Final Project: Maintaining and Evolving Minesweeper

CSE 671 Software Quality – Fall Term, 2024

200 pts

Improving the Existing Design (30 pts):

For this part of the assignment, the goal is to take an existing software system (https://github.com/ripexz/python-tkinter-minesweeper) and improve the design. The system is a GUI-based Minesweeper game that consists of one class and a main function to execute the game. In its current form, it would not be easy to evolve this system to support multiple views like a text-based version of the game. To better facilitate this, we need the code to be adapted to adhere to the MVC (model-view-controller) pattern. With this pattern, the game model can be separated from the views and UI interactions, and this will allow for the creating of a new text-based version of the game where coordinates can be given instead of boxes clicked. Because of this, we do not need to implement two games simply we can specify which view we want when we run the game.

The **model** component (usually a class, but could be a subsystem consisting of a collection of classes) should capture the logic of the game. The model represents the "brains" of the game. For example, the model should know when a player wins.

The **view** component is responsible for interactions with the user -- both getting inputs and presenting results. In the end, you should have two possible views: one that is a simple text-oriented view, the second that is a graphical interface, which is has currently been implemented. Switching from one view to the other should not require changes to the model.

The **controller** component is aware of both the view and the model. It is the component that determines what method is called next. In a sense, it is the manager of the game.

Documenting the process should be clear and explicitly outlined. Recall, the goal is not to create a new program from scratch, but to take the existing system and improve its design. To facilitate this (and help guide you in considering how perform the reengineering), you will need to provide a mapping of the existing functions to your new classes (e.g., the functions that are now part of the model). It is important that this mapping is clear. For example, you may want to provide a table that has the original function in one column and the new location in the second column (it would be beneficial to include the class and method name, especially if you opted to rename the method).

The expectations are the following:

1. Documentation of reengineering process.
    a. It should provide a clear mapping between the existing project with your newly designed project at the method-level (i.e., you should specify where the methods are now located in the reengineered system).

2. Refactor the existing project so that it adheres to MVC. Please be sure to use comments to draw attention to any new or modified code.
    a. It should be clear/annotated to show the correspondence of the components (i.e., it should be clear what makes up the model, view, and controller).
    b. In the spirit of using object-oriented programming, I expect to see a new class that defines the contents of a cell on the Minesweeper board as follows:
        i. A value for the cell representing the number of mines adjacent to it
        ii. A mine that has not been flagged
        iii. A mine that has been flagged
        iv. A flag without a mine – in this case, a flag should be removed
        v. A hidden treasure – see new game features below

    This cell should be used to define the board for the current game.
3. Implement a new text-based view of the game.
    a. This should be an extension that leverage the model from the prior step and not an independent game.

Existing base project is located here: https://github.com/ripexz/python-tkinter-minesweeper

A Tutorial on the playing of the game is here: https://www.wikihow.com/Play-Minesweeper

Extending an Existing Game with New Features (60 pts):

Before starting this section, please see instructions for your expected test suite so that you can plan your approach accordingly 😊

With the refactored game, we now want to extend the rules of the game. There are several features to implement.

1. We want to add the concept of *hidden treasure*. Instead of populating a board only with hidden mines, it will now also have a single *hidden treasure* location. When uncovering a *hidden treasure*, a player will automatically win.
2. There are 3 possibilities for level of play: beginner, intermediate, and expert. The level of play determines:
    a. Size of the board: 8 x 8, 16 x 16, and 30 x 16
    b. Number of mines: 10, 40, and 99
    c. Number of hidden treasures: always less than the maximum number of mines for the play category

Extra Credit Feature (5 pts):

This version of minesweeper allows for a game board to be read as a csv file. You are being asked to allow the game player to keep a partially-played game so that they may complete it at a future time. If the player asks to suspend play, obtain a file name from them and store the current board for future use. When minesweeper is started, the player should be

prompted to state whether they wish to continue a previous game and that game must be reloaded.

Generating Requires/Ensures clauses for Additional Functionality (50 pts):

The new functionality should be embellished with DbC clauses for future use by a Design by Contract tool or for documentation purposes. You are expected to provide these clauses for any component of your reengineered game that contributes towards the implementation of the two new features listed above.

Testing your Enhanced Minesweeper Game (60 pts):

Testing in this realm is complicated by the random placement of mines and hidden treasures. Therefore, it will be your goal not to test the software, but to implement a testing mode that will enable manual testing of the game. This will mean implementing the following:

- When the game first launches, the user should be asked if they would like to enter testing mode. If they select no, the game resumes as normal. If they select yes, the process below shall be followed.
- The testing mode will then ask for a test game board, specified by a csv file (see format below) and read the board in from the file.
- The game will then validate the test board to make sure it meets test board criteria (see below). If the game does not meet criteria, inform the user of an invalid board and then go back to the original prompt of whether they want to enter testing mode.
- If the provided game board does meet the criteria, populate the board with the specified content, and then play will resume as it would for a normal game, but with the fixed/specified board so that the user knows the solution and can test the game for specific behaviors.

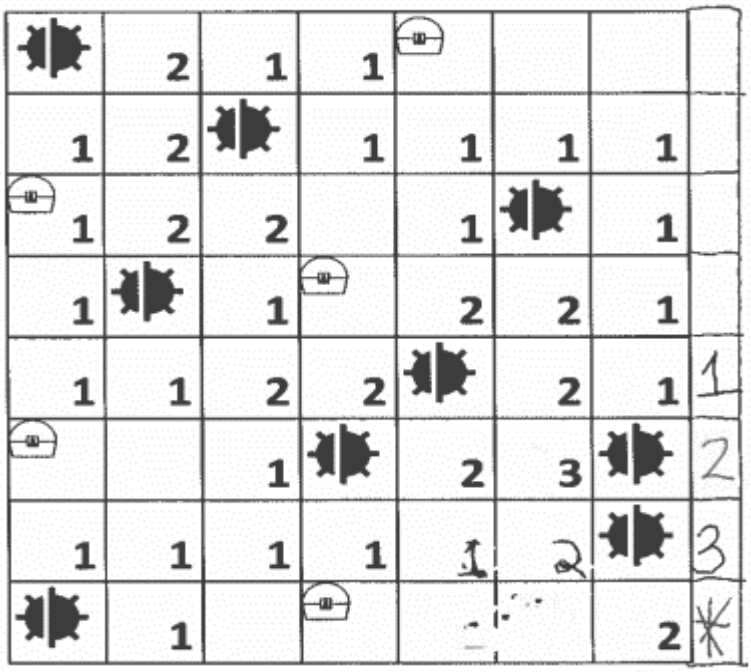
Testing mode will only need to be implemented for the beginner mode (8x8 puzzle, 10 mines), and puzzles will be specified by a csv file meeting the following criteria:

- Each value in the file represents one cell in the game.
- There should be 8 values per row and 8 rows to the file.
- A cell can hold one of three values:
    - 0 – an empty cell
    - 1 – a mine
    - 2 – a treasure

A testing board must meet certain criteria to validate the robustness of the system and ensure consistent testing. As such the following rules must be followed for all test board (these are the criteria you must evaluate against:

- The first eight mines must be placed in such a way that there is exactly one mine in each row and each column. Looking a few bullets down, when adding the ninth and tenth mine, this characterization will no longer be true. Another way to think of this is that if a mine position is indicated by (row, column), each row and column cover the range 1..8.
- None of the first eight mines should be adjacent to each other. Please note that adjacency means either by row or column or by diagonal. However, one of the mines must be in a position where row and column are the same (e.g. 4,4).
- The ninth mine must be placed adjacent to one of the first 8 mines (either by row or column, and a diagonal is not adjacent).
- The tenth mine must be placed completely isolated from any of the existing mines (as in it doesn't touch any other mines).
- There must be no more than 9 treasures; there are no specific placement rules for treasures.
- As an example, the following csv file (left) represents the corresponding game board (right) and is considered a valid testing board for testing mode.

1	0	0	0	2	0	0	0
0	0	1	0	0	0	0	
2	0	0	0	0	1	0	
0	1	0	2	0	0	0	
0	0	0	0	1	0	0	
2	0	0	1	0	0	1	0
0	0	0	0	0	0	1	
1	0	0	2	0	0	0	1



What to submit?

The following should be uploaded to Canvas:

- Please compress (zip) the source code for our game into one zip file and upload this file
- Please submit a video (mpg3 or mpg4) that shows while describing
    - the individual code files,
    - the building of the executable,
    - a demonstration of the play of a game which illustrates an uncover of treasure, the isolation of a mine, and at the end the selection of a mine
    - the use of the testing mode

Don’t forget to provide explanations of what you are demonstrating!