MSc. Computer Science for Communication Networks

Advanced Software Engineering for Smart Devices

15-puzzle

Authors:
Oscar GUILLEN
Patricia REINOSO

Adviser:
Paul GIBSON

February 9th, 2018

Contents

15-puzzle

0.1 Introduction

The 15-puzzle is a puzzle game that consists of a board with numbered squared sliding tiles with the objective of placing them all in order. In this report we present the implementation of this game as an application that runs on smart devices (Android).

The project was developed for the Advanced Software Engineering course, in a team of two people, following the best practices of software development. Analysis and specification, design, implementation and testing were the stages of the software development life cycle covered.

Three advanced software techniques were addressed in this project: device programming, artificial intelligences and web services. The details about the implementation and the decision making of the project are found in this report.

0.2 Specification and Design

The specification follow the project description in [3].

- User flow chart: describes the logic of th game from the point of view of the user. It helps to define the activities and the front-end of the application.
- Class diagram: describes the structure of of the model, which manages the logic of the application.

0.3 Implementation

The application was developed in Java and Android Studio. We try to follow good habit of software engineering and the following techniques were taken into account.

- Exception handling: to face abnormal situations. It was especially implemented in the models. *InvalidMovementException* and *InvariantBrokenException* are exceptions added to the program.
- Generics: used for the tiles. It allows code reusability, which means that an implementation of the game with tiles based on other than integers (ex: letters) may be created more easily.
- Accessors: used in combination of private attribute for the classes. This allows the encapsulation of the behavior, hides the internal representation of the class, grants difference access levels and improves interoperability.
- Invariant: a method that verifies whether the state of an object is valid. It was implemented in the *Board* and *IntTile* classes. They check properties such as the size of the board and the values of the tiles. However, the invariants created are not comprehensive and can be improved.
- Code reuse: *InvariantBrokenException* was taken from previous software engineering module. It was developed by J Paul Gibson. In addition, the code for the web server is based on the one we developed for the *Middleware for Distributed Application* course.

However, most these are included in a part of the code and not extensively.

0.4 Testing

Some Unit Testing was implemented on the classes Board and IntTile in order test the individual functionalities of the methods on these classes. This allowed to fixed some detailed immediately and remark other issues.

The web server is tested at the time of being compiled. In addition, it can be tested using the existing client.

However the test cases created are not extensive and a more complete test suite can be generated.

0.5 Tools

Several tool were used during this project. Among them:

- UMLet: an open-source tool for creating UML diagrams. Used for creating the diagrams. [2]

- Java: programming language.
- JUnit: to perform unit test using JUnit4 library.
- JavaDocs: comments about classes and method are included with JavaDoc format which allows to generate documentation for the source code.
- Android Studio: the official IDE for Android, was used to code the front-end of the application. [1]. Version 3.0.1 was used.
- GitHub: the git version control repository hosting service, was used for version control of this project. The url of the repository can be found in [4].

0.6 Advanced Techniques

0.6.1 Device Programming

The employment of this technique consisted on coding the front-end of the application, which enables to run it on a smart Android device. It was necessary to learn Android Studio in order do it.

Basically, the front-end is built based on a grid view (the board) that allows to place the tiles. It follows the specification on the user flow chart. First, a menu screen is presented, the player can choose whether to use show the instructions or to start playing.

The application lets the player move the tiles as whished, then a solution to the board is request to a web server, and the solution is "played" to user.

The graphic interface is very simple and can definitely be improved. Also, it should benefit of the model for handling the behavior of the board and tiles. Finally it was not possible to play the solution received to player, the application takes some time to get the board the solved state but there is no animation.

0.6.2 Artificial Intelligence

The artificial intelligence was included through the A* algorithm to solve the puzzle. This is graph algorithm that allows to find paths between nodes. This algorithm uses heuristics to estimate the cost to get to the solution (a solved board). It was adapted to fit the needs of our problem.

For our algorithm, a tree is created, where the root is the puzzle to solved. Then, children nodes are included according to the next possible movement on the board. The computation of a fitness function (based on the heuristics) allows to decide the next node to take into account for the path to the solution. The node with the smallest fitness value is selected and its children and included into the tree only if the it is not previously found in the tree. This procedure continues until the puzzle is solved.

Three fitness functions were considered:

- Manhattan distance: sum of the vertical and horizontal distances of each tile to their solved position.
- Disorder: count of the tiles that are located in the wrong position.
- Linear: sum of distance of each tile and its goal position.

All the fitness functions were normalized to obtained valued between 0 and 1. Where 0 is the solved stated and 1 is the board the 'most mixed-up.

An analysis of the performance of the solutions was done.

0.6.3 Web Services

A REST service was created. It is in charge of solving puzzles, for this, it receives POST request with a string that represents a state of the board. Then, the intelligent algorithm is applied and the solving solution is sent back to the client.

There is GET request that allows to check whether the server is active, and it is also possible to choose which fitness function to use for the solution of the board.

0.7 Team Contribution Breakdown

Important decision-making tasks such as the specification, basic design, heuristics to used were discussed among the member of the teams. The performance evaluation was a shared task. However the actual implementation of the functionalities was distributed as follows:

- Oscar Guillen: A* algorithm and the related model classes, web server implementation and Android Studio web server connection, performance evaluation of the solution.
- Patricia Reinoso: specification-writing, model implementation of the classes related to the board and tiles, front-end implementation using Android Studio, unit testing on the model, javadoc.

0.8 Future Works

We can add, that some advanced tools can be applied to improve this implementation. For this we have the following possibilities:

- Aspects: You can add aspects to perform debugging and profiling on the implementation. Both tools can be used to improve existing bugs.
- Parallelism: You can add an implementation based on multiprocessing. This can be added on the analysis of the generated tree. In which each process is given an independent part of the tree seeking to finish first than others and then be able to communicate with the other processes in order to make a more efficient calculation. This can be quite useful for bigger puzzle.

0.9 Conclusions

The previous knowledge in Java programming was an advantage during the development of the project. Nevertheless, Android Studio was a tool completely new for us and learning it took some time. In addition, this tool raised some problems because it worked very slowly in our computers which made the task not very pleasant.

Software engineering best practices and techniques were used during the development of the project. However, most of them were not used extensively which is an improvement point.

There is a big limitation in the graphical interface of the game, we did not succeed to make it work properly (no animation is shown when the application receives the solving sequence). Also the front-end should use the model representation of the board and tiles.

More unit testing is needed in order to guarantee a better quality of the solution. A TDD approach would have been appropriate for this project. It would have reduced many problems at the moment of introducing changes in the code.

The creation of interfaces for the model classes would provide a better documentation to use them.

Bibliography

[1] Android Studio. https://developer.android.com/studio/index.html.

[2] UMLet. http://www.umlet.com.

[3] J. P. Gibson. 15-puzzle. Project Description. http://www-public.tem-tsp.eu/~gibson/Teaching/CSC7336/CSC7336-Project-2017-18.pdf.

[4] O. Guillén and P. Reinoso. 15-puzzle. Git Hub Repository. https://github.com/patriciareinoso/15puzzle.