Pseudo-3D Crossword

Grade Arguments

Functionality: Our functionality is a fairly strong aspect of our project. At runtime, our program is very robust. Once the crossword is generated and the program is running, there is basically nothing the user can do to to break the program. This is partly because user input is largely limited to button presses; however, even when the users are prompted to enter from the keyboard, we ensure that these are of an appropriate format and length for each word. We are also somewhat robust to input errors, implementing a NoSuchElement exception with a clear error message and suggested solution.

Design: Our design is probably the strongest part of our program. We believe that we have implemented very elegant solutions and had a good structural design for the program we set out to write. Drawing your attention to both the Program Specification.pdf and Updates.pdf, we want to emphasize some major benefits of our design. Firstly, we made an effort to keep working with the grid itself at a minimum; the grid is only really used for the structure of the puzzle and the graphics. All the checking and comparing is done mostly between instances of the Word class, which is much more efficient than constantly running through the grid. Secondly, we believe that our algorithm for placing words in a plus-shaped grid using a scoring is very elegant - it ensures a uniform distribution of the words, while minimal messing with the grid keeps it running quickly. Finally, we believe that coming up with the LinkedBounds data structure was a great design for displaying limited areas of the grid, and for easily switching between them.

Creativity: While creativity is not the strongest point of our project, we believe that it is more creative than it appears to be. While crosswords themselves are fairly widespread, there aren't many programs dedicated to generating them. Furthermore, we believe that we have offered a fun gimmick and unique tweak over normal crosswords, by having this Pseudo-3D aspect of perspective switching. Finally, we want to point out that most crossword generating programs do not present you with the opportunity of filling out the crossword within the program - they simply create a printable crossword for you. Thus, as per your examples, we believe we have done a "tweak to something that has been done before, but not by many people", warranting "lots of points".

Sophistication: While in its separate parts, our project is may not seem sophisticated, we believe the sophistication comes in the sum of its many parts - all coming together nicely. Firstly, we developed our own algorithm for creating a crossword puzzle on a plus-shaped grid, which required a fair amount of critical thinking and refinement (see Updates.pdf for details). We also created our own data-structure for handling the grid display smoothly, which took some creativity. Furthermore, we handle file input and parsing, with some level of error detection. Finally, we wrap all of this up in a nice-looking user interface, with many functionalities that are all robust to user input and facilitate usage of the program. It was making all of these aspects work together that took the most work.

Broadness: We believe we have ticked boxes 1, 3, 4, 5, and 6. For 1, the java libraries, we made extensive use of the java.awt library in constructing our user interface, particularly by making use of several different JComponents that we did not see in class. For 3, the interface, we had our Word class implement the Comparable<T> interface, in order to facilitate sorting using functionalities of java.util.Arrays. For 4, the user-defined data structures, we created our own data structure, LinkedBounds, to facilitate graphical display. For 5, built in data structures, we made us of ArrayList<T> in order to easily append objects to arrays in different parts of our program. For 6, File I/O, we handled user file input and parsing with some error detection in order to get data for a list of Words. While we only have 5 of the 6 boxes required for full credit, we believe they are all strongly justified and that our score for Broadness should be close to full.

Code Quality: We used the commenting style from Projects 1 and 2 all throughout our code, with very explicit explanations for each and every aspect of the program. We have also commented throughout each method, translating the more abstract parts of code into understandable text. All variables are named in a clear, unambiguous manner, and we followed standard guidelines for Class/Method/Variable naming. The code is organized over 5 different files to minimize clutter. However, we admit that some of the code got messy in the Crossword.java, particularly when dealing with vertical vs. horizontal placement. We believe that our program looks and feels semi-professional, with a clean user interface and streamlined style.