4CCS1PPA/APPA Programming Practice and Applications

Coursework 2: World of Zuul

Jeffery Raphael, Ian Kenny, and Jie Zhang

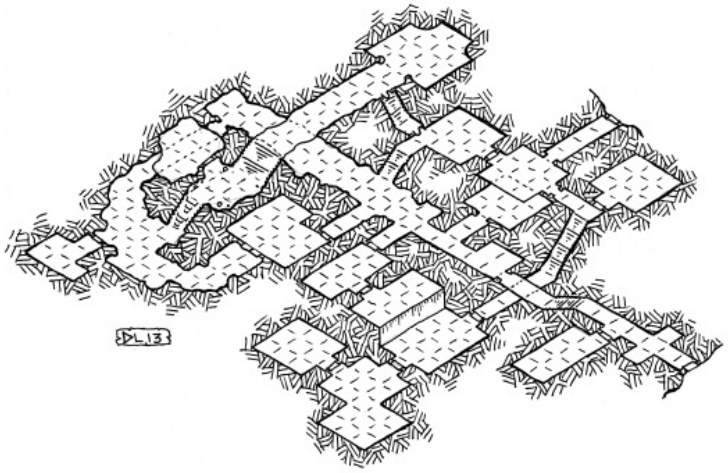
Your task is to invent and implement an adventure game. Along with this document, you have been given a simple framework (`zuul-better`) that lets you walk through a few rooms. You can use this as a starting point.

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.

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

- change the name of a location
- 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, ...)



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 (50 points)

The base functionality that you have to implement is:

- The game has at *least* 6 locations/rooms
- 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 them. 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. Furthermore, the player has to visit at least two rooms to win
- Implement a command *back* that takes you back to the last room you've been in. The *back* command should keep track of every move made, allowing the player to eventually return to it's starting room
- Add at least three *new* commands (in addition to those that are present in the base code); The *back* command will not count as a new command.

Challenge Tasks (30 points)

- Add at *least* three 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

Report (10 points)

You must also write a report (*less than* four pages) describing your game. The report should contain 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 the important 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.
- 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 walk-through of your game, consisting of the commands that need to be entered to complete/win the game.

Submission (10 points) and Deadline

- You must submit your assignment on Gradescope via KEATS by **Wed., Dec.** 6^{th} **16:00 (4pm)**. You'll submit a zip file containing the following

    1. A Jar file of your BlueJ project. —You can create a Jar from within BlueJ by going to Project, and then "Create Jar File...". You do not need to change any of the default options, and so you should just click the "Continue" button. **The Jar file must contain your source code, i.e., the *.java files, and it must run on BlueJ.**
    2. A report (PDF)
    3. All of your Java files (*.java)

- **Your assignment will be penalised if you are missing any files or included files that were not asked for in the task sheet.**

- Click the *Assignment 2: Submission Link* to submit your work. Follow all instructions in the 'Student Submission Guide'. If you have any trouble submitting your work, email Jeffery Raphael as soon as possible. Do not wait until the last hour to attempt your first submission.

- Marking details can be found in the 'Marking Rubric'.

Late Submission Policy.

All coursework must be submitted on time. If you submit coursework late and have not applied for an extension or have not had a mitigating circumstances claim upheld, you will have an automatic penalty applied. If you submit late, but within 24 hours of the stated deadline, the work will be marked, and 10 raw marks will be deducted. If this deduction brings your mark for the assessment below the usual pass mark (40%), your assessment mark will be capped at the pass mark. **All work submitted more than 24 hours late will receive a mark of zero.**