Additions to UrbanSim, Progress Displays and Diagnostic Indicators

Joseph Chen, Timothy La Fond, and Koos Kleven

Intro & Background

UrbanSim is an urban modeling project that has been developed to better inform urban planning decisions by modeling an area over time. Modelers setup simulations involving certain parameters such as population, job distribution, and travel patterns. These parameters vary by year, based on the results of the previous year. The final values are called indicators and can be used to inform planning departments on the impacts of a proposed project. A single run of a simulation could many hours, and due to the complexity of the models, runs are not always guaranteed to complete successfully. In light of this, our quarter's project has been to add features to UrbanSim show not only the progress of a simulation, but to display the "health" of a simulation as well. Our project has two distinct subsections; the progress display and the diagnostic display for judging the health of a run.

Over the course of the quarter, the progress display has been modified to incorporate three progress bars for finer granularity of than the initial one progress bar design. The diagnostic display was previously nonexistent but now includes drop-down

boxes (combo boxes) for viewing certain pre-existing indicators. Specifics of these improvements will be discussed in greater detail below.

Development Tools, Language, & Libraries

The first step of adding to UrbanSim was to become acclimated to the code base, as well as learn the language in which it was written. Python has proven to be an interesting language with many powerful features. Its cleanliness and forced use of whitespace constrain developers to use better style. As a loosely typed, Object Oriented, and yet also functional language Python is easily applied to many situations.

Subversion (svn) was used for version control. For the purpose of keeping main code clean and working, svn was used to create a branch for the progress bars and diagnostic indicators display. While this did serve its purpose, the use of svn came with a bit of a learning curve, particularly when the time came to update the capstone project's working version (the branch) with code from the Point 5 main version (the trunk). Generally there was only difficultly where merging two documents were concerned.

Qt4 was used as the cross platform graphical user interface framework. To call functions of Qt4 from Python, the pyQt package was used. Luckily, the existing code had a preexisting progress bar, and various types of window, tabs, and text areas which

served as excellent examples. With the help of examples, pyQt came with very little learning curve.

Progress Display

The progress display was necessary for two reasons. Primarily, it provides the user with a more detailed way to see that the simulation is still running, and informs them as to what specifically is going on. Secondarily, it adds to the aesthetic appeal of the program, replacing a scrolling log of output with a series of progress bars.

There were initially a few ideas as to how to display the progress of a simulation. Different ideas were quickly simulated with paper prototypes, for the purpose of getting quick feed back from modelers at Point 5. Early iterations of the progress display were combined with the idea of showing diagnostic indicators. For instance one iteration had a two dimensional array of buttons, where each button represented a year. As years were completed their corresponding button was to become active, and when clicked, display a set of predetermined diagnostic indicators, diagnostic indicators to be discussed more later. Paper prototypes of this version yielded the opinion that this would not be as aesthetically appealing as other ideas.

Other proposed forms included something of a tree-like document where each

year would have a percentage corresponding to its progress. This proved to be unnecessary, as each simulation runs linearly with each year depending on the last. Showing all years at the same time would make a parallel display of all years pointless. The repeated iterations of paper prototyping and experimenting with code culminated in a tab displaying three progress bars. The progress bars represent overall simulation completion, progress within the current year, and the status of the currently running model within the year. The labels of the status bars change to reflect the current year, model, and sub-model.

The code for the progress display is all located in a single file (runmanagerbase.py). Our need for more detailed run status required the changing of the format and content of 'status.txt', a file used for simple inter-thread communication. Basic finer grained details were added into status.txt, showing the current status of the run in terms of years, models, and pieces. Within the preexisting code, we changed the function responsible for updating the earlier single progress bar. By changing this function to parse the new status file for key information we calculated an estimate for each of the status bars. After a run is complete the status file is no longer checked, as the thread has already returned and the status file destroyed. This left our progress bars reporting a false status of still in progress. Additions to a function called on the completion of a simulation sets all progress bars to one hundred percent after the thread has completed.

After many cycles of tweaking and getting feedback, the status display has finally received consistent positive feedback. While the finer grained display of simulation progress may not be as critical as other aspects of the project, it is hoped that the flashier display will aid in winning over new users for UrbanSim. It is also intended to be more comfortable than a single anonymous bar and better show users what aspects of the project are being run.

Diagnostic Indicators

The diagnostic indicators were the most daunting part of this project, but also the most critical. Seeing early results from a run of a simulation allows modelers to check key values that indicate the health of a simulation. For instance, if a run was going bad a modeler may find that the number of vacancies in the housing market has reached zero. A zero in this value makes future changes to that value impossible, and the run will not produce useful results. Another indicator modelers are interested in monitoring is the population. The constraints of a simulation stipulate that each year the population matches some projected population given as input into the simulation. If these numbers diverge, the run has gone bad, and any further calculation would be a waste. With these basic ideas of what was needed in mind it was possible to begin designing a feature to bring early results to modelers.

Starting with paper prototyping seemed useful but our final implementation turned

out quite dissimilar from our original plans. All previous conceptions of the diagnostic indicators tab were based off of a single table being displayed for each year. As years would complete they would become available to the user for selection, at which point we intended for a bank of indicators to be calculated and displayed. The final iteration allows the user to select the desired preexisting indicator, and the data set (gridcell or zone). It then produces a new sub-tab with a single table displaying the result. While this is not the ideal solution, adding tabs maintains the functionality as well as reducing the clutter on the main tab.

Once development of the diagnostic indicators display started it was quickly decided that the display would have to be partitioned from the progress display. Previous work done on the progress display proved useful, as code was added to the same function that updates the progress bar. The run manager UI keeps track of the years to run, and whether or not a year has been completed. When it is discovered that a year has completed it is then added to the list of available years for diagnostic display. On the diagnostic tab, selection of the year, indicator, and data-set start the calculation of the indicator. This is initiated by the call back function associated with the combo box of available years, which has some interesting implications with regard to the design, both in a layout sense as well as with the code.

Getting the callback from the combo box to initiate the calculation of the diagnostic indicators correctly was one of the largest hurdles in this project. Code was

adapted from the various forms to not only generate the indicators, but also those that display the results. Unfortunately, the version of the code this feature was being added to had not been updated from the trunk since the beginning of this project. The out of date code would have required that the results generators and visualizers be passed an XML node, as opposed to the values stored in the node. Thanks to teamwork from another developer (Travis) this had been made to be a more general function that interfaces easily with our code. This marked the start of the Subversion update woes.

Upgrading the code in the branch proved quite difficult using the svn merge tools. Finally a second branch was made and this projects code was merged in by hand. With the updates to the results generators and visualizers in place it was possible to easily and successfully adapt it to display diagnostic indicators. Diagnostic indicators can currently be viewed by gridcell or zone, though it would be very useful if a cumulative version of each indicator could also be displayed.

Improvement & Expansion Opportunities

This project has proven to be an open door, in that it has opened a host of new paths for expansion. The existing code has many possibilities to be streamlined as well as extended with new features. The display of diagnostic indicators has the greatest potential for additions, whereas the status bars are mostly complete.

There are many improvements to be made to the usability aspects of this project. One major improvement involves the callback function that initiates the calculation and display of diagnostic indicators. As the project's code currently stands, the calculation of indicators is initiated when a year is selected from the year combo box on the diagnostics tab. This causes an issue in the case that a user specifies a year before specifying the data set or the indicator, nothing will be generated. Currently the way around this would be for the user to specify the year once more, to initiate the callback related to the combo box. This convoluted work flow could be easily escaped with the addition of a button to start the calculation of diagnostic indicators. Another way to improve usability would be to change the bank of diagnostic indicators, to a diagnostic specific set. Currently, diagnostic indicators are calculated based on how they are configured in a simulation's XML configuration. If the diagnostic indicators could display totals rather than per-cell values, like total population instead of population per gridcell, the results would be more human readable. The resultant table from a more diagnostic specific indicator would be easier to compare to control totals, and would not carry extraneous information. While these improvements to the internal workings of the diagnostic indicators are important, changes could also be made to better the appearance of the user interface as well.

Once diagnostic indicators have been simplified, their display could also follow suit. Much could be done to the display. In the current state of the project a new tab is opened for each time an indicator is calculated, regardless if it has already been

displayed. One simple improvement would be to check for this redundancy and switch to the desired tab as opposed to creating a new one and adding the ability to close tabs. In the same vein, if the diagnostic indicators are made to be more succinct, then it would be a better choice stylistically to display all the indicators for any given year on one tab. This would make the UI easier to read and therefore more usable. Certain indicators are also better displayed as maps, so another possible improvement could be found in displaying those maps. While modelers would mostly judge the health of a simulation from numerical data, maps could provide a secondary means to judge the health of a simulation. While these improvements are extremely related to this project, they do not extend the ideas of the progress and diagnostic displays as much as is possible.

One way to use the general idea of the progress and diagnostic displays in a new way would be to make a web interface for modelers. This would allow for users to check the status of large runs from anywhere, instead of at site.

Ethical Implications & Conclusion

UrbanSim allows urban planners to make informed decisions. The models it runs can predict the population, industrial, and traffic impacts of a proposed project and determine if it is a desirable change. Often this includes evaluating the environmental effects of a project and potentially redesigning the project to reduce negative impacts such as air pollution. However, urban planners and modelers may not have the specific

programming knowledge needed to manage UrbanSim at a code level. This is why an accessible graphical user interface is essential for an interdisciplinary tool such as UrbanSim. Our changes to the interface make UrbanSim more appealing and usable to clients, but also intend to help expand its user base and uphold its professional appearance. In addition, the diagnostics system that has been added may help users to cancel flawed simulations at an early stage, conserving both time and money. In conclusion, the changes made by this capstone project increase UrbanSim's accessibility to non-programmers as well as make it a more efficient tool.