Mazedoku

Due

12AM Monday, 10 September 2018

Task description

In this assignment you will be designing a maze game.

You will have a maze board and a player moving around the board.

The player can step left, right, up or down.

However, you need to complete the maze within a given number of steps.

As in any maze, there are walls that you cannot move through.

If you try to move through a wall, you lose a life.

There is also gold on the board that you can collect if you move ontop of it.

The game can end in one of the following ways:

- The player has completed the maze. That is, they have reached the destination.
- The player has no lives.
- The player has no remaining steps.

Please implement the methods provided, as some of the marking is making sure that these methods work as specified.

You are also allowed to write your own methods.

Please also read the entire specification - it is quite comprehensive.

Implementation details

Command line arguments

The MazeGame takes 1 command line argument.

This is the file that we are going to be reading in the game configuration from.

For example:

```
> java MazeGame game_configuration.txt
```

The game configuration file

The game configuration contains 5 pieces of information.

- The number of lives remaining.
- The number of steps remaining.
- The players starting amount of gold.
- The number of rows on the board. [Must be at least 1.]
- The maze board itself.

The layout of the game configuration file is as follows:

```
<number of lives> <number of steps> <amount of gold> <number of rows on the board>
<BOARD>
```

IMPORTANT:

You may assume that if the configuration file exists, that it is in the format given above.

Characters on the maze board

- a `' '` is a space in the board.
- a `'#'` is a wall of the maze.
- a number `0-9` is the gold that you can collect on the board.
- a `'.'` is a place the player has been.
- the `'@'` is the destination that the player is trying to reach. Note: there is exactly one destination.
- the `'&'` is the person. Note: there is exactly one person.
- You may assume that there are no invalid characters on the board.

Here are some sample game configuration files.

easy_board.txt

```
3 4 0 3
######
#@ &2#
```

```
##  ##
```

DEFAULT

```
3 20 0 4
#@ ##  &4#
##  # ## #
###  3#
#######  #
```

Please note: the board is a grid.

This means, it has the same number of columns on each row.

Running the program

The program can be run in 2 ways.

1. Running the program by reading in from an input game configuration file.

```
> java MazeGame input_file_name.txt
```

2. Running the program by using the DEFAULT board provided above.

```
> java MazeGame DEFAULT
```

Error handling

If there is an error initialising the board / program, print out the appropriate error message and exit the program.

If no arguments are given:

```
> java MazeGame
Error: Too few arguments given. Expected 1 argument, found 0.
Usage: MazeGame [<game configuration file>|DEFAULT]
```

If too many arguments are given:

```
> java MazeGame file1.txt file2.txt
Error: Too many arguments given. Expected 1 argument, found 2.
Usage: MazeGame [<game configuration file>|DEFAULT]

> java MazeGame file1.txt file2.txt file3.txt file4.txt
Error: Too many arguments given. Expected 1 argument, found 4.
Usage: MazeGame [<game configuration file>|DEFAULT]
```

The file could not be found, read, etc.

```
> java MazeGame invalid_file_here.txt
Error: Could not load the game configuration from 'invalid_file_here.txt'.
```

List of commands

Once the board is loaded, you can use the following commands in your program.

You need to implement the functionality for all of them.

- **help** Print the help message.
- **board** Print the current board.
- **status** Print the current status.
- **left** Move the player 1 square to the left.
- **right** Move the player 1 square to the right.
- **up** Move the player 1 square up.
- **down** Move the player 1 square down.
- **save <file>** Save the current game configuration to the given file.

Reading and running commands

Reading in commands is case insensitive. If a line is entered and the line is

blank (i.e. only white space), do nothing.

That is the following, for example, are valid commands:

- help
- hElP
- SAVE file_name.txt
- UP
- board

The following commands are invalid as they have either have the wrong number of arguments next to the command, or don't match a given command:

- help up
- up blah 123
- save file1.txt file2.txt
- chocolate
- h e l p
- save

Example with invalid command.

```
> java MazeGame DEFAULT
```

```
help up
Error: Could not find command 'help up'.
To find the list of valid commands, please type 'help'.
up blah 123
Error: Could not find command 'up blah 123'.
To find the list of valid commands, please type 'help'.
```

Basic commands: Help, board and status

Help command

Prints out the following help message. Example below

```
> java MazeGame DEFAULT
help
Usage: You can type one of the following commands.
help         Print this help message.
board        Print the current board.
status       Print the current status.
left         Move the player 1 square to the left.
right        Move the player 1 square to the right.
up           Move the player 1 square up.
down         Move the player 1 square down.
save <file>  Save the current game configuration to the given file.
```

Board command

Prints out the current board. Example below.

```
> java MazeGame DEFAULT
board
#@ ##  &4#
##  # ## #
###  3#
#######  #
```

Status command

Prints out the current status.

```
> java MazeGame DEFAULT
status
Number of live(s): 3
Number of step(s) remaining: 20
Amount of gold: 0
```

Board Coordinates

The x coordinate is the column number (starting from 0).

The y coordinate is the row number (starting from 0).

For the following file:

easy_board.txt

```
3 4 0 3
######
#@ &2#
##  ##
```

The board coordinates for the easy board can be interpreted by the grid below.

```
              x
     0   1   2   3   4   5
   -------------------------
  0 | # | # | # | # | # | # |
   -------------------------
y 1 | # | @ |   | & | 2 | # |
   -------------------------
  2 | # | # |   |   | # | # |
   -------------------------
```

Interpreting this:

- The player is at (3, 1).
- The destination is at (1, 1).
- There is gold at (4, 1).
- There are 3 spaces, they are at (2, 1), (2, 2), (3, 2).

Movement

There are 4 movement commands.

They are **left**, **right**, **up** and **down**.

Each movement counts as a step.

If the player moves to a valid square (a `' '`, `'.'` or a digit `0-9`), the program

should print out "`Moved to (x, y).`" where x and y are the players new

coordinates on the board.

If the player moves to a square with gold, they get all the gold in that square.

A message is printed out saying "`Plus <number> gold.`"

When they move away from the square, the square is changed to a `'.'`.

If the player trys to make an invalid move, that is either:

- move to a square that is a wall `'#'`,
- move out of the boundary of the board

The player to loses a life, and it still counts as a step.

It should also print out "`Invalid move. One life lost.`"

Example 1. Failed due to losing all their lives.

DEFAULT game configuration

```
3 20 0 4
#@ ##  &4#
##  # ## #
###  3#
#######  #
```

```
> java MazeGame DEFAULT
board
#@ ##  &4#
##  # ## #
###  3#
```

Example 2. Success.

easy_board.txt game configuration

```
3 4 0 3
######
#@ &2#
##  ##
```

```
> java MazeGame easy_board.txt
right
Moved to (4, 1).
Plus 2 gold.
left
Moved to (3, 1).
left
Moved to (2, 1).
board
######
#@&..#
##  ##
left
Moved to (1, 1).
Congratulations! You completed the maze!
Your final status is:
Number of live(s): 3
Number of step(s) remaining: 0
Amount of gold: 2
```

Example 3. Failed due to having no more steps.

easy_board.txt game configuration

```
3 4 0 3
######
#@ &2#
##  ##
```

```
> java MazeGame easy_board.txt
right
Moved to (4, 1).
Plus 2 gold.
left
Moved to (3, 1).
down
Moved to (3, 2).
board
######
#@ ..#
## &##
status
Number of live(s): 3
Number of step(s) remaining: 1
Amount of gold: 2
left
Moved to (2, 2).
Oh no! You have no steps left.
Better luck next time!
```

Example 4. Failed due to having no more lives and no more steps.

easy_board.txt game configuration

```
3 4 0 3
######
#@ &2#
##  ##
```

```
> java MazeGame easy_board.txt
down
Moved to (3, 2).
status
Number of live(s): 3
Number of step(s) remaining: 3
Amount of gold: 0
down
```

```
Invalid move. One life lost.
status
Number of live(s): 2
Number of step(s) remaining: 2
Amount of gold: 0
down
Invalid move. One life lost.
status
Number of live(s): 1
Number of step(s) remaining: 1
Amount of gold: 0
down
Invalid move. One life lost.
Oh no! You have no lives and no steps left.
Better luck next time!
```

Saving the game

To save the game, we write out the current game configuration to a file.

If there is an error saving the game configuration to the given file, print out the error message as shown below.

That is, if the file does not exist and it cannot be created, or if it is not writable etc.

easy_board.txt game configuration

```
3 4 0 3
######
#@ &2#
##  ##
```

```
> java MazeGame easy_board.txt
right
Moved to (4, 1).
Plus 2 gold.
status
Number of live(s): 3
Number of step(s) remaining: 3
Amount of gold: 2
board
######
#@ .&#
##  ##
save current_board.txt
Successfully saved the current game configuration to 'current_board.txt'.
left
Moved to (3, 1).
save .
Error: Could not save the current game configuration to '.'.
left
Moved to (2, 1).
left
Moved to (1, 1).
Congratulations! You completed the maze!
Your final status is:
Number of live(s): 3
Number of step(s) remaining: 0
Amount of gold: 2
```

The contents of current_board.txt after the above is:

```
3 3 2 3
######
#@ .&#
##  ##
```

Ending the game without finishing / losing.

If there is no more input from the user. That is, you reach `EOF` in `System.in`

and the game has not ended, print out "`You did not complete the game.`"

For example, if after starting, the user hit CTRL+D, the following would happen:

```
> java MazeGame DEFAULT
You did not complete the game.
```