Object-Oriented Programming

Project Statement

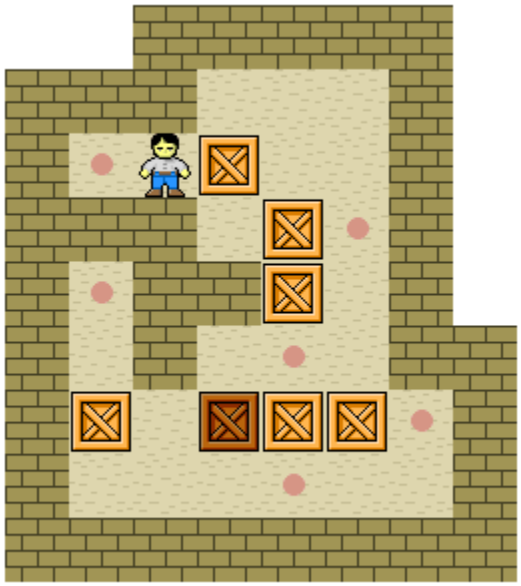
Academic Year 2021-2022

This project can be completed by groups of up to two students and must be submitted to `bvergain@uliege.be` *for April 24th at the latest. Projects returned after the deadline will not be corrected. Plagiarism is not tolerated, in line with the policy of the university (*`https://matheo.uliege.be/page/plagiat`*).*

The goal of this project is to program in Java a Sokoban game. Sokoban is a puzzle game that is played on a 2D grid of square cells. These cells can either be empty, or contain

- the (unique) *character* controled by the player, that can be moved up, down, left or right,
- a piece of *wall*, representing an obstacle that cannot be moved and cannot be passed,
- a *box*, that can be pushed by the player to an adjacent cell if this destination cell is empty, or
- a *target* for a box, which is a specially marked empty cell.

There are as many targets as there are boxes. The goal of the game is to move all the boxes to a target location.



(Source: Wikipedia. The targets are represented by pink dots. The dark box indicates that this box is already located on a target.)

In your implementation, you can freely choose the layout of the game. Your program must be able to detect when the game is solved, i.e., when every box has been placed on a target. In this situation, you can either indicate that the game is finished by removing the player from the grid, which leaves to the user the only option of quitting the game by closing the window, or start a new game with a different layout.

In addition to the elements defined in this statement, you are free to add you own objects as well as additional types of cells and mechanisms to the game. For instance, you can implement cells that can only be passed in one direction, doors that can only be opened after having collected a key, obstacles that appear and disappear periodically, ... Be creative and have fun! But do not get carried away too much, the goal is to keep the project simple.

Graphical user interface:

A library `sokoban-gui.jar` will be provided in order to simplify the design of a graphical user interface for the game. This library contains the class `SokobanGUI`, that offers the following interface:

- `public SokobanGUI(int w, int h) throws SokobanError`: Creates a new Sokoban window of dimension `w`×`h` (expressed in numbers of cells). This window is initially empty and invisible.
- `public int loadImage(String filename) throws SokobanError`: Prior to starting the game, this method should be called once for each potential type of cell content. It loads from the file named `filename` an JPG or PNG image to be displayed in cells. The methods return a number that can be later used as a parameter to the `setCell` operation.

  Notes:

  - Only the first 32x32 pixels from the top-left corner of the image are taken into account.
  - An archive containing sample images for the player, walls, boxes, targets and empty cells will be provided.
- `public void setCell(int x, int y, int type) throws SokobanError`: Modifies the contents of the cell at location `x, y` with `type`. The values of `x` and `y` start at 0, and respectively increase from left to right and from top to bottom. The value of `type` must correspond to an integer previously returned by `loadImage`.
- `public void show() throws SokobanError`: Makes the window visible, as well as the last modifications applied to its content since this method has been called.
- `public int getEvent() throws SokobanError`: Waits for the user to perform an action, and returns this action. The values that can be returned are:
  - `SokobanGUI.QUIT`: The player has terminated the game. The program should exit.
  - `SokobanGUI.UP`: The player wants to move up.
  - `SokobanGUI.DOWN`: The player wants to move down.
  - `SokobanGUI.LEFT`: The player wants to move left.
  - `SokobanGUI.RIGHT`: The player wants to move right.

  Note: The move actions are triggered by pressing an arrow key on the keyboard, or one of the keys "I", "J", "K" or "L".

In the case of an error, the exception `SokobanError` is thrown, containing a message describing the nature of the problem.

Important guidelines:

- The first game layout to be displayed by your program should be simple and immediate to solve, in order to be able to easily check that your program correctly detects end-of-game configurations.
- The GUI library belongs to the package `be.uliege.boigelot.oop.sokoban.gui`, and your own code must be placed in a package `be.uliege.boigelot.oop.sokoban.main`.
- Your submission must take the form of a `.zip` or `.tar.gz` archive containing the Java source code of your project. After extracting the contents of this archive in the current directory, it should be possible to compile your project with the command

  `javac -cp .:sokoban-gui.jar be/uliege/boigelot/oop/sokoban/main/Sokoban.java`,

  and to run it with

  `java -cp .:sokoban-gui.jar be/uliege/boigelot/oop/sokoban/main/Sokoban`
- You cannot modify the provided library `sokoban-gui.jar`, and must not include it in your archive.
- The name of your archive should contain the name of the members of your group, in the form `firstName1-lastName1--firstName2-lastName2.zip` (or `.tar.gz`).
- Your archive should also contain a file `readme.txt` containing the list of all features implemented in your game.
- 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 principes of object-oriented programming, as well as its modularity, readability, simplicity, and portability.

(Submissions that do not strictly adhere to those guidelines will not be evaluated.)