Team Name: On The Train Of
Team Members: Joelle Lum, Nikita Borisov, Dimitriy Leksanov
Period 1

Diner Dash

Similar to the original Diner Dash game, users will play as the waiter of a diner. They will be expected to drag customers from the entrance onto tables to seat them. Once customers are seated, their tables will have a timers, indicating their patience. Then, every time the customers need service, the user will click a table to tell the waiter to go to it. When the waiter gets close enough to the table, the waiter will automatically serve the customer. The timers will restart every time the waiter interacts with the customerasdfasdfasdfasdfasdfasdfasdfasdfasdfasdfasdfasdfasdfsdafs (take their order, serve them food, give the check, etc). If the waiter fails to serve customer(s) before their timer(s) run(s) out, the customer(s) will leave. If the waiter collects enough money before time runs out, the user wins and the level ends. The user loses if he/she fails to reach the required monetary goal before the store closes.

Classes:

Level:

- Int difficulty (1->5)
- Waiter user
- Table t0
- Table t1
- Table t2
- Table t3
- Final int duration = 2 (minutes)
- Int secondsPassed : if secondsPassed == duration, level is over
- APQ sideBar with customers' ids
- ArrayList of checks
- Chef boyardee
- Stack : composed of timeStamp, user's xcor/ycor,
- Int monetaryGoal
- Int moneyEarned

+ insertCustomer(int num)
+ increaseTime(secondsPassed)

- APQ checks

Waiter

- Int Xcor

- Int Ycor
- Food order
- Food inHands
- Check c
- Int[] tableWaitedRecord
- move(xcor, ycor)
- pickUpFood() : picks up food from kitchen and holds onto it
- takeOrder(Customer) : takes customer(s) order(s)
- giveCheck(Check) : gives check to chef and adds foods to chef's orders
- giveCheck(Check, Customer) : gives check to customer
- takeMoney() : collects money from customer and removes check from APQ of checks
- redo()

Customer

- Final Food[] menu
- String description (such as "businessman", "food critic", "mother", etc)
- Int id
- Int partyOf
- Int timerSec : patience duration that correlates to description
- Int sec : amount of seconds since last interaction with user
- Food desiredFood
+ askToOrder() : indicates to waiter that he/she wants to order
+ order() : randomly picks from menu
+ askToPay() : indicates to waiter that he/she wants his/her check
+ pay() : gives money to waiter and "leaves"

Food

- Int price
- String description (such as "cake", "pasta", "rice", etc)
- Int timeToMake

Chef

- APQ of food named orders
- presentFood(food) : places food on counter for waiter to serve according to each food's timeToMake, removes that food from APQ of orders

Table

- Int numSeats

- Int color
- AL of customers (customer's party size can be < numSeats)

Check
- Food[] order
- Int total
- Int customerID

Array Priority Queue:
Customers are prioritized into an APQ by the order they are seated. This APQ will be visible to users as a sidebar.

Another APQ is needed to keep track of the foods the chef has to make. Foods are prioritized by the order in which the waiter tells the chef what to make.

Stack:
The stack will allow users to "undo" (with penalty). By storing an array of different table values (time, color, xcor, ycor, etc.), a timestamp, in essence, can be created for each table that is waited. That way, it is easy to keep track of all of the data during a scene in the game when a "move" happens. Using that, a stack of timestamps can be created, storing all records of "moves" made from the beginning of the game until the current moment. By use of an "Undo Move" button, we can pop the top of the stack use that as a base to revert the game back to a previous state, before the most recent move. This could be a prominent feature in early levels, with the price steadily rising as the game continues.

Doubly-linked list
Though there are many possible spots for a linked list, the easiest place to implement a doubly-linked list in the case of these project would be in the opening menu. By creating a list that stores String values, representing the titles and descriptions of various levels, we would have a "scene selection"-style level chooser. Each interface in the menu would be a different node, and the user would be able to toggle through by clicking certain buttons (arrows pointing left and right). This would traverse the DLL one-by-one, each time displaying the current screen.

ArrayList
Checks will be kept in an ArrayList, and whenever the waiter receives money from customer(s), the check will be removed from the ArrayList