4CCS1PPA Programming Practice and Applications

Coursework 2: World of Zuul

Michael Kölling, Josh Murphy, Jeffery Raphael
Questions? programming@kcl.ac.uk

Your task is to invent and implement an adventure game. Along with this document, you have been given a simple framework (`zuul-better`) that lets you walk through a few rooms. You can use this as a starting point.

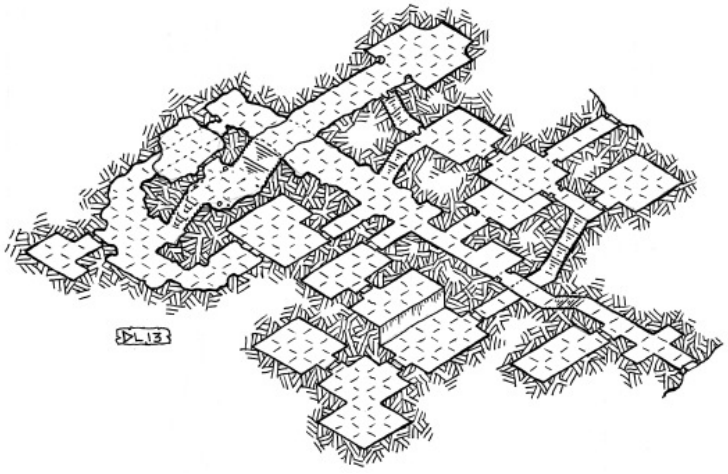
Getting started

The first step is to read the code! Reading code is an important skill that you need to practice. Your first task is to read some of the existing code and try to understand what it does. By the end of the assignment, you will need to understand most of it.

As a little exercise to get warmed up, make some changes to the code. For example:

- change the name of a location to something different.
- change the exits – pick a room that currently is to the west of another room and put it to the north
- add a room (or two, or three, ...)

These and similar exercises should get you familiar with the game.



Designing your game

First, you should decide what the setting, goal, and story of your game is going to be. It could be something along the lines of:

- "You are at Bush House, King's College London. You have to find out where your programming lab is. To find this, you have to find the department office and ask. At the end, you need to find the exam room. If you get there on time, and you have found your textbook somewhere along the way, and you have also been to the lab, then you win. And if you've been to the student bar to have a drink more than five times during the game, your exam mark halves."

Or:

- "You are lost in a dungeon. You meet a dwarf. If you find something to eat that you can give to the dwarf, then the dwarf tells you where to find a magic wand. If you use the magic wand in the big cave, the exit opens, you get out and win."

It can be anything, really — so be creative. Think about the scenery you want to use (a dungeon, a city, a building, etc) and decide what your locations (rooms) are. Make it interesting, but don't make it too complicated. (I would suggest no more than 12 rooms.) Put objects in the scenery, maybe people, monsters, etc. Decide what task the player has to master.

Base tasks

The base functionality that you have to implement is:

- The game has several locations/rooms.
- The player can walk through the locations. (This is already implemented in the code you are given.)
- There are items in some rooms. Every room can hold any number of items. Some items can be picked up by the player, others can't.
- The player can carry some items with him. Every item has a weight. The player can carry items only up to a certain total weight.
- The player can win. There has to be some situation that is recognised as the end of the game where the player is informed that they have won.
- Implement a command "back" that takes you back to the last room you've been in.
- Add at least four new commands (in addition to those that were present in the code you got from us).

Challenge tasks

- Add characters to your game. Characters are people or animals or monsters – anything that moves, really. Characters are also in rooms (like the player and the items). Unlike items, characters can move around by themselves.
- Extend the parser to recognise three-word commands. You could, for example, have a command `give bread dwarf` to give some bread (which you are carrying) to the dwarf.
- Add a magic transporter room – every time you enter it you are transported to a random room in your game.
- Others. You can invent additional challenge tasks yourself. Several other challenge tasks are suggested in the textbook for you to get an idea of the level of difficulty that is appropriate.

Submission and Deadline

The submission consists of two parts: your code and a report documenting your submission.

The code

You have to submit a Jar of your project to the "Assignment 2: code submission" link in the assignment 2 section on the PPA KEATS page, before the due date.

The report

You also need to submit a written report that includes the following.

- The name and a short description of your game.
- The description should include at least a user level description (what does the game do?) and a brief implementation description (what are important implementation features?).
- A bullet point list of each base task you completed and how you completed it.
- A bullet point list of each challenge task you completed and how you completed it. This should include the challenge tasks listed in this document, as well as ones you came up with yourself.
- For each of the following code quality considerations, give and explain an example in your project where you considered it: coupling, cohesion, responsibility-driven design, maintainability.
- A walkthrough of your game, consisting of the commands that need to be entered to complete/win the game.
- Known bugs or problems (Note: for a bug in your code that you document yourself, you may not lose many marks — maybe none, if it is in a challenge task. For bugs that we find that you did not document you will probably lose marks.)
- A copy of the source of all classes.

The report should be no more than four pages long (excluding the code printout). The report must be submitted to the "Assignment 2: report submission" link in the assignment 2 section on the PPA KEATS page, before the deadline.

Marking

Applications are given four marks, one for each of the following categories (the final mark is an average of the four marks):

1. Program Correctness —The application meets all of the program specifications, i.e., the student has completed all of the base tasks including following submission instructions (e.g., the student submitted a Jar file of their BlueJ project).

2. Code Elegance —The application is written in such a way that the code is reusable and efficient (i.e., memory usage and complexity). The application appropriately uses loops and functions to reduce code complexity and/or repeated code. The application does not have hard-coded solutions or poorly designed solutions. A poorly designed solution is overly complicated, utilises excessive amounts of memory or utilises a slower approach to a problem.

3. Documentation —The application is sufficiently documented. Good documentation/comments should explain what the code does and how it does it. Comments can also be used to highlight nuances in your solution, e.g., a segment of code that only works under certain conditions.

4. Readability —The application is easy to understand and uses good programming practices.

The report will be marked out of 100. Additional details on how marks are given can be found in the Marking Rubric. Once your submission is marked and written feedback has been returned to you, you will have the opportunity to get additional verbal feedback on your submission from your lab TAs.

Deadline

This assignment (code and report) is due **Friday 29th November, 23:55**.