MottSt APCS2
Period 5
Brooke Jin, Bryan Chan, Jennifer Yu
2017-05-16

Final Project Proposal
Panda House

Abstract

Similar to the popular game "Diner Dash," Panda House is a restaurant game where the user, playing from the point of view of a waiter, tries to serve demanding customers in the famous Panda House restaurant located on Mott Street, Chinatown, NYC. This unique game features one primary objective: to serve customers satisfactorily before they get upset. The command line version of the game will consist of a table of customers being displayed and an ASCII representation of the layout of the restaurant. The Processing version of the restaurant will include images and more advanced graphics for an enhanced user experience. The game will progressively become faster after each round of service, with the restaurant getting busier and busier. It is possible for the player, by making mistakes or neglecting the customer, to make a customer angry enough to leave the restaurant. During the course of the game, the player accrues points for each customer served, with bonus points for speed and a percentage of received tips. In order to proceed to the next round, the player must gain enough points specified by the goal.

Mechanics

The user is playing in the point of view of a waiter. The waiter will seat customers at tables, take down orders, bring out food to the correct customers, and deliver the bill. The waiter will receive points based on what they do correctly and be deducted points if they do something incorrectly. Additionally, mistakes will cause a customer's satisfaction meter to go down. If the meter hits zero or below, then the customer will leave the restaurant.

Seating customers:

New reservations are added to a "Reservations" Priority Heap with each reservation denoting a certain priority level (e.g., a celebrity would have a higher priority). When customers arrive, they will be placed on a "Waiting" Priority Heap where those with reservations are prioritized. The waiter will bring the customers in the waiting list to their reserved table, or to an unreserved table with appropriate space.

Taking down orders and bringing out food:

When a table is ready to order, the waiter will record the order and add it to an "Order" ArrayList and the "Pending Food" Deque, adding fast orders such as drinks to the front of the deque to be processed first, and the other, slower, orders to the end of the deque. However, the waiter retains the freedom to add a dish either to the front or end of the deque; they just have to make sure the customer stays happy.

The chef will add completed food items to a "Finished Food" Queue, and the waiter will deliver the next item in the queue to the correct table. Food items take different amounts of time to prepare, but will be listed in three main categories: small, medium and large. Small dishes include items such as beverages, salads, and soups. Medium dishes involve desserts, sandwiches, and general breakfast/lunch items. Large dishes consist of main courses and dinner meals.

Delivering the bill:

Customers will ask for a bill after they have finished their meal. The waiter can refer to the "Order" ArrayList to retrieve the customer's order and write up a bill totaling the cost. Based on the size of the party, a suggested tip will also be included in the writing up of the bill. The tip will also be based on the customer's satisfaction meter. The size of the tip is important, as a percentage of it will become the player's bonus points at the end of each round of service.

Points System:

The user is given points doing tasks correctly. These tasks include:

- Seating customers at their correct tables
- Prioritizing customers with reservations, and making sure their table is available when they arrive
- Taking and bringing out food orders in a timely fashion
    - Drinks and appetizers should be brought out first, then the meal, then dessert.
- Giving the bill to the correct table at the end

Points will be deducted for any task done incorrectly, and the customers' satisfaction meters will also decrease if mistakes are made.

Graphics

The user will being playing in the point of view of a waiter. They will input commands via the command line. The user will be presented with choices such as bringing a table to a customer, serving food, taking orders, or clearing the table and getting the bill. Each of these interactions will lead to other interactions that determine the amount of points the user receives. There is a money counter to show how much money is earned, and a goal for the day. The player can type in commands to see the current status of waiting customers and table space, food orders, bills, and customers' satisfaction levels. The cook will also display a message when a new dish is ready to be served.

Vanilla Version

Yes, the game sounds rather complicated so far. However, we plan to develop the game incrementally, starting from the very basics and adding on new features as we go along. The vanilla version of "Panda House" will include these basic components:

Customer	Order

`private int table#`
`ArrayList<String> dishes = …`
[initialize a collection of dishes]
`private ArrayList<Order> orders`
---- 	
`Customer()`: adds random items from `dishes` to `orders`
 	
`void addOrder(Order newOrder)`: adds a dish to `orders`
 	
`Order removeOrder()`: removes a dish that has been served from `orders`
`ArrayList<String> getOrders()`: returns the ArrayList orders
 	
`boolean equals(Customer c)`: returns if the `table#`s are equal
	`private String name`
	`private int table#`
	---- 	
	`Order()`: defaults name to null and table# to 0.
	
	`Order(String s, int num)`: sets the variable `name` to `s` and `table#` to `num`
	
	`String getName()`: returns the name of the dish being ordered
	
	`int getTable()`: returns the table number that placed the order
	
	`void setTable(int t)`: sets the table number to the parameter

Waiter	**Restaurant**

`private ArrayList<Customer> customers`
`private ArrayList<Integer> tables`
---- 	
`Waiter(ArrayList<Customer> c)`: creates a waiter with a list of customers
 	
`void assignTable(Customer c)`: Assigns a table to the next customer by removing one of the values from the ArrayList `tables`
 	
`void addCustomer()`: adds a
	`private int points`
	`private int goal`
	`private int level`
	`Waiter w`
	`ArrayList<Customer> clientList`
	---- 	
	`Restaurant(int levelNum)`: fills `clientList`,`goal = level * 6`, `clientList.size() = goal + 5` (each customer = 1 point)
	
	`void addPoints(int p)`:adds p points to the total points
	
	`boolean hasCust()`: returns `true` if

customer to the `customers` ArrayList
 	
`void serve(Customer c, Order o)`: serves the order `o` to customer `c`, which invokes `removeOrder()` of `Customer`
 	
`void removeCustomer(Customer c)`: removes the customer specified from `customers`, adds to `tables`
	there are still customers in `clientList`, false otherwise
	
	Main method: create a `Restaurant` and a `Waiter`, and while there are still customers to serve, a waiter serves them.
	*runs more like a simulation, commands come in later versions.

Solidifying and Showcasing

We will implement various topics learned in AP Computer Science in the game.

- Priority Heap
    - Reservations List -- customers will be added to the reservations list based on how important they are (ie. VIP or whether they have a reservation).
    - Waiting List -- customers arriving will be added to the waiting list, where customers with reservations get priority.
- Deque
    - Pending Food List -- each new order can be added to the front or back of the list based on how long it will take the chef to cook the food.
- Queue
    - Finished Food List -- each completed order will be added to the queue, and the waiter will deliver the next order in the queue to the customers.

Necessarily, we would need some mechanism for keeping track of elapsed time. This mechanism would be utilized for calculating how long a customer waits before her/his mood begins to decrease. Mood of the customer will determine how many points the waiter receives after the customer has received all her/his orders, or if the mood drops low enough, will cause the customer to leave the restaurant entirely.

- System.nanoTime() should be sufficient to execute reach this goal.

Stretching

We may stretch the project by:

- Making a GUI for the project via Processing.
- Different customers may have different personalities. Each customer can have a unique trait in that they might leave faster than others if not seated.
- A more reality-based game where the player would be able to upgrade their restaurant and accrue wealth and become a capitalist monopoly.