Andrew Suezaki

Sprint 5 Assignment: Wrap Up

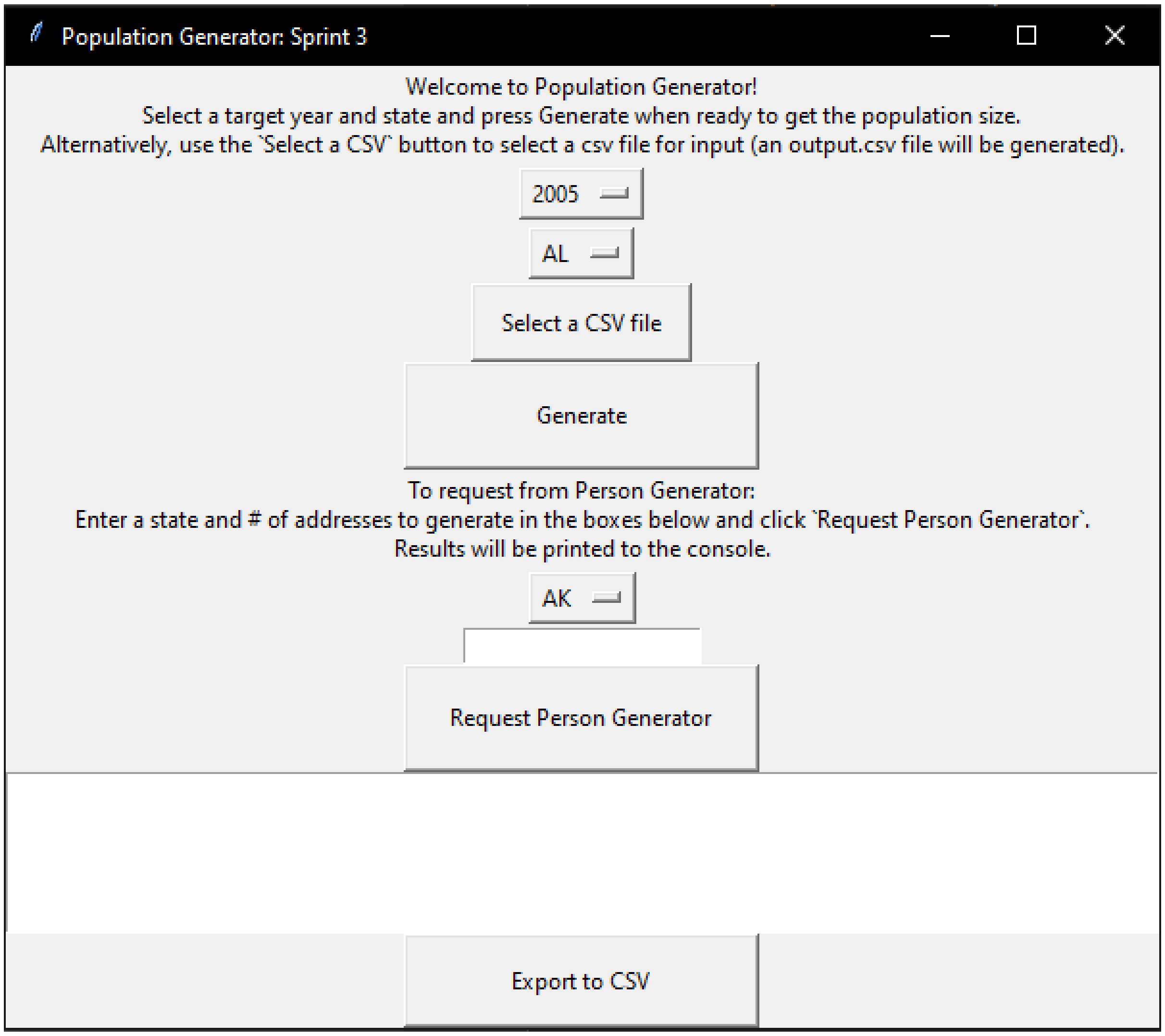
Project: Population Generator

CSH Evaluation

- CSH 1: Explain what new features do and why they are useful
  - The GUI reflects this heuristic in that there is a text prompt explaining how to use the new optional ‘Request Person Generator’ feature.
- CSH 2: Explain what existing features do, and why they are useful
  - The GUI reflects this heuristic. There is a prompt at the top explaining how to use the program and its purpose. However, there is not an explanation of the required format for input CSVs which would improve reflection of this heuristic.
- CSH 3: Let people gather as much information as they want, and no more than they want
  - The GUI reflects this heuristic in that only the current requests results will display to the text box and users are able to refresh this box with new results as many times as they would like. The two requesting services are also clearly divided on the GUI so only those looking to use Person Generator would read the text underneath the Generate button. The GUI is also pretty intuitive and some users may not even need to read the instructions unless they want the extra information.
- CSH 4: Keep familiar features available
  - The Sprint 4 update only added features to below the Generate button in the GUI which kept most of the original/familiar features the same. The program can be used as it normally would before the Sprint 4 additions.
- CSH 5: Make undo/redo and backtracking available
  - The GUI does not reflect this heuristic, because there are no undo options built into the GUI. However, the flow of the program using dropdown menus for easily changeable input and constant refreshing of the text box with current results somewhat removes the need for an undo/backtracking option.
- CSH 6: Provide ways to try out different approaches
  - This heuristic is reflected in the GUI, because the program offers the option to select a CSV file for input as an alternative path to selecting input from the dropdown menus. The program can also be run via the command line, however there are no instructions within the GUI on how to do that.
- CSH 7: Communicate the amount of effort that will be required to use a feature
  - This heuristic is reflected, because the requirements/effort for each requesting service are clearly explained in a text prompt above their respective input options. The ‘Select a CSV file’ button and the instructions above both indicate that a CSV file is necessary to use that feature.
- CSH 8: Provide a path through the task
  - The text prompt clearly describes the path through the task with the optional entry points of using the dropdown menus, a CSV file, or a person generator request. However, it could also include an explanation of where to look for the results so that the entire path through the task is described to the user.

- CSH 9: Encourage mindful tinkering
    - This heuristic is not reflected in the design as there is not much room for tinkering and the flow of the program is linear. However, the additional options to request from Person Generator and select or export a CSV do allow for some tinkering. Although the buttons are self-explanatory and there are instructions, there are no popup windows to confirm any button clicks.

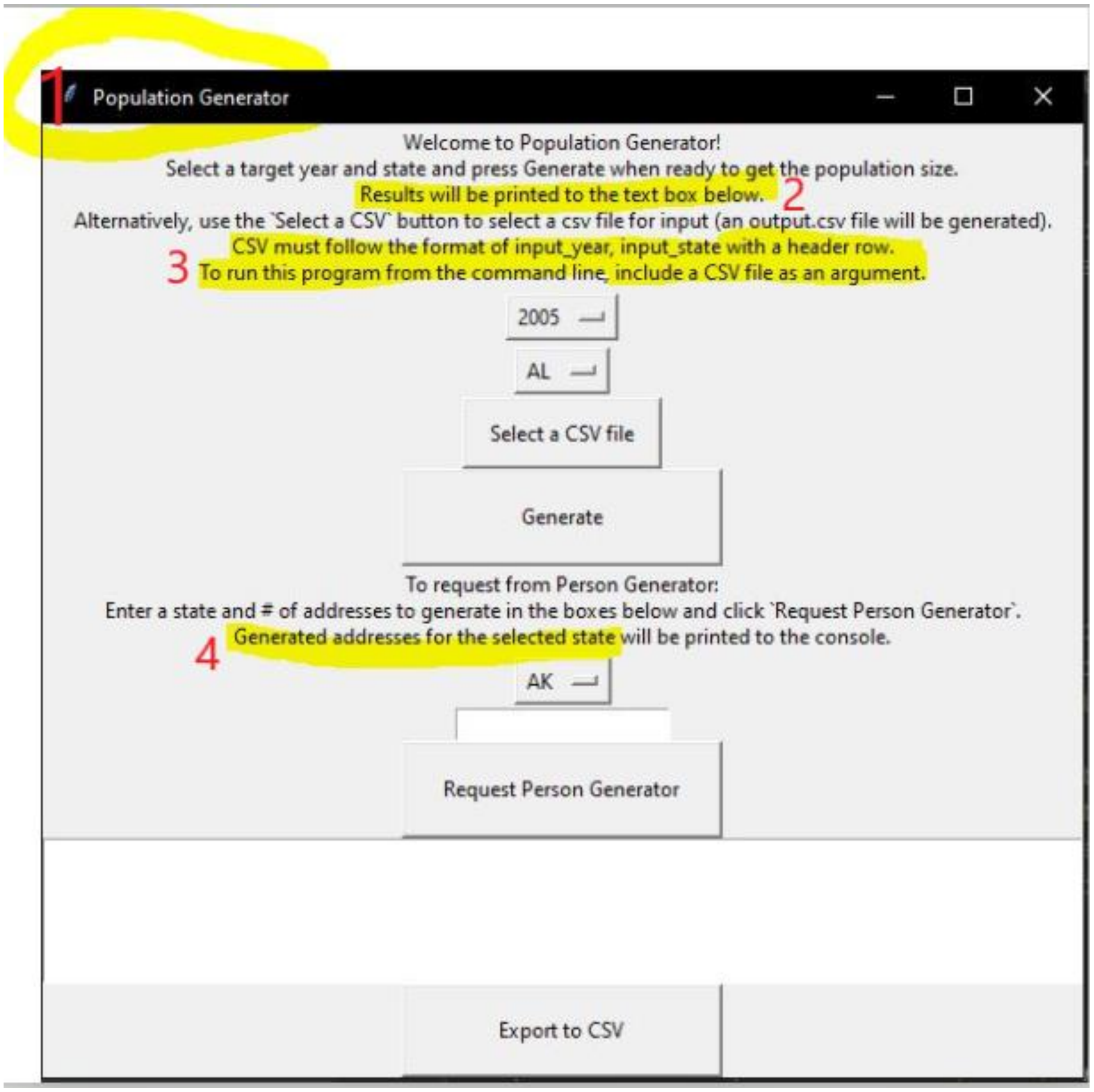
Before any revisions:



Welcome to Population Generator!
Select a target year and state and press Generate when ready to get the population size.
Alternatively, use the `Select a CSV` button to select a csv file for input (an output.csv file will be generated).

To request from Person Generator:
Enter a state and # of addresses to generate in the boxes below and click `Request Person Generator`.
Results will be printed to the console.

After:

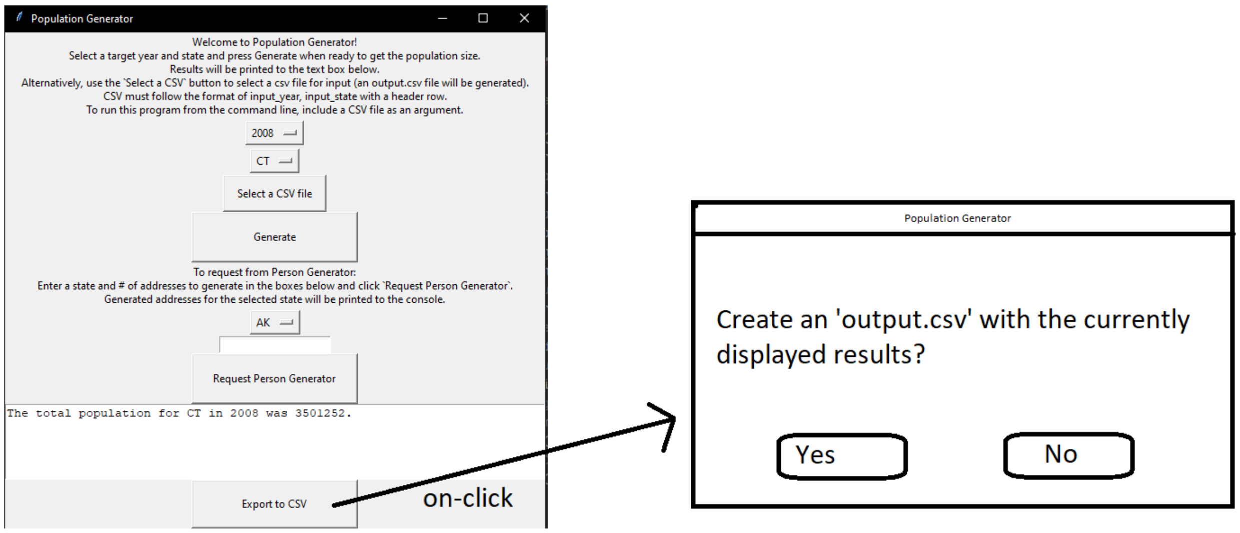

1
Population Generator

Welcome to Population Generator!
Select a target year and state and press Generate when ready to get the population size.
Results will be printed to the text box below. 2
Alternatively, use the `Select a CSV` button to select a csv file for input (an output.csv file will be generated).
3 CSV must follow the format of input_year, input_state with a header row.
To run this program from the command line, include a CSV file as an argument.

To request from Person Generator:
Enter a state and # of addresses to generate in the boxes below and click `Request Person Generator`.
4 Generated addresses for the selected state will be printed to the console.

1. Adjusted window size to accommodate changes in instruction prompts and removed Sprint 3 from the window title.
2. Added an explanation of where Population Generator results will print so that the path through the task is clearer (CSH 8) and the existing features are better explained (CSH 2)
3. Added an explanation of how to use the program from the command line and of the required format of input csv files so the GUI better reflects CSH 6 and CSH 2.
4. I added in an explanation of what kind of data Person Generator returns so that CSH 1 could be better reflected in the GUI.

Paper Prototype



Welcome to Population Generator!
Select a target year and state and press Generate when ready to get the population size.
Results will be printed to the text box below.
Alternatively, use the `Select a CSV` button to select a csv file for input (an output.csv file will be generated).
CSV must follow the format of input_year, input_state with a header row.
To run this program from the command line, include a CSV file as an argument.

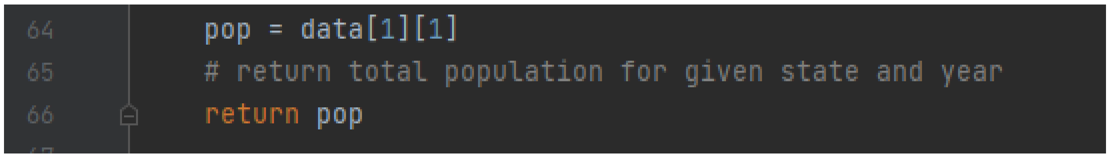
To request from Person Generator:
Enter a state and # of addresses to generate in the boxes below and click `Request Person Generator`.
Generated addresses for the selected state will be printed to the console.

```
The total population for CT in 2008 was 3501252.
```

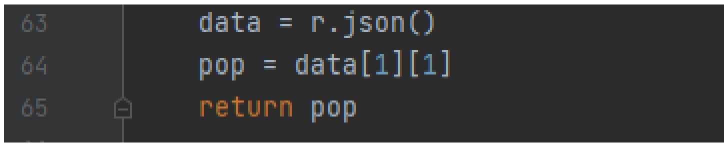
Create an 'output.csv' with the currently displayed results?

Above is a small paper prototype showing a popup window to confirm a button click on ‘Export to CSV’ added. I think the addition of this extra confirmation would help the GUI to better reflect encouraged mindful tinkering (CSH 9).

Code Evaluation

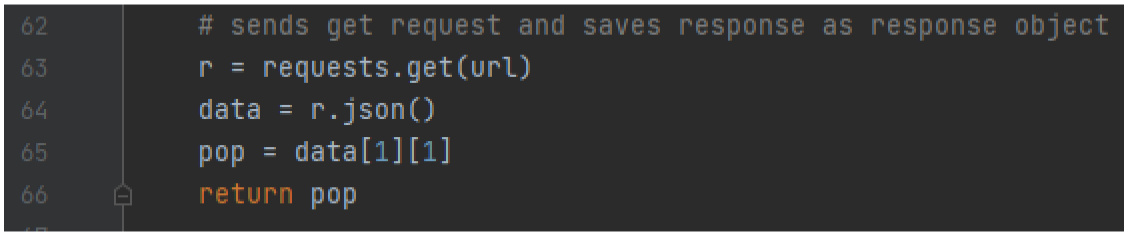
- Code Smells About Comments
  - Obsolete Comment
    - My code does not contain any obsolete comments. All of the comments describe current functionality.
  - Commented-Out Code
    - There are no lines of commented-out code present in my program.
  - Redundant Comment
    - 

```
64        pop = data[1][1]
65        # return total population for given state and year
66        return pop
```

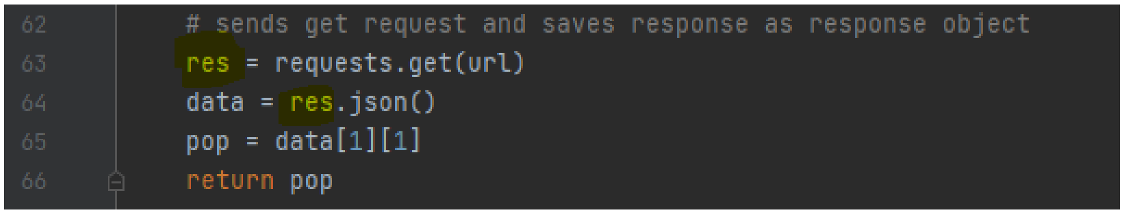
    - My code had a redundant comment in my getreq() function on line 65. It is already apparent that the population is being returned and was also stated in the docstring below the function declaration. To refactor, I removed the comment (see refactor image below). Aside from this, no other redundant comments were found in my code.
    - 

```
63        data = r.json()
64        pop = data[1][1]
65        return pop
```

  - Long Comment
    - There are no long comments in the code. Docstrings describing functions are only 1 sentence long.

- Code Smells About Functions
    - Long function
        - The only long function in my code is the function runGUI() that initializes the GUI. As explained in the Sprint 4 smells document, I feel that the length is necessary in that it is better to keep all GUI related stuff together for readability and neatness. I chose to keep the button declaration lines expanded because I think it would be better not to sacrifice the code readability. I could not make a separate function to pack the GUI elements because it involves having to pass the declared buttons and menus to a separate function which would just create another code smell: Function with Many Parameters.
    - Function with Many Jobs
        - There are no functions with jobs that are beyond what the name suggests in the code.
    - Function with Many Parameters
        - There are no functions with more than four parameters present in the code.
- Code Smells about Code in General
    - Duplicate Code
        - There are no sections of duplicate code present.
    - Long Lines
        - The code does not have any long lines. All long lines are broken into multiple smaller lines.
    - Inconsistent Conventions
        - There are no inconsistent conventions in the code. All style conventions are consistent.
    - Vague Naming
        - 

```
62      # sends get request and saves response as response object
63      r = requests.get(url)
64      data = r.json()
65      pop = data[1][1]
66      return pop
```

        - There was one cause of vague naming with my use of the variable r in lines 63-64. I refactored this smell by changing the name r to 'res' to help show that it is referencing the get request response.
        - 

```
62      # sends get request and saves response as response object
63      res = requests.get(url)
64      data = res.json()
65      pop = data[1][1]
66      return pop
```

Software Process Models

If I were to continue this project and I would be working on it by myself, I would choose to not use a formal approach and use my own methods instead since I feel that is how I would work most efficiently. However, if I were to continue it with a high possibility of group work or collaboration in the future, I would choose to use Scrum. I feel Scrum is an appropriate approach to the project since the client was not initially sure what they wanted and there was a lot of back-and-forth communication required to get the initial understanding of the requirements/functionality down. I think Scrum is also a better choice for going back and making changes if the client changes their mind about anything. I also think a more formal approach such as Scrum is necessary when working in groups since it keeps everyone organized and on the same page.