CSCI 201 Programming Exam 1 Grading Criteria

% of Final Grade	Criteria
2.5%	There is a Manager Office image at the correct location in the factory
1.5%	The Manager Office is properly stored in the factory backend
0.75%	There is a button for each worker displaying the correct name
0.75%	Each button executes an action specific to the index of the worker it belongs to
1.5%	Backend receives an action when the manager wants to see a worker
3.0%	Worker visits Manager Office when appropriate

Grading Setup

- Add the provided factory2.txt file to the student's project in the "resources" folder. The only changes are on line 4 (Factory Name|Computer Factory|**6|15|15** -> Factory Name|Computer Factory|**7|13|13**), however we won't edit the factory.txt file directly as this causes a slew of server issues.

- Now go to WebSocketEndpoint.js and on line 24 update "factory.txt" to "factory2.txt".

- Finally in FactoryWorker.java, add a List with names for the workers, for example:

  private transient static List<String> workerNames = Arrays.asList("Alana", "Bob", "Caleb", "Dana", "Edgar", "Frank", "Gary");

- Then update the first line in the constructor to be:

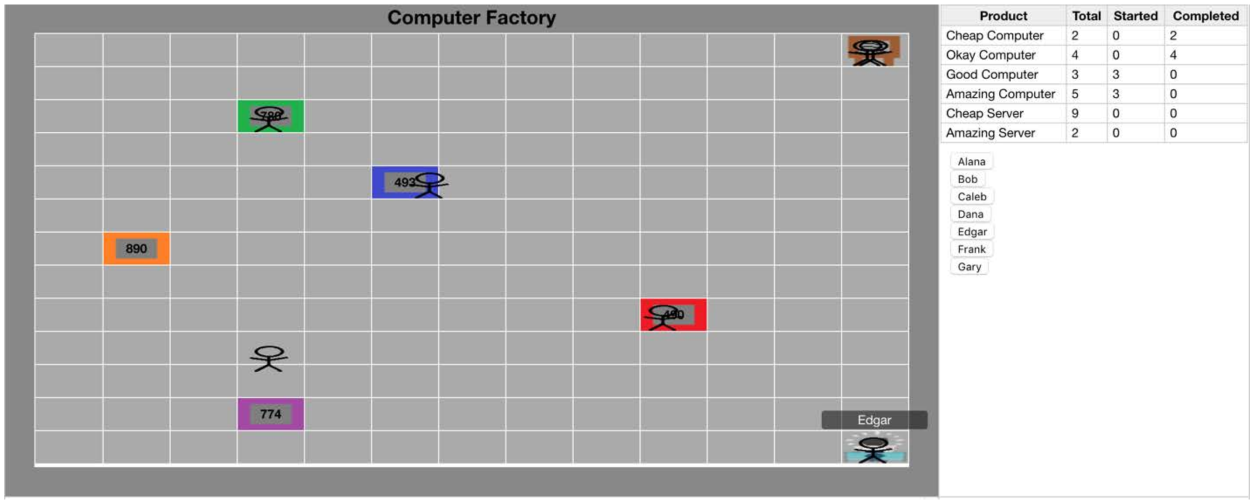
  super(workerNames.get(inNumber), "Worker" + Constants.png);

Grading

Take a look at the Programming Exam Instructions for clarity on certain instructions. Here was the goal of the exam:

> Add buttons for each worker that trigger the corresponding worker to visit the Manager Office, located at the bottom right of the factory, once they are done with their task.
>
> **Note:** A task includes going back to the Task Board to update the product table. If you, the manager, clicks on the "Worker 4" button, that worker should finish making their product and go back to the Task Board before going to the Manager Office.

If their factory looks like this, and clicking Edgar makes Edgar go to the Manager Office only once after he is finished with his task, the student gets full points.



Product	Total	Started	Completed
Cheap Computer	2	0	2
Okay Computer	4	0	4
Good Computer	3	3	0
Amazing Computer	5	3	0
Cheap Server	9	0	0
Amazing Server	2	0	0

Fig. 1: Working exam solution.

2.5%	There is a Manager Office image at the correct location in the factory
0.5%	In Factory.js, in the Factory(factoryData) function there is a call to a method that draws the ManagerOffice received in the factoryData, e.g. this.drawManagerOffice(factoryData.managerOffice);
1.25%	The drawManagerOffice(…) function takes steps to add the ManagerOffice, like parsing the x and y positions or designating the grid cell to append to (see solution code for details). If they do not have a ManagerOffice.js file do not deduct points for not having “new ManagerOffice(cell, managerOffice);”
0.75%	There is a ManagerOffice.js file that takes in a cell and the managerOffice JSON object to be placed on that cell. See solution code for details.

Partial credit points if student missed the above (only totals 10% because it omits a lot of work and is not proper coding practice).

1.0%	There is a Manager Office at the correct location, but it was not read in but instead added directly in the Factory.js file. For example, hardcoding the image, or adding it to the cell in this fashion: var cell = this.simulation.rows[factoryData.width-1].cells[factoryData.height-1]; instead of reading in the position from the object data sent to the frontend.

1.5%	The Manager Office is properly stored in the factory backend
0.2%	There is a ManagerOffice.java or similarly-named class
0.2%	The class extends FactoryObject

0.3%	The class's constructor properly sets its parent's parameters, e.g. super("Manager Office", "ManagerOffice" + Constants.png, x, y);
0.3%	In Factory.java, a ManagerOffice object is instantiated with a line like managerOffice = new ManagerOffice(width-1,height-1); so that the ManagerOffice is always in the bottom right corner.
0.5%	Still in Factory.java, there is a method of some sort that adds this instance to the factory with calls to mFObjects.add(…), mFNodes[…][…], and mFNodeMap.put(…). See the solution code for details.

0.75%	There is a button for each worker displaying the correct name
0.375%	There are the correct number of buttons
0.375%	The buttons have the correct names (e.g. Alana, Bob, etc.)

Partial credit points if student missed the above:

0.5%	In Factory.js, there is a function like "Factory.prototype.drawWorkerButtons" which takes in workers from the factoryData and takes steps to add them to the HTML, e.g. a call to document.getElementById(…), appendChild(…), a for loop, etc. See solution code for details.
0.25%	In a for loop, buttons are dynamically created for each worker, with a call to something like document.createElement(…) or "buttonDiv.innerHTML += '<button …" like in the solution. The button should display worker.name, not something like "Worker " + numWorker.

0.75%	Each button executes an action specific to the index of the worker it belongs to
0.75%	Clicking a worker's button either makes that worker go to the ManagerOffice or prints something to the Chrome DevTools console (console.log(…)) with something like "Pressed Worker 3 button". It must be specific to the worker.

Partial credit points if student missed the above:

0.25%	The buttons have an attribute similar to "onclick = 'Factory.prototype.sendManagerAction(" + i + ")'", where i is the worker index.
0.25%	In the "Factory.prototype.sendManagerAction" or related function, write console.log(index) and test the buttons by clicking on them and checking the console.

0.25%	The "Factory.prototype.sendManagerAction" function has the following code:
	`socket.send(JSON.stringify({`
	`        action: 'MustSeeManager',`
	`        workerNumber: workerNumber`
	`    }));`

1.5%	Backend receives an action when the manager wants to see a worker
0.5%	There is a MustSeeManager.java class (or similar name) that extends Action and defines its execute(…) method.
0.25%	In ActionFactory.java, an Action is added to the actionMap with: actionMap.put("MustSeeManager", new MustSeeManagerAction());
0.75%	If the execute(…) method has a line similar to: int workerIndex = msg.get("workerNumber").getAsInt(); use System.out.print(workerIndex) to check that the backend received the right index from the frontend button.

3.0%	Worker visits Manager Office when appropriate
1.5%	Click on Frank. Frank, at some point, goes to the ManagerOffice.
1.0%	Frank only goes to the ManagerOffice from the task board after he is done with his task. After seeing the manager he returns to the task board and resumes his work.
0.5%	Keep looking at the simulation to be sure that Frank doesn't repeatedly return to the task board, which would happen if the student forgot to set their mustSeeManager boolean back to false.

Partial credit points if student missed the above:

0.5%	MustSeeManager.java calls factory.getWorker(workerIndex).mustSeeManager(); or a similar line to signal to a specific worker that they must see the manager.
0.5%	In FactoryWorker.java, there is some added logic within while loop of the worker's run() method.
1.5%	The following lines were added there: mDestinationNode = mFactory.getNode("Manager Office"); mShortestPath = currentNode.findShortestPath(mDestinationNode); mFactory.sendWorkerMoveToPath(this, mShortestPath);
0.5%	There is some logic to ensure the worker only goes to the ManagerOffice once after being requested, e.g. this.mustSeeManager = false;.