수 있음을 확인하였고, 이전 버전의 코드를 가져와 다시 push한 뒤에 다른 인원은 이를 pull하여 다시 수정하여 코드를 올렸습니다. 이런 번거로운 작업을 다시 거치지 않기 위해 저희는 진행 중 코드를 수정하여 push를 할 때 충돌이 생기지 않게 지속적으로 연락했으며 이 후에 이런 문제는 발생하지 않았습니다.

3. 서로 다른 운영체제를 사용하며 생긴 이슈 (Window-LINUX-MAC OS)

플랫폼 독립적인 자바를 이용하여 프로젝트를 진행하였으나 각 플랫폼에서만 사용할 수 있는 프로그램, 예를 들어 mac에서의 m4a 포맷 등은 사용할 수 없었습니다. 또한 자바 작성 시에도 글자 포맷 등의 호환문제가 발생하여 저희 팀원들은 해당 문제의 원인을 파악해 보았고 그 원인은 한글폰트에서만 발생한다는 것을 알게 되어 프로젝트 중간 진행시에는 주석을 영어로 처리하도록 하였습니다.

구현 세부 사항

버그 수정

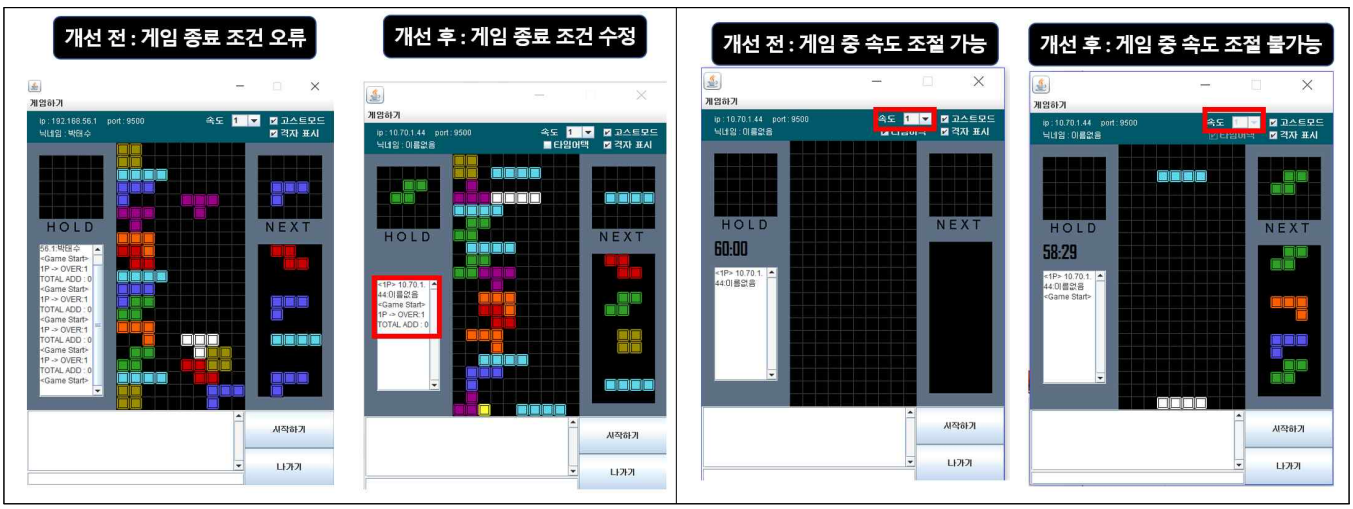
테트리스라는 게임의 특성은 경쟁적이며(Competitive) 신속함(Rapidity)과 무작위성(Randomness)이 요구되는 게임입니다. 특히 이 게임의 가장 중요한 조건은 블록이 화면의 끝까지 올라올 경우 게임이 종료되어야 한다는 것에 있습니다.

1. 종료조건의 수정

처음 Jalhaebojo의 버전에서는 이 종료조건이 올바르지 않게 되어있었습니다. 해당 문제를 코드에서 찾아 수정하는 과정을 통해 버그를 해결할 수 있었습니다.

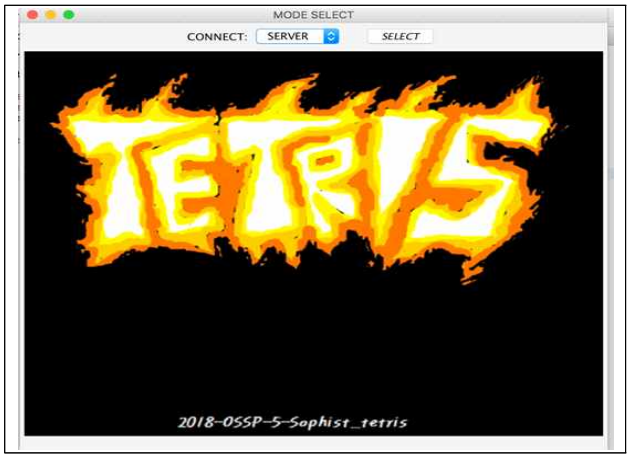
2. 공정한 경쟁 방해 요소 제거

또한 다른 중요한 점은 모든 PvP게임에서 중요한 요소로서 이미 시작한 게임은 중간에 조건이 변경되어서는 안 된다는 점입니다. 만약 플레이어가 자신이 원하는 조건의 게임에 접속을 하였는데 그 조건이 중간에 임의로 수정이 된다면 플레이어의 입장에서는 불공정한 상황이 연출이 될 것입니다. 따라서 이를 사전에 방지하기 위해 게임 중 속도조절이 가능한 점을 변경하여 게임 시작 후에는 속도조절을 할 수 없도록 바꾸었습니다.



시작 화면 구현

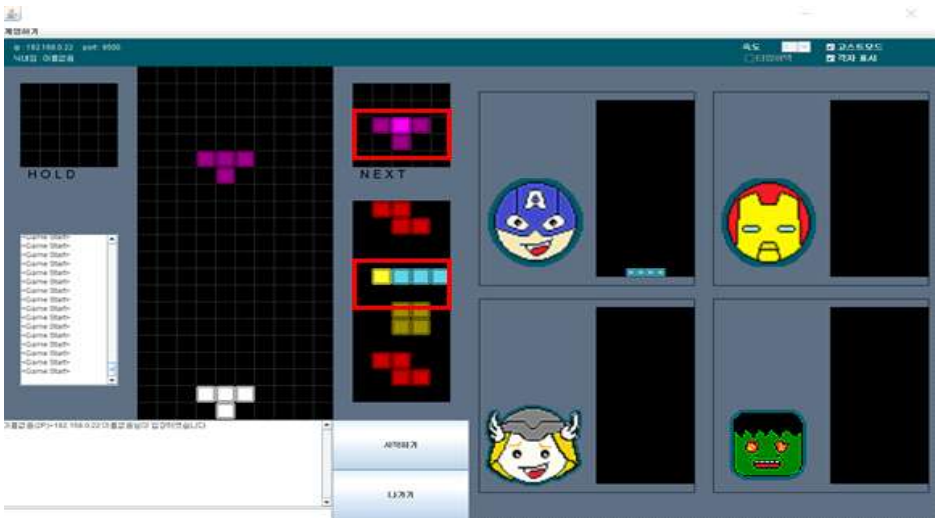
시장에 나와 있는 많은 게임 중 어떤 게임도 바로 게임이 시작되지는 않습니다. 게임의 시작화면은 유저들로 하여금 게임에 대한 첫인상을 결정하고 그 첫인상은 게임의 질과 상관없이 유저의 마음을 긍정적으로, 혹은 부정적으로 움직이게 합니다. 따라서 저희 조는 UX에 중점을 두고 테트리스를 확장하기로 하였기 때문에 시작부분 또한 간단하지만 큰 영향을 준다고 생각을 하여 시작화면을 구성하였습니다. 또한 이전과는 다르게 시작화면에서 서버, 클라이언트를 선택하여 접속할 수 있도록 했고 사용자 입장에서 좀 더 편하게 게임을 즐길 수 있게 되었습니다. (프로그램의 모든 이미지는 저작권 문제를 방지하기 위해 자체제작 하였습니다.)



아이템 구현

기존의 테트리스는 단순한 블록 쌓기 게임이었습니다. 블록만 쌓는 게임은 30년 전의 테트리스이며 현재 시장의 테트리스는 좀 더 경쟁을 할 수 있는 요소들이 추가되어 있습니다. 따라서 저희 조는 몇가지 아이템을 게임에 구현하고자 했습니다.

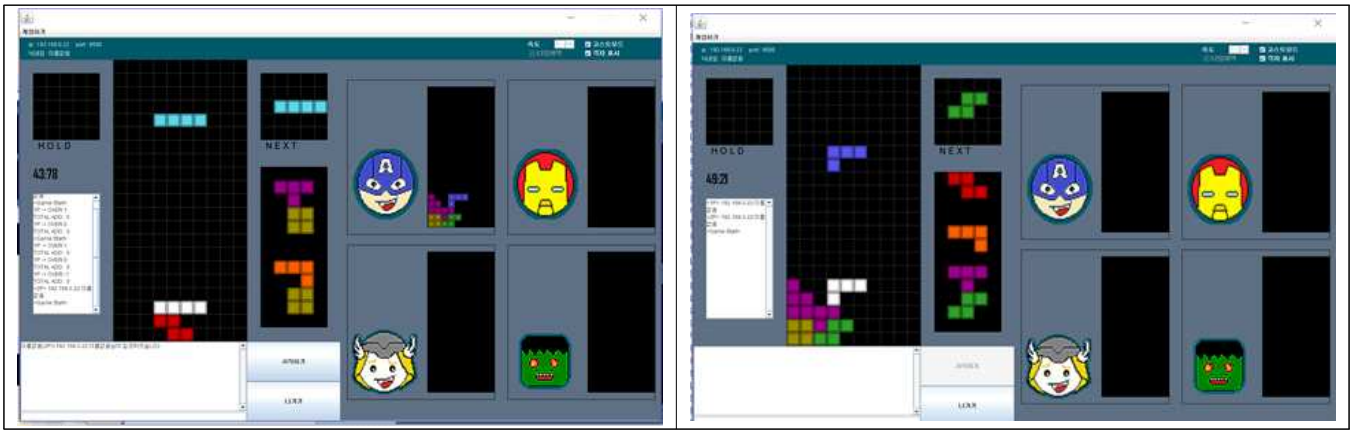
저희 테트리스는 총 2가지 종류의 아이템을 더했습니다.
Clear 아이템은 플레이어의 화면에 있는 모든 블록을 삭제해 위기를 탈출할 수 있게 도와줍니다.
Blind 아이템은 플레이어를 제외한 상대방의 화면을 잠시 보이지 않게 가려버리는 효과를 나타냅니다.
게임이 너무 쉬워지는 것을 방지하기 위해 등장 확률을 낮게 조정하였고, 바로바로 사용되는 코드로 구현하였으나, 플레이어가 원할 때 사용하는 것이 좀 더 게임을 즐길 수 있다고 판단하여 저장하는 형태로 구현 하였습니다.



멀티윈도우 구현

저희 조의 최종 목적은 유저의 흥미를 끌어올려 게임으로서의 가치를 갖출 수 있는 게임의 구현이었습니다. 시장의 게임들은 상업적으로 게임을 운영하기 위하여 유저 입장에서 게임을 구현해 왔습니다. 여기서 시장의 테트리스의 공통점을 찾아보면 모두 단순히 자기 화면만 보여지는 테트리스가 아닌, 다른 플레이어들의 화면도 보이도록 게임이 구성되어 있었습니다.

따라서 저희 Sophist 역시 프로젝트의 중점을 이 환경을 제공해보고자 하는 것에 두었습니다. 구현 과정은 다른 아이템이나 사운드 추가에 필요했던 과정들보다 훨씬 복잡한 과정을 필요로 하였으며 모든 팀원의 협력이 필요했습니다. 구현하기 위해서는 코드의 완벽한 이해와 앞서 구현했던 기능들과의 추가적인 연동이 필요했으며, 기존 프레임의 확장 역시 이루어져야 했습니다.



사운드 구현

게임을 하는 것에 있어서 아무런 소리가 들리지 않는 게임과 여러 상황에서 다양한 사운드요소가 재생되는 게임은 큰 차이가 있습니다. 이는 유저의 게임 몰입에 큰 영향을 끼칩니다. 처음에 아무런 추가구현이 이루어지지 않은 테트리스를 플레이해보며 가장 먼저 느낀 소감은 사운드의 부재로 인한 지루함이었습니다. 따라서 저희 조는 이를 극복하기 위해 다양한 사운드를 추가하였습니다.

먼저 게임 진행에 있어 가장 큰 느낌을 불러오는 BGM을 적용하였으며, 이 후의 과정에서는 각 이벤트 마다 상황에 맞는 사운드가 재생되도록 했습니다. 블록이 떨어지는 경우, 블록이 깨지는 경우, 아이템을 사용하는 경우, 게임이 종료되었을 경우 등을 모두 나누어 각기 다른 사운드를 입혔습니다. 사운드의 경우 저작권 문제가 발생할 우려가 있어 공개된 음원을 이용하여 BGM을 추가하였으며 여러 이벤트 사운드의 경우에는 팀원들이 각각 오픈소스 효과음 제작 툴을 이용하여 제작 후 추가하였습니다.(효과음 제작 툴의 주소는 보고서 상단에 첨부하였습니다.)

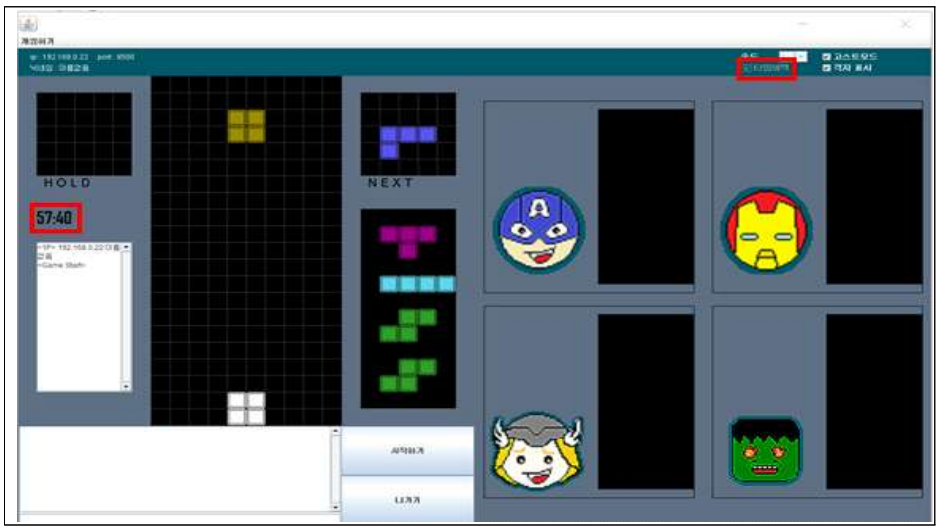
모드 구현

인기 있는 게임들이 한가지의 게임 진행방식만을 고수하지 않고 여러 가지 게임모드를 지원하여 플레이어의 재미를 끌어올리는 것처럼, 저희 조원들은 다양한 게임방식이 있다면 사용자들이 좀 더 흥미를 느낄 수 있을 것이라고 판단했습니다. 아이템을 사용한 게임방식 외에 추가적인 모드가 필요했습니다. 아이디어를 내던 중, 시간에 쫓길수록 좀 더 경쟁심과 스릴을 자극할 수 있다는 아이디어가 나왔습니다. 여기서 구현하게 된 것이 바로 타임어택 모드입니다.

타임어택 모드에서, 모든 플레이어들은 60초를 할당 받게 되며 블록이 깨질 때마다 추가시간을 얻어 좀 더 오래 게임을 진행할 수 있도록 구현하였습니다. 해당 모드에서도 역시 가장 오래 살아남는 플레이어가 승리하게 됩니다만 좀 더 운보다는 실력에 중점을 둔 모드입니다.

본 게임의 모드들은 서버클라이언트에 의하여 선택이 되도록 구현이 되어있습니다. 물론, 게임이 시작이 되면 모드선택 또한 비활성화 되도록 하였습니다.

다음 이미지는 타임어택 모드로 플레이를 했을 때의 플레이어 화면입니다. 기존의 아이템 모드에서 등장했던 아이템 요소들은 등장하지 않으며 화면의 좌측에 게임오버까지 남은 시간을 표시해 주는 타이머가 생기게 됩니다. 해당 타이머는 플레이어들 마다 다르게 적용되어 결국 가장 오래 살아남은 플레이어가 승리하게 되는 방식입니다.



프로젝트를 마치며

프로젝트를 시작하며 완벽히 분석하였다고 생각하였던 코드에 대하여 미쳐 정확히 파악하지 못한 부분들이 있어 프로젝트를 진행하며 여러 난관에 부딪혔습니다. 특히 테트리스 블록들이 기존의 테트리스들과 달리 실제 블록과 화면에 보여지는 블록이 따로 구성이 되어있어 아이템 블록 및 멀티화면을 구현할 때에 많은 시행착오를 불러왔으며 팀원들 간의 협동이 많이 필요하였습니다.

프로젝트를 진행하며 깃허브를 최대한 이용해보려 하였으며 팀원들 간의 약속으로 유튜브의 생활 코딩 채널의 깃허브 강좌를 보고 진행하도록 하였습니다. 깃허브에 관한 강의를 들었음에도 불구하고 브랜치를 사용하는 데 있어 깃허브를 처음 이용하는 팀원들끼리 사용함에는 약간의 어려움이 있었습니다. 그래도 브랜치를 활용한 협업을 어느 정도 할 수 있게 되었으며 저희 조의 마지막 최종이자 공통 과제인 멀티윈도우를 구현할 때에는 따로 브랜치를 생성할 필요가 없어 계속 사용 중이던 master 브랜치에 직접 작업하였습니다. 이런 과정들을 통해 비록 규모는 적지만 협업을 할 때의 깃허브 사용에 좀 더 익숙해 질 수 있겠다고 생각했습니다.

또한 저희 프로젝트에서 미처 해결하지 못한 과제가 한 가지가 있습니다. 바로 서로 다른 네트워크간 통신 시 NAT 환경에서도 접속이 가능하도록 하는 것입니다. 이를 여러 가지 방법을 통해 조사해본 결과 Hole Punching이라는 기능을 이용하여 해결할 수 있음을 찾았고 오픈되어 있는 여러 코드가 있음을 확인하였습니다. 다만 네트워크에 대한 지식이 부족하여 이 해결법을 다소 늦게 확인하였으며 구현하기 위한 학습시간과 실제 구현을 할 시간이 부족해 결국 프로젝트 마무리까지 완성하지 못했습니다. 여기서 프로그래밍에 있어 네트워크에 대한 다양한 지식의 필요성에 대해 학습해야 한다는 것을 느꼈고, 이에 대해 토론했습니다.

현재 개선된 테트리스는 초기의 테트레스에 비하여 유저의 입장에서 보다 개선된 환경을 제공받는다는 점에서 저희 조의 목표를 어느 정도 달성했다고 생각했고 앞으로 여기서 개선해야할 사항으로는 위에서 언급한 NAT 환경에서 서버로의 접속이며 원활한 기능 구현을 위해 방학 중 학습하고, 가능하다면 끝까지 완성하도록 할 것입니다.