Category Partition and Boundary Value Analysis

October 11th, 2023
Ana Paiva, José Campos

In this recitation class, we are going to explore ‘Category-Partition’ and ‘Boundary Value Analysis’, *two black-box testing techniques*, in the `jpacman` project.

Please make sure your machine is configured properly, i.e.:

- Java installed on your machine and available through the command line. Disclaimer: this tutorial has been validated under Java-11. It may or may not work on other versions of Java. Let us know whether it does not work under Java-X, where X is a version higher than 11.
- Apache Maven to be installed on your machine and available through the command line. In case Maven is not installed, please follow the following steps:
    - Download `apache-maven-3.9.4-bin.zip`
    - Extract `apache-maven-3.9.4-bin.zip`
    - On Windows, augment your environment variables with the full path to the `<extracted directory>/bin`. On Linux/MacOS, run `export PATH="<extracted directory>/bin:$PATH"`. (You might have to run the `export` everytime you restart the computer. For a more permanent solution, please consider adding that command to your bash profile.)

1. Perform ‘Category-Partition’ and ‘Boundary Value Analysis’

Given the “specification” of the `jpacman` project, which you could find in here, we expect you to perform ‘Category-Partition’ and ‘Boundary Value Analysis’ on the following 5 functions. (Offline access to the “specification” can be found in here.)

In a nutshell, apply Category-Partition and Boundary Value Analysis as described in the lecture classes to the input/outputs of the functions described below. Then, write down in a txt file (1) the partitions you managed to find according to the Category-Partition’s procedure and (2) the inputs you are going to use to test each partition according to the Boundary Value Analysis’ procedure.

1.1 `squareAt` in the `nl.tudelft.jpacman.board.Board` class.

```
Unset
public Square squareAt(int x, int y)

Returns the square at the given x,y position. Precondition: The (x, y)
coordinates are within the width and height of the board.

Parameters:
x - The x position (column) of the requested square.
y - The y position (row) of the requested square.

Returns:
The square at the given x,y position (never null).
```

1.2 `createBoard` in the `nl.tudelft.jpacman.board.BoardFactory` class.

```
Unset
public Board createBoard(Square[][] grid)

Creates a new board from a grid of cells and connects it.

Parameters:
grid - The square grid of cells, in which grid[x][y] corresponds to the
square at position x,y.

Returns:
A new board, wrapping a grid of connected cells.
```

1.3 `createLevel` in the `nl.tudelft.jpacman.level.LevelFactory` class.

```
Unset
public Level createLevel(Board board, List<Ghost> ghosts,
List<Square> startPositions)
```

```
Creates a new level from the provided data.

Parameters:
board - The board with all ghosts and pellets occupying their squares.
ghosts - A list of all ghosts on the board.
startPositions - A list of squares from which players may start the
game.

Returns:
A new level for the board.
```

1.4 `makeGhostSquare` in the `nl.tudelft.jpacman.level.MapParser` class.

```
Unset
protected Square makeGhostSquare(List<Ghost> ghosts, Ghost ghost)

creates a Square with the specified ghost on it and appends the placed
ghost into the ghost list.

Parameters:
ghosts - all the ghosts in the level so far, the new ghost will be
appended
ghost - the newly created ghost to be placed

Returns:
a square with the ghost on it.
```

1.5 `draw` in the `nl.tudelft.jpacman.sprite.ImageSprite` class.

```
Unset
public void draw(Graphics graphics, int x, int y, int width, int
height)

Draws the sprite on the provided graphics context.
```

```
Specified by:
draw in interface nl.tudelft.jpacman.sprite.Sprite

Parameters:
graphics - The graphics context to draw.
x - The destination x coordinate to start drawing.
y - The destination y coordinate to start drawing.
width - The width of the destination draw area.
height - The height of the destination draw area.
```

2. Exercise: write unit tests

Take the jpacman project you configured/used in the previous recitation class and write unit test cases using the JUnit framework to every single combination/input you found in section 1 of this sheet. Note: in maven projects, tests must be developed under `src/test/java`.

You may run `mvn compile test` on the command line to run all tests in the project.

3. What should you submit/deliver?

Zip the project's directory and submit it here (M.EIC's moodle) or here (MESW's moodle).

Deadline: October 11, 2023, 11:59:00 pm.
Grades: available on October 23, 2023.

Miscellaneous

- Guide to Configuring Maven Plug-ins
- JUnit framework
- Learn how to write unit tests
- JUnit 5 User Guide
- Parameterized Tests and JUnit 5 Tutorial: Writing Parameterized Tests