heJEMony
Jenny Han, Eric Li, Matteo Wong
APCS2 pd3 -- ProPro
2017-05-17

# Burger heJEMony

## General Overview

Our project will be a simulation of a fast food empire. You will begin as an employee and eventually be able to rise to the position of manager. Once you are a manager, you will have the ability to build new stores, source your own beef, hire employees, etc. The ultimate goal will be to open up X stores and generate $XXXXXXXXXXXXXXXXXX of revenue (we have not decided on final goals yet).

We will use Processing for visuals and have queues as the main data structure we implement.

## Part I: Service Mini-game

When you begin, you will simply be manning the grill and making burgers. Random orders will be generated with ingredients such as: bun, patty, lettuce, tomato, pickle, etc. These orders will be stored in a queue. With Processing, there will be different "buttons" on the screen that you have to click to cook patties and assemble the burger. When a burger is finished, it will be dequeued, and you will move on to the next one. Each ingredient (patty, lettuce, etc.) will extend Food.java (so Patty is-a Food, Lettuce is-a Food, etc.). They will all have different visual representations, which is why we want all the different classes. The burgers will be stored as a queue of the different types of food (a Stack seems more intuitive but since you build burgers from the bottom up and we would construct the order in the same way, FIFO seems better than FILO).

We will use draw() and keep a counter, since draw() runs 60 times per second. The orders will need to be completed in a certain number of seconds to keep customers happy. The game will end after you have served X customers successfully or after you have failed X number of times (failure constitutes building the wrong burger).

The mini-game will always be available to play, but in the beginning it will be all you are allowed to do. Serving enough customers will increase customer satisfaction and increase profits. Once you have served enough customers, you will be promoted to a manager position, and then phase 2 of the game will begin.

## Part II: Building Your Empire

When you first become a manager, you are able to buy a new store. We will have a class Store.java. We will use draw() again as a timing mechanism to make you earn money as time progresses. The class Empire.java will store everything (it will be your empire). It will have instance variables int _employeeHappiness, int _customerSatisfaction, ArrayList<Store> _stores,

double _budget, int _salary, etc. As you have more customer satisfaction, more customers will come and you can make more money. But you will also need more employees to handle them all, and need to buy more meat, etc. If customers aren't served in a timely manner (ie. the ratio of employees to customers is off) you will start to lose customer satisfaction and make less money.

The game will unlock new features as you reach new milestones. The first milestone will be serving X number of customers as a manager. This will unlock the feature of sourcing your own meat and potatoes (for french fries!). You will be able to choose what type of cow and what percent of your burgers will be artificial additives and preservatives. Thus we will need a int _meatPercent instance variable. More preservatives will be cheaper but make customers less happy.

Once you make a certain amount of money, you will be able to purchase more stores (which will be added to the ArrayList of stores). We will add on other features if we are able to, like buying advertisements or happy meals.

There will also be random elements, like an e coli outbreak, having bad employees that need to be fired (Employee.java class), or strikes if your workers are too unhappy (which could cause you to lose the game).

Every time you do something it will be added to a queue and take a certain amount of time to complete. An optional method might be to make it a priorityQueue and allow the user to decide what should take priority in the queue (stopping an e coli outbreak might matter more than launching a new ad, for instance).

**Problems that can arise**

At any point we will have a random probability of issues arising. The main ones we are thinking of are an e coli outbreak (popularity plummets), a strike (randomly or if workers are too unhappy), and an employee being bad (you have to discipline them in some way).

**Classes**

| Empire |
| --- |
| Private ArrayList<Store> _stores<br>Private int _employeeSalary<br>Private double _budget<br>Private int _totalEmployeeSatisfaction<br>Private int _totalCustomerSatisfaction<br>Private int _percentMeat<br>Private ConcurrentLinkedQueue<String> _actionsList //there will be specific actions you can add that will be added automatically by the computer, such as buying a store, buying more cows, etc, each with an allotted time and will only be dequeued when that ends<br>Private Stack<String> _milestones //they get popped off<br><br>Public void buyStore(Store s, int cost)<br>Public Store seeStore(String name)<br>Public int getEmplSat()<br>Public int getCustSat()<br>Public int budget()<br>Public void setBudget(int amount)<br>Public int getMeatPercent()<br>Public void setMeatPercent() |

| Store |
| --- |
| Private ArrayList<Employee> _employees<br>Private int _customerSatisfaction<br>Private int _employeeSatisfaction<br>Private int _salary<br>Private int _priceBurger<br>Private int _costDailyOperations //will be updated automatically by summing salaries, revenue, etc<br><br>Public void hire(Employee e)<br>Public Employee fire(Employee e)<br>Public void costDaily()<br>Public int getEmplSat()<br>Public int getCustSat()<br>Public int getSalary()<br>Public void setSalary(int s)<br>Public void modEmplSat(int i)<br>Public void modCustSat(int i) |

| |
|---|
| Public int getPrice()<br>Public int setPrice() |

| Employee |
|---|
| Private String _name<br>Private int _satisfaction<br>Boolean _hasAttitude //could lead to needing to fire employees<br><br>Public String getName()<br>Public String getSat()<br>Public String setSat() |

| Farm |
|---|
| Private int numCows<br>Private int costCow<br>Private int acresPotato<br>Private int costPotato<br><br>//accessors + mutators |

| Interface Food |
|---|
| //will have several subclasses. Patty, Lettuce, Tomato, Bun, etc. Each will have it's own representation in Processing. No methods other than the patty |

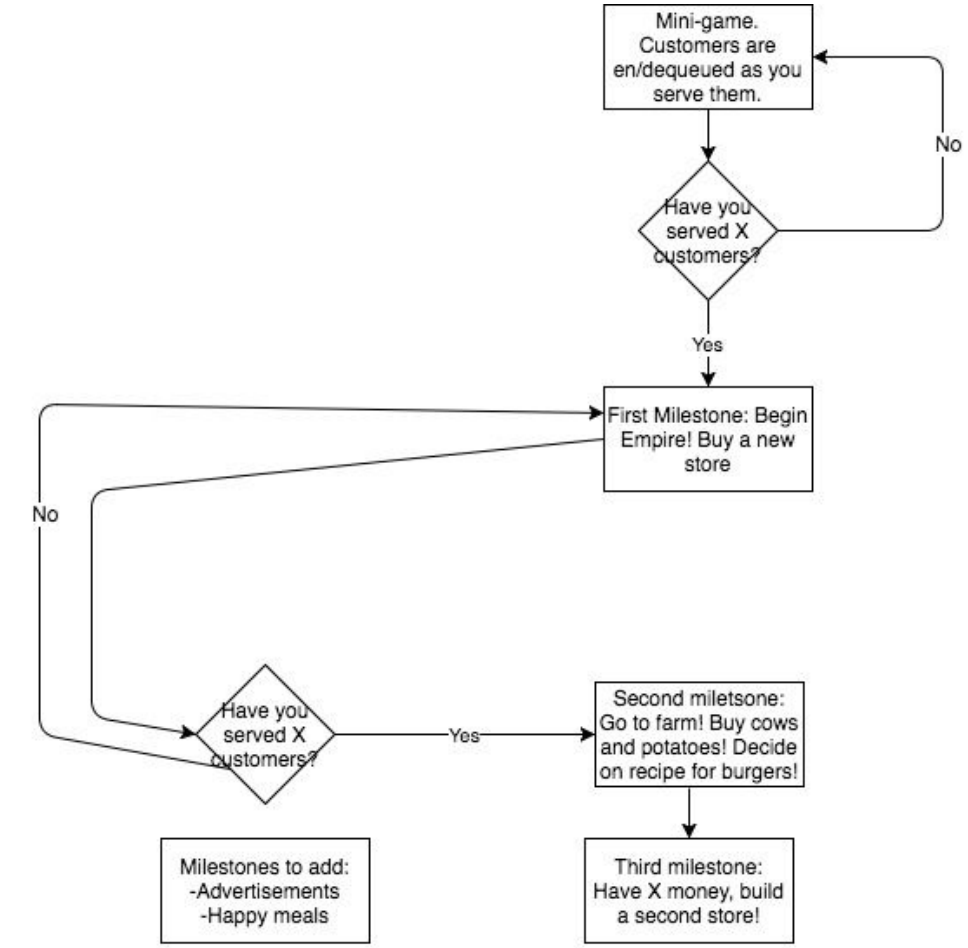
| Patty implements Food |
|---|
| Boolean isCooked<br><br>Public void cook() |

| Order |
|---|
| Private ConcurrentLinkedQueue<Food> _order |

Public void addToOrder()
Public Food constructOrder()

Mini-game

```
ALDeque<Order> _orders=new ALDeque<Order>();
ArrayList<Integer> burgerTimes = new ArrayList <Integer>();
int time=0;//time variable
int y=170;//for printing orders
Order currOrder = new Order();
int placeForFood=500;//where you add food to
int currOrdNum=1;//which order currently on
float cash=0;


int realTime=0;
int counterT=0;

PImage tomato;
PImage lettuce;
PImage patty;
PImage bun;
PImage cheese;


void setupMinigame()

void timeP()

void mouseClicked()

void buttons()

void loadOrders()

void drawButtons()

void checkOrder()
```

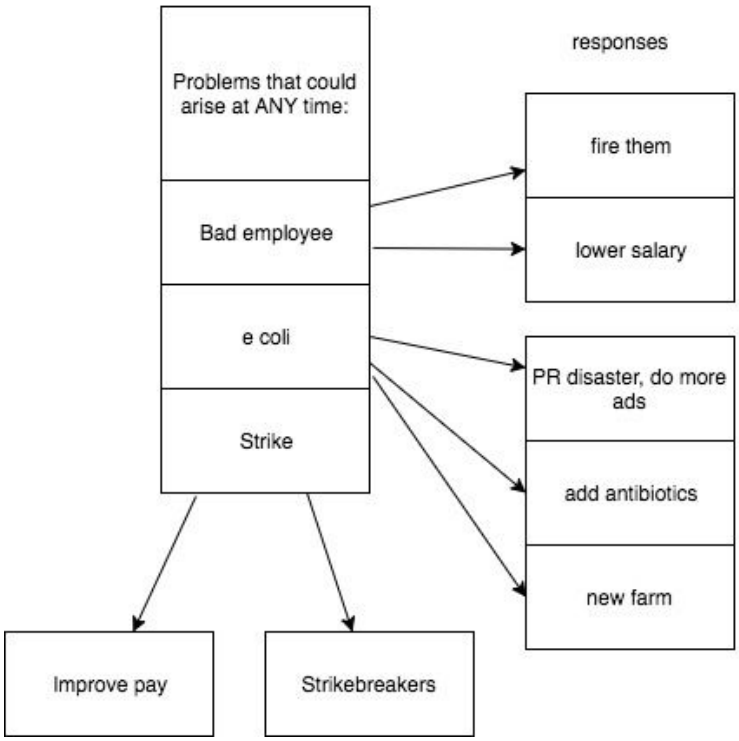


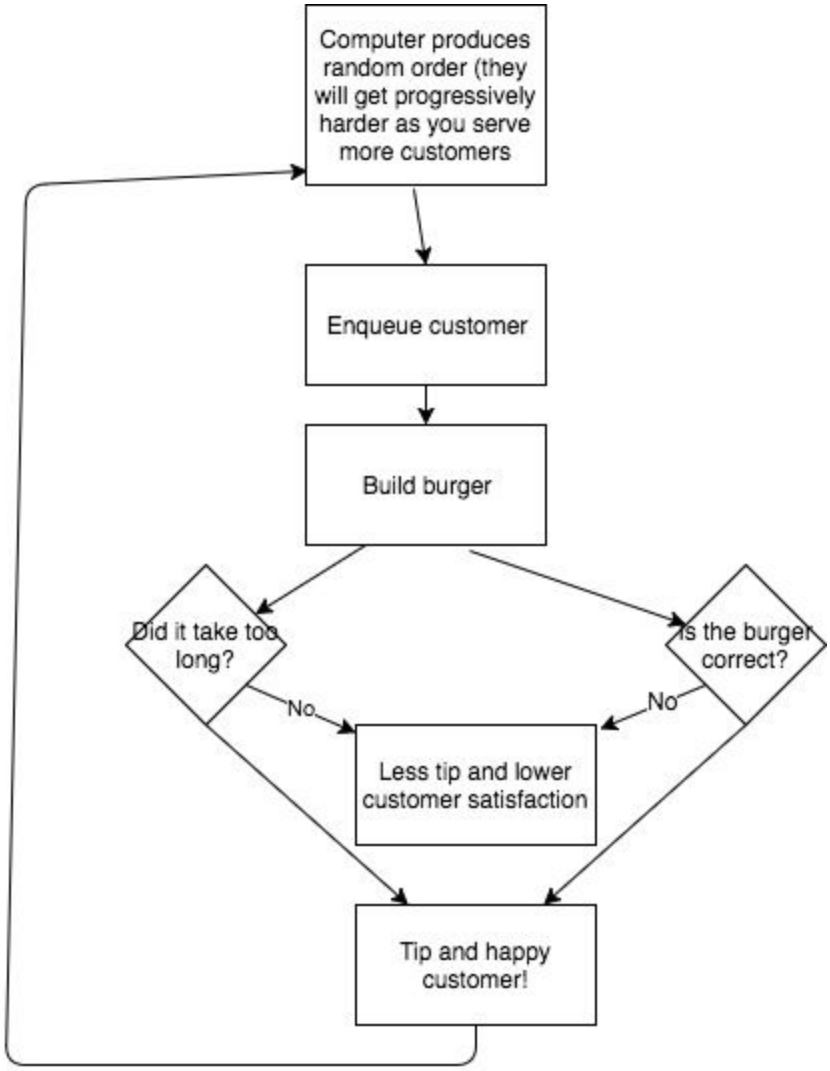

Stuff You can do as of Milestone I

| |
|---|
| Change Prices |
| Change Salaries |
| Mini-game |
| Hire/fire Employees |

Now you can also:

| |
|---|
| Mess with recipe |
| Purchase cows, potatoes |




Computer produces random order (they will get progressively harder as you serve more customers
Enqueue customer
Build burger
Did it take too long?
Is the burger correct?
No
No
Less tip and lower customer satisfaction
Tip and happy customer!