테트리스 구현 보고서

## 141079 정원철

뭐할까……

치킨 집 프로젝트를 하려다 잉여력이 부족하여 테트리스를 하기로 결정하였다.

테트리스로 결정한 것은 이번 학기에서 배운 객체지향적인 프로그램을 하기에 전체적으로 소재가 적당해 보였다. 블록들이 모여 블록을 이루고 있고 실시간으로 움직여야 하는 것들 등이 배운 것들을 활용하기에 알맞아 보였다.

시작하는 과정에서 이상하게 시간을 많이 소모한 부분이 2가지가 있다. 일전에 다른 프로그램에서 호환문제가 걸려 JDK 1.8 버전을 1.7로 내린 적이 있다. 이로 인해 path가 꼬였는지 JRE가 로드가 안되기도 하고 실행이 안 되는 등 여러 가지 문제가 생겼다. 윈도우 환경에서 주로 작업하였는데 환경이 바뀌는 경우 이 버전의 다름도 문제가 되었으며 윈도우의 경우 Registry에서 관리하는 설정 값이 있어 결국 모든 Java와 registry값을 삭제하고 재설치 한 다음 환경변수를 하나하나 직접 잡아주었다.(Classpath, path 등)

다른 한 가지는 테트리스를 제작함에 있어 그래도 모양이 조금 예쁘면 좋겠다고 생각했는데 교수님께서 이번 학기에는 GUI보다는 콘솔 환경을 사용할 것을 학기 중반에 권장하셔서 Console color에 대해 여러 가지로 고민하였다. 상당히 많은 시간을 쏟아 외부 Library와 몇 가지 Plug-in을 사용하여 구현하려 했지만 실행 환경에 따라 작동하지 않는 경우가 많고 eclipse console에서 표현하여도 다른 pc로 이동했을 때 작동하지 않거나 의도치 않은 버그가 발생할 수 있어 이 부분은 한 발 물러섰다.

주요이슈

Colsole clear

콘솔로 프로젝트를 진행함에 따라 Console clear가 문제되었다. Java에 System.getProperties()를 사용하여 환경감지 후 해당 콘솔의 clear(linux, mac)나 cls(windows) 메시지를 보낼 수 있었지만 2가지 모두 eclipse console에서는 작동하지 않았다. 추가적인 검색 결과 eclipse에서는 행을 내려 해결하는 방법을 보고 고전적이면서 마음에 들지 않지만 별다른 수가 없어 채택하였다.

Rotation 문제

회전축을 어디로 해야 할까? 회전하려는데 주변에 벽이 있다면 어떻게 처리해야 할까? 그래서 게임을 해봤다. 게임마다 다소 차이가 있었지만 보통은 벽에 붙어 회전을 하게 되면 도는 반경만큼 벽에서 밀려났다.

Keyinput

중간쯤 진행하다가 이제 슬슬 키 입력을 받아야 되지 않을까 하고 생각했다. 그런데 뭔가 마땅한 게 없는 것이 다소 찝찝했는데 검색을 통해 확인한 결과, 자바는 콘솔에 대단히 약하다는 것이다. 일단 콘솔용 keyListener는 존재하지 않고 Color나 Clear또한 존재하지 않는다고 한다. 하지만 콘솔로 하기로 정했으니 그냥 강행한다.

Data Sharing

가장 많은 시간 고민했지만 가장 안 되었던 부분이 데이터 관리이다. 필연적으로 데이터박스가 존재하게 되고 그 데이터를 어느 클래스가 어디까지 사용하는지, 어느 클래스가 소유하는지, 한정자와 데이터의 자료구조 등 여러 가지를 고려했으나 뜻대로 되지 않고 대부분 공개되어버리는 상황이 발생했다. 그 사이에서 최대한 get, set메소드는 삭제했고 chaining또한 자르고 잘라서 줄였다.

Drawing

배열과 직접적인 값 관리를 사용하지 않으려 했지만 출력부에서 많은 부분이 포기됐다. 어떻게 좋은 방법이 없을까 인터넷을 뒤지고 책을 찾아봤지만 그럴싸한 방법은 보이지 않았고 출력하는 부분에서 대부분의 제한사항을 허용해주는 상황이 발생했다.

Interface and Polymorphism

블록이라는 것을 위치에 대한 것으로만 정의하다 보니 굳이 인터페이스나 상속관계가 필요한 상황이 아니게 되어버렸다. 최대한 기능을 넣어 사용하려 했지만 한다면 Block기본에서 7가지 각각의 블록을 상속하는 정도가 될 것인데 애당초 Position말고는 상속을 받을 이유도 받아서도 안 되는 상황이 되어 지금의 코드가 더 합리적이라 판단했다.

정리하며

이래저래 코드가 폭발해버려서 다시 처음부터 짰다. 난 데이터를 잘 날려먹는 습성이 있는 것 같다. 하지만 놀라운 것은 다시 짜는 동안 코드가 놀라울 정도로 작아졌다. 아주 쉬울 것이라 생각했는데 기간을 고려했을 때 버거운 난이도였으며, 다시 비슷한 주제로 한다면 새로운 방식으로 하고싶다.

Restart

처음부터 새로 짜기 시작한 횟수가 이번 프로젝트는 총 4번인 것 같다. 골조가 바뀌어야만 하는 상황이 오면 그 때 프로젝트를 갈아엎는데, 다시 하면서 느낀 것은 같은 내용이 나오지 않는다는 것과 구성의 치밀함 이다.

마지막에 다시 짜기 시작할 때는 코딩을 하는 시간보다 종이에 프로그램 전체 맵을 그리는 시간이 더 많을 정도로 고민했다. 데이터는 어떻게 감출지, 어떤 클래스는 만들고 관계는 어떻게 할지 하나하나 그려보고 충돌이 나면 다시 그리고 이런 지도가 지면에 그려지고 코드로 옮겼더니 테스트 과정에서 에러가 확실하게 줄어들었다. 코드 작성 중반에까지 테스트 한번 없이 진행했지만 버그가 평소에 비해 현저히 낮았다.

Time

예상시간보다 몇 배의 시간이 소모되었다. 당초 할당하였던 코딩시간은 30시간 남짓이었지만 중간에 고집을 부리는 통에 말도 안되게 늘어났다. 아마 처음 기획대로 갔다면 시간 안에 작성했을지도 모르겠다. 하지만 이번 학기에 배운 기능들을 최대한 활용해 보고 싶었고 내가 생각하는 객체지향을 코드로 옮겨보고 싶었다. 막바지에 최종마감시간을 넘길 수는 없어 기능과 코드 모두 다소 타협을 봐야 했지만 나쁘지 않은 선택이라고 생각한다.

마감시간에 조금은 민감할 필요가 있다고 생각했다. 처음 주어진 마감시간에 비해 4일정도 지연되었는데 실제 프로젝트나 업무였다면 상당한 문제라고 생각한다. 내가 작성할 프로그램을 어느 정도 퀄리티로 어느 정도 시간 안에 완료할 수 있는지 알 고 있는 것도 괜찮겠다 생각한다.