

**2차과제보고서**



|  |  |  |
| --- | --- | --- |
| 수강과목 | | 소스코드분석 |
| 담당교수 | | 김순태 교수님 |
| 제출날짜 | 2019년 12월 09일 | |
| 팀 | 11조 | |
| 팀원 | 201613998설경원 | |
| 201818218한별 | |

목차

1. 개요3
2. 팀원 소개3
3. 개선 사항4
4. 기존/ 개선 다이어그램5
5. Refactoring 전후 비교9
6. Refactoring13
7. 모호한 이름의 변수/메소드/클래스 이름 변경13
8. 같은 클래스 내에 존재하는 중복 코드 - 메소드화14
9. 서로 다른 클래스에 존재하는 중복코드 - 클래스 통합16
10. 서로 다른 클래스에 존재하는 중복 코드 – 클래스 상속화17
11. GameWindow 클래스의 복기기능 – 클래스화 및 불필요하게 나누어진 클래스 통합18
12. Timer 클래스 수정 – 분리되어 있는 요소 통합 및 중복 코드 해결21
13. 불필요한 main 제거 및 사용하고 있는 main 클래스화23
14. MVC Pattern Refactoring24
15. Error 처리26
16. 코드의 Relocation30
17. 2차 개선 아이디어 구현31
18. 느낀 점36

1. 개요

1차과제는 지뢰찾기를 구현한 코드를 분석하고 개선사항을 구현하는 것이었습니다.

1차 과제를 통해 개선한 저희의 코드는 컴퓨터는 이해할 수 있지만 사람이 이해하기 힘든 코드였습니다.

2차 과제는 개선 아이디어를 추가적으로 구현하고 기존 코드를 사람이 이해하기 쉽도록 리팩토링을 수행하는 것입니다.

1. 팀원 소개

|  |  |
| --- | --- |
|  |  |
| 201613998  소프트웨어공학과  설경원 | 201818218  소프트웨어공학과  한별 |

1. 개선사항
   1. 1차 과제 개선사항
      1. 타임어택

게임 시작 전, 사용자가 게임시간을 설정하고 설정한 시간을 초과하면 사용자에게 게임을 계속 할 것인지 여부를 물어보는 기능을 넣었습니다.

* + 1. 배경음악, 효과음 삽입

아무런 음향효과가 없는 기존의 프로그램에 배경음악과 효과음(버튼을 눌렀을 때, BUSTED되었을 때)을 삽입했습니다.

* + 1. 키보드를 이용해 플레이할 수 있는 기능 추가

마우스와 키보드를 이용해 플레이 할 수 있도록 하였습니다.

* + 1. 사용자가 맵을 설정할 수 있는 2인용 모드

키보드를 이용하여 플레이 할 수 있는 2인용 모드를 추가하였습니다.

* + 1. 모든 지뢰를 나타내는 기능

지뢰를 찾아 패배하였을 때, 모든 지뢰를 나타내는 기능입니다.

* + 1. 복기 기능 추가

게임에서 이겼을 때, 게임의 진행과정을 복기할 수 있는 기능을 추가하였습니다.

* 1. 2차 과제 개선사항
     1. 방패 아이템 추가

지뢰를 선택하더라도 특정 횟수에 한해 게임을 계속할 수 있도록 하는 아이템을 추가하였습니다

* + 1. 힌트 아이템 추가

특정 횟수에 한해 특정 위치에 지뢰가 있는지 확인할 수 있는 아이템을 추가하였습니다.

(사용 방법: 키보드를 이용하여 지뢰가 있는지 확인하고 싶은 위치로 이동한 후, ‘H’ 버튼을 눌러 힌트 아이템을 이용한다. )

1. 기존/ 개선 다이어그램
   1. 기존 클래스 다이어그램

텍스트이(가) 표시된 사진

자동 생성된 설명

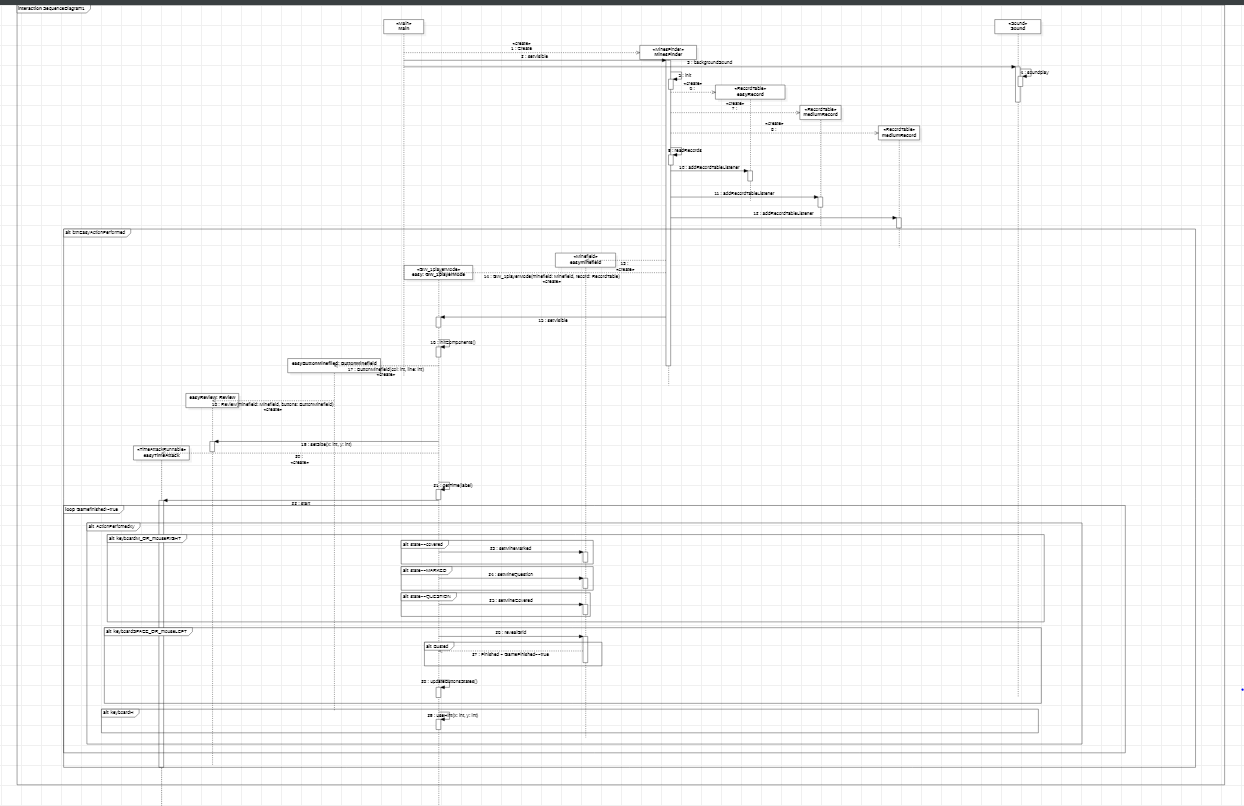
* 1. 리팩토링 후 클래스 다이어그램

텍스트, 지도이(가) 표시된 사진

자동 생성된 설명

* 1. 개선 후 클래스 다이어그램텍스트, 지도이(가) 표시된 사진

     자동 생성된 설명
  2. 개선 후 시퀀스 다이어그램



1. Refactoring 전후 비교
   1. Refactoring 전

|  |
| --- |
|  |
| Stan4j-Pollution Chart |
| 지도, 텍스트이(가) 표시된 사진  자동 생성된 설명 |
| Stan4j-Composition View |
| 스크린샷, 지도이(가) 표시된 사진  자동 생성된 설명 |
| Stan4j-Matrics |

* 1. Refactoring 후

|  |
| --- |
| 스크린샷이(가) 표시된 사진  자동 생성된 설명 |
| Stan4j-Pollution Chart |
| 텍스트, 지도이(가) 표시된 사진  자동 생성된 설명 |
| Stan4j-Composition View |

|  |
| --- |
| 스크린샷이(가) 표시된 사진  자동 생성된 설명 |
| Stan4j-Matrics |

1. Refactoring

|  |  |  |  |
| --- | --- | --- | --- |
| 개선 전 클래스 | 개선 전 변수/메소드/클래스 이름 | 개선 후 클래스 | 개선 후 변수/메소드/클래스 이름 |
| SoundEffect | SoundEffect | Sound | Sound |
| SoundEffect | Effectplay | Sound | pressButtonSound |
| SoundEffect | Effectplay\_1 | Sound | BackgroundSound |
| GameWindow | GameWindow | GW\_1playerMode | GW\_1playerMode |
| MultiGameWindow | MultiGameWindow | GW\_2playerMode | GW\_2playerMode |
| MultiGameWindow | userChooseMap | GW\_2palyerMode | setMapSize |
| MultiGameWindow | R | GW\_2playerMode | randomTurn |
| MultiGameWinow | Random\_1 | GW\_2playerMode | Player1Turn |
| MultiGameWindow | Rnadom\_2 | GW\_2playerMode | Player2Turn |
| TheRecordOfAGame | TheRecordOfAGame | Review | Review |
| TheRecordOfAGame | showRec1, showRec2 | Review | ReviewTheGame |
| TheRecordOfAGame | recClickX | Review | reviewMapClickedX |
| TheRecordOfAGame | recClickY | Review | reviewMapClickedY |
| TheRecordOfAGame | recClcik | Review | reviewMapClicked |
| TheRecordOfAGame | recSizeX | Review | reviewMapSizeX |
| TheRecordOfAGame | recSizeY | Review | reviewMapSizeY |

* 1. 모호한 이름의 변수/메소드/클래스 이름 변경

Eclipse에서는Alt+Shift+R(Rename - Refactoring) key binding을 이용하여 리팩토링 과정에서 rename을 수행할 수 있다.

* 1. 같은 클래스 내에 존재하는 중복 코드 – 메소드화
     1. Sound 클래스

|  |
| --- |
|  |
| 개선 전 |
|  |
| 개선 후 |
| 설명 |
| 개선 전과 개선 후를 나타낸 위의 두 그림을 비교하여 보면 한 클래스 내의 두 메소드 안에 같은 코드가 들어있어 메소드 추출 기법을 이용하여 중복되는 코드를 없앤 것을 볼 수 있다. |

* + 1. Minefield클래스

|  |  |
| --- | --- |
|  | |
| 개선 전 | |
|  | 그림입니다. 원본 그림의 이름: CLP000044a00014.bmp 원본 그림의 크기: 가로 401pixel, 세로 127pixel그림입니다. 원본 그림의 이름: CLP000044a00015.bmp 원본 그림의 크기: 가로 643pixel, 세로 337pixel |
| 개선 후 | |
| 설명 | |
| Minefield 클래스와 multiminefield 클래스를 minefield클래스로 통합함으로써 minefield 클래스 내에 유사한 2개의 생성자가 만들어졌습니다. 개선 후를 나타낸 위의 그림을 보면 중복되는 코드를 메소드화하여 중복 코드를 제거한 것을 볼 수 있습니다. | |

* 1. 서로 다른 클래스에 존재하는 중복코드 - 클래스 통합

|  |
| --- |
| 그림입니다. 원본 그림의 이름: CLP000044a00014.bmp 원본 그림의 크기: 가로 401pixel, 세로 127pixel |
| 개선 후 |
| 설명 |
| 기존 코드에서는 Minefield 클래스와 MultiMinefield 클래스에 중복 코드가 존재하는 문제가 있어 두 클래스를 하나의 클래스로 통합하였습니다.  또한, 두 클래스를 통합한 이후, 통합된 클래스 내에서 생성자의 중복 코드를 메소드화 하였는데, 이에 대해서는 ‘같은 클래스 내에 존재하는 중복코드 – 메소드화’에서 다루고 있습니다. |

* 1. 서로 다른 클래스에 존재하는 중복코드 – 클래스 상속화

|  |  |  |  |
| --- | --- | --- | --- |
| 스크린샷이(가) 표시된 사진  자동 생성된 설명 | | 스크린샷, 지도이(가) 표시된 사진  자동 생성된 설명 | |
| 개선 전 | | | |
|  | | | |
| 스크린샷, 지도이(가) 표시된 사진  자동 생성된 설명 | 스크린샷, 지도이(가) 표시된 사진  자동 생성된 설명 | | 스크린샷, 지도이(가) 표시된 사진  자동 생성된 설명 |
| 개선 후 | | | |
| 설명 | | | |
| MultiGameWindow는 GameWindow클래스에서 두명의 사용자가 게임을 플레이할 수 있도록 하는 필드/알고리즘을 추가하였기 때문에, 다른 클래스 내의 중복 코드가 존재한다는 공통점이 있지만, minefield/multiminefield의 경우와는 달리 상속 구조를 통해 필요하는 부분을 추가하는 방법을 사용하였습니다.  이를 위해 기존 GameWindow에서 private로 선언되어있던 변수/메소드들을 protected로 수정하고, 1인용 모드인 GW\_1playerMode, 2인용 모드인 GW\_2playerMode 라는 클래스를 각각 생성하여 GameWindow를 상속받도록 하였습니다. | | | |

* 1. GameWindow 클래스의 복기 기능 - 클래스화 및 불필요하게 나누어진 메소드 통합
     1. 클래스화

|  |  |
| --- | --- |
|  |  |
| 개선 전 | |
|  | 지도, 텍스트이(가) 표시된 사진  자동 생성된 설명 |
| 개선 후 | |

|  |  |  |
| --- | --- | --- |
|  | |  |
| 개선 전 | 개선 후 | |
| 설명 | | |
| 변경 전 , GameWindow 클래스는 여러 기능이 혼합되어 클래스가 지나치게 방대했다.  GameWindow클래스에서 복기 기능을 TheRecordOfAGme클래스로 분리함으로써  위의 그림과 같이 GameWindow의 ELOC(Estimated Lines Of Code)와 Fat이 크게 줄은 것을 볼 수 있다. | | |

|  |
| --- |
| 스크린샷이(가) 표시된 사진  자동 생성된 설명 |
| 개선 전 |
| 스크린샷이(가) 표시된 사진  자동 생성된 설명 |
| 개선 후 |
| 설명 |
| 복기를 구현하기 위한 메소드 중 디버깅을 위해 불필요하게 나누어져 있던 메소드가 존재했다.  개선 전과 개선 후를 나타낸 위의 두 그림을 비교해 보면 불필요하게 나누어져 있던 메소드가 하나로 합쳐진 것을 볼 수 있다. |

* + 1. 불필요하게 나누어진 메소드 통합
  1. Timer 클래스 수정 – 분리되어 있는 요소 통합 및 중복 코드 해결
     1. 분리되어 있는 요소 통합

|  |
| --- |
|  |
| 개선 전 |
|  |
| 개선 후 |
| 설명 |
| 개선 전 코드에서는 위와 같이 타이머가 다른 창에 존재해 분리되어 있었다.  또한, 이용하지 않는 버튼이 존재해 프로그램에 적합하지 않았다.  따라서, 이용하지 않는 코드를 삭제해 프로그램을 개선하였다. |

* + 1. 중복 코드 해결

|  |  |
| --- | --- |
|  |  |
| 개선 전 | |
|  | |
| 개선 후 | |
| 설명 | |
| 개선 전 코드에서는 첫 화면에서 각각의 모드를 선택할 때, 타이머를 선택하게끔 하여 코드가 중복되었다. 개선 후에는 Easy/Medium/Hard모드에 상관없이GW\_1playerMode,GW\_2playerMode에서 게임 윈도우를 설정할 때 한번에 호출할 수 있도록 하여 코드 중복을 해결하였다 | |

* 1. 불필요한 main 제거 및 사용하고 있는 main 클래스화

|  |
| --- |
|  |
| 개선 전 |
|  |
| 개선 후 |
| 설명 |
| main함수가 2개여서 실행 시에 어떤 main 함수를 이용하여야 하는지 구분이 어려우므로, 사용하지 않는 main 함수를 삭제하고, 이용하는 main함수는 클래스화 했다. |

* 1. MVC Pattern Refactoring
     1. MVC Pattern

MVC Pattern : MVC(Model-View-Controller) 아키텍처 패턴은 애플리케이션을 모델, 뷰, 컨트롤러라는 세 가지 주요 구성 요소로 구분합니다. 이 패턴은 문제를 분리하는 데 도움이 됩니다.

MVC(Model-View-Controller) 아키텍처 패턴은 응용 프로그램을 모델, 뷰, 컨트롤러라는 세 가지 주요 구송 요소로 구분합니다.

모델(예: 데이터 정보)은 순수 애플리케이션 데이터만 포함합니다

뷰(예: 프리젠테이션 정보)는 모델의 데이터를 사용자에게 전달하는 역할을 합니다. 뷰는 모델의 데이터에 접근하는 방법을 알고 있지만, 이 데이터가 무엇을 의미하는지 또는 사용자가 이를 조작하는 방법을 알지 못합니다.

마지막으로, 컨트롤러(예: 제어 정보)는 뷰와 모델 사이에 존재합니다. 뷰(또는 다른 외부 소스)에 의해 발생한 이벤트의 리스너 기능을 수행합니다

오늘날 대부분의 어플리케이션은 이 패턴을 따르고 있습니다. 이는 응용 프로그램의 구조를 더 단순하게 할 뿐만 아니라, 코드 재사용을 가능하게 합니다.

* + 1. Minefiled/MinesFinder에서의 MVC 구현

|  |
| --- |
| 그림입니다. 원본 그림의 이름: CLP000044a00017.bmp 원본 그림의 크기: 가로 642pixel, 세로 276pixel |
| 개선 전 |
| 그림입니다. 원본 그림의 이름: CLP000044a0001a.bmp 원본 그림의 크기: 가로 472pixel, 세로 157pixel |
| 개선 후 |
| 설명 |
| 기존의 Minefield는 view와 model이 혼재된 클래스였습니다.  이러한 문제점을 해결하기 위해 view와 관련된 클래스인 Minesfinder클래스로 view를 정의한 부분을 이동시켜 Minefield를 model과 관련된 클래스로 만들었습니다. |

* 1. Error 처리

|  |
| --- |
|  |
| 개선 전 |
|  |
| 개선 후 |
| 설명 |
| 기존의 코드에는 숫자 입력을 받으려고 할 때 입력받은 문자열에 공백이나 숫자가 아닌 문자가 포함되어 있을 때에 오류가 발생하고, 그 오류가 처리되지 않았습니다.  String 타입을 숫자 타입 중 하나로 변환하려고 할 때, 문자열이 숫자 타입에 적합하지 않은 형식을 가지고 있을 때, NumberFormatException이 발생하는 것을 이용하여, 예외가 발생하였을 때 스스로를 다시 호출하는 메소드를 만들었습니다. |

* + 1. 숫자 입력에 숫자가 아닌 문자, 공백이 입력되었을 때
    2. 이름 입력이 5글자 이상일 때

|  |
| --- |
|  |
| 개선 후 |
| 설명 |
| 이름 입력을 받을 때, 이름이 지나치게 길면 Label 등에 나타내기 곤란하기 때문에 이름을 입력할 때 글자수에 제한을 주고자 하였습니다.  입력 문자열의 길이에 제한을 주는 메소드는 기존 메소드가 있는 함수에 존재하는 것이 부적절하다고 생각하여 JTextFieldLimit 라는 별도의 클래스를 생성하여, 해당 클래스에 정의하였습니다 |

* + 1. State가 covered가 아닌 버튼에 키보드 입력이 주어졌을 때(winner\_flag 증가)

|  |
| --- |
|  |
| 개선 전 |
|  |
| 개선 후 |
| 설명 |
| 개선 전에는 키보드 입력이 이미 밝혀진 버튼에 주어졌을 때도 winner\_flag가 증가하여 2인용 모드에서의 승자가 바뀌는 일이 생기기도 했습니다.  키보드 입력이 주어진 버튼이State==9 즉, state == covered인 버튼일 때만 winner\_flag가 증가하도록 조건문을 설정하여 해당 문제를 해결하였습니다. |

* + 1. 입력받은 지뢰의 수가 버튼의 수보다 많을 때

|  |
| --- |
|  |
| 개선 후 |
| 설명 |
| 기존 코드에는 지뢰의 개수가 버튼의 개수보다 클 때를 명시적으로 규정하지않았는데,  기존 코드에 존재하던 지뢰의 개수가 음수일 때 실행하는 조건문을 이용하여 지뢰의 개수가 버튼 개수보다 더 크게 설정하는 문제가 발생하였을 때도 해당 문제를 해결할 수 있도록 개선하였습니다. |

* 1. 코드의 Relocation

|  |  |
| --- | --- |
|  |  |
| 개선 전 | |
|  |  |
| 개선 후 | |
| 설명 | |
| 기존 코드에서는 여러 조건마다 변수를 설정하여 특정 조건이 만족되었을 때 특정 위치에서 원하는 동작을 구현하기 위해 한 동작과 관련된 코드들이 여러 클래스의 여러 메소드에 분산되어 있었지만, 이번 개선으로 인하여 두 개의 클래스에서 두 개의 메소드를 통하여 원하는 기능을 구현할 수 있게 되었다.  이로 인해 결합도가 줄었고, 유지보수성이 좋아졌다. | |

1. 2차 개선 아이디어 구현

실행 화면 캡쳐

|  |
| --- |
|  |
| EasyMode 첫 실행 화면  상단 패널의 버튼과 label이 잔여 시간과 아이템 개수를 표시하고 있다. |
|  |
| 지뢰가 있는 버튼을 눌렀을 때 방패가 실행되는 모습 |

|  |
| --- |
|  |
| 힌트를 사용하였을 때 – 힌트를 사용한 버튼에 지뢰가 있을 때 Marked 상태가 된다. |
|  |
| 힌트를 사용하였을 때 – 힌트를 사용한 버튼에 지뢰가 없을 때 Empty 상태가 된다. |

|  |
| --- |
|  |
| 상단의 아이템에 대한 정보를 눌렀을 때- 아이템에 대한 정보를 알려준다. |

* 1. 방패 아이템 추가

|  |
| --- |
| 방패 아이템이 남아있을 때 지뢰가 있는 버튼에 키보드나 마우스 입력이 이루어지면 방패를 이용했다는 창이 뜨며 버튼이 COVERED상태로 유지된다.  방패 아이템은 매 게임마다 3개가 주어지며, 상단의 패널을 통해 남은 방패 아이템의 개수를 알 수 있다. 또한, 방패는 1인용 모드에서만 사용할 수 있다. |
| 버튼에 입력이 이루어졌을 때, 해당 칸에 지뢰가 있고 방패 아이템의 개수가 1개 이상이라면 남아있는 방패 아이템의 개수를 감소시킨 후, 방패가 이용되었다는 창을 띄운다.  그 후, 감소시킨 방패 아이템의 개수를 상단 패널에 있는 label에 반영시키고, RecGame.deleteMapClick()메소드를 실행시킨다. 이는 복기를 위해 ArrayList에 저장한 좌표값을 제거하는 메소드이다. |

* 1. 힌트 아이템 추가

|  |
| --- |
| 버튼 배열의 특정 칸에 포커스를 둔 후 ‘H’버튼을 누르면 힌트 아이템을 사용할 수 있다.  힌트 버튼은 해당 아이템을 이용한 버튼이 지뢰를 가지고 있다면 버튼의 상태를 ‘MARKED’로 변경해 주고, 그렇지 않은 경우에는 해당 칸에 revealGrid(int X, int Y) 메소드를 적용하여 해당 칸의 지뢰 유무를 판별해 주는 아이템이다.  힌트 아이템은 매 게임마다 3개가 주어지며, 상단의 패널을 통해 남은 힌트 아이템의 개수를 알 수 있다. 또한, 힌트는 1인용 모드에서만 사용할 수 있다. |
| 키보드의 키 값이 입력되면 현재 포커스가 있는 버튼에 useHint(int x, int y)메소드를 적용시킨다.    힌트가 1개 이상일 때, 힌트를 사용할 수 있다.  만약 해당 칸이 지뢰를 가지고 있다면 해당 칸을 MARKED 상태로 변경하고, updateButtonState() 메소드를 실행한다. 해당 칸이 지뢰를 가지고 있지 않을 경우,  revealGrid(x,y)메소드를 실행하고, updateButtonState() 메소드를 실행한다.  힌트를 사용한 후에는 남은 힌트의 개수를 감소시키고 상단의 label에 숫자를 반영시킨다. 만약 힌트의 개수가 1개 이상이 아니라면, 힌트를 사용할 수 없다는 창이 뜬다. |

1. 느낀 점

설경원:

소스코드분석 2차과제는 1차과제에 비해 좀 수월했던 것 같습니다. 1차과제때는 아예 모르는코드를 처음부터 분석하고 개선사항 추가를 위해 어떤 필드나 메소드, 클래스를 건드려야할지 막막했지만 이번에는 1차과제때 충분히 분석하고 개선사항을 추가한 결과 필요한 메소드와 클래스의 위치를 금방알 수 있어 1차과제에 비해선 수월했다고 생각합니다. 그래도 새로운 개념인 리팩토링을 처음해보는거라 어려웠고 가장 많이들었던 생각은 ‘이게 리팩토링이 될수 있을까?’라는 생각이였습니다. 1학기때 자바를 처음배우면서 서서히 잊혀져갔던 try-catch문이나 thread도 실제로 사용해보는건 처음이였고, 이렇게 긴 코드를 작성하고 수정한 것도 처음이라 항상 처음이 어렵듯이 저도 무척 어렵고 힘들었지만 그로인해 얻어간것도 많은 수업이었다고 생각합니다. 몇시간씩 책상에 앉아 코딩을 하다보니 평소 하던 게임들에 버그가 많다고 욕했던 저를 되돌아보게 되었고, 개발자들은 정말 대단하다는 생각과 저도 나중엔 고급 개발자들처럼 코드를 막힘없이 작성해보고 싶다는 욕심이 생겼던 것 같습니다. 방학때도 교수님이 말씀하셨던 것처럼 만들어보고 싶은 게임을 분석하고 개선사항 추가를 통해 실력을 기르려고 합니다. 힘들었지만 그만큼 남는게 많았던 수업이었던 것 만큼 이 방법이 가장 실력을 기르기엔 적합한 방법이라는 생각이 들었기 때문입니다. 한 학기동안 열심히 가르쳐주셔서 감사합니다.

한별: 소프트웨어공학과 컴퓨터공학의 차이는 개발의 질을 개선하는 방법을 배우는 것이라는 말을 여러 번 들었습니다.

이번 과제는 저번 1차 과제보다 코드와 단축키에 익숙해져서 특정 기능을 구현하는 과정에서는 속도가 붙었지만, 코드의 질을 개선하는 과정은 생각보다 어려웠습니다.

이번 과제를 하면서 여러 가지 코드의 질을 향상시킬 수 있는 방법을 알게 되었습니다.

소스코드 분석 수업, 과제를 통해 소프트웨어 공학이 무엇인지를 조금이나마 경험해 보게 된 것 같습니다. 한 학기 동안 열심히 가르쳐주셔서 감사합니다.