Programming Assignment: Game of Life, Serialization and Final Submission

Assignment Breakdown

a. Next Generation
b. Graphical Interface
c. Animation
d. **Serialization**

Here are the additional notes about the Game of Life.

Objectives

- Serialization.
- File I/O.
- Using javax.swing.JFileChooser.
- [optional] JFileChooser file filtering
- [extra-credit] MRU list

Instructions

In previous PAs, you created an implementation of the Game of Life. If you consider the pieces of the Game of Life, clearly the core of the Game of Life is what you coded up in Next Generation; this part included all of the 'business rules', that is the conditions where birth, survival, overcrowding, and loneliness occur. This is the **Model**. However, it was a rather unsatisfying implementation from a user experience point of view. In Graphical Interface, we made it look 'pretty' with graphical output, rather than using ASCII output, which made it look better. This is the **View**. One of the other unsatisfying things about that initial implementation was the need to hard-code a starting point. The Challenge Level of the Graphical Interface PA integrated a Controller into the View, allowing the user to click on the 19x19 board to toggle the state of the corresponding cell. In Animation, the Game of Life added additional functionality by automating the calls to nextGeneration.

Save State

For this phase of the project, you will support saving and recalling the state of the Game of Life. Now, the state information is what's stored in the "two-dimensional" matrix in the Model. (The quotes are there since technically Java does not support two-dimensional matrices, *per se*, but handles them as arrays of arrays.) What you need to do is serialize the data for storage into a file. You may find the API documentation for java.io.Seralizable to be helpful.

Now, this serialized data (yes, it just means that it's been arranged into a single string of bytes) needs to be stored. Recall that java.io is structured around input and output streams. So, you will need to create an output stream (with an associated file open for writing) to save the state of the game. The serialized data can be stored using that output stream. Similarly, to restore the stored state of the game, you will need to create an input stream (with an associated file open for reading). This storage shall use byte streams

The question arises about user interface for saving and restoring the state information.

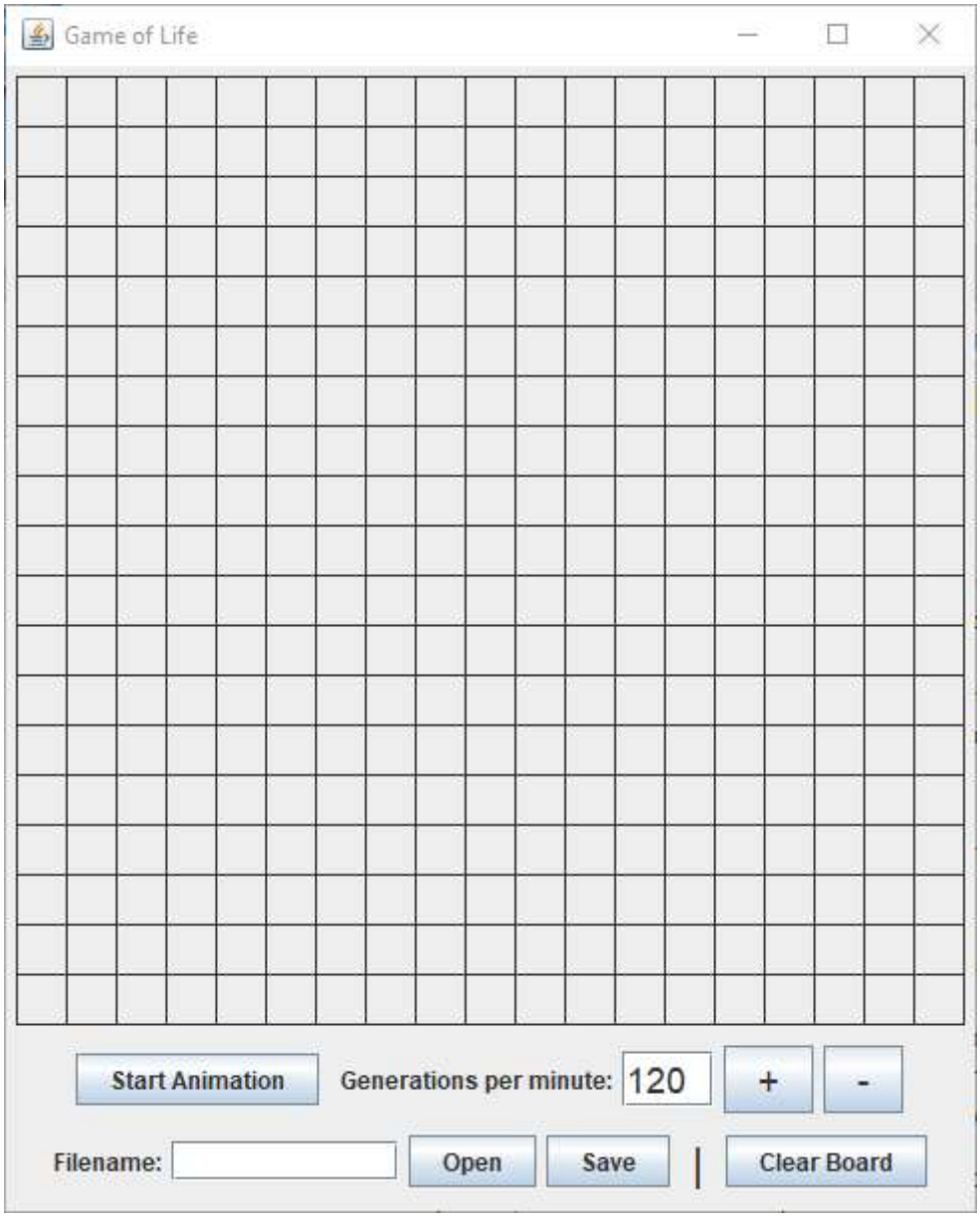
Minimal Level

There are two tasks for the Minimal Level:

- Toolbar for file handling
- Use command-line arguments to load an initial board

File Handling Toolbar

For the Minimal Level, the interface for serialization can be very simple, a second toolbar for saving and restoring the game state. The additional toolbar should contain a textbox for the filename and two buttons, Open and Save.



Here is a video showing an example of the Minimal Level.

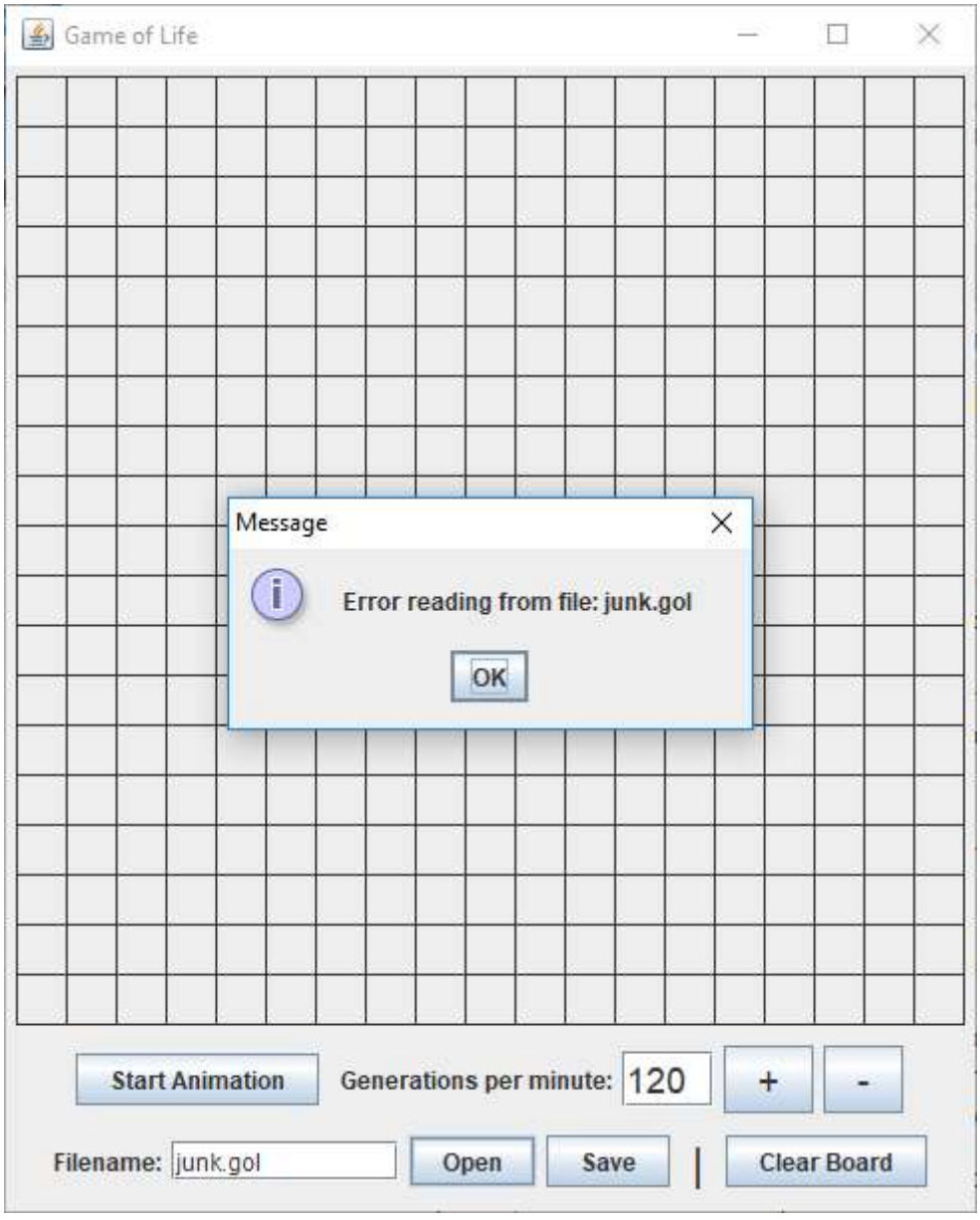
The actions of saving and restoring the state of the Game of Life are operations on the model, MyGameOfLife. So, these operations should be implemented as methods in MyGameOfLife.

Use the following method headers for the additional methods in MyGameOfLife:

```
public void fileOpen(String filename) throws java.io.IOException;
public void fileSave(String filename) throws java.io.IOException;
```

Notice that these methods do not handle the IOException, but instead let it propagate back to the caller. This allows the caller to take appropriate actions. In this case, the caller is the button handler for the Open or Save button, depend on which button was clicked. If the filename were to come from a different source, a different response to the IOExcpetion could make sense.

In this case, the filename was supplied by the user, so a message dialog is appropriate. In response to the IOException, the handler can display a simple message dialog box. Use JOptionPane.showMessageDialog.

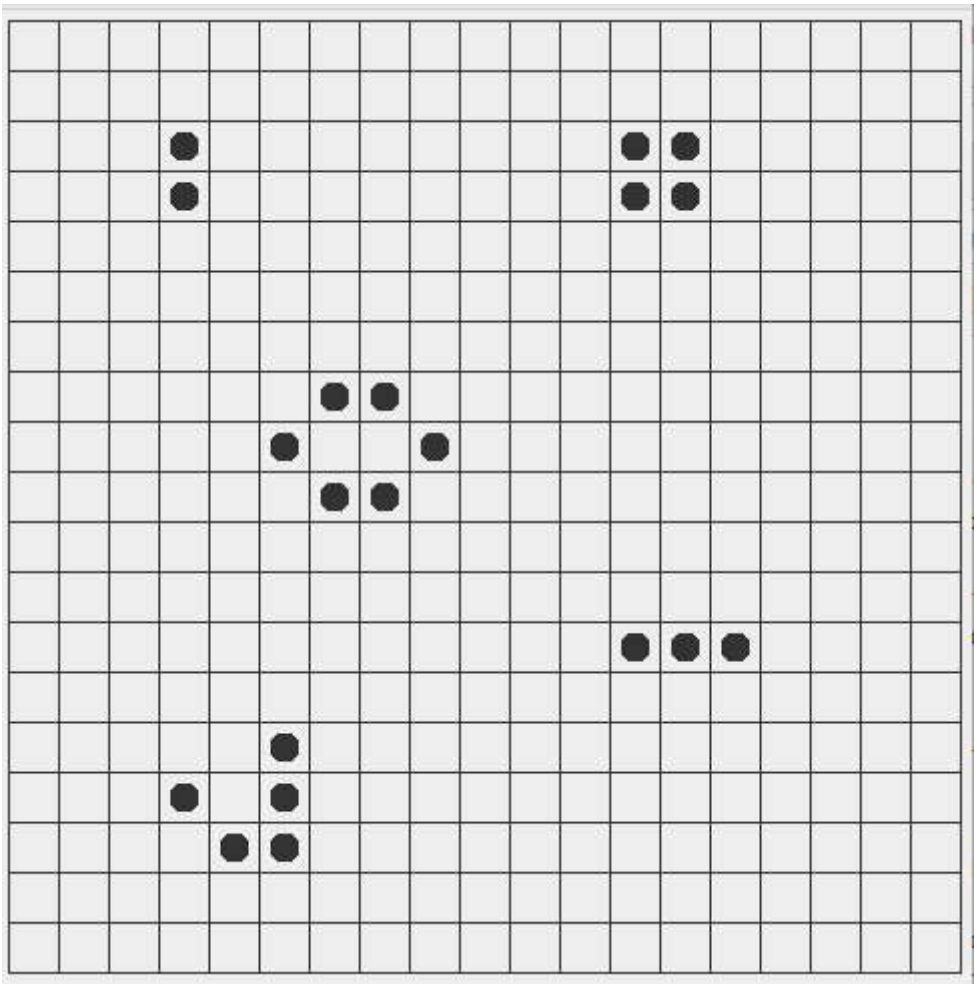


Yes, it's an "in your face" user interface design, but it's often good to use something very noticeable to communicate back to the user; a modal dialog box with a system alert works well for this.

Command-Line Argument

Update main to use the command-line arguments. If there are more than zero command-line arguments, assume that the first command-line arguments, args[0], is the name of the Game of Life data type. Use the fileOpen method of MyGameOfLife to load the board from the file.

Test your application by creating a Game of Life data file in your file format with the starting point configuration of ALIVE and DEAD cells that we have been using for the entire project.



Use your implementation of Game of Life to store this in a file. Name the file: **start.gol**. You will submit this file as part of the assignment.

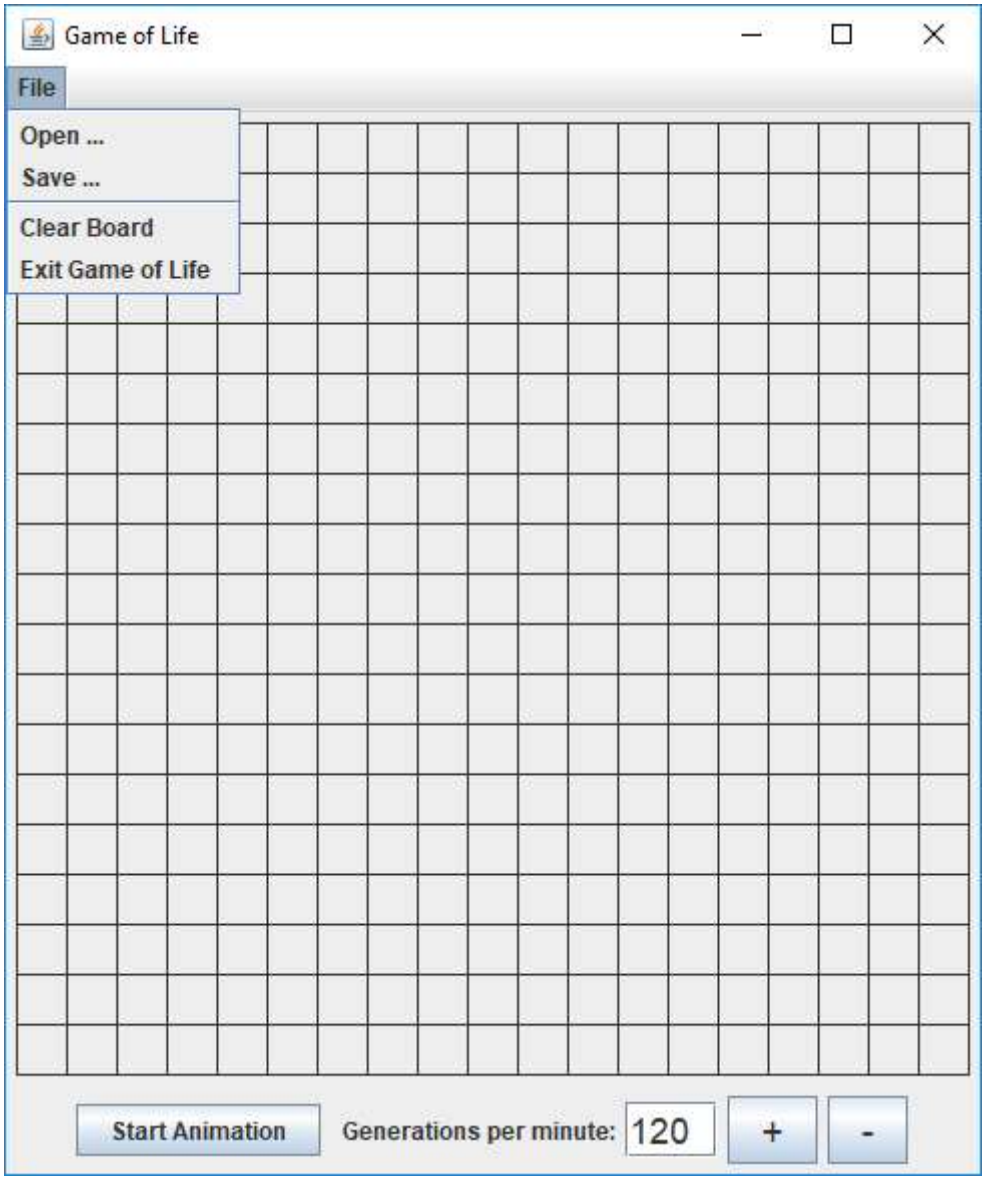
Standard Level

For the Standard Level, the support for files will be enhanced in several ways:

- The commands will be moved to the File menu
- The Open and Save commands will use JFileChooser to help with finding files in folders
- JFileChoose will use a FileFilter to make the list of files more manageable
- The Save command will check before "clobbering" existing data files

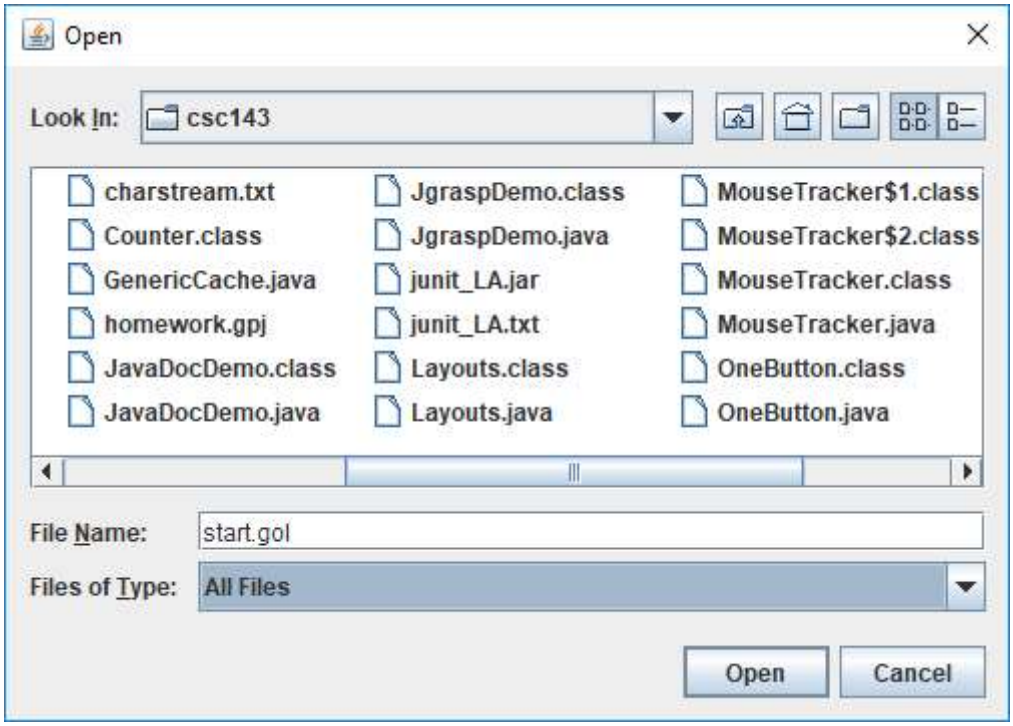
The File Menu

The menu can be quite simple, four commands: Open, Save, Clear Board, and Exit Game of Life.



Menus in java.awt and javax.swing are quite straight forward. Here is a link to the tutorial on menus in swing in the Oracle documentation. You will see that the individual menu items are like buttons, with an ActionListener re handle the menu item being selected.

Both the File : Open and File : Save menu items are marked with ellipses. This is an indication that a dialog box will appear when the menu item is selected. The dialog box will give the user a way to specify the file for opening or saving.



Use JFileChooser box to get the file designation for the Open and Save commands.

JFileChooser

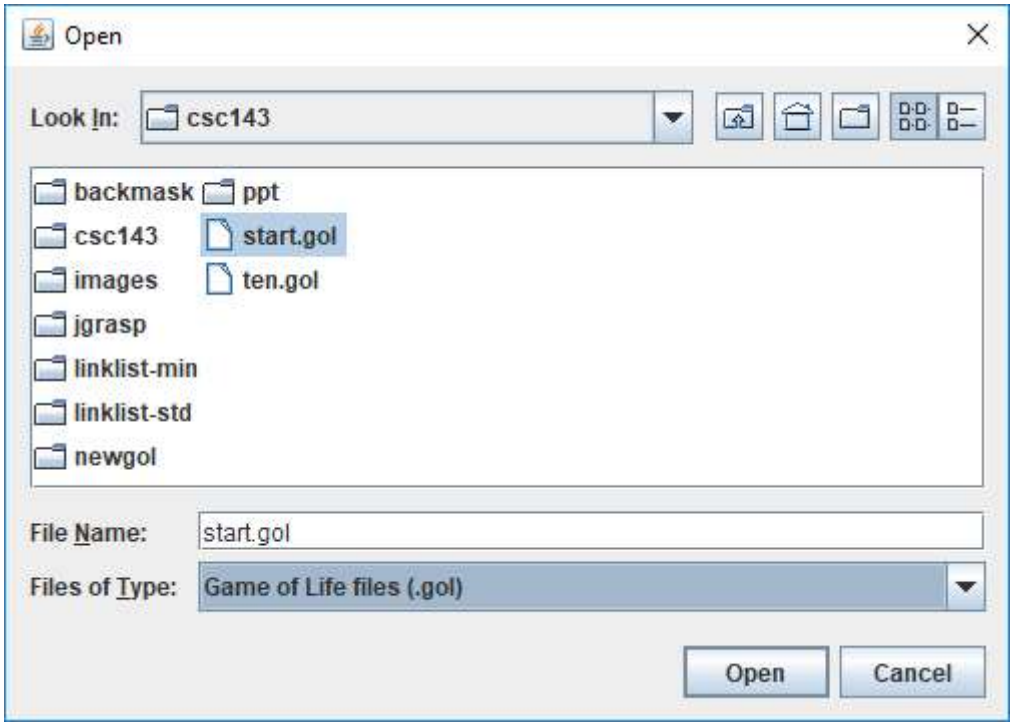
javax.swing.JFileChoose provides support for file selection. This may be the most intricate task for the Standard Level. The tutorial on JFileChooser from Oracle gives you all the information that you will need.

A couple of notes:

- JFileChooser works with the java.io.File type. This provides much better support for file handling than a String filename.
- To support working the java.io.File, add overloads to MyGameOfLife:

```
public void void fileOpen(java.io.File file) throws
java.io.IOException;public void fileSave(java.io.File file) throws
java.io.IOException;
```

The JFileChooser will be more user friendly if it has a FileFilter.



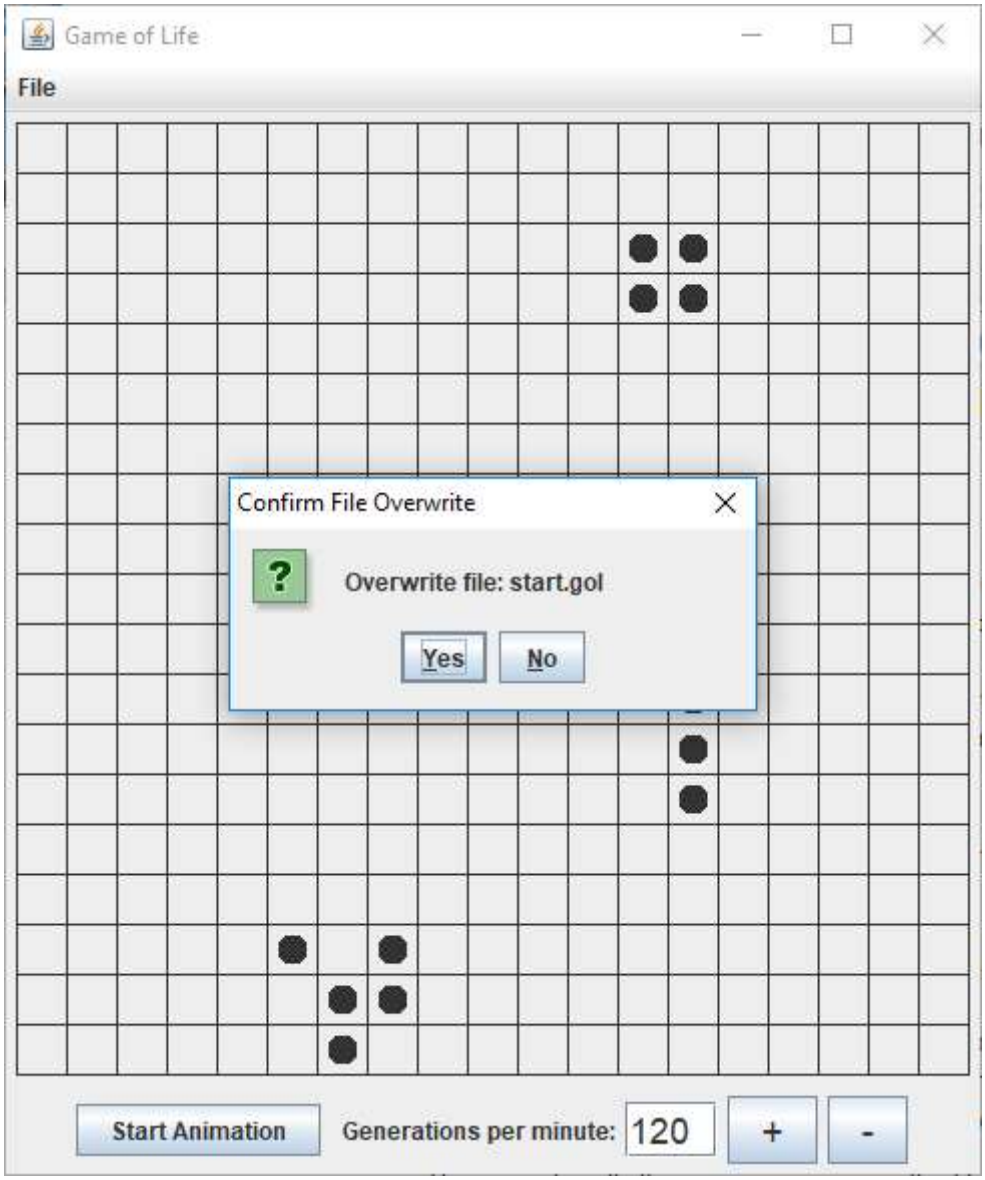
Set up the FileFilter to accept the .gol file extension, as shown above.

There is information about setting up a FileFilter in the JFileChooser tutorial.

No "Clobber"

Unlike most applications, it is reasonable for the application to 'ask' before it overwrites a file. Because the Game of Life, as a cellular automaton, can always calculate later generations using the rules about "birth", "loneliness", "overcrowding", and "survival", there is little reason to have automatic overwriting of a data file with a new configuration of ALIVE and DEAD cells.

So, when saving a file, if that data file already exists, the application shall verify that the overwrite is what is desired.



This can use the showConfirmDialog method of JOptionPane. That's what's displayed in this image.

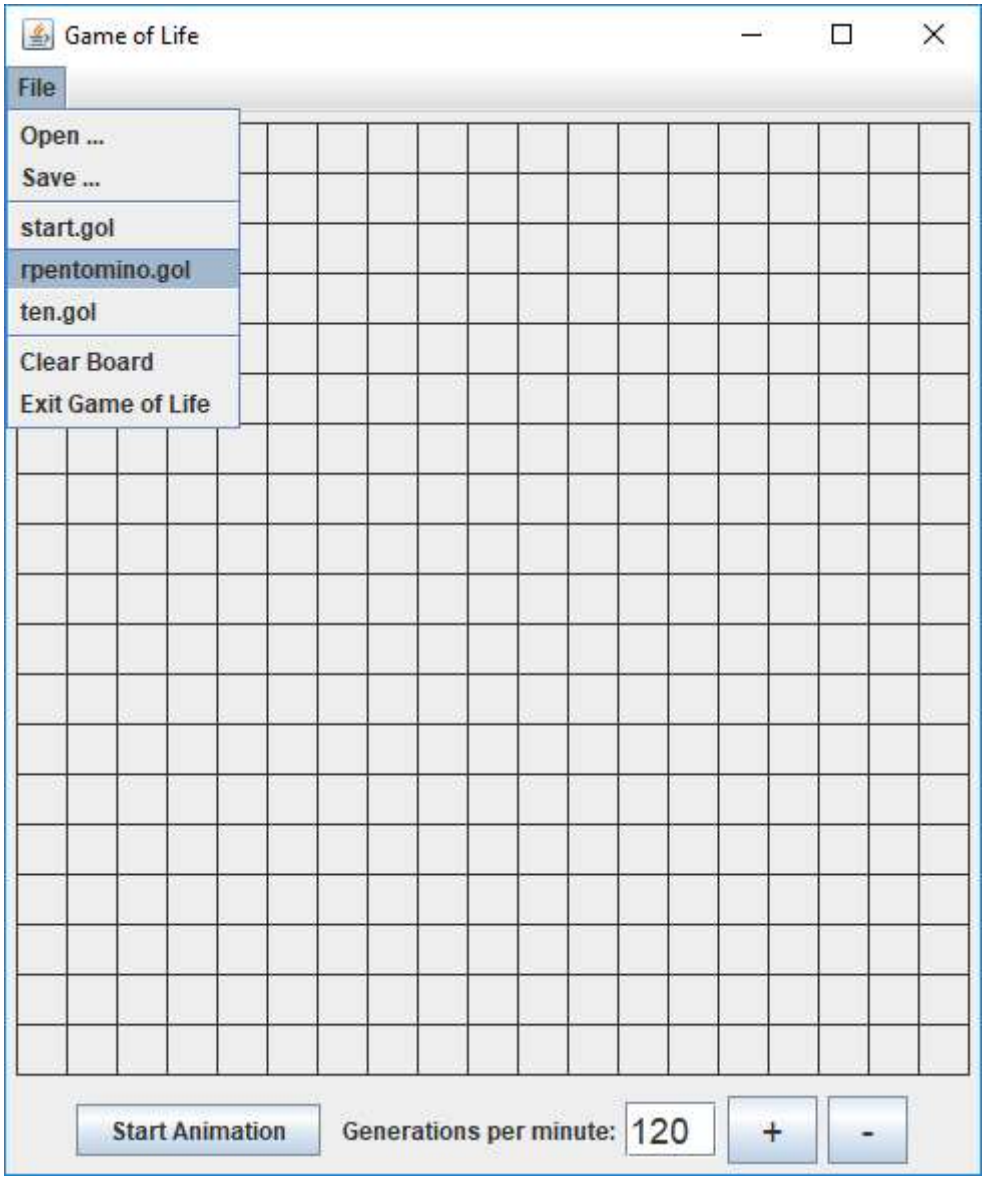
If the user selects Yes, the file is overwritten; if No, the file is not written.

Challenge Level

For the Challenge Level, implement a MRU file list.

MRU File List

One of the common user-interface features we find in applications these days is the MRU file list (Most Recently Used). This is an list of the *N* most recently used files, ordered by the time of access, most recently accessed on the top of the list. (A typical value for is 4.) As data files are opened or saved, the MRU list should be appropriately updated. Selecting one of the MRU menu items will open the corresponding data file.



The MRU list is configuration information. The configuration data needs to be stored in a different file format than the ALIVE and DEAD cell pattern of the .gol files. When the application is started, load the MRU file list into the File Menu. Update this list whenever files are opened or saved. And, when the application is closing, store the MRU list in the configuration file.

Written Report

Write a report in which you describe:

- the surprises or problems you encountered while implementing this assignment the most important thing(s) you learned from this assignment
- what you would do differently next timeext time
- what changes, if any, you made in your implementation to Parts a, b, or c
- a brief discussion of your decision about when to store the configuration file

I expect a clear report that has some thought in it. It will be easiest if you take notes about the process as you work on the assignment, rather than waiting till the end to start the written report.

Note:
This is the final report for the project. It could provide information about the complete process you used in

implementing the Game of Life, not just notes for this phase, Serialization. Once again, the rationale for this is that reflecting on what went right and what went wrong is probably best means to use these programming assignments as tools for learning.

Submission

Three (3) files:

- One (1) Java archive file containing the Game of Life application
  - both source code and bytecode
  - entry point for the Game of Life application
- One (1) Game of Life data file, start.gol
- One (1) ASCII, plain-text report file (.txt)

Grading Rubric

New Functionality: 9 points
Minimal Level - 5 points
- Game of Life serialization (file I/O), with toolbar support
- application loads command-line argument, if supplied; reports error if file cannot be opened and read
Standard Level - 9 points
- Game of Life serialization (file I/O), with File menu support
- JFileChooser for Open and Save commands, FileFilter for .gol files
- Confirmation dialog before overwriting existing data file
Challenge Level - 11 points
- Standard Level functionality
- Functional MRU file list in File menu
- MRU configuration data stored and retrieved at application shutdown and startup

Previous Funtionality: 9 points
- nextGeneration method of MyGameOfLife correct - 3 points
- GameOfLifeBoard as view - 3 points
- Observer pattern implemented - 2 points
- Animation - 1 point

Style: 3 points
- conforms to Homework Guidelines regarding style

Documentation: 3 points
- JavaDoc comments on all public and protected elements
- a separate comment on each class, field, method (can be JavaDoc comment)
- JavaDoc comments generate no warning or error messages
- each JavaDoc comment include a brief description in addition to required tags
- method / constructor internal comments to explain functionality
- report addresses the questions posed.

Back