Programación orientada a objetos
Códigos fuente TPE
Dungeon Game

6 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      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     @SuppressWarnings("unchecked")
25     public DungeonGameImp(BoardObtainer boardObtainer, GameListener ↩
           gameListener) {
26         this.boardObtainer = boardObtainer;
27         this.gameListener = gameListener;
28         boardName = boardObtainer.getBoardName();
29         boardDimension = boardObtainer.getBoardDimension();
30         board = boardObtainer.getBoard();
31         PlayerData playerData = new PlayerData(null, 0, 0, LIFE, LIFE,
32                 STRENGTH, boardObtainer.getPlayerPosition(),
33                 boardObtainer.getPlayerSteps());
34         if (!(boardObtainer instanceof LoadGame)) {
35             playerData.setName(gameListener.playerNameRequest());
36             player = new Player(playerData);
37         } else {
38             playerData
39                     .setName(((LoadGame<Game>) boardObtainer).↩
                           getPlayerName());
40             playerData.setHealth(((LoadGame<Game>) boardObtainer)
41                     .getPlayerLoadedHealth());
42             playerData.setMaxHealth(((LoadGame<Game>) boardObtainer)
43                     .getPlayerLoadedMaxHealth());
44             playerData.setStrength(((LoadGame<Game>) boardObtainer)
45                     .getPlayerLoadedStrength());
46             playerData.setExperience(((LoadGame<Game>) boardObtainer)
47                     .getPlayerLoadedExperience());
48             player = new Player(playerData,
49                     ((LoadGame<Game>) boardObtainer).↩
                           getPlayerLoadedLevel(),
50                     ((LoadGame<Game>) boardObtainer).↩
                           getPlayerLoadedSteps());
51         }
52         firstDiscoveredCells();
53     }
54
55     private void firstDiscoveredCells() {
56         Point p = player.getPosition();
57
58         board[p.x][p.y].setVisible();
59
60         board[p.x + 1][p.y - 1].setVisible();
61         board[p.x + 1][p.y].setVisible();
62         board[p.x + 1][p.y + 1].setVisible();
63
64         board[p.x][p.y - 1].setVisible();
65         board[p.x][p.y].setVisible();
66         board[p.x][p.y + 1].setVisible();
67
68         board[p.x - 1][p.y - 1].setVisible();
69         board[p.x - 1][p.y].setVisible();
70         board[p.x - 1][p.y + 1].setVisible();
71     }
72
73     /**
74      * @see back.Game#receiveMoveStroke(back.MoveTypes) Is't allow the↩
            game to
```

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

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

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

1.1.9. DungeonGameListener.java

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

1.1.10. Floor.java

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

1.1.11. Game.java

```
1   package back;
2
3   public interface Game {
4
5       public void 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      void executeWhenLevelUp();
20
21  }
```

1.1.13. GrabBonus.java

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

1.1.14. LoadGame.java

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

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

1.1.15. Monster.java

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

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

1.1.19. PlayerData.java

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

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

1.1.20. Point.java

```java
package back;

public class Point {
    public int x;
    public int y;

    public Point(Point p) {
        this(p.x, p.y);
    }

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public Point add(Point p) {
        return new Point(this.x + p.x, this.y + p.y);
    }

    @Override
    public String toString() {
        return "[ " + x + ", " + y + " ]";
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + x;
        result = prime * result + y;
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Point other = (Point) obj;
        if (x != other.x)
            return false;
        if (y != other.y)
            return false;
        return true;
    }

    public Point sub(Point p) {
        return new Point(this.x - p.x, this.y - p.y);
    }

    public Point add(int i, int j) {
        return add(new Point(i, j));
    }

    public Point sub(int i, int j) {
        return sub(new Point(i, j));
    }
}
```

1.1.21. Putable.java

```java
package back;

public interface Putable {

    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

```java
package front;

import javax.swing.JFrame;

public class App {
    public static void main(String[] args) {
        GameFrame dungeonGameFrame = new DungeonGameFrame();
        dungeonGameFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        dungeonGameFrame.setVisible(true);
    }
}
```

1.2.2. DataPanel.java

```java
package front;

import java.awt.Color;
import java.awt.Font;
import java.util.HashMap;
import java.util.Map;

import javax.swing.BoxLayout;
import javax.swing.JLabel;
import javax.swing.JPanel;

import back.Game;
import back.Monster;
import back.Player;
import back.Point;
import back.Putable;

/**
 * @author tmehdi Class that extends the class J|Panel. This class is used for
 *         the Dungeon panel that is into the DungeonGameFrame.
 *
 */
public class DataPanel extends JPanel {

    private static final long serialVersionUID = 1L;

    @SuppressWarnings("unused")
    private JLabel[] playerLabel;
    private Map<Monster, JLabel[]> monstersLabels = new HashMap<Monster, JLabel[]>();

    public DataPanel(Player player, Color color) {
        setBackground(Color.WHITE);
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        addCharacter(player);
    }

    public void addCharacter(Player character) {
        JLabel[] playerLabel = new JLabel[6];
        playerLabel[0] = new JLabel("  " + character.getName());
        playerLabel[0].setFont(new Font("Serif", Font.BOLD, 16));
        playerLabel[0].setForeground(Color.BLUE);
        playerLabel[1] = new JLabel("  " + "Health: " + character.getHealth()
                + "/" + character.getMaxHealth());
        playerLabel[2] = new JLabel("  " + "Strength: "
                + character.getStrength());
        playerLabel[3] = new JLabel("  " + "Level: " + character.getLevel());
        playerLabel[4] = new JLabel("  " + "Experience: "
                + character.getExperience() + "/"
                + 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                          drawDungeonPanel();
78                          drawDataPanel();
79                          dataPanel.refresh(game, dungeonPanel);
80                          dungeonPanel.updateUI();
81                      }
82                  } catch (Exception e1) {
83                      e1.printStackTrace();
84                      JOptionPane.showMessageDialog(null,
85                              "Level file is corrupt", "Error",
86                              JOptionPane.ERROR_MESSAGE);
87                  }
88              }
89          });
90
91          setRestartGameItemAction(new ActionListener() {
92              @Override
93              public void actionPerformed(ActionEvent e) {
94                  try {
95                      if (game == null) {
96                          JOptionPane.showMessageDialog(null,
97                                  "You are not playing a level.");
98                      } else {
99                          game.restart();
100                         dataPanel.setVisible(false);
101                         dungeonPanel.setVisible(false);
102                         remove(dataPanel);
103                         remove(dungeonPanel);
104                         drawDungeonPanel();
105                         drawDataPanel();
106                         dataPanel.refresh(game, dungeonPanel);
107                         dungeonPanel.updateUI();
108                     }
109                 } catch (CorruptedFileException e1) {
```

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

```
179                 File file;
180                 JFileChooser fc = new JFileChooser();
181                 fc.setCurrentDirectory(new File("./savedGames"));
182                 fc.showOpenDialog(DungeonGameFrame.this);
183                 file = fc.getSelectedFile();
184                 if (file == null) {
185                     JOptionPane.showMessageDialog(null,
186                             "You didn't select any file.");
187                 } else {
188                     try {
189                         LoadGame<DungeonGameImp> loadGame = new ↩
                                LoadGameFromFile<DungeonGameImp>(
190                                     file);
191                         game = loadGame.getGame(DungeonGameImp.class,
192                                 new DungeonGameListenerImp());
193                         drawDungeonPanel();
194                         drawDataPanel();
195                         dataPanel.updateUI();
196                         dungeonPanel.updateUI();
197                     } catch (CorruptedFileException e2) {
198                         e2.printStackTrace();
199                         JOptionPane
200                                 .showMessageDialog(
201                                         null,
202                                         "Files loading error occours. ↩
                                            Try again later.",
203                                         "Error", JOptionPane.↩
                                            ERROR_MESSAGE);
204                     }
205                 }
206             }
207         });
208
209         setExitGameItemAction(new ActionListener() {
210             @Override
211             public void actionPerformed(ActionEvent e) {
212                 try {
213                     DungeonGameFrame.this.setVisible(false);
214                     DungeonGameFrame.this.dispose();
215                 } catch (Throwable e1) {
216                     JOptionPane.showMessageDialog(null, "Exit fault", ↩
                            "Error",
217                             JOptionPane.ERROR_MESSAGE);
218                 }
219             }
220         });
221
222     }
223
224     /**
225      * Method to make appear the data panel.
226      */
227     private void drawDataPanel() {
228         dataPanel = new DataPanel(game.getPlayer(), Color.GRAY);
229         add(dataPanel, BorderLayout.EAST);
230     }
231
232     /**
233      * Method to make appear the dungeon panel.
234      */
235     private void drawDungeonPanel() {
236         dungeonPanel = new DungeonPanel(game, dataPanel,
237                 new DungeonPanelListenerImp());
238         add(dungeonPanel, BorderLayout.CENTER);
239     }
240
241     /**
242      * Getter of the dungeon panel.
243      *
244      * @return DungeonPanel
245      */
246     public DungeonPanel getDungeonPanel() {
247         return dungeonPanel;
248     }
```

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

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

```
388                 }
389                 dataPanel.refresh(game, dungeonPanel);
390                 dataPanel.updateUI();
391 
392             }
393 
394         }
395 }
```

1.2.6. DungeonPanel.java

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

```
52          *            "onMouseMoved" method. It allows us to know in what cell is
53          *            and make the different actions.
54          */
55         public DungeonPanel(Game game, DataPanel dataPanel,
               DungeonPanelListener dungeonListener) {
56             super(game.getBoardDimension().x - 2,
57                     game.getBoardDimension().y - 2, CELL_SIZE,
                           dungeonListener
58                     , Color.BLACK);
59             playerImage();
60             boardImagesByClass();
61             monstersImagesInitialize();
62             bonusImagesInitialize();
63             drawDungeon(game);
64             setVisible(true);
65         }
66
67         /**
68          * @param monsterUnderMouse
69          *            Setter of the monster under mouse.
70          */
71         public void setMonsterUnderMouse(Monster monsterUnderMouse) {
72             this.monsterUnderMouse = monsterUnderMouse;
73         }
74
75         /**
76          * @param dungeonGameFrame
77          *            Draw the full dungeon panel.
78          */
79         public void dwarFullDungeon(DungeonGameFrame dungeonGameFrame) {
80             Image image;
81             Image floorImage = boardImagesByClass.get(Floor.class);
82             Image bloodyFloorImage = overlap(floorImage,
83                     boardImagesByClass.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,
102                                (((Bonus) cell).getBonusType().
                                       getBonusAmount())
103                                        .toString(), Color.RED);
104                        put(image, i - 1, j - 1);
105                    } else {
106                        image = boardImagesByClass.get(cell.getClass());
107                        if (cell.getClass().equals(Wall.class)) {
108                            put(image, i - 1, j - 1);
109                        } else if (cell.getClass().equals(BloodyFloor.
                               class)) {
110                            put(bloodyFloorImage, i - 1, j - 1);
111                        } else {
112                            put(floorImage, i - 1, j - 1);
113                        }
114                    }
115                }
116            }
117
```

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

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

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

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

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

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

1.2.7. DungeonPanelListener.java

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

1.2.8. GameFrame.java

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

1.3. parser

1.3.1. BoardDimensionLine.java

```
1  package parser;
2
3  import back.Point;
4
5  public class BoardDimensionLine extends Lines {
6
7      private static final int elemsCuantity = 2;
8      private Point boardDimension;
9
10     public BoardDimensionLine(String line) {
```

```
11          super(elemsCuantity, line);
12          lineProcess();
13          boardDimension = new Point(getData(0), getData(1));
14      }
15
16      public Point getBoardDimension() {
17          return boardDimension;
18      }
19
20  }
```

1.3.2. BoardLine.java

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

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

1.3.3. BoardNameLine.java

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

1.3.4. BoardParserFromFile.java

```
1  package parser;
2
3  import java.io.BufferedReader;
4  import java.io.File;
5  import java.io.FileReader;
6  import java.io.IOException;
7
8  import back.BoardObtainer;
9  import back.Bonus;
10 import back.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
```

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

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

```
210
211 }
```

1.3.5. CorruptedFileException.java

```
1  package parser;
2
3  public class CorruptedFileException extends RuntimeException {
4
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

```java
package professorShipSrc;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;

import javax.swing.JPanel;

/**
 * Panel que representa una grilla de imÃ¡genes, siendo posible agregarle y quitarle imÃ¡genes. Asimismo, cuenta con una
 * interfaz que permite a quien la utilice ser notificada cuando el usuario posiciona el mouse sobre una celda de la grilla.
 */
public class GamePanel extends JPanel {

    private int rows, columns;
    private int cellSize;
    private Color color;
    private Image[][] images;

    /**
     * Crea un nuevo panel con las dimensiones indicadas.
     *
     * @param rows Cantidad de filas.
     * @param columns Cantidad de columnas.
     * @param cellSize Ancho y alto de cada imagen en pÃxeles.
     * @param listener Listener que serÃ¡ notificado cuando el usuario se posicione sobre una celda de la grilla.
     * @param color Color de fondo del panel.
     */
    public GamePanel(final int rows, final int columns, final int cellSize, final GamePanelListener listener, Color color) {
        setSize(columns * cellSize, rows * cellSize);
        images = new Image[rows][columns];
        this.rows = rows;
        this.columns = columns;
        this.cellSize = cellSize;
        this.color = color;

        addMouseMotionListener(new MouseMotionAdapter() {

            private Integer currentRow;
            private Integer currentColumn;

            @Override
            public void mouseMoved(MouseEvent e) {
                int row = e.getY() / cellSize;
                int column = e.getX() / cellSize;
                if (row >= rows || column >= columns || row < 0 || column < 0) {
                    return;
                }

                if (!nullSafeEquals(currentRow, row) || !nullSafeEquals(currentColumn, column)) {
                    currentRow = row;
                    currentColumn = column;
                    listener.onMouseMoved(row, column);
                }
            }

            private boolean nullSafeEquals(Object o1, Object o2) {
                return o1 == null ? o2 == null : o1.equals(o2);
            }
        });
    }

    /**
     * Ubica una imagen en la fila y columna indicadas.
     */
```

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

1.4.2. GamePanelListener.java

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

1.4.3. ImageUtils.java

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

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

1.5. saveLoadImplementation

1.5.1. Criteria.java

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

1.5.2. FilterArrayFileList.java

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

1.5.3. FilterFileList.java

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

```
76
77      @Override
78      public void parseMonster(Point point, BoardLine cell) {
79          putCell(point.x,
80                  point.y,
81                  new Monster(point, cell.getData(3), cell.getData(4), ↩
                        Math
82                                .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 }
```

1.5.5. SaveGameOnFile.java

```java
package saveLoadImplementation;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

import back.BloodyFloor;
import back.Bonus;
import back.Floor;
import back.Game;
import back.Monster;
import back.SaveGame;
import back.Wall;

/**
 * @author tomas SaveGame implementation that save on a file.
 */
public class SaveGameOnFile implements SaveGame {

    private Game gameToSave;
    private File placeToSave;

    public SaveGameOnFile(Game gameToSave) {
        this.gameToSave = gameToSave;
        File file = new File("./savedGames");
        FilterFileList filterFileList = new FilterArrayFileList(file);
        filterFileList = filterFileList.filter("savedGame");
        int number = filterFileList.size();
        if (number > 0) {
            placeToSave = new File("./savedGames/savedGame" + "(" + ↩
                number
                    + ")");
        } else {
            placeToSave = new File("./savedGames/savedGame");
        }
        try {
            save();
        } catch (IOException e) {
            throw new SavingCorruptedException();
        }
    }

    public SaveGameOnFile(Game gameToSave, File placeToSave) {
        this.gameToSave = gameToSave;
        this.placeToSave = placeToSave;
        FilterFileList filterFileList = new FilterArrayFileList(
                placeToSave.getParentFile());
        filterFileList = filterFileList.filter(placeToSave.getName());
        int number = filterFileList.size();
        if (number > 0) {
            this.placeToSave = new File(placeToSave.getPath() + "(" + ↩
                number
                    + ")");
        } else {
            this.placeToSave = new File(placeToSave.getPath());
        }
        try {
            save();
        } catch (IOException e) {
            throw new SavingCorruptedException();
        }
    }

    /**
     * The format of the file saved is: board dimension (10,11) board ↩
        name
     * ("Board name") player (1,row pos, col pos,exp,health,max health↩
        ,
     * strength, steps, level, name) walls (2,row pos, col pos, 0 ,0, ↩
        [0 is
     * visible 1 not visible]) bloodyFloor(2,row pos, col pos, 2 ,0, ↩
        [0 is
```

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

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

1.5.6. SavingCorruptedException.java

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

1.6. tests

1.6.1. GameTests.java

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

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

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

```
206     }
207
208 }
```

1.6.2. PlayerTests.java

```
1  package tests;
2
3  import static org.junit.Assert.assertEquals;
4
5  import java.io.File;
6
7  import org.junit.Before;
8  import org.junit.Test;
9
10 import parser.BoardParserFromFile;
11 import back.BoardObtainer;
12 import back.Bonus;
13 import back.Monster;
14 import back.MoveTypes;
15 import back.Player;
16 import back.PlayerData;
17 import back.Point;
18
19 public class PlayerTests {
20     BoardObtainer boardParser;
21     Player player;
22
23     @Before
24     public void setup() {
25         boardParser = new BoardParserFromFile(new File(
26                 "./testBoard/boardForTest1"));
27         player = new Player(new PlayerData("Tomas", 0, 0, 10, 10, 5,
28                 boardParser.getPlayerPosition(),0));
29     }
30
31     @Test
32     public void goodFunctionamientPlayerMovementTest() {
33         assertEquals(new Point(4, 4), player.getPosition());
34         player.move(MoveTypes.UP);
35         assertEquals(new Point(3, 4), player.getPosition());
36         player.move(MoveTypes.LEFT);
37         assertEquals(new Point(3, 3), player.getPosition());
38         player.move(MoveTypes.DOWN);
39         assertEquals(new Point(4, 3), player.getPosition());
40         player.move(MoveTypes.RIGHT);
41         assertEquals(new Point(4, 4), player.getPosition());
42     }
43
44     @Test
45     public void goodFunctionamientPlayerVsMonsterFightTest() {
46         Monster monster = ((Monster) boardParser.getBoard()[5][7]);
47         player.fightAnotherCharacter(monster);
48         assertEquals(
49                 new Integer(player.getMaxHealth() - monster.↩
                        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"));
27      }
28
29      @Test(expected = CorruptedFileException.class)
30      public void startPlayerPositionOverAMonsterTest() {
31          new BoardParserFromFile(new File("./testBoard/boardForTest2"))↩
                ;
32      }
33
34      @Test(expected = CorruptedFileException.class)
35      public void startPlayerPositionOverAWallTest() {
36          new BoardParserFromFile(new File("./testBoard/boardForTest3"))↩
                ;
37      }
38
39      @Test
40      public void mapWithoutSurroundingWalls() {
41          BoardObtainer boardParser = new BoardParserFromFile(new File(
42                  "./testBoard/boardForTest4"));
43          assertEquals(Wall.class, boardParser.getBoardElem(new Point(0,↩
                  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"))↩
                ;
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 }
```

front
parser
professorShipSrc
saveLoadImplementation
tests