

Programación orientada a objetos

Códigos fuente TPE

Dungeon Game

6 de junio de 2011

Autores:

<i>Tomás Mehdi</i>	Legajo: 51014
<i>Alan Pomerantz</i>	Legajo: 51233

Índice

1. Codigos fuente	3
1.1. back	3
1.1.1. Algoritms.java	3
1.1.2. BloodyFloor.java	3
1.1.3. BoardObtainer.java	3
1.1.4. Bonus.java	4
1.1.5. BonusTypes.java	4
1.1.6. Cell.java	5
1.1.7. Character.java	6
1.1.8. DungeonGameImp.java	8
1.1.9. DungeonGameListener.java	12
1.1.10. Floor.java	12
1.1.11. Game.java	12
1.1.12. GameListener.java	13
1.1.13. GrabBonus.java	13
1.1.14. LoadGame.java	13
1.1.15. Monster.java	14
1.1.16. MonsterTypes.java	15
1.1.17. MoveTypes.java	16
1.1.18. Player.java	17
1.1.19. PlayerData.java	18
1.1.20. Point.java	19
1.1.21. Putable.java	20
1.1.22. SaveGame.java	21
1.1.23. Strokes.java	21
1.1.24. Wall.java	21
1.2. front	21
1.2.1. App.java	21
1.2.2. DataPanel.java	22
1.2.3. DataPanelListener.java	23
1.2.4. DefaultGameMenuBar.java	24
1.2.5. DungeonGameFrame.java	24
1.2.6. DungeonPanel.java	30
1.2.7. DungeonPanelListener.java	37
1.2.8. GameFrame.java	37
1.2.9. LevelSelector.java	38
1.2.10. LevelSelectorImp.java	39
1.3. parser	39
1.3.1. BoardDimensionLine.java	39
1.3.2. BoardLine.java	40

1.3.3.	BoardNameLine.java	41
1.3.4.	BoardParserFromFile.java	42
1.3.5.	CorruptedFileException.java	45
1.3.6.	Lines.java	45
1.3.7.	SavedBoardPlayerLine.java	46
1.4.	professorShipSrc	46
1.4.1.	GamePanel.java	46
1.4.2.	GamePanelListener.java	48
1.4.3.	ImageUtils.java	48
1.5.	saveLoadImplementation	49
1.5.1.	Criteria.java	49
1.5.2.	FilterArrayFileList.java	50
1.5.3.	FilterFileList.java	50
1.5.4.	LoadGameFromFile.java	50
1.5.5.	SaveGameOnFile.java	52
1.5.6.	SavingCorruptedException.java	55
1.6.	tests	56
1.6.1.	GameTests.java	56
1.6.2.	PlayerTests.java	59
1.6.3.	ParserTests.java	60

1. Codigos fuente

1.1. back

1.1.1. Algoritms.java

```
1 package back;
2
3 /**
4  * @author tomas
5  * Interface that represents the function/algorithm of monsters life ↵
6  * and strength.
7  */
8 public interface Algoritms {
9     public Integer lifeAlgoritm(int level);
10
11     public Integer strengthAlgoritm(int level);
12 }
```

1.1.2. BloodyFloor.java

```
1 package back;
2
3 public class BloodyFloor extends Floor{
4     @Override
5     public String toString() {
6         return "Blood";
7     }
8 }
```

1.1.3. BoardObtainer.java

```
1 package back;
2
3 import java.io.File;
4
5 public interface BoardObtainer {
6
7     public void obtainBoard() throws Exception;
8
9     public Point getBoardDimension();
10
11     public Putable [][] getBoard();
12
13     public Point getPlayerPosition();
14
15     public String getBoardName();
16
17     public Putable getBoardElem(Point point);
18
19     public int getBoardRows();
20
21     public int getBoardColumns();
22
23     public File getFile();
24
25     public int getPlayerSteps();
26 }
```

```
27 | }
```

1.1.4. Bonus.java

```
1 package back;
2
3 public class Bonus extends Cell implements Putable {
4
5     private BonusTypes bonusType;
6
7     public Bonus(Point position, int numberBonusType, int bonusAmount) {
8         {
9             bonusType = BonusTypes.getBonusType(numberBonusType);
10            bonusType.setBonusAmount(bonusAmount);
11        }
12
13     public void giveBonus(Character character) {
14         bonusType.giveBonus(character);
15     }
16
17     @Override
18     public boolean allowMovement(DungeonGameImp game) {
19         return true;
20     }
21
22     public void standOver(DungeonGameImp game) {
23         giveBonus(game.getPlayer());
24         Point point = new Point(game.getPlayer().getPosition().x, game
25             .getPlayer().getPosition().y);
26
27         Floor f = new Floor();
28         f.setVisible();
29         game.getBoard()[point.x][point.y] = f;
30
31         game.getGameListener().executeWhenBonusGrabed(
32             new Point(point.x, point.y));
33     }
34
35     public BonusTypes getBonusType() {
36         return bonusType;
37     }
38
39     public int getAmountBonus() {
40         return bonusType.getBonusAmount();
41     }
42
43     @Override
44     public String toString() {
45         return "Bonus";
46     }
47 }
```

1.1.5. BonusTypes.java

```
1 package back;
2
3 /**
4  * @author tomas
5  * A beautiful enumerate for the different types of Bonuses.
6  */
7 public enum BonusTypes {
8
```

```

9      LIFE("Life", 0, new GrabBonus(){
10
11          @Override
12          public void grabBonus(Character character, Integer bonusAmount) {
13              character.winLife(bonusAmount);
14          }
15      }), STRENGTH("Strength", 0, new GrabBonus(){
16
17          @Override
18          public void grabBonus(Character character, Integer bonusAmount) {
19              character.grabStrengthBonus(bonusAmount);
20          }
21      });
22
23      private String name;
24      private Integer bonusAmount;
25      private GrabBonus bonusGrabbed;
26
27      private BonusTypes(String name, Integer bonusAmount, GrabBonus bonusGrabbed) {
28          this.name = name;
29          this.bonusAmount = bonusAmount;
30          this.bonusGrabbed = bonusGrabbed;
31      }
32
33      public Integer getBonusAmount() {
34          return bonusAmount;
35      }
36
37      public void setBonusAmount(Integer bonusAmount) {
38          this.bonusAmount = bonusAmount;
39      }
40
41      public String getName() {
42          return name;
43      }
44
45      public static BonusTypes getBonusType(int data) {
46          switch (data) {
47              case (4):
48                  return BonusTypes.LIFE;
49              case (5):
50                  return BonusTypes.STRENGTH;
51              default:
52                  return null;
53          }
54      }
55
56      public void giveBonus(Character character) {
57          bonusGrabbed.grabBonus(character, getBonusAmount());
58      }
59
60  }
61

```

1.1.6. Cell.java

```

1  package back;
2
3  /**
4   * @author tomas
5   * Abstract class inserted on the hierarchy to make every class that
6   * can be on the board
7   * to be visible or invisible. Particular feature of this game.
8   */
9  public abstract class Cell {

```

```
10     boolean isVisible = false;
11
12     public boolean isVisible() {
13         return isVisible;
14     }
15
16     public void setVisible() {
17         this.isVisible = true;
18     }
19
20     public void setNotVisible() {
21         this.isVisible = false;
22     }
23
24 }
```

1.1.7. Character.java

```
1 package back;
2
3 /**
4  * @author tomas Abstract class that extends cell. So it can be
5  * visible or
6  * invisible in the board.
7  */
8 public abstract class Character extends Cell {
9
10     private String name;
11     private Integer level;
12     private Integer maxHealth;
13     private Integer health;
14     private Integer strength;
15     private Point position;
16
17     public Character(String name, Integer level, Point position) {
18         this.name = name;
19         this.level = level;
20         this.position = position;
21     }
22
23     public void winFight(Character character) {
24     }
25
26     public void fightAnotherCharacter(Character character) {
27         this.hited(character.getStrength());
28         if (!this.isDead()) {
29             character.hited(this.getStrength());
30             if (character.isDead()) {
31                 this.winFight(character);
32             }
33         } else {
34             character.winFight(this);
35         }
36     }
37
38     public void hited(Integer strength) {
39         health -= strength;
40     }
41
42     public String getName() {
43         return name;
44     }
45
46     public boolean isDead() {
47         return health <= 0;
48     }
49
50     public Integer getLevel() {
```

```

51         return level;
52     }
53
54     public void increaseLevel() {
55         this.level += 1;
56     }
57
58     public Integer getMaxHealth() {
59         return maxHealth;
60     }
61
62     public Integer getHealth() {
63         return health;
64     }
65
66     public Integer getStrength() {
67         return strength;
68     }
69
70     public Point getPosition() {
71         return position;
72     }
73
74     @Override
75     public String toString() {
76         String resp;
77         resp = "Name=" + getName();
78         resp += "Level=" + getLevel();
79         resp += "MaxHealth=" + getMaxHealth();
80         resp += "Health=" + getHealth();
81         resp += "Strength=" + getStrength();
82         resp += "Position=" + getPosition();
83         return resp;
84     }
85
86     public void winLife(Integer bonusAmount) {
87         if (health + bonusAmount > maxHealth) {
88             health = maxHealth;
89         } else {
90             health += bonusAmount;
91         }
92     }
93
94     public void grabStrengthBonus(Integer bonusAmount) {
95         strength += bonusAmount;
96     }
97
98     /**
99     * Method just for tests
100    *
101    * @param position
102    */
103    public void setPosition(Point position) {
104        this.position = position;
105    }
106
107    public void setMaxHealth(int maxHealth) {
108        this.maxHealth = maxHealth;
109    }
110
111    public void setStrength(int strength) {
112        this.strength = strength;
113    }
114
115    public void setHealth(Integer health) {
116        this.health = health;
117    }
118
119    @Override
120    public int hashCode() {
121        final int prime = 31;
122        int result = 1;
123        result = prime * result + ((health == null) ? 0 : health.↵
            hashCode());

```



```

124         result = prime * result + ((level == null) ? 0 : level.hashCode());
125         result = prime * result
126             + ((maxHealth == null) ? 0 : maxHealth.hashCode());
127         result = prime * result + ((name == null) ? 0 : name.hashCode());
128         result = prime * result
129             + ((position == null) ? 0 : position.hashCode());
130         result = prime * result
131             + ((strength == null) ? 0 : strength.hashCode());
132         return result;
133     }
134
135     @Override
136     public boolean equals(Object obj) {
137         if (this == obj)
138             return true;
139         if (obj == null)
140             return false;
141         if (getClass() != obj.getClass())
142             return false;
143         Character other = (Character) obj;
144         if (health == null) {
145             if (other.health != null)
146                 return false;
147         } else if (!health.equals(other.health))
148             return false;
149         if (level == null) {
150             if (other.level != null)
151                 return false;
152         } else if (!level.equals(other.level))
153             return false;
154         if (maxHealth == null) {
155             if (other.maxHealth != null)
156                 return false;
157         } else if (!maxHealth.equals(other.maxHealth))
158             return false;
159         if (name == null) {
160             if (other.name != null)
161                 return false;
162         } else if (!name.equals(other.name))
163             return false;
164         if (position == null) {
165             if (other.position != null)
166                 return false;
167         } else if (!position.equals(other.position))
168             return false;
169         if (strength == null) {
170             if (other.strength != null)
171                 return false;
172         } else if (!strength.equals(other.strength))
173             return false;
174         return true;
175     }
176
177     public void setLevel(int level) {
178         this.level = level;
179     }
180
181 }

```

1.1.8. DungeonGameImp.java

```

1 package back;
2
3 import java.io.File;
4 import java.util.ArrayList;
5 import java.util.List;
6

```

```

7  /**
8  * @author tomas Back end most important class. It contents all the ↵
      data to play
9  *      a Dungeon Game. This class implements Game.
10  */
11 public class DungeonGameImp implements Game {
12
13     final static Integer LEVEL = 3;
14     final static Integer LIFE = 10;
15     final static Integer STRENGTH = 5;
16
17     private String boardName;
18     private Player player;
19     private Point boardDimension;
20     private Putable[][] board;
21     private GameListener gameListener;
22     private BoardObtainer boardObtainer;
23
24     @SuppressWarnings("unchecked")
25     public DungeonGameImp(BoardObtainer boardObtainer, GameListener ↵
      gameListener) {
26         this.boardObtainer = boardObtainer;
27         this.gameListener = gameListener;
28         boardName = boardObtainer.getBoardName();
29         boardDimension = boardObtainer.getBoardDimension();
30         board = boardObtainer.getBoard();
31         PlayerData playerData = new PlayerData(null, 0, 0, LIFE, LIFE,
32             STRENGTH, boardObtainer.getPlayerPosition(),
33             boardObtainer.getPlayerSteps());
34         if (!(boardObtainer instanceof LoadGame)) {
35             playerData.setName(gameListener.playerNameRequest());
36             player = new Player(playerData);
37         } else {
38             playerData
39                 .setName(((LoadGame<Game>) boardObtainer).↵
      getPlayerName());
40             playerData.setHealth(((LoadGame<Game>) boardObtainer)
41                 .getPlayerLoadedHealth());
42             playerData.setMaxHealth(((LoadGame<Game>) boardObtainer)
43                 .getPlayerLoadedMaxHealth());
44             playerData.setStrength(((LoadGame<Game>) boardObtainer)
45                 .getPlayerLoadedStrength());
46             playerData.setExperience(((LoadGame<Game>) boardObtainer)
47                 .getPlayerLoadedExperience());
48             player = new Player(playerData,
49                 ((LoadGame<Game>) boardObtainer).↵
      getPlayerLoadedLevel(),
50                 ((LoadGame<Game>) boardObtainer).↵
      getPlayerLoadedSteps());
51         }
52         firstDiscoveredCells();
53     }
54
55     private void firstDiscoveredCells() {
56         Point p = player.getPosition();
57
58         board[p.x][p.y].setVisible();
59
60         board[p.x + 1][p.y - 1].setVisible();
61         board[p.x + 1][p.y].setVisible();
62         board[p.x + 1][p.y + 1].setVisible();
63
64         board[p.x][p.y - 1].setVisible();
65         board[p.x][p.y].setVisible();
66         board[p.x][p.y + 1].setVisible();
67
68         board[p.x - 1][p.y - 1].setVisible();
69         board[p.x - 1][p.y].setVisible();
70         board[p.x - 1][p.y + 1].setVisible();
71     }
72
73     /**
74     * @see back.Game#receiveMoveStroke(back.MoveTypes) Is't allow the ↵
      game to

```

```

75     *     receive a Stroke. In this case a MoveTypes stroke. Before ←
76     this the
77     *     player moves.
78     */
79     @Override
80     public void receiveMoveStroke(MoveTypes moveType) {
81         Point nextPlayerPosition = player.getPosition().add(
82             moveType.getDirection());
83         int playerLevelBeforeFight = player.getLevel();
84         if (board[nextPlayerPosition.x][nextPlayerPosition.y]
85             .allowMovement(this)) {
86             MoveTypes moveMade = player.move(moveType);
87             dicoverBoard(nextPlayerPosition, moveType);
88             gameListener.executeWhenPlayerMoves(moveMade);
89             board[nextPlayerPosition.x][nextPlayerPosition.y].←
90                 standOver(this);
91         }
92         if (player.getLevel() != playerLevelBeforeFight) {
93             gameListener.executeWhenLevelUp();
94         }
95     }
96     /**
97     * When player moves exist the possibility of discover ←
98     undiscovered board
99     * parts. When this happen the game have to give life to ←
100    characters on the
101    * parts of the board already discovered. This amount is equals of←
102    the
103    * character level.
104    */
105    private void dicoverBoard(Point pos, MoveTypes dir) {
106        int countDiscover = 0;
107        List<Point> points = new ArrayList<Point>();
108        points.add(pos.add(dir.getDirection()));
109        if (dir == MoveTypes.LEFT || dir == MoveTypes.RIGHT) {
110            points.add(pos.add(1, 0).add(dir.getDirection()));
111            points.add(pos.sub(1, 0).add(dir.getDirection()));
112        } else {
113            points.add(pos.add(0, 1).add(dir.getDirection()));
114            points.add(pos.sub(0, 1).add(dir.getDirection()));
115        }
116        for (Point poo : points) {
117            if (!board[poo.x][poo.y].isVisible()) {
118                countDiscover++;
119                board[poo.x][poo.y].setVisible();
120            }
121        }
122        if (countDiscover > 0) {
123            player.winLife(countDiscover * player.getLevel());
124            for (int i = 1; i < boardDimension.x - 1; i++) {
125                for (int j = 1; j < boardDimension.y - 1; j++) {
126                    if (board[i][j].isVisible()
127                        && board[i][j] instanceof Character) {
128                        ((Character) board[i][j]).winLife(←
129                            countDiscover
130                            * ((Character) board[i][j]).getLevel()←
131                        );
132                    }
133                }
134            }
135        }
136    }
137    @Override
138    public Player getPlayer() {
139        return player;
140    }
141    @Override
142    public void wonned() {
143        gameListener.executeWhenGameWonned();
144    }

```

```

142     }
143
144     @Override
145     public void loosed() {
146         gameListener.executeWhenGameLoosed();
147     }
148
149     /**
150     * @param character
151     *      Method executed when a fight end. It's validate if a
152     *      character
153     *      died. If any died execute a listener was provided by
154     *      the
155     *      front.
156     */
157     public void fightEnd(Character character) {
158         if (character.isDead()) {
159             Point point = new Point(character.getPosition().x,
160                                     character.getPosition().y);
161             BloodyFloor bf = new BloodyFloor();
162             bf.setVisible();
163             board[point.x][point.y] = bf;
164             gameListener.executeWhenCharacterDie(point);
165         }
166         if (player.isDead()) {
167             Point point = new Point(player.getPosition().x,
168                                     player.getPosition().y);
169             BloodyFloor bf = new BloodyFloor();
170             bf.setVisible();
171             board[point.x][point.y] = bf;
172             gameListener.executeWhenCharacterDie(point);
173             loosed();
174         }
175         gameListener.executeWhenFight();
176     }
177
178     @Override
179     public Putable[][] getBoard() {
180         return board;
181     }
182
183     @Override
184     public Point getBoardDimension() {
185         return boardDimension;
186     }
187
188     @Override
189     public String getBoardName() {
190         return boardName;
191     }
192
193     @Override
194     public GameListener getGameListener() {
195         return gameListener;
196     }
197
198     @Override
199     public void addGameListener(GameListener d) {
200         gameListener = d;
201     }
202
203     @Override
204     public BoardObtainer getBoardObtainer() {
205         return boardObtainer;
206     }
207
208     /**
209     * @see back.Game#restart() The desition of making restart a
210     *      method of a
211     *      game and not a class like loadGame is that a restart game
212     *      need the

```

```
211     *      same boardObtainer that the instance of the game. Then is ↵
212     *      have no
213     *      sense make a new instance.
214     */
215     @Override
216     public void restart() {
217         File file = boardObtainer.getFile();
218         try {
219             board = boardObtainer.getClass().getConstructor(File.class, ↵
220             )
221                 .newInstance(file).getBoard();
222         } catch (Exception e) {
223             PlayerData playerData = new PlayerData(player.getName(), 0, 0, ↵
224             LIFE,
225             LIFE, STRENGTH, boardObtainer.getPlayerPosition(),
226             player.getSteps());
227             player = new Player(playerData);
228         }
229     }
230 }
```

1.1.9. DungeonGameListener.java

```
1 package back;
2
3 public interface DungeonGameListener extends GameListener {}
```

1.1.10. Floor.java

```
1 package back;
2
3 public class Floor extends Cell implements Putable {
4     @Override
5     public String toString() {
6         return "Floor";
7     }
8
9     @Override
10    public boolean allowMovement(DungeonGameImp game) {
11        return true;
12    }
13
14    @Override
15    public void standOver(DungeonGameImp game) {}
16
17 }
```

1.1.11. Game.java

```
1 package back;
2
3 public interface Game {
4
5     public void wonned();
6
7     public void loosed();
8 }
```

```
9      public Player getPlayer();
10
11     public Putable [][] getBoard();
12
13     public Point getBoardDimension();
14
15     public String getBoardName();
16
17     public GameListener getGameListener();
18
19     public void addGameListener(GameListener d);
20
21     public BoardObtainer getBoardObtainer();
22
23     public void restart();
24
25     public void receiveMoveStroke(MoveTypes moveType);
26
27 }
```

1.1.12. GameListener.java

```
1 package back;
2
3 public interface GameListener {
4
5     public void executeWhenPlayerMoves(MoveTypes moveType);
6
7     public void executeWhenFight();
8
9     public void executeWhenBonusGrabed(Point pos);
10
11     public void executeWhenCharacterDie(Point pos);
12
13     public void executeWhenGameLoosed();
14
15     public void executeWhenGameWon();
16
17     public String playerNameRequest();
18
19     void executeWhenLevelUp();
20
21 }
```

1.1.13. GrabBonus.java

```
1 package back;
2
3 public interface GrabBonus {
4     public void grabBonus(Character character, Integer bonusAmount);
5 }
```

1.1.14. LoadGame.java

```
1 package back;
2
3 public interface LoadGame<T extends Game> {
4
```

```

5      public T getGame(Class<T> gameImpClass, GameListener listener);
6
7      public Integer getPlayerLoadedSteps();
8
9      Integer getPlayerLoadedExperience();
10
11     Integer getPlayerLoadedStrength();
12
13     public int getPlayerLoadedLevel();
14
15     public Integer getPlayerLoadedHealth();
16
17     public Integer getPlayerLoadedMaxHealth();
18
19     public String getPlayerName();
20
21 }

```

1.1.15. Monster.java

```

1  package back;
2
3  public class Monster extends Character implements Putable {
4
5      @Override
6      public int hashCode() {
7          final int prime = 31;
8          int result = super.hashCode();
9          result = prime * result
10             + ((monsterType == null) ? 0 : monsterType.hashCode())↵
11             ;
12         return result;
13     }
14
15     @Override
16     public boolean equals(Object obj) {
17         if (this == obj)
18             return true;
19         if (!super.equals(obj))
20             return false;
21         if (getClass() != obj.getClass())
22             return false;
23         Monster other = (Monster) obj;
24         if (monsterType == null) {
25             if (other.monsterType != null)
26                 return false;
27         } else if (!monsterType.equals(other.monsterType))
28             return false;
29         return true;
30     }
31
32     private MonsterTypes monsterType;
33
34     public Monster(Point position, int numberMonsterType, int level) {
35         this(position, numberMonsterType, level, MonsterTypes.↵
36             getMonsterType(
37                 numberMonsterType).getMaxLife(level));
38     }
39
40     public Monster(Point position, int numberMonsterType, int level, ↵
41         int health) {
42         super(MonsterTypes.getMonsterType(numberMonsterType).getName()↵
43             , level,
44             position);
45         monsterType = MonsterTypes.getMonsterType(numberMonsterType);
46         setMaxHealth(monsterType.getMaxLife(level));
47         setStrength(monsterType.getStrength(level));
48         setHealth(health);
49     }
50 }

```

```

46
47     public MonsterTypes getMonsterType() {
48         return monsterType;
49     }
50
51     @Override
52     public String toString() {
53         return monsterType.getName();
54     }
55
56     @Override
57     public boolean allowMovement(DungeonGameImp game) {
58         game.getPlayer().fightAnotherCharacter(this);
59         game.fightEnd(this);
60         if (this.isDead()) {
61             if (this.getLevel() == DungeonGameImp.LEVEL) {
62                 game.winned();
63             }
64         }
65         return false;
66     }
67
68     @Override
69     public void standOver(DungeonGameImp game) {
70     }
71
72 }

```

1.1.16. MonsterTypes.java

```

1  package back;
2
3  public enum MonsterTypes {
4
5
6      GOLEM("Golem", new Algorithms() {
7
8          @Override
9          public Integer lifeAlgoritm(int level) {
10             return (int) Math.floor((((level + 3) * (level + 3)) - 10)↵
11                 * GOLEMLIFE);
12         }
13
14         @Override
15         public Integer strengthAlgoritm(int level) {
16             return (int) Math.floor(((level * level) + 5 * level) * ↵
17                 0.5 * GOLEMSTRENGTH);
18         }
19     }), DRAGON("Dragon", new Algorithms() {
20
21         @Override
22         public Integer lifeAlgoritm(int level) {
23             return (int) Math.floor((((level + 3) * (level + 3)) - 10)↵
24                 * DRAGONLIFE);
25         }
26
27         @Override
28         public Integer strengthAlgoritm(int level) {
29             return (int) Math.floor(((level * level) + 5 * level) * ↵
30                 0.5 * DRAGONSTRENGTH);
31         }
32     }), SNAKE("Snake", new Algorithms() {
33
34         @Override
35         public Integer lifeAlgoritm(int level) {
36             return (int) Math.floor((((level + 3) * (level + 3)) - 10)↵
37                 * SNAKELIFE);
38         }
39     })
40 }

```



```

35
36         @Override
37         public Integer strengthAlgoritm(int level) {
38             return (int) Math.floor(((level * level) + 5 * level) * ↵
39                 0.5 * SNAKESTRENGTH);
40         }
41     });
42
43     private static double GOLEMLIFE = 1;
44     private static double GOLEMSTRENGTH = 0.7;
45     private static double DRAGONLIFE = 1.35;
46     private static double DRAGONSTRENGTH = 1;
47     private static double SNAKELIFE = 1;
48     private static double SNAKESTRENGTH = 1;
49
50     private String name;
51     private Algorithms lifeStrengthAlgoritms;
52
53     private MonsterTypes(String name, Algorithms lifeStrengthAlgoritms)↵
54     {
55         this.name = name;
56         this.lifeStrengthAlgoritms = lifeStrengthAlgoritms;
57     }
58
59     public Integer getMaxLife(int level) {
60         return lifeStrengthAlgoritms.lifeAlgoritm(level);
61     }
62
63     public Integer getStrength(int level) {
64         return lifeStrengthAlgoritms.strengthAlgoritm(level);
65     }
66
67     public static MonsterTypes getMonsterType(int data) {
68         switch (data) {
69             case (1):
70                 return MonsterTypes.GOLEM;
71             case (2):
72                 return MonsterTypes.DRAGON;
73             default:
74                 return MonsterTypes.SNAKE;
75         }
76     }
77
78     public String getName() {
79         return name;
80     }

```

1.1.17. MoveTypes.java

```

1 package back;
2
3 public enum MoveTypes implements Strokes{
4     UP(new Point(-1, 0)), DOWN(new Point(1, 0)), LEFT(new Point(0, -1)↵
5     ), RIGHT(
6         new Point(0, 1));
7
8     private Point direction;
9
10    private MoveTypes(Point p){
11        this.direction=p;
12    }
13
14    public Point getDirection(){
15        return direction;
16    }
17
18    public int x(){

```

```
18         return direction.x;
19     }
20
21     public int y(){
22         return direction.y;
23     }
24
25 }
```

1.1.18. Player.java

```
1 package back;
2
3 public class Player extends Character {
4
5     private static Integer EXPERIENCECONSTANT = 5;
6
7     private Integer experience;
8     private Integer experienceToLevelUp;
9     private Integer steps = 0;
10
11     public Player(PlayerData playerData) {
12         super(playerData.getName(), 1, playerData.getPosition());
13         this.experience = 0;
14         this.experienceToLevelUp = EXPERIENCECONSTANT * getLevel();
15         setHealth(playerData.getHealth());
16         setMaxHealth(playerData.getMaxHealth());
17         setStrength(playerData.getStrength());
18     }
19
20     public Player(PlayerData playerData, int level, int steps) {
21         this(playerData);
22         this.steps = steps;
23         setLevel(level);
24     }
25
26     public MoveTypes move(MoveTypes moveType) {
27         setPosition(getPosition().add(moveType.getDirection()));
28         steps++;
29         return moveType;
30     }
31
32     public void winExperience(Integer experience) {
33         if ((this.experience + experience) >= experienceToLevelUp) {
34             levelUp();
35         } else {
36             this.experience += experience;
37         }
38     }
39
40     private void levelUp() {
41         increaseLevel();
42         this.experience = 0;
43         this.experienceToLevelUp = EXPERIENCECONSTANT * getLevel();
44         setMaxHealth(getLevel() * DungeonGameImp.LIFE);
45         setStrength(getStrength() + DungeonGameImp.STRENGTH);
46     }
47
48     public Integer getExperience() {
49         return experience;
50     }
51
52     public void winFight(Character character) {
53         winExperience(character.getLevel());
54     }
55
56     @Override
57     public String toString() {
58         String resp;
```

```

59         resp = super.toString();
60         resp += "Exp=" + experience;
61         resp += "ExpNeeded=" + experienceToLevelUp;
62         return resp;
63     }
64
65     public Integer getSteps() {
66         return steps;
67     }
68
69     public Integer getExperienceToLevelUp() {
70         return experienceToLevelUp;
71     }
72
73     @Override
74     public int hashCode() {
75         final int prime = 31;
76         int result = super.hashCode();
77         result = prime * result
78             + ((experience == null) ? 0 : experience.hashCode());
79         result = prime
80             * result
81             + ((experienceToLevelUp == null) ? 0 : ↵
82                 experienceToLevelUp
83                 .hashCode());
84         result = prime * result + ((steps == null) ? 0 : steps.↵
85             hashCode());
86         return result;
87     }
88
89     @Override
90     public boolean equals(Object obj) {
91         if (this == obj)
92             return true;
93         if (!super.equals(obj))
94             return false;
95         if (getClass() != obj.getClass())
96             return false;
97         Player other = (Player) obj;
98         if (experience == null) {
99             if (other.experience != null)
100                 return false;
101         } else if (!experience.equals(other.experience))
102             return false;
103         if (experienceToLevelUp == null) {
104             if (other.experienceToLevelUp != null)
105                 return false;
106         } else if (!experienceToLevelUp.equals(other.↵
107             experienceToLevelUp))
108             return false;
109         if (steps == null) {
110             if (other.steps != null)
111                 return false;
112         } else if (!steps.equals(other.steps))
113             return false;
114         return true;
115     }
116 }

```

1.1.19. PlayerData.java

```

1 package back;
2
3 public class PlayerData {
4
5     String name;
6     int level;
7     int experience;

```

```
8      int maxHealth;
9      int health;
10     int strength;
11     Point position;
12
13     public PlayerData(String name, int level, int experience, int ←
        health,
14         int maxHealth, int strength, Point position, int steps) {
15         this.name = name;
16         this.experience = experience;
17         this.health = health;
18         this.maxHealth = maxHealth;
19         this.strength = strength;
20         this.position = position;
21     }
22
23
24     public int getExperience() {
25         return experience;
26     }
27
28     public void setExperience(int experience) {
29         this.experience = experience;
30     }
31
32     public int getHealth() {
33         return health;
34     }
35
36     public void setHealth(int health) {
37         this.health = health;
38     }
39
40     public String getName() {
41         return name;
42     }
43
44     public int getMaxHealth() {
45         return maxHealth;
46     }
47
48     public Point getPosition() {
49         return position;
50     }
51
52     public int getStrength() {
53         return strength;
54     }
55
56     public void setName(String name) {
57         this.name = name;
58     }
59
60     public void setMaxHealth(int maxHealth) {
61         this.maxHealth = maxHealth;
62     }
63
64     public void setPosition(Point position) {
65         this.position = position;
66     }
67
68     public void setStrength(int strength) {
69         this.strength = strength;
70     }
71
72 }
```

1.1.20. Point.java

```
1 package back;
2
3 public class Point {
4     public int x;
5     public int y;
6
7     public Point(Point p) {
8         this(p.x, p.y);
9     }
10
11     public Point(int x, int y) {
12         this.x = x;
13         this.y = y;
14     }
15
16     public Point add(Point p) {
17         return new Point(this.x + p.x, this.y + p.y);
18     }
19
20     @Override
21     public String toString() {
22         return "[" + x + ", " + y + "]";
23     }
24
25     @Override
26     public int hashCode() {
27         final int prime = 31;
28         int result = 1;
29         result = prime * result + x;
30         result = prime * result + y;
31         return result;
32     }
33
34     @Override
35     public boolean equals(Object obj) {
36         if (this == obj)
37             return true;
38         if (obj == null)
39             return false;
40         if (getClass() != obj.getClass())
41             return false;
42         Point other = (Point) obj;
43         if (x != other.x)
44             return false;
45         if (y != other.y)
46             return false;
47         return true;
48     }
49
50     public Point sub(Point p) {
51         return new Point(this.x - p.x, this.y - p.y);
52     }
53
54     public Point add(int i, int j) {
55         return add(new Point(i, j));
56     }
57
58     public Point sub(int i, int j) {
59         return sub(new Point(i, j));
60     }
61 }
```

1.1.21. Putable.java

```
1 package back;
2
3 public interface Putable {
4
5     public boolean allowMovement(DungeonGameImp game);
6 }
```

```
6      public void standOver(DungeonGameImp game);
7
8      public boolean isVisible();
9
10     public void setVisible();
11
12     public void setNotVisible();
13
14 }
15
```

1.1.22. SaveGame.java

```
1 package back;
2
3 public interface SaveGame {
4     public void save() throws Exception;
5 }
```

1.1.23. Strokes.java

```
1 package back;
2
3 public interface Strokes {
4
5 }
```

1.1.24. Wall.java

```
1 package back;
2
3 public class Wall extends Cell implements Putable {
4
5     @Override
6     public String toString() {
7         return "Wall";
8     }
9
10    @Override
11    public boolean allowMovement(DungeonGameImp game) {
12        return false;
13    }
14
15    @Override
16    public void standOver(DungeonGameImp game) {}
17
18 }
```

1.2. front

1.2.1. App.java

```

1 package front;
2
3 import javax.swing.JFrame;
4
5 public class App {
6     public static void main(String[] args) {
7         GameFrame dungeonGameFrame = new DungeonGameFrame();
8         dungeonGameFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9         dungeonGameFrame.setVisible(true);
10    }
11 }

```

1.2.2. DataPanel.java

```

1 package front;
2
3 import java.awt.Color;
4 import java.awt.Font;
5 import java.util.HashMap;
6 import java.util.Map;
7
8 import javax.swing.BoxLayout;
9 import javax.swing.JLabel;
10 import javax.swing.JPanel;
11
12 import back.Game;
13 import back.Monster;
14 import back.Player;
15 import back.Point;
16 import back.Putable;
17
18 /**
19  * @author tmehdi Class that extends the class JPanel. This class is
20  *     used for
21  *     the Dungeon panel that is into the DungeonGameFrame.
22  */
23 public class DataPanel extends JPanel {
24
25     private static final long serialVersionUID = 1L;
26
27     @SuppressWarnings("unused")
28     private JLabel[] playerLabel;
29     private Map<Monster, JLabel[]> monstersLabels = new HashMap<
30         Monster, JLabel[]>();
31
32     public DataPanel(Player player, Color color) {
33         setBackground(Color.WHITE);
34         setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
35         addCharacter(player);
36     }
37
38     public void addCharacter(Player character) {
39         JLabel[] playerLabel = new JLabel[6];
40         playerLabel[0] = new JLabel(" " + character.getName());
41         playerLabel[0].setFont(new Font("Serif", Font.BOLD, 16));
42         playerLabel[0].setForeground(Color.BLUE);
43         playerLabel[1] = new JLabel(" " + "Health: " + character.
44             getHealth()
45             + "/" + character.getMaxHealth());
46         playerLabel[2] = new JLabel(" " + "Strength: "
47             + character.getStrength());
48         playerLabel[3] = new JLabel(" " + "Level: " + character.
49             getLevel());
50         playerLabel[4] = new JLabel(" " + "Experience: "
51             + character.getExperience() + "/"
52             + character.getExperienceToLevelUp() + " ");

```

```

50         playerLabel[5] = new JLabel(" ");
51         this.playerLabel = playerLabel;
52         for (JLabel pl : playerLabel) {
53             add(pl);
54         }
55     }
56
57     public void addCharacter(Monster character) {
58         JLabel[] playerLabel = new JLabel[4];
59         playerLabel[0] = new JLabel(" " + character.getName());
60         playerLabel[0].setFont(new Font("Serif", Font.BOLD, 12));
61         playerLabel[0].setForeground(Color.RED);
62         playerLabel[1] = new JLabel(" " + "Health: " + character.↵
            getHealth()
63             + "/" + character.getMaxHealth());
64         playerLabel[2] = new JLabel(" " + "Strength: "
65             + character.getStrength());
66         playerLabel[3] = new JLabel(" " + "Level: " + character.↵
            getLevel());
67         for (JLabel pl : playerLabel) {
68             add(pl);
69         }
70         monstersLabels.put(character, playerLabel);
71     }
72
73     public void removeCharacter(Monster character) {
74         JLabel[] labels = monstersLabels.get(character);
75         for (JLabel ml : labels) {
76             remove(ml);
77         }
78     }
79
80     public void refresh(Game game, DungeonPanel dungeonPanel) {
81         Putable[] possibleMonsters = new Putable[5];
82         Point p = game.getPlayer().getPosition();
83
84         possibleMonsters[0] = game.getBoard()[p.x + 1][p.y];
85         possibleMonsters[1] = game.getBoard()[p.x - 1][p.y];
86         possibleMonsters[2] = game.getBoard()[p.x][p.y + 1];
87         possibleMonsters[3] = game.getBoard()[p.x][p.y - 1];
88         possibleMonsters[4] = dungeonPanel.getMonsterUnderMouse();
89
90         removeAll();
91
92         for (int i = 0; possibleMonsters[4] != null && i < 4; i++) {
93             if (possibleMonsters[4].equals(possibleMonsters[i])) {
94                 possibleMonsters[4] = null;
95             }
96         }
97
98         addCharacter(game.getPlayer());
99         for (Putable put : possibleMonsters) {
100             if (put != null && put instanceof Monster) {
101                 addCharacter((Monster) put);
102             }
103         }
104     }
105 }
106 }

```

1.2.3. DataPanelListener.java

```

1 package front;
2
3
4 public interface DataPanelListener {
5
6 }

```


1.2.4. DefaultGameMenuBar.java

```
1 package front;
2
3 import java.awt.event.ActionListener;
4
5 public interface DefaultGameMenuBar {
6
7     public void setNewGameItemAction(ActionListener a);
8
9     public void setRestartGameItemAction(ActionListener a);
10
11     public void setSaveGameItemAction(ActionListener a);
12
13     public void setSaveGameAsItemAction(ActionListener a);
14
15     public void setLoadGameItemAction(ActionListener a);
16
17     public void setExitGameItemAction(ActionListener a);
18
19     public void createDefaultJMenuActionListeners();
20
21 }
```

1.2.5. DungeonGameFrame.java

```
1 package front;
2
3 import static professorShipSrc.ImageUtils.loadImage;
4
5 import java.awt.BorderLayout;
6 import java.awt.Color;
7 import java.awt.event.ActionEvent;
8 import java.awt.event.ActionListener;
9 import java.awt.event.KeyAdapter;
10 import java.awt.event.KeyEvent;
11 import java.io.File;
12 import java.io.IOException;
13
14 import javax.swing.JFileChooser;
15 import javax.swing.JOptionPane;
16
17 import parser.BoardParserFromFile;
18 import parser.CorruptedFileException;
19 import saveLoadImplementation.LoadGameFromFile;
20 import saveLoadImplementation.SaveGameOnFile;
21 import saveLoadImplementation.SavingCorruptedException;
22 import back.BoardObtainer;
23 import back.DungeonGameImp;
24 import back.DungeonGameListener;
25 import back.LoadGame;
26 import back.Monster;
27 import back.MoveTypes;
28 import back.Point;
29 import back.Putable;
30
31 /**
32  * @author tmehdi Class that extends GameFrame. It's used for the ↵
33  * frame of the
34  * game.
35  */
36 public class DungeonGameFrame extends GameFrame {
37
38     private static final long serialVersionUID = 1L;
39     private DataPanel dataPanel;
40     private DungeonPanel dungeonPanel;
```

```

40
41 public DungeonGameFrame() {
42     super("Dungeon game");
43     setIcon();
44     addKeyListener();
45 }
46
47 /**
48  * DungeonGameFrame menu. It have 6 options: New game, Restart, ↵
49  * Save game,
50  * Save game as..., Load game and Exit
51  * @see front.GameFrame#createDefaultJMenuActionListeners()
52  */
53 @Override
54 public void createDefaultJMenuActionListeners() {
55
56     setNewGameItemAction(new ActionListener() {
57         @Override
58         public void actionPerformed(ActionEvent e) {
59             try {
60                 if (game != null) {
61                     dataPanel.setVisible(false);
62                     dungeonPanel.setVisible(false);
63                     remove(dataPanel);
64                     remove(dungeonPanel);
65                     repaint();
66                     game = null;
67                 }
68                 File file = null;
69                 LevelSelector levelSelector = new LevelSelectorImp(
70                     DungeonGameFrame.this);
71                 file = levelSelector.getLevelSelected();
72                 if (file != null) {
73                     BoardObtainer boardObtainer = new ↵
74                         BoardParserFromFile(
75                             file);
76                     game = new DungeonGameImp(boardObtainer,
77                         new DungeonGameListenerImp());
78                     drawDungeonPanel();
79                     drawDataPanel();
80                     dataPanel.refresh(game, dungeonPanel);
81                     dungeonPanel.updateUI();
82                 }
83             } catch (Exception e1) {
84                 JOptionPane.showMessageDialog(null,
85                     "Level file is corrupt", "Error",
86                     JOptionPane.ERROR_MESSAGE);
87             }
88         });
89
90     setRestartGameItemAction(new ActionListener() {
91         @Override
92         public void actionPerformed(ActionEvent e) {
93             try {
94                 if (game == null) {
95                     JOptionPane.showMessageDialog(null,
96                         "You are not playing a level.");
97                 } else {
98                     game.restart();
99                     dataPanel.setVisible(false);
100                     dungeonPanel.setVisible(false);
101                     remove(dataPanel);
102                     remove(dungeonPanel);
103                     drawDungeonPanel();
104                     drawDataPanel();
105                     dataPanel.refresh(game, dungeonPanel);
106                     dungeonPanel.updateUI();
107                 }
108             } catch (CorruptedFileException e1) {
109                 JOptionPane.showMessageDialog(null, "The file is ↵
110                     corrupt",

```

```

110         "Error", JOptionPane.ERROR_MESSAGE);
111     }
112 }
113 });
114
115 setSaveGameItemAction(new ActionListener() {
116
117     @Override
118     public void actionPerformed(ActionEvent e) {
119         if (game != null) {
120             File directory = new File(".") + File.separator
121                 + "savedGames";
122             if (!directory.exists()) {
123                 directory.mkdir();
124             }
125             try {
126                 new SaveGameOnFile(game);
127             } catch (SavingCorruptedException e1) {
128                 JOptionPane.showMessageDialog(null,
129                     "Files saving error occurs. Try again ↵
130                     later.",
131                     "Error", JOptionPane.ERROR_MESSAGE);
132             }
133         }
134     });
135
136 setSaveGameAsItemAction(new ActionListener() {
137
138     @Override
139     public void actionPerformed(ActionEvent e) {
140         if (game != null) {
141             File directory = new File(".") + File.separator
142                 + "savedGames";
143             if (!directory.exists()) {
144                 directory.mkdir();
145             }
146             File file;
147             JFileChooser fc = new JFileChooser();
148             fc.setCurrentDirectory(new File(".") + File.separator
149                 + "savedGames");
150             fc.showOpenDialog(DungeonGameFrame.this);
151             file = fc.getSelectedFile();
152             if (file == null) {
153                 JOptionPane.showMessageDialog(null,
154                     "You didn't select any file.");
155             } else {
156                 try {
157                     new SaveGameOnFile(game, file);
158                 } catch (SavingCorruptedException e1) {
159                     JOptionPane
160                         .showMessageDialog(
161                             null,
162                             "Files saving error ↵
163                             occurs. Try again ↵
164                             later.",
165                             "Error", JOptionPane.ERROR_MESSAGE);
166                 }
167             }
168         }
169     });
170
171 setLoadGameItemAction(new ActionListener() {
172
173     @Override
174     public void actionPerformed(ActionEvent e) {
175         if (game != null) {
176             dataPanel.setVisible(false);
177             dungeonPanel.setVisible(false);
178             remove(dataPanel);
179             remove(dungeonPanel);
180             repaint();

```

```

179         game = null;
180     }
181     File file;
182     JFileChooser fc = new JFileChooser();
183     fc.setCurrentDirectory(new File(".") + File.separator
184         + "savedGames");
185     fc.showOpenDialog(DungeonGameFrame.this);
186     file = fc.getSelectedFile();
187     if (file == null) {
188         JOptionPane.showMessageDialog(null,
189             "You didn't select any file.");
190     } else {
191         try {
192             LoadGame<DungeonGameImp> loadGame = new ↵
193                 LoadGameFromFile<DungeonGameImp>(
194                     file);
195             game = loadGame.getGame(DungeonGameImp.class,
196                 new DungeonGameListenerImp());
197             drawDungeonPanel();
198             drawDataPanel();
199             dataPanel.updateUI();
200             dungeonPanel.updateUI();
201         } catch (CorruptedFileException e2) {
202             JOptionPane
203                 .showMessageDialog(
204                     null,
205                     "Files loading error occurs. ↵
206                     Try again later.",
207                     "Error", JOptionPane.↵
208                         ERROR_MESSAGE);
209         }
210     }
211     setExitGameItemAction(new ActionListener() {
212         @Override
213         public void actionPerformed(ActionEvent e) {
214             try {
215                 DungeonGameFrame.this.setVisible(false);
216                 DungeonGameFrame.this.dispose();
217             } catch (Throwable e1) {
218                 JOptionPane.showMessageDialog(null, "Exit fault", ↵
219                     "Error",
220                     JOptionPane.ERROR_MESSAGE);
221             }
222         }
223     });
224 }
225
226 /**
227  * Method to make appear the data panel.
228  */
229 private void drawDataPanel() {
230     dataPanel = new DataPanel(game.getPlayer(), Color.GRAY);
231     add(dataPanel, BorderLayout.EAST);
232 }
233
234 /**
235  * Method to make appear the dungeon panel.
236  */
237 private void drawDungeonPanel() {
238     dungeonPanel = new DungeonPanel(game, dataPanel,
239         new DungeonPanelListenerImp());
240     add(dungeonPanel, BorderLayout.CENTER);
241 }
242
243 /**
244  * Getter of the dungeon panel.
245  * @return DungeonPanel
246  */
247 public DungeonPanel getDungeonPanel() {

```

```

249         return dungeonPanel;
250     }
251
252     /**
253      * Getter of the data panel.
254      *
255      * @return DataPanel
256      */
257     public DataPanel getDataPanel() {
258         return dataPanel;
259     }
260
261     /**
262      * Listener of the move keys, up down left right.
263      *
264      * @see front.GameFrame#addKeyListener()
265      */
266     @Override
267     public void addKeyListener() {
268
269         addKeyListener(new KeyAdapter() {
270
271             @Override
272             public void keyPressed(final KeyEvent e) {
273                 switch (e.getKeyCode()) {
274                     case KeyEvent.VK_LEFT:
275                         game.receiveMoveStroke(MoveTypes.LEFT);
276
277                         break;
278                     case KeyEvent.VK_UP:
279                         game.receiveMoveStroke(MoveTypes.UP);
280
281                         break;
282                     case KeyEvent.VK_RIGHT:
283                         game.receiveMoveStroke(MoveTypes.RIGHT);
284
285                         break;
286                     case KeyEvent.VK_DOWN:
287                         game.receiveMoveStroke(MoveTypes.DOWN);
288
289                         break;
290                 }
291             }
292         });
293     }
294
295     /**
296      * @author tmehdi Inner class for the listener of this game ↵
297      * implementation.
298      */
299     private class DungeonGameListenerImp implements ↵
300         DungeonGameListener {
301
302         @Override
303         public void executeWhenBonusGrabed(Point p) {
304             dungeonPanel.drawGrabedBonus(p);
305         }
306
307         @Override
308         public void executeWhenCharacterDie(Point p) {
309             dungeonPanel.drawDiedCharacter(p);
310         }
311
312         @Override
313         public void executeWhenGameLoosed() {
314             JOptionPane.showMessageDialog(DungeonGameFrame.this,
315                 "You loose the level.");
316             DungeonGameFrame.this.remove(DungeonGameFrame.this
317                 .getDungeonPanel());
318             DungeonGameFrame.this.remove(DungeonGameFrame.this.↵
319                 getDataPanel());
320             repaint();
321         }
322     }

```

```

320     @Override
321     public void executeWhenGameWon() {
322         JOptionPane.showMessageDialog(DungeonGameFrame.this, "↵
323             WINNER!"
324             + '\n' + "You win the level with "
325             + game.getPlayer().getSteps() + " steps.");
326         DungeonGameFrame.this.remove(DungeonGameFrame.this
327             .getDungeonPanel());
328         DungeonGameFrame.this.remove(DungeonGameFrame.this.↵
329             getDataPanel());
330         repaint();
331     }
332
333     @Override
334     public void executeWhenPlayerMoves(MoveTypes moveType) {
335         dungeonPanel.drawPlayerMove(game, moveType);
336         dataPanel.refresh(game, dungeonPanel);
337         dataPanel.updateUI();
338         dungeonPanel.drawDiscoveredCell(game, moveType);
339     }
340
341     @Override
342     public String playerNameRequest() {
343         String name = null;
344         while (name == null || name.isEmpty()) {
345             name = JOptionPane.showInputDialog("Player name");
346         }
347         return name;
348     }
349
350     @Override
351     public void executeWhenFight() {
352         dataPanel.refresh(game, dungeonPanel);
353         dataPanel.updateUI();
354     }
355
356     @Override
357     public void executeWhenLevelUp() {
358         dungeonPanel.drawLevelUp(game);
359     }
360 }
361
362 /**
363  * Add the hero image as frame icon.
364  */
365 private void setIcon() {
366     try {
367         setIconImage(loadImage("../resources/images/hero.png"));
368     } catch (IOException e) {
369         JOptionPane.showMessageDialog(null, "Unexpected Error", "↵
370             Error",
371             JOptionPane.ERROR_MESSAGE);
372     }
373 }
374
375 /**
376  * @author tomas Implementation of DungeonPanelListener used for ↵
377  * the actions
378  * performed on dungeonPanel with the mouse.
379  */
380 private class DungeonPanelListenerImp implements ↵
381     DungeonPanelListener {
382
383     @Override
384     public void onMouseMoved(int row, int column) {
385
386         Monster monster = dungeonPanel.getMonsterUnderMouse();
387         if (monster != null) {
388             dataPanel.removeCharacter(monster);
389             dungeonPanel.setMonsterUnderMouse(null);
390         }
391         Putable putable = game.getBoard()[row + 1][column + 1];
392         if (putable instanceof Monster && putable.isVisible()) {
393             dungeonPanel.setMonsterUnderMouse((Monster) putable);
394         }
395     }
396 }

```

```

389         dataPanel.addCharacter(dungeonPanel.↵
390             getMonsterUnderMouse());
391     }
392     dataPanel.refresh(game, dungeonPanel);
393     dataPanel.updateUI();
394 }
395 }
396 }
397 }

```

1.2.6. DungeonPanel.java

```

1  package front;
2
3  import static professorShipSrc.ImageUtils.drawString;
4  import static professorShipSrc.ImageUtils.loadImage;
5  import static professorShipSrc.ImageUtils.overlap;
6
7  import java.awt.Color;
8  import java.awt.Image;
9  import java.io.IOException;
10 import java.util.ArrayList;
11 import java.util.HashMap;
12 import java.util.List;
13 import java.util.Map;
14
15 import javax.swing.JOptionPane;
16
17 import professorShipSrc.GamePanel;
18 import back.BloodyFloor;
19 import back.Bonus;
20 import back.Character;
21 import back.Floor;
22 import back.Game;
23 import back.Monster;
24 import back.MoveTypes;
25 import back.Point;
26 import back.Putable;
27 import back.Wall;
28
29 /**
30  * @author tmehdi Class that extends the professor ship class ↵
31  *   GamePanel. This
32  *   class is used for the Dungeon panel that is into the
33  *   DungeonGameFrame.
34  */
35 public class DungeonPanel extends GamePanel {
36
37     private static final long serialVersionUID = 1L;
38     private static final int CELL_SIZE = 30;
39
40     private Image playerImage;
41     private Map<Class<? extends Putable>, Image> boardImagesByClass = ↵
42         new HashMap<Class<? extends Putable>, Image>();
43     private Map<String, Image> monsterImagesByName = new HashMap<↵
44         String, Image>();
45     private Map<String, Image> bonusImagesByName = new HashMap<String, ↵
46         Image>();
47     private Monster monsterUnderMouse = null;
48
49     /**
50      * @param game
51      * @param dataPanel
52      * @param dungeonListener
53      * Call the super constructor and draw the pane. The interface
54      * DungeonPanelListener that extends the professor ship ↵
55      * interface

```

```

51      *      GamePanelListener is used to make an implementation ←
52      *      of the "onMouseClicked" method. It allows us to know in what ←
53      *      cell is and make the different actions.
54      */
55      public DungeonPanel(Game game, DataPanel dataPanel, ←
56          DungeonPanelListener dungeonListener) {
57          super(game.getBoardDimension().x - 2,
58              game.getBoardDimension().y - 2, CELL_SIZE, ←
59              Color.BLACK);
60          playerImage();
61          boardImagesByClass();
62          monstersImagesInitialize();
63          bonusImagesInitialize();
64          drawDungeon(game);
65          setVisible(true);
66      }
67      /**
68      * @param monsterUnderMouse
69      *      Setter of the monster under mouse.
70      */
71      public void setMonsterUnderMouse(Monster monsterUnderMouse) {
72          this.monsterUnderMouse = monsterUnderMouse;
73      }
74      /**
75      * @param dungeonGameFrame
76      *      Draw the full dungeon panel.
77      */
78      public void drawFullDungeon(DungeonGameFrame dungeonGameFrame) {
79          Image image;
80          Image floorImage = boardImagesByClass.get(Floor.class);
81          Image bloodyFloorImage = overlap(floorImage,
82              boardImagesByClass.get(BloodyFloor.class));
83          int row = dungeonGameFrame.game.getBoardDimension().x - 2;
84          int col = dungeonGameFrame.game.getBoardDimension().y - 2;
85
86          for (int i = 1; i <= row; i++) {
87              for (int j = 1; j <= col; j++) {
88                  Putable cell = dungeonGameFrame.game.getBoard()[i][j];
89                  if (cell.getClass().equals(Monster.class)) {
90                      image = monsterImagesByName.get(((Monster) cell)
91                          .getMonsterType().toString());
92                      image = overlap(floorImage, image);
93                      image = drawString(image, ((Character) cell).←
94                          getLevel()
95                          .toString(), Color.WHITE);
96                      put(image, i - 1, j - 1);
97                  } else if (cell.getClass().equals(Bonus.class)) {
98                      image = bonusImagesByName.get(((Bonus) cell).←
99                          getBonusType()
100                          .toString());
101                      image = overlap(floorImage, image);
102                      image = drawString(image,
103                          (((Bonus) cell).getBonusType().←
104                          getBonusAmount()
105                          .toString(), Color.RED);
106                      put(image, i - 1, j - 1);
107                  } else {
108                      image = boardImagesByClass.get(cell.getClass());
109                      if (cell.getClass().equals(Wall.class)) {
110                          put(image, i - 1, j - 1);
111                      } else if (cell.getClass().equals(BloodyFloor.←
112                          class)) {
113                          put(bloodyFloorImage, i - 1, j - 1);
114                      } else {
115                          put(floorImage, i - 1, j - 1);
116                      }
117                  }
118              }
119          }
120      }

```



```

117         Point p = new Point(dungeonGameFrame.game.getPlayer().↵
118             getPosition());
119
120         if (dungeonGameFrame.game.getBoard()[p.x][p.y] instanceof ↵
121             BloodyFloor) {
122             image = overlap(bloodyFloorImage, playerImage);
123         }
124         image = overlap(floorImage, playerImage);
125         image = drawString(image, dungeonGameFrame.game.getPlayer().↵
126             getLevel()
127             .toString(), Color.WHITE);
128         put(image, p.x - 1, p.y - 1);
129     }
130
131     /**
132     * @param dungeonGameFrame
133     *
134     *         Draw the dungeon panel when a game begins.
135     */
136     private void drawDungeon(Game game) {
137         drawRestOfDungeon(game);
138         drawDungeonArroundPlayer(game);
139     }
140
141     /**
142     * @param dungeonGameFrame
143     *
144     *         Draw all the visible cells (it's just for loaded ↵
145     *         games in this
146     *         game implementation)
147     */
148     private void drawRestOfDungeon(Game game) {
149         Image image;
150         List<Point> points = new ArrayList<Point>();
151         Image floorImage = boardImagesByClass.get(Floor.class);
152         Image bloodyFloorImage = overlap(floorImage,
153             boardImagesByClass.get(BloodyFloor.class));
154
155         int row = game.getBoardDimension().x - 2;
156         int col = game.getBoardDimension().y - 2;
157
158         for (int i = 1; i <= row; i++) {
159             for (int j = 1; j <= col; j++) {
160                 Putable cell = game.getBoard()[i][j];
161                 if (cell.isVisible() && cell.getClass().equals(Monster↵
162                     .class)) {
163                     image = monsterImagesByName.get(((Monster) cell)
164                         .getMonsterType().toString());
165                     image = overlap(floorImage, image);
166                     image = drawString(image, ((Character) cell).↵
167                         getLevel()
168                         .toString(), Color.WHITE);
169                     put(image, i - 1, j - 1);
170                     points.add(new Point(i, j));
171                 } else if (cell.isVisible() && cell.getClass().equals(Bonus.class)) {
172                     image = bonusImagesByName.get(((Bonus) cell).↵
173                         getBonusType()
174                         .toString());
175                     image = overlap(floorImage, image);
176                     image = drawString(image,
177                         (((Bonus) cell).getBonusType().↵
178                             getBonusAmount()
179                             .toString(), Color.RED);
180                     put(image, i - 1, j - 1);
181                     points.add(new Point(i, j));
182                 } else {
183                     if (cell.isVisible() && cell.getClass().equals(Wall.class)) {
184                         image = boardImagesByClass.get(cell.getClass()↵
185                             );
186                         put(image, i - 1, j - 1);
187                         points.add(new Point(i, j));
188                     }
189                 }
190             }
191         }
192     }

```

```

181         } else if (cell.isVisible()
182             && cell.getClass().equals(BloodyFloor.class)) {
183             put(bloodyFloorImage, i - 1, j - 1);
184             points.add(new Point(i, j));
185         } else if (cell.isVisible()) {
186             put(floorImage, i - 1, j - 1);
187             points.add(new Point(i, j));
188         }
189     }
190 }
191 }
192 }
193 }
194 }
195 /**
196  * @param dungeonGameFrame
197  * Draw the 8 cells around the player and the cell
198  * under the player. Before that draw the player
199  */
200 private void drawDungeonArroundPlayer(Game game) {
201     Image image;
202     Image floorImage = boardImagesByClass.get(Floor.class);
203     Image bloodyFloorImage = overlap(floorImage,
204         boardImagesByClass.get(BloodyFloor.class));
205
206     Point pPos = game.getPlayer().getPosition();
207     pPos = pPos.sub(2, 2);
208
209     for (int i = 1; i <= 3; i++) {
210         for (int j = 1; j <= 3; j++) {
211             Putable cell = game.getBoard()[pPos.x + i][pPos.y
212                 + j];
213             if (cell.getClass().equals(Monster.class)) {
214                 image = monsterImagesByName.get(((Monster) cell)
215                     .getMonsterType().toString());
216                 image = overlap(floorImage, image);
217                 image = drawString(image, ((Character) cell).
218                     getLevel()
219                     .toString(), Color.WHITE);
220                 put(image, pPos.x + i - 1, pPos.y + j - 1);
221             } else if (cell.getClass().equals(Bonus.class)) {
222                 image = bonusImagesByName.get(((Bonus) cell).
223                     getBonusType()
224                     .toString());
225                 image = overlap(floorImage, image);
226                 image = drawString(image,
227                     (((Bonus) cell).getBonusType().
228                     getBonusAmount()
229                     .toString(), Color.RED);
230                 put(image, pPos.x + i - 1, pPos.y + j - 1);
231             } else {
232                 image = boardImagesByClass.get(cell.getClass());
233                 if (cell.getClass().equals(Wall.class)) {
234                     put(image, pPos.x + i - 1, pPos.y + j - 1);
235                 } else if (cell.getClass().equals(BloodyFloor.class)) {
236                     put(bloodyFloorImage, pPos.x + i - 1, pPos.y +
237                         j - 1);
238                 } else {
239                     put(floorImage, pPos.x + i - 1, pPos.y + j -
240                         1);
241                 }
242             }
243         }
244     }
245
246     Point p = new Point(game.getPlayer().getPosition());
247
248     if (game.getBoard()[p.x][p.y] instanceof BloodyFloor) {
249         image = overlap(bloodyFloorImage, playerImage);
250     }
251     image = overlap(floorImage, playerImage);

```

```

247         image = drawString(image, game.getPlayer().getLevel()
248             .toString(), Color.WHITE);
249         put(image, p.x - 1, p.y - 1);
250     }
251
252     /**
253      * @return Getter of the monsterUnderMouse.
254      */
255     public Monster getMonsterUnderMouse() {
256         return monsterUnderMouse;
257     }
258
259     /**
260      * @param game of class Game
261      * @param moveType instance of enumerative MoveTypes
262      *
263      * Redraw if necessary the DungeonPanel.
264      */
265     public void drawPlayerMove(Game game,
266         MoveTypes moveType) {
267         Image bloodyFloor;
268         Image floor;
269         Point afterMove = new Point(game.getPlayer()
270             .getPosition().x, game.getPlayer()
271             .getPosition().y);
272         Point beforeMove = afterMove.sub(moveType.getDirection());
273         floor = boardImagesByClass.get(Floor.class);
274         bloodyFloor = boardImagesByClass.get(BloodyFloor.class);
275         bloodyFloor = overlap(floor, bloodyFloor);
276         clear(beforeMove.x - 1, beforeMove.y - 1);
277         if (game.getBoard()[beforeMove.x][beforeMove.y]
278             .getClass().equals(BloodyFloor.class)) {
279             put(bloodyFloor, beforeMove.x - 1, beforeMove.y - 1);
280         } else {
281             put(floor, beforeMove.x - 1, beforeMove.y - 1);
282         }
283
284         clear(afterMove.x - 1, afterMove.y - 1);
285         Image image;
286         if (game.getBoard()[afterMove.x][afterMove.y]
287             .getClass().equals(BloodyFloor.class)) {
288             image = overlap(bloodyFloor, playerImage);
289             image = drawString(image, game.getPlayer()
290                 .getLevel().toString(), Color.WHITE);
291             put(image, afterMove.x - 1, afterMove.y - 1);
292         } else {
293             image = overlap(floor, playerImage);
294             image = drawString(image, game.getPlayer()
295                 .getLevel().toString(), Color.WHITE);
296
297             put(image, afterMove.x - 1, afterMove.y - 1);
298         }
299         updateUI();
300     }
301
302     /**
303      * @param p
304      *
305      * Draw blood on the floor where a character die.
306      */
307     public void drawDiedCharacter(Point p) {
308         Image imagFloor = boardImagesByClass.get(Floor.class);
309         Image imagBloodFloor = boardImagesByClass.get(BloodyFloor.
310             class);
311         clear(p.x - 1, p.y - 1);
312         put(overlap(imagFloor, imagBloodFloor), p.x - 1, p.y - 1);
313         repaint();
314     }
315
316     /**
317      * @param p
318      *
319

```

```

320         *           Remove the image of the bonus and draw a floor.
321         */
322     public void drawGrabedBonus(Point p) {
323         Image floor = boardImagesByClass.get(Floor.class);
324         clear(p.x - 1, p.y - 1);
325         put(overlap(floor, playerImage), p.x - 1, p.y - 1);
326         repaint();
327     }
328 }
329
330 public void drawDiscoveredCell(Game game,
331     MoveTypes dir) {
332     Point pPos = game.getPlayer().getPosition();
333     List<Point> points = new ArrayList<Point>();
334     points.add(pPos.add(dir.getDirection()));
335     if (dir == MoveTypes.LEFT || dir == MoveTypes.RIGHT) {
336         points.add(pPos.add(1, 0).add(dir.getDirection()));
337         points.add(pPos.sub(1, 0).add(dir.getDirection()));
338     } else {
339         points.add(pPos.add(0, 1).add(dir.getDirection()));
340         points.add(pPos.sub(0, 1).add(dir.getDirection()));
341     }
342
343     Image image;
344     Image floorImage = boardImagesByClass.get(Floor.class);
345     Image bloodyFloorImage = overlap(floorImage,
346         boardImagesByClass.get(BloodyFloor.class));
347
348     for (Point p : points) {
349         if (game.getBoard()[p.x][p.y].isVisible()) {
350             game.getBoard()[p.x][p.y].setVisible();
351             Putable cell = game.getBoard()[p.x][p.y];
352             if (cell.getClass().equals(Monster.class)) {
353                 image = monsterImagesByName.get(((Monster) cell)
354                     .getMonsterType().toString());
355                 image = overlap(floorImage, image);
356                 image = drawString(image, ((Character) cell).←
357                     getLevel()
358                     .toString(), Color.WHITE);
359                 put(image, p.x - 1, p.y - 1);
360             } else if (cell.getClass().equals(Bonus.class)) {
361                 image = bonusImagesByName.get(((Bonus) cell).←
362                     getBonusType()
363                     .toString());
364                 image = overlap(floorImage, image);
365                 image = drawString(image,
366                     (((Bonus) cell).getBonusType().←
367                     getBonusAmount()
368                     .toString(), Color.RED);
369                 put(image, p.x - 1, p.y - 1);
370             } else {
371                 image = boardImagesByClass.get(cell.getClass());
372                 if (cell.getClass().equals(Wall.class)) {
373                     put(image, p.x - 1, p.y - 1);
374                 } else if (cell.getClass().equals(BloodyFloor.←
375                     class)) {
376                     put(bloodyFloorImage, p.x - 1, p.y - 1);
377                 } else {
378                     put(floorImage, p.x - 1, p.y - 1);
379                 }
380             }
381         }
382     }
383 }
384
385 /**
386  * Method to initialize player image.
387  */
388 private void playerImage() {
389     try {
390         playerImage = loadImage("../resources/images/hero.png");
391     } catch (IOException e) {

```

```

389         JOptionPane.showMessageDialog(null, "Unexpected Error", "←
390         Error",
391         JOptionPane.ERROR_MESSAGE);
392     }
393
394     /**
395     * Method to initialize board images.
396     */
397     private void boardImagesByClass() {
398         try {
399             boardImagesByClass.put(Wall.class,
400                 loadImage("./resources/images/wall.png"));
401             boardImagesByClass.put(Floor.class,
402                 loadImage("./resources/images/background.png"));
403             boardImagesByClass.put(BloodyFloor.class,
404                 loadImage("./resources/images/blood.png"));
405         } catch (IOException e) {
406             JOptionPane.showMessageDialog(null, "Unexpected Error", "←
407             Error",
408             JOptionPane.ERROR_MESSAGE);
409         }
410     }
411
412     /**
413     * Method to initialize bonus images.
414     */
415     private void bonusImagesInitialize() {
416         try {
417             bonusImagesByName.put("LIFE",
418                 loadImage("./resources/images/healthBoost.png"));
419             bonusImagesByName.put("STRENGTH",
420                 loadImage("./resources/images/attackBoost.png"));
421         } catch (IOException e) {
422             JOptionPane.showMessageDialog(null, "Unexpected Error", "←
423             Error",
424             JOptionPane.ERROR_MESSAGE);
425         }
426     }
427
428     /**
429     * Method to initialize monsters images.
430     */
431     private void monstersImagesInitialize() {
432         try {
433             monsterImagesByName.put("GOLEM",
434                 loadImage("./resources/images/golem.png"));
435             monsterImagesByName.put("DRAGON",
436                 loadImage("./resources/images/dragon.png"));
437             monsterImagesByName.put("SNAKE",
438                 loadImage("./resources/images/serpent.png"));
439         } catch (IOException e) {
440             JOptionPane.showMessageDialog(null, "Unexpected Error", "←
441             Error",
442             JOptionPane.ERROR_MESSAGE);
443         }
444     }
445
446     public void drawLevelUp(Game game) {
447         Image image;
448         Image bloodyFloor;
449         Image floor;
450         Point playerPos = new Point(game.getPlayer()
451             .getPosition().x, game.getPlayer()
452             .getPosition().y);
453         floor = boardImagesByClass.get(Floor.class);
454         bloodyFloor = boardImagesByClass.get(BloodyFloor.class);
455         bloodyFloor = overlap(floor, bloodyFloor);
456
457         clear(playerPos.x - 1, playerPos.y - 1);
458         if (game.getBoard()[playerPos.x][playerPos.y] instanceof ←
459             BloodyFloor) {
460             image = overlap(bloodyFloor, playerImage);
461             image = drawString(image, game.getPlayer()

```

```

458         .getLevel().toString(), Color.WHITE);
459         put(image, playerPos.x - 1, playerPos.y - 1);
460     } else {
461         image = overlap(floor, playerImage);
462         image = drawString(image, game.getPlayer()
463             .getLevel().toString(), Color.WHITE);
464
465         put(image, playerPos.x - 1, playerPos.y - 1);
466     }
467     updateUI();
468 }
469
470 }

```

1.2.7. DungeonPanelListener.java

```

1 package front;
2
3 import professorShipSrc.GamePanelListener;
4
5 public interface DungeonPanelListener extends GamePanelListener {
6
7 }

```

1.2.8. GameFrame.java

```

1 package front;
2
3 import java.awt.event.ActionListener;
4 import java.awt.event.InputEvent;
5
6 import javax.swing.JFrame;
7 import javax.swing.JMenu;
8 import javax.swing.JMenuBar;
9 import javax.swing.JMenuItem;
10 import javax.swing.KeyStroke;
11
12 import back.Game;
13
14 public abstract class GameFrame extends JFrame implements ↵
15     DefaultGameMenuBar {
16
17     private static final long serialVersionUID = 1L;
18     private static final int CELL_SIZE = 30;
19     public Game game;
20     private JMenuBar menuBar;
21     private JMenu fileMenu;
22     private JMenuItem newGameItem;
23     private JMenuItem restartGameItem;
24     private JMenuItem saveGameItem;
25     private JMenuItem saveGameAsItem;
26     private JMenuItem loadGameItem;
27     private JMenuItem exitGameItem;
28
29     public GameFrame(String name) {
30         super(name);
31         setTitle(name);
32         setSize(13 * CELL_SIZE + 26, 11 * CELL_SIZE + 20);
33         menuBar = new JMenuBar();
34         fileMenu = new JMenu("File");
35         newGameItem = fileMenu.add("New game");
36         restartGameItem = fileMenu.add("Restart");
37         loadGameItem = fileMenu.add("Load game");

```

```

37         saveGameItem = fileMenu.add("Save game");
38         saveGameAsItem = fileMenu.add("Save game as ...");
39         exitGameItem = fileMenu.add("Exit");
40
41         newGameItem.setAccelerator(KeyStroke.getKeyStroke('N',
42             InputEvent.CTRL_DOWN_MASK));
43
44         restartGameItem.setAccelerator(KeyStroke.getKeyStroke('R',
45             InputEvent.CTRL_DOWN_MASK));
46
47         saveGameItem.setAccelerator(KeyStroke.getKeyStroke('S',
48             InputEvent.CTRL_DOWN_MASK));
49
50         saveGameAsItem.setAccelerator(KeyStroke.getKeyStroke('D',
51             InputEvent.CTRL_DOWN_MASK));
52
53         loadGameItem.setAccelerator(KeyStroke.getKeyStroke('L',
54             InputEvent.CTRL_DOWN_MASK));
55
56         exitGameItem.setAccelerator(KeyStroke.getKeyStroke('Q',
57             InputEvent.CTRL_DOWN_MASK));
58
59         menuBar.add(fileMenu);
60         setJMenuBar(menuBar);
61         createDefaultJMenuActionListeners();
62     }
63
64     public void setNewGameItemAction(ActionListener a) {
65         newGameItem.addActionListener(a);
66     }
67
68     public void setRestartGameItemAction(ActionListener a) {
69         restartGameItem.addActionListener(a);
70     }
71
72     public void setSaveGameItemAction(ActionListener a) {
73         saveGameItem.addActionListener(a);
74     }
75
76     public void setSaveGameAsItemAction(ActionListener a) {
77         saveGameAsItem.addActionListener(a);
78     }
79
80     public void setLoadGameItemAction(ActionListener a) {
81         loadGameItem.addActionListener(a);
82     }
83
84     public void setExitGameItemAction(ActionListener a) {
85         exitGameItem.addActionListener(a);
86     }
87
88     public abstract void addKeyListener();
89
90     public abstract void createDefaultJMenuActionListeners();
91
92 }

```

1.2.9. LevelSelector.java

```

1 package front;
2
3 import java.io.File;
4
5 /**
6  * @author tomas
7  * Interface to select level.
8  */
9 public interface LevelSelector {
10

```

```
11     public File getLevelSelected();
12
13 }
```

1.2.10. LevelSelectorImp.java

```
1 package front;
2
3 import java.awt.Frame;
4 import java.io.File;
5
6 import javax.swing.JFrame;
7 import javax.swing.JOptionPane;
8
9 /**
10  * @author tomas Class for show the player a list of levels that are ↵
11  * saved on
12  * the directory boards. It use a list of directorys and some ↵
13  * class of
14  * java swing.
15  */
16 public class LevelSelectorImp extends JFrame implements LevelSelector ↵
17 {
18     private static final long serialVersionUID = 1L;
19
20     private File levelSelected;
21
22     public LevelSelectorImp(Frame frameToShowOn) {
23         String[] listBoards;
24         File directory = new File(".") + File.separator + "boards";
25         listBoards = directory.list();
26         Object levelSelected = JOptionPane.showInputDialog(↵
27             frameToShowOn,
28             "Select level", "Levels selector",
29             JOptionPane.QUESTION_MESSAGE, null, listBoards, ↵
30             listBoards[0]);
31         if (levelSelected != null) {
32             this.levelSelected = new File(".") + File.separator + "↵
33             boards"
34             + File.separator + levelSelected);
35         }
36     }
37
38     public File getLevelSelected() {
39         return levelSelected;
40     }
41 }
```

1.3. parser

1.3.1. BoardDimensionLine.java

```
1 package parser;
2
3 import back.Point;
4
5 public class BoardDimensionLine extends Lines {
6
7     private static final int elemsCuantity = 2;
```



```

8     private Point boardDimension;
9
10    public BoardDimensionLine(String line) {
11        super(elemsCantidad, line);
12        lineProcess();
13        boardDimension = new Point(getData(0), getData(1));
14    }
15
16    public Point getBoardDimension() {
17        return boardDimension;
18    }
19
20 }

```

1.3.2. BoardLine.java

```

1  package parser;
2
3  import back.Point;
4
5  public class BoardLine extends Lines {
6
7      private static final int elemsCantidad = 6;
8      private Point boardDimension;
9
10     public BoardLine(String line, Point boardDimension) {
11         super(elemsCantidad, line);
12         this.boardDimension = boardDimension;
13         lineProcess();
14         lineCheck();
15     }
16
17     /**
18      * This methods Checks which type of cell the parsed line is, and ↵
19      * sets the
20      * cell into the board.
21      */
22     @Override
23     protected void lineCheck() {
24         switch (data[0]) {
25
26             case 1:
27                 // Player
28                 if (data[1] < 0 || data[1] >= boardDimension.x - 2 || data↵
29                     [2] < 0
30                     || data[2] >= boardDimension.y - 2 || data[3] != 0
31                     || data[4] != 0 || data[5] != 0) {
32                     throw new CorruptedFileException();
33                 }
34                 break;
35
36             case 2:
37                 // Wall
38                 if (data[1] < 0 || data[1] >= boardDimension.x - 2 || data↵
39                     [2] < 0
40                     || data[2] >= boardDimension.y - 2 || data[4] != ↵
41                     0) {
42                     throw new CorruptedFileException();
43                 }
44                 break;
45
46             case 3:
47                 // Monster
48                 if (data[1] < 0 || data[1] >= boardDimension.x - 2 || data↵
49                     [2] < 0
50                     || data[2] >= boardDimension.y - 2 || data[3] <= 0
51                     || data[3] > 3 || data[4] <= 0 || data[4] > 3) {
52                     throw new CorruptedFileException();
53                 }
54                 break;
55         }
56     }
57 }

```

```

49         }
50         break;
51
52     case 4:
53         // Life Bonus
54         if (data[1] < 0 || data[1] >= boardDimension.x - 2 || data[2] < 0
55             || data[2] >= boardDimension.y - 2 || data[3] != 0
56             || data[5] == 0) {
57             throw new CorruptedFileException();
58         }
59         break;
60
61     case 5:
62         // Strength Bonus
63         if (data[1] < 0 || data[1] >= boardDimension.x - 2 || data[2] < 0
64             || data[2] >= boardDimension.y - 2 || data[3] != 0
65             || data[5] == 0) {
66             throw new CorruptedFileException();
67         }
68         break;
69
70     default:
71         throw new CorruptedFileException();
72     }
73 }
74
75 public boolean isPlayerLine() {
76     return data[0] == 1;
77 }
78
79 public boolean isWallLine() {
80     return data[0] == 2;
81 }
82
83 public boolean isMonsterLine() {
84     return data[0] == 3;
85 }
86
87 public boolean isBonusLine() {
88     return data[0] >= 4;
89 }
90 }

```

1.3.3. BoardNameLine.java

```

1 package parser;
2
3 public class BoardNameLine extends Lines {
4
5     private static final int elemsQuantity = 1;
6     private String name;
7
8     public BoardNameLine(String line) {
9         super(elemsQuantity, line);
10        this.name = getLine();
11    }
12
13    @Override
14    protected void lineProcess() {}
15
16    public String getName() {
17        return name;
18    }
19
20 }

```

1.3.4. BoardParserFromFile.java

```
1 package parser;
2
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileReader;
6 import java.io.IOException;
7
8 import back.BoardObtainer;
9 import back.Bonus;
10 import back.Floor;
11 import back.Monster;
12 import back.Point;
13 import back.Putable;
14 import back.Wall;
15
16 /**
17  * @author tomas Class full dedicated to read a file and transform it ↵
18  * to a
19  * board.
20  */
21 public class BoardParserFromFile implements BoardObtainer {
22     private BufferedReader inputBoard;
23     private Point boardDimension;
24     private String boardName;
25     private Point playerPosition;
26     private Putable[][] board;
27     private File inputFile;
28
29     public BoardParserFromFile(File file) {
30         try {
31             inputFile = file;
32             inputBoard = new BufferedReader(new FileReader(file));
33             obtainBoard();
34         } catch (IOException e) {
35             throw new CorruptedFileException();
36         }
37     }
38
39     public void obtainBoard() throws IOException {
40
41         boolean dimensionFlag = false;
42         boolean nameFlag = false;
43         boolean playerFlag = false;
44         String line;
45
46         while ((line = inputBoard.readLine()) != null) {
47
48             line = line.replace(" ", "").replace("\t", "").replace("\n↵
49             ", "")
50                 .split("#")[0];
51
52             if (!line.isEmpty()) {
53                 if (!dimensionFlag) {
54                     parseDimension(line);
55                     dimensionFlag = true;
56                 } else if (!nameFlag) {
57                     parseBoardName(line);
58                     nameFlag = true;
59                 } else {
60                     if (line.startsWith("1")) {
61                         if (playerFlag == true) {
62                             throw new CorruptedFileException();
63                         }
64                         parsePlayer(line);
65                         playerFlag = true;
66                     } else {
```

```

67         BoardLine cell = new BoardLine(line, ←
68             boardDimension);
69         Point point = (new Point(cell.getData(1), cell
70             .getData(2))).add(new Point(1, 1));
71
72         if (cell.isWallLine()) {
73             parseWall(point, cell);
74         } else if (cell.isMonsterLine()) {
75             parseMonster(point, cell);
76         } else if (cell.isBonusLine()) {
77             parseBonus(point, cell);
78         }
79     }
80 }
81
82
83 if (!nameFlag || !playerFlag || !dimensionFlag) {
84     throw new CorruptedFileException();
85 }
86 validation();
87 }
88
89 public void validation() {
90     protectionWalls();
91     putFloor();
92     if (!(board[getPlayerPosition().x][getPlayerPosition().y] ←
93         instanceof Floor)) {
94         throw new CorruptedFileException();
95     }
96 }
97
98 public void parseBonus(Point point, BoardLine cell) {
99     putCell(point.x, point.y, new Bonus(point, cell.getData(0), ←
100         cell
101         .getData(5)));
102 }
103
104 public void parsePlayer(String line) {
105     BoardLine cell = new BoardLine(line, boardDimension);
106     Point point = (new Point(cell.getData(1), cell.getData(2)))
107         .add(new Point(1, 1));
108     playerPosition = point;
109 }
110
111 public void parseMonster(Point point, BoardLine cell) {
112     putCell(point.x, point.y, new Monster(point, cell.getData(3), ←
113         cell
114         .getData(4)));
115 }
116
117 public void parseWall(Point point, BoardLine cell) {
118     putCell(point.x, point.y, new Wall());
119 }
120
121 public void parseBoardName(String line) {
122     BoardNameLine boardNameLine = new BoardNameLine(line);
123     this.boardName = boardNameLine.getName();
124 }
125
126 public void parseDimension(String line) {
127     BoardDimensionLine boardDimensionLine = new BoardDimensionLine ←
128         (line);
129     boardDimension = boardDimensionLine.getBoardDimension().add(
130         new Point(2, 2));
131     board = new Putable[boardDimension.x][boardDimension.y];
132 }
133
134 public void putFloor() {
135     for (int i = 1; i < boardDimension.x - 1; i++) {
136         for (int j = 1; j < boardDimension.y - 1; j++) {
137             if (getBoardElem(i, j) == null) {
138                 putCell(i, j, new Floor());
139             }
140         }
141     }
142 }

```

```
136         }
137     }
138 }
139
140
141 public void protectionWalls() {
142     for (int i = 0; i < boardDimension.y; i++) {
143         Wall aux = new Wall();
144         aux.setVisible();
145         putCell(0, i, aux);
146         Wall aux1 = new Wall();
147         aux1.setVisible();
148         putCell(boardDimension.x - 1, i, aux1);
149     }
150     for (int i = 0; i < boardDimension.x; i++) {
151         Wall aux = new Wall();
152         aux.setVisible();
153         putCell(i, 0, aux);
154         Wall aux1 = new Wall();
155         aux1.setVisible();
156         putCell(i, boardDimension.y - 1, aux1);
157     }
158 }
159
160
161 public Point getBoardDimension() {
162     return boardDimension;
163 }
164
165 public String getBoardName() {
166     return boardName;
167 }
168
169 public Point getPlayerPosition() {
170     return playerPosition;
171 }
172
173 public Putable[][] getBoard() {
174     return board;
175 }
176
177 public int getBoardRows() {
178     return boardDimension.x;
179 }
180
181 public int getBoardColumns() {
182     return boardDimension.y;
183 }
184
185 public Putable getBoardElem(Point position) {
186     return board[position.x][position.y];
187 }
188
189 public Putable getBoardElem(int x, int y) {
190     return board[x][y];
191 }
192
193 public void putCell(int i, int j, Putable cell) {
194     putCell(new Point(i, j), cell);
195 }
196
197 public void putCell(Point p, Putable cell) {
198     board[p.x][p.y] = cell;
199 }
200
201 @Override
202 public File getFile() {
203     return inputFile;
204 }
205
206 @Override
207 public int getPlayerSteps() {
208     return 0;
209 }
```

```
210  
211 }
```

1.3.5. CorruptedFileException.java

```
1 package parser;  
2  
3 public class CorruptedFileException extends RuntimeException {  
4     private static final long serialVersionUID = 1L;  
5  
6  
7 }
```

1.3.6. Lines.java

```
1 package parser;  
2  
3 public abstract class Lines {  
4  
5     protected int[] data;  
6     private final int elemsCantidad;  
7     private String line;  
8  
9     public Lines(int elemsCantidad, String line) {  
10         this.elemsCantidad = elemsCantidad;  
11         this.line = line;  
12     }  
13  
14     /**  
15      * Process the line parsed by separating it by "," and removing ↵  
16      * the spaces,  
17      * enters and tabs in between.  
18      */  
19     protected void lineProcess() {  
20         data = new int[elemsCantidad];  
21         int k = 0;  
22         String[] arrayString;  
23  
24         arrayString = line.split(",");  
25  
26         if (arrayString.length == elemsCantidad) {  
27             for (k = 0; k < elemsCantidad; k++) {  
28                 try {  
29                     data[k] = Integer.valueOf(arrayString[k]);  
30                 } catch (NumberFormatException e) {  
31                     throw new CorruptedFileException();  
32                 }  
33             }  
34         } else {  
35             System.out.println(line);  
36             throw new CorruptedFileException();  
37         }  
38     }  
39  
40     public int getData(int i) {  
41         return data[i];  
42     }  
43  
44     public String getLine() {  
45         return line;  
46     }  
47 }
```

```

48     protected void lineCheck() {}
49 }

```

1.3.7. SavedBoardPlayerLine.java

```

1  package parser;
2
3  import back.Point;
4
5  public class SavedBoardPlayerLine extends Lines {
6
7      private static int elemsCantidad = 10;
8      private Point boardDimension;
9      private String playerName;
10
11     public SavedBoardPlayerLine(String line, Point boardDimension) {
12         super(elemsCantidad, line);
13         this.boardDimension = boardDimension;
14         lineProcess();
15         lineCheck();
16     }
17
18     @Override
19     protected void lineProcess() {
20         data = new int[elemsCantidad];
21         int k = 0;
22         String[] arrayString;
23
24         arrayString = getLine().split(",");
25
26         if (arrayString.length == elemsCantidad) {
27             for (k = 0; k < elemsCantidad - 1; k++) {
28                 try {
29                     data[k] = Integer.valueOf(arrayString[k]);
30                 } catch (NumberFormatException e) {
31                     throw new CorruptedFileException();
32                 }
33             }
34             playerName = arrayString[elemsCantidad - 1];
35         } else {
36             throw new CorruptedFileException();
37         }
38     }
39
40     @Override
41     protected void lineCheck() {
42
43         if (data[1] < 0 || data[1] >= boardDimension.x - 2 || data[2] <
44             0 || data[2] >= boardDimension.y || data[3] < 0
45             || data[3] > data[4] || data[5] < 0) {
46             throw new CorruptedFileException();
47         }
48     }
49
50     public String getPlayerName() {
51         return playerName;
52     }
53
54 }

```

1.4. professorShipSrc

1.4.1. GamePanel.java

```

1 package professorShipSrc;
2
3 import java.awt.Color;
4 import java.awt.Graphics;
5 import java.awt.Image;
6 import java.awt.event.MouseEvent;
7 import java.awt.event.MouseMotionAdapter;
8
9 import javax.swing.JPanel;
10
11 /**
12  * Panel que representa una grilla de imágenes, siendo posible ↵
13  * agregarle y quitarle imágenes. Asimismo, cuenta con una
14  * interfaz que permite a quien la utilice ser notificada cuando el ↵
15  * usuario posiciona el mouse sobre una celda de la grilla.
16  */
17 public class GamePanel extends JPanel {
18
19     private int rows, columns;
20     private int cellSize;
21     private Color color;
22     private Image [][] images;
23
24     /**
25      * Crea un nuevo panel con las dimensiones indicadas.
26      *
27      * @param rows Cantidad de filas.
28      * @param columns Cantidad de columnas.
29      * @param cellSize Ancho y alto de cada imagen en pÃxeles.
30      * @param listener Listener que serÃá notificado cuando el usuario ↵
31      * se posicione sobre una celda de la grilla.
32      * @param color Color de fondo del panel.
33      */
34     public GamePanel(final int rows, final int columns, final int ↵
35         cellSize, final GamePanelListener listener, Color color) {
36         setSize(columns * cellSize, rows * cellSize);
37         images = new Image[rows][columns];
38         this.rows = rows;
39         this.columns = columns;
40         this.cellSize = cellSize;
41         this.color = color;
42
43         addMouseMotionListener(new MouseMotionAdapter() {
44
45             private Integer currentRow;
46             private Integer currentColumn;
47
48             @Override
49             public void mouseMoved(MouseEvent e) {
50                 int row = e.getY() / cellSize;
51                 int column = e.getX() / cellSize;
52                 if (row >= rows || column >= columns || row < 0 || ↵
53                     column < 0) {
54                     return;
55                 }
56
57                 if (!nullSafeEquals(currentRow, row) || !↵
58                     nullSafeEquals(currentColumn, column)) {
59                     currentRow = row;
60                     currentColumn = column;
61                     listener.onMouseMoved(row, column);
62                 }
63             }
64
65             private boolean nullSafeEquals(Object o1, Object o2) {
66                 return o1 == null ? o2 == null : o1.equals(o2);
67             }
68         });
69     }
70
71     /**
72      * Ubica una imagen en la fila y columna indicadas.
73      */

```



```

68     public void put(Image image, int row, int column) {
69         images[row][column] = image;
70     }
71
72     /**
73      * Elimina la imagen ubicada en la fila y columna indicadas.
74      */
75     public void clear(int row, int column) {
76         images[row][column] = null;
77     }
78
79     @Override
80     public void paint(Graphics g) {
81         super.paint(g);
82         g.setColor(color);
83         g.fillRect(0, 0, columns * cellSize, rows * cellSize);
84
85         for (int i = 0; i < rows; i++) {
86             for (int j = 0; j < columns; j++) {
87                 if (images[i][j] != null) {
88                     g.drawImage(images[i][j], j * cellSize, i * cellSize, null);
89                 }
90             }
91         }
92     }
93 }

```

1.4.2. GamePanelListener.java

```

1 package professorShipSrc;
2
3 /**
4  * Listener para eventos ocurridos en el GamePanel.
5  */
6 public interface GamePanelListener {
7
8     /**
9      * Notifica cuando el usuario ubica el mouse sobre una celda de la grilla.
10     */
11     public void onMouseMoved(int row, int column);
12 }

```

1.4.3. ImageUtils.java

```

1 package professorShipSrc;
2
3 import java.awt.Color;
4 import java.awt.Font;
5 import java.awt.Graphics2D;
6 import java.awt.Image;
7 import java.awt.geom.Rectangle2D;
8 import java.awt.image.BufferedImage;
9 import java.io.File;
10 import java.io.IOException;
11 import java.io.InputStream;
12
13 import javax.imageio.ImageIO;
14
15 /**
16  * Clase con métodos útiles para el manejo de imágenes.
17  */

```

```

18 public class ImageUtils {
19
20     /**
21      * Carga una imagen y retorna una instancia de la misma. Si hay ↵
22      * algun problema al leer el archivo lanza una ↵
23      * excepcion. ↵
24      */
25     public static Image loadImage(String fileName) throws IOException ↵
26     {
27         InputStream stream = ClassLoader.getResourceAsStream(↵
28             fileName);
29         if (stream == null) {
30             return ImageIO.read(new File(fileName));
31         } else {
32             return ImageIO.read(stream);
33         }
34     }
35
36     /**
37      * Dibuja un texto en el vÃ©rtice inferior derecho de la imagen, ↵
38      * con el color indicado. Retorna una imagen nueva con ↵
39      * los cambios, la imagen original no se modifica. ↵
40      */
41     public static Image drawString(Image img, String text, Color color↵
42     ) {
43         BufferedImage result = new BufferedImage(img.getWidth(null), ↵
44             img.getHeight(null), BufferedImage.TYPE_INT_ARGB);
45         Graphics2D g = (Graphics2D) result.getGraphics();
46         g.drawImage(img, 0, 0, null);
47
48         Font font = new Font(Font.SANS_SERIF, Font.BOLD, 12);
49         g.setFont(font);
50         g.setColor(color);
51         Rectangle2D r = font.getStringBounds(text, g.↵
52             getFontRenderContext());
53         g.drawString(text, img.getWidth(null) - (int) r.getWidth() - ↵
54             2, img.getHeight(null) - 2);
55         return result;
56     }
57
58     /**
59      * Superpone dos imÃ¡genes. Retorna una nueva imagen con las 2 ↵
60      * imÃ¡genes recibidas superpuestas. Las ↵
61      * originales no se modifican. ↵
62      */
63     public static Image overlap(Image image1, Image image2) {
64         BufferedImage result = new BufferedImage(image1.getWidth(null)↵
65             , image1.getHeight(null),
66             BufferedImage.TYPE_INT_ARGB);
67         Graphics2D g = (Graphics2D) result.getGraphics();
68         g.drawImage(image1, 0, 0, null);
69         g.drawImage(image2, 0, 0, null);
70         return result;
71     }
72 }

```

1.5. saveLoadImplementation

1.5.1. Criteria.java

```

1 package saveLoadImplementation;
2
3 public interface Criteria<T> {
4     boolean satisfies(T obj);
5 }

```

1.5.2. FilterArrayFileList.java

```
1 package saveLoadImplementation;
2
3 import java.io.File;
4 import java.util.ArrayList;
5
6 public class FilterArrayFileList extends ArrayList<File> implements
7     FilterFileList {
8
9     /**
10      *
11      */
12     private static final long serialVersionUID = 1L;
13
14     public FilterArrayFileList() {
15     }
16
17     public FilterArrayFileList(File file) {
18         if (file.isDirectory()) {
19             File[] files = file.listFiles();
20             for (File f : files) {
21                 this.add(f);
22             }
23         }
24     }
25
26     @Override
27     public FilterFileList filter(String string) {
28         FilterArrayFileList filterArrayFileList = new FilterArrayFileList();
29         for (File t : this) {
30             if (t.getName().startsWith(string)) {
31                 filterArrayFileList.add(t);
32             }
33         }
34         return filterArrayFileList;
35     }
36
37 }
```

1.5.3. FilterFileList.java

```
1 package saveLoadImplementation;
2
3 import java.io.File;
4 import java.util.List;
5
6 public interface FilterFileList extends List<File>{
7
8     public FilterFileList filter(String string);
9
10 }
```

1.5.4. LoadGameFromFile.java

```
1 package saveLoadImplementation;
2
3 import java.io.File;
4
```

```

5 import parser.BoardLine;
6 import parser.BoardParserFromFile;
7 import parser.CorrruptedFileException;
8 import parser.SavedBoardPlayerLine;
9 import back.BloodyFloor;
10 import back.BoardObtainer;
11 import back.Floor;
12 import back.Game;
13 import back.GameListener;
14 import back.LoadGame;
15 import back.Monster;
16 import back.Point;
17
18 public class LoadGameFromFile<T extends Game> extends ←
    BoardParserFromFile
19     implements LoadGame<T> {
20
21     private Point playerLoadedPosition;
22     private Integer loadedLevel;
23     private Integer playerLoadedExperience;
24     private Integer playerLoadedHealth;
25     private Integer playerLoadedMaxHealth;
26     private Integer playerLoadedStrength;
27     private Integer playerLoadedSteps;
28     private String playerName;
29
30     public LoadGameFromFile(File placeToLoad) {
31         super(placeToLoad);
32     }
33
34     @Override
35     public void parsePlayer(String line) {
36         SavedBoardPlayerLine playerData = new SavedBoardPlayerLine(←
            line,
37             getBoardDimension());
38         Point point = (new Point(playerData.getData(1), playerData.←
            getData(2)))
39             .add(new Point(1, 1));
40         playerLoadedPosition = point;
41         playerLoadedExperience = playerData.getData(3);
42         playerLoadedHealth = playerData.getData(4);
43         playerLoadedMaxHealth = playerData.getData(5);
44         playerLoadedStrength = playerData.getData(6);
45         playerLoadedSteps = playerData.getData(7);
46         loadedLevel = playerData.getData(8);
47         playerName = playerData.getPlayerName();
48     }
49
50
51     private void setBoardCellVisivility(Point point, int num) {
52         if (num == 0) {
53             getBoardElem(point).setVisible();
54         } else {
55             getBoardElem(point).setNotVisible();
56         }
57     }
58
59     @Override
60     public void parseWall(Point point, BoardLine cell) {
61         if (cell.getData(3) == 2) {
62             putCell(point, new BloodyFloor());
63         } else if (cell.getData(3) == 1) {
64             putCell(point, new Floor());
65         } else {
66             super.parseWall(point, cell);
67         }
68         setBoardCellVisivility(point, cell.getData(5));
69     };
70
71     @Override
72     public void parseBonus(Point point, BoardLine cell) {
73         super.parseBonus(point, cell);
74         setBoardCellVisivility(point, cell.getData(4));
75     }

```

```

76
77
78     @Override
79     public void parseMonster(Point point, BoardLine cell) {
80         putCell(point.x,
81                 point.y,
82                 new Monster(point, cell.getData(3), cell.getData(4), ←
83                             Math
84                             .abs(cell.getData(5))));
85         if (cell.getData(5) < 0) {
86             setBoardCellVisivility(point, 0);
87         } else if (cell.getData(5) > 0) {
88             setBoardCellVisivility(point, 1);
89         }
90     }
91
92     @Override
93     public Point getPlayerPosition() {
94         return playerLoadedPosition;
95     }
96
97     @Override
98     public Integer getPlayerLoadedHealth() {
99         return playerLoadedHealth;
100     }
101
102     @Override
103     public Integer getPlayerLoadedMaxHealth() {
104         return playerLoadedMaxHealth;
105     }
106
107     @Override
108     public Integer getPlayerLoadedExperience() {
109         return playerLoadedExperience;
110     }
111
112     @Override
113     public Integer getPlayerLoadedStrength() {
114         return playerLoadedStrength;
115     }
116
117     @Override
118     public Integer getPlayerLoadedSteps() {
119         return playerLoadedSteps;
120     }
121
122     public T getGame(Class<T> gameImpClass, GameListener listener) {
123         T game;
124         try {
125             game = gameImpClass.getConstructor(BoardObtainer.class,
126                                                 GameListener.class).newInstance(this, listener);
127         } catch (Exception e) {
128             e.printStackTrace();
129             throw new CorruptedFileException();
130         }
131         return game;
132     }
133
134     @Override
135     public int getPlayerLoadedLevel() {
136         return loadedLevel;
137     }
138
139     @Override
140     public String getPlayerName() {
141         return playerName;
142     }
143 }

```

1.5.5. SaveGameOnFile.java

```

1 package saveLoadImplementation;
2
3 import java.io.BufferedWriter;
4 import java.io.File;
5 import java.io.FileWriter;
6 import java.io.IOException;
7
8 import back.BloodyFloor;
9 import back.Bonus;
10 import back.Floor;
11 import back.Game;
12 import back.Monster;
13 import back.SaveGame;
14 import back.Wall;
15
16 /**
17  * @author tomas SaveGame implementation that save on a file.
18  */
19 public class SaveGameOnFile implements SaveGame {
20
21     private Game gameToSave;
22     private File placeToSave;
23
24     public SaveGameOnFile(Game gameToSave) {
25         this.gameToSave = gameToSave;
26         File file = new File("./savedGames");
27         FilterFileList filterFileList = new FilterArrayFileList(file);
28         filterFileList = filterFileList.filter("savedGame");
29         int number = filterFileList.size();
30         if (number > 0) {
31             placeToSave = new File("./savedGames/savedGame" + "(" + number
32                 + ")");
33         } else {
34             placeToSave = new File("./savedGames/savedGame");
35         }
36         try {
37             save();
38         } catch (IOException e) {
39             throw new SavingCorruptedException();
40         }
41     }
42
43     public SaveGameOnFile(Game gameToSave, File placeToSave) {
44         this.gameToSave = gameToSave;
45         this.placeToSave = placeToSave;
46         FilterFileList filterFileList = new FilterArrayFileList(
47             placeToSave.getParentFile());
48         filterFileList = filterFileList.filter(placeToSave.getName());
49         int number = filterFileList.size();
50         if (number > 0) {
51             this.placeToSave = new File(placeToSave.getPath() + "(" + number
52                 + ")");
53         } else {
54             this.placeToSave = new File(placeToSave.getPath());
55         }
56         try {
57             save();
58         } catch (IOException e) {
59             throw new SavingCorruptedException();
60         }
61     }
62
63     /**
64     * The format of the file saved is: board dimension (10,11) board
65     * name
66     * ("Board name") player (1,row pos, col pos,exp,health,max health
67     * strength, steps, level, name) walls (2,row pos, col pos, 0 ,0,
68     * [0 is
69     * visible 1 not visible]) bloodyFloor(2,row pos, col pos, 2 ,0,
70     * [0 is

```

```

68     * visible 1 not visible]) floor(2,row pos, col pos, 1 ,0,[0 is ←
        visible 1
69     * not visible]) monsters (3,row pos, col pos, monster type, level←
        , [0 is
70     * visible 1 not visible]) bonus (4 or 5, row pos, col pos, 0,[0 ←
        is visible
71     * 1 not visible],amount of bonus)
72     */
73     public void save() throws IOException {
74         placeToSave.createNewFile();
75         BufferedWriter out = new BufferedWriter(new FileWriter(←
            placeToSave));
76         out.write("#Board dimensions");
77         out.newLine();
78         out.write((gameToSave.getBoardDimension().x - 2) + ", "
79             + (gameToSave.getBoardDimension().y - 2));
80         out.newLine();
81         out.write("#Board name");
82         out.newLine();
83         out.write(gameToSave.getBoardName());
84         out.newLine();
85         out.write("#Player current position, "
86             + "current exp, current health, maxHealth, current ←
            strength, steps, name");
87         out.newLine();
88         out.write(1 + ", " + (gameToSave.getPlayer().getPosition().x - ←
            1) + ", "
89             + (gameToSave.getPlayer().getPosition().y - 1) + ", "
90             + gameToSave.getPlayer().getExperience() + ", "
91             + gameToSave.getPlayer().getHealth() + ", "
92             + gameToSave.getPlayer().getMaxHealth() + ", "
93             + gameToSave.getPlayer().getStrength() + ", "
94             + gameToSave.getPlayer().getSteps() + ", "
95             + gameToSave.getPlayer().getLevel() + ", "
96             + gameToSave.getPlayer().getName());
97         out.newLine();
98         out.write("#Map");
99         out.newLine();
100        for (int i = 1; i < gameToSave.getBoardDimension().x - 1; i++)←
            {
101            for (int j = 1; j < gameToSave.getBoardDimension().y - 1; ←
                j++) {
102                if (Wall.class.equals((gameToSave.getBoard()[i][j]).←
                    getClass())) {
103                    out.write(2 + ", " + (i - 1) + ", " + (j - 1) + ", " ←
                        + 0 + ", "
104                        + 0 + ", ");
105                    if (gameToSave.getBoard()[i][j].isVisible()) {
106                        out.write("0");
107                    } else {
108                        out.write("1");
109                    }
110                    out.newLine();
111                } else if (Floor.class.equals((gameToSave.getBoard()[i]←
                    [j]).
112                    getClass())) {
113                    out.write(2 + ", " + (i - 1) + ", " + (j - 1) + ", " ←
                        + 1 + ", "
114                        + 0 + ", ");
115                    if (gameToSave.getBoard()[i][j].isVisible()) {
116                        out.write("0");
117                    } else {
118                        out.write("1");
119                    }
120                    out.newLine();
121                } else if (BloodyFloor.class
122                    .equals((gameToSave.getBoard()[i][j]).getClass()←
                    ())) {
123                    out.write(2 + ", " + (i - 1) + ", " + (j - 1) + ", " ←
                        + 2 + ", "
124                        + 0 + ", ");
125                    if (gameToSave.getBoard()[i][j].isVisible()) {
126                        out.write("0");
127                    } else {

```

```

128         out.write("1");
129     }
130     out.newLine();
131 } else if (Monster.class.equals((gameToSave.getBoard()[i][j])
132         .getClass())) {
133     out.write(3
134         + " "
135         + (i - 1)
136         + " "
137         + (j - 1)
138         + " "
139         + (((Monster) gameToSave.getBoard()[i][j])
140             .getMonsterType().ordinal() + 1)
141         + " "
142         + (((Monster) gameToSave.getBoard()[i][j])
143             .getLevel() + " ");
144     if (gameToSave.getBoard()[i][j].isVisible()) {
145         out.write((((Monster) gameToSave.getBoard()[i][j])
146             .getHealth() * -1) + " ");
147     } else {
148         out.write((((Monster) gameToSave.getBoard()[i][j])
149             .getHealth() + " ");
150     }
151     out.newLine();
152 } else if (Bonus.class.equals((gameToSave.getBoard()[i][j])
153     .getClass())) {
154     out.write((((Bonus) gameToSave.getBoard()[i][j])
155         .getBonusType().ordinal() + 4)
156         + " "
157         + (i - 1)
158         + " " + (j - 1) + " " + 0 + " ");
159     if (gameToSave.getBoard()[i][j].isVisible()) {
160         out.write("0");
161     } else {
162         out.write("1");
163     }
164     out.write(" "
165         + ((Bonus) gameToSave.getBoard()[i][j])
166             .getAmountBonus());
167     out.newLine();
168 }
169 }
170 }
171
172 out.flush();
173 out.close();
174
175 }
176 }

```

1.5.6. SavingCorruptedException.java

```

1 package saveLoadImplementation;
2
3 public class SavingCorruptedException extends RuntimeException {
4
5     /**
6      *
7      */
8     private static final long serialVersionUID = 1L;
9
10 }

```


1.6. tests

1.6.1. GameTests.java

```
1 package tests;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.assertTrue;
5
6 import java.io.File;
7
8 import javax.swing.JOptionPane;
9
10 import org.junit.Before;
11 import org.junit.Test;
12
13 import parser.BoardParserFromFile;
14 import saveLoadImplementation.FilterArrayFileList;
15 import saveLoadImplementation.FilterFileList;
16 import saveLoadImplementation.LoadGameFromFile;
17 import saveLoadImplementation.SaveGameOnFile;
18 import back.BloodyFloor;
19 import back.Bonus;
20 import back.DungeonGameImp;
21 import back.DungeonGameListener;
22 import back.LoadGame;
23 import back.Monster;
24 import back.MoveTypes;
25 import back.Point;
26
27 public class GameTests {
28
29     private DungeonGameImp game;
30
31     @Before
32     public void setup() {
33         game = new DungeonGameImp(new BoardParserFromFile(new File(
34             "./testBoard/boardForTest1"), new DungeonGameListener() {
35
36             @Override
37             public String playerNameRequest() {
38                 return "Tom";
39             }
40
41             @Override
42             public void executeWhenPlayerMoves(MoveTypes moveType) {
43             }
44
45             @Override
46             public void executeWhenGameWon() {
47             }
48
49             @Override
50             public void executeWhenGameLoosed() {
51             }
52
53             @Override
54             public void executeWhenCharacterDie(Point p) {
55             }
56
57             @Override
58             public void executeWhenBonusGrabed(Point p) {
59             }
60
61             @Override
62             public void executeWhenFight() {
63             }
64
65             @Override
66             public void executeWhenLevelUp() {
```

```

67     }
68     });
69 }
70
71 @Test
72 public void goodFunctionamientOfmovePlayerTest() {
73     game.receiveMoveStroke(MoveTypes.LEFT);
74     game.receiveMoveStroke(MoveTypes.LEFT);
75     assertEquals(new Integer(4), game.getPlayer().getHealth());
76     assertEquals(new Integer(1), game.getPlayer().getExperience());
77
78     game.receiveMoveStroke(MoveTypes.LEFT);
79     assertEquals(new Point(4, 3), game.getPlayer().getPosition());
80     game.receiveMoveStroke(MoveTypes.RIGHT);
81     assertEquals(new Point(4, 4), game.getPlayer().getPosition());
82     game.receiveMoveStroke(MoveTypes.DOWN);
83     assertEquals(new Point(5, 4), game.getPlayer().getPosition());
84     game.receiveMoveStroke(MoveTypes.UP);
85     assertEquals(new Point(4, 4), game.getPlayer().getPosition());
86 }
87
88 @Test
89 public void goodFunctionamientOfWiningWhenKillMonsterLevel3Test() {
90     game.getPlayer().winLife(40);
91     Bonus bonus = new Bonus(new Point(7,7),4,50);
92     Bonus bonus2 = new Bonus(new Point(7,7),5,50);
93     bonus.giveBonus(game.getPlayer());
94     bonus2.giveBonus(game.getPlayer());
95     game.getPlayer().setPosition(new Point(8, 2));
96     game.receiveMoveStroke(MoveTypes.LEFT);
97 }
98
99 @Test
100 public void goodFunctionamientOfResetGameTest() {
101     game.getPlayer().winLife(40);
102     Bonus bonus = new Bonus(new Point(7,7),4,50);
103     Bonus bonus2 = new Bonus(new Point(7,7),5,50);
104     bonus.giveBonus(game.getPlayer());
105     bonus2.giveBonus(game.getPlayer());
106     game.getPlayer().setPosition(new Point(4, 6));
107     game.receiveMoveStroke(MoveTypes.UP);
108     assertEquals(BloodyFloor.class, ((game.getBoard()[3][6])).getClass());
109     game.restart();
110     assertEquals(Monster.class, ((game.getBoard()[3][6])).getClass());
111     assertEquals(new Point(4, 4), game.getPlayer().getPosition());
112 }
113
114 @Test
115 public void forWatchTheGameSavedTest() {
116     File directory = new File("./savedGames");
117     if (!directory.exists()) {
118         directory.mkdir();
119     }
120     new SaveGameOnFile(game);
121     File file = new File("./savedGames");
122     FilterFileList filterFileList = new FilterArrayFileList(file);
123     filterFileList = filterFileList.filter("savedGame");
124     int number = filterFileList.size();
125     if (number > 1) {
126         File f = new File("./savedGames/savedGame" + "(" + (number - 1) + ")");
127         assertTrue(f.exists());
128         f.delete();
129     } else {
130         File f = new File("./savedGames/savedGame");
131         assertTrue(f.exists());
132         f.delete();
133     }
134 }
135

```

```

136  @Test
137  public void loadGameTest() {
138      File file = new File("./savedGames/testWithPath");
139      new SaveGameOnFile(game, file);
140      LoadGame<DungeonGameImp> loadGame = new LoadGameFromFile(<←
          DungeonGameImp>(file);
141      DungeonGameImp game = loadGame.getGame(DungeonGameImp.class, <←
          new DungeonGameListener() {
142
143          @Override
144          public String playerNameRequest() {
145              String name = null;
146              while (name == null || name.isEmpty()) {
147                  name = JOptionPane.showInputDialog("Player name");
148              }
149              return name;
150          }
151
152          @Override
153          public void executeWhenPlayerMoves(MoveTypes moveType) {
154          }
155
156          @Override
157          public void executeWhenGameWonned() {
158          }
159
160          @Override
161          public void executeWhenGameLoosed() {
162          }
163
164          @Override
165          public void executeWhenCharacterDie(Point p) {
166          }
167
168          @Override
169          public void executeWhenBonusGrabed(Point p) {
170          }
171
172          @Override
173          public void executeWhenFight() {
174          }
175
176          @Override
177          public void executeWhenLevelUp() {
178          }
179      });
180      assertEquals(new Integer(0), game.getPlayer().getExperience());<←
          ;
181      assertEquals(new Point(4, 4), game.getPlayer().getPosition());
182      file.delete();
183  }
184
185  @Test
186  public void forWatchTheGameSavedWithPathTest() {
187      File directory = new File("./savedGames");
188      if (!directory.exists()) {
189          directory.mkdir();
190      }
191      File file = new File("./savedGames/testWithPath");
192      new SaveGameOnFile(game, file);
193      FilterFileList filterFileList = new FilterArrayFileList(
194          file.getParentFile());
195      filterFileList = filterFileList.filter(file.getName());
196      int number = filterFileList.size();
197      if (number > 1) {
198          File f = new File(file.getPath() + "(" + (number - 1) + ")<←
              ";
199          assertTrue(f.exists());
200          f.delete();
201      } else {
202          File f = new File(file.getPath());
203          assertTrue(f.exists());
204          f.delete();
205      }

```

```
206     }
207
208 }
```

1.6.2. PlayerTests.java

```
1  package tests;
2
3  import static org.junit.Assert.assertEquals;
4
5  import java.io.File;
6
7  import org.junit.Before;
8  import org.junit.Test;
9
10 import parser.BoardParserFromFile;
11 import back.BoardObtainer;
12 import back.Bonus;
13 import back.Monster;
14 import back.MoveTypes;
15 import back.Player;
16 import back.PlayerData;
17 import back.Point;
18
19 public class PlayerTests {
20     BoardObtainer boardParser;
21     Player player;
22
23     @Before
24     public void setup() {
25         boardParser = new BoardParserFromFile(new File(
26             "./testBoard/boardForTest1"));
27         player = new Player(new PlayerData("Tomas", 0, 0, 10, 10, 5,
28             boardParser.getPlayerPosition(), 0));
29     }
30
31     @Test
32     public void goodFunctionamientPlayerMovementTest() {
33         assertEquals(new Point(4, 4), player.getPosition());
34         player.move(MoveTypes.UP);
35         assertEquals(new Point(3, 4), player.getPosition());
36         player.move(MoveTypes.LEFT);
37         assertEquals(new Point(3, 3), player.getPosition());
38         player.move(MoveTypes.DOWN);
39         assertEquals(new Point(4, 3), player.getPosition());
40         player.move(MoveTypes.RIGHT);
41         assertEquals(new Point(4, 4), player.getPosition());
42     }
43
44     @Test
45     public void goodFunctionamientPlayerVsMonsterFightTest() {
46         Monster monster = ((Monster) boardParser.getBoard()[5][7]);
47         player.fightAnotherCharacter(monster);
48         assertEquals(
49             new Integer(player.getMaxHealth() - monster.↵
50                 getStrength()),
51             player.getHealth());
52         assertEquals(
53             new Integer(monster.getMaxHealth() - player.↵
54                 getStrength()),
55             monster.getHealth());
56     }
57
58     @Test
59     public void goodFunctionamientPlayerEarningBonusTest() {
60         player.hited(9);
61         ((Bonus) boardParser.getBoard()[8][2]).giveBonus(player);
62         ((Bonus) boardParser.getBoard()[2][8]).giveBonus(player);
63         assertEquals(new Integer(6), player.getHealth());
64     }
65 }
```

```
62         assertEquals(new Integer(8), player.getStrength());
63     }
64 }
65 }
66 }
```

1.6.3. ParserTests.java

```
1 package tests;
2
3 import static org.junit.Assert.assertEquals;
4
5 import java.io.File;
6
7 import org.junit.Before;
8 import org.junit.Test;
9
10 import parser.BoardParserFromFile;
11 import parser.CorruptedFileException;
12 import back.BoardObtainer;
13 import back.Bonus;
14 import back.Monster;
15 import back.MonsterTypes;
16 import back.Point;
17 import back.Wall;
18
19 public class ParserTests {
20
21     BoardObtainer boardParser;
22
23     @Before
24     public void setup() {
25         boardParser = new BoardParserFromFile(new File(
26             "./testBoard/boardForTest1"));
27     }
28
29     @Test(expected = CorruptedFileException.class)
30     public void startPlayerPositionOverAMonsterTest() {
31         new BoardParserFromFile(new File("./testBoard/boardForTest2"))↵
32     }
33
34     @Test(expected = CorruptedFileException.class)
35     public void startPlayerPositionOverAWallTest() {
36         new BoardParserFromFile(new File("./testBoard/boardForTest3"))↵
37     }
38
39     @Test
40     public void mapWithoutSurroundingWalls() {
41         BoardObtainer boardParser = new BoardParserFromFile(new File(
42             "./testBoard/boardForTest4"));
43         assertEquals(Wall.class, boardParser.getBoardElem(new Point(0,↵
44             0))
45             .getClass());
46         assertEquals(Wall.class, boardParser.getBoardElem(new Point(↵
47             11, 0))
48             .getClass());
49         assertEquals(Wall.class, boardParser.getBoardElem(new Point(0,↵
50             11))
51             .getClass());
52         assertEquals(Wall.class, boardParser.getBoardElem(new Point(↵
53             11, 11))
54             .getClass());
55     }
56
57     @Test(expected = CorruptedFileException.class)
58     public void positionOutOfBoardDimensionsTest() {
```

```

55         new BoardParserFromFile(new File("./testBoard/boardForTest5"))↵
56     };
57 }
58 @Test(expected = CorruptedFileException.class)
59 public void badPathPassedTest() {
60     new BoardParserFromFile(new File("./noExist"));
61 }
62
63 @Test
64 public void goodParseOfBoardDimensionTest() {
65     assertEquals(new Point(12, 12), boardParser.getBoardDimension↵
66         ());
67 }
68
69 @Test
70 public void goodParseOfBoardNameTest() {
71     assertEquals("ejemplotablero", boardParser.getBoardName());
72 }
73
74 @Test
75 public void goodParseOfPlayerPositionTest() {
76     assertEquals(new Point(4, 4), boardParser.getPlayerPosition())↵
77         ;
78 }
79
80 @Test
81 public void goodParseOfAnyCellPositionTest() {
82     assertEquals(Wall.class, boardParser.getBoard()[1][1].getClass()↵
83         ());
84     assertEquals(Wall.class, boardParser.getBoard()[10][1].↵
85         getClass());
86     assertEquals(Wall.class, boardParser.getBoard()[1][10].↵
87         getClass());
88     assertEquals(Wall.class, boardParser.getBoard()[10][10].↵
89         getClass());
90     assertEquals(Bonus.class,
91         boardParser.getBoard()[2][8].getClass());
92     assertEquals(Bonus.class, boardParser.getBoard()[8][2].↵
93         getClass());
94     assertEquals(Monster.class, boardParser.getBoard()[5][7].↵
95         getClass());
96     assertEquals(Monster.class, boardParser.getBoard()[3][6].↵
97         getClass());
98     assertEquals(Monster.class, boardParser.getBoard()[2][4].↵
99         getClass());
100 }
101
102 @Test
103 public void goodParseOfMonsterTest() {
104     assertEquals(MonsterTypes.DRAGON,
105         ((Monster) boardParser.getBoard()[9][2]).↵
106         getMonsterType());
107     assertEquals(new Integer(3),
108         ((Monster) boardParser.getBoard()[9][2]).getLevel());
109 }
110
111 @Test
112 public void goodParseOfBonusTest() {
113     assertEquals(5,
114         ((Bonus) boardParser.getBoard()[8][2]).getAmountBonus↵
115         ());
116     assertEquals(3,
117         ((Bonus) boardParser.getBoard()[2][8])
118         .getAmountBonus());
119 }
120
121 @Test
122 public void boardWatchTest() {
123     String resp = "";
124     for (int i = 0; i < boardParser.getBoardRows(); i++) {
125         for (int j = 0; j < boardParser.getBoardColumns(); j++) {
126             resp += boardParser.getBoard()[i][j] + " ";
127         }
128     }

```

```
116         resp += "\n";
117     }
118     System.out.println(resp);
119 }
120
121 }
```