Sokoban

Guillermo Román

Programming Project

Escuela Técnica Superior de Ingenieros en Informática
Universidad Politécnica de Madrid

November 4, 2019

1 Norms

- A first meeting with the teacher must take place between the following dates:

  10th December 2019 - 20th December 2019

  This meeting is the presentation of the first *sprint* you have to do in the project. In this metting, the design of the project will be discussed with the teacher and **some requirements of the project should be working for this meeting**. The mark of this first evaluation will be the **20% of the final grade of the project**.

- The deadline for finishing the project is on:

  Friday 18th January 2018

  Modifications done after this date will not be considered.

- For the project evaluation, the project source code will be taken from the `GitLab` URL provided in the registration form, thus, check that you have the correct version pushed in the repository for the deadline date.

- A system for detecting plagiarisms will be used and the groups involved in plagiarism will be penalized with a failing grade. According to the UPM norms, other disciplinary measures can be adopted in case of plagiarism.

2 Tools

The use of the following tools and libraries is mandatory:

- `Java` 1.8 (or greater)
- `GIT` as version control system
- `JUnit` 5 for test automation
- `Maven` as build tool

2.1 `GitLab`

Remember that the use of `GitLab` as remote GIT server is mandatory. The `GitLab` server of our course is available at the URL:

`http://costa.ls.fi.upm.es/gitlab`

The `GitLab` project used to develop the assignment must fulfill the following requirements:

- The *Project visibility* must be **private**. If your code is accessible by other groups is your responsibility.
- All members of the group must be registered in `GitLab` and your username must be your *número de matrícula*.
- All project members must have, at least, *developer* role for accessing the project.
- Email notifications are mandatory and must be sent to all project members from the beginning of the project.
- The username `@pproject` must have access to the project with *developer* role.
- In order to monitor the activity of the project, the email `programming.project@fi.upm.es` must be included in the notification emails from the beginning of the assignment.

The repository must include a README.md file. The README.md file must contain the authors of the project. This file must also include relevant information to use the system, mainly a description of the interface and how to use it. Remember that the format of the README.md file is *Markdown*[1].

2.2 `Maven`

The use of `Maven` as build tool is also mandatory and the file `pom.xml` file must be in the root directory of the project. The project must follow the standard directory layout of Maven and details of this layout can be found at:

`https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html`

The `pom.xml` must include the dependencies needed to work with `JUnit` 5.20 and it must work without any manual intervention, that is, after cloning the GIT project, the command `mvn test` must properly work passing all tests of the system.

The ***groupid*** of the project must be `es.upm.pproject` and the ***artifactid*** must include the name of the project, that is, *sokoban*.

2.3 `JUnit`

It is mandatory to develop your own test suite using `JUnit 5.20` to check the correct behaviour of the program developed. Take a look to the *Testing* slides to read the good practices in testing. Remember that all tests must be automatically run by the `Maven` commands. Those tests that do not executed automatically by means of the `Maven` command will not be considered.

[1]`https://www.markdownguide.org/`

2.4 SonarQube

`SonarQube` will be used to evaluate the assignment. In addition to the functional requirements described in Section 4 your project must passed the *Quality Gate* established in `SonarQube` server. The `SonarQube` server that must be used is accessible by means of the following URL:

`http://costa.ls.fi.upm.es:9000/sonar`

Remember that he use of `SonarLint` Eclipse plugin is recommended:

`https://www.sonarlint.org/`

Its installation instructions can be found in Eclipse Marketplace:

`https://marketplace.eclipse.org/content/sonarlint`

2.5 Continuous Integration

The use of *Continuous Integration* ideas is recommended and the project should include a file `.gitlab-ci.yml` which launches `mvn test` in order to run the tests.

2.6 Agile

The project must be developed following the ideas and good practices of Agile (see the slides of Agile for details). To do so, you have to use `GitLab` **to register the *backlog* of the project**. The GitLab of the project must include the following elements:

- The project must include (at least) **two milestones**, the first one corresponding to the first sprint (December) and the second one for the final deadline of the project (January).
- To register the different backlog elements you have to use `GitLab` ***issues*** (do not forget to create the corresponding label if needed). **All backlog elements must be assigned to a milestone**.
  - **Features** must be added with label *feature*
  - **Stories** must be added with label *user-story*
  - **Bugs** must be added with label *bug*
  - **Work-items** must be added with label *work-item*
- All backlog elements must also have a **priority label** (*high*, *medium*, *low*) (do not forget to create them label if needed)
- **Planning and review meetings** must be registered as an issue (open and closed inmediately) with label *meeting*

2.7 Graphical User Interface

The application **must include a *graphical user interface* (GUI)** to interact with the user. It is recommended to use the classes of *Java Swing* graphical library to develop the GUI of the application, however, other libraries like *JavaFX* can be used.

3 Introduction

This document describes the requirements of a program for playing *Sokoban*, a jigsaw puzzle whose goal is to move boxes from their origninal place to a goal place in a small warehouse. *Sokoban* is a Japanese game created by Hiroyuki Imabayashi in the 1980's decade, and the word *Sokoban* means *warehouse man*. There exits hundred of versions of this game and thousands of levels. Figure 1 shows some screenshots of the game which can be played at `http://www.abelmartin.com/rj/sokobanJS/sokoban.html` and `https://sokoban.info`.

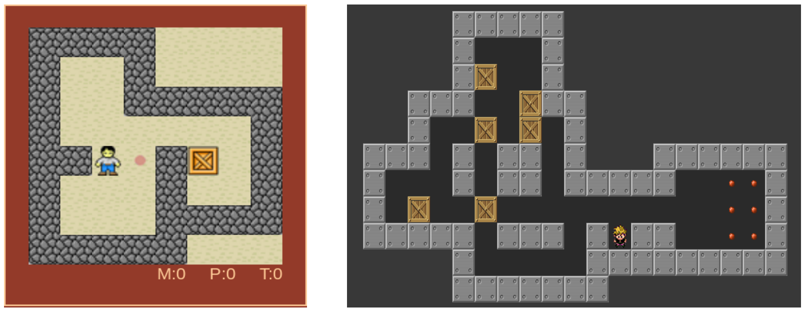

Figure 1: Screenshots of online implementations of Sokoban

4 Requirements

The following list summarizes the requirements of the system that must be implemented:

1. The game is composed by several levels and each level has its own board.
2. The board is divided in squares, such that each square contains one of the following items:
   - Wall
   - One Warehouse Man
   - Box
   - Goal position
3. Two or more consecutive walls should seem as if they are one one piece.
4. The warehouse man can only move horizontally or vertically.
5. Walls cannot be traversed.
6. The warehouse man can move boxes pushing them horizontally or vertically. A pushed box can move to the next position when this position is free or it is a goal position.
7. When a box is placed in a goal position, the color of the box must change to show that it in a goal position.
8. A goal position only can have one box.
9. A level is completed when all boxes are in a goal positon. The application must detect it automatically.
10. The different levels are read from files with name `level_N.txt`, where N is a positive number. The game starts with the first level, that ir, file `level_1.txt`. When the user passes a level, the program will try to read the next level. Level numbers must be consecutive.
11. The program must include at least 3 different levels with different boards.
12. Level files must be plain text files which adhere the following format:
    - The first line contains the *name* of the level.
    - The second line contains two numbers which corresponds to the dimensions, `nRows` and `nColumns`, of the board
    - The following `nRows` lines contains the elements of the board. Each row must contain `nColums` with the following elements:

`+` : Wall
`.` : Empty square
`*` : Goal Position
`#` : Box
`W` : Warehouse man

An level example file with this format is:

```
Initial level
8 8
++++
+  +
+  +++++
+      +
++W*+# +
+   +  +
+   ++++
+++++
```

13. Levels must be correct: (1) there are at least one box and one goal position; (2) there is only one warehouse man; and, (3) the board contains the same number of boxes and goal positions. If the level is not correct, the application must spot it, showing an error message and trying to read the next level.

14. The punctuation of the level is the number of movements done by the warehouse man from the beginning of the level.

15. The punctuation of the game is the addition of the punctuation of all levels.

16. When the last level, the one whose consecutive number is not found, is passed, the program must show a congratulations message with its punctuation (number of movements).

17. The game screen must show: (1) the board; (2) the level name; (3) the punctuation of the level; (4) the punctutation of the game.

18. All movements in the current level can be undone, thus, the application must record all the movements done since the beginning of the level.

19. A game can be saved and loaded, that is, the application should be able to save in a file selected by the user the information needed to restore a game: (1) the current level and its current board, (2) the movements done in the current level, (3) the punctuation of the level, (4) the punctuation of the game.

20. The game menu of the application must allow the user to:
    - Start a new game (resest the game and starts from the first level)
    - Restart the current level
    - Undo the last movement
    - Save the game
    - Open a saved game
    - Close the application

References

[1] `https://maven.apache.org/`

[2] `https://www.oracle.com/technetwork/java/javase/tech/index-137868.html`

[3] `https://git-scm.com/`

[4] `https://www.sonarqube.org/`