CS12130_project: Individual Programming Project

Reyer Zwiggelaar

October 18, 2013

This assignment is intended to stretch and challenge your ability to write Java programs to solve relatively well-defined problems on a small scale.

The overall aim of the assignment is to produce an interactive program of the Spread-and-Die game.

I am expecting the use of a full OO approach at this stage.

The program you must complete in this exercise implements the Spread-and-Die game. The playing area is a square and has 12 by 12 cells. At the start of the game the user decides how many regions there will be on the playing area, which should be between 2 and 4 (each region is represented by a unique character of your choice, but clearly not the same as any of the specified characters used). The playing area is randomly filled with the characters representing the regions. The player character (represented as **P** on the screen) is subsequently randomly dropped on the playing area and needs to avoid the spreading disease (**D**). The starting point of the disease is indicated by the user. At every step in the game two things are happening, firstly the player character is moved (at level one of the game this is randomly, while at level two this is intelligently away from the disease), while secondly the disease spreads into neighbouring cells (all cells neighbouring a diseased cell in the same region become diseased, while neighbouring cells in another region only become infected (**I**) at this stage and will become diseased in the next step). For every step the player character survives a point is awarded, and the player character dies if caught by the disease (i.e. occupying the same cell as the disease). If player character obtains 20 points at level one the game starts again at level two. At level three (again reached after twenty points), the player character should randomly (or intelligently) decide to move or to change a random number of their four direct neighbouring regions to one of the other regions. If another twenty points are obtained the user should be declared winner of the game.

The above implementation, when done perfectly, will give you a good pass-mark (close to distinction). However, for those interested in obtaining a distinction. There could be a fourth level in the game, where the player character could develop a cure for the disease after a number of steps, but at the same time the disease would randomly mutate and build resistance to the cure. Another variation on the game, could be that the player character gets more than one

life, but this should be indicated by an option at the start of the game.

The play should either be slow enough to follow on screen, or the user should press the letter n for the next step.

It should be noted that the difficulty of levels is tailored towards the difficulty of implementing them - which is quite different from *ordinary* games where more advanced levels make things more difficult for the user. Here things become easier for the player character (since escaping the disease with a random walk in level 1 is harder than escaping via directed moves in level 2). This is done on purpose, as this is a software development challenge.

Additional functionality would be the provision of a highest score file, which would be loaded at the start of the game and updated as appropriate during (or at the end of) the game.

The functionality of the game includes being able to see:

- the positions of the player character, disease, infected, and regions;
- the current level and variation of the game played;
- the score accumulated, and the final score;
- if implemented, the highest score list (top 10 and where the current user is).

You are required to provide a Java program implementing the game as described above. In implementing your solution you should bear in mind that the aim of this exercise is to gain experience in object-oriented programming, so your program is expected to include more than one class.

The user interface is expected to be a command-line based, Java application, in line with all the programs done so far. No graphical user interface is expected and no additional marks will be awarded for this. There should be no need for multi-threading.

Hints:

- You should use a full set of UML diagrams as the basis for your analysis and design. The Java code should be a direct representation of and consistent with your UML diagrams.
- Tackle the problem one class at a time. Only once you have implemented and tested classes representing each type of object, should you try to implement the game.
- Make sure you have a good idea of what you have to do before you start writing code.
- It is important to test each class in isolation (do not write all the code for all the classes and expect them to work with the game first time). Each class should have a test class.

- Implementation is likely to use arrays (or **Collections**).
- It is not expected that the game is very dynamic as every move of the user is likely to include an n and *an enter* and a redisplay of the playing field.

1 Handing In

You must submit the following materials:

- The analysis/design documentation, which should include all relevant UML diagrams and a short (500-1500 words) justification of the resulting design. The justification should contain a self-evaluation part in which you look back on the way you have done your project and provide a realistic mark for your work in the 0-100 range. See http://www.aber.ac.uk/~dcswww/Dept/Teaching/Handbook/assign_feedback.txt for a form that needs to be completed (besides the Expected Letter Grade I want a mark in the 0-100 range). **Without the self-evaluation the project will not be marked.**
- A printout of the source code (.java) files comprising your solution.
- Example output of your programming demonstrating that you have implemented the game correctly. You could include a short description (200-300 words) for each class you tested. In an ideal case this would link back to the UML diagrams.
- A zip file (not any other compression format, the file should be named *userid_cs*12130*_project.zip*, where userid is your own university userid) of the java files via email to rrz@aber.ac.uk **and** at the same time upload the same zip file on Blackboard under the cs12130 assignment tab. Your main method should be in a file named *RunGame.java*.

Items 1 to 3 should be firmly attached via a staple in the top-left hand corner. Do **NOT** use envelopes, plastic wallets or folders (it will not be marked). Deposit this in the "postbox" in the CS lobby on Dec. 6th between 9am and 4pm (earlier submission need to be arranged with the departmental office). Ensure that your top-sheet is clearly marked with your name. You should also ensure that you include at the front the declaration relating to plagiarism (see the student handbook on the web, section 5). This is available via the same web address listed above.

2 Marking

See: http://www.aber.ac.uk/~dcswww/Dept/Teaching/Handbook/ (Appendices) for details of assessment criteria. You should describe your work as requested using text and diagrams. Use comments in your code to increase readability. Make sure that you test your program and make sure that you describe this testing.

Marks will be awarded for the design, implementation and testing of this assignment:

- Indentation and source code formatting (including commenting) (10%)
- Analysis and design documentation. (20%)
- Good use of programming constructs, including polymorphism. (20%)
- Correct operation (proportion of specification implemented). (40%)
- Evidence of testing. (10%)

3 Plagiarism

Cases of plagiarism in student assignments are not taken lightly by the department, and the consequences of plagiarism by students can be severe. Please ensure that you understand what constitutes plagiarism and read the plagiarism notice from the course handbook or on the web and that you fill out the usual form (http://www.aber.ac.uk/~dcswww/Dept/Teaching/Handbook/ Section 6).