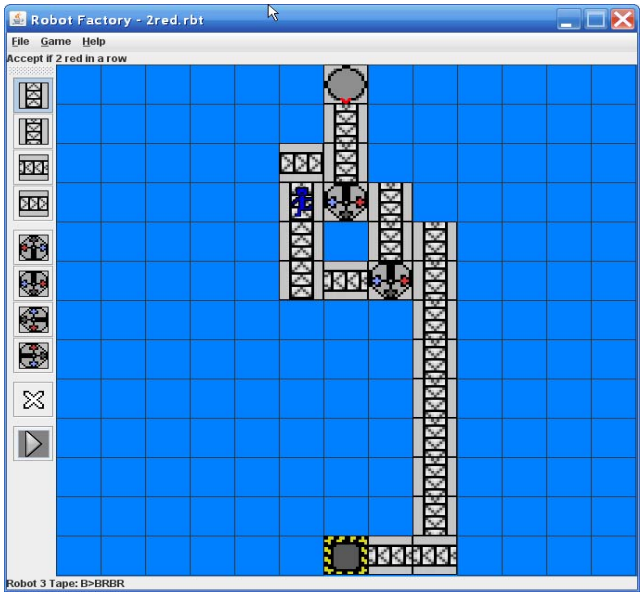
CSCD 370, GUI Programming with JFC/Swing
Project: Robot Factory Game
Due: 9 AM, Thursday, March 22 – No late submissions will be accepted!

A deterministic finite automaton (DFA) is an abstract machine that reads input from a serial (nonreversible) stream and changes between a finite number of states according to the current state and the current input. When the input stream is exhausted the final state of the machine is used to draw some general conclusion about the input string. There is usually one state that indicates the input string was accepted (according to some criterion), and any other final state may indicate that the string was rejected. In Computer Science these concepts are applied in areas such as determining whether some input meets the grammar rules of a programming language. When a DFA is enhanced with the ability to move backwards and forwards over the stream and to overwrite the input symbols with output symbols it is called a standard Turing machine, which provides an important theoretical model for what it means to "compute." Turing machines appear prominently in decidability theory, which is the study of problems that can and cannot be solved with computers. I hope you are aware that there are fairly straightforward problems that can be proven to be unsolvable by computers. Some of these proofs are surprisingly simple as long as you think recursively. If you were not aware of this, I encourage you to do some reading about the Halting Problem.

Your course project is to create a puzzle-solving game that provides animation of deterministic finite automata. I call this game Robot Factory, and it is modeled after an online Flash game called Manufactoria which in turn was inspired by Zachtronic's Games for Engineers. The general idea is that Robots emerge from a source tunnel and follow conveyor belts that are placed by the player. Each Robot carries a sequential tape containing red and blue marks. When the Robot encounters a switch (also placed by the player), it moves in one of three directions depending on whether the current mark is red, blue, or the tape has reached the end. After taking one of those directions, the tape is advanced one position to the next mark. Each Robot (and tape) represents a test case for a particular Goal that is stated in a message bar just under the Menu Bar. The Goal states conditions under which the Robot should be accepted by moving it to a sink tunnel. A Robot is rejected by directing it onto any blank cell on the game board. Goals are expressed in terms of features of the tape, such as "contains at least two consecutive reds, followed by one and only one blue." The Goal statements, test tapes, and outcomes will be expressed in text files that are loaded from the File menu.



Following are detailed specifications for the project, along with some hints on how to proceed. Read the specifications very carefully - each behavior has some points associated with it. I am also providing the scoring sheet that I will use to grade your project. If you are uncertain about any of the requirements, please ask. I am distributing the project definition early so that you can make incremental progress as you acquire knowledge. I suggest that you get started as soon as possible in order to take maximum advantage of your opportunities to receive my help. Only bad things can come if you wait until the last week of the quarter to get started. I will provide a 32-bit Windows executable (it is not a class file, so don't waste your time trying to decompile it) of a solution that goes beyond the minimum

requirements. I will also supply you with some GIF files for the playfield elements, but feel free to design your own. You can use other image formats with Swing – I often use GIF for such things because it allows the specification of transparent pixels. I offer some ideas for how you might improve the implementation if you are so inclined. Feel free to pursue those or any other enhancements you think of as long as you don't violate the requirements. I suggest you start by playing around with my solution, including creating a goal statement and some test cases beyond the ones I provide. You are not, however, required to provide any test files with your solution.

Additional Specifications

1) Your playing surface should initialize to a 13x13 grid of squares. You may add functionality to modify the playfield size if you wish.

2) The main window should be sized around the grid (and of course the status, menu, and title bars), and stretching the window to a larger or smaller size should not be allowed. This is easily accomplished by calling pack() after the main frame window has been assembled, followed by a call to setResizable(false).

3) Your playing surface should initialize with 1 source at the top center and 1 sink at the bottom center. You may add functionality to move these items around if you wish. You may support more than 1 sink if you wish, but your game should allow 1 and only 1 source.

4) Your game must support the placement of conveyor belts for movement in the up, down, left, and right directions. These must be available for placement from a toolbar that is initially placed at the left, as shown. You may provide 4 tool buttons for this, or a single tool button along with some mechanism for rotation, at your discretion. If you provide rotation you MUST also provide a toolbar button for the rotation along with some visual feedback to the user of the orientation that is about to be placed. Some ways to do that would be to have it rotate the conveyor button picture or to change the cursor to a picture of the conveyor in the current rotation.

5) You may make fix the toolbar or make it detachable and moveable.

6) The pictures on the toolbar are not required to be exactly the same as those on the playfield. My solution does that for convenience, but you may decide, for example, that you want the toolbar pictures to be smaller.

7) Your game must support the placement of switches in at least the top-entry orientation. This part must also be available from the toolbar. Note that my implementation supplies three other orientations as well. In addition to supplying rotations you may also supply flipping of the red/blue exit directions.

8) Item placement on the game board must be such that a new item can be placed over the top of another item, thereby replacing it. Your toolbar must also contain a tool to remove an item from the game board, reverting to a blank cell. You may provide for the movement of items on the game board by dragging them, but this is not required.

9) The toolbar must somehow provide feedback as to the currently selected tool. The easiest way to do this is to somehow change the appearance of the selected button, such as by outlining it, changing the background color, giving it a depressed look, or changing the cursor to match it.

10) Your toolbar must also contain a momentary run button. Like the corresponding Go menu item (described below), it should start a loaded game or resume a paused game. If the game is running, pushing the run button should have no effect. You may disable it while the game is running if you wish. You may also design it to toggle between run and pause. If you do, then the picture should change appropriately between typical run and pause pictures that you would find on a media player. You may also supply a separate pause button if you wish.

11) If a Challenge file has not been loaded, then selecting Game/Go or pushing the run button should either not be allowed or should pop up an appropriate message dialog. My implementation does the latter.

12) Your game must include a File menu with at least three items: Open Challenge, Load Solution, and Quit. They must have the same shortcuts and accelerators as my solution. The Open Challenge and Load Solution selections must support filtering of the current directory for Challenge files with the extension .rbt (for robot) and for Solution files with the extension .grd (for grid), respectively. The required file formats are described below. You may add other appropriate menu items if you wish.

13) A successful load of Challenge file should incorporate the filename into the title bar.

14) Your game must include a Game menu with at least three items: Go, Pause, and a third selection that toggles between Faster and Slower. The action of the Faster/Slower selection is described below. The first two must have the same shortcuts and accelerators as my solution. The third may have shortcuts and accelerators as well, but they must then toggle in an appropriate way with the text of the menu item. As long as a Challenge file has been

loaded, selecting Go should start or resume the animation. Pause must pause it. When one of these is enabled the other should be disabled.

15) The player must also be able to control the speedup and slowdown by pressing the cursor up and cursor down keyboard keys, respectively. Doing so should toggle the Faster/Slower option on the Game menu appropriately. See below for a hint on getting this done.

16) Your game must include a Help menu with an About item that launches an appropriate About box. It should, at the least, contain your name and the year.

17) The default animation speed should be one grid cell every second. The animation should show smooth movement between cells in 4 substeps (250 msec per substep), but there is partial credit for moving between cells in a single step. Selecting Faster should increase the speed of animation by a factor of 10, and selecting Slower should reduce it back to the default. Note that because the menu option toggles, the player should not be able to accumulate multiple speedups or slowdowns. Feel free to animate the robot's legs (and arms) if you like. Feel free to animate the conveyor belts if you like.

18) Your game should animate only 1 test case at a time from the current Challenge file. A new robot should not emerge from the source until the current robot has been discarded or entered the sink. Before animating the next test case, a dialog box should be displayed indicating whether the test case passed. Once the user dismisses that dialog, the discarded robot should be erased from the playfield. Please note that success or failure of a test case is not the same as saying whether the robot was accepted or rejected. The test case file indicates whether the robot should be accepted or rejected and passing or failing means that the result of animation matched the expectation. Note that the player can put a robot into an endless loop. Your game is, after all, a programmable machine.

19) If any test case fails, animation of the current goal set should terminate. Pressing or selecting Run should cause it to start with the first test case again. The player thus does not get to peek ahead to other test cases after a failure. Unless, of course, they open the file with a text editor, which is pretty easy to interpret.

20) Some indication should be provided when all test cases pass successfully.

21) Your game must be able to repaint itself when the window, or part of it, is obscured and redisplayed.

22) The game should include a status window at the bottom that displays the current test tape (robot number) along with the current read position. Mine does that with a simple string containing R for red, B for blue, and a > symbol pointing to the current value. Feel free to get fancier here if you wish. Perhaps red dots, blue dots and a square surrounding the current value.

23) Your robot should change color to match the current tape value. Red for red, Blue for blue, and something else once the end of the tape is reached (I used a black outline and transparent interior). Note that it is possible for a robot to still be traveling around after the end of the tape has been reached.

24) Your game should include a status window between the menu bar and the main playfield that displays the Goal as stated in the file.

25) You may add sounds to your game but this is not required. We will cover how to play sound clips in class. See below for some hints on how I implemented sound in my solution.

26) Your game must read Challenge files that are formatted as in the following example:

```
Accept if at least 2 red in a row
5
F
BB
P
BRRRB
F
BBRBR
F
BRBRB
P
RBRRB
```

The first line states the goal. The second line gives the number of test cases in the file. The test cases come next and each test case occupies two lines. The first line of a test case is F for Failure (the tape does not meet the goal

so the robot should be rejected) or P for Pass (the tape meets the goal so the robot should be accepted). The next line represents the marks on the tape. B (for Blue) and R (for Red) are the only entries for a tape. It is not required that you detect invalid tapes. Note that there is no blank line at the end. Your game must be able to handle this.

27) Your game must read Solution files that are formatted as in the following example:

```
13
13
0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 6 6 4 0 0 0 0 0 0
0 0 0 0 3 5 8 4 5 5 0 0 0
0 0 0 0 0 0 0 8 6 3 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0
0 0 0 0 0 0 2 5 0 0 0 0 0
```

The first line is the number of rows in the game grid and the second line is the number of columns. My solution ignores these because it uses a fixed 13 x 13 grid. They are included in case you want to implement variable game sizes. I will only test your solution with 13 x 13 grid. What follows is a specification of the content of each grid space, as follows: 0=empty cell, 1 = robot source, 2 = robot sink, 3 = up conveyor, 4 = down conveyor, 5 = left conveyor, 6 = right conveyor, 8 = switch with top entry. These are the minimum components that you are required to support and the only ones I will test your solution with. Feel free to make use of unused codes for other components, such as switches in other orientations (my implementation does that). Note that there is no blank line at the end. Your game must be able to handle this.

Additional Hints

1. You need only one timer to drive the animation. I recommend that you use javax.Swing.Timer, instead of java.util.Timer.
2. Because the screen changes do not happen rapidly, the easiest way to get things redrawn during animation (timer hits) is to simply call repaint(). You might want to note how this is a different (and simpler) approach from what we took for our line drawing lab.
3. The easiest way to draw an image file is to create an ImageIcon from the file, call getImage() on the ImageIcon, and pass that to the drawImage() method of the Graphics object.
4. Don't go crazy on the objects. Do you really need an object for each cell, or is it sufficient to use a 2D array of integers to represent cells and what they contain? In games like this I generally find the latter to be more straightforward (and more efficient and less prone to error).
5. You'll need to implement a KeyListener in order to process the up/down keys. Think carefully about where you do that. An appropriate choice should allow you to reuse code that responds to the menu selection. The class you choose may not be a focusable by default, but as long as it is derived from Component you can call setFocusable(true) to fix that. A problem that has been hanging around for some time is that a toolbar, if present, will grab the key focus and NOT pass along any unprocessed keystrokes. Thus, for example, if you choose to process keystrokes in your Frame class, you're going to need to recapture the key focus at some convenient time (like perhaps when Run is selected). You can do that by calling requestFocus().
6. Note that you'll need an array of Strings to hold the tapes from a test file and that the String class has a charAt method.
7. I am not requiring that the various classes you create be reusable, and thus I will not complain if all your classes are in a single file. In this case I prefer it, so that you can submit your work by simply sending me a single java source file, rather than a zip of multiple source files.

8. My solution uses javax.sound.sampled, which is what I recommend if you decide to incorporate sound. All clips are read from .wav files at game construction time as shown in the class notes. The clip for conveyor sounds is looped continuously and then stopped when another clip needs to be played. For Robot entry, rejection, and acceptance, I play the clip once. I have found that I have to call start and play on the clip after calling loop(1) or it will fail to play the next time I need it.

9. Start simple, and add functionality incrementally. Here is a rough suggestion for a development sequence: Start by building the frame, canvas, menus, and status bars. Add skeleton handlers for the menu selections. Handle the Help/About selection. Add data members that represent the board state and add code to draw the board. Cells that contain items can be drawn from GIF files. Add the toolbar, buttons, and event handlers that record the selected tool in a member variable. Add a mouse event handler that determines the board cell being clicked and changes its state according to the currently selected tool. Add a class for the robot with methods that draw it and set its color. Make a decision as to whether your canvas or the robot will be responsible for knowing the robot position. I did the latter so my robot class also has methods that return its position, set its direction, and tell it to move one increment. Add a timer and get the robot animating. Add pause and go functionality. Add detection of robot movement into a new cell, changing direction appropriately if it is a switch, or ending the test case when it encounters a blank cell or the sink. Add code to read the test case file and display the goal and the tape. Add code to advance the tape and do so when the robot encounters a switch (change the robot's color as well). Add code to increase and decrease the animation speed. Wait until you have all required functionality in before adding optional things like sound clips.

Project Score Sheet

Pts	Val	Description of requirement
	4	Play area initializes to a 13 x 13 grid of squares
	3	Main window is sized around the grid
	2	Main window cannot be resized by stretching it
	2	Play area initializes with a source at top middle and sink at the bottom middle
	2	Game has a Help/About box
	2	Menus have the required shortcuts and accelerators
	4	File dialogs have working filters for Robot files (.rbt) and Grid files (.grd)
	2	Pressing run before loading a Challenge file is either not allowed or results in a warning.
	5	Solution files are correctly parsed
	5	Challenge files are correctly parsed
	1	Challenge filename appears in title bar
	2	Goals and test tapes are shown in status windows
	4	Game supports placement of conveyor belts in 4 orientations
	4	Game supports placement of switches
	3	Switches and conveyors can be placed over top of each other
	3	Game repaints when a portion is obscured and redisplayed
	3	Conveyors and switches can be removed from the play area
	4	Robot emerges from the source at the start of a test case
	5	Robot moves at a default speed of 1 cell per second
	4	Robot moves between cells in 4 substeps
	3	Game can be run and paused from Game menu and run from toolbar
	2	Game can be sped up and slowed back down from Game menu
	2	Faster/Slower menu item changes state appropriately
	2	Game can be sped up and slowed down using cursor up and cursor down
	2	Cursor up and cursor down modify the Game menu appropriately
	5	Robot changes direction correctly at a switch
	3	Robot tape is advanced at a switch
	2	Robot changes color with the tape
	4	Test case ends when (and only when) Robot reaches an empty cell or a sink
	3	Test case success or failure is correctly detected and reported
	3	Game advances to next test case on success
	2	Failure resets to the first test case
	2	Success for all test cases in the file is reported
	1	Game can be quit from File menu