Guided Project - Iteration #2

Iteration #2

A. Refactoring to objects

Task #1 [at class]. For your programming solution to grow in a sustainable manner, you must now prepare the code for the accommodation of the next features. Therefore, you are going to "refactor" your code, by separating concerns and responsibilities into classes. Therefore, perform the following tasks:

- *Separate user interaction from game logic.* Create two separate *packages*, one containing the class(es) for user input, for example: ***dkeep.cli*** (*command line interface),* and another containing the classes for the game logic, for example: ***dkeep.logic****.* This separation will allow, in the future, to have multiple ways of using/running the game logic code: through the command line interface, through automatic unit tests (to develop at a later iteration), or through a graphical user interface (to develop at a later iteration).
- *User interaction package internals.* The user interaction package should have, at least, one class with the *main* method entry point and possible auxiliary methods for handling input. It is recommended to place the "*game loop/cycle*" in this package. The game "loop" consists of: continuously asking the user to enter a command; running the game logic accordingly (to that command) and updating the game state; printing the game screen; until the game is over (hero dies or wins).
- *Game logic package internals.* This package should have, at least, the following classes (with attributes *private* or *protected*, and non static):
  - A class to represent the state of the game, storing the current map and the game elements (hero, guard, ogre, lever, etc.). It should provide an API (constructors and public methods), to be used by the *user interaction package* to, a least: instantiate a new game, issue a hero's movement action, checking the game state (game over?), getting the map in order to print it (using *toString* might be a possibility);
  - A class to represent the current level map.
  - Several classes to represent the game elements that might be present within the game level (hero, guard, ...) and their status. These should have a super-class with the common properties (coordinates, etc.), exploiting *inheritance* and *polymorphism* accordingly (having the specific behaviour of each game element in its respective class).

Iteration #2

B. Advanced Game Logic

Task #2 [at class]. The Dungeon's Guard Militia is composed of several guards that take turns at patrolling the dungeon. Each guard has its own "personality. There is the "*Rookie*", always alert and scrupulously walking the patrolling path with no mishaps. There is the "*Drunken*", a tormented pour soul that finds in liquor, refuge from its troubled life, and that, while patrolling, might fall asleep (randomly, stops and stays at same position for a while, changing his representation to lowercase "g", and that might reverse his patrolling direction when he wakes up. While asleep, hero can move next to the guard and not loose the game). And finally, there's the "*Suspicious*", an insecure, over-zealous guard, that keeps turning back to

check for something he though he heard (randomly reverses patrolling direction, after a while). Create these strategies for the Guard, ensuring that your code allows for easy adding of other strategies in the future (without the need to modify existing code). Debate with your colleagues or teacher possible solutions for this.

Task #3 [start at class, end at home]. *Oh no! Now, you don't have to slip past a Crazy Ogre, but an entire horde of crazy ogres!!! Luckily for you, there is a spare club next to you, so you pick it up. Beware, foul fiends! You got a basher now!! But wait, a club that size won't kill them, it will merely stun them. Oh well, better than nothing...got to get that key.* There is now the possibility of having more that one Ogre expecting you at the Keep level. A single position on the map can hold multiple ogres, so, they can overlap each other. ***Use a proper JAVA Collection to store and handle the ogres***. The Hero is now armed with a club, so his/hers representation now changes to 'A' (on picking the key, representation changes to 'K', but that does not mean he/she becomes unarmed). Being armed does not kill the Ogres when moving next to them, but will only stun them (they stop moving for two turns, but can still swing their clubs and hit the Hero. Their representation changes to '8', while stunned).

Última alteração: Quarta, 21 Fevereiro 2018, 06:20

ADMINISTRAÇÃO DA PÁGINA/UNIDADE

© 2018 UPdigital - Tecnologias Educativas