"Dragon's Bane" (mini-quest II)

The plot thickens...

"You just tapped your way through a damp corridor, following a draft until a sturdy shut door. The poorly lit doorway reveals a charred knob where a dim light protrudes through an obfuscated keyhole. You try to pry the door open, to no avail. There must be a way to unlock this door. You press onward along the faintly lit walkway, trying not to lose your bearings until heavy breathing halts you...Something or someone is alive in here...with you...the moment you take a turn, the sight before you has you gasping for air, as a surge of primal fear paralyses you...its...its a dragon.

Although snuggled and asleep, nothing would prepare you for such an encounter, as his scales sparkly reflect the dimness of the ambient light, revealing his imposing body, almost blocking the entire passage. But the crackly snap of hopelessness hits you when you spot another shiny object, hanging from his neck: a key. Surely, to the door. Undoubtedly, such an indomitable creature would not surrender his key on a mere request. Maybe you can snatch the key without waking the foul beast...

But you need something to cut the necklace that holds the key...you carefully trace back your steps, and begin burrowing and scavenging throughout the passageways, trying to find something that might have been left behind by some other lost adventurer. Suddenly, you notice another dim sparkling light from a pile of rubble...as you move closer, you feel a foul stench of rotten and putrid flesh...amidst the rubble there is a corpse of a humanoid creature you cannot recognise. The smell is so bad, you barely dare to touch the remains, but the shiny light draws you to push the corpse aside...its...its....a sword!

Praise the Gods! What a holy gift! Not even in your most wild expectations would you expect to find such a valuable item. The sword is sturdy, long and seems robust. Oddly enough it does not seem rusty at all...as if it had always been there for you to find it. Luckily, it appears to be quite sharp...cutting that key off the dragon should be sweet...Hopeful, you hustle back to where the dragon was, careful enough not to reveal your presence, and take a careful look over the corner at the sleeping....wait...where is he??!... Oh no!! ITS AWAKE!!!...

Note: You are supposed to reuse the results from mini-quest I and evolve them.

Task #1. "Refactoring" to Objects.

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, or through a graphical user interface.
- *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, dragon, sword, etc.). It should provide an API (constructors and public methods), to be used by the *user interaction package* to, at 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 map.
  - Several classes to represent the game elements that might be present within the game level (hero, dragon, ...) 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).

Task #2. Advanced Game Logic

a) Change the basic game logic, so that instead of a key, there is a sword ('S'). The hero needs to slay the dragon in order to get the key to open the exit door. When the hero picks the sword, his/hers representation changes from 'H' to 'A' (meaning "Armed"). If the hero reaches any of the dragon's adjacent positions while armed, the dragon dies, otherwise, (unarmed) the hero dies. Therefore the hero needs to pick the sword first, then kill the dragon, and only then he/she is allowed to go through the exit.

b) The dragon is now awake and moves one position in a random direction, every time the hero moves. It must be a valid movement direction (e.g. the dragon must progress through the corridor, and not hit the wall and stay there). As such, when the user enters a moving command for the hero, both hero and dragon move (hero first, dragon next). You should check the game logic (win/lose situation) at every game element move. For example: (1) User enters command, (2) hero moves, (3) game logic checks win/lose status (if hero just moved next to the dragon; is armed or not; is the hero going through the exit; has the dragon been slain, etc...), (4) if game is still not over then move dragon, (5) check game logic again, (6) if game still not over, go back to (1). The dragon cannot pick the sword if he steps over it, but his representation changes to 'F' (while he is on the same position as the sword). Leaving that position, and everything reverts back to normal ('D' for the dragon and 'S' for the sword). The dragon can never go through the exit.

c) There are now multiple dragons, all scattered across the map. All dragons move in the same fashion, and the hero must slay all in order to leave through the exit. You must ask the user how many dragons will there be, before starting the game. *Rule: You must use a proper java collection to solve this issue.*

Specific Tasks per Encampment

These are to be done by the packs that belong to that specific encampment.

- Encampment 2a

d) Create a new moving strategy for the dragon where he can, randomly, fall asleep and stay asleep for a random number of turns. If the dragon is asleep and the hero goes near him, unarmed, he/she does not die. The hero can slay a sleeping dragon. When the dragon awakes, he moves randomly again. When asleep, the dragon representation changes to 'd' (lowercase d).

- Encampment 2b

d) Create a new moving strategy for the dragon where he can "teleport" (jumps to another position, as if there were no walls) to another random valid (free cell, without the hero there) position. Teleportation is triggered by: (1) if the hero is armed and reaches the dragon to slay him, there is 35% change the dragon will teleport to safety (hopefully) or (2) During the normal movement of the dragon, there is a 20% change the dragon will teleport at will.

- Encampment 5

d) Create a new moving strategy for the dragon where he can move diagonally through corners. *Hint: generate the next two valid dragon movements and check if they are "corners". (only four possibilities). If so, move directly to the second position.*

Good luck!

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 you see that your method code length goes beyond a page, then it is probably a symptom that it is too big. Chunk the code into smaller, self-contained methods, to be called by the original method, and thus, improving 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.*

P.S.-
Walkthrough videos

- Package separation (tutorial video - mini-quest #2) :https://www.youtube.com/watch?v=ob-8gabEB_M
- New Game Logic (tutorial video - mini-quest #2): https://www.youtube.com/watch?v=GcybhYNrOo8
- Dragon movement and multiple dragons (tutorial video - mini-quest #2): https://www.youtube.com/watch?v=TfEQHLTe3_E