Programación orientada a objetos
Códigos fuente TPE
Dungeon Game

7 de junio de 2011

Autores:

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

Índice

1. Codigos fuente

1.1. back

1.1.1. Algoritms.java

```java
package back;

/**
 * @author tomas
 *  Interface that represents the function/algorithm of monsters life ↩
    and strength.
 */
public interface Algoritms {
    public Integer lifeAlgoritm(int level);

    public Integer strengthAlgoritm(int level);

}
```

1.1.2. BloodyFloor.java

```java
package back;

public class BloodyFloor extends Floor{
@Override
public String toString() {
    return "Blood";
}
}
```

1.1.3. BoardObtainer.java

```java
package back;

import java.io.File;

public interface BoardObtainer {

    public void obtainBoard() throws Exception;

    public Point getBoardDimension();

    public Putable[][] getBoard();

    public Point getPlayerPosition();

    public String getBoardName();

    public Putable getBoardElem(Point point);

    public int getBoardRows();

    public int getBoardColums();

    public File getFile();

    public PlayerData getPlayerData();

}
```

1.1.4. Bonus.java

```java
package back;

public class Bonus extends Cell implements Putable {

    private BonusTypes bonusType;

    public Bonus(Point position, int numberBonusType, int bonusAmount)
         {
        bonusType = BonusTypes.getBonusType(numberBonusType);
        bonusType.setBonusAmount(bonusAmount);
    }

    public void giveBonus(Character character) {
        bonusType.giveBonus(character);
    }

    @Override
    public boolean allowMovement(DungeonGameImp game) {
        return true;
    }

    public void standOver(DungeonGameImp game) {
        giveBonus(game.getPlayer());
        Point point = new Point(game.getPlayer().getPosition().x, game
                .getPlayer().getPosition().y);

        Floor f = new Floor();
        f.setVisible();
        game.getBoard()[point.x][point.y] = f;

        game.getGameListener().executeWhenBonusGrabed(
                new Point(point.x, point.y));
    }

    public BonusTypes getBonusType() {
        return bonusType;
    }

    public int getAmountBonus() {
        return bonusType.getBonusAmount();
    }

    @Override
    public String toString() {
        return "Bonus";
    }

}
```

1.1.5. BonusTypes.java

```java
package back;

/**
 * @author tomas
 *  A beautiful enumerate for the different types of Bonuses.
 */
public enum BonusTypes {

    LIFE("Life", 0, new GrabBonus(){

        @Override
        public void grabBonus(Character character, Integer bonusAmount) {
            character.winLife(bonusAmount);
        }

    }), STRENGTH("Strength", 0, new GrabBonus(){

        @Override
        public void grabBonus(Character character, Integer bonusAmount) {
            character.grabStrengthBonus(bonusAmount);
        }

    });

    private String name;
    private Integer bonusAmount;
    private GrabBonus bonusGrabbed;

    private BonusTypes(String name, Integer bonusAmount, GrabBonus bonusGrabbed) {
        this.name = name;
        this.bonusAmount = bonusAmount;
        this.bonusGrabbed = bonusGrabbed;
    }

    public Integer getBonusAmount(){
        return bonusAmount;
    }

    public void setBonusAmount(Integer bonusAmount){
        this.bonusAmount = bonusAmount;
    }

    public String getName() {
        return name;
    }

    public static BonusTypes getBonusType(int data) {
        switch (data) {
        case (4):
            return BonusTypes.LIFE;
        case (5):
            return BonusTypes.STRENGTH;
        default:
            return null;
        }
    }

    public void giveBonus(Character character) {
        bonusGrabbed.grabBonus(character,getBonusAmount());
    }
}
```

1.1.6. Cell.java

```
package back;

/**
 * @author tomas
 *  Abstract class inserted on the hierarchy to make every class that can be on the board
 *  to be visible or invisible. Particular feature of this game.
 */
public abstract class Cell {

```

```java
    private boolean isVisible = false;

    public boolean isVisible() {
        return isVisible;
    }

    public void setVisible() {
        this.isVisible = true;
    }

    public void setNotVisible() {
        this.isVisible = false;
    }

}
```

1.1.7. Character.java

```java
package back;

/**
 * @author tomas Abstract class that extends cell. So it can ve ↩
     visible or
 *         invisible in the board.
 */
public abstract class Character extends Cell {

    private String name;
    private Integer level;
    private Integer maxHealth;
    private Integer health;
    private Integer strength;
    private Point position;

    public Character(String name, Integer level, Point position) {
        this.name = name;
        this.level = level;
        this.position = position;
    }

    public void winFight(Character character) {
    }

    public void fightAnotherCharacter(Character character) {
        this.hited(character.getStrength());
        if (!this.isDead()) {
            character.hited(this.getStrength());
            if (character.isDead()) {
                this.winFight(character);
            }
        } else {
            character.winFight(this);
        }

    }

    public void hited(Integer strength) {
        health -= strength;
    }

    public String getName() {
        return name;
    }

    public boolean isDead() {
        return health <= 0;
    }

    public Integer getLevel() {
        return level;
    }

    public void increaseLevel() {
        this.level += 1;
    }

    public Integer getMaxHealth() {
        return maxHealth;
    }

    public Integer getHealth() {
        return health;
    }

    public Integer getStrength() {
        return strength;
    }

    public Point getPosition() {
        return position;
    }

    @Override
    public String toString() {
        String resp;
        resp = "Name=" + getName();
        resp += "Level=" + getLevel();
        resp += "MaxHealth=" + getMaxHealth();
        resp += "Health=" + getHealth();
        resp += "Strength=" + getStrength();
        resp += "Position=" + getPosition();
        return resp;
    }

    public void winLife(Integer bonusAmount) {
        if (health + bonusAmount > maxHealth) {
            health = maxHealth;
        } else {
            health += bonusAmount;
        }
    }

    public void grabStrengthBonus(Integer bonusAmount) {
        strength += bonusAmount;
    }

    /**
     * Method just for tests
     *
     * @param position
     */
    public void setPosition(Point position) {
        this.position = position;
    }

    public void setMaxHealth(int maxHealth) {
        this.maxHealth = maxHealth;
    }

    public void setStrength(int strength) {
        this.strength = strength;
    }

    public void setHealth(Integer health) {
        this.health = health;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((health == null) ? 0 : health.↩
            hashCode());
        result = prime * result + ((level == null) ? 0 : level.
            hashCode());
        result = prime * result
                + ((maxHealth == null) ? 0 : maxHealth.hashCode());
        result = prime * result + ((name == null) ? 0 : name.hashCode
            ());
        result = prime * result
                + ((position == null) ? 0 : position.hashCode());
        result = prime * result
                + ((strength == null) ? 0 : strength.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Character other = (Character) obj;
        if (health == null) {
            if (other.health != null)
                return false;
        } else if (!health.equals(other.health))
            return false;
        if (level == null) {
            if (other.level != null)
                return false;
        } else if (!level.equals(other.level))
            return false;
        if (maxHealth == null) {
            if (other.maxHealth != null)
                return false;
        } else if (!maxHealth.equals(other.maxHealth))
            return false;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        if (position == null) {
            if (other.position != null)
                return false;
        } else if (!position.equals(other.position))
            return false;
        if (strength == null) {
            if (other.strength != null)
                return false;
        } else if (!strength.equals(other.strength))
            return false;
        return true;
    }

    public void setLevel(int level) {
        this.level = level;
    }

}
```

1.1.8. DungeonGameImp.java

```
package back;

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

```

```java
/**
 * @author tomas Back end most important class. It contents all the ↩
     data to play
 *          a Dungeon Game. This class implements Game.
 */
public class DungeonGameImp implements Game {

    public final static Integer LEVEL = 3;
    public final static Integer LIFE = 10;
    public final static Integer STRENGTH = 5;

    private String boardName;
    private Player player;
    private Point boardDimension;
    private Putable[][] board;
    private GameListener gameListener;
    private BoardObtainer boardObtainer;

    public DungeonGameImp(BoardObtainer boardObtainer, GameListener ↩
         gameListener) {
        this.boardObtainer = boardObtainer;
        this.gameListener = gameListener;
        boardName = boardObtainer.getBoardName();
        boardDimension = boardObtainer.getBoardDimension();
        board = boardObtainer.getBoard();
        PlayerData playerData = boardObtainer.getPlayerData();
        if (!(boardObtainer instanceof LoadGame)) {
            playerData.setName(gameListener.playerNameRequest());
        }
        player = new Player(playerData);
        firstDiscoveredCells();
    }

    private void firstDiscoveredCells() {
        Point p = player.getPosition();

        board[p.x][p.y].setVisible();

        board[p.x + 1][p.y - 1].setVisible();
        board[p.x + 1][p.y].setVisible();
        board[p.x + 1][p.y + 1].setVisible();

        board[p.x][p.y - 1].setVisible();
        board[p.x][p.y].setVisible();
        board[p.x][p.y + 1].setVisible();

        board[p.x - 1][p.y - 1].setVisible();
        board[p.x - 1][p.y].setVisible();
        board[p.x - 1][p.y + 1].setVisible();
    }

    /**
     * @see back.Game#receiveMoveStroke(back.MoveTypes) Is't allow the↩
          game to
     *       receive a Stroke. In this case a MoveTypes stroke. Before ↩
         this the
     *       player moves.
     **/
    @Override
    public void receiveMoveStroke(MoveTypes moveType) {
        Point nextPlayerPosition = player.getPosition().add(
                moveType.getDirection());
        int playerLevelBeforeFight = player.getLevel();
        if (board[nextPlayerPosition.x][nextPlayerPosition.y]
                .allowMovement(this)) {
            MoveTypes moveMade = player.move(moveType);
            dicoverBoard(nextPlayerPosition, moveType);
            gameListener.executeWhenPlayerMoves(moveMade);
            board[nextPlayerPosition.x][nextPlayerPosition.y].↩
                 standOver(this);
        }
        if (player.getLevel() != playerLevelBeforeFight) {
            gameListener.executeWhenLevelUp();
        }
    }

    /**
     * When player moves exist the possibility of discover undiscovered board
     * parts. When this happen the game have to give life to characters on the
     * parts of the board already discovered. This amount is equals of the
     * character level.
     */
    private void dicoverBoard(Point pos, MoveTypes dir) {
        int countDiscover = 0;
        List<Point> points = new ArrayList<Point>();
        points.add(pos.add(dir.getDirection()));
        if (dir == MoveTypes.LEFT || dir == MoveTypes.RIGHT) {
            points.add(pos.add(1, 0).add(dir.getDirection()));
            points.add(pos.sub(1, 0).add(dir.getDirection()));
        } else {
            points.add(pos.add(0, 1).add(dir.getDirection()));
            points.add(pos.sub(0, 1).add(dir.getDirection()));
        }

        for (Point poo : points) {
            if (!board[poo.x][poo.y].isVisible()) {
                countDiscover++;
                board[poo.x][poo.y].setVisible();
            }
        }

        if (countDiscover > 0) {
            player.winLife(countDiscover * player.getLevel());
            for (int i = 1; i < boardDimension.x - 1; i++) {
                for (int j = 1; j < boardDimension.y - 1; j++) {
                    if (board[i][j].isVisible()
                            && board[i][j] instanceof Character) {
                        ((Character) board[i][j]).winLife(countDiscover
                                * ((Character) board[i][j]).getLevel());
                    }
                }
            }
        }
    }

    @Override
    public Player getPlayer() {
        return player;
    }

    @Override
    public void winned() {
        gameListener.executeWhenGameWinned();
    }

    @Override
    public void loosed() {
        gameListener.executeWhenGameLoosed();
    }

    /**
     * @param character
     *            Method executed when a fight end. It's validate if a character
     *            died. If any died execute a listener was provided by the
     *            front.
     */
    public void fightEnd(Character character) {
        if (character.isDead()) {
            Point point = new Point(character.getPosition().x, character
                    .getPosition().y);
            BloodyFloor bf = new BloodyFloor();
            bf.setVisible();
            board[point.x][point.y] = bf;
            gameListener.executeWhenCharacterDie(point);

        }
        if (player.isDead()) {
            Point point = new Point(player.getPosition().x, player
                    .getPosition().y);
            BloodyFloor bf = new BloodyFloor();
            bf.setVisible();
            board[point.x][point.y] = bf;
            gameListener.executeWhenCharacterDie(point);
            loosed();
        }
        gameListener.executeWhenFight();

    }

    @Override
    public Putable[][] getBoard() {
        return board;
    }

    @Override
    public Point getBoardDimension() {
        return boardDimension;
    }

    @Override
    public String getBoardName() {
        return boardName;
    }

    @Override
    public GameListener getGameListener() {
        return gameListener;
    }

    @Override
    public void addGameListener(GameListener d) {
        gameListener = d;
    }

    @Override
    public BoardObtainer getBoardObtainer() {
        return boardObtainer;
    }

    /**
     * @see back.Game#restart() The desition of making restart a ↩
         method of a
     *      game and not a class like loadGame is that a restart game ↩
         need the
     *      same boardObtainer that the instance of the game. Then is ↩
         have no
     *      sense make a new instance.
     **/
    @Override
    public void restart() {
        File file = boardObtainer.getFile();
        try {
            board = boardObtainer.getClass().getConstructor(File.class↩
                )
                    .newInstance(file).getBoard();
        } catch (Exception e) {
        }
        PlayerData playerData = new PlayerData(player.getName(), 0, 0,↩
             LIFE,
                LIFE, STRENGTH, boardObtainer.getPlayerPosition(), ↩
                    player
                        .getSteps());
        player = new Player(playerData);
    }
```

```java
}
```

1.1.9. DungeonGameListener.java

```java
package back;

public interface DungeonGameListener extends GameListener{}
```

1.1.10. Floor.java

```java
package back;

public class Floor extends Cell implements Putable {

    @Override
    public String toString() {
        return "Floor";
    }

    @Override
    public boolean allowMovement(DungeonGameImp game) {
        return true;
    }

    @Override
    public void standOver(DungeonGameImp game) {}

}
```

1.1.11. Game.java

```java
package back;

public interface Game {

    public void winned();

    public void loosed();

    public Player getPlayer();

    public Putable[][] getBoard();

    public Point getBoardDimension();

    public String getBoardName();

    public GameListener getGameListener();

    public void addGameListener(GameListener d);

    public BoardObtainer getBoardObtainer();

    public void restart();

    public void receiveMoveStroke(MoveTypes moveType);

}
```

1.1.12. GameListener.java

```
package back;

public interface GameListener {

    public void executeWhenPlayerMoves(MoveTypes moveType);

    public void executeWhenFight();

    public void executeWhenBonusGrabed(Point pos);

    public void executeWhenCharacterDie(Point pos);

    public void executeWhenGameLoosed();

    public void executeWhenGameWinned();

    public String playerNameRequest();

    public void executeWhenLevelUp();

}
```

1.1.13. GrabBonus.java

```
package back;

public interface GrabBonus {
    public void grabBonus(Character character, Integer bonusAmount);
}
```

1.1.14. LoadGame.java

```
package back;

public interface LoadGame<T extends Game> {

    public T getGame(Class<T> gameImpClass, GameListener listener);

    public Integer getPlayerLoadedSteps();

    public Integer getPlayerLoadedExperience();

    public Integer getPlayerLoadedStrength();

    public int getPlayerLoadedLevel();

    public Integer getPlayerLoadedHealth();

    public Integer getPlayerLoadedMaxHealth();

    public String getPlayerName();

}
```

1.1.15. Monster.java

```java
package back;

public class Monster extends Character implements Putable {

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = super.hashCode();
        result = prime * result
                + ((monsterType == null) ? 0 : monsterType.hashCode())↩
                    ;
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (!super.equals(obj))
            return false;
        if (getClass() != obj.getClass())
            return false;
        Monster other = (Monster) obj;
        if (monsterType == null) {
            if (other.monsterType != null)
                return false;
        } else if (!monsterType.equals(other.monsterType))
            return false;
        return true;
    }

    private MonsterTypes monsterType;

    public Monster(Point position, int numberMonsterType, int level) {
        this(position, numberMonsterType, level, MonsterTypes.↩
            getMonsterType(
                numberMonsterType).getMaxLife(level));
    }

    public Monster(Point position, int numberMonsterType, int level, ↩
        int health) {
        super(MonsterTypes.getMonsterType(numberMonsterType).getName()↩
            , level,
                position);
        monsterType = MonsterTypes.getMonsterType(numberMonsterType);
        setMaxHealth(monsterType.getMaxLife(level));
        setStrength(monsterType.getStrength(level));
        setHealth(health);
    }

    public MonsterTypes getMonsterType() {
        return monsterType;
    }

    @Override
    public String toString() {
        return monsterType.getName();
    }

    @Override
    public boolean allowMovement(DungeonGameImp game) {
        game.getPlayer().fightAnotherCharacter(this);
        game.fightEnd(this);
        if (this.isDead()) {
            if (this.getLevel() == DungeonGameImp.LEVEL) {
                game.winned();
            }
        }
        return false;
    }

    @Override
    public void standOver(DungeonGameImp game) {
    }

}
```

1.1.16. MonsterTypes.java

```
package back;

public enum MonsterTypes {


    GOLEM("Golem", new Algoritms() {

        @Override
        public Integer lifeAlgoritm(int level) {
            return (int) Math.floor((((level + 3) * (level + 3)) - 10)↩
                 * GOLEMLIFE);
        }

        @Override
        public Integer strengthAlgoritm(int level) {
            return (int) Math.floor(((level * level) + 5 * level) * ↩
                0.5 * GOLEMSTRENGTH);
        }

    }), DRAGON("Dragon", new Algoritms() {

        @Override
        public Integer lifeAlgoritm(int level) {
            return (int) Math.floor((((level + 3) * (level + 3)) - 10)↩
                 * DRAGONLIFE);
        }

        @Override
        public Integer strengthAlgoritm(int level) {
            return (int) Math.floor(((level * level) + 5 * level) * ↩
                0.5 * DRAGONSTRENGTH);
        }
    }), SNAKE("Snake", new Algoritms() {

        @Override
        public Integer lifeAlgoritm(int level) {
            return (int) Math.floor((((level + 3) * (level + 3)) - 10)↩
                 * SNAKELIFE);
        }

        @Override
        public Integer strengthAlgoritm(int level) {
            return (int) Math.floor(((level * level) + 5 * level) * ↩
                0.5 * SNAKESTRENGTH);
        }

    });

    private static double GOLEMLIFE = 1;
    private static double GOLEMSTRENGTH = 0.7;
    private static double DRAGONLIFE = 1.35;
    private static double DRAGONSTRENGTH = 1;
    private static double SNAKELIFE = 1;
    private static double SNAKESTRENGTH = 1;

    private String name;
    private Algoritms lifeStrengthAlgoritms;

```

```java
    private MonsterTypes(String name, Algoritms lifeStrengthAlgoritms) {
        this.name = name;
        this.lifeStrengthAlgoritms = lifeStrengthAlgoritms;
    }

    public Integer getMaxLife(int level) {
        return lifeStrengthAlgoritms.lifeAlgoritm(level);
    }

    public Integer getStrength(int level) {
        return lifeStrengthAlgoritms.strengthAlgoritm(level);
    }

    public static MonsterTypes getMonsterType(int data) {
        switch (data) {
        case (1):
            return MonsterTypes.GOLEM;
        case (2):
            return MonsterTypes.DRAGON;
        default:
            return MonsterTypes.SNAKE;
        }
    }

    public String getName() {
        return name;
    }
}
```

1.1.17. MoveTypes.java

```java
package back;

public enum MoveTypes implements Strokes{
    UP(new Point(-1, 0)), DOWN(new Point(1, 0)), LEFT(new Point(0, -1)), RIGHT(
            new Point(0, 1));

    private Point direction;

    private MoveTypes(Point p){
        this.direction=p;
    }

    public Point getDirection(){
        return direction;
    }

    public int x(){
        return direction.x;
    }

    public int y(){
        return direction.y;
    }

}
```

1.1.18. Player.java

```java
package back;

```

```java
public class Player extends Character {

    private static Integer EXPERIENCECONSTANT = 5;

    private Integer experience;
    private Integer experienceToLevelUp;
    private Integer steps = 0;

    public Player(PlayerData playerData) {
        super(playerData.getName(), playerData.getLevel(), playerData.↩
            getPosition());
        this.experienceToLevelUp = EXPERIENCECONSTANT * getLevel();
        this.experience = playerData.getExperience();
        setMaxHealth(playerData.getMaxHealth());
        setHealth(playerData.getHealth());
        setStrength(playerData.getStrength());

    }

    public MoveTypes move(MoveTypes moveType) {
        setPosition(getPosition().add(moveType.getDirection()));
        steps++;
        return moveType;
    }

    public void winExperience(Integer experience) {
        if ((this.experience + experience) >= experienceToLevelUp) {
            levelUp();
        } else {
            this.experience += experience;
        }
    }

    private void levelUp() {
        increaseLevel();
        this.experience = 0;
        this.experienceToLevelUp = EXPERIENCECONSTANT * getLevel();
        setMaxHealth(getLevel() * DungeonGameImp.LIFE);
        setStrength(getStrength() + DungeonGameImp.STRENGTH);
    }

    public Integer getExperience() {
        return experience;
    }

    public void winFight(Character character) {
        winExperience(character.getLevel());
    }

    @Override
    public String toString() {
        String resp;
        resp = super.toString();
        resp += "Exp=" + experience;
        resp += "ExpNeeded=" + experienceToLevelUp;
        return resp;
    }

    public Integer getSteps() {
        return steps;
    }

    public Integer getExperienceToLevelUp() {
        return experienceToLevelUp;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = super.hashCode();
        result = prime * result
                + ((experience == null) ? 0 : experience.hashCode());
        result = prime
                * result
                + ((experienceToLevelUp == null) ? 0 : experienceToLevelUp
                        .hashCode());
        result = prime * result + ((steps == null) ? 0 : steps.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (!super.equals(obj))
            return false;
        if (getClass() != obj.getClass())
            return false;
        Player other = (Player) obj;
        if (experience == null) {
            if (other.experience != null)
                return false;
        } else if (!experience.equals(other.experience))
            return false;
        if (experienceToLevelUp == null) {
            if (other.experienceToLevelUp != null)
                return false;
        } else if (!experienceToLevelUp.equals(other.experienceToLevelUp))
            return false;
        if (steps == null) {
            if (other.steps != null)
                return false;
        } else if (!steps.equals(other.steps))
            return false;
        return true;
    }

}
```

1.1.19. PlayerData.java

```java
package back;

public class PlayerData {

    private String name;
    private int level;
    private int experience;
    private int maxHealth;
    private int health;
    private int strength;
    private Point position;
    private int steps;

    public PlayerData(String name, int level, int experience, int health,
            int maxHealth, int strength, Point position, int steps) {
        this.level = level;
        this.name = name;
        this.experience = experience;
        this.health = health;
        this.maxHealth = maxHealth;
        this.strength = strength;
        this.position = position;
        this.steps = steps;

    }

    public int getExperience() {
        return experience;
    }

    public void setExperience(int experience) {
        this.experience = experience;
    }

    public int getHealth() {
        return health;
    }

    public String getName() {
        return name;
    }

    public int getMaxHealth() {
        return maxHealth;
    }

    public Point getPosition() {
        return position;
    }

    public int getStrength() {
        return strength;
    }

    public int getLevel() {
        return level;
    }

    public int getSteps() {
        return steps;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```

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);

    public void standOver(DungeonGameImp game);

    public boolean isVisible();

    public void setVisible();

    public void setNotVisible();

}
```

1.1.22. SaveGame.java

```java
package back;

public interface SaveGame {
    public void save() throws Exception;
}
```

1.1.23. Strokes.java

```java
package back;

public interface Strokes {

}
```

1.1.24. Wall.java

```java
package back;

public class Wall extends Cell implements Putable {

    @Override
    public String toString() {
        return "Wall";
    }

    @Override
    public boolean allowMovement(DungeonGameImp game) {
        return false;
    }

    @Override
    public void standOver(DungeonGameImp game) {}

}
```

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() + "  ");
        playerLabel[5] = new JLabel(" ");
        this.playerLabel = playerLabel;
        for (JLabel pl : playerLabel) {
            add(pl);
        }
    }

    public void addCharacter(Monster character) {
        JLabel[] playerLabel = new JLabel[4];
        playerLabel[0] = new JLabel("  " + character.getName());
        playerLabel[0].setFont(new Font("Serif", Font.BOLD, 12));
        playerLabel[0].setForeground(Color.RED);
        playerLabel[1] = new JLabel("  " + "Health: " + character.getHealth()
                + "/" + character.getMaxHealth());
        playerLabel[2] = new JLabel("  " + "Strength: "
                + character.getStrength());
        playerLabel[3] = new JLabel("  " + "Level: " + character.getLevel());
        for (JLabel pl : playerLabel) {
            add(pl);
        }
        monstersLabels.put(character, playerLabel);
    }

    public void removeCharacter(Monster character) {
        JLabel[] labels = monstersLabels.get(character);
        for (JLabel ml : labels) {
            remove(ml);
        }
    }

    public void refresh(Game game, DungeonPanel dungeonPanel) {
        Putable[] posibleMonsters = new Putable[5];
        Point p = game.getPlayer().getPosition();

        posibleMonsters[0] = game.getBoard()[p.x + 1][p.y];
        posibleMonsters[1] = game.getBoard()[p.x - 1][p.y];
        posibleMonsters[2] = game.getBoard()[p.x][p.y + 1];
        posibleMonsters[3] = game.getBoard()[p.x][p.y - 1];
        posibleMonsters[4] = dungeonPanel.getMonsterUnderMouse();

        removeAll();

        for (int i = 0; posibleMonsters[4] != null && i < 4; i++) {
            if (posibleMonsters[4].equals(posibleMonsters[i])) {
                posibleMonsters[4] = null;
            }
        }

        addCharacter(game.getPlayer());
        for (Putable put : posibleMonsters) {
            if (put != null && put instanceof Monster) {
                addCharacter((Monster) put);
            }
        }
    }

}
```

1.2.3. DataPanelListener.java

```java
package front;


public interface DataPanelListener {

}
```

1.2.4. DefaultGameMenuBar.java

```java
package front;

import java.awt.event.ActionListener;

public interface DefaultGameMenuBar {

    public void setNewGameItemAction(ActionListener a);

    public void setRestartGameItemAction(ActionListener a);

    public void setSaveGameItemAction(ActionListener a);

    public void setSaveGameAsItemAction(ActionListener a);

    public void setLoadGameItemAction(ActionListener a);

    public void setExitGameItemAction(ActionListener a);

    public void createDefaultJMenuActionListeners();

}
```

1.2.5. DungeonGameFrame.java

```
package front;

import static professorShipSrc.ImageUtils.loadImage;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.io.File;
import java.io.IOException;

import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

import parser.BoardParserFromFile;
import parser.CorruptedFileException;
import saveLoadImplementation.LoadGameFromFile;
import saveLoadImplementation.SaveGameOnFile;
import saveLoadImplementation.SavingCorruptedException;
import back.BoardObtainer;
import back.DungeonGameImp;
import back.DungeonGameListener;
import back.LoadGame;
import back.Monster;
import back.MoveTypes;
import back.Point;
import back.Putable;

/**
 * @author tmehdi Class that extends GameFrame. It's used for the ↩
     frame of the
 *          game.
 */
public class DungeonGameFrame extends GameFrame {

    private static final long serialVersionUID = 1L;
    private DataPanel dataPanel;
    private DungeonPanel dungeonPanel;

    public DungeonGameFrame() {
        super("Dungeon game");
        setIcon();
        addKeyListener();
    }

    /**
     * DungeonGameFrame menu. It have 6 options: New game, Restart,↩
         Save game,
     * Save game as..., Load game and Exit
     *
     * @see front.GameFrame#createDefaultJMenuActionListeners()
     **/
    @Override
    public void createDefaultJMenuActionListeners() {

        setNewGameItemAction(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    if (game != null) {
                        dataPanel.setVisible(false);
                        dungeonPanel.setVisible(false);
                        remove(dataPanel);
                        remove(dungeonPanel);
                        repaint();
                        game = null;
                    }
                    File file = null;
                    LevelSelector levelSelector = new LevelSelectorImp↩
                        (
                            DungeonGameFrame.this);
                    file = levelSelector.getLevelSelected();
                    if (file != null) {
                        BoardObtainer boardObtainer = new ↩
                            BoardParserFromFile(
                                file);
                        game = new DungeonGameImp(boardObtainer,
                                new DungeonGameListenerImp());
                        setSize();
                        drawDungeonPanel();
                        drawDataPanel();
                        dataPanel.refresh(game, dungeonPanel);
                        dungeonPanel.updateUI();
                    }
                } catch (Exception e1) {
                    JOptionPane.showMessageDialog(null,
                            "Level file is corrupt", "Error",
                            JOptionPane.ERROR_MESSAGE);
                }
            }

        });

        setRestartGameItemAction(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    if (game == null) {
                        JOptionPane.showMessageDialog(null,
                                "You are not playing a level.");
                    } else {
                        game.restart();
                        dataPanel.setVisible(false);
                        dungeonPanel.setVisible(false);
                        remove(dataPanel);
                        remove(dungeonPanel);
                        drawDungeonPanel();
                        drawDataPanel();
                        dataPanel.refresh(game, dungeonPanel);
                        dungeonPanel.updateUI();
                    }
                } catch (CorruptedFileException e1) {
                    JOptionPane.showMessageDialog(null, "The file is ↩
                        corrupt",
                            "Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        });

        setSaveGameItemAction(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                if (game != null) {
                    File directory = new File("." + File.separator
                            + "savedGames");
                    if (!directory.exists()) {
                        directory.mkdir();
                    }
                    try {
                        new SaveGameOnFile(game);
                    } catch (SavingCorruptedException e1) {
                        JOptionPane.showMessageDialog(null,
                                "Files saving error occours. Try again↩
                                     later.",
                                "Error", JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        });

        setSaveGameAsItemAction(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (game != null) {
                    File directory = new File("." + File.separator
                            + "savedGames");
                    if (!directory.exists()) {
                        directory.mkdir();
                    }
                    File file;
                    JFileChooser fc = new JFileChooser();
                    fc.setCurrentDirectory(new File("." + File.↩
                         separator
                            + "savedGames"));
                    fc.showOpenDialog(DungeonGameFrame.this);
                    file = fc.getSelectedFile();
                    file = new File(file.getPath() + ".board");
                    if (file == null) {
                        JOptionPane.showMessageDialog(null,
                                "You didn't select any file.");
                    } else {
                        try {
                            new SaveGameOnFile(game, file);
                        } catch (SavingCorruptedException e1) {
                            JOptionPane
                                    .showMessageDialog(
                                            null,
                                            "Files saving error ↩
                                                occours. Try again ↩
                                                later.",
                                            "Error", JOptionPane.↩
                                                ERROR_MESSAGE);
                        }
                    }
                }
            }
        });

        setLoadGameItemAction(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                if (game != null) {
                    dataPanel.setVisible(false);
                    dungeonPanel.setVisible(false);
                    remove(dataPanel);
                    remove(dungeonPanel);
                    repaint();
                    game = null;
                }
                File file;
                JFileChooser fc = new JFileChooser();
                fc.setCurrentDirectory(new File("." + File.separator
                        + "savedGames"));
                fc.showOpenDialog(DungeonGameFrame.this);
                file = fc.getSelectedFile();
                if (file == null) {
                    JOptionPane.showMessageDialog(null,
                            "You didn't select any file.");
                } else {
                    try {
                        LoadGame<DungeonGameImp> loadGame = new ↩
                            LoadGameFromFile<DungeonGameImp>(
                                file);
                        game = loadGame.getGame(DungeonGameImp.class,
                                new DungeonGameListenerImp());
                        setSize();
                        drawDungeonPanel();
                        drawDataPanel();
                        dataPanel.updateUI();
                        dungeonPanel.updateUI();
                    } catch (CorruptedFileException e2) {
                        JOptionPane
                                .showMessageDialog(
                                        null,
                                        "Files loading error occours. ↩
                                            Try again later.",
                                        "Error", JOptionPane.↩
                                            ERROR_MESSAGE);
                    }
                }
            }
        });

        setExitGameItemAction(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    DungeonGameFrame.this.setVisible(false);
                    DungeonGameFrame.this.dispose();
                } catch (Throwable e1) {
                    JOptionPane.showMessageDialog(null, "Exit fault", ↩
                        "Error",
                            JOptionPane.ERROR_MESSAGE);
                }
            }
        });

    }

    private void setSize() {
        setSize((game.getBoardDimension().y + 2)
                * DungeonPanel.CELL_SIZE, (game
                .getBoardDimension().x)
                * DungeonPanel.CELL_SIZE - 7);
    }

    /**
     * Method to make appear the data panel.
     */
    private void drawDataPanel() {
        dataPanel = new DataPanel(game.getPlayer(), Color.GRAY);
        add(dataPanel, BorderLayout.EAST);
    }

    /**
     * Method to make appear the dungeon panel.
     */
    private void drawDungeonPanel() {
        dungeonPanel = new DungeonPanel(game, dataPanel,
                new DungeonPanelListenerImp());
        add(dungeonPanel, BorderLayout.CENTER);
    }

    /**
     * Getter of the dungeon panel.
     *
     * @return DungeonPanel
     */
    public DungeonPanel getDungeonPanel() {
        return dungeonPanel;
    }

    /**
     * Getter of the data panel.
     *
     * @return DataPanel
     */
    public DataPanel getDataPanel() {
        return dataPanel;
    }

    /**
```

```
     * Listener of the move keys, up down left right.
     *
     * @see front.GameFrame#addKeyListener()
     **/
    @Override
    public void addKeyListener() {

        addKeyListener(new KeyAdapter() {

            @Override
            public void keyPressed(final KeyEvent e) {
                switch (e.getKeyCode()) {
                case KeyEvent.VK_LEFT:
                    game.receiveMoveStroke(MoveTypes.LEFT);

                    break;
                case KeyEvent.VK_UP:
                    game.receiveMoveStroke(MoveTypes.UP);

                    break;
                case KeyEvent.VK_RIGHT:
                    game.receiveMoveStroke(MoveTypes.RIGHT);

                    break;
                case KeyEvent.VK_DOWN:
                    game.receiveMoveStroke(MoveTypes.DOWN);

                    break;
                }
            }
        });
    }

    /**
     * @author tmehdi Inner class for the listener of this game ↩
         implementation.
     */
    private class DungeonGameListenerImp implements ↩
        DungeonGameListener {

        @Override
        public void executeWhenBonusGrabed(Point p) {
            dungeonPanel.drawGrabedBonus(p);
        }

        @Override
        public void executeWhenCharacterDie(Point p) {
            dungeonPanel.drawDiedCharacter(p);
        }

        @Override
        public void executeWhenGameLoosed() {
            JOptionPane.showMessageDialog(DungeonGameFrame.this,
                    "You loose the level.");
            DungeonGameFrame.this.remove(DungeonGameFrame.this
                    .getDungeonPanel());
            DungeonGameFrame.this.remove(DungeonGameFrame.this.↩
                getDataPanel());
            repaint();
        }

        @Override
        public void executeWhenGameWinned() {
            JOptionPane.showMessageDialog(DungeonGameFrame.this, "↩
                WINNER!"
                    + '\n' + "You win the level with "
                    + game.getPlayer().getSteps() + " steps.");
            DungeonGameFrame.this.remove(DungeonGameFrame.this
                    .getDungeonPanel());
            DungeonGameFrame.this.remove(DungeonGameFrame.this.↩
                getDataPanel());
            repaint();
        }

        @Override
        public void executeWhenPlayerMoves(MoveTypes moveType) {
            dungeonPanel.drawPlayerMove(game, moveType);
            dataPanel.refresh(game, dungeonPanel);
            dataPanel.updateUI();
            dungeonPanel.drawDiscoveredCell(game, moveType);
        }

        @Override
        public String playerNameRequest() {
            String name = null;
            while (name == null || name.isEmpty()) {
                name = JOptionPane.showInputDialog("Player name");
            }
            return name;
        }

        @Override
        public void executeWhenFight() {
            dataPanel.refresh(game, dungeonPanel);
            dataPanel.updateUI();
        }

        @Override
        public void executeWhenLevelUp() {
            dungeonPanel.drawLevelUp(game);
        }
    }

    /**
     * Add the hero image as frame icon.
     */
    private void setIcon() {
        try {
            setIconImage(loadImage("./resources/images/hero.png"));
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Unexpected Error", "↩
                Error",
                    JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * @author tomas Implementation of DungeonPaneListener used for ↩
         the actions
     *          performed on dungeonPanel with the mouse.
     */
    private class DungeonPanelListenerImp implements ↩
        DungeonPanelListener {

        @Override
        public void onMouseMoved(int row, int column) {

            Monster monster = dungeonPanel.getMonsterUnderMouse();
            if (monster != null) {
                dataPanel.removeCharacter(monster);
                dungeonPanel.setMonsterUnderMouse(null);
            }
            Putable putable = game.getBoard()[row + 1][column + 1];
            if (putable instanceof Monster && putable.isVisible()) {
                dungeonPanel.setMonsterUnderMouse((Monster) putable);
                dataPanel.addCharacter(dungeonPanel.↩
                    getMonsterUnderMouse());
            }
            dataPanel.refresh(game, dungeonPanel);
            dataPanel.updateUI();

        }

    }
}
```

1.2.6. DungeonPanel.java

```java
package front;

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

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

import javax.swing.JOptionPane;

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

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

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

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

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

    /**
     * @param monsterUnderMouse
     *            Setter of the monster under mouse.
     */
    public void setMonsterUnderMouse(Monster monsterUnderMouse) {
        this.monsterUnderMouse = monsterUnderMouse;
    }

    /**
     * @param dungeonGameFrame
     *            Draw the full dungeon panel.
     */
    public void dwarFullDungeon(DungeonGameFrame dungeonGameFrame) {
        Image image;
        Image floorImage = boardImagesByClass.get(Floor.class);
        Image bloodyFloorImage = overlap(floorImage, ↩
            boardImagesByClass
                .get(BloodyFloor.class));
        int row = dungeonGameFrame.game.getBoardDimension().x - 2;
        int col = dungeonGameFrame.game.getBoardDimension().y - 2;

        for (int i = 1; i <= row; i++) {
            for (int j = 1; j <= col; j++) {
                Putable cell = dungeonGameFrame.game.getBoard()[i][j];
                if (cell.getClass().equals(Monster.class)) {
                    image = monsterImagesByName.get(((Monster) cell)
                            .getMonsterType().toString());
                    image = overlap(floorImage, image);
                    image = drawString(image, ((Character) cell).↩
                        getLevel()
                            .toString(), Color.WHITE);
                    put(image, i - 1, j - 1);
                } else if (cell.getClass().equals(Bonus.class)) {
                    image = bonusImagesByName.get(((Bonus) cell).↩
                        getBonusType()
                            .toString());
                    image = overlap(floorImage, image);
                    image = drawString(image, (((Bonus) cell).↩
                        getBonusType()
                            .getBonusAmount()).toString(), Color.RED);
                    put(image, i - 1, j - 1);
                } else {
                    image = boardImagesByClass.get(cell.getClass());
                    if (cell.getClass().equals(Wall.class)) {
                        put(image, i - 1, j - 1);
                    } else if (cell.getClass().equals(BloodyFloor.↩
                        class)) {
                        put(bloodyFloorImage, i - 1, j - 1);
                    } else {
                        put(floorImage, i - 1, j - 1);
                    }
                }
            }
        }

        Point p = new Point(dungeonGameFrame.game.getPlayer().↩
            getPosition());

        if (dungeonGameFrame.game.getBoard()[p.x][p.y] instanceof ↩
            BloodyFloor) {
            image = overlap(bloodyFloorImage, playerImage);
        }
        image = overlap(floorImage, playerImage);
        image = drawString(image, dungeonGameFrame.game.getPlayer().↩
            getLevel()
                .toString(), Color.WHITE);
        put(image, p.x - 1, p.y - 1);
    }

    /**
     * @param dungeonGameFrame
     *
     *            Draw the dungeon panel when a game begins.
     */
    private void drawDungeon(Game game) {
        drawRestOfDungeon(game);
        drawDungeonArroundPlayer(game);

    }

    /**
     * @param dungeonGameFrame
     *            Draw all the visible cells (it's just for loaded ↪
         games in this
     *            game implementation)
     */
    private void drawRestOfDungeon(Game game) {
        Image image;
        List<Point> points = new ArrayList<Point>();
        Image floorImage = boardImagesByClass.get(Floor.class);
        Image bloodyFloorImage = overlap(floorImage, ↪
            boardImagesByClass
                .get(BloodyFloor.class));

        int row = game.getBoardDimension().x - 2;
        int col = game.getBoardDimension().y - 2;

        for (int i = 1; i <= row; i++) {
            for (int j = 1; j <= col; j++) {
                Putable cell = game.getBoard()[i][j];
                if (cell.isVisible() && cell.getClass().equals(Monster↪
                    .class)) {
                    image = monsterImagesByName.get(((Monster) cell)
                            .getMonsterType().toString());
                    image = overlap(floorImage, image);
                    image = drawString(image, ((Character) cell).↪
                        getLevel()
                            .toString(), Color.WHITE);
                    put(image, i - 1, j - 1);
                    points.add(new Point(i, j));
                } else if (cell.isVisible()
                        && cell.getClass().equals(Bonus.class)) {
                    image = bonusImagesByName.get(((Bonus) cell).↪
                        getBonusType()
                            .toString());
                    image = overlap(floorImage, image);
                    image = drawString(image, (((Bonus) cell).↪
                        getBonusType()
                            .getBonusAmount()).toString(), Color.RED);
                    put(image, i - 1, j - 1);
                    points.add(new Point(i, j));
                } else {
                    if (cell.isVisible() && cell.getClass().equals(↪
                        Wall.class)) {
                        image = boardImagesByClass.get(cell.getClass()↪
                            );
                        put(image, i - 1, j - 1);
                        points.add(new Point(i, j));
                    } else if (cell.isVisible()
                            && cell.getClass().equals(BloodyFloor.↪
                                class)) {
                        put(bloodyFloorImage, i - 1, j - 1);
                        points.add(new Point(i, j));
                    } else if (cell.isVisible()) {
                        put(floorImage, i - 1, j - 1);
                        points.add(new Point(i, j));
                    }
                }
            }
        }

    }

    /**
     * @param dungeonGameFrame
     *             Draw the 8 cells around the player and the cell ↩
         under the
     *             player. Before that draw the player
     */
    private void drawDungeonArroundPlayer(Game game) {
        Image image;
        Image floorImage = boardImagesByClass.get(Floor.class);
        Image bloodyFloorImage = overlap(floorImage, ↩
            boardImagesByClass
                .get(BloodyFloor.class));

        Point pPos = game.getPlayer().getPosition();
        pPos = pPos.sub(2, 2);

        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                if (pPos.x + i > 0 && pPos.x+i < game.↩
                    getBoardDimension().x-1
                        && pPos.y + j > 0 && pPos.x+j < game.↩
                            getBoardDimension().y-1) {
                    Putable cell = game.getBoard()[pPos.x + i][pPos.y ↩
                        + j];
                    if (cell.getClass().equals(Monster.class)) {
                        image = monsterImagesByName.get(((Monster) ↩
                            cell)
                                .getMonsterType().toString());
                        image = overlap(floorImage, image);
                        image = drawString(image, ((Character) cell).↩
                            getLevel()
                                .toString(), Color.WHITE);
                        put(image, pPos.x + i - 1, pPos.y + j - 1);
                    } else if (cell.getClass().equals(Bonus.class)) {
                        image = bonusImagesByName.get(((Bonus) cell)
                                .getBonusType().toString());
                        image = overlap(floorImage, image);
                        image = drawString(image, (((Bonus) cell)
                                .getBonusType().getBonusAmount()).↩
                                    toString(),
                                Color.RED);
                        put(image, pPos.x + i - 1, pPos.y + j - 1);
                    } else {
                        image = boardImagesByClass.get(cell.getClass()↩
                            );
                        if (cell.getClass().equals(Wall.class)) {
                            put(image, pPos.x + i - 1, pPos.y + j - 1)↩
                                ;
                        } else if (cell.getClass().equals(BloodyFloor.↩
                            class)) {
                            put(bloodyFloorImage, pPos.x + i - 1, pPos↩
                                .y + j
                                    - 1);
                        } else {
                            put(floorImage, pPos.x + i - 1, pPos.y + j↩
                                - 1);
                        }
                    }
                }
            }
        }

        Point p = new Point(game.getPlayer().getPosition());

        if (game.getBoard()[p.x][p.y] instanceof BloodyFloor) {
            image = overlap(bloodyFloorImage, playerImage);
        }
        image = overlap(floorImage, playerImage);
        image = drawString(image, game.getPlayer().getLevel().toString↩
            (),
                Color.WHITE);
        put(image, p.x - 1, p.y - 1);
    }
```

```java

    /**
     * @return Getter of the monsterUnderMouse.
     */
    public Monster getMonsterUnderMouse() {
        return monsterUnderMouse;
    }

    /**
     * @param game
     *            of class Game
     * @param moveType
     *            instance of enumerative MoveTypes
     *
     *            Redraw if necessary the DungeonPanel.
     */
    public void drawPlayerMove(Game game, MoveTypes moveType) {
        Image bloodyFloor;
        Image floor;
        Point afterMove = new Point(game.getPlayer().getPosition().x, ↩
            game
                .getPlayer().getPosition().y);
        Point beforeMove = afterMove.sub(moveType.getDirection());
        floor = boardImagesByClass.get(Floor.class);
        bloodyFloor = boardImagesByClass.get(BloodyFloor.class);
        bloodyFloor = overlap(floor, bloodyFloor);
        clear(beforeMove.x - 1, beforeMove.y - 1);
        if (game.getBoard()[beforeMove.x][beforeMove.y].getClass().↩
            equals(
                BloodyFloor.class)) {
            put(bloodyFloor, beforeMove.x - 1, beforeMove.y - 1);
        } else {
            put(floor, beforeMove.x - 1, beforeMove.y - 1);
        }

        clear(afterMove.x - 1, afterMove.y - 1);
        Image image;
        if (game.getBoard()[afterMove.x][afterMove.y].getClass().↩
            equals(
                BloodyFloor.class)) {
            image = overlap(bloodyFloor, playerImage);
            image = drawString(image, game.getPlayer().getLevel().↩
                toString(),
                    Color.WHITE);
            put(image, afterMove.x - 1, afterMove.y - 1);
        } else {
            image = overlap(floor, playerImage);
            image = drawString(image, game.getPlayer().getLevel().↩
                toString(),
                    Color.WHITE);

            put(image, afterMove.x - 1, afterMove.y - 1);
        }
        updateUI();
    }

    /**
     * @param p
     *
     *            Draw blood on the floor where a character die.
     */
    public void drawDiedCharacter(Point p) {
        Image imagFloor = boardImagesByClass.get(Floor.class);
        Image imagBloodFloor = boardImagesByClass.get(BloodyFloor.↩
            class);
        clear(p.x - 1, p.y - 1);
        put(overlap(imagFloor, imagBloodFloor), p.x - 1, p.y - 1);
        repaint();

    }

    /**
     * @param p
     *
     *              Remove the image of the bonus and draw a floor.
     */
    public void drawGrabedBonus(Point p) {
        Image floor = boardImagesByClass.get(Floor.class);
        clear(p.x - 1, p.y - 1);
        put(overlap(floor, playerImage), p.x - 1, p.y - 1);
        repaint();

    }

    public void drawDiscoveredCell(Game game, MoveTypes dir) {
        Point pPos = game.getPlayer().getPosition();
        List<Point> points = new ArrayList<Point>();
        points.add(pPos.add(dir.getDirection()));
        if (dir == MoveTypes.LEFT || dir == MoveTypes.RIGHT) {
            points.add(pPos.add(1, 0).add(dir.getDirection()));
            points.add(pPos.sub(1, 0).add(dir.getDirection()));
        } else {
            points.add(pPos.add(0, 1).add(dir.getDirection()));
            points.add(pPos.sub(0, 1).add(dir.getDirection()));
        }

        Image image;
        Image floorImage = boardImagesByClass.get(Floor.class);
        Image bloodyFloorImage = overlap(floorImage, ↩
            boardImagesByClass
                .get(BloodyFloor.class));

        for (Point p : points) {
            if (p.x > 0 && p.x < game.getBoardDimension().x - 1 && p.y↩
                 > 0
                    && p.y < game.getBoardDimension().y - 1) {
                if (game.getBoard()[p.x][p.y].isVisible()) {
                    game.getBoard()[p.x][p.y].setVisible();
                    Putable cell = game.getBoard()[p.x][p.y];
                    if (cell.getClass().equals(Monster.class)) {
                        image = monsterImagesByName.get(((Monster) ↩
                            cell)
                                .getMonsterType().toString());
                        image = overlap(floorImage, image);
                        image = drawString(image, ((Character) cell).↩
                            getLevel()
                                .toString(), Color.WHITE);
                        put(image, p.x - 1, p.y - 1);
                    } else if (cell.getClass().equals(Bonus.class)) {
                        image = bonusImagesByName.get(((Bonus) cell)
                                .getBonusType().toString());
                        image = overlap(floorImage, image);
                        image = drawString(image, (((Bonus) cell)
                                .getBonusType().getBonusAmount()).↩
                                    toString(),
                                Color.RED);
                        put(image, p.x - 1, p.y - 1);
                    } else {
                        image = boardImagesByClass.get(cell.getClass()↩
                            );
                        if (cell.getClass().equals(Wall.class)) {
                            put(image, p.x - 1, p.y - 1);
                        } else if (cell.getClass().equals(BloodyFloor.↩
                            class)) {
                            put(bloodyFloorImage, p.x - 1, p.y - 1);
                        } else {
                            put(floorImage, p.x - 1, p.y - 1);
                        }
                    }
                }
            }
        }

    }

    /**
     * Method to initialize player image.
     */
    private void playerImage() {
        try {
            playerImage = loadImage("./resources/images/hero.png");
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Unexpected Error", "↩
                Error",
                    JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * Method to initialize board images.
     */
    private void boardImagesByClass() {
        try {
            boardImagesByClass.put(Wall.class,
                    loadImage("./resources/images/wall.png"));
            boardImagesByClass.put(Floor.class,
                    loadImage("./resources/images/background.png"));
            boardImagesByClass.put(BloodyFloor.class,
                    loadImage("./resources/images/blood.png"));
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Unexpected Error", "↩
                Error",
                    JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * Method to initialize bonus images.
     */
    private void bonusImagesInitialize() {
        try {
            bonusImagesByName.put("LIFE",
                    loadImage("./resources/images/healthBoost.png"));
            bonusImagesByName.put("STRENGTH",
                    loadImage("./resources/images/attackBoost.png"));
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Unexpected Error", "↩
                Error",
                    JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * Method to initialize monsters images.
     */
    private void monstersImagesInitialize() {
        try {
            monsterImagesByName.put("GOLEM",
                    loadImage("./resources/images/golem.png"));
            monsterImagesByName.put("DRAGON",
                    loadImage("./resources/images/dragon.png"));
            monsterImagesByName.put("SNAKE",
                    loadImage("./resources/images/serpent.png"));
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Unexpected Error", "↩
                Error",
                    JOptionPane.ERROR_MESSAGE);
        }
    }

    public void drawLevelUp(Game game) {
        Image image;
        Image bloodyFloor;
        Image floor;
        Point playerPos = new Point(game.getPlayer().getPosition().x, ↩
            game
                .getPlayer().getPosition().y);
        floor = boardImagesByClass.get(Floor.class);
        bloodyFloor = boardImagesByClass.get(BloodyFloor.class);
        bloodyFloor = overlap(floor, bloodyFloor);

        clear(playerPos.x - 1, playerPos.y - 1);
        if (game.getBoard()[playerPos.x][playerPos.y] instanceof ↩
            BloodyFloor) {
            image = overlap(bloodyFloor, playerImage);
            image = drawString(image, game.getPlayer().getLevel().↩
                toString(),
                    Color.WHITE);
            put(image, playerPos.x − 1, playerPos.y − 1);
        } else {
            image = overlap(floor, playerImage);
            image = drawString(image, game.getPlayer().getLevel().↩
                toString(),
                    Color.WHITE);

            put(image, playerPos.x − 1, playerPos.y − 1);
        }
        updateUI();
    }

}
```

1.2.7. DungeonPanelListener.java

```
package front;

import professorShipSrc.GamePanelListener;

public interface DungeonPanelListener extends GamePanelListener {

}
```

1.2.8. GameFrame.java

```
package front;

import java.awt.event.ActionListener;
import java.awt.event.InputEvent;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;

import back.Game;

public abstract class GameFrame extends JFrame implements ↩
    DefaultGameMenuBar {

    private static final long serialVersionUID = 1L;
    private static final int CELL_SIZE = 30;
    public Game game;
    private JMenuBar menuBar;
    private JMenu fileMenu;
    private JMenuItem newGameItem;
    private JMenuItem restartGameItem;
    private JMenuItem saveGameItem;
    private JMenuItem saveGameAsItem;
    private JMenuItem loadGameItem;
    private JMenuItem exitGameItem;

    public GameFrame(String name) {
        super(name);
        setTitle(name);
        setSize(13 * CELL_SIZE + 26, 11 * CELL_SIZE + 20);
        menuBar = new JMenuBar();
        fileMenu = new JMenu("File");
        newGameItem = fileMenu.add("New game");
        restartGameItem = fileMenu.add("Restart");
        loadGameItem = fileMenu.add("Load game");
        saveGameItem = fileMenu.add("Save game");
        saveGameAsItem = fileMenu.add("Save game as ...");
        exitGameItem = fileMenu.add("Exit");

        newGameItem.setAccelerator(KeyStroke.getKeyStroke('N',
                InputEvent.CTRL_DOWN_MASK));

        restartGameItem.setAccelerator(KeyStroke.getKeyStroke('R',
                InputEvent.CTRL_DOWN_MASK));

        saveGameItem.setAccelerator(KeyStroke.getKeyStroke('S',
                InputEvent.CTRL_DOWN_MASK));

        saveGameAsItem.setAccelerator(KeyStroke.getKeyStroke('D',
                InputEvent.CTRL_DOWN_MASK));

        loadGameItem.setAccelerator(KeyStroke.getKeyStroke('L',
                InputEvent.CTRL_DOWN_MASK));

        exitGameItem.setAccelerator(KeyStroke.getKeyStroke('Q',
                InputEvent.CTRL_DOWN_MASK));

        menuBar.add(fileMenu);
        setJMenuBar(menuBar);
        createDefaultJMenuActionListeners();
    }

    public void setNewGameItemAction(ActionListener a) {
        newGameItem.addActionListener(a);
    }

    public void setRestartGameItemAction(ActionListener a) {
        restartGameItem.addActionListener(a);
    }

    public void setSaveGameItemAction(ActionListener a) {
        saveGameItem.addActionListener(a);
    }

    public void setSaveGameAsItemAction(ActionListener a) {
        saveGameAsItem.addActionListener(a);
    }

    public void setLoadGameItemAction(ActionListener a) {
        loadGameItem.addActionListener(a);
    }

    public void setExitGameItemAction(ActionListener a) {
        exitGameItem.addActionListener(a);
    }

    public abstract void addKeyListener();

    public abstract void createDefaultJMenuActionListeners();

}
```

1.2.9. LevelSelector.java

```
package front;

import java.io.File;

```

```java
/**
 * @author tomas
 *  Interface to select level.
 */
public interface LevelSelector {

    public File getLevelSelected();

}
```

1.2.10. LevelSelectorImp.java

```java
package front;

import java.awt.Frame;
import java.io.File;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

/**
 * @author tomas Class for show the player a list of levels that are ↩
     saved on
 *         the directory boards. It use a list of directorys and some ↩
     class of
 *         java swing.
 */
public class LevelSelectorImp extends JFrame implements LevelSelector ↩
    {

    private static final long serialVersionUID = 1L;

    private File levelSelected;

    public LevelSelectorImp(Frame frameToShowOn) {

        String[] auxFiles, listBoardsShowed;
        List<String> listBoards = new ArrayList<String>();
        File directory = new File("." + File.separator + "boards");
        auxFiles = directory.list();
        for (String s : auxFiles) {
            if (s.endsWith(".board")) {
                listBoards.add(s.replace(".board", ""));
            }
        }
        listBoardsShowed = new String[listBoards.size()];
        for (int k = 0; k < listBoards.size(); k++) {
            listBoardsShowed[k] = listBoards.get(k);
        }

        Object levelSelected = JOptionPane.showInputDialog(↩
            frameToShowOn,
                "Select level", "Levels selector",
                JOptionPane.QUESTION_MESSAGE, null, listBoardsShowed,
                listBoardsShowed[0]);
        if (levelSelected != null) {
            this.levelSelected = new File("." + File.separator + "↩
                boards"
                    + File.separator + levelSelected + ".board");
        }

    }

    public File getLevelSelected() {
        return levelSelected;
    }

}
```

1.3. parser

1.3.1. BoardDimensionLine.java

```
package parser;

import back.Point;

public class BoardDimensionLine extends Lines {

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

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

    public Point getBoardDimension() {
        return boardDimension;
    }

}
```

1.3.2. BoardLine.java

```
package parser;

import back.Point;

public class BoardLine extends Lines {

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

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

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

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

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

```
                break;

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

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

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

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

            default:
                throw new CorruptedFileException();
            }
    }

    public boolean isPlayerLine() {
        return data[0] == 1;
    }

    public boolean isWallLine() {
        return data[0] == 2;
    }

    public boolean isMonsterLine() {
        return data[0] == 3;
    }

    public boolean isBonusLine() {
        return data[0] >= 4;
    }
}
```

1.3.3. BoardNameLine.java

```
package parser;

public class BoardNameLine extends Lines {

    private static final int elemsCuantity = 1;
    private String name;

    public BoardNameLine(String line) {
        super(elemsCuantity, line);
        this.name = getLine();
    }

    @Override
    protected void lineProcess() {}

    public String getName() {
        return name;
    }

}
```

1.3.4. BoardParserFromFile.java

```
package parser;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import back.BoardObtainer;
import back.Bonus;
import back.DungeonGameImp;
import back.Floor;
import back.Monster;
import back.PlayerData;
import back.Point;
import back.Putable;
import back.Wall;

/**
 * @author tomas Class full dedicated to read a file and transform it ↪
     to a
 *         board.
 */
public class BoardParserFromFile implements BoardObtainer {

    private BufferedReader inputBoard;
    private Point boardDimension;
    private String boardName;
    private Point playerPosition;
    private Putable[][] board;
    private File inputFile;

    public BoardParserFromFile(File file) {
        try {
            inputFile = file;
            inputBoard = new BufferedReader(new FileReader(file));
            obtainBoard();
        } catch (IOException e) {
            throw new CorruptedFileException();
        }
    }

    public void obtainBoard() throws IOException {

        boolean dimensionFlag = false;
        boolean nameFlag = false;
        boolean playerFlag = false;
        String line;

        while ((line = inputBoard.readLine()) != null) {

            line = line.replace(" ", "").replace("\t", "").replace("\n↩
                ", "")
                    .split("#")[0];

            if (!line.isEmpty()) {
                if (!dimensionFlag) {
                    parseDimension(line);
                    dimensionFlag = true;
                } else if (!nameFlag) {
                    parseBoardName(line);
                    nameFlag = true;
                } else {
                    if (line.startsWith("1")) {
                        if (playerFlag == true) {
                            throw new CorruptedFileException();
                        }
                        parsePlayer(line);
                        playerFlag = true;
                    } else {
                        BoardLine cell = new BoardLine(line, ↩
                            boardDimension);
                        Point point = (new Point(cell.getData(1), cell
                                .getData(2))).add(new Point(1, 1));

                        if (cell.isWallLine()) {
                            parseWall(point, cell);
                        } else if (cell.isMonsterLine()) {
                            parseMonster(point, cell);
                        } else if (cell.isBonusLine()) {
                            parseBonus(point, cell);
                        }
                    }
                }
            }
        }

        if (!nameFlag || !playerFlag || !dimensionFlag) {
            throw new CorruptedFileException();
        }
        validation();
    }

    public void validation() {
        protectionWalls();
        putFloor();
        if (!(board[getPlayerPosition().x][getPlayerPosition().y] ↩
            instanceof Floor)) {
            throw new CorruptedFileException();
        }
    }

    public void parseBonus(Point point, BoardLine cell) {
        putCell(point.x, point.y, new Bonus(point, cell.getData(0), ↩
            cell
                .getData(5)));
    }

    public void parsePlayer(String line) {
        BoardLine cell = new BoardLine(line, boardDimension);
        Point point = (new Point(cell.getData(1), cell.getData(2)))
                .add(new Point(1, 1));
        playerPosition = point;
    }

    public void parseMonster(Point point, BoardLine cell) {
        putCell(point.x, point.y, new Monster(point, cell.getData(3), ↩
            cell
                .getData(4)));
    }

    public void parseWall(Point point, BoardLine cell) {
        putCell(point.x, point.y, new Wall());
    }

    public void parseBoardName(String line) {
        BoardNameLine boardNameLine = new BoardNameLine(line);
        this.boardName = boardNameLine.getName();
    }

    public void parseDimension(String line) {
        BoardDimensionLine boardDimensionLine = new BoardDimensionLine↩
            (line);
        boardDimension = boardDimensionLine.getBoardDimension().add(
                new Point(2, 2));
        board = new Putable[boardDimension.x][boardDimension.y];

    }

    public void putFloor() {
        for (int i = 1; i < boardDimension.x - 1; i++) {
            for (int j = 1; j < boardDimension.y - 1; j++) {
                if (getBoardElem(i, j) == null) {
                    putCell(i, j, new Floor());
                }
            }
        }
    }

    public void protectionWalls() {
        for (int i = 0; i < boardDimension.y; i++) {
            Wall aux = new Wall();
            aux.setVisible();
            putCell(0, i, aux);
            Wall aux1 = new Wall();
            aux1.setVisible();
            putCell(boardDimension.x - 1, i, aux1);
        }
        for (int i = 0; i < boardDimension.x; i++) {
            Wall aux = new Wall();
            aux.setVisible();
            putCell(i, 0, aux);
            Wall aux1 = new Wall();
            aux1.setVisible();
            putCell(i, boardDimension.y - 1, aux1);
        }

    }

    public Point getBoardDimension() {
        return boardDimension;
    }

    public String getBoardName() {
        return boardName;
    }

    public Point getPlayerPosition() {
        return playerPosition;
    }

    public Putable[][] getBoard() {
        return board;
    }

    public int getBoardRows() {
        return boardDimension.x;
    }

    public int getBoardColums() {
        return boardDimension.y;
    }

    public Putable getBoardElem(Point position) {
        return board[position.x][position.y];
    }

    public Putable getBoardElem(int x, int y) {
        return board[x][y];
    }

    public void putCell(int i, int j, Putable cell) {
        putCell(new Point(i, j), cell);
    }

    public void putCell(Point p, Putable cell) {
        board[p.x][p.y] = cell;
    }

    @Override
    public File getFile() {
        return inputFile;
    }

    @Override
    public PlayerData getPlayerData() {
        PlayerData playerData = new PlayerData(null, 1, 0, ↩
            DungeonGameImp.LIFE, DungeonGameImp.LIFE, DungeonGameImp.↩
            STRENGTH,
                playerPosition, 0);
        return playerData;
    }

}
```

1.3.5. CorruptedFileException.java

```
package parser;

public class CorruptedFileException extends RuntimeException {

    private static final long serialVersionUID = 1L;

}
```

1.3.6. Lines.java

```
package parser;

public abstract class Lines {

    protected int[] data;
    private final int elemsCuantity;
    private String line;

    public Lines(int elemsCuantity, String line) {
        this.elemsCuantity = elemsCuantity;
        this.line = line;
    }

    /**
     * Process the line parsed by separating it by "," and removing ↩
        the spaces,
     * enters and tabs in between.
     *
     */
    protected void lineProcess() {
        data = new int[elemsCuantity];
        int k = 0;
        String[] arrayString;

```

```java
        arrayString = line.split(",");

        if (arrayString.length == elemsCuantity) {
            for (k = 0; k < elemsCuantity; k++) {
                try {
                    data[k] = Integer.valueOf(arrayString[k]);
                } catch (NumberFormatException e) {
                    throw new CorruptedFileException();
                }
            }
        } else {
            System.out.println(line);
            throw new CorruptedFileException();
        }
    }

    public int getData(int i) {
        return data[i];
    }

    public String getLine() {
        return line;
    }

    protected void lineCheck(){}
}
```

1.3.7. SavedBoardPlayerLine.java

```java
package parser;

import back.Point;

public class SavedBoardPlayerLine extends Lines {

    private static int elemsCuantity = 10;
    private Point boardDimension;
    private String playerName;

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

    @Override
    protected void lineProcess() {
        data = new int[elemsCuantity];
        int k = 0;
        String[] arrayString;

        arrayString = getLine().split(",");

        if (arrayString.length == elemsCuantity) {
            for (k = 0; k < elemsCuantity - 1; k++) {
                try {
                    data[k] = Integer.valueOf(arrayString[k]);
                } catch (NumberFormatException e) {
                    throw new CorruptedFileException();
                }
            }
            playerName = arrayString[elemsCuantity - 1];
        } else {
            throw new CorruptedFileException();
        }
    }

    @Override
    protected void lineCheck() {

        if (data[1] < 0 || data[1] >= boardDimension.x − 2 || data[2] ↩
            < 0
                || data[2] >= boardDimension.y || data[3] < 0
                || data[3] > data[4] || data[5] < 0) {
            throw new CorruptedFileException();
        }
    }

    public String getPlayerName() {
        return playerName;
    }

}
```

1.4. professorShipSrc

1.4.1. GamePanel.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.
     */
    public void put(Image image, int row, int column) {
        images[row][column] = image;
    }

    /**
     * Elimina la imagen ubicada en la fila y columna indicadas.
     */
    public void clear(int row, int column) {
        images[row][column] = null;
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(color);
        g.fillRect(0, 0, columns * cellSize, rows * cellSize);

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                if (images[i][j] != null) {
                    g.drawImage(images[i][j], j * cellSize, i * ↩
                        cellSize, null);
                }
            }
        }
    }
}
```

1.4.2. GamePanelListener.java

```
package professorShipSrc;

/**
 * Listener para eventos ocurridos en el GamePanel.
 */
public interface GamePanelListener {

    /**
     * Notifica cuando el usuario ubica el mouse sobre una celda de la↩
         grilla.
     */
    public void onMouseMoved(int row, int column);
}
```

1.4.3. ImageUtils.java

```java
package professorShipSrc;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;

import javax.imageio.ImageIO;

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

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

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

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

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

1.5. saveLoadImplementation

1.5.1. Criteria.java

```
package saveLoadImplementation;

public interface Criteria<T> {
    boolean satisfies(T obj);
}
```

1.5.2. FilterArrayFileList.java

```
package saveLoadImplementation;

import java.io.File;
import java.util.ArrayList;

public class FilterArrayFileList extends ArrayList<File> implements
        FilterFileList {

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

    public FilterArrayFileList() {
    }

    public FilterArrayFileList(File file) {
        if (file.isDirectory()) {
            File[] files = file.listFiles();
            for (File f : files) {
                this.add(f);
            }
        }
    }

    @Override
    public FilterFileList filter(String string) {
        FilterArrayFileList filterArrayFileList = new ↩
            FilterArrayFileList();
        for (File t : this) {
            if (t.getName().startsWith(string)) {
                filterArrayFileList.add(t);
            }
        }
        return filterArrayFileList;
    }

}
```

1.5.3. FilterFileList.java

```java
package saveLoadImplementation;

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

public interface FilterFileList extends List<File>{

    public FilterFileList filter(String string);

}
```

1.5.4. LoadGameFromFile.java

```java
package saveLoadImplementation;

import java.io.File;

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

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

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

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

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

    }

    private void setBoardCellVisivility(Point point, int num) {
        if (num == 0) {
            getBoardElem(point).setVisible();
        } else {
            getBoardElem(point).setNotVisible();
        }
    }

    @Override
    public void parseWall(Point point, BoardLine cell) {
        if (cell.getData(3) == 2) {
            putCell(point, new BloodyFloor());
        } else if (cell.getData(3) == 1) {
            putCell(point, new Floor());
        } else {
            super.parseWall(point, cell);
        }
        setBoardCellVisivility(point, cell.getData(5));
    };

    @Override
    public void parseBonus(Point point, BoardLine cell) {
        super.parseBonus(point, cell);
        setBoardCellVisivility(point, cell.getData(4));
    }

    @Override
    public void parseMonster(Point point, BoardLine cell) {
        putCell(point.x, point.y, new Monster(point, cell.getData(3), cell
                .getData(4), Math.abs(cell.getData(5))));
        if (cell.getData(5) < 0) {
            setBoardCellVisivility(point, 0);
        } else if (cell.getData(5) > 0) {
            setBoardCellVisivility(point, 1);
        }
    }

    @Override
    public Point getPlayerPosition() {
        return playerLoadedPosition;
    }

    @Override
    public Integer getPlayerLoadedHealth() {
        return playerLoadedHealth;
    }

    @Override
    public Integer getPlayerLoadedMaxHealth() {
        return playerLoadedMaxHealth;
    }

    @Override
    public Integer getPlayerLoadedExperience() {
        return playerLoadedExperience;
    }

    @Override
    public Integer getPlayerLoadedStrength() {
        return playerLoadedStrength;
    }

    @Override
    public Integer getPlayerLoadedSteps() {
        return playerLoadedSteps;
    }

    public T getGame(Class<T> gameImpClass, GameListener listener) {
        T game;
        try {
            game = gameImpClass.getConstructor(BoardObtainer.class,
                    GameListener.class).newInstance(this, listener);
        } catch (Exception e) {
            e.printStackTrace();
            throw new CorruptedFileException();
        }
        return game;
    }

    @Override
    public int getPlayerLoadedLevel() {
        return loadedLevel;
    }

    @Override
    public String getPlayerName() {
        return playerName;
    }

    @Override
    public PlayerData getPlayerData() {
        PlayerData playerData = new PlayerData(playerName, loadedLevel ↩
            ,
                playerLoadedExperience, playerLoadedHealth,
                playerLoadedMaxHealth, playerLoadedStrength,
                playerLoadedPosition, playerLoadedSteps);
        return playerData;
    }

}
```

1.5.5. SaveGameOnFile.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 + ")"
                    + ".board");
        } else {
            placeToSave = new File("./savedGames/savedGame.board");
        }
        try {
            save();
        } catch (IOException e) {
            throw new SavingCorruptedException();
        }
```

```java
    }

    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
                    + ")"+ ").board");
        } 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
     * visible 1 not visible]) floor(2,row pos, col pos, 1 ,0,[0 is visible 1
     * not visible]) monsters (3,row pos, col pos, monster type, level, [0 is
     * visible 1 not visible]) bonus (4 or 5, row pos, col pos, 0,[0 is visible
     * 1 not visible],amount of bonus)
     **/
    public void save() throws IOException {
        placeToSave.createNewFile();
        BufferedWriter out = new BufferedWriter(new FileWriter(placeToSave));
        out.write("#Board dimensions");
        out.newLine();
        out.write((gameToSave.getBoardDimension().x - 2) + ","
                + (gameToSave.getBoardDimension().y - 2));
        out.newLine();
        out.write("#Board name");
        out.newLine();
        out.write(gameToSave.getBoardName());
        out.newLine();
        out.write("#Player current position, "
                + "current exp, current health, maxHealth, current strength, steps, name");
        out.newLine();
        out.write(1 + "," + (gameToSave.getPlayer().getPosition().x - 1) + ","
                + (gameToSave.getPlayer().getPosition().y - 1) + ","
                + gameToSave.getPlayer().getExperience() + ","
                + gameToSave.getPlayer().getHealth() + ","
                + gameToSave.getPlayer().getMaxHealth() + ","
                + gameToSave.getPlayer().getStrength() + ","
                + gameToSave.getPlayer().getSteps() + ","
                + gameToSave.getPlayer().getLevel() + ","
                + gameToSave.getPlayer().getName());
        out.newLine();
        out.write("#Map");
        out.newLine();
        for (int i = 1; i < gameToSave.getBoardDimension().x - 1; i++) {
            for (int j = 1; j < gameToSave.getBoardDimension().y - 1; j++) {
                if (Wall.class.equals((gameToSave.getBoard()[i][j]).getClass())) {
                    out.write(2 + "," + (i - 1) + "," + (j - 1) + "," + 0 + ","
                            + 0 + ",");
                    if (gameToSave.getBoard()[i][j].isVisible()) {
                        out.write("0");
                    } else {
                        out.write("1");
                    }
                    out.newLine();
                } else if (Floor.class.equals((gameToSave.getBoard()[i][j])
                        .getClass())) {
                    out.write(2 + "," + (i - 1) + "," + (j - 1) + "," + 1 + ","
                            + 0 + ",");
                    if (gameToSave.getBoard()[i][j].isVisible()) {
                        out.write("0");
                    } else {
                        out.write("1");
                    }
                    out.newLine();
                } else if (BloodyFloor.class
                        .equals((gameToSave.getBoard()[i][j]).getClass())) {
                    out.write(2 + "," + (i - 1) + "," + (j - 1) + "," + 2 + ","
                            + 0 + ",");
                    if (gameToSave.getBoard()[i][j].isVisible()) {
                        out.write("0");
                    } else {
                        out.write("1");
                    }
                    out.newLine();
                } else if (Monster.class.equals((gameToSave.getBoard()[i][j])
                        .getClass())) {
                    out.write(3
                            + ","
                            + (i - 1)
                            + ","
                            + (j - 1)
                            + ","
                            + (((Monster) gameToSave.getBoard()[i][j])
                                    .getMonsterType().ordinal() + 1)
                            + ","
                            + ((Monster) gameToSave.getBoard()[i][j])
                                    .getLevel() + ",");
                    if (gameToSave.getBoard()[i][j].isVisible()) {
                        out.write((((Monster) gameToSave.getBoard()[i][j])
                                .getHealth() * -1) + "");
                    } else {
                        out.write((((Monster) gameToSave.getBoard()[i][j])
                                .getHealth()) + "");
                    }
                    out.newLine();
                } else if (Bonus.class.equals((gameToSave.getBoard()[i][j])
                        .getClass())) {
                    out.write((((Bonus) gameToSave.getBoard()[i][j])
                            .getBonusType().ordinal() + 4)
                            + ","
                            + (i - 1)
                            + "," + (j - 1) + "," + 0 + ",");
                    if (gameToSave.getBoard()[i][j].isVisible()) {
                        out.write("0");
                    } else {
                        out.write("1");
                    }
                    out.write(","
                            + ((Bonus) gameToSave.getBoard()[i][j])
```

```
                                        .getAmountBonus());
                    out.newLine();
                }
            }
        }

        out.flush();
        out.close();

    }
}
```

1.5.6. SavingCorruptedException.java

```
package saveLoadImplementation;

public class SavingCorruptedException extends RuntimeException {

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

}
```

1.6. tests

1.6.1. GameTests.java

```
package tests;

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

import java.io.File;

import javax.swing.JOptionPane;

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

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

public class GameTests {

    private DungeonGameImp game;

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

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

            @Override
            public void executeWhenPlayerMoves(MoveTypes moveType) {
            }

            @Override
            public void executeWhenGameWinned() {
            }

            @Override
            public void executeWhenGameLoosed() {
            }

            @Override
            public void executeWhenCharacterDie(Point p) {
            }

            @Override
            public void executeWhenBonusGrabed(Point p) {
            }

            @Override
            public void executeWhenFight() {
            }

            @Override
            public void executeWhenLevelUp() {
            }
        });
    }

    @Test
    public void goodFunctionamientOfmovePlayerTest() {
        game.receiveMoveStroke(MoveTypes.LEFT);
        game.receiveMoveStroke(MoveTypes.LEFT);
        assertEquals(new Integer(4), game.getPlayer().getHealth());
        System.out.println(game.getPlayer().getExperience());
        assertEquals(new Integer(1), game.getPlayer().getExperience())↩
            ;
        game.receiveMoveStroke(MoveTypes.LEFT);
        assertEquals(new Point(4, 3), game.getPlayer().getPosition());
        game.receiveMoveStroke(MoveTypes.RIGHT);
        assertEquals(new Point(4, 4), game.getPlayer().getPosition());
        game.receiveMoveStroke(MoveTypes.DOWN);
        assertEquals(new Point(5, 4), game.getPlayer().getPosition());
        game.receiveMoveStroke(MoveTypes.UP);
        assertEquals(new Point(4, 4), game.getPlayer().getPosition());
    }

    @Test
    public void goodFunctionamientOfWiningWhenKillMonsterLevel3Test() ↩
        {
        game.getPlayer().winLife(40);
        Bonus bonus = new Bonus(new Point(7,7),4,50);
        Bonus bonus2 = new Bonus(new Point(7,7),5,50);
        bonus.giveBonus(game.getPlayer());
        bonus2.giveBonus(game.getPlayer());
        game.getPlayer().setPosition(new Point(8, 2));
        game.receiveMoveStroke(MoveTypes.LEFT);
    }

    @Test
    public void goodFunctionamientOfResetGameTest() {
        game.getPlayer().winLife(40);
        Bonus bonus = new Bonus(new Point(7,7),4,50);
        Bonus bonus2 = new Bonus(new Point(7,7),5,50);
        bonus.giveBonus(game.getPlayer());
        bonus2.giveBonus(game.getPlayer());
        game.getPlayer().setPosition(new Point(4, 6));
        game.receiveMoveStroke(MoveTypes.UP);
        assertEquals(BloodyFloor.class, ((game.getBoard()[3][6])).↩
            getClass());
        game.restart();
        assertEquals(Monster.class, ((game.getBoard()[3][6])).getClass↩
            ());
        assertEquals(new Point(4, 4), game.getPlayer().getPosition());
    }

    @Test
    public void forWatchTheGameSavedTest() {
        File directory = new File("./savedGames");
        if (!directory.exists()) {
            directory.mkdir();
        }
        new SaveGameOnFile(game);
        File file = new File("./savedGames");
        FilterFileList filterFileList = new FilterArrayFileList(file);
        filterFileList = filterFileList.filter("savedGame");
        int number = filterFileList.size();
        if (number > 1) {
            File f = new File("./savedGames/savedGame" + "(" + (number↩
                - 1)
                    + ")" + ".board");
            assertTrue(f.exists());
            f.delete();
        } else {
            File f = new File("./savedGames/savedGame.board");
            assertTrue(f.exists());
            f.delete();
        }
    }

    @Test
    public void loadGameTest() {
        File file = new File("./savedGames/testWithPath.board");
        new SaveGameOnFile(game, file);
        LoadGame<DungeonGameImp> loadGame = new LoadGameFromFile<↩
            DungeonGameImp>(file);
        DungeonGameImp game = loadGame.getGame(DungeonGameImp.class, ↩
            new DungeonGameListener() {

            @Override
            public String playerNameRequest() {
                String name = null;
                while (name == null || name.isEmpty()) {
                    name = JOptionPane.showInputDialog("Player name");
                }
                return name;
            }

            @Override
            public void executeWhenPlayerMoves(MoveTypes moveType) {
            }

            @Override
            public void executeWhenGameWinned() {
            }

            @Override
            public void executeWhenGameLoosed() {
            }

            @Override
            public void executeWhenCharacterDie(Point p) {
            }

            @Override
            public void executeWhenBonusGrabed(Point p) {
            }

            @Override
```

```java
            public void executeWhenFight() {
            }

            @Override
            public void executeWhenLevelUp() {
            }
        });
        assertEquals(new Integer(0), game.getPlayer().getExperience());
        assertEquals(new Point(4, 4), game.getPlayer().getPosition());
        file.delete();
    }

    @Test
    public void forWatchTheGameSavedWithPathTest() {
        File directory = new File("./savedGames.board");
        if (!directory.exists()) {
            directory.mkdir();
        }
        File file = new File("./savedGames/testWithPath.board");
        new SaveGameOnFile(game, file);
        FilterFileList filterFileList = new FilterArrayFileList(
                file.getParentFile());
        filterFileList = filterFileList.filter(file.getName());
        int number = filterFileList.size();
        if (number > 1) {
            File f = new File(file.getPath() + "(" + (number - 1) + ")");
            assertTrue(f.exists());
            f.delete();
        } else {
            File f = new File(file.getPath());
            assertTrue(f.exists());
            f.delete();
        }
    }

}
```

1.6.2. PlayerTests.java

```java
package tests;

import static org.junit.Assert.assertEquals;

import java.io.File;

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

import parser.BoardParserFromFile;
import back.BoardObtainer;
import back.Bonus;
import back.Monster;
import back.MoveTypes;
import back.Player;
import back.PlayerData;
import back.Point;

public class PlayerTests {
    BoardObtainer boardParser;
    Player player;

    @Before
    public void setup() {
        boardParser = new BoardParserFromFile(new File(
                "./testBoard/boardForTest1.board"));
        player = new Player(new PlayerData("Tomas", 0, 0, 10, 10, 5,
                boardParser.getPlayerPosition(),0));
    }

    @Test
    public void goodFunctionamientPlayerMovementTest() {
        assertEquals(new Point(4, 4), player.getPosition());
        player.move(MoveTypes.UP);
        assertEquals(new Point(3, 4), player.getPosition());
        player.move(MoveTypes.LEFT);
        assertEquals(new Point(3, 3), player.getPosition());
        player.move(MoveTypes.DOWN);
        assertEquals(new Point(4, 3), player.getPosition());
        player.move(MoveTypes.RIGHT);
        assertEquals(new Point(4, 4), player.getPosition());
    }

    @Test
    public void goodFunctionamientPlayerVsMonsterFightTest() {
        Monster monster = ((Monster) boardParser.getBoard()[5][7]);
        player.fightAnotherCharacter(monster);
        assertEquals(
                new Integer(player.getMaxHealth() - monster.↩
                    getStrength()),
                player.getHealth());
        assertEquals(
                new Integer(monster.getMaxHealth() - player.↩
                    getStrength()),
                monster.getHealth());
    }

    @Test
    public void goodFunctionamientPlayerEarningBonusTest() {
        player.hited(9);
        ((Bonus) boardParser.getBoard()[8][2]).giveBonus(player);
        ((Bonus) boardParser.getBoard()[2][8]).giveBonus(player);
        assertEquals(new Integer(6), player.getHealth());
        assertEquals(new Integer(8), player.getStrength());

    }

}
```

1.6.3. ParserTests.java

```
package tests;

import static org.junit.Assert.assertEquals;

import java.io.File;

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

import parser.BoardParserFromFile;
import parser.CorruptedFileException;
import back.BoardObtainer;
import back.Bonus;
import back.Monster;
import back.MonsterTypes;
import back.Point;
import back.Wall;

public class ParserTests {

    BoardObtainer boardParser;

    @Before
    public void setup() {
        boardParser = new BoardParserFromFile(new File(
                "./testBoard/boardForTest1.board"));
```

```java
    }

    @Test(expected = CorruptedFileException.class)
    public void startPlayerPositionOverAMonsterTest() {
        new BoardParserFromFile(new File("./testBoard/boardForTest2.↩
            board"));
    }

    @Test(expected = CorruptedFileException.class)
    public void startPlayerPositionOverAWallTest() {
        new BoardParserFromFile(new File("./testBoard/boardForTest3.↩
            board"));
    }

    @Test
    public void mapWithoutSurroundingWalls() {
        BoardObtainer boardParser = new BoardParserFromFile(new File(
                "./testBoard/boardForTest4.board"));
        assertEquals(Wall.class, boardParser.getBoardElem(new Point(0,↩
             0))
                .getClass());
        assertEquals(Wall.class, boardParser.getBoardElem(new Point↩
            (11, 0))
                .getClass());
        assertEquals(Wall.class, boardParser.getBoardElem(new Point(0,↩
             11))
                .getClass());
        assertEquals(Wall.class, boardParser.getBoardElem(new Point↩
            (11, 11))
                .getClass());
    }

    @Test(expected = CorruptedFileException.class)
    public void positionOutOfBoardDimensionsTest() {
        new BoardParserFromFile(new File("./testBoard/boardForTest5.↩
            board"));
    }

    @Test(expected = CorruptedFileException.class)
    public void badPathPassedTest() {
        new BoardParserFromFile(new File("./noExist"));
    }

    @Test
    public void goodParseOfBoardDimensionTest() {
        assertEquals(new Point(12, 12), boardParser.getBoardDimension↩
            ());
    }

    @Test
    public void goodParseOfBoardNameTest() {
        assertEquals("ejemplotablero", boardParser.getBoardName());
    }

    @Test
    public void goodParseOfPlayerPositionTest() {
        assertEquals(new Point(4, 4), boardParser.getPlayerPosition())↩
            ;
    }

    @Test
    public void goodParseOfAnyCellPositionTest() {
        assertEquals(Wall.class, boardParser.getBoard()[1][1].getClass↩
            ());
        assertEquals(Wall.class, boardParser.getBoard()[10][1].↩
            getClass());
        assertEquals(Wall.class, boardParser.getBoard()[1][10].↩
            getClass());
        assertEquals(Wall.class, boardParser.getBoard()[10][10].↩
            getClass());
        assertEquals(Bonus.class,
                boardParser.getBoard()[2][8].getClass());
        assertEquals(Bonus.class, boardParser.getBoard()[8][2].↩
            getClass());
        assertEquals(Monster.class, boardParser.getBoard()[5][7].getClass());
        assertEquals(Monster.class, boardParser.getBoard()[3][6].getClass());
        assertEquals(Monster.class, boardParser.getBoard()[2][4].getClass());
    }

    @Test
    public void goodParseOfMonsterTest() {
        assertEquals(MonsterTypes.DRAGON,
                ((Monster) boardParser.getBoard()[9][2]).getMonsterType());
        assertEquals(new Integer(3),
                ((Monster) boardParser.getBoard()[9][2]).getLevel());
    }

    @Test
    public void goodParseOfBonusTest() {
        assertEquals(5,
                ((Bonus) boardParser.getBoard()[8][2]).getAmountBonus());
        assertEquals(3,
                ((Bonus) boardParser.getBoard()[2][8])
                        .getAmountBonus());
    }

    @Test
    public void boardWatchTest() {
        String resp = "";
        for (int i = 0; i < boardParser.getBoardRows(); i++) {
            for (int j = 0; j < boardParser.getBoardColums(); j++) {
                resp += boardParser.getBoard()[i][j] + " ";
            }
            resp += "\n";
        }
        System.out.println(resp);
    }

}
```