**INTPROG JAVA TEAM COURSEWORK 2013/14**

This coursework is based on one devised by Michael Kolling and is used with his kind permission.

**Set : Feb 6th 2014**

**Demonstrations: In practical classes March 18th – March 21st**

**Worth: 15% of unit marks.**

**Electronic Submission: Submit zipped BlueJ project to Moodle by 23.55 on March 17th.**

**Work to be done in groups of 3/4 students. As the assessment will primarily be a demonstration of the students' final program it will not be marked anonymously.**

## AIMS

**To develop object oriented programming skills, including designing, documenting, testing and debugging of Java code.**

## THE TASK

Your task is to invent and implement an adventure game. You have been given a simple framework (the *zuul-better* project available on Moodle) that lets you walk through a few rooms. You must use this as a starting point. We will discuss this project in future lectures.

### 1 Read the Code

Reading code is an important skill that you need to practise. Your first task is to read some of the existing code and try to understand what it does. To successfully complete the assignment, you will need to understand all of it.

### 2 Make small extensions

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 (see Barnes/Kolling Chapter 6).

### 3 Design Your Game

First, you should decide what the goal of your game is. It should be something along the lines of: You have to find some items and take them to a certain room (or a certain person?). Then you can get another item. If you take that to another room, you win.

For example: *You are at the University of Portsmouth. You have to find out where your tutorial class is. To find this, you have to find the admin 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 tutorial class, then you win.*

Or: *You are lost on Southsea Beach. 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 D Day Museum, the exit opens, you get out and win.*

Or: It can be anything, really. Think about the scenario you want to use (a beach, Southsea, Spinnaker Tower, University Library, a student hall of residence, a gym, the Kings Theatre etc.) and decide what your locations (rooms) are. Make it interesting, but don't make it too complicated. (I would suggest no more than 15 rooms). Each group should have their own scenario, based on Portsmouth – identical scenarios would suggest excessive collaboration.

Put objects in the scenario, maybe people, monsters, etc. Decide what task the player has to achieve.

## 4 Implement the Game

Decide what classes you need to implement the game, then implement and test them. You should divide the coding work up between all members of your group. Each member of the group must do coding, testing and documenting. Remember that in order for a class to interact correctly with other classes you need to have a very clear specification of the class interface (the public parts of the class). Ideally you should define the interface for all classes you require before you implement them.

## 5 Levels

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 he/she has won. Of course a player may also lose.
- Implement a command "back" that takes you back to the last room you've been in. Repeated consecutive use of this command makes you move between two rooms only.
- Add at least four new commands (in addition to those that were present in the original code you were given).
- 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.

### Challenge Tasks

- 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 specialist rooms – rooms that have extra properties/characteristics
- Implement a command "back n" where n specifies the number of moves you wish to go back (i.e. back 2 will take you back 2 rooms, back 3 will take you back 3 rooms)
- Make use of files in your program – you could load the initial scenario from a file and/or provide a facility to save and load from file at midgame.

## 6 Submission and Assessment

Please submit via Moodle a zipped BlueJ project file by 23.55hrs on March 17th 2014. The primary assessment of this work will be via a practical demonstration given in a computer practical lab class between March 18th and March 21st. All group members must attend the demonstration. Each demonstration will last approximately 10 minutes. Each group will be asked to rate individuals contributions to the overall task. Each group will receive an overall mark and each individual their own mark based on their contribution. Failure to demonstrate your program will result in a group's mark of 0%. Individual students who do not attend a scheduled demonstration will also receive 0%. (subject to ECF amendment).

**Marks Breakdown**

| | |
|---|---|
| Demonstration of basic functionality | 30% |
| Demonstration of challenge tasks | 20% |
| Testing | 20% |
| Code quality | 20% |
| Documentation | 10% |

### Advice on completing the coursework

You may ask me or your tutor for any help you may require – you will not be penalized for asking for help. We will not do the coursework for you but will answer any specific problems you have. You may also get help from other students taking the unit but if any submissions use the same game scenario and have an excessive amount of duplicate code the University's plagiarism procedures will have to be enforced. In simple terms – what you submit must be your work not copied from someone else.

All code must be professionally written (comments and indentation!) and will be assessed for:

- correctness
- appropriate use of language constructs
- style (commenting, indentation, etc.)
- difficulty (extra marks for difficult extensions)

Develop your code in small parts. Test each part as you implement it. Compile your program frequently – don't add more than a handful of program statements without compiling it to check what you've done. Keep backup copies of working code.

Think carefully about the design of each class you create – minimize coupling and maximize cohesion.

Don't reject well thought out code just because you get compilation errors – work out why you've got the error and correct it, don't throw away the good with the bad.

Don't be over ambitious. Do not spend excessive time on this coursework. You should spend approximately 20-30 hours (3-4 days).

If you have any queries about this coursework please ask me.

Robert Topp Feb 2014

***KEY POINTS***

- Each group should have their own game scenario.
- Each student in a group must do coding, testing and documentation for some part of the program.
- The work done by individual members of the group must be clearly identified.
- I expect testing code (both standalone test classes and "junit" style test classes) to be demonstrated (see chapter 7 of textbook)
- All students in a group must attend the practical demonstration.
- Failure to attend the demonstration will result in 0 marks

**INTPROG ZUUL DEMONSTRATION FORM 2013/14**

Names : Date:

| **BASIC TASKS** | Done by | Tutors comments |
|---|---|---|
| Implement your own scenario | | |
| Implement an end to the game where the player either wins or loses. | | |
| Include between 10 and 15 rooms in the game. | | |
| Include items in rooms | | |
| Allow players to carry items up to some maximum weight | | |
| Implements “back” command | | |
| Adds at least four new commands | | |
| Add characters to game | | |
| **CHALLENGE TASKS** | | |
| Extend the parser | | |
| Add “special” rooms | | |
| Implement “back n” command | | |
| Use files in the program | | |
| Other challenge task completed | | |

| Task | Good | Ok | Poor | Comment |
|---|---|---|---|---|
| Appropriate use of language constructs | | | | |
| Programmed in good style | | | | |
| Attempted challenging enhancements | | | | |
| Testing | | | | |
| Bugs identified | | | | |
| User documentation | | | | |

STUDENT CONTRIBUTION – DEFAULT IS EQUAL – TOTALS 100

| S1 | S2 | S3 | S4 |
|---|---|---|---|
| | | | |

GROUP MARK (%)

| | |
|---|---|
| Functionality (max 50) | |
| Testing (max 20) | |
| Documentation (max 10) | |
| Code Quality (max 20) | |
| Total (100) | |

INDIVIDUAL MARKS (%)

| | |
|---|---|
| Student1 | |
| Student2 | |
| Student3 | |
| Student4 | |