Programación orientada a objetos
Códigos fuente TPE
Dungeon Game

7 de junio de 2011

Autores:

Tomás Mehdi	Legajo: *51014*
Alan Pomerantz	Legajo: *51233*

Índice

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 ↩
        and strength.
6   */
7  public interface Algoritms {
8      public Integer lifeAlgoritm(int level);
9
10     public Integer strengthAlgoritm(int level);
11
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 getBoardColums();
22
23     public File getFile();
24
25     public PlayerData getPlayerData();
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          bonusType = BonusTypes.getBonusType(numberBonusType);
9          bonusType.setBonusAmount(bonusAmount);
10     }
11
12     public void giveBonus(Character character) {
13         bonusType.giveBonus(character);
14     }
15
16     @Override
17     public boolean allowMovement(DungeonGameImp game) {
18         return true;
19     }
20
21     public void standOver(DungeonGameImp game) {
22         giveBonus(game.getPlayer());
23         Point point = new Point(game.getPlayer().getPosition().x, game
24                 .getPlayer().getPosition().y);
25
26         Floor f = new Floor();
27         f.setVisible();
28         game.getBoard()[point.x][point.y] = f;
29
30         game.getGameListener().executeWhenBonusGrabed(
31                 new Point(point.x, point.y));
32     }
33
34     public BonusTypes getBonusType() {
35         return bonusType;
36     }
37
38     public int getAmountBonus() {
39         return bonusType.getBonusAmount();
40     }
41
42     @Override
43     public String toString() {
44         return "Bonus";
45     }
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
16      }), STRENGTH("Strength", 0, new GrabBonus(){
17
18          @Override
19          public void grabBonus(Character character, Integer bonusAmount↩
                ) {
20              character.grabStrengthBonus(bonusAmount);
21          }
22
23      });
24
25      private String name;
26      private Integer bonusAmount;
27      private GrabBonus bonusGrabbed;
28
29      private BonusTypes(String name, Integer bonusAmount, GrabBonus ↩
            bonusGrabbed) {
30          this.name = name;
31          this.bonusAmount = bonusAmount;
32          this.bonusGrabbed = bonusGrabbed;
33      }
34
35      public Integer getBonusAmount(){
36          return bonusAmount;
37      }
38
39      public void setBonusAmount(Integer bonusAmount){
40          this.bonusAmount = bonusAmount;
41      }
42
43      public String getName() {
44          return name;
45      }
46
47      public static BonusTypes getBonusType(int data) {
48          switch (data) {
49          case (4):
50              return BonusTypes.LIFE;
51          case (5):
52              return BonusTypes.STRENGTH;
53          default:
54              return null;
55          }
56      }
57
58      public void giveBonus(Character character) {
59          bonusGrabbed.grabBonus(character,getBonusAmount());
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 ↩
         can be on the board
6    *  to be visible or invisible. Particular feature of this game.
7    */
8   public abstract class Cell {
9
```

```
10      private 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 ve ↩
         visible or
5    *         invisible in the board.
6    */
7   public abstract class Character extends Cell {
8
9       private String name;
10      private Integer level;
11      private Integer maxHealth;
12      private Integer health;
13      private Integer strength;
14      private Point position;
15
16      public Character(String name, Integer level, Point position) {
17          this.name = name;
18          this.level = level;
19          this.position = position;
20      }
21
22      public void winFight(Character character) {
23      }
24
25      public void fightAnotherCharacter(Character character) {
26          this.hited(character.getStrength());
27          if (!this.isDead()) {
28              character.hited(this.getStrength());
29              if (character.isDead()) {
30                  this.winFight(character);
31              }
32          } else {
33              character.winFight(this);
34          }
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     public final static Integer LEVEL = 3;
14     public final static Integer LIFE = 10;
15     public 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     public DungeonGameImp(BoardObtainer boardObtainer, GameListener ↩
           gameListener) {
25         this.boardObtainer = boardObtainer;
26         this.gameListener = gameListener;
27         boardName = boardObtainer.getBoardName();
28         boardDimension = boardObtainer.getBoardDimension();
29         board = boardObtainer.getBoard();
30         PlayerData playerData = boardObtainer.getPlayerData();
31         if (!(boardObtainer instanceof LoadGame)) {
32             playerData.setName(gameListener.playerNameRequest());
33         }
34         player = new Player(playerData);
35         firstDiscoveredCells();
36     }
37
38     private void firstDiscoveredCells() {
39         Point p = player.getPosition();
40
41         board[p.x][p.y].setVisible();
42
43         board[p.x + 1][p.y - 1].setVisible();
44         board[p.x + 1][p.y].setVisible();
45         board[p.x + 1][p.y + 1].setVisible();
46
47         board[p.x][p.y - 1].setVisible();
48         board[p.x][p.y].setVisible();
49         board[p.x][p.y + 1].setVisible();
50
51         board[p.x - 1][p.y - 1].setVisible();
52         board[p.x - 1][p.y].setVisible();
53         board[p.x - 1][p.y + 1].setVisible();
54     }
55
56     /**
57      * @see back.Game#receiveMoveStroke(back.MoveTypes) Is't allow the↩
            game to
58      *      receive a Stroke. In this case a MoveTypes stroke. Before ↩
           this the
59      *      player moves.
60      **/
61     @Override
62     public void receiveMoveStroke(MoveTypes moveType) {
63         Point nextPlayerPosition = player.getPosition().add(
64                 moveType.getDirection());
65         int playerLevelBeforeFight = player.getLevel();
66         if (board[nextPlayerPosition.x][nextPlayerPosition.y]
67                 .allowMovement(this)) {
68             MoveTypes moveMade = player.move(moveType);
69             dicoverBoard(nextPlayerPosition, moveType);
70             gameListener.executeWhenPlayerMoves(moveMade);
71             board[nextPlayerPosition.x][nextPlayerPosition.y].↩
                   standOver(this);
72         }
73         if (player.getLevel() != playerLevelBeforeFight) {
74             gameListener.executeWhenLevelUp();
75         }
```

```
76      }
77
78      /**
79       * When player moves exist the possibility of discover ↩
             undiscovered board
80       * parts. When this happen the game have to give life to ↩
             characters on the
81       * parts of the board already discovered. This amount is equals of↩
              the
82       * character level.
83       */
84      private void dicoverBoard(Point pos, MoveTypes dir) {
85          int countDiscover = 0;
86          List<Point> points = new ArrayList<Point>();
87          points.add(pos.add(dir.getDirection()));
88          if (dir == MoveTypes.LEFT || dir == MoveTypes.RIGHT) {
89              points.add(pos.add(1, 0).add(dir.getDirection()));
90              points.add(pos.sub(1, 0).add(dir.getDirection()));
91          } else {
92              points.add(pos.add(0, 1).add(dir.getDirection()));
93              points.add(pos.sub(0, 1).add(dir.getDirection()));
94          }
95
96          for (Point poo : points) {
97              if (!board[poo.x][poo.y].isVisible()) {
98                  countDiscover++;
99                  board[poo.x][poo.y].setVisible();
100             }
101         }
102
103         if (countDiscover > 0) {
104             player.winLife(countDiscover * player.getLevel());
105             for (int i = 1; i < boardDimension.x - 1; i++) {
106                 for (int j = 1; j < boardDimension.y - 1; j++) {
107                     if (board[i][j].isVisible()
108                             && board[i][j] instanceof Character) {
109                         ((Character) board[i][j]).winLife(↩
                                 countDiscover
110                                     * ((Character) board[i][j]).getLevel()↩
                                         );
111                     }
112                 }
113             }
114         }
115     }
116
117     @Override
118     public Player getPlayer() {
119         return player;
120     }
121
122     @Override
123     public void winned() {
124         gameListener.executeWhenGameWinned();
125     }
126
127     @Override
128     public void loosed() {
129         gameListener.executeWhenGameLoosed();
130     }
131
132     /**
133      * @param character
134      *            Method executed when a fight end. It's validate if a↩
             character
135      *            died. If any died execute a listener was provided by↩
             the
136      *            front.
137      */
138     public void fightEnd(Character character) {
139         if (character.isDead()) {
140             Point point = new Point(character.getPosition().x, ↩
                     character
141                         .getPosition().y);
```

```
142             BloodyFloor bf = new BloodyFloor();
143             bf.setVisible();
144             board[point.x][point.y] = bf;
145             gameListener.executeWhenCharacterDie(point);
146 
147         }
148         if (player.isDead()) {
149             Point point = new Point(player.getPosition().x, player
150                     .getPosition().y);
151             BloodyFloor bf = new BloodyFloor();
152             bf.setVisible();
153             board[point.x][point.y] = bf;
154             gameListener.executeWhenCharacterDie(point);
155             loosed();
156         }
157         gameListener.executeWhenFight();
158 
159     }
160 
161     @Override
162     public Putable[][][] getBoard() {
163         return board;
164     }
165 
166     @Override
167     public Point getBoardDimension() {
168         return boardDimension;
169     }
170 
171     @Override
172     public String getBoardName() {
173         return boardName;
174     }
175 
176     @Override
177     public GameListener getGameListener() {
178         return gameListener;
179     }
180 
181     @Override
182     public void addGameListener(GameListener d) {
183         gameListener = d;
184     }
185 
186     @Override
187     public BoardObtainer getBoardObtainer() {
188         return boardObtainer;
189     }
190 
191     /**
192      * @see back.Game#restart() The desition of making restart a ↩
           method of a
193      *      game and not a class like loadGame is that a restart game ↩
           need the
194      *      same boardObtainer that the instance of the game. Then is ↩
           have no
195      *      sense make a new instance.
196      **/
197     @Override
198     public void restart() {
199         File file = boardObtainer.getFile();
200         try {
201             board = boardObtainer.getClass().getConstructor(File.class↩
                  )
202                     .newInstance(file).getBoard();
203         } catch (Exception e) {
204         }
205         PlayerData playerData = new PlayerData(player.getName(), 0, 0,↩
               LIFE,
206                 LIFE, STRENGTH, boardObtainer.getPlayerPosition(), ↩
                    player
207                         .getSteps());
208         player = new Player(playerData);
209     }
```

```
210
211 }
```

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
5      @Override
6      public String toString() {
7          return "Floor";
8      }
9
10     @Override
11     public boolean allowMovement(DungeonGameImp game) {
12         return true;
13     }
14
15     @Override
16     public void standOver(DungeonGameImp game) {}
17
18 }
```

1.1.11. Game.java

```
1  package back;
2
3  public interface Game {
4
5      public void winned();
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 executeWhenGameWinned();
16
17     public String playerNameRequest();
18
19     public 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      public Integer getPlayerLoadedExperience();
10
11     public 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         return result;
12     }
13
14     @Override
15     public boolean equals(Object obj) {
16         if (this == obj)
17             return true;
18         if (!super.equals(obj))
19             return false;
20         if (getClass() != obj.getClass())
21             return false;
22         Monster other = (Monster) obj;
23         if (monsterType == null) {
24             if (other.monsterType != null)
25                 return false;
26         } else if (!monsterType.equals(other.monsterType))
27             return false;
28         return true;
29     }
30
31     private MonsterTypes monsterType;
32
33     public Monster(Point position, int numberMonsterType, int level) {
34         this(position, numberMonsterType, level, MonsterTypes.↩
               getMonsterType(
35                 numberMonsterType).getMaxLife(level));
36     }
37
38     public Monster(Point position, int numberMonsterType, int level, ↩
           int health) {
39         super(MonsterTypes.getMonsterType(numberMonsterType).getName()↩
               , level,
40                 position);
41         monsterType = MonsterTypes.getMonsterType(numberMonsterType);
42         setMaxHealth(monsterType.getMaxLife(level));
43         setStrength(monsterType.getStrength(level));
44         setHealth(health);
45     }
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 Algoritms() {
7
8           @Override
9           public Integer lifeAlgoritm(int level) {
10              return (int) Math.floor((((level + 3) * (level + 3)) - 10)↩
                    * GOLEMLIFE);
11          }
12
13          @Override
14          public Integer strengthAlgoritm(int level) {
15              return (int) Math.floor(((level * level) + 5 * level) * ↩
                    0.5 * GOLEMSTRENGTH);
16          }
17
18      }), DRAGON("Dragon", new Algoritms() {
19
20          @Override
21          public Integer lifeAlgoritm(int level) {
22              return (int) Math.floor((((level + 3) * (level + 3)) - 10)↩
                    * DRAGONLIFE);
23          }
24
25          @Override
26          public Integer strengthAlgoritm(int level) {
27              return (int) Math.floor(((level * level) + 5 * level) * ↩
                    0.5 * DRAGONSTRENGTH);
28          }
29      }), SNAKE("Snake", new Algoritms() {
30
31          @Override
32          public Integer lifeAlgoritm(int level) {
33              return (int) Math.floor((((level + 3) * (level + 3)) - 10)↩
                    * SNAKELIFE);
34          }
35
36          @Override
37          public Integer strengthAlgoritm(int level) {
38              return (int) Math.floor(((level * level) + 5 * level) * ↩
                    0.5 * SNAKESTRENGTH);
39          }
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 Algoritms lifeStrengthAlgoritms;
52
```

```
53      private MonsterTypes(String name, Algoritms lifeStrengthAlgoritms)↩
            {
54          this.name = name;
55          this.lifeStrengthAlgoritms = lifeStrengthAlgoritms;
56      }
57
58      public Integer getMaxLife(int level) {
59          return lifeStrengthAlgoritms.lifeAlgoritm(level);
60      }
61
62      public Integer getStrength(int level) {
63          return lifeStrengthAlgoritms.strengthAlgoritm(level);
64      }
65
66      public static MonsterTypes getMonsterType(int data) {
67          switch (data) {
68          case (1):
69              return MonsterTypes.GOLEM;
70          case (2):
71              return MonsterTypes.DRAGON;
72          default:
73              return MonsterTypes.SNAKE;
74          }
75      }
76
77      public String getName() {
78          return name;
79      }
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)↩
            ), RIGHT(
5               new Point(0, 1));
6
7       private Point direction;
8
9       private MoveTypes(Point p){
10          this.direction=p;
11      }
12
13      public Point getDirection(){
14          return direction;
15      }
16
17      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(), playerData.getLevel(), playerData.↩
               getPosition());
13         this.experienceToLevelUp = EXPERIENCECONSTANT * getLevel();
14         this.experience = playerData.getExperience();
15         setMaxHealth(playerData.getMaxHealth());
16         setHealth(playerData.getHealth());
17         setStrength(playerData.getStrength());
18
19     }
20
21     public MoveTypes move(MoveTypes moveType) {
22         setPosition(getPosition().add(moveType.getDirection()));
23         steps++;
24         return moveType;
25     }
26
27     public void winExperience(Integer experience) {
28         if ((this.experience + experience) >= experienceToLevelUp) {
29             levelUp();
30         } else {
31             this.experience += experience;
32         }
33     }
34
35     private void levelUp() {
36         increaseLevel();
37         this.experience = 0;
38         this.experienceToLevelUp = EXPERIENCECONSTANT * getLevel();
39         setMaxHealth(getLevel() * DungeonGameImp.LIFE);
40         setStrength(getStrength() + DungeonGameImp.STRENGTH);
41     }
42
43     public Integer getExperience() {
44         return experience;
45     }
46
47     public void winFight(Character character) {
48         winExperience(character.getLevel());
49     }
50
51     @Override
52     public String toString() {
53         String resp;
54         resp = super.toString();
55         resp += "Exp=" + experience;
56         resp += "ExpNeeded=" + experienceToLevelUp;
57         return resp;
58     }
59
60     public Integer getSteps() {
61         return steps;
62     }
63
64     public Integer getExperienceToLevelUp() {
65         return experienceToLevelUp;
66     }
67
68     @Override
69     public int hashCode() {
70         final int prime = 31;
71         int result = super.hashCode();
72         result = prime * result
73                 + ((experience == null) ? 0 : experience.hashCode());
74         result = prime
75                 * result
```

```
76                  + ((experienceToLevelUp == null) ? 0 : ↩
                        experienceToLevelUp
77                          .hashCode());
78          result = prime * result + ((steps == null) ? 0 : steps.↩
                hashCode());
79          return result;
80      }
81
82      @Override
83      public boolean equals(Object obj) {
84          if (this == obj)
85              return true;
86          if (!super.equals(obj))
87              return false;
88          if (getClass() != obj.getClass())
89              return false;
90          Player other = (Player) obj;
91          if (experience == null) {
92              if (other.experience != null)
93                  return false;
94          } else if (!experience.equals(other.experience))
95              return false;
96          if (experienceToLevelUp == null) {
97              if (other.experienceToLevelUp != null)
98                  return false;
99          } else if (!experienceToLevelUp.equals(other.↩
                experienceToLevelUp))
100             return false;
101         if (steps == null) {
102             if (other.steps != null)
103                 return false;
104         } else if (!steps.equals(other.steps))
105             return false;
106         return true;
107     }
108
109 }
```

1.1.19. PlayerData.java

```
1  package back;
2
3  public class PlayerData {
4
5      private String name;
6      private int level;
7      private int experience;
8      private int maxHealth;
9      private int health;
10     private int strength;
11     private Point position;
12     private int steps;
13
14     public PlayerData(String name, int level, int experience, int ↩
           health,
15             int maxHealth, int strength, Point position, int steps) {
16         this.level = level;
17         this.name = name;
18         this.experience = experience;
19         this.health = health;
20         this.maxHealth = maxHealth;
21         this.strength = strength;
22         this.position = position;
23         this.steps = steps;
24
25     }
26
27     public int getExperience() {
28         return experience;
```

```
29      }
30
31      public void setExperience(int experience) {
32          this.experience = experience;
33      }
34
35      public int getHealth() {
36          return health;
37      }
38
39      public String getName() {
40          return name;
41      }
42
43      public int getMaxHealth() {
44          return maxHealth;
45      }
46
47      public Point getPosition() {
48          return position;
49      }
50
51      public int getStrength() {
52          return strength;
53      }
54
55      public int getLevel() {
56          return level;
57      }
58
59      public int getSteps() {
60          return steps;
61      }
62
63      public void setName(String name) {
64          this.name = name;
65      }
66
67  }
```

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
7       public void standOver(DungeonGameImp game);
8
9       public boolean isVisible();
10
11      public void setVisible();
12
13      public void setNotVisible();
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 J|Panel. This class is used for
20  *         the Dungeon panel that is into the DungeonGameFrame.
21  *
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<Monster, JLabel[]>();
30
31     public DataPanel(Player player, Color color) {
32         setBackground(Color.WHITE);
33         setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
34         addCharacter(player);
35     }
36
37     public void addCharacter(Player character) {
38         JLabel[] playerLabel = new JLabel[6];
39         playerLabel[0] = new JLabel("  " + character.getName());
40         playerLabel[0].setFont(new Font("Serif", Font.BOLD, 16));
41         playerLabel[0].setForeground(Color.BLUE);
42         playerLabel[1] = new JLabel("  " + "Health: " + character.getHealth()
43                 + "/" + character.getMaxHealth());
44         playerLabel[2] = new JLabel("  " + "Strength: "
45                 + character.getStrength());
46         playerLabel[3] = new JLabel("  " + "Level: " + character.getLevel());
47         playerLabel[4] = new JLabel("  " + "Experience: "
48                 + character.getExperience() + "/"
49                 + 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[] posibleMonsters = new Putable[5];
82          Point p = game.getPlayer().getPosition();
83
84          posibleMonsters[0] = game.getBoard()[p.x + 1][p.y];
85          posibleMonsters[1] = game.getBoard()[p.x - 1][p.y];
86          posibleMonsters[2] = game.getBoard()[p.x][p.y + 1];
87          posibleMonsters[3] = game.getBoard()[p.x][p.y - 1];
88          posibleMonsters[4] = dungeonPanel.getMonsterUnderMouse();
89
90          removeAll();
91
92          for (int i = 0; posibleMonsters[4] != null && i < 4; i++) {
93              if (posibleMonsters[4].equals(posibleMonsters[i])) {
94                  posibleMonsters[4] = null;
95              }
96          }
97
98          addCharacter(game.getPlayer());
99          for (Putable put : posibleMonsters) {
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 ↩
        frame of the
33  *         game.
34  */
35 public class DungeonGameFrame extends GameFrame {
36
37     private static final long serialVersionUID = 1L;
38     private DataPanel dataPanel;
39     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,↩
           Save game,
49      * Save game as..., Load game and Exit
50      *
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 ↩
                                BoardParserFromFile(
74                                  file);
75                          game = new DungeonGameImp(boardObtainer,
76                                  new DungeonGameListenerImp());
77                          setSize();
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          });
91
92          setRestartGameItemAction(new ActionListener() {
93              @Override
94              public void actionPerformed(ActionEvent e) {
95                  try {
96                      if (game == null) {
97                          JOptionPane.showMessageDialog(null,
98                                  "You are not playing a level.");
99                      } else {
100                         game.restart();
101                         dataPanel.setVisible(false);
102                         dungeonPanel.setVisible(false);
103                         remove(dataPanel);
104                         remove(dungeonPanel);
105                         drawDungeonPanel();
106                         drawDataPanel();
107                         dataPanel.refresh(game, dungeonPanel);
108                         dungeonPanel.updateUI();
109                     }
110                 } catch (CorruptedFileException e1) {
111                     JOptionPane.showMessageDialog(null, "The file is ↩
                            corrupt",
112                             "Error", JOptionPane.ERROR_MESSAGE);
113                 }
114             }
115         });
116
117         setSaveGameItemAction(new ActionListener() {
118
119             @Override
120             public void actionPerformed(ActionEvent e) {
121                 if (game != null) {
122                     File directory = new File("." + File.separator
123                             + "savedGames");
124                     if (!directory.exists()) {
125                         directory.mkdir();
126                     }
127                     try {
128                         new SaveGameOnFile(game);
129                     } catch (SavingCorruptedException e1) {
130                         JOptionPane.showMessageDialog(null,
131                                 "Files saving error occours. Try again↩
                                        later.",
132                                 "Error", JOptionPane.ERROR_MESSAGE);
```

```
133                     }
134                 }
135             }
136         });
137
138         setSaveGameAsItemAction(new ActionListener() {
139             @Override
140             public void actionPerformed(ActionEvent e) {
141                 if (game != null) {
142                     File directory = new File("." + File.separator
143                             + "savedGames");
144                     if (!directory.exists()) {
145                         directory.mkdir();
146                     }
147                     File file;
148                     JFileChooser fc = new JFileChooser();
149                     fc.setCurrentDirectory(new File("." + File.↩
                            separator
150                             + "savedGames"));
151                     fc.showOpenDialog(DungeonGameFrame.this);
152                     file = fc.getSelectedFile();
153                     file = new File(file.getPath() + ".board");
154                     if (file == null) {
155                         JOptionPane.showMessageDialog(null,
156                                 "You didn't select any file.");
157                     } else {
158                         try {
159                             new SaveGameOnFile(game, file);
160                         } catch (SavingCorruptedException e1) {
161                             JOptionPane
162                                     .showMessageDialog(
163                                             null,
164                                             "Files saving error ↩
                                                occours. Try again ↩
                                                later.",
165                                             "Error", JOptionPane.↩
                                                ERROR_MESSAGE);
166                         }
167                     }
168                 }
169             }
170         });
171
172         setLoadGameItemAction(new ActionListener() {
173
174             @Override
175             public void actionPerformed(ActionEvent e) {
176                 if (game != null) {
177                     dataPanel.setVisible(false);
178                     dungeonPanel.setVisible(false);
179                     remove(dataPanel);
180                     remove(dungeonPanel);
181                     repaint();
182                     game = null;
183                 }
184                 File file;
185                 JFileChooser fc = new JFileChooser();
186                 fc.setCurrentDirectory(new File("." + File.separator
187                         + "savedGames"));
188                 fc.showOpenDialog(DungeonGameFrame.this);
189                 file = fc.getSelectedFile();
190                 if (file == null) {
191                     JOptionPane.showMessageDialog(null,
192                             "You didn't select any file.");
193                 } else {
194                     try {
195                         LoadGame<DungeonGameImp> loadGame = new ↩
                                LoadGameFromFile<DungeonGameImp>(
196                                     file);
197                         game = loadGame.getGame(DungeonGameImp.class,
198                                 new DungeonGameListenerImp());
199                         setSize();
200                         drawDungeonPanel();
201                         drawDataPanel();
```

```
202                         dataPanel.updateUI();
203                         dungeonPanel.updateUI();
204                     } catch (CorruptedFileException e2) {
205                         JOptionPane
206                                 .showMessageDialog(
207                                         null,
208                                         "Files loading error occours. ↩
                                                Try again later.",
209                                         "Error", JOptionPane.↩
                                                ERROR_MESSAGE);
210                     }
211                 }
212             }
213         });
214
215         setExitGameItemAction(new ActionListener() {
216             @Override
217             public void actionPerformed(ActionEvent e) {
218                 try {
219                     DungeonGameFrame.this.setVisible(false);
220                     DungeonGameFrame.this.dispose();
221                 } catch (Throwable e1) {
222                     JOptionPane.showMessageDialog(null, "Exit fault", ↩
                            "Error",
223                             JOptionPane.ERROR_MESSAGE);
224                 }
225             }
226         });
227
228     }
229
230     private void setSize() {
231         setSize((game.getBoardDimension().y + 2)
232                 * DungeonPanel.CELL_SIZE, (game
233                 .getBoardDimension().x)
234                 * DungeonPanel.CELL_SIZE - 7);
235     }
236
237     /**
238      * Method to make appear the data panel.
239      */
240     private void drawDataPanel() {
241         dataPanel = new DataPanel(game.getPlayer(), Color.GRAY);
242         add(dataPanel, BorderLayout.EAST);
243     }
244
245     /**
246      * Method to make appear the dungeon panel.
247      */
248     private void drawDungeonPanel() {
249         dungeonPanel = new DungeonPanel(game, dataPanel,
250                 new DungeonPanelListenerImp());
251         add(dungeonPanel, BorderLayout.CENTER);
252     }
253
254     /**
255      * Getter of the dungeon panel.
256      *
257      * @return DungeonPanel
258      */
259     public DungeonPanel getDungeonPanel() {
260         return dungeonPanel;
261     }
262
263     /**
264      * Getter of the data panel.
265      *
266      * @return DataPanel
267      */
268     public DataPanel getDataPanel() {
269         return dataPanel;
270     }
271
272     /**
```

```
273         * Listener of the move keys, up down left right.
274         *
275         * @see front.GameFrame#addKeyListener()
276         **/
277        @Override
278        public void addKeyListener() {
279
280            addKeyListener(new KeyAdapter() {
281
282                @Override
283                public void keyPressed(final KeyEvent e) {
284                    switch (e.getKeyCode()) {
285                    case KeyEvent.VK_LEFT:
286                        game.receiveMoveStroke(MoveTypes.LEFT);
287
288                        break;
289                    case KeyEvent.VK_UP:
290                        game.receiveMoveStroke(MoveTypes.UP);
291
292                        break;
293                    case KeyEvent.VK_RIGHT:
294                        game.receiveMoveStroke(MoveTypes.RIGHT);
295
296                        break;
297                    case KeyEvent.VK_DOWN:
298                        game.receiveMoveStroke(MoveTypes.DOWN);
299
300                        break;
301                    }
302                }
303            });
304        }
305
306        /**
307         * @author tmehdi Inner class for the listener of this game ↩
                implementation.
308         */
309        private class DungeonGameListenerImp implements ↩
            DungeonGameListener {
310
311            @Override
312            public void executeWhenBonusGrabed(Point p) {
313                dungeonPanel.drawGrabedBonus(p);
314            }
315
316            @Override
317            public void executeWhenCharacterDie(Point p) {
318                dungeonPanel.drawDiedCharacter(p);
319            }
320
321            @Override
322            public void executeWhenGameLoosed() {
323                JOptionPane.showMessageDialog(DungeonGameFrame.this,
324                        "You loose the level.");
325                DungeonGameFrame.this.remove(DungeonGameFrame.this
326                        .getDungeonPanel());
327                DungeonGameFrame.this.remove(DungeonGameFrame.this.↩
                        getDataPanel());
328                repaint();
329            }
330
331            @Override
332            public void executeWhenGameWinned() {
333                JOptionPane.showMessageDialog(DungeonGameFrame.this, "↩
                    WINNER!"
334                        + '\n' + "You win the level with "
335                        + game.getPlayer().getSteps() + " steps.");
336                DungeonGameFrame.this.remove(DungeonGameFrame.this
337                        .getDungeonPanel());
338                DungeonGameFrame.this.remove(DungeonGameFrame.this.↩
                        getDataPanel());
339                repaint();
340            }
341
```

```
342         @Override
343         public void executeWhenPlayerMoves(MoveTypes moveType) {
344             dungeonPanel.drawPlayerMove(game, moveType);
345             dataPanel.refresh(game, dungeonPanel);
346             dataPanel.updateUI();
347             dungeonPanel.drawDiscoveredCell(game, moveType);
348         }
349 
350         @Override
351         public String playerNameRequest() {
352             String name = null;
353             while (name == null || name.isEmpty()) {
354                 name = JOptionPane.showInputDialog("Player name");
355             }
356             return name;
357         }
358 
359         @Override
360         public void executeWhenFight() {
361             dataPanel.refresh(game, dungeonPanel);
362             dataPanel.updateUI();
363         }
364 
365         @Override
366         public void executeWhenLevelUp() {
367             dungeonPanel.drawLevelUp(game);
368         }
369     }
370 
371     /**
372      * Add the hero image as frame icon.
373      */
374     private void setIcon() {
375         try {
376             setIconImage(loadImage("./resources/images/hero.png"));
377         } catch (IOException e) {
378             JOptionPane.showMessageDialog(null, "Unexpected Error", "↩
                Error",
379                     JOptionPane.ERROR_MESSAGE);
380         }
381     }
382 
383     /**
384      * @author tomas Implementation of DungeonPaneListener used for ↩
            the actions
385      *         performed on dungeonPanel with the mouse.
386      */
387     private class DungeonPanelListenerImp implements ↩
        DungeonPanelListener {
388 
389         @Override
390         public void onMouseMoved(int row, int column) {
391 
392             Monster monster = dungeonPanel.getMonsterUnderMouse();
393             if (monster != null) {
394                 dataPanel.removeCharacter(monster);
395                 dungeonPanel.setMonsterUnderMouse(null);
396             }
397             Putable putable = game.getBoard()[row + 1][column + 1];
398             if (putable instanceof Monster && putable.isVisible()) {
399                 dungeonPanel.setMonsterUnderMouse((Monster) putable);
400                 dataPanel.addCharacter(dungeonPanel.↩
                    getMonsterUnderMouse());
401             }
402             dataPanel.refresh(game, dungeonPanel);
403             dataPanel.updateUI();
404 
405         }
406 
407     }
408 }
```

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 ↩
        GamePanel. This
31  *         class is used for the Dungeon panel that is into the
32  *         DungeonGameFrame.
33  */
34 public class DungeonPanel extends GamePanel {
35
36     private static final long serialVersionUID = 1L;
37     static final int CELL_SIZE = 30;
38
39     private Image playerImage;
40     private Map<Class<? extends Putable>, Image> boardImagesByClass = ↩
           new HashMap<Class<? extends Putable>, Image>();
41     private Map<String, Image> monsterImagesByName = new HashMap<↩
           String, Image>();
42     private Map<String, Image> bonusImagesByName = new HashMap<String,↩
            Image>();
43     private Monster monsterUnderMouse = null;
44
45     /**
46      * @param game
47      * @param dataPanel
48      * @param dungeonListener
49      *             Call the super constructor and draw the pane. The ↩
           interface
50      *             DungeonPanelListener that extends the professor ship↩
            interface
51      *             GamePanelListener is used to make an implementation ↩
           of the
52      *             "onMouseMoved" method. It allows us to know in what ↩
           cell is
53      *             and make the different actions.
54      */
55     public DungeonPanel(Game game, DataPanel dataPanel,
56             DungeonPanelListener dungeonListener) {
57         super(game.getBoardDimension().x - 2, game.getBoardDimension()↩
               .y - 2,
58                 CELL_SIZE, dungeonListener, Color.BLACK);
59         playerImage();
60         boardImagesByClass();
```

```
61          monstersImagesInitialize();
62          bonusImagesInitialize();
63          drawDungeon(game);
64          setVisible(true);
65      }
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      /**
76       * @param dungeonGameFrame
77       *            Draw the full dungeon panel.
78       */
79      public void dwarFullDungeon(DungeonGameFrame dungeonGameFrame) {
80          Image image;
81          Image floorImage = boardImagesByClass.get(Floor.class);
82          Image bloodyFloorImage = overlap(floorImage, ↩
                boardImagesByClass
83                  .get(BloodyFloor.class));
84          int row = dungeonGameFrame.game.getBoardDimension().x - 2;
85          int col = dungeonGameFrame.game.getBoardDimension().y - 2;
86
87          for (int i = 1; i <= row; i++) {
88              for (int j = 1; j <= col; j++) {
89                  Putable cell = dungeonGameFrame.game.getBoard()[i][j];
90                  if (cell.getClass().equals(Monster.class)) {
91                      image = monsterImagesByName.get(((Monster) cell)
92                              .getMonsterType().toString());
93                      image = overlap(floorImage, image);
94                      image = drawString(image, ((Character) cell).↩
                            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).↩
                            getBonusType()
99                              .toString());
100                     image = overlap(floorImage, image);
101                     image = drawString(image, (((Bonus) cell).↩
                            getBonusType()
102                             .getBonusAmount()).toString(), Color.RED);
103                     put(image, i - 1, j - 1);
104                 } else {
105                     image = boardImagesByClass.get(cell.getClass());
106                     if (cell.getClass().equals(Wall.class)) {
107                         put(image, i - 1, j - 1);
108                     } else if (cell.getClass().equals(BloodyFloor.↩
                            class)) {
109                         put(bloodyFloorImage, i - 1, j - 1);
110                     } else {
111                         put(floorImage, i - 1, j - 1);
112                     }
113                 }
114             }
115         }
116
117         Point p = new Point(dungeonGameFrame.game.getPlayer().↩
                getPosition());
118
119         if (dungeonGameFrame.game.getBoard()[p.x][p.y] instanceof ↩
                BloodyFloor) {
120             image = overlap(bloodyFloorImage, playerImage);
121         }
122         image = overlap(floorImage, playerImage);
123         image = drawString(image, dungeonGameFrame.game.getPlayer().↩
                getLevel()
124                 .toString(), Color.WHITE);
125         put(image, p.x - 1, p.y - 1);
126     }
```

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

```
192
193     /**
194      * @param dungeonGameFrame
195      *            Draw the 8 cells around the player and the cell ↩
             under the
196      *            player. Before that draw the player
197      */
198     private void drawDungeonArroundPlayer(Game game) {
199         Image image;
200         Image floorImage = boardImagesByClass.get(Floor.class);
201         Image bloodyFloorImage = overlap(floorImage, ↩
                boardImagesByClass
202                     .get(BloodyFloor.class));
203
204         Point pPos = game.getPlayer().getPosition();
205         pPos = pPos.sub(2, 2);
206
207         for (int i = 1; i <= 3; i++) {
208             for (int j = 1; j <= 3; j++) {
209                 if (pPos.x + i > 0 && pPos.x+i < game.↩
                        getBoardDimension().x-1
210                         && pPos.y + j > 0 && pPos.x+j < game.↩
                            getBoardDimension().y-1) {
211                     Putable cell = game.getBoard()[pPos.x + i][pPos.y ↩
                            + j];
212                     if (cell.getClass().equals(Monster.class)) {
213                         image = monsterImagesByName.get(((Monster) ↩
                                cell)
214                                 .getMonsterType().toString());
215                         image = overlap(floorImage, image);
216                         image = drawString(image, ((Character) cell).↩
                                getLevel()
217                                 .toString(), Color.WHITE);
218                         put(image, pPos.x + i - 1, pPos.y + j - 1);
219                     } else if (cell.getClass().equals(Bonus.class)) {
220                         image = bonusImagesByName.get(((Bonus) cell)
221                                 .getBonusType().toString());
222                         image = overlap(floorImage, image);
223                         image = drawString(image, (((Bonus) cell)
224                                 .getBonusType().getBonusAmount()).↩
                                    toString(),
225                                 Color.RED);
226                         put(image, pPos.x + i - 1, pPos.y + j - 1);
227                     } else {
228                         image = boardImagesByClass.get(cell.getClass()↩
                                );
229                         if (cell.getClass().equals(Wall.class)) {
230                             put(image, pPos.x + i - 1, pPos.y + j - 1)↩
                                    ;
231                         } else if (cell.getClass().equals(BloodyFloor.↩
                                class)) {
232                             put(bloodyFloorImage, pPos.x + i - 1, pPos↩
                                    .y + j
233                                     - 1);
234                         } else {
235                             put(floorImage, pPos.x + i - 1, pPos.y + j↩
                                     - 1);
236                         }
237                     }
238                 }
239             }
240         }
241
242         Point p = new Point(game.getPlayer().getPosition());
243
244         if (game.getBoard()[p.x][p.y] instanceof BloodyFloor) {
245             image = overlap(bloodyFloorImage, playerImage);
246         }
247         image = overlap(floorImage, playerImage);
248         image = drawString(image, game.getPlayer().getLevel().toString↩
                (),
249                 Color.WHITE);
250         put(image, p.x - 1, p.y - 1);
251     }
```

```
252
253     /**
254      * @return Getter of the monsterUnderMouse.
255      */
256     public Monster getMonsterUnderMouse() {
257         return monsterUnderMouse;
258     }
259
260     /**
261      * @param game
262      *             of class Game
263      * @param moveType
264      *             instance of enumerative MoveTypes
265      *
266      *             Redraw if necessary the DungeonPanel.
267      */
268     public void drawPlayerMove(Game game, MoveTypes moveType) {
269         Image bloodyFloor;
270         Image floor;
271         Point afterMove = new Point(game.getPlayer().getPosition().x, ↩
                game
272                 .getPlayer().getPosition().y);
273         Point beforeMove = afterMove.sub(moveType.getDirection());
274         floor = boardImagesByClass.get(Floor.class);
275         bloodyFloor = boardImagesByClass.get(BloodyFloor.class);
276         bloodyFloor = overlap(floor, bloodyFloor);
277         clear(beforeMove.x - 1, beforeMove.y - 1);
278         if (game.getBoard()[beforeMove.x][beforeMove.y].getClass().↩
                equals(
279                 BloodyFloor.class)) {
280             put(bloodyFloor, beforeMove.x - 1, beforeMove.y - 1);
281         } else {
282             put(floor, beforeMove.x - 1, beforeMove.y - 1);
283         }
284
285         clear(afterMove.x - 1, afterMove.y - 1);
286         Image image;
287         if (game.getBoard()[afterMove.x][afterMove.y].getClass().↩
                equals(
288                 BloodyFloor.class)) {
289             image = overlap(bloodyFloor, playerImage);
290             image = drawString(image, game.getPlayer().getLevel().↩
                    toString(),
291                     Color.WHITE);
292             put(image, afterMove.x - 1, afterMove.y - 1);
293         } else {
294             image = overlap(floor, playerImage);
295             image = drawString(image, game.getPlayer().getLevel().↩
                    toString(),
296                     Color.WHITE);
297
298             put(image, afterMove.x - 1, afterMove.y - 1);
299         }
300         updateUI();
301     }
302
303     /**
304      * @param p
305      *
306      *             Draw blood on the floor where a character die.
307      */
308     public void drawDiedCharacter(Point p) {
309         Image imagFloor = boardImagesByClass.get(Floor.class);
310         Image imagBloodFloor = boardImagesByClass.get(BloodyFloor.↩
                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     /**
318      * @param p
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, MoveTypes dir) {
331            Point pPos = game.getPlayer().getPosition();
332            List<Point> points = new ArrayList<Point>();
333            points.add(pPos.add(dir.getDirection()));
334            if (dir == MoveTypes.LEFT || dir == MoveTypes.RIGHT) {
335                points.add(pPos.add(1, 0).add(dir.getDirection()));
336                points.add(pPos.sub(1, 0).add(dir.getDirection()));
337            } else {
338                points.add(pPos.add(0, 1).add(dir.getDirection()));
339                points.add(pPos.sub(0, 1).add(dir.getDirection()));
340            }
341
342            Image image;
343            Image floorImage = boardImagesByClass.get(Floor.class);
344            Image bloodyFloorImage = overlap(floorImage, ↩
                    boardImagesByClass
345                        .get(BloodyFloor.class));
346
347            for (Point p : points) {
348                if (p.x > 0 && p.x < game.getBoardDimension().x - 1 && p.y↩
                        > 0
349                        && p.y < game.getBoardDimension().y - 1) {
350                    if (game.getBoard()[p.x][p.y].isVisible()) {
351                        game.getBoard()[p.x][p.y].setVisible();
352                        Putable cell = game.getBoard()[p.x][p.y];
353                        if (cell.getClass().equals(Monster.class)) {
354                            image = monsterImagesByName.get(((Monster) ↩
                                    cell)
355                                    .getMonsterType().toString());
356                            image = overlap(floorImage, image);
357                            image = drawString(image, ((Character) cell).↩
                                    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().toString());
363                            image = overlap(floorImage, image);
364                            image = drawString(image, (((Bonus) cell)
365                                    .getBonusType().getBonusAmount()).↩
                                            toString(),
366                                    Color.RED);
367                            put(image, p.x - 1, p.y - 1);
368                        } else {
369                            image = boardImagesByClass.get(cell.getClass()↩
                                    );
370                            if (cell.getClass().equals(Wall.class)) {
371                                put(image, p.x - 1, p.y - 1);
372                            } else if (cell.getClass().equals(BloodyFloor.↩
                                    class)) {
373                                put(bloodyFloorImage, p.x - 1, p.y - 1);
374                            } else {
375                                put(floorImage, p.x - 1, p.y - 1);
376                            }
377                        }
378                    }
379                }
380            }
381
382        }
383
384        /**
385         * Method to initialize player image.
386         */
```

```
387     private void playerImage() {
388         try {
389             playerImage = loadImage("./resources/images/hero.png");
390         } catch (IOException e) {
391             JOptionPane.showMessageDialog(null, "Unexpected Error", "↩
                    Error",
392                     JOptionPane.ERROR_MESSAGE);
393         }
394     }
395 
396     /**
397      * Method to initialize board images.
398      */
399     private void boardImagesByClass() {
400         try {
401             boardImagesByClass.put(Wall.class,
402                     loadImage("./resources/images/wall.png"));
403             boardImagesByClass.put(Floor.class,
404                     loadImage("./resources/images/background.png"));
405             boardImagesByClass.put(BloodyFloor.class,
406                     loadImage("./resources/images/blood.png"));
407         } catch (IOException e) {
408             JOptionPane.showMessageDialog(null, "Unexpected Error", "↩
                    Error",
409                     JOptionPane.ERROR_MESSAGE);
410         }
411     }
412 
413     /**
414      * Method to initialize bonus images.
415      */
416     private void bonusImagesInitialize() {
417         try {
418             bonusImagesByName.put("LIFE",
419                     loadImage("./resources/images/healthBoost.png"));
420             bonusImagesByName.put("STRENGTH",
421                     loadImage("./resources/images/attackBoost.png"));
422         } catch (IOException e) {
423             JOptionPane.showMessageDialog(null, "Unexpected Error", "↩
                    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", "↩
                    Error",
441                     JOptionPane.ERROR_MESSAGE);
442         }
443     }
444 
445     public void drawLevelUp(Game game) {
446         Image image;
447         Image bloodyFloor;
448         Image floor;
449         Point playerPos = new Point(game.getPlayer().getPosition().x, ↩
                game
450                 .getPlayer().getPosition().y);
451         floor = boardImagesByClass.get(Floor.class);
452         bloodyFloor = boardImagesByClass.get(BloodyFloor.class);
453         bloodyFloor = overlap(floor, bloodyFloor);
454 
455         clear(playerPos.x - 1, playerPos.y - 1);
```

```
456             if (game.getBoard()[playerPos.x][playerPos.y] instanceof ↩
                    BloodyFloor) {
457                 image = overlap(bloodyFloor, playerImage);
458                 image = drawString(image, game.getPlayer().getLevel().↩
                        toString(),
459                             Color.WHITE);
460                 put(image, playerPos.x - 1, playerPos.y - 1);
461             } else {
462                 image = overlap(floor, playerImage);
463                 image = drawString(image, game.getPlayer().getLevel().↩
                        toString(),
464                             Color.WHITE);
465 
466                 put(image, playerPos.x - 1, playerPos.y - 1);
467             }
468             updateUI();
469         }
470 
471 }
```

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 ↩
       DefaultGameMenuBar {
15 
16     private static final long serialVersionUID = 1L;
17     private static final int CELL_SIZE = 30;
18     public Game game;
19     private JMenuBar menuBar;
20     private JMenu fileMenu;
21     private JMenuItem newGameItem;
22     private JMenuItem restartGameItem;
23     private JMenuItem saveGameItem;
24     private JMenuItem saveGameAsItem;
25     private JMenuItem loadGameItem;
26     private JMenuItem exitGameItem;
27 
28     public GameFrame(String name) {
29         super(name);
30         setTitle(name);
```

```
31          setSize(13 * CELL_SIZE + 26, 11 * CELL_SIZE + 20);
32          menuBar = new JMenuBar();
33          fileMenu = new JMenu("File");
34          newGameItem = fileMenu.add("New game");
35          restartGameItem = fileMenu.add("Restart");
36          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  import java.util.ArrayList;
6  import java.util.List;
7
8  import javax.swing.JFrame;
9  import javax.swing.JOptionPane;
10
11 /**
12  * @author tomas Class for show the player a list of levels that are saved on
13  *         the directory boards. It use a list of directorys and some 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
24         String[] auxFiles, listBoardsShowed;
25         List<String> listBoards = new ArrayList<String>();
26         File directory = new File("." + File.separator + "boards");
27         auxFiles = directory.list();
28         for (String s : auxFiles) {
29             if (s.endsWith(".board")) {
30                 listBoards.add(s.replace(".board", ""));
31             }
32         }
33         listBoardsShowed = new String[listBoards.size()];
34         for (int k = 0; k < listBoards.size(); k++) {
35             listBoardsShowed[k] = listBoards.get(k);
36         }
37
38         Object levelSelected = JOptionPane.showInputDialog(frameToShowOn,
39                 "Select level", "Levels selector",
40                 JOptionPane.QUESTION_MESSAGE, null, listBoardsShowed,
41                 listBoardsShowed[0]);
42         if (levelSelected != null) {
43             this.levelSelected = new File("." + File.separator + "boards"
44                     + File.separator + levelSelected + ".board");
45         }
46
47     }
48
49     public File getLevelSelected() {
50         return levelSelected;
51     }
52
```

```
53 }
```

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(elemsCuantity, 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 elemsCuantity = 6;
8      private Point boardDimension;
9
10     public BoardLine(String line, Point boardDimension) {
11         super(elemsCuantity, 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 ↩
            sets the
19      * cell into the board.
20      */
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↩
                    [2] < 0
29                     || data[2] >= boardDimension.y - 2 || data[3] != 0
30                     || data[4] != 0 || data[5] != 0) {
31                 throw new CorruptedFileException();
32             }
```

```
33              break;
34
35          case 2:
36              // Wall
37              if (data[1] < 0 || data[1] >= boardDimension.x - 2 || data↩
                    [2] < 0
38                      || data[2] >= boardDimension.y - 2 || data[4] != ↩
                            0) {
39                  throw new CorruptedFileException();
40              }
41              break;
42
43          case 3:
44              // Monster
45              if (data[1] < 0 || data[1] >= boardDimension.x - 2 || data↩
                    [2] < 0
46                      || data[2] >= boardDimension.y - 2 || data[3] <= 0
47                      || data[3] > 3 || data[4] <= 0 || data[4] > 3) {
48                  throw new CorruptedFileException();
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 elemsCuantity = 1;
6       private String name;
7
8       public BoardNameLine(String line) {
9           super(elemsCuantity, 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.DungeonGameImp;
11  import back.Floor;
12  import back.Monster;
13  import back.PlayerData;
14  import back.Point;
15  import back.Putable;
16  import back.Wall;
17
18  /**
19   * @author tomas Class full dedicated to read a file and transform it ↩
         to a
20   *         board.
21   */
22  public class BoardParserFromFile implements BoardObtainer {
23
24      private BufferedReader inputBoard;
25      private Point boardDimension;
26      private String boardName;
27      private Point playerPosition;
28      private Putable[][] board;
29      private File inputFile;
30
31      public BoardParserFromFile(File file) {
32          try {
33              inputFile = file;
34              inputBoard = new BufferedReader(new FileReader(file));
35              obtainBoard();
36          } catch (IOException e) {
37              throw new CorruptedFileException();
38          }
39      }
40
41      public void obtainBoard() throws IOException {
42
43          boolean dimensionFlag = false;
44          boolean nameFlag = false;
45          boolean playerFlag = false;
46          String line;
47
48          while ((line = inputBoard.readLine()) != null) {
```

```
49
50              line = line.replace(" ", "").replace("\t", "").replace("\n", "")
51                  .split("#")[0];
52
53              if (!line.isEmpty()) {
54                  if (!dimensionFlag) {
55                      parseDimension(line);
56                      dimensionFlag = true;
57                  } else if (!nameFlag) {
58                      parseBoardName(line);
59                      nameFlag = true;
60                  } else {
61                      if (line.startsWith("1")) {
62                          if (playerFlag == true) {
63                              throw new CorruptedFileException();
64                          }
65                          parsePlayer(line);
66                          playerFlag = true;
67                      } else {
68                          BoardLine cell = new BoardLine(line,
                                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
84          if (!nameFlag || !playerFlag || !dimensionFlag) {
85              throw new CorruptedFileException();
86          }
87          validation();
88      }
89
90      public void validation() {
91          protectionWalls();
92          putFloor();
93          if (!(board[getPlayerPosition().x][getPlayerPosition().y]
                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),
                cell
100                 .getData(5)));
101     }
102
103     public void parsePlayer(String line) {
104         BoardLine cell = new BoardLine(line, boardDimension);
105         Point point = (new Point(cell.getData(1), cell.getData(2)))
106                 .add(new Point(1, 1));
107         playerPosition = point;
108     }
109
110     public void parseMonster(Point point, BoardLine cell) {
111         putCell(point.x, point.y, new Monster(point, cell.getData(3),
                cell
112                 .getData(4)));
113     }
114
115     public void parseWall(Point point, BoardLine cell) {
116         putCell(point.x, point.y, new Wall());
117     }
```

```
118
119     public void parseBoardName(String line) {
120         BoardNameLine boardNameLine = new BoardNameLine(line);
121         this.boardName = boardNameLine.getName();
122     }
123
124     public void parseDimension(String line) {
125         BoardDimensionLine boardDimensionLine = new BoardDimensionLine↩
                (line);
126         boardDimension = boardDimensionLine.getBoardDimension().add(
127                 new Point(2, 2));
128         board = new Putable[boardDimension.x][boardDimension.y];
129
130     }
131
132     public void putFloor() {
133         for (int i = 1; i < boardDimension.x - 1; i++) {
134             for (int j = 1; j < boardDimension.y - 1; j++) {
135                 if (getBoardElem(i, j) == null) {
136                     putCell(i, j, new Floor());
137                 }
138             }
139         }
140     }
141
142     public void protectionWalls() {
143         for (int i = 0; i < boardDimension.y; i++) {
144             Wall aux = new Wall();
145             aux.setVisible();
146             putCell(0, i, aux);
147             Wall aux1 = new Wall();
148             aux1.setVisible();
149             putCell(boardDimension.x - 1, i, aux1);
150         }
151         for (int i = 0; i < boardDimension.x; i++) {
152             Wall aux = new Wall();
153             aux.setVisible();
154             putCell(i, 0, aux);
155             Wall aux1 = new Wall();
156             aux1.setVisible();
157             putCell(i, boardDimension.y - 1, aux1);
158         }
159
160     }
161
162     public Point getBoardDimension() {
163         return boardDimension;
164     }
165
166     public String getBoardName() {
167         return boardName;
168     }
169
170     public Point getPlayerPosition() {
171         return playerPosition;
172     }
173
174     public Putable[][] getBoard() {
175         return board;
176     }
177
178     public int getBoardRows() {
179         return boardDimension.x;
180     }
181
182     public int getBoardColums() {
183         return boardDimension.y;
184     }
185
186     public Putable getBoardElem(Point position) {
187         return board[position.x][position.y];
188     }
189
190     public Putable getBoardElem(int x, int y) {
```

```
191         return board[x][y];
192     }
193
194     public void putCell(int i, int j, Putable cell) {
195         putCell(new Point(i, j), cell);
196     }
197
198     public void putCell(Point p, Putable cell) {
199         board[p.x][p.y] = cell;
200     }
201
202     @Override
203     public File getFile() {
204         return inputFile;
205     }
206
207     @Override
208     public PlayerData getPlayerData() {
209         PlayerData playerData = new PlayerData(null, 1, 0, ↩
                DungeonGameImp.LIFE, DungeonGameImp.LIFE, DungeonGameImp.↩
                STRENGTH,
210                 playerPosition, 0);
211         return playerData;
212     }
213
214 }
```

1.3.5. CorruptedFileException.java

```
1  package parser;
2
3  public class CorruptedFileException extends RuntimeException {
4
5      private static final long serialVersionUID = 1L;
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 elemsCuantity;
7      private String line;
8
9      public Lines(int elemsCuantity, String line) {
10         this.elemsCuantity = elemsCuantity;
11         this.line = line;
12     }
13
14     /**
15      * Process the line parsed by separating it by "," and removing ↩
            the spaces,
16      * enters and tabs in between.
17      *
18      */
19     protected void lineProcess() {
20         data = new int[elemsCuantity];
21         int k = 0;
22         String[] arrayString;
23
```

```
24          arrayString = line.split(",");
25
26          if (arrayString.length == elemsCuantity) {
27              for (k = 0; k < elemsCuantity; 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 elemsCuantity = 10;
8       private Point boardDimension;
9       private String playerName;
10
11      public SavedBoardPlayerLine(String line, Point boardDimension) {
12          super(elemsCuantity, line);
13          this.boardDimension = boardDimension;
14          lineProcess();
15          lineCheck();
16      }
17
18      @Override
19      protected void lineProcess() {
20          data = new int[elemsCuantity];
21          int k = 0;
22          String[] arrayString;
23
24          arrayString = getLine().split(",");
25
26          if (arrayString.length == elemsCuantity) {
27              for (k = 0; k < elemsCuantity - 1; k++) {
28                  try {
29                      data[k] = Integer.valueOf(arrayString[k]);
30                  } catch (NumberFormatException e) {
31                      throw new CorruptedFileException();
32                  }
33              }
34              playerName = arrayString[elemsCuantity - 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] ↩
                < 0
44                  || 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 imÃ¡genes, siendo posible ↩
         agregarle y quitarle imÃ¡genes. Asimismo, cuenta con una
13   * interfaz que permite a quien la utilice ser notificada cuando el ↩
         usuario posiciona el mouse sobre una celda de la grilla.
14   */
15  public class GamePanel extends JPanel {
16
17      private int rows, columns;
18      private int cellSize;
19      private Color color;
20      private Image[][] images;
21
22      /**
23       * Crea un nuevo panel con las dimensiones indicadas.
24       *
25       * @param rows Cantidad de filas.
26       * @param columns Cantidad de columnas.
27       * @param cellSize Ancho y alto de cada imagen en pÃxeles.
28       * @param listener Listener que serÃ¡ notificado cuando el usuario↩
             se posicione sobre una celda de la grilla.
29       * @param color Color de fondo del panel.
30       */
31      public GamePanel(final int rows, final int columns, final int ↩
             cellSize, final GamePanelListener listener, Color color) {
32          setSize(columns * cellSize, rows * cellSize);
33          images = new Image[rows][columns];
34          this.rows = rows;
35          this.columns = columns;
36          this.cellSize = cellSize;
37          this.color = color;
38
39          addMouseMotionListener(new MouseMotionAdapter() {
40
41              private Integer currentRow;
42              private Integer currentColumn;
43
44              @Override
```

```
45              public void mouseMoved(MouseEvent e) {
46                  int row = e.getY() / cellSize;
47                  int column = e.getX() / cellSize;
48                  if (row >= rows || column >= columns || row < 0 || ↩
                        column < 0) {
49                      return;
50                  }
51
52                  if (!nullSafeEquals(currentRow, row) || !↩
                        nullSafeEquals(currentColumn, column)) {
53                      currentRow = row;
54                      currentColumn = column;
55                      listener.onMouseMoved(row, column);
56                  }
57              }
58
59              private boolean nullSafeEquals(Object o1, Object o2) {
60                  return o1 == null ? o2 == null : o1.equals(o2);
61              }
62          });
63      }
64
65      /**
66       * Ubica una imagen en la fila y columna indicadas.
67       */
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 ↩
                algun problema al leer el archivo lanza una
22      * excepcion.
23      */
24     public static Image loadImage(String fileName) throws IOException ↩
            {
25         InputStream stream = ClassLoader.getSystemResourceAsStream(↩
                fileName);
26         if (stream == null) {
27             return ImageIO.read(new File(fileName));
28         } else {
29             return ImageIO.read(stream);
30         }
31     }
32
33     /**
34      * Dibuja un texto en el vértice inferior derecho de la imagen, ↩
                con el color indicado. Retorna una imagen nueva con
35      * los cambios, la imagen original no se modifica.
36      */
37     public static Image drawString(Image img, String text, Color color↩
            ) {
38         BufferedImage result = new BufferedImage(img.getWidth(null), ↩
                img.getHeight(null), BufferedImage.TYPE_INT_ARGB);
39         Graphics2D g = (Graphics2D) result.getGraphics();
40         g.drawImage(img, 0, 0, null);
41
42         Font font = new Font(Font.SANS_SERIF, Font.BOLD, 12);
43         g.setFont(font);
44         g.setColor(color);
45         Rectangle2D r = font.getStringBounds(text, g.↩
                getFontRenderContext());
46         g.drawString(text, img.getWidth(null) - (int) r.getWidth() - ↩
                2, img.getHeight(null) - 2);
47         return result;
48     }
49
50     /**
51      * Superpone dos imágenes. Retorna una nueva imagen con las 2 ↩
                imágenes recibidas superpuestas. Las
52      * originales no se modifican.
53      */
54     public static Image overlap(Image image1, Image image2) {
55         BufferedImage result = new BufferedImage(image1.getWidth(null)↩
                , image1.getHeight(null),
56                 BufferedImage.TYPE_INT_ARGB);
57         Graphics2D g = (Graphics2D) result.getGraphics();
58         g.drawImage(image1, 0, 0, null);
59         g.drawImage(image2, 0, 0, null);
```

```
60          return result;
61      }
62  }
```

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

```java
package saveLoadImplementation;

import java.io.File;
import java.util.List;

public interface FilterFileList extends List<File>{

    public FilterFileList filter(String string);

}
```

1.5.4. LoadGameFromFile.java

```java
package saveLoadImplementation;

import java.io.File;

import parser.BoardLine;
import parser.BoardParserFromFile;
import parser.CorruptedFileException;
import parser.SavedBoardPlayerLine;
import back.BloodyFloor;
import back.BoardObtainer;
import back.Floor;
import back.Game;
import back.GameListener;
import back.LoadGame;
import back.Monster;
import back.PlayerData;
import back.Point;

public class LoadGameFromFile<T extends Game> extends ↩
    BoardParserFromFile
        implements LoadGame<T> {

    private Point playerLoadedPosition;
    private Integer loadedLevel;
    private Integer playerLoadedExperience;
    private Integer playerLoadedHealth;
    private Integer playerLoadedMaxHealth;
    private Integer playerLoadedStrength;
    private Integer playerLoadedSteps;
    private String playerName;

    public LoadGameFromFile(File placeToLoad) {
        super(placeToLoad);
    }

    @Override
    public void parsePlayer(String line) {
        SavedBoardPlayerLine playerData = new SavedBoardPlayerLine(↩
            line,
                getBoardDimension());
        Point point = (new Point(playerData.getData(1), playerData.↩
            getData(2)))
                .add(new Point(1, 1));
        playerLoadedPosition = point;
        playerLoadedExperience = playerData.getData(3);
        playerLoadedHealth = playerData.getData(4);
        playerLoadedMaxHealth = playerData.getData(5);
        playerLoadedStrength = playerData.getData(6);
        playerLoadedSteps = playerData.getData(7);
        loadedLevel = playerData.getData(8);
        playerName = playerData.getPlayerName();

    }

    private void setBoardCellVisivility(Point point, int num) {
        if (num == 0) {
```

```
54              getBoardElem(point).setVisible();
55          } else {
56              getBoardElem(point).setNotVisible();
57          }
58      }
59
60      @Override
61      public void parseWall(Point point, BoardLine cell) {
62          if (cell.getData(3) == 2) {
63              putCell(point, new BloodyFloor());
64          } else if (cell.getData(3) == 1) {
65              putCell(point, new Floor());
66          } else {
67              super.parseWall(point, cell);
68          }
69          setBoardCellVisivility(point, cell.getData(5));
70      };
71
72      @Override
73      public void parseBonus(Point point, BoardLine cell) {
74          super.parseBonus(point, cell);
75          setBoardCellVisivility(point, cell.getData(4));
76      }
77
78      @Override
79      public void parseMonster(Point point, BoardLine cell) {
80          putCell(point.x, point.y, new Monster(point, cell.getData(3), ↩
                cell
81                  .getData(4), Math.abs(cell.getData(5))));
82          if (cell.getData(5) < 0) {
83              setBoardCellVisivility(point, 0);
84          } else if (cell.getData(5) > 0) {
85              setBoardCellVisivility(point, 1);
86          }
87      }
88
89      @Override
90      public Point getPlayerPosition() {
91          return playerLoadedPosition;
92      }
93
94      @Override
95      public Integer getPlayerLoadedHealth() {
96          return playerLoadedHealth;
97      }
98
99      @Override
100     public Integer getPlayerLoadedMaxHealth() {
101         return playerLoadedMaxHealth;
102     }
103
104     @Override
105     public Integer getPlayerLoadedExperience() {
106         return playerLoadedExperience;
107     }
108
109     @Override
110     public Integer getPlayerLoadedStrength() {
111         return playerLoadedStrength;
112     }
113
114     @Override
115     public Integer getPlayerLoadedSteps() {
116         return playerLoadedSteps;
117     }
118
119     public T getGame(Class<T> gameImpClass, GameListener listener) {
120         T game;
121         try {
122             game = gameImpClass.getConstructor(BoardObtainer.class,
123                     GameListener.class).newInstance(this, listener);
124         } catch (Exception e) {
125             e.printStackTrace();
126             throw new CorruptedFileException();
```

```
127         }
128         return game;
129     }
130 
131     @Override
132     public int getPlayerLoadedLevel() {
133         return loadedLevel;
134     }
135 
136     @Override
137     public String getPlayerName() {
138         return playerName;
139     }
140 
141     @Override
142     public PlayerData getPlayerData() {
143         PlayerData playerData = new PlayerData(playerName, loadedLevel↩
                ,
144                 playerLoadedExperience, playerLoadedHealth,
145                 playerLoadedMaxHealth, playerLoadedStrength,
146                 playerLoadedPosition, playerLoadedSteps);
147         return playerData;
148     }
149 
150 }
```

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                     + ".board");
33         } else {
34             placeToSave = new File("./savedGames/savedGame.board");
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                      + ")"+ ").board");
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
                name
65       * ("Board name") player (1,row pos, col pos,exp,health,max health
                ,
66       * strength, steps, level, name) walls (2,row pos, col pos, 0 ,0,
                [0 is
67       * visible 1 not visible]) bloodyFloor(2,row pos, col pos, 2 ,0,
                [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.board")),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 executeWhenGameWinned() {
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          System.out.println(game.getPlayer().getExperience());
77          assertEquals(new Integer(1), game.getPlayer().getExperience())↩
                ;
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                     + ")" + ".board");
128             assertTrue(f.exists());
129             f.delete();
130         } else {
131             File f = new File("./savedGames/savedGame.board");
132             assertTrue(f.exists());
133             f.delete();
134         }
135     }
136
137     @Test
138     public void loadGameTest() {
139         File file = new File("./savedGames/testWithPath.board");
140         new SaveGameOnFile(game, file);
141         LoadGame<DungeonGameImp> loadGame = new LoadGameFromFile<↩
                DungeonGameImp>(file);
142         DungeonGameImp game = loadGame.getGame(DungeonGameImp.class, ↩
                new DungeonGameListener() {
143
144             @Override
145             public String playerNameRequest() {
146                 String name = null;
147                 while (name == null || name.isEmpty()) {
148                     name = JOptionPane.showInputDialog("Player name");
149                 }
150                 return name;
151             }
152
153             @Override
154             public void executeWhenPlayerMoves(MoveTypes moveType) {
155             }
156
157             @Override
158             public void executeWhenGameWinned() {
159             }
160
161             @Override
162             public void executeWhenGameLoosed() {
163             }
164
165             @Override
166             public void executeWhenCharacterDie(Point p) {
167             }
168
169             @Override
170             public void executeWhenBonusGrabed(Point p) {
171             }
172
173             @Override
```

```
174                 public void executeWhenFight() {
175                 }
176
177                 @Override
178                 public void executeWhenLevelUp() {
179                 }
180             });
181             assertEquals(new Integer(0), game.getPlayer().getExperience())
                    ;
182             assertEquals(new Point(4, 4), game.getPlayer().getPosition());
183             file.delete();
184         }
185
186         @Test
187         public void forWatchTheGameSavedWithPathTest() {
188             File directory = new File("./savedGames.board");
189             if (!directory.exists()) {
190                 directory.mkdir();
191             }
192             File file = new File("./savedGames/testWithPath.board");
193             new SaveGameOnFile(game, file);
194             FilterFileList filterFileList = new FilterArrayFileList(
195                     file.getParentFile());
196             filterFileList = filterFileList.filter(file.getName());
197             int number = filterFileList.size();
198             if (number > 1) {
199                 File f = new File(file.getPath() + "(" + (number - 1) + ")
                        ");
200                 assertTrue(f.exists());
201                 f.delete();
202             } else {
203                 File f = new File(file.getPath());
204                 assertTrue(f.exists());
205                 f.delete();
206             }
207         }
208
209 }
```

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.board"));
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.↩
                        getStrength()),
50                  player.getHealth());
51          assertEquals(
52                  new Integer(monster.getMaxHealth() - player.↩
                        getStrength()),
53                  monster.getHealth());
54      }
55
56      @Test
57      public void goodFunctionamientPlayerEarningBonusTest() {
58          player.hited(9);
59          ((Bonus) boardParser.getBoard()[8][2]).giveBonus(player);
60          ((Bonus) boardParser.getBoard()[2][8]).giveBonus(player);
61          assertEquals(new Integer(6), player.getHealth());
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.board"));
```

```
27      }
28
29      @Test(expected = CorruptedFileException.class)
30      public void startPlayerPositionOverAMonsterTest() {
31          new BoardParserFromFile(new File("./testBoard/boardForTest2.↩
                board"));
32      }
33
34      @Test(expected = CorruptedFileException.class)
35      public void startPlayerPositionOverAWallTest() {
36          new BoardParserFromFile(new File("./testBoard/boardForTest3.↩
                board"));
37      }
38
39      @Test
40      public void mapWithoutSurroundingWalls() {
41          BoardObtainer boardParser = new BoardParserFromFile(new File(
42                  "./testBoard/boardForTest4.board"));
43          assertEquals(Wall.class, boardParser.getBoardElem(new Point(0,↩
                 0))
44                  .getClass());
45          assertEquals(Wall.class, boardParser.getBoardElem(new Point↩
                (11, 0))
46                  .getClass());
47          assertEquals(Wall.class, boardParser.getBoardElem(new Point(0,↩
                 11))
48                  .getClass());
49          assertEquals(Wall.class, boardParser.getBoardElem(new Point↩
                (11, 11))
50                  .getClass());
51      }
52
53      @Test(expected = CorruptedFileException.class)
54      public void positionOutOfBoardDimensionsTest() {
55          new BoardParserFromFile(new File("./testBoard/boardForTest5.↩
                board"));
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      @Test
69      public void goodParseOfBoardNameTest() {
70          assertEquals("ejemplotablero", boardParser.getBoardName());
71      }
72
73      @Test
74      public void goodParseOfPlayerPositionTest() {
75          assertEquals(new Point(4, 4), boardParser.getPlayerPosition())↩
                ;
76      }
77
78      @Test
79      public void goodParseOfAnyCellPositionTest() {
80          assertEquals(Wall.class, boardParser.getBoard()[1][1].getClass↩
                ());
81          assertEquals(Wall.class, boardParser.getBoard()[10][1].↩
                getClass());
82          assertEquals(Wall.class, boardParser.getBoard()[1][10].↩
                getClass());
83          assertEquals(Wall.class, boardParser.getBoard()[10][10].↩
                getClass());
84          assertEquals(Bonus.class,
85                  boardParser.getBoard()[2][8].getClass());
86          assertEquals(Bonus.class, boardParser.getBoard()[8][2].↩
                getClass());
```

```
87          assertEquals(Monster.class, boardParser.getBoard()[5][7].↩
                getClass());
88          assertEquals(Monster.class, boardParser.getBoard()[3][6].↩
                getClass());
89          assertEquals(Monster.class, boardParser.getBoard()[2][4].↩
                getClass());
90      }
91
92      @Test
93      public void goodParseOfMonsterTest() {
94          assertEquals(MonsterTypes.DRAGON,
95                  ((Monster) boardParser.getBoard()[9][2]).↩
                        getMonsterType());
96          assertEquals(new Integer(3),
97                  ((Monster) boardParser.getBoard()[9][2]).getLevel());
98      }
99
100     @Test
101     public void goodParseOfBonusTest() {
102         assertEquals(5,
103                 ((Bonus) boardParser.getBoard()[8][2]).getAmountBonus↩
                        ());
104         assertEquals(3,
105                 ((Bonus) boardParser.getBoard()[2][8])
106                         .getAmountBonus());
107     }
108
109     @Test
110     public void boardWatchTest() {
111         String resp = "";
112         for (int i = 0; i < boardParser.getBoardRows(); i++) {
113             for (int j = 0; j < boardParser.getBoardColums(); j++) {
114                 resp += boardParser.getBoard()[i][j] + " ";
115             }
116             resp += "\n";
117         }
118         System.out.println(resp);
119     }
120
121 }
```