Project Development Process

GEOG5003M Independent Project

This document is meant to provide context for the software developed in fulfillment of the University of Leeds GEOG5003M independent project.

Software Intention

The software provided for the independent project is meant to simulate the fallout of a chemical weapon released from the top of a building. The requirements were obtained from the Bacterial Bomb project suggestion given in the online course materials. The resulting program should allow the user to visualize where particles from the bomb would land, provided a set of model parameters. The required model parameters include wind direction probabilities, particle fall probabilities and the number of particles released from the bomb. The program should allow modification of these model parameters and allow the user to run an arbitrary number of simulations for any given parameter set. Finally, the program should provide a mechanism to output the resulting particle density raster as a text file.

Beyond the requirements indicated in the project suggestion web page, some additional features are included to enhance the model. First, the bomb height is made available as an editable parameter to allow testing the spread of particles from differing starting heights. The maximum number of iterations allowed is also made available as an editable parameter. This allows the user to set an upper limit on the model iterations to avoid long-running simulations.

Software Design

At a high level, the software design follows a Model-View-Controller (MVC) pattern. In this pattern, the Model represents the chemical weapon fallout model and provides an Application Programming Interface (API) to run model simulations and update the model parameters. The View provides a Graphical User Interface (GUI) that visualizes the model simulation result and acts as a mechanism for user input to run simulations, save raster text files and update model parameters. The Controller links the Model with the View by channeling user input into the model and model output back to the View.

Python modules were leveraged to decouple and organize reusable software components. Modules were created to house classes related to the Model and View, along with a module to provide logging utilities shared by the other modules. All modules related to the reusable model components fall under the **simulation** package, while the program's **main** file, which consumes the model components, is housed in the parent directory. Use of the MVC pattern provides a straightforward path to changing the model or view implementations. Any other model or view

class that implements the same public API can replace or integrate with the current implementations. For example, this would allow a developer to reuse the existing view component in the display of an entirely different model implementation. Similarly, different views could be attached to the same underlying model.

The GUI design was intended to both fulfil the project requirements and provide a friendly User Experience (UX). The GUI window follows a familiar pattern, with a toolbar at the top and entry fields, buttons and a model visualization filling the majority of window real estate below. Entry field labels include units, where applicable, to disambiguate the entered values. A Run Model button was also added to the main window contents to allow quick iterations of model simulations. This feature can be helpful for users that want to view several outcomes using the same parameters or experiment with parameter values and quickly view the outcome.

Software Development Process

Following an initial high-level design of the basic program components, an iterative process was used to develop the independent project.

During the high-level design phase, the general requirements indicated in the Bacterial Bomb project documentation were analyzed. As the requirements necessitated a model implementation and a GUI to enter input parameters and view model simulation output, the Model-View-Controller pattern was selected as a high-level design. This design led to an initial package and module structure to house the individual program components.

After organization of the source code files and directories, an iterative process was used to develop the program. In this process, smaller pieces of functionality were completed sequentially, with associated manual or automated testing. This approach minimized both the implementation and testing complexity as the program was developed by limiting active changes to a subset of the full requirements. With a narrow scope of changes, the code production and testing remained focused and had less risk of unintended effects in other areas of the codebase. Continuous refactoring is a consequence of this process because code is expected to be restructured or moved as the program functionality grows. However, with adequate decoupling and program structure, this consequence is minimized and the benefits of lowered complexity far outweigh the extra time needed for refactoring.

The first goal in the iterative development process was to create a model that could be run from the command line without a GUI. Following a fully-functioning model, initial View and Controller classes were created to visualize the model simulation results. Subsequent development iterations added user input functionality, model enhancements, error handling, documentation and unit testing. At each iteration, manual or automated testing was performed, which usually uncovered any major bugs or User Interface (UI) issues. As the iterations were relatively

focused, debugging proved to be less complicated, with quicker resolution times than could be expected if testing was performed after implementing a large feature-set.

Issues Encountered

Several issues were encountered throughout the development process. Many of the issues were of a technical nature, though some UI issues were also observed.

While periodically testing the program during development, erroneous or unexpected program results were sometimes observed. In these cases, temporary print statements were used to help follow the program flow and observe exactly where the actual result differed from the expected result. One particular technical issue that required significant refactoring was the observation of false-positive error handling when parsing user input for percentage values. The initial implementation stored percentage values as a float type in the range of 0-1. Although this implementation worked correctly when the default percentage values were used, some user-entered data would cause a validity check to fail due to floating point rounding errors. As a result, refactoring was undertaken to store percentage values as integers in the range of 0-100, which mitigated the false-positive error messages.

Among the prominent UI issues was the inability to easily detect environment cells with low values in the particle density plot. This issue was due to the default plot cell colour scheme, which displayed a deep purple for low cell values and gradually transitioned to a bright yellow. In this scheme, plot cells with low values were difficult to distinguish from cells that had a value of zero. To address this issue the plot background was changed to a grey colour, while the colour scheme was updated to improve visibility against the grey background.

Although a good UX was an objective of the project, there remain several UX aspects that could be improved. Among the more prominent issues are the need for a responsive layout for smaller window sizes and the inability to change the bomb location or environment size without modifying or replacing the hard-coded input raster text file.

Sources

Several sources were used during development of the independent project. The high-level design and model implementation were heavily influenced by previous experience in the GEOG5003M course, with some pieces of common code and documentation ported and modified from the portfolio assignment. The course notes and practical assignments were also used as reference material during the development process. The official Python 3.7 documentation and matplotlib documentation websites were referenced when debugging program issues or using newly encountered API functions and parameters.

Final Result

The final project output consists of a functioning implementation of the Bacterial Bomb model and its associated documentation in the README file. There are no known issues in the code, although there is the possibility of software bugs that have not yet been detected. The project version intended for assessment is located on the master branch of the GitHub repository (https://github.com/anth-dj/geog5003m_project/tree/master).