# 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 int getPlayerSteps();
26
```

```
27 }
```

### 1.1.4. Bonus.java

```
1  package back;
2
3  public class Bonus extends Cell implements Putable {
4
5      private BonusTypes bonusType;
6
7      public Bonus(Point position, int numberBonusType, int bonusAmount)↩
           {
8          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     final static Integer LEVEL = 3;
14     final static Integer LIFE = 10;
15     final static Integer STRENGTH = 5;
16
17     private String boardName;
18     private Player player;
19     private Point boardDimension;
20     private Putable[][][] board;
21     private GameListener gameListener;
22     private BoardObtainer boardObtainer;
23
24     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         setPlayer();
31         firstDiscoveredCells();
32     }
33
34     private void setPlayer() {
35         if (!(boardObtainer instanceof LoadGame)) {
36             PlayerData playerData = new PlayerData(gameListener
37                     .playerNameRequest(), 1, 0, LIFE, LIFE, STRENGTH,
38                     boardObtainer.getPlayerPosition(), boardObtainer
39                             .getPlayerSteps());
40             player = new Player(playerData);
41         } else {
42             player = ((LoadGame)boardObtainer).getLoadedPlayer();
43         }
44
45     }
46
47     private void firstDiscoveredCells() {
48         Point p = player.getPosition();
49
50         board[p.x][p.y].setVisible();
51
52         board[p.x + 1][p.y - 1].setVisible();
53         board[p.x + 1][p.y].setVisible();
54         board[p.x + 1][p.y + 1].setVisible();
55
56         board[p.x][p.y - 1].setVisible();
57         board[p.x][p.y].setVisible();
58         board[p.x][p.y + 1].setVisible();
59
60         board[p.x - 1][p.y - 1].setVisible();
61         board[p.x - 1][p.y].setVisible();
62         board[p.x - 1][p.y + 1].setVisible();
63     }
64
65     /**
66      * @see back.Game#receiveMoveStroke(back.MoveTypes) Is't allow the game to
67      *      receive a Stroke. In this case a MoveTypes stroke. Before this the
68      *      player moves.
69      **/
70     @Override
71     public void receiveMoveStroke(MoveTypes moveType) {
72         Point nextPlayerPosition = player.getPosition().add(
73                 moveType.getDirection());
74         int playerLevelBeforeFight = player.getLevel();
75         if (board[nextPlayerPosition.x][nextPlayerPosition.y]
76                 .allowMovement(this)) {
```

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

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

```
212             } catch (Exception e) {
213             }
214             PlayerData playerData = new PlayerData(player.getName(), 0, 0,↩
                    LIFE,
215                     LIFE, STRENGTH, boardObtainer.getPlayerPosition(), ↩
                            player
216                             .getSteps());
217             player = new Player(playerData);
218         }
219
220 }
```

### 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      public Player getLoadedPlayer();
22  }
```

### 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  }
```

### 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((game.getBoardDimension().y + 2)
78                                  * DungeonPanel.CELL_SIZE, (game
79                                  .getBoardDimension().x)
80                                  * DungeonPanel.CELL_SIZE - 7);
81                          drawDungeonPanel();
82                          drawDataPanel();
83                          dataPanel.refresh(game, dungeonPanel);
84                          dungeonPanel.updateUI();
85                      }
86                  } catch (Exception e1) {
87                      JOptionPane.showMessageDialog(null,
88                              "Level file is corrupt", "Error",
89                              JOptionPane.ERROR_MESSAGE);
90                  }
91              }
92          });
93
94          setRestartGameItemAction(new ActionListener() {
95              @Override
96              public void actionPerformed(ActionEvent e) {
97                  try {
98                      if (game == null) {
99                          JOptionPane.showMessageDialog(null,
100                                 "You are not playing a level.");
101                     } else {
102                         game.restart();
103                         dataPanel.setVisible(false);
104                         dungeonPanel.setVisible(false);
105                         remove(dataPanel);
106                         remove(dungeonPanel);
107                         drawDungeonPanel();
108                         drawDataPanel();
109                         dataPanel.refresh(game, dungeonPanel);
110                         dungeonPanel.updateUI();
111                     }
112                 } catch (CorruptedFileException e1) {
113                     JOptionPane.showMessageDialog(null, "The file is ↩
                            corrupt",
114                             "Error", JOptionPane.ERROR_MESSAGE);
115                 }
116             }
117         });
118
119         setSaveGameItemAction(new ActionListener() {
120
121             @Override
122             public void actionPerformed(ActionEvent e) {
123                 if (game != null) {
124                     File directory = new File("." + File.separator
125                             + "savedGames");
126                     if (!directory.exists()) {
127                         directory.mkdir();
128                     }
129                     try {
130                         new SaveGameOnFile(game);
131                     } catch (SavingCorruptedException e1) {
132                         JOptionPane.showMessageDialog(null,
```

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

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

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

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

### 1.2.6. DungeonPanel.java

```java
package front;

import static professorShipSrc.ImageUtils.drawString;
import static professorShipSrc.ImageUtils.loadImage;
import static professorShipSrc.ImageUtils.overlap;

import java.awt.Color;
import java.awt.Image;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.swing.JOptionPane;

import professorShipSrc.GamePanel;
import back.BloodyFloor;
import back.Bonus;
import back.Character;
import back.Floor;
import back.Game;
import back.Monster;
import back.MoveTypes;
import back.Point;
import back.Putable;
import back.Wall;

/**
 * @author tmehdi Class that extends the professor ship class GamePanel. This
 *         class is used for the Dungeon panel that is into the
 *         DungeonGameFrame.
 */
public class DungeonPanel extends GamePanel {

    private static final long serialVersionUID = 1L;
    static final int CELL_SIZE = 30;

    private Image playerImage;
    private Map<Class<? extends Putable>, Image> boardImagesByClass = new HashMap<Class<? extends Putable>, Image>();
    private Map<String, Image> monsterImagesByName = new HashMap<String, Image>();
    private Map<String, Image> bonusImagesByName = new HashMap<String, Image>();
    private Monster monsterUnderMouse = null;

    /**
     * @param game
     * @param dataPanel
     * @param dungeonListener
     *            Call the super constructor and draw the pane. The interface
     *            DungeonPanelListener that extends the professor ship interface
     *            GamePanelListener is used to make an implementation of the
     *            "onMouseMoved" method. It allows us to know in what cell is
     *            and make the different actions.
     */
    public DungeonPanel(Game game, DataPanel dataPanel,
            DungeonPanelListener dungeonListener) {
        super(game.getBoardDimension().x - 2, game.getBoardDimension().y - 2,
                CELL_SIZE, dungeonListener, Color.BLACK);
        playerImage();
        boardImagesByClass();
        monstersImagesInitialize();
        bonusImagesInitialize();
        drawDungeon(game);
        setVisible(true);
    }
```

```
66
67      /**
68       * @param monsterUnderMouse
69       *            Setter of the monster under mouse.
70       */
71      public void setMonsterUnderMouse(Monster monsterUnderMouse) {
72          this.monsterUnderMouse = monsterUnderMouse;
73      }
74
75      /**
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                    Putable cell = game.getBoard()[pPos.x + i][pPos.y + j↩
                           ];
210                    if (cell.getClass().equals(Monster.class)) {
211                        image = monsterImagesByName.get(((Monster) cell)
212                                .getMonsterType().toString());
213                        image = overlap(floorImage, image);
214                        image = drawString(image, ((Character) cell).↩
                               getLevel()
215                                .toString(), Color.WHITE);
216                        put(image, pPos.x + i - 1, pPos.y + j - 1);
217                    } else if (cell.getClass().equals(Bonus.class)) {
218                        image = bonusImagesByName.get(((Bonus) cell).↩
                               getBonusType()
219                                .toString());
220                        image = overlap(floorImage, image);
221                        image = drawString(image, (((Bonus) cell).↩
                               getBonusType()
222                                .getBonusAmount()).toString(), Color.RED);
223                        put(image, pPos.x + i - 1, pPos.y + j - 1);
224                    } else {
225                        image = boardImagesByClass.get(cell.getClass());
226                        if (cell.getClass().equals(Wall.class)) {
227                            put(image, pPos.x + i - 1, pPos.y + j - 1);
228                        } else if (cell.getClass().equals(BloodyFloor.↩
                               class)) {
229                            put(bloodyFloorImage, pPos.x + i - 1, pPos.y +↩
                                   j - 1);
230                        } else {
231                            put(floorImage, pPos.x + i - 1, pPos.y + j - ↩
                                   1);
232                        }
233                    }
234                }
235            }
236
237            Point p = new Point(game.getPlayer().getPosition());
238
239            if (game.getBoard()[p.x][p.y] instanceof BloodyFloor) {
240                image = overlap(bloodyFloorImage, playerImage);
241            }
242            image = overlap(floorImage, playerImage);
243            image = drawString(image, game.getPlayer().getLevel().toString↩
                   (),
244                    Color.WHITE);
245            put(image, p.x - 1, p.y - 1);
246        }
247
248        /**
249         * @return Getter of the monsterUnderMouse.
250         */
251        public Monster getMonsterUnderMouse() {
252            return monsterUnderMouse;
253        }
254
255        /**
256         * @param game
257         *            of class Game
258         * @param moveType
259         *            instance of enumerative MoveTypes
260         *
```

```
261         *            Redraw if necessary the DungeonPanel.
262         */
263        public void drawPlayerMove(Game game, MoveTypes moveType) {
264            Image bloodyFloor;
265            Image floor;
266            Point afterMove = new Point(game.getPlayer().getPosition().x, ↩
                   game
267                        .getPlayer().getPosition().y);
268            Point beforeMove = afterMove.sub(moveType.getDirection());
269            floor = boardImagesByClass.get(Floor.class);
270            bloodyFloor = boardImagesByClass.get(BloodyFloor.class);
271            bloodyFloor = overlap(floor, bloodyFloor);
272            clear(beforeMove.x - 1, beforeMove.y - 1);
273            if (game.getBoard()[beforeMove.x][beforeMove.y].getClass().↩
                   equals(
274                        BloodyFloor.class)) {
275                put(bloodyFloor, beforeMove.x - 1, beforeMove.y - 1);
276            } else {
277                put(floor, beforeMove.x - 1, beforeMove.y - 1);
278            }
279
280            clear(afterMove.x - 1, afterMove.y - 1);
281            Image image;
282            if (game.getBoard()[afterMove.x][afterMove.y].getClass().↩
                   equals(
283                        BloodyFloor.class)) {
284                image = overlap(bloodyFloor, playerImage);
285                image = drawString(image, game.getPlayer().getLevel().↩
                       toString(),
286                            Color.WHITE);
287                put(image, afterMove.x - 1, afterMove.y - 1);
288            } else {
289                image = overlap(floor, playerImage);
290                image = drawString(image, game.getPlayer().getLevel().↩
                       toString(),
291                            Color.WHITE);
292
293                put(image, afterMove.x - 1, afterMove.y - 1);
294            }
295            updateUI();
296        }
297
298        /**
299         * @param p
300         *
301         *            Draw blood on the floor where a character die.
302         */
303        public void drawDiedCharacter(Point p) {
304            Image imagFloor = boardImagesByClass.get(Floor.class);
305            Image imagBloodFloor = boardImagesByClass.get(BloodyFloor.↩
                   class);
306            clear(p.x - 1, p.y - 1);
307            put(overlap(imagFloor, imagBloodFloor), p.x - 1, p.y - 1);
308            repaint();
309
310        }
311
312        /**
313         * @param p
314         *
315         *            Remove the image of the bonus and draw a floor.
316         */
317        public void drawGrabedBonus(Point p) {
318            Image floor = boardImagesByClass.get(Floor.class);
319            clear(p.x - 1, p.y - 1);
320            put(overlap(floor, playerImage), p.x - 1, p.y - 1);
321            repaint();
322
323        }
324
325        public void drawDiscoveredCell(Game game, MoveTypes dir) {
326            Point pPos = game.getPlayer().getPosition();
327            List<Point> points = new ArrayList<Point>();
328            points.add(pPos.add(dir.getDirection()));
```

```
329             if (dir == MoveTypes.LEFT || dir == MoveTypes.RIGHT) {
330                 points.add(pPos.add(1, 0).add(dir.getDirection()));
331                 points.add(pPos.sub(1, 0).add(dir.getDirection()));
332             } else {
333                 points.add(pPos.add(0, 1).add(dir.getDirection()));
334                 points.add(pPos.sub(0, 1).add(dir.getDirection()));
335             }
336 
337             Image image;
338             Image floorImage = boardImagesByClass.get(Floor.class);
339             Image bloodyFloorImage = overlap(floorImage, ↩
                    boardImagesByClass
340                         .get(BloodyFloor.class));
341 
342             for (Point p : points) {
343                 if (p.x > 0 && p.x < game.getBoardDimension().x-1 && p.y >↩
                        0
344                         && p.y < game.getBoardDimension().y-1) {
345                     if (game.getBoard()[p.x][p.y].isVisible()) {
346                         game.getBoard()[p.x][p.y].setVisible();
347                         Putable cell = game.getBoard()[p.x][p.y];
348                         if (cell.getClass().equals(Monster.class)) {
349                             image = monsterImagesByName.get(((Monster) ↩
                                    cell)
350                                     .getMonsterType().toString());
351                             image = overlap(floorImage, image);
352                             image = drawString(image, ((Character) cell).↩
                                    getLevel()
353                                     .toString(), Color.WHITE);
354                             put(image, p.x - 1, p.y - 1);
355                         } else if (cell.getClass().equals(Bonus.class)) {
356                             image = bonusImagesByName.get(((Bonus) cell)
357                                     .getBonusType().toString());
358                             image = overlap(floorImage, image);
359                             image = drawString(image, (((Bonus) cell)
360                                     .getBonusType().getBonusAmount()).↩
                                            toString(),
361                                     Color.RED);
362                             put(image, p.x - 1, p.y - 1);
363                         } else {
364                             image = boardImagesByClass.get(cell.getClass()↩
                                    );
365                             if (cell.getClass().equals(Wall.class)) {
366                                 put(image, p.x - 1, p.y - 1);
367                             } else if (cell.getClass().equals(BloodyFloor.↩
                                    class)) {
368                                 put(bloodyFloorImage, p.x - 1, p.y - 1);
369                             } else {
370                                 put(floorImage, p.x - 1, p.y - 1);
371                             }
372                         }
373                     }
374                 }
375             }
376 
377         }
378 
379         /**
380          * Method to initialize player image.
381          */
382         private void playerImage() {
383             try {
384                 playerImage = loadImage("./resources/images/hero.png");
385             } catch (IOException e) {
386                 JOptionPane.showMessageDialog(null, "Unexpected Error", "↩
                        Error",
387                         JOptionPane.ERROR_MESSAGE);
388             }
389         }
390 
391         /**
392          * Method to initialize board images.
393          */
394         private void boardImagesByClass() {
```

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

```
462             }
463             updateUI();
464         }
465
466 }
```

### 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(
39                 frameToShowOn, "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

```java
package parser;

import back.Point;

public class BoardDimensionLine extends Lines {

    private static final int elemsCuantity = 2;
    private Point boardDimension;

    public BoardDimensionLine(String line) {
        super(elemsCuantity, line);
        lineProcess();
        boardDimension = new Point(getData(0), getData(1));
    }

    public Point getBoardDimension() {
        return boardDimension;
    }

}
```

### 1.3.2. BoardLine.java

```java
package parser;

import back.Point;

public class BoardLine extends Lines {

    private static final int elemsCuantity = 6;
    private Point boardDimension;

    public BoardLine(String line, Point boardDimension) {
        super(elemsCuantity, line);
        this.boardDimension = boardDimension;
        lineProcess();
        lineCheck();
    }

    /**
     * This methods Checks which type of cell the parsed line is, and sets the
     * cell into the board.
     */

    @Override
    protected void lineCheck() {
        switch (data[0]) {

        case 1:
            // Player
            if (data[1] < 0 || data[1] >= boardDimension.x - 2 || data[2] < 0
                    || data[2] >= boardDimension.y - 2 || data[3] != 0
                    || data[4] != 0 || data[5] != 0) {
                throw new CorruptedFileException();
            }
            break;

        case 2:
            // Wall
            if (data[1] < 0 || data[1] >= boardDimension.x - 2 || data[2] < 0
                    || data[2] >= boardDimension.y - 2 || data[4] != 0) {
                throw new CorruptedFileException();
            }
            break;

```

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

```
60                          if (playerFlag == true) {
61                              throw new CorruptedFileException();
62                          }
63                          parsePlayer(line);
64                          playerFlag = true;
65                      } else {
66                          BoardLine cell = new BoardLine(line, ↩
                                boardDimension);
67                          Point point = (new Point(cell.getData(1), cell
68                                  .getData(2))).add(new Point(1, 1));
69
70                          if (cell.isWallLine()) {
71                              parseWall(point, cell);
72                          } else if (cell.isMonsterLine()) {
73                              parseMonster(point, cell);
74                          } else if (cell.isBonusLine()) {
75                              parseBonus(point, cell);
76                          }
77                      }
78                  }
79              }
80          }
81
82          if (!nameFlag || !playerFlag || !dimensionFlag) {
83              throw new CorruptedFileException();
84          }
85          validation();
86      }
87
88      public void validation() {
89          protectionWalls();
90          putFloor();
91          if (!(board[getPlayerPosition().x][getPlayerPosition().y] ↩
                instanceof Floor)) {
92              throw new CorruptedFileException();
93          }
94      }
95
96      public void parseBonus(Point point, BoardLine cell) {
97          putCell(point.x, point.y, new Bonus(point, cell.getData(0), ↩
                cell
98                  .getData(5)));
99      }
100
101     public void parsePlayer(String line) {
102         BoardLine cell = new BoardLine(line, boardDimension);
103         Point point = (new Point(cell.getData(1), cell.getData(2)))
104                 .add(new Point(1, 1));
105         playerPosition = point;
106     }
107
108     public void parseMonster(Point point, BoardLine cell) {
109         putCell(point.x, point.y, new Monster(point, cell.getData(3), ↩
                cell
110                 .getData(4)));
111     }
112
113     public void parseWall(Point point, BoardLine cell) {
114         putCell(point.x, point.y, new Wall());
115     }
116
117     public void parseBoardName(String line) {
118         BoardNameLine boardNameLine = new BoardNameLine(line);
119         this.boardName = boardNameLine.getName();
120     }
121
122     public void parseDimension(String line) {
123         BoardDimensionLine boardDimensionLine = new BoardDimensionLine↩
                (line);
124         boardDimension = boardDimensionLine.getBoardDimension().add(
125                 new Point(2, 2));
126         board = new Putable[boardDimension.x][boardDimension.y];
127
128     }
```

```
129
130     public void putFloor() {
131         for (int i = 1; i < boardDimension.x - 1; i++) {
132             for (int j = 1; j < boardDimension.y - 1; j++) {
133                 if (getBoardElem(i, j) == null) {
134                     putCell(i, j, new Floor());
135                 }
136             }
137         }
138     }
139
140     public void protectionWalls() {
141         for (int i = 0; i < boardDimension.y; i++) {
142             Wall aux = new Wall();
143             aux.setVisible();
144             putCell(0, i, aux);
145             Wall aux1 = new Wall();
146             aux1.setVisible();
147             putCell(boardDimension.x - 1, i, aux1);
148         }
149         for (int i = 0; i < boardDimension.x; i++) {
150             Wall aux = new Wall();
151             aux.setVisible();
152             putCell(i, 0, aux);
153             Wall aux1 = new Wall();
154             aux1.setVisible();
155             putCell(i, boardDimension.y - 1, aux1);
156         }
157
158     }
159
160     public Point getBoardDimension() {
161         return boardDimension;
162     }
163
164     public String getBoardName() {
165         return boardName;
166     }
167
168     public Point getPlayerPosition() {
169         return playerPosition;
170     }
171
172     public Putable[][][] getBoard() {
173         return board;
174     }
175
176     public int getBoardRows() {
177         return boardDimension.x;
178     }
179
180     public int getBoardColums() {
181         return boardDimension.y;
182     }
183
184     public Putable getBoardElem(Point position) {
185         return board[position.x][position.y];
186     }
187
188     public Putable getBoardElem(int x, int y) {
189         return board[x][y];
190     }
191
192     public void putCell(int i, int j, Putable cell) {
193         putCell(new Point(i, j), cell);
194     }
195
196     public void putCell(Point p, Putable cell) {
197         board[p.x][p.y] = cell;
198     }
199
200     @Override
201     public File getFile() {
202         return inputFile;
```

```
203     }
204
205     @Override
206     public int getPlayerSteps() {
207         return 0;
208     }
209
210 }
```

### 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;
```

```java
import java.io.InputStream;

import javax.imageio.ImageIO;

/**
 * Clase con métodos útiles para el manejo de imágenes.
 */
public class ImageUtils {

    /**
     * Carga una imagen y retorna una instancia de la misma. Si hay algun problema al leer el archivo lanza una
     * excepcion.
     */
    public static Image loadImage(String fileName) throws IOException {
        InputStream stream = ClassLoader.getSystemResourceAsStream(fileName);
        if (stream == null) {
            return ImageIO.read(new File(fileName));
        } else {
            return ImageIO.read(stream);
        }
    }

    /**
     * Dibuja un texto en el vértice inferior derecho de la imagen, con el color indicado. Retorna una imagen nueva con
     * los cambios, la imagen original no se modifica.
     */
    public static Image drawString(Image img, String text, Color color) {
        BufferedImage result = new BufferedImage(img.getWidth(null), img.getHeight(null), BufferedImage.TYPE_INT_ARGB);
        Graphics2D g = (Graphics2D) result.getGraphics();
        g.drawImage(img, 0, 0, null);

        Font font = new Font(Font.SANS_SERIF, Font.BOLD, 12);
        g.setFont(font);
        g.setColor(color);
        Rectangle2D r = font.getStringBounds(text, g.getFontRenderContext());
        g.drawString(text, img.getWidth(null) - (int) r.getWidth() - 2, img.getHeight(null) - 2);
        return result;
    }

    /**
     * Superpone dos imágenes. Retorna una nueva imagen con las 2 imágenes recibidas superpuestas. Las
     * originales no se modifican.
     */
    public static Image overlap(Image image1, Image image2) {
        BufferedImage result = new BufferedImage(image1.getWidth(null), image1.getHeight(null),
                BufferedImage.TYPE_INT_ARGB);
        Graphics2D g = (Graphics2D) result.getGraphics();
        g.drawImage(image1, 0, 0, null);
        g.drawImage(image2, 0, 0, null);
        return result;
    }
}
```

## 1.5. saveLoadImplementation

### 1.5.1. Criteria.java

```java
package saveLoadImplementation;
```

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

### 1.5.2. FilterArrayFileList.java

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

### 1.5.3. FilterFileList.java

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

### 1.5.4. LoadGameFromFile.java

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

```
71      };
72
73      @Override
74      public void parseBonus(Point point, BoardLine cell) {
75          super.parseBonus(point, cell);
76          setBoardCellVisivility(point, cell.getData(4));
77      }
78
79      @Override
80      public void parseMonster(Point point, BoardLine cell) {
81          putCell(point.x, point.y, new Monster(point, cell.getData(3), ↩
                cell
82                  .getData(4), Math.abs(cell.getData(5))));
83          if (cell.getData(5) < 0) {
84              setBoardCellVisivility(point, 0);
85          } else if (cell.getData(5) > 0) {
86              setBoardCellVisivility(point, 1);
87          }
88      }
89
90      @Override
91      public Point getPlayerPosition() {
92          return playerLoadedPosition;
93      }
94
95      @Override
96      public Integer getPlayerLoadedHealth() {
97          return playerLoadedHealth;
98      }
99
100     @Override
101     public Integer getPlayerLoadedMaxHealth() {
102         return playerLoadedMaxHealth;
103     }
104
105     @Override
106     public Integer getPlayerLoadedExperience() {
107         return playerLoadedExperience;
108     }
109
110     @Override
111     public Integer getPlayerLoadedStrength() {
112         return playerLoadedStrength;
113     }
114
115     @Override
116     public Integer getPlayerLoadedSteps() {
117         return playerLoadedSteps;
118     }
119
120     public T getGame(Class<T> gameImpClass, GameListener listener) {
121         T game;
122         try {
123             game = gameImpClass.getConstructor(BoardObtainer.class,
124                     GameListener.class).newInstance(this, listener);
125         } catch (Exception e) {
126             e.printStackTrace();
127             throw new CorruptedFileException();
128         }
129         return game;
130     }
131
132     @Override
133     public int getPlayerLoadedLevel() {
134         return loadedLevel;
135     }
136
137     @Override
138     public String getPlayerName() {
139         return playerName;
140     }
141
142     @Override
143     public Player getLoadedPlayer() {
```

```
144         PlayerData playerData = new PlayerData(playerName, loadedLevel↩
                ,
145                 playerLoadedExperience, playerLoadedHealth,
146                 playerLoadedMaxHealth, playerLoadedStrength,
147                 playerLoadedPosition, playerLoadedSteps);
148         return new Player(playerData);
149     }
150
151 }
```

### 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

```java
package saveLoadImplementation;

public class SavingCorruptedException extends RuntimeException {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

## 1.6. tests

### 1.6.1. GameTests.java

```java
package tests;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import java.io.File;

import javax.swing.JOptionPane;

import org.junit.Before;
import org.junit.Test;

import parser.BoardParserFromFile;
import saveLoadImplementation.FilterArrayFileList;
import saveLoadImplementation.FilterFileList;
import saveLoadImplementation.LoadGameFromFile;
import saveLoadImplementation.SaveGameOnFile;
import back.BloodyFloor;
import back.Bonus;
import back.DungeonGameImp;
import back.DungeonGameListener;
import back.LoadGame;
import back.Monster;
import back.MoveTypes;
import back.Point;

public class GameTests {

    private DungeonGameImp game;

    @Before
    public void setup() {
        game = new DungeonGameImp(new BoardParserFromFile(new File(
                "./testBoard/boardForTest1.board")),new ↩
                    DungeonGameListener() {

            @Override
            public String playerNameRequest() {
                return "Tom";
            }

            @Override
            public void executeWhenPlayerMoves(MoveTypes moveType) {
            }

            @Override
            public void executeWhenGameWinned() {
            }

            @Override
            public void executeWhenGameLoosed() {
            }
```

```
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 }
```