Object-Oriented Programming

Programming Project

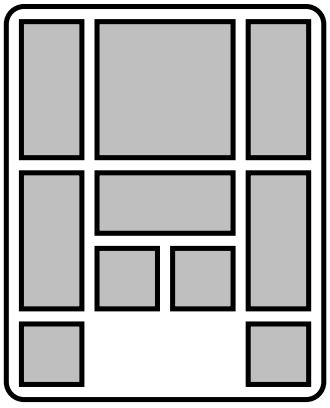
Academic Year 2024-2025

This project can be completed individually or by pairs of students, and must be submitted by email to `bvergain@uliege.be` *for April 17th at the latest. Projects returned after the deadline will not be graded. Except for classes present in the Java Class Library which you are allowed to use, and the graphical library provided in the scope of this project, all the code that you submit must be written by yourselves. Cases of fraud will be prosecuted according to the policy of ULiège (cf.* `https://www.student.uliege.be/cms/c_11187649/`*). This include copying or adapting code obtained from other students, from online sources, or generated by AI tools.*

Sliding puzzles

A *sliding puzzle* is a brain teaser game played on a rectangular board, with some number of rectangular pieces that can be slid horizontally or vertically. Initially, those pieces are placed in a specific configuration. Then, they are moved one at a time by sliding them towards a free area on the board. The goal of the puzzle is to reach a configuration in which one or several key pieces have been moved to designated target positions.

For instance, the following figure shows the *Klotski* puzzle, whose goal is for the largest square piece to reach the bottommost center position of the board.



Subject of the project

The project consists in writing a Java program that makes it possible to play a sliding puzzle described by a specification file, the name of which is provided as a command-line argument. Upon being started, this program must read the specification file, display the initial configuration of the puzzle, and let the user play until he or she either manages to reach the goal, or decides to close the game. The program must be able to detect when the goal is reached, and to signal it by changing visibly the color of the pieces that have reached their target position.

The format of the specification file is imposed and described in the next section. A graphical library is provided on the web page of the course, and must be used for implementing the graphical user interface of the program. Its documentation is given after that of the specification file.

Format of the specification file

The specification file is a text file in which line separators are either line feeds (`"\n"`) or carriage returns followed by line feeds (`"\r\n"`). It contains a description of the initial configuration of the puzzle, followed by a description of the goal.

The description of the initial configuration contains, in that order:

- a line `"ncols nrows npieces"` specifying the size (`ncols`, `nrows`) of the board, expressed as a number of columns `ncols` and a number of rows `nrows`, followed by the number `npieces` of pieces of the puzzle. The parameters `ncols`, `nrows` and `npieces` must be strictly positive decimal integer numbers.
- `npieces` lines of the form `"width height xpos ypos"`, each describing one separate piece of the puzzle. The values `width` and `height` specify (respectively) the number of columns and rows spanned by the piece, and `xpos` and `ypos` give its initial position, corresponding to the number of (respectively) the first column and the first row occupied by the piece. Rows and columns are numbered from 1, starting from the upper left corner of the board.

The description of the goal contains, in that order:

- a line `"ngoals"` specifying the number of pieces that must reach a specific position to solve the puzzle.
- `ngoals` lines of the form `"nb xpos ypos"` specifying that the piece numbered `"nb"` must reach the position expressed by `xpos` and `ypos`. The pieces are numbered starting from 1, in the same order as in the description of the initial configuration, and the positions are expressed in the same way as in this description.

For example, the Klotski puzzle is described by the following specification file:

```
4 5 10
1 2 1 1
2 2 2 1
1 2 4 1
1 2 1 3
2 1 2 3
1 2 4 3
1 1 2 4
1 1 3 4
1 1 1 5
1 1 4 5
1
2 2 4
```

Graphical library

The library that must be used for implementing the graphical user interface of this project provides a class `be.uliege.montefiore.oop.SlidingPuzzleGUI` with the following interface:

- `public SlidingPuzzleGUI(int w, int h) throws GUIException`: Creates a new window for displaying the puzzle, with a width of `w` pixels and a height of `h` pixels. An exception is triggered in the case of an invalid dimension.

- `public void startFrame()`: Starts a new frame, in which a list of rectangles to be displayed will be provided one by one by invoking `newRectangle()`. The contents of the window will be replaced by this list of rectangles only when the frame is completed by calling `endFrame()`.
- `public void newRectangle(int x, int y, int w, int h, int r, int g, int b) throws GUIException`: Adds a new rectangle to the current frame, with an upper left corner at $(\texttt{x}, \texttt{y})$, a width equal to `w`, a height equal to `h`, and filled with the color $(\texttt{r}, \texttt{g}, \texttt{b})$. The parameters `x`, `y`, `w`, `h` are expressed in pixels, in a coordinate system whose origin $(0, 0)$ is the upper left corner of the window. The parameters `r`, `g`, `b` correspond respectively to the red, green and blue components of the color, and must have a value between 0 and 255. An exception is triggered in the case of an invalid parameter value, or if the frame has not been properly started.
- `public void endFrame()`: Ends the current frame and displays its contents.
- `public int[] nextMove()`: Waits for the user to interact with the game by dragging the mouse on the board, or by closing the window. In the former case, returns an array `[x0, y0, x, y]` containing the initial and final coordinates $(\texttt{x0}, \texttt{y0})$ and $(\texttt{x}, \texttt{y})$ of the mouse movement, expressed in pixels. In the latter case, the window disappears and the method returns the null pointer.

Important guidelines

- The main class of your project must be named `SlidingPuzzle`. Your program must expect to be invoked with a single argument containing the name of the specification file. In the case of an error in this specification file, the program should immediately exit with a message describing the nature of the error, for instance `"Error at line 7: invalid position"`. In particular, the program must be able to detect overlapping pieces, pieces that are not entirely contained in the board, and the absence of any free space on the board. If the specification file is correct, the program should start the game, let the user play, and exit when the window is closed.

  Examples of correct specification files are provided on the web page of the course. You are strongly encouraged to check that your program works as expected on those particular examples.

- The user interface of your program must be implemented with the library `sliding-puzzle-gui` provided in the scope of this project. You cannot, in particular, use JavaFX, Swing and/or AWT. The windows created by the program must not exceed 1600 by 900 pixels.

- Your submission must take the form of a `.zip` or `.tar.gz` archive containing the Java source code of your project as well as a copy of the graphical library file `sliding-puzzle-gui.jar`. The name of this archive must be formed by the student IDs of the members of your team separated by a dash symbol, for instance `"s2312345-s2054321.zip"`. This file must be attached to an email bearing the subject `"OOP project submission"`.

- After extracting the contents of your submitted archive in the current directory, it should be possible to compile your project on a Linux system with the command

  ```
  javac -cp .:sliding-puzzle-gui.jar be/uliege/montefiore/oop/*.java
  ```

  (Note that if you develop under another operating system, or within an IDE, the syntax of this command can slightly differ.)

- Your project will be evaluated with Java 8. It is thus important to make sure that your submission does not rely on features introduced in later versions of the Java language.

- The evaluation will take into account not only the correct implementation of the requirements of this problem statement, but also the compliance of your source code to the principles of object-oriented programming, as well as its modularity, efficiency, readability, simplicity, and portability.