Guided Project - Iteration #1

"Dungeon Keep"

"*Unfairly captured by the tyrant ruler of your Realm, you have been thrown into a dark, damp and ghastly dungeon, left there to rot into forgetfulness... You keep yelling for the Guard, but he seems oblivious to your calls...You need to escape...wait...but...in the midst of all the fuss that was your capture, they forgot to lock your cell door...there is hope...but the task is not simple...will you have it in you to elude your jailers and make your way back to freedom?*"

Iteration #1

A. Setting up your development environment.

At this section, it is assumed you are using the Eclipse (Oxygen) IDE, with the following plug-ins installed (free, available at the Eclipse Marketplace): Pitclipse 0.31.4, EclEmma Java Code Coverage 2.3.3 and WindowBuilder 1.9.0. This is the configuration installed at the classes computer rooms.

1. The current project is to be developed by groups of 2 students. Upon group formation (check with your practical class teacher), you should be given a group number.
2. Create a new github **private** repository having the following name: "**LPOO1718_T#G$**", where '**#**' is your class ("turma") number and '**$**' is your group number. For instance, group 1 in class 2MIEIC01, should create a repository named "LPOO1718_T1G1". (Note: Select the "Initialize this repository with a README" checkbox).
3. The owner of the repo (the student that has created it) should then add as collaborators, his/hers group colleague and the practical class teacher. (On github, select the "Settings" tab on your repo and then the "Collaborators" option. Search for the usernames of your group colleague and class teacher and add them).
4. Next, you should create your Java project on Eclipse and connect it to your github repository.

   (If you are already proficient with using git/github with Eclipse or any other IDE you prefer, you may skip the next steps)

   1. Open the "Git Repositories" view on Eclipse (Window>Show View>Other... and choose "Git Repositories" inside "Git" folder).
   2. On the "Git Repositories" view, select "Clone a Git repository". On the "URI" textbox, paste the URL from your github repository (e.g. "**https://github.com/<yourgithubusername>/LPOO_T#G$**"). On the "Authentication" section, enter your github credentials.
   3. The branch "master" should appear selected on the "Branch Selection" window, and upon clicking "Next", choose a local directory to clone your github repo into, and click "Finish". On completion, your git repository should appear on the "Git Repositories" view.
   4. Create an Empty Java Project on Eclipse (give it the name you want).
   5. Share the project (Right-click on the name of the project on Package Explorer, and select "Team>Share Project..." on the context menu.

6. On "Configure Git Repository", the repository created at step 4.3 should appear in the dropdown combo-box. Select it and click "Finish".
7. Add a "HelloWorld" class to the project (with a **main** program, printing out "Hello World") and make a first commit. ("Team>Commit..").
8. The "Git Staging" tab should open, and you can add a commit message (something like "first commit"). You should also verify and configure which files you want to include in the commit action ("Staged files") or not ("Unstaged files"). *Note: At the first commit, as you have no commits yet, Eclipse won't assume that you want to commit all files. Later on, Eclipse will keep track of which files are modified, added or deleted during development and will give you a more adequate listing of staged files (that you can configure at your discretion).*
9. After selecting which files you wish to commit, you can commit only to your local repository ("Commit" button) or also push to your remote repository on github ("Commit and Push..."). This management is completely up to you.
10. From this point on, you are set for using github to control your source-code and collaborate with your colleague. **Don't forget to go through the git/github documentation available at the moodle's main course page.**

Iteration #1

B. Basic Game Logic.

- **Task #1 [at class].** Let's start with the game map. Store in memory and print out the following map (no colors, textual characters only).

X	X	X	X	X	X	X	X	X	X
X	H			I		X		G	X
X	X	X		X	X	X			X
X		I		I		X			X
X	X	X		X	X	X			X
I									X
I									X
X	X	X		X	X	X	X		X
X		I		I		X	k		X
X	X	X	X	X	X	X	X	X	X

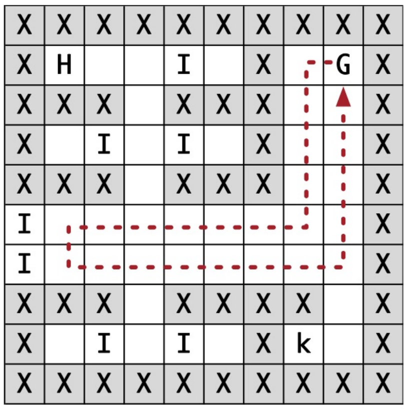
```
Legend:

X - Wall
I - Door
H - Hero
G - Guard
k - lever
empty cell - free space.
```

- **Task #2 [at class].** The user should be asked to enter single character commands in order to move the main character ("hero") in 4 possible directions (up, down, left, right). If there is a wall or door towards that direction, the hero remains at the same position. Upon entering each command, the program should update the game and reprint the game map.

- **Task #3 [at class].** Add the following game logic: The hero needs to get to the "lever" ('k') that opens the dungeon exit doors (contiguous doors on the left wall, that should change from 'I' to 'S', the moment the hero steps over the lever). If the hero reaches anywhere near the guard (any adjacent square, except diagonals), it is immediately captured and the game is over. At this point, the guard is asleep, so the hero should be safe, as long as it keeps away from him. Reaching (stepping over) the open exit doors, the game ends with victory.

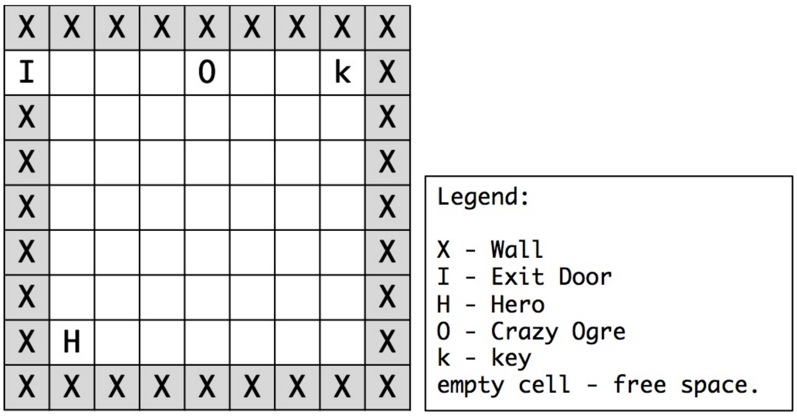
- **Task #4 [at class]**. The Guard is now awake and patrolling the dungeon. He undertakes a pre-defined route across the main corridor, according to the schematic below:



X	X	X	X	X	X	X	X	X	X
X	H			I		X		G	X
X	X	X		X	X	X			X
X		I		I		X			X
X	X	X		X	X	X			X
I									X
I									X
X	X	X		X	X	X	X		X
X		I		I		X	k		X
X	X	X	X	X	X	X	X	X	X

The Guard moves after the hero has moved. He keeps patrolling indefinitely. *#DEV HINT: Store the fixed set of movement commands for the Guard and re-use the hero movement code. Remember: The Guard starts the game always at the same position.*

- **Task #5 [at home]**. *And just when you thought your captivity had ended, you realise you still have another challenge to overcome...go through the Keep's Crazy Ogre.* Extend the game logic, so that when the hero reaches the dungeon exit doors ('S' actually mean "Stairs"), he/she progresses to a new game level, with the following map:



X	X	X	X	X	X	X	X	X
I				0			k	X
X								X
X								X
X								X
X								X
X								X
X	H							X
X	X	X	X	X	X	X	X	X

```
Legend:

X - Wall
I - Exit Door
H - Hero
0 - Crazy Ogre
k - key
empty cell - free space.
```

The hero needs to get the "key" ('k') that unlocks the Keep's exit door (top left corner), while avoiding the "berserking" Crazy Ogre ('0'). The Ogre moves randomly in one of four directions (up, down, left, right). The Ogre captures the Hero similarly to the Guard. Upon reaching the key, the hero picks it up, changing his/hers representation from 'H' to 'K' (Uppercase 'k'). If the Ogre moves into the key position, it changes his representation to '$' (without picking the key, key remains there after Ogre leaves that position, reverting its representation back to '0'). When the hero reaches the exit door with the key, it needs to spend an extra movement (onto the door) to unlock it ('I' turns to 'S') and only then can go through it and end the game.

- **Task #6 [at home].** *The Crazy Ogre is now armed with a massive club.* Extend the Crazy Ogre logic so that every time it moves it also swings its club in a random direction (up, down, left, right), landing on a position adjacent to the Ogre. The club is represented by a '*' (asterisk) and hits the hero (game over!) if it lands on an adjacent position to the hero. If the club lands on the key position, its representation changes to '$' (key remains unharmed).

GENERIC DEVELOPMENT TIPS

- ***Try to keep code duplication at a minimum. Extract code to a new method and re-use it, whenever you feel appropriate.***
- ***Try to keep methods short. If your see that the length of the code in you method goes beyond a page, it is probably a symptom that it is too big. Chunk the code into smaller, self-contained, methods and call them in the original method, to improve the readability of the code. It will be easier to spot bugs and correct them.***

- ***There are good ways of writing, legible, readable and easily browsable code. Here are some code writing style conventions and guidelines that may help to improve how you write your code.***

Última alteração: Terça, 13 Fevereiro 2018, 19:48

ADMINISTRAÇÃO DA PÁGINA/UNIDADE