|  |
| --- |
| 2019-2 소스코드 분석 |
| 8조 보고서 |
| 2차 과제 |

|  |
| --- |
| 201613989 김진산 201818207 김현수  2019-12-9 |

목차

[1. 개선 아이디어 정리 및 난이도 4](#_Toc26783783)

[1.1. 1차 기능추가 4](#_Toc26783784)

[1.1.1. 사용자 설정 게임 (1점) 4](#_Toc26783785)

[1.1.2. 효과음 추가 (1점) 4](#_Toc26783786)

[1.1.3. 클릭방법의 추가(1점) 4](#_Toc26783787)

[1.1.4. 네트워크를 통한 2인용 게임 (7.5점) 4](#_Toc26783788)

[1.2. 2차 기능추가 4](#_Toc26783789)

[1.2.1. 목숨 기능 (1점) 4](#_Toc26783790)

[1.2.2. Challenge 모드 (1점) 5](#_Toc26783791)

[2. 분석 및 개선 다이어그램 6](#_Toc26783792)

[2.1. 기존 코드 분석 클래스 다이어그램 6](#_Toc26783793)

[2.2. 1차 개선 후 분석 클래스 다이어그램 7](#_Toc26783794)

[2.3. 2차 Refactoring 후 클래스 다이어그램 8](#_Toc26783795)

[3. 시퀀스 다이어그램 9](#_Toc26783796)

[4. 구현 방법 설명 및 CodeSnippet 10](#_Toc26783797)

[4.1. 실행 화면 10](#_Toc26783798)

[4.1.1. 메인 화면 10](#_Toc26783799)

[4.1.2. 기본 게임 화면 11](#_Toc26783800)

[4.1.3. 사용자 설정게임 화면 11](#_Toc26783801)

[4.1.4. Multi 게임 화면 12](#_Toc26783802)

[4.1.5. 서버 화면 12](#_Toc26783803)

[4.2. 1차 추가 기능 13](#_Toc26783804)

[4.2.1. 사용자 설정 모드 13](#_Toc26783805)

[4.2.2. 효과음 추가 13](#_Toc26783806)

[4.2.3. 클릭 방법 추가 14](#_Toc26783807)

[4.2.4. 네트워크 2인용 게임 15](#_Toc26783808)

[4.3. 2차 추가 기능 17](#_Toc26783809)

[4.3.1. 목숨 기능 17](#_Toc26783810)

[4.3.2. Challenge 모드 18](#_Toc26783811)

[5. Refactoring 20](#_Toc26783812)

[5.1. Stan4J 비교 20](#_Toc26783813)

[5.1.1. Refactoring 전 오염 20](#_Toc26783814)

[5.1.2. Refactoring 후 오염 20](#_Toc26783815)

[5.1.3. Refactoring 전 Distance 21](#_Toc26783816)

[5.1.4. Refactoring 후 Distance 21](#_Toc26783817)

[5.2. Sonar Lint 비교 22](#_Toc26783818)

[5.2.1. Refactoring 전 22](#_Toc26783819)

[5.2.2. Refactoring 후 22](#_Toc26783820)

[5.3. 변경사항 23](#_Toc26783821)

[5.3.1. 가시성, 이름 정리 23](#_Toc26783822)

[5.3.2. 불필요 메소드, 임포트 제거 23](#_Toc26783823)

[5.3.3. 불필요한 비교 변경 24](#_Toc26783824)

[5.3.4. Run class 생성 24](#_Toc26783825)

[5.3.5. MultiMode class 생성 25](#_Toc26783826)

[5.3.6. CustomMode class 생성 27](#_Toc26783827)

[5.3.7. ChallengeMode class 생성 28](#_Toc26783828)

[5.3.8. GameWindow 상속관계 생성 29](#_Toc26783829)

[5.3.9. 예외처리, server 콘솔 출력 메시지 log로 변경 32](#_Toc26783830)

[5.3.10. 소리 끄기 버튼 생성 33](#_Toc26783831)

[5.3.11. SoundEffect class 조정 33](#_Toc26783832)

[5.3.12. Sonar Lint tool 사용 34](#_Toc26783833)

[5.3.13. 주석 정리 35](#_Toc26783834)

1. 개선 아이디어 정리 및 난이도

1차 기능추가

* + 1. 사용자 설정 게임 (1점)

사용자에게 가로 칸, 세로 칸, 지뢰의 수를 각각 입력 받아 그에 맞게 게임 화면을 만든다. 사용자 설정 게임을 시작하면 가로 칸, 세로 칸, 지뢰의 수를 입력하는 창이 뜨고 그에 맞게 게임 화면을 만들어 플레이 할 수 있도록 한다. 가로 칸, 세로 칸, 지로의 수에 따라 기록이 기록은 저장하지 않는다.

* + 1. 효과음 추가 (1점)

BGM, 버튼 선택 시 효과음, 지뢰를 선택했을 경우의 효과음을 추가한다. BGM은 게임을 시작할 때 켜지고 종료 시 꺼진다. 효과음은 .wav 파일을 사용한다.

* + 1. 클릭방법의 추가(1점)

마우스 좌, 우 클릭을 동시에 하였을 경우 주변의 지뢰 개수와 사용자가 지뢰가 있을 것으로 판단하여 표시한 것의 개수를 비교하여 만일 같을 경우 사용자가 표시한 부분을 제외하고 주변 칸을 선택한다. 사용자가 지뢰의 위치를 잘못 추측한 경우에도 주변을 선택하기 때문에 지뢰가 터질 수 있다.

* + 1. 네트워크를 통한 2인용 게임 (7.5점)

서버와 클라이언트 클래스를 정의하고 이를 바탕으로 네트워크를 이용하여 2인용 게임을 만든다.

게임의 승리 조건은 상대방이 지뢰를 선택하거나 자신이 먼저 게임을 성공할 경우 이 2가지이다.

2차 기능추가

* + 1. 목숨 기능 (1점)

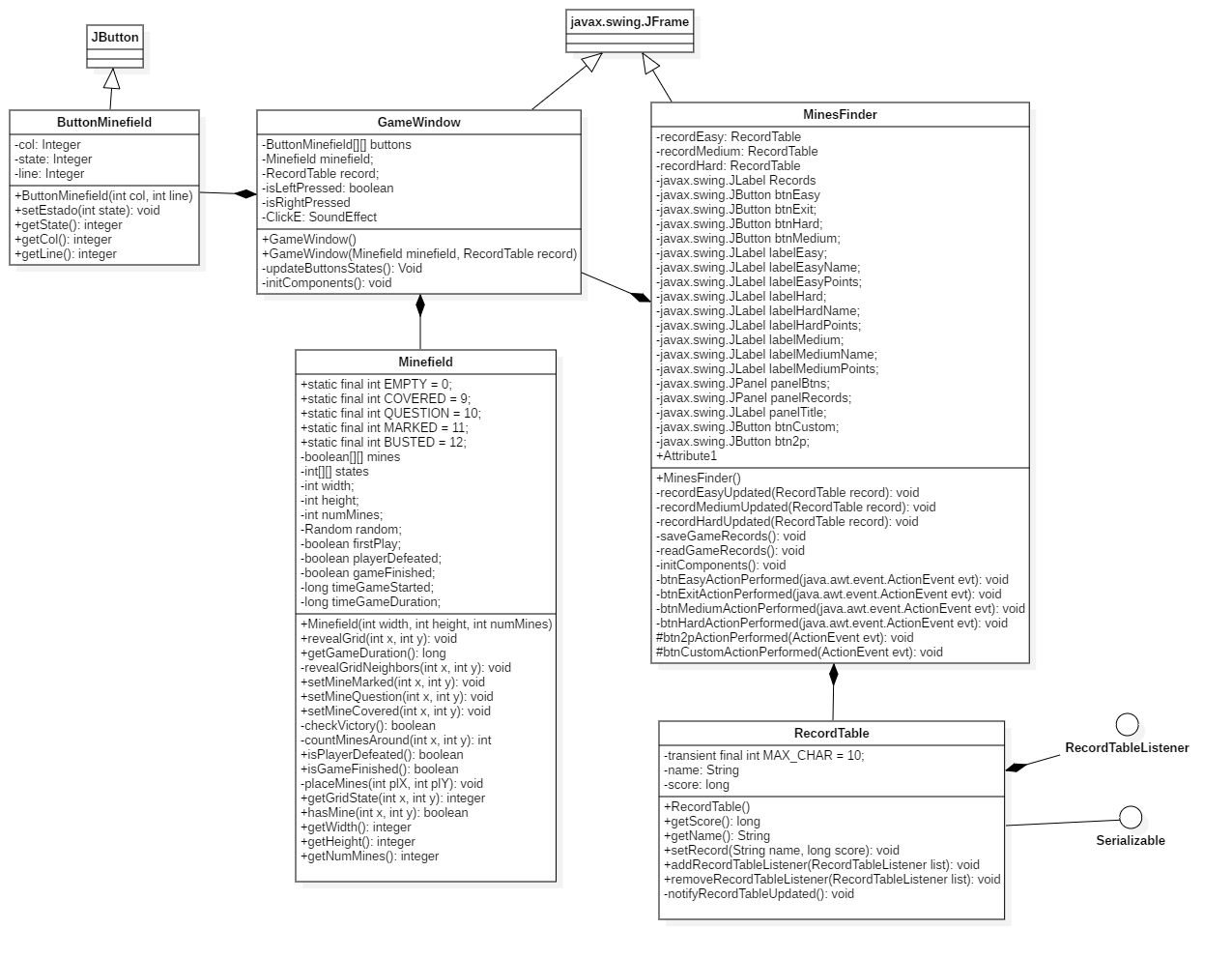
GameWindow에 목숨 속성을 추가해 지뢰를 밟았을 때 작동하는 메소드에서 목숨의 개수가 한 개씩 줄도록 설정했고 난이도에 따라 생성자에서 목숨의 개수를 초기화 한다. Easy는 목숨 1개 Medium은 목숨 2개 Hard는 목숨 3개를 설정했다. Challenge와 Multi는 1개, Custom은 입력하도록 설정했다.

* + 1. Challenge 모드 (1점)

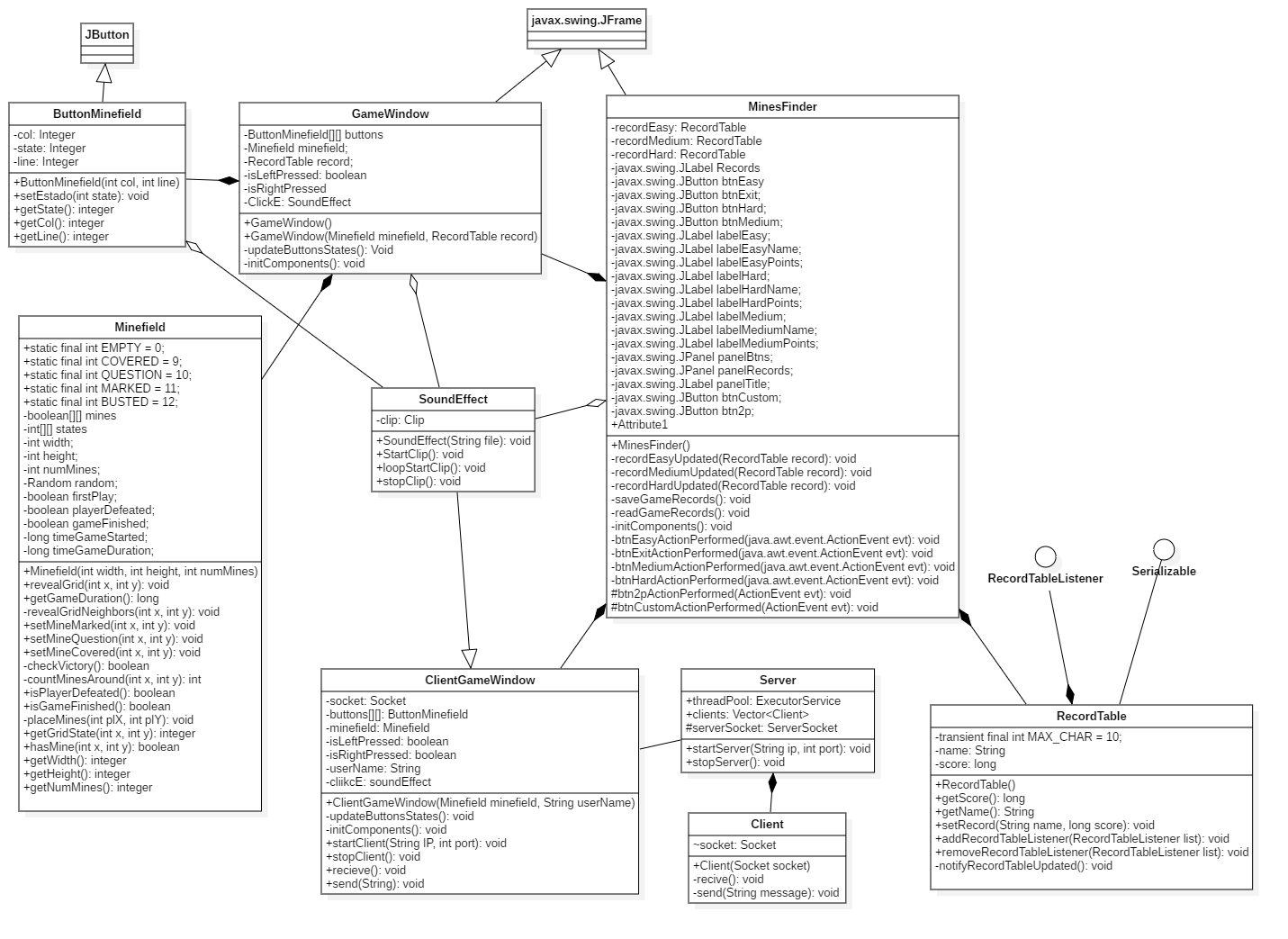
Challenge 모드는 클래스로 만들어 메인 화면에 버튼을 누르면 객체를 생성하는 방식으로 설계하였고 목숨 1개로 시작하며 죽지 않으면 계속 일정 개수의 버튼과 폭탄이 늘어나는 방식으로 설계하였다.

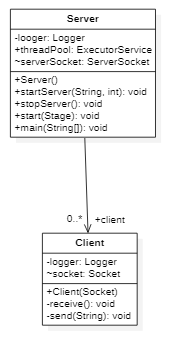
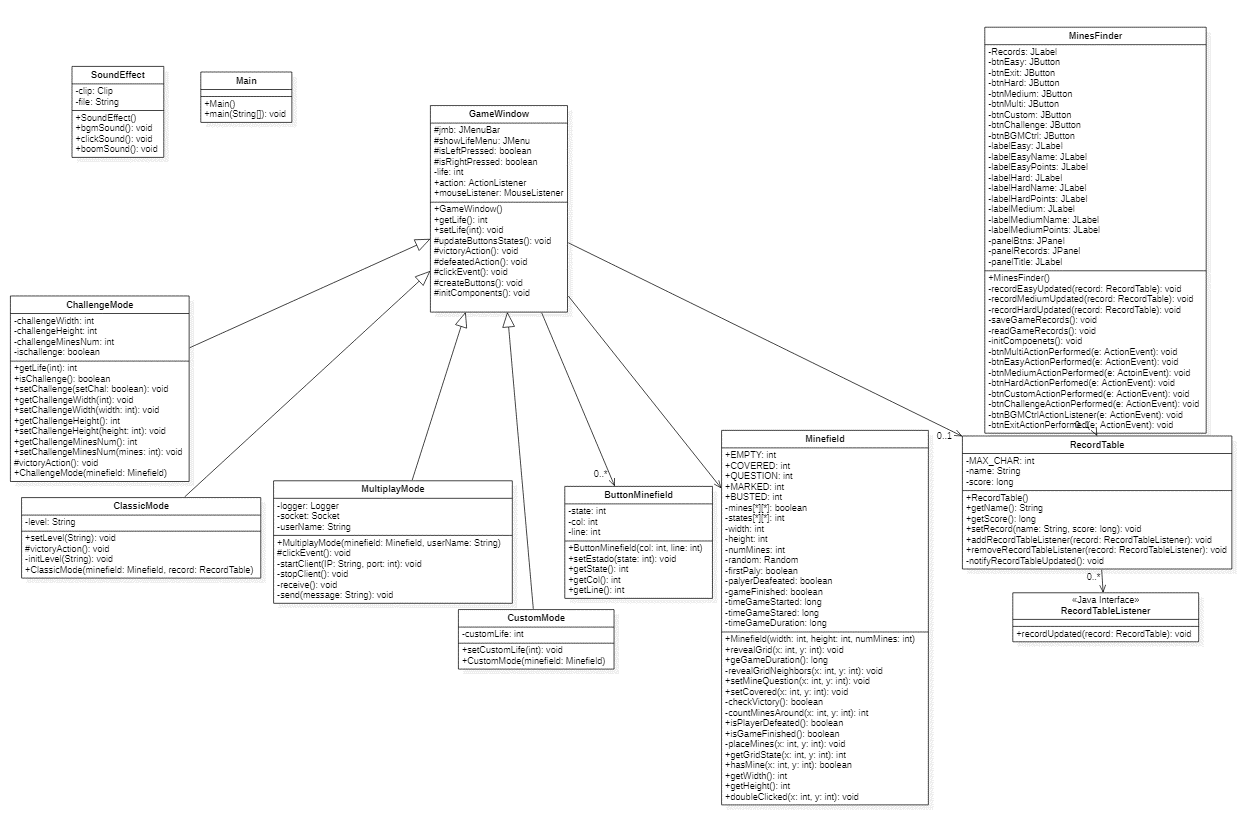
1. 분석 및 개선 다이어그램

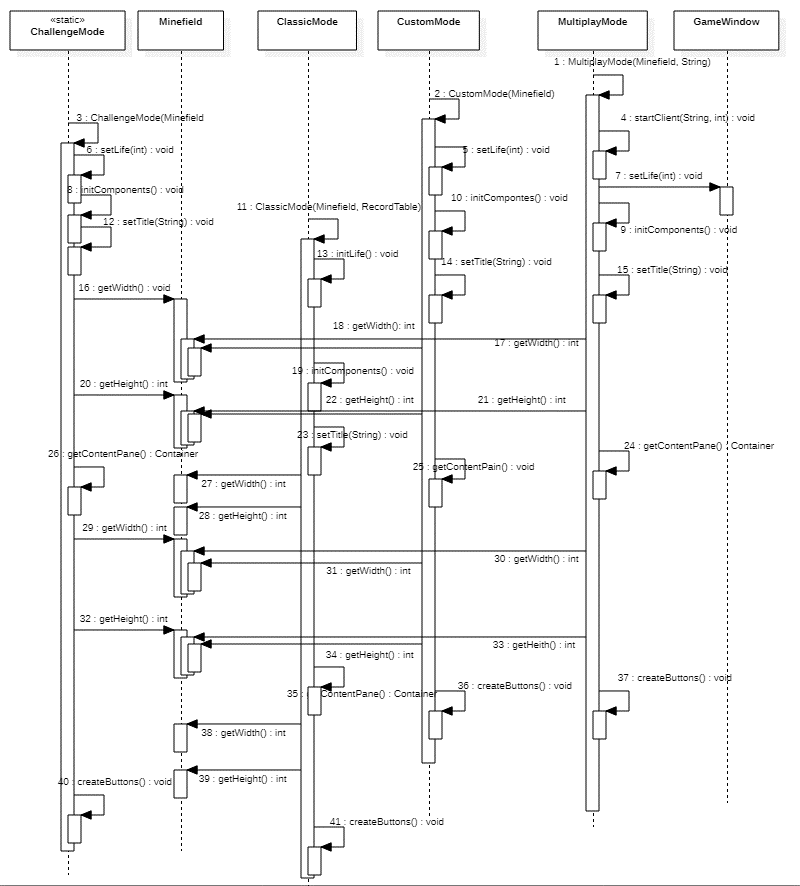
기존 코드 분석 클래스 다이어그램



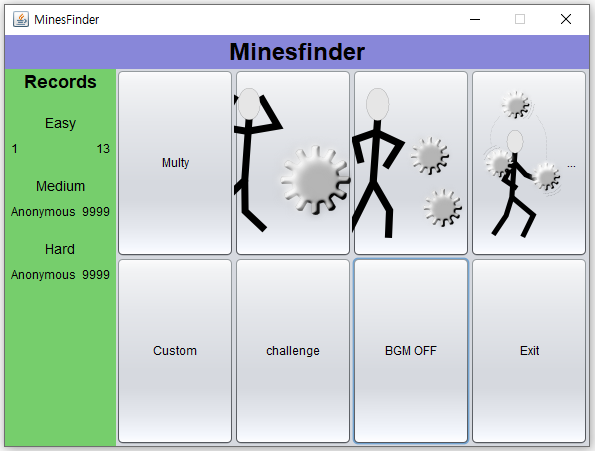
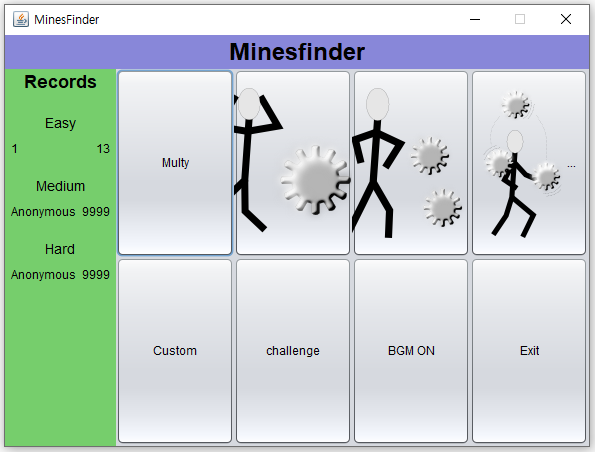
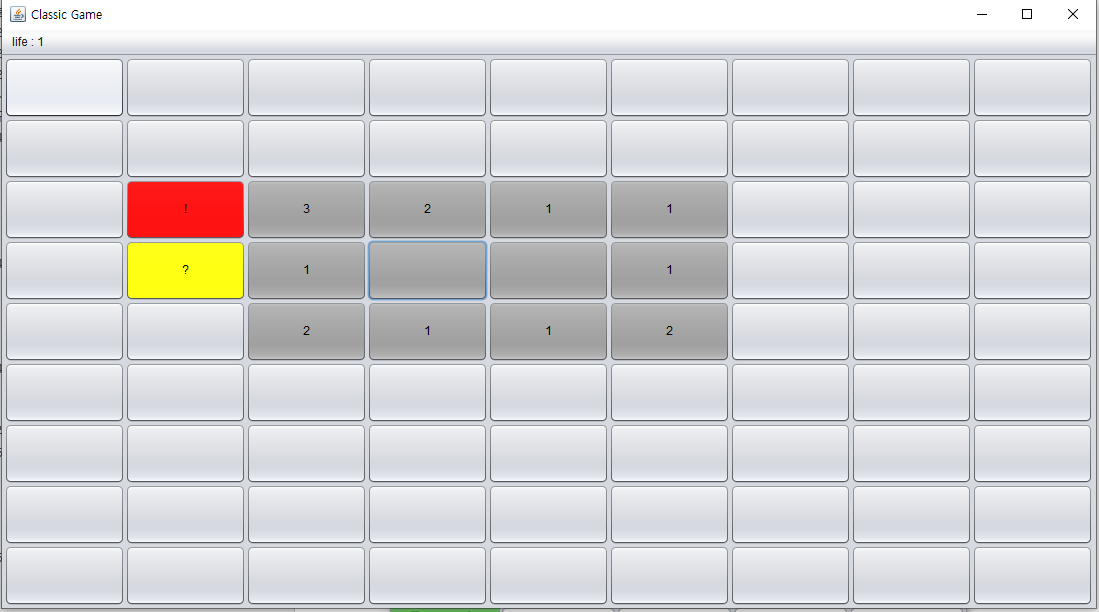
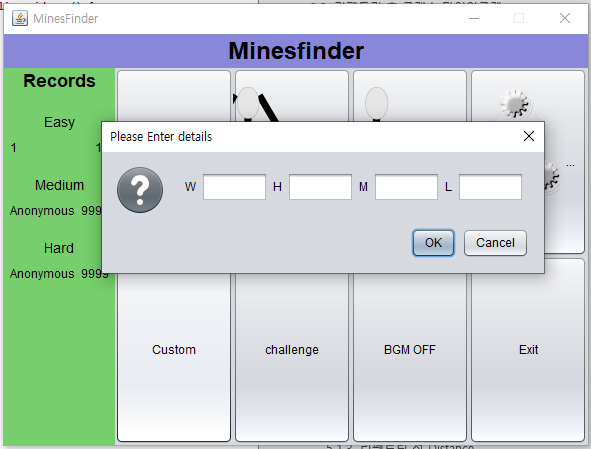
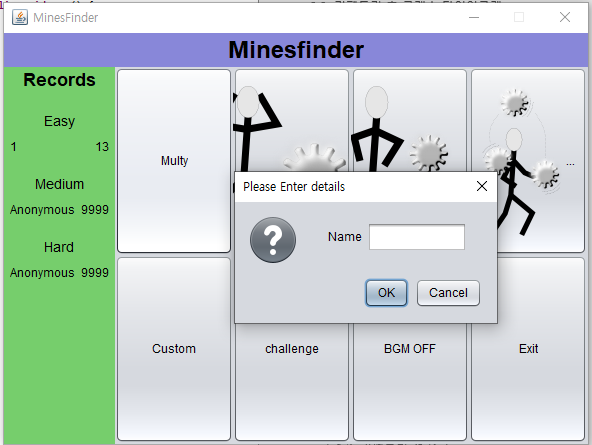
1차 개선 후 분석 클래스 다이어그램

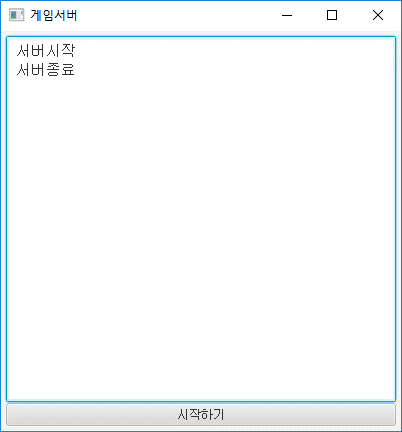


2차 Refactoring 후 클래스 다이어그램

1. 시퀀스 다이어그램
2. 구현 방법 설명 및 Code Snippet

실행 화면

* + 1. 메인 화면
    2. 기본 게임 화면
    3. 사용자 설정게임 화면
    4. Multi 게임 화면
    5. 서버 화면



1차 추가 기능

* + 1. 사용자 설정 모드

개선 설명

1. 타이틀이 Custom인 사용자 설정 게임 버튼을 만든다.
2. 사용자 설정 게임은 기록을 하지 않기 때문에 새로 생성자를 만든다.
3. 게임 창이 만들어 지기 전 가로 칸, 세로 칸, 지뢰 수를 입력한다.
4. 3에서 받은 값으로 새로 만든 생성자를 실행한다.

코드

|  |
| --- |
| **protected** **void** btnCustomActionPerformed(ActionEvent evt) {  // **TODO** Auto-generated method stub  JTextField customWidth = **new** JTextField(5);  JTextField customHeight = **new** JTextField(5);  JTextField customMinesNum = **new** JTextField(5);    JPanel customPanel = **new** JPanel();  customPanel.add(**new** JLabel("W"));  customPanel.add(customWidth);  customPanel.add(**new** JLabel("H"));  customPanel.add(customHeight);  customPanel.add(**new** JLabel("M"));  customPanel.add(customMinesNum);    **int** result = JOptionPane.*showConfirmDialog*(**null**, customPanel,"Please Enter details", JOptionPane.***OK\_CANCEL\_OPTION***);  **if**(result == JOptionPane.***OK\_OPTION***) {  GameWindow gameWindow = **new** GameWindow(**new** Minefield(Integer.*parseInt*(customWidth.getText()),Integer.*parseInt*(customHeight.getText()),Integer.*parseInt*(customMinesNum.getText())));  gameWindow.setVisible(**true**);  }  } |

* + 1. 효과음 추가

- 개선 설명

1. AudioInputStream으로 .wav파일을 읽는다.
   1. 받는 경로를 절대 경로로 지정할 시 저장위치가 지정된 것이 아니기 때문에 사용자마다 음원의 위치가 달라진다. 그래서 상대 경로로 지정을 해 어떤 경로에 다운받았든지 똑같이 접근할 수 있도록 했다.
2. Clip을 이용하여 효과음을 재생한다.

- 코드

|  |
| --- |
| **public** **class** SoundEffect {  **private** Clip clip;    **public** SoundEffect(String file)  {  **try** {  File base = **new** File(file);  String basePath = base.getAbsolutePath();  File audio = **new** File(basePath);    AudioInputStream ais = AudioSystem.*getAudioInputStream*(audio);  clip = AudioSystem.*getClip*();  clip.open(ais);  }  **catch**(Exception e) {  e.printStackTrace();  }    }  **public** **void** startClip() {  clip.start();  }  **public** **void** loopStartClip() {  clip.start();  clip.loop(clip.***LOOP\_CONTINUOUSLY***);  }  **public** **void** stopClip() {  clip.stop();  }    } |

* + 1. 클릭 방법 추가

- 개선 설명

1. 양쪽클릭 리스너를 만든다.
2. 양쪽클릭 시 동작하는 메소드를 만든다.
3. 메소드 조건 : 버튼의 state 값과 주변의 Marked 된 버튼의 개수가 같으면 근처 버튼에 revealGrid메소드를 실행한다.

- 코드

|  |
| --- |
| **public** **void** doubleCliked(**int** x, **int** y) {  **int** sum = 0;  **for** (**int** col = Math.*max*(0, x - 1); col < Math.*min*(width, x + 2); col++) {  **for** (**int** line = Math.*max*(0, y - 1); line < Math.*min*(height, y + 2); line++) {  **if**(states[col][line] == Minefield.***MARKED***)  sum++;  }  }  **if** (sum == states[x][y]) {  **for** (**int** col = Math.*max*(0, x - 1); col < Math.*min*(width, x + 2); col++) {  **for** (**int** line = Math.*max*(0, y - 1); line < Math.*min*(height, y + 2); line++) {  revealGrid(col, line);  }  }  }  sum = 0;  } |

* + 1. 네트워크 2인용 게임

- 개선 설명

1. IP주소와 port값을 입력 받아 통신할 수 있는 서버 클래스를 만든다.
2. MinesFinder에서 서버를 켜고, 끄는 버튼을 만들어 서버를 제어한다.
3. 시작 시 이름을 받고 네트워크 게임을 동작시키는 클래스를 만든다.
4. 게임이 끝날 경우 알림 창을 통해 승, 패를 알리고 게임판에서 나가진다.

- 코드

|  |
| --- |
| **public** **void** startServer(String IP, **int** port) {  **try** {  serverSocket = **new** ServerSocket();  serverSocket.bind(**new** InetSocketAddress(IP, port));  }**catch**(Exception e) {  e.printStackTrace();  **if**(!serverSocket.isClosed()) {  stopServer();  }  **return**;  }    //클라이언트 접속까지 기다리는 쓰레드  Runnable thread = **new** Runnable() {  @Override  **public** **void** run() {  **while**(**true**) {  **try** {  Socket socket = serverSocket.accept();  *clients*.add(**new** Client(socket));  System.***out***.println("클라이언트 접속" + socket.getRemoteSocketAddress() + " : " + Thread.*currentThread*().getName());  }**catch**(Exception e) {  **if**(!serverSocket.isClosed()) {  stopServer();  }  **break**;  }  }    }  };  *threadPool* = Executors.*newCachedThreadPool*();  *threadPool*.submit(thread);  } |

|  |
| --- |
| **public** **void** startClient(String IP, **int** port) {  Thread thread = **new** Thread() {  **public** **void** run() {  **try** {  socket = **new** Socket(IP, port);  receive();  } **catch** (Exception e) {  **if** (!socket.isClosed()) {  stopClient();  System.***out***.println("서버 접속 실패");  Platform.*exit*();  }  }  }  };  thread.start();  } |

|  |
| --- |
| **public** **void** stopClient() {  **try** {  **if**(socket!=**null** && !socket.isClosed()) {  socket.close();  }  }**catch**(Exception e) {  e.printStackTrace();  }  } |

|  |
| --- |
| **public** **void** receive() {  **while** (**true**) {  **try** {  InputStream in = socket.getInputStream();  **byte**[] buffer = **new** **byte**[512];  **int** length = in.read(buffer);  **if** (length == -1)  **throw** **new** IOException();  String message = **new** String(buffer, 0, length, "UTF-8");  String[] strarr = message.split(" ");  **if**(strarr[0].equals(userName)) {  **if**(strarr[1].equals("finish")) {  JOptionPane.*showMessageDialog*(**null**, userName + " win",  "Result!", JOptionPane.***INFORMATION\_MESSAGE***);  } **else** {  JOptionPane.*showMessageDialog*(**null**, userName + " lose",  "Result!", JOptionPane.***INFORMATION\_MESSAGE***);  }  }  **else**{  **if**(strarr[1].equals("finish")) {  JOptionPane.*showMessageDialog*(**null**, userName + " lose",  "Result!", JOptionPane.***INFORMATION\_MESSAGE***);  }  **else** {  JOptionPane.*showMessageDialog*(**null**, userName + " win",  "Result!", JOptionPane.***INFORMATION\_MESSAGE***);  }  }  setVisible(**false**);  stopClient();  } **catch** (Exception e) {  stopClient();  **break**;  }  }  } |

|  |
| --- |
| **public** **void** send(String message) {  Thread thread = **new** Thread() {  **public** **void** run() {  **try** {  OutputStream out = socket.getOutputStream();  **byte**[] buffer = message.getBytes("UTF-8");  out.write(buffer);  out.flush();  }**catch**(Exception e) {  stopClient();  }  }  };  thread.start();  } |

2차 추가 기능

* + 1. 목숨 기능

개선설명

1. 지뢰를 찾아도 바로 죽지 않도록 목숨을 추가한다.

2. 목숨이 1에서 0이 될 경우 게임을 패배한다.

3. 게임의 난이도에 따라 Easy : 1, Nomal : 2, Hard : 3으로 설정한다. Custom은 목숨 수를 입력하여 설정한다. 나머지 모드는 1로 설정한다.

코드

|  |
| --- |
| **protected** JMenuBar jmb = **new** JMenuBar();  **protected** JMenu showLifeMenu = **new** JMenu();  **private** **static** **int** *life* = 1;  **public** **static** **int** getLife() {  **return** *life*;  }  **private** **static** String *level* = "";  **public** **static** **void** setLevel(String setlevel) {  *level* = setlevel;  }    **private** **void** initLife() {  **if**(*level*.equals("Easy")) {  *setLife*(1);  }  **else** **if** (*level*.equals("Nomal")){  *setLife*(2);  }  **else** **if**(*level*.equals("Hard")) {  *setLife*(3);  }  }  **private** **static** **int** *customLife*;    **public** **static** **void** setCustomLife(**int** life) {  *customLife* = life;  } |

* + 1. Challenge 모드

개선설명

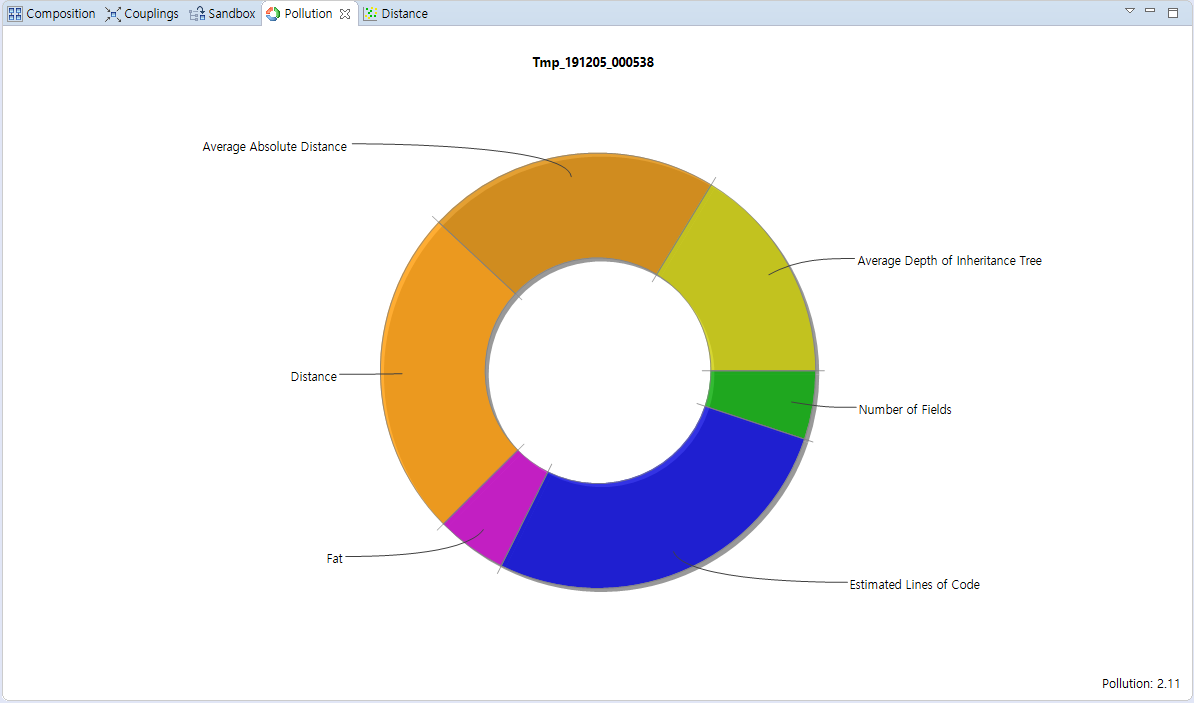
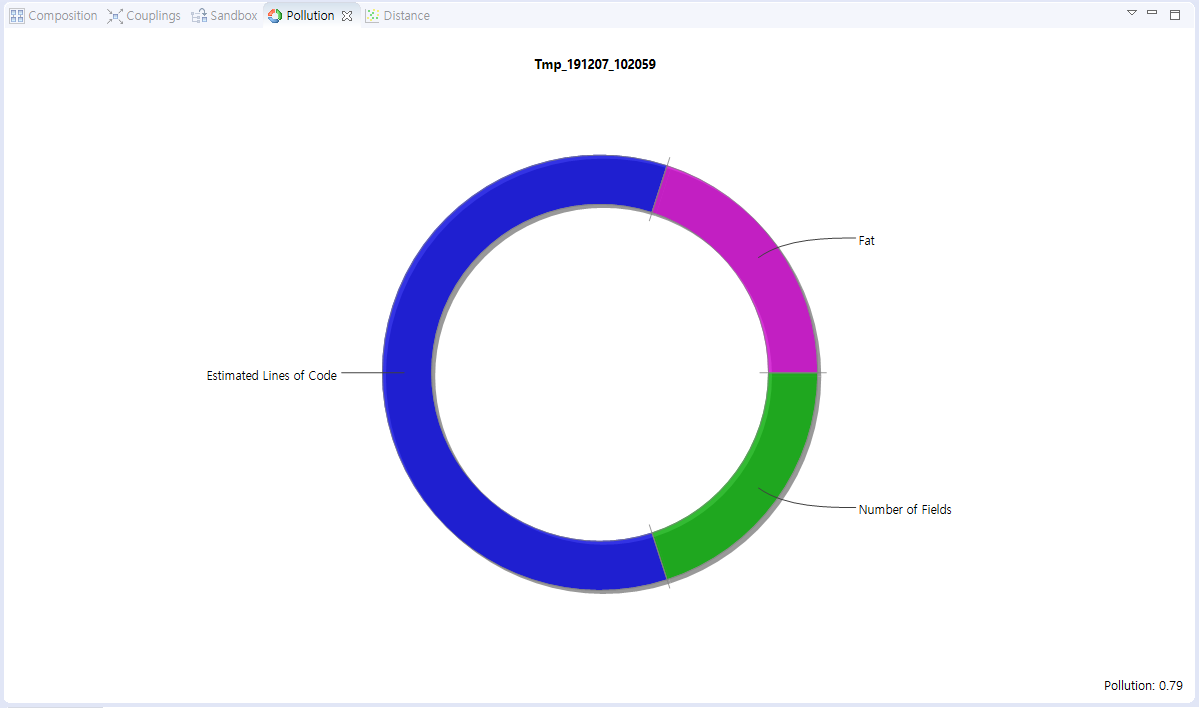
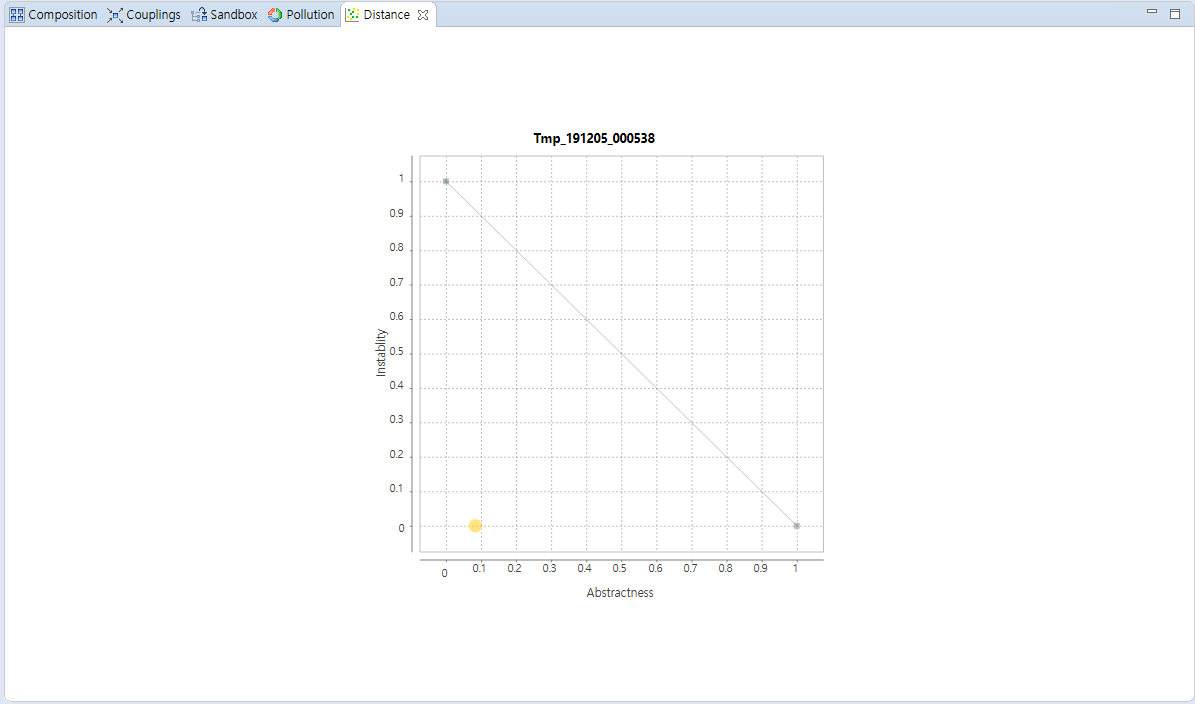
1. 게임이 승리로 끝날 경우에 다음 게임이 이어지도록 한다. 패배 시 게임은 끝난다.

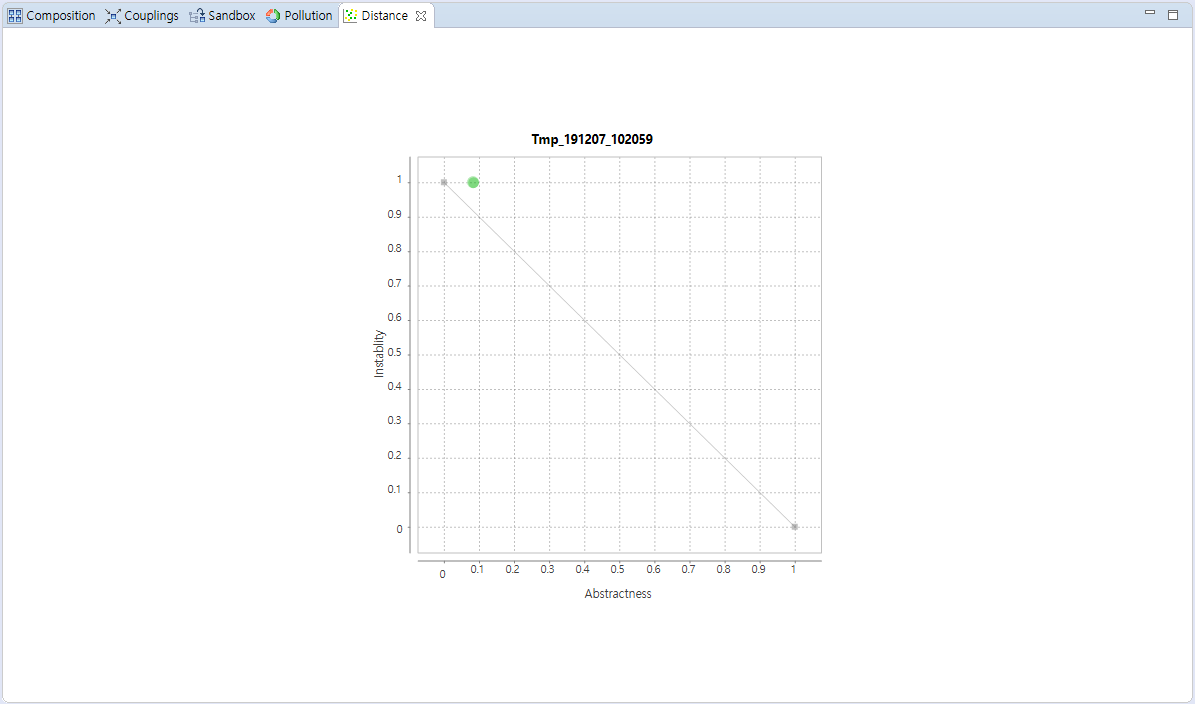
2. 게임판과 지뢰 수는 일정하게 증가한다.

코드

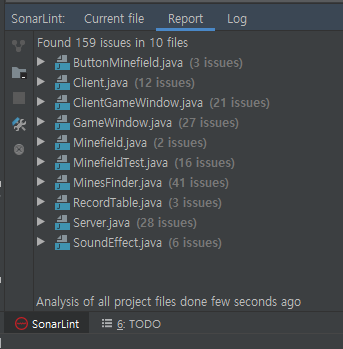
|  |
| --- |
| **public** **class** ChallengeMode **extends** GameWindow {  **private** **static** **int** *challengeWidth* = 0;  **private** **static** **int** *challegneHeight* = 0;  **private** **static** **int** *challengeMinesNum* = 0;  **private** **static** **boolean** *ischallenge* = **false**;  **public** **int** getLife(**int** life){  **return** life;  }  **public** **static** **boolean** isChallenge() {  **return** *ischallenge*;  }  **public** **static** **void** setChallenge(**boolean** setChal) {  *ischallenge* = setChal;  }  **public** **static** **int** getChallengeWidth() {  **return** *challengeWidth*;  }  **public** **static** **void** setChallengeWidth(**int** width) {  *challengeWidth* = width;  }  **public** **static** **int** getChallengeHeight() {  **return** *challegneHeight*;  }  **public** **static** **void** setChallengeHeight(**int** height) {  *challegneHeight* = height;  }  **public** **static** **int** getChallengeMinesNum() {  **return** *challengeMinesNum*;  }  **public** **static** **void** setChallengeMinesNum(**int** mines) {  *challengeMinesNum* = mines;  }  @Override  **protected** **void** victoryAction() {  JOptionPane.*showMessageDialog*(  **null**, "Congratulations. You managed to discover all the mines in "  + (minefield.getGameDuration() / 1000) + " seconds",  "victory", JOptionPane.***INFORMATION\_MESSAGE***);  // --------------------------------------------------------  **if** (*isChallenge*()) {  *setChallengeMinesNum*(*getChallengeMinesNum*() + 4);  *setChallengeWidth*(*getChallengeWidth*() + 3);  *setChallengeHeight*(*getChallengeHeight*() + 3);  ChallengeMode gameWindow = **new** ChallengeMode(  **new** Minefield(*getChallengeWidth*(), *getChallengeHeight*(), *getChallengeMinesNum*()));  gameWindow.setVisible(**true**);  }  } |

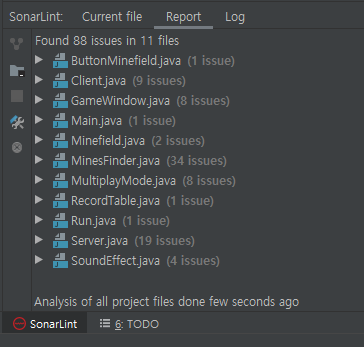
|  |
| --- |
| **public** ChallengeMode(Minefield minefield) {  *setLife*(1);  initComponents();  setTitle("Challenge Game");  isLeftPressed = **false**;  isRightPressed = **false**;  **this**.minefield = minefield;  buttons = **new** ButtonMinefield[minefield.getWidth()][minefield.getHeight()];  getContentPane().setLayout(**new** GridLayout(minefield.getWidth(), minefield.getHeight()));  createButtons();  }  } |

1. Refactoring
   1. Stan4J 비교
      1. Refactoring 전 Pollution
      2. Refactoring 후 Pollution
      3. Refactoring 전 Distance
      4. Refactoring 후 Distance



* 1. Sonar Lint 비교
     1. Refactoring 전



* + 1. Refactoring 후
  1. 변경사항
     1. 가시성, 이름 정리

Refactoring 이유

코드에 잘못된 가시성 표현, 불명확한 이름을 정리한다. 상속 등으로 인하여 변경되어야 할 가시성, 잘못 설정한 가시성을 고친다. 이름은 기능에 맞도록, 직관적으로 알 수 있도록 수정한다.

변경 내용

|  |
| --- |
| - MinesFinder – protected btn2ActionPerformed -> private btnMultiActionPerformed  - MinesFinder – protected btnCustomActionPerformed -> private btnCustomActionPerformed  - GameWindow – private isRightPressed -> protected isRightPressed  - GameWindow – private isLeftPressed -> protected isLeftPressed  - GameWindow – private initComponents - > protected initComponets  - GameWindow – public createButtons() -> protected createButtons()  - GameWindow – moksum -> life  - Server – public serverSocket -> pakage serverSocket  - Client – public socket -> pakage socket  - SoundEffect – bgmClip() -> bgmSound  - SoundEffect – clickClip() -> clickSound  - SoundEffect – boomClip() -> boomSound |

* + 1. 불필요 메소드, 임포트 제거

Refactoring 이유

사용하지 않는 메소드는 코드를 처음 이해할 때 낭비되는 시간을 가져오며 혼란을 불러온다. 그러므로 관리상 부담이다.

Copy and Paste로 인하여 불필요한 import가 생성된다. 이것도 라인수를 증가시킨다. 또한 다른 import를 해야 할 경우 혼란을 야기할 수 있다.

변경 내용

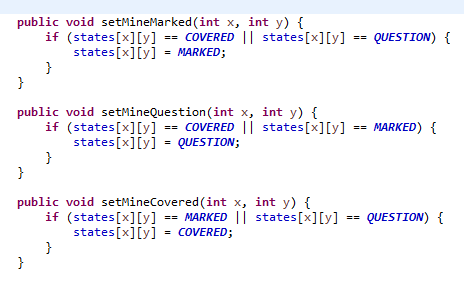
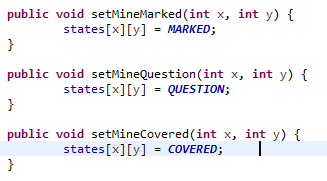
|  |
| --- |
| - GameWindow – main 함수 제거  - keyboardlistener 제거  - copy and paste로 인하여 생성된 임포트 제거 |

* + 1. 불필요한 비교 변경

Refactoring 이유

같은 비교를 여러 번 하여 불필요한 비교를 한다. 같은 기능을 하는 비교연산으로 인하여 불필요한 낭비가 발생한다. 비교를 1번만 하도록 변경한다.

변경 내용

* 변경 전
* 변경 후
  + 1. Run class 생성

Refactoring 이유

현재 MinesFinder클래스에 main함수가 존재하고 있다. Main을 Run클래스로 새로 생성하여 실행위치를 직관적으로 알 수 있도록 한다. 또한 MineFinder 클래스의 라인수를 줄인다.

변경 내용



* + 1. MultiMode class 생성

Refactoring 이유

기능을 클래스로 생성한다. 클래스로 기능을 설정함으로써 유지보수성이 증가한다. 또한 상속을 하여 중복 코드를 줄일 수 있다.

변경 내용

|  |
| --- |
| **public** **class** MultiplayMode **extends** GameWindow {    **private** **final** **static** Logger ***logger*** = Logger.*getGlobal*();  **private** Socket socket;  **private** String userName;  **public** MultiplayMode(Minefield minefield, String userName) {  startClient("211.33.5.110", 9999); //write Server ip and port  **this**.userName = userName;  GameWindow.*setLife*(1);  initComponents();  setTitle("Multi Game");  **this**.minefield = minefield;  buttons = **new** ButtonMinefield[minefield.getWidth()][minefield.getHeight()];  getContentPane().setLayout(**new** GridLayout(minefield.getWidth(), minefield.getHeight()));  createButtons();  }  @Override  **protected** **void** clickEvent() {  **if** (minefield.isGameFinished()) {  **if** (minefield.isPlayerDefeated()) {  send(userName + " " + "broke");  } **else** {  send(userName + " " + "finish");  }  }  }  //  **private** **void** startClient(String IP, **int** port) {  Thread thread = **new** Thread() {  **public** **void** run() {  **try** {  socket = **new** Socket(IP, port);  receive();  } **catch** (Exception e) {  **if** (!socket.isClosed()) {  stopClient();  ***logger***.log(Level.***INFO***, "server access fail");  Platform.*exit*();  }  }  }  };  thread.start();  }  **private** **void** stopClient() {  **try** {  **if** (socket != **null** && !socket.isClosed()) {  socket.close();  }  } **catch** (Exception e) {  ***logger***.log(Level.***INFO***, "exception msg", e);  }  }  **private** **void** receive() {  String dialog = "Result!";  **while** (**true**) {  **try** {  InputStream in = socket.getInputStream();  **byte**[] buffer = **new** **byte**[512];  **int** length = in.read(buffer);  **if** (length == -1)  **throw** **new** IOException();  String message = **new** String(buffer, 0, length, "UTF-8");  String[] strarr = message.split(" ");  **if** (strarr[0].equals(userName)) {  **if** (strarr[1].equals("finish")) {  JOptionPane.*showMessageDialog*(**null**, userName + " win", dialog,  JOptionPane.***INFORMATION\_MESSAGE***);  } **else** {  JOptionPane.*showMessageDialog*(**null**, userName + " lose", dialog,  JOptionPane.***INFORMATION\_MESSAGE***);  }  } **else** {  **if** (strarr[1].equals("finish")) {  JOptionPane.*showMessageDialog*(**null**, userName + " lose", dialog,  JOptionPane.***INFORMATION\_MESSAGE***);  } **else** {  JOptionPane.*showMessageDialog*(**null**, userName + " win", dialog,  JOptionPane.***INFORMATION\_MESSAGE***);  }  }  setVisible(**false**);  stopClient();  } **catch** (Exception e) {  stopClient();  **break**;  }  }  }  **private** **void** send(String message) {  Thread thread = **new** Thread() {  **public** **void** run() {  **try** {  OutputStream out = socket.getOutputStream();  **byte**[] buffer = message.getBytes("UTF-8");  out.write(buffer);  out.flush();  } **catch** (Exception e) {  stopClient();  }  }  };  thread.start();  }  } |

* + 1. CustomMode class 생성

Refactoring 이유

기능을 클래스로 생성한다. 클래스로 기능을 설정함으로써 유지보수성이 증가한다. 또한 상속을 하여 중복 코드를 줄일 수 있다.

변경 내용

|  |
| --- |
| **public** **class** CustomMode **extends** GameWindow {    **private** **static** **int** *customLife*;    **public** **static** **void** setCustomLife(**int** life) {  *customLife* = life;  }    **public** CustomMode(Minefield minefield) {    *setLife*(*customLife*);  initComponents();  setTitle("Custom Game");  **this**.minefield = minefield;  buttons = **new** ButtonMinefield[minefield.getWidth()][minefield.getHeight()];  getContentPane().setLayout(**new** GridLayout(minefield.getWidth(), minefield.getHeight()));    createButtons();  }  } |

* + 1. ChallengeMode class 생성

Refactoring 이유

기능을 클래스로 생성한다. 클래스로 기능을 설정함으로써 유지보수성이 증가한다. 또한 상속을 하여 중복 코드를 줄일 수 있다.

변경 내용

|  |
| --- |
| **public** **class** ChallengeMode **extends** GameWindow {  **private** **static** **int** *challengeWidth* = 0;  **private** **static** **int** *challegneHeight* = 0;  **private** **static** **int** *challengeMinesNum* = 0;  **private** **static** **boolean** *ischallenge* = **false**;  **public** **int** getLife(**int** life){  **return** life;  }  **public** **static** **boolean** isChallenge() {  **return** *ischallenge*;  }  **public** **static** **void** setChallenge(**boolean** setChal) {  *ischallenge* = setChal;  }  **public** **static** **int** getChallengeWidth() {  **return** *challengeWidth*;  }  **public** **static** **void** setChallengeWidth(**int** width) {  *challengeWidth* = width;  }  **public** **static** **int** getChallengeHeight() {  **return** *challegneHeight*;  }  **public** **static** **void** setChallengeHeight(**int** height) {  *challegneHeight* = height;  }  **public** **static** **int** getChallengeMinesNum() {  **return** *challengeMinesNum*;  }  **public** **static** **void** setChallengeMinesNum(**int** mines) {  *challengeMinesNum* = mines;  }      @Override  **protected** **void** victoryAction() {  JOptionPane.*showMessageDialog*(  **null**, "Congratulations. You managed to discover all the mines in "  + (minefield.getGameDuration() / 1000) + " seconds",  "victory", JOptionPane.***INFORMATION\_MESSAGE***);    **if** (*isChallenge*()) {  *setChallengeMinesNum*(*getChallengeMinesNum*() + 4);  *setChallengeWidth*(*getChallengeWidth*() + 3);  *setChallengeHeight*(*getChallengeHeight*() + 3);  ChallengeMode gameWindow = **new** ChallengeMode(  **new** Minefield(*getChallengeWidth*(), *getChallengeHeight*(), *getChallengeMinesNum*()));  gameWindow.setVisible(**true**);  }  }  **public** ChallengeMode(Minefield minefield) {  *setLife*(1);  initComponents();  setTitle("Challenge Game");  isLeftPressed = **false**;  isRightPressed = **false**;  **this**.minefield = minefield;  buttons = **new** ButtonMinefield[minefield.getWidth()][minefield.getHeight()];  getContentPane().setLayout(**new** GridLayout(minefield.getWidth(), minefield.getHeight()));  createButtons();  }  } |

* + 1. GameWindow 상속관계 생성

Refactoring 이유

GameWindow 클래스를 부모, Multi, Custom, Classic, Chellange 클래스를 자식으로 하여 상속관계를 생성한다. 중첩되는 코드들을 GameWindow 클래스에 정의, 사용하여 중복코드를 제거한다. 필요시 오버라이딩으로 필요한 기능으로 하여 사용한다.

변경 내용

|  |
| --- |
| **public** **class** GameWindow **extends** javax.swing.JFrame {  **protected** JMenuBar jmb = **new** JMenuBar();  **protected** JMenu showLifeMenu = **new** JMenu();  **protected** ButtonMinefield[][] buttons;  **protected** Minefield minefield;  **protected** RecordTable record;  **protected** **boolean** isLeftPressed;  **protected** **boolean** isRightPressed;  **private** **static** **int** *life* = 1;  **public** **static** **int** getLife() {  **return** *life*;  }  **public** **static** **void** setLife(**int** setlife) {  *life* = setlife;  }  **protected** **void** updateButtonsStates() {  **for** (**int** x = 0; x < minefield.getWidth(); x++) {  **for** (**int** y = 0; y < minefield.getHeight(); y++) {  buttons[x][y].setEstado(minefield.getGridState(x, y));  }  }  }  **protected** **void** victoryAction() {  JOptionPane.*showMessageDialog*(  **null**, "Congratulations. You managed to discover all the mines in "  + (minefield.getGameDuration() / 1000) + " seconds",  "victory", JOptionPane.***INFORMATION\_MESSAGE***);  }  **protected** **void** defeatedAction() {  JOptionPane.*showMessageDialog*(**null**, "Oh, a mine broke", "Lost!", JOptionPane.***INFORMATION\_MESSAGE***);  }  **protected** **void** clickEvent() {  **if** (minefield.isGameFinished()) {  **if** (minefield.isPlayerDefeated()) {  defeatedAction();  } **else** {  victoryAction();  }  setVisible(**false**);  }  }  ActionListener action = **new** ActionListener() {  @Override  **public** **void** actionPerformed(ActionEvent e) {  ButtonMinefield button = (ButtonMinefield) e.getSource();  **int** x = button.getCol();  **int** y = button.getLine();  minefield.revealGrid(x, y);  showLifeMenu.setText("life : " + Integer.*toString*(*getLife*()));  SoundEffect.*clickClip*();  updateButtonsStates();  clickEvent();  }  };  MouseListener mouseListener = **new** MouseListener() {  @Override  **public** **void** mousePressed(MouseEvent e) {  ButtonMinefield botao = (ButtonMinefield) e.getSource();  **int** x = botao.getCol();  **int** y = botao.getLine();  **if** (minefield.getGridState(x, y) >= 1 && minefield.getGridState(x, y) <= 8) {  **if** (SwingUtilities.*isLeftMouseButton*(e)) {  isLeftPressed = **true**;  } **else** **if** (SwingUtilities.*isRightMouseButton*(e)) {  isRightPressed = **true**;  }  **if** (isLeftPressed && isRightPressed) {  minefield.doubleCliked(x, y);  }  }    **if** (e.getButton() == MouseEvent.***BUTTON3***) {  **if** (minefield.getGridState(x, y) == minefield.***COVERED***) {  minefield.setMineMarked(x, y);  } **else** **if** (minefield.getGridState(x, y) == minefield.***MARKED***) {  minefield.setMineQuestion(x, y);  } **else** **if** (minefield.getGridState(x, y) == minefield.***QUESTION***) {  minefield.setMineCovered(x, y);  }  updateButtonsStates();  }  }  @Override  **public** **void** mouseClicked(MouseEvent me) {  }  @Override  **public** **void** mouseReleased(MouseEvent e) {  **if** (SwingUtilities.*isLeftMouseButton*(e)) {  isLeftPressed = **false**;  } **else** **if** (SwingUtilities.*isRightMouseButton*(e)) {  isRightPressed = **false**;  }  }  @Override  **public** **void** mouseEntered(MouseEvent me) {  }  @Override  **public** **void** mouseExited(MouseEvent me) {  }  };    **protected** **void** createButtons() {  **for** (**int** x = 0; x < minefield.getWidth(); x++) {  **for** (**int** y = 0; y < minefield.getHeight(); y++) {  buttons[x][y] = **new** ButtonMinefield(x, y);  buttons[x][y].addActionListener(action);  buttons[x][y].addMouseListener(mouseListener);  getContentPane().add(buttons[x][y]);  }  }  }  @SuppressWarnings("unchecked")  **protected** **void** initComponents() {  setDefaultCloseOperation(javax.swing.WindowConstants.***DISPOSE\_ON\_CLOSE***);  setTitle("default");  setCursor(**new** java.awt.Cursor(java.awt.Cursor.***DEFAULT\_CURSOR***));  showLifeMenu.setText("life : " + Integer.*toString*(*getLife*()));  jmb.add(showLifeMenu);  setJMenuBar(jmb);  javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());  getContentPane().setLayout(layout);  layout.setHorizontalGroup(  layout.createParallelGroup(javax.swing.GroupLayout.Alignment.***LEADING***).addGap(0, 1094, Short.***MAX\_VALUE***));  layout.setVerticalGroup(  layout.createParallelGroup(javax.swing.GroupLayout.Alignment.***LEADING***).addGap(0, 553, Short.***MAX\_VALUE***));  pack();  }  } |

* + 1. 예외처리, server 콘솔 출력 메시지 log로 변경

Refactoring 이유

오류 발생 시점을 잡기 좋다. Sever 접속 정보 등의 표현이 용이하다.

변경 내용

|  |
| --- |
| **private** **void** startClient(String IP, **int** port) {  Thread thread = **new** Thread() {  **public** **void** run() {  **try** {  socket = **new** Socket(IP, port);  receive();  } **catch** (Exception e) {  **if** (!socket.isClosed()) {  stopClient();  ***logger***.log(Level.***INFO***, "server access fail");  Platform.*exit*();  }  }  }  };  thread.start();  } |

|  |
| --- |
| **public** **void** startServer(String IP, **int** port) {  **try** {  serverSocket = **new** ServerSocket();  serverSocket.bind(**new** InetSocketAddress(IP, port));  } **catch** (Exception e) {  ***logger***.log(Level.***INFO***, "exception msg", e);  **if** (!serverSocket.isClosed()) {  stopServer();  }  **return**;  } |

|  |
| --- |
| Runnable thread = **new** Runnable() {  @Override  **public** **void** run() {  **while** (**true**) {  **try** {  Socket socket = serverSocket.accept();  *clients*.add(**new** Client(socket));  ***logger***.log(Level.***INFO***, "client access" +socket.getRemoteSocketAddress() + " : " + Thread.*currentThread*().getName());  } **catch** (Exception e) {  **if** (!serverSocket.isClosed()) {  stopServer();  }  **break**;  }  }  }  }; |

* + 1. 소리 끄기 버튼 생성

Refactoring 이유

사용자가 BGM을 안 듣고 싶을 때 끌 수 없었다. 그래서 사용자에게 BGM을 켜고 끌 수 있는 선택권을 부여한다.

변경 내용

|  |
| --- |
| btnBGMCtrl.setText("BGM ON");  btnBGMCtrl.addActionListener(**new** java.awt.event.ActionListener() {  **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {  **if**(btnBGMCtrl.getText().equals("BGM ON")) {  SoundEffect.*bgmStop*();  btnBGMCtrl.setText("BGM OFF");  }  **else** {  SoundEffect.*bgmClip*();  btnBGMCtrl.setText("BGM ON");  }  }  });  panelBtns.add(btnBGMCtrl); |

* + 1. SoundEffect class 조정

Refactoring 이유

효과음이 실행될 때마다 파일의 경로를 입력하고 객체를 생성하게 되어 개발자 관점에서 불편하다. 그래서 SoundEffect class에 각 효과음 별로 실행하는 method를 만들어 사용편의성을 높인다. 또한 SoundEffect class에서 경로를 관리함으로써 수정이 용이하다.

변경 내용

|  |
| --- |
| **public** **class** SoundEffect {  **private** **final** **static** Logger ***logger*** = Logger.*getGlobal*();  **private** **static** Clip[] *clip* = **new** Clip[3];  **private** **static** String[] *file* = { "src/pt/technic/apps/minesfinder/music/BGM.wav",  "src/pt/technic/apps/minesfinder/music/ClickS.wav", "src/pt/technic/apps/minesfinder/music/BoomS.wav" };  // 0 : BGM  // 1 : ClickS  // 2 : BoomS  **public** **static** **void** bgmClip() {  **try** {  File base = **new** File(*file*[0]);  String basePath = base.getAbsolutePath();  File audio = **new** File(basePath);  AudioInputStream ais = AudioSystem.*getAudioInputStream*(audio);  *clip*[0] = AudioSystem.*getClip*();  *clip*[0].open(ais);  } **catch** (Exception e) {  ***logger***.log(Level.***INFO***, "exception msg", e);  }  *clip*[0].flush();  *clip*[0].start();  *clip*[0].loop(*clip*[0].***LOOP\_CONTINUOUSLY***);  }    **public** **static** **void** bgmStop() {  *clip*[0].stop();  }  **public** **static** **void** clickClip() {  **try** {  File base = **new** File(*file*[1]);  String basePath = base.getAbsolutePath();  File audio = **new** File(basePath);  AudioInputStream ais = AudioSystem.*getAudioInputStream*(audio);  *clip*[1] = AudioSystem.*getClip*();  *clip*[1].open(ais);  } **catch** (Exception e) {  ***logger***.log(Level.***INFO***, "exception msg", e);  }  *clip*[1].start();  }  **public** **static** **void** boomClip() {  **try** {  File base = **new** File(*file*[2]);  String basePath = base.getAbsolutePath();  File audio = **new** File(basePath);  AudioInputStream ais = AudioSystem.*getAudioInputStream*(audio);  *clip*[2] = AudioSystem.*getClip*();  *clip*[2].open(ais);  } **catch** (Exception e) {  ***logger***.log(Level.***INFO***, "exception msg", e);  }  *clip*[2].start();  }  } |

* + 1. Sonar Lint tool 사용

Refactoring 이유

정적 코드 분석 도구인 sonar lint를 사용하였다. 발생한 오류를 최대한 해결하기 위하여 Refactoring 하였다.

변경 내용

5.2의 Sonar Link의 5.2.1그림에서 5.2.2의 그림으로 변경되었다.

* + 1. 주석 정리

Refactoring 이유

주석이 없어도 기능을 알 수 있도록 Refactoring 한다.

변경 내용

불필요한 주석을 제거한다. 예로 공동작업을 하며 수정 부분을 표기하기 위해 사용한 주석 등을 제거하였다. 최대한 주석이 필요 없는 코드를 만든다.

꼭 필요한 내용은 주석으로 표기하였다. IP, port 입력 등을 주석으로 표기하였다.