CMPN102

Data Structures and Algorithms

Cairo University

Faculty of Engineering

Restaurant Management Project Final Assessment

Function: Simulation ()

Member of: Class Restaurant

Inputs:

- Mod: the integer that indicate the program mod according the user input

Returns:

- No return
  But use the ofstream to save the data in txt file

Called by:

- Restaurant::RunSimulation ();

Calls:

- Restaurant::checkCooksInBreakAndRest (CurrentTimeStep);
- Restaurant::checkBusyCooks (CurrentTimeStep)
- Restaurant::incrementOrdersWaitTime ();
- Restaurant::checkUrgentOrders ();
- Restaurant::checkAutoPromotion ();
- Restaurant::serviceUrgentOrder (CurrentTimeStep)
- Restaurant::ServiceVipOrder (CurrentTimeStep);
- Restaurant::serviceNormalOrder (CurrentTimeStep);
- Restaurant::serviceVeganOrder(CurrentTimeStep);
- Restaurant::injureCook ();
- Restaurant::FILLDrawingList ()
- Restaurant::ExecuteEvents (CurrentTimeStep);
- Restaurant::LoadingFunction(ifstream & input);
- Restaurant::outputFunction(ofstream & output);
- GUI::waitforclick()
- GUI::GetString ();
- GUI::updateInterface()
- GUI::ResetDrawinglist()
- GUI::PrintMessage ()

Function logic description:

This is the simulation for the all program that call the functions in the right sequence in the same time control the GUI interference of the program also take data from user , and do three modes of interference (interactive, silent, step-by-step)in the end save the data by calling the output function

First take the name of the file of the test case. Then, enter while loop end when there is no events or orders remain, In this loop start to execute events then move the orders through the queues and lists

Function: LoadingFunction

Member of: Class Restaurant

Inputs:

- Variable of type ifstream passed by reference

Returns:

- Doesn't return anything (void)

Called by:

- Restaurant::Simple_simulator()
- Restaurant::Simulation(int mod)

Calls:

- Cook::setType(COOK_TYPE t)
- rand() //including the library: #include <cstdlib>
- srand(time(NULL) //including the library: #include <time.h>
- Cook :: setBreakDuration (int bd)
- Cook::setID (int id)
- Cook :: setSpeed (int s)
- Restaurant :: addAvailableCook(Cook* avCook)
- Queue<T>::enqueue(const T& newEntry)
- GUI::PrintMessage(string msg) const
- GUI::waitForClick() const

Function logic description:

This function takes from the user a text file filled with data and inputs this data to our system. The new change in phase 2 is that all different cooks have different speeds. Minimum and maximum speed are given for every cook as well as break time ranges for every cook. A random number is generated for every cook speed and break using the rand function. And the srand function is used to change the random numbers every time we run our project (if the srand function is not used, if for example we get the random numbers: 1 3 6 9 2 the first time we run our project, then when we try it the second time the numbers will still be: 1 3 6 9 2). In phase 2 we also added the input of the injProp (injured probability) as well as the RstPrd (rest period).

Function: OutputFunction

Member of: Class Restaurant

Inputs:

- Variable of type ofstream passed by reference

Returns:

- Doesn't return anything (void)

Called by:

- Restaurant::Simulation(int mod)

Calls:

- Queue<T>::toArray(int& count)
- Queue<T>:: peekFront(T& frntEntry) const
- Order :: getFinishTime () const
- Order::GetID() const
- Order :: getArrTime () const
- Order :: getWaitTime () const
- Order :: getServTime () const
- Queue<T>:: dequeue(T& frntEntry)
- SortedList<T> :: getLength () const
- Restaurant :: getInjuredCount () const
- Restaurant::getUrgentCount() const
- Restaurant::getPromotionCount() const

Function logic description:

This function is used to output the required data to an external text file. To get the number of total orders, I used the function toArray and using the finished list (as it contains all the orders) to get the number of total orders. And to get the number of orders for each type (normal,vegan,VIP) I used a for loop using the total number of orders to peak front of the finished list and get all the times required (finished time, arrival time, wait time, serve time and ID). And then using the GetType function, I check if the type is normal then make a counter for it and +1 and so on for vegan and VIP. To get the Total wait and Total serve, we add the total wait to the waiting time and serve time respectively. We then dequeue the finished list at the end of the for loop to go through the entire list. To get the total number of cooks, we use the function GetLength with the normal cooks list, vegan cooks list and VIP cooks list and we add them all together to get the total number of cooks. To get the average wait and average serve, we divide the TotalWait by

the total number of orders and the TotalServe divided by the total number of orders respectively. And to get the auto promoted percentage, we divide the AutoPromotedCount by the total number of orders and then multiply the outcome by 100 to get the percentage.

Function: checkAutoPromotion

Member of: class Restaurant

Inputs:

- none

Returns:

- nothing (void)

Called by:

- Restaurant::simulation(int mod)

Calls:

- UnsortedList::isEmpty()
- UnsortedList::getLength()
- UnsortedList::getEntry(int position, Order *& anEntry)
- Order::getWaitTime()
- UnsortedList::remove(int position)
- Order::setType(ORD_TYPE t)
- Restaurant::Add_Order(Order* o)

Function logic description:

The function loops on the waiting normal orders and searches for any orders that waited longer than autoP. If there is, then it changes it to a VIP order, removes it from the waiting normal orders adds it to the waiting VIP orders list

Function : checkUrgentOrders

Member of: Class Restaurant

Inputs:

- None

Returns:

- Nothing (void)

Called by:

- Restaurant::Simulation(int mod)

Calls:

- SortedList::isEmpty()
- SortedList::getLength()
- SortedList::getEntry(int position, Order*& anEntry)
- PriorityData::getData()
- Order::getWaitTime()
- SortedList::remove(int position)
- Queue::enqueue(Order* const& newEntry)

Function Logic Description:

This function loops on the VIP waiting list and searches for any order that waited longer than VIP_WT. if there is, then it changes it to an urgent order and adds it to the waiting urgent orders list and removes it from the waiting VIP orders.

Function: ServiceUrgentOrder

Member of: Class Restaurant

Inputs:

- Current time step

Returns:

- Nothing (void)

Called by:

- Restaurant::Simulation

Calls:

- Queue::isEmpty()
- Queue::toArray(int count)
- Queue::PeekFront(Order *& anEntry)
- SortedList::getLength()
- SortedList::getEntry(int position, Order *& anEntry)
- Restaurant::assigncook(Cook*cook. Order *order, int currentTimeStep)

- PriorityData::getData()
- Restaurant::cooksOnBreakAlternative(Order* UrgentOrder, int currentTimeStep)
- Restaurant::cooksInRestAlternative(Order* UrgentOrder, int currentTimeStep)

Function Logic Description:

This function is responsible for assigning every waiting urgent order to a cook if possible by calling function assigncook. It searches for a cook that can take this size of order. First it loops on available VIP cooks list for a VIP cook that can take this order until one is found. If this failed, then it loops on available normal cooks until one is found. If this also failed, then it loops on available vegan cooks until one is found. If it failed, then it searches for the cooks that are on break by calling function CooksOnBreakAlternative. Then if this also fails, then it searches for the cooks that are in rest by calling function CooksInRestAlternative. Failing to assign a cook to this order means that the size of the order is bigger than what any cook can take before his break or that there is no available or on break or in rest cook. Then it moves onto the next urgent order until all the orders are checked.

Function: ServiceVipOrder

Member of: Class Restaurant

Inputs:

- Current time step

Returns:

- Nothing (void)

Called by:

- Restaurant::Simulation()

Calls:

- SortedList::isEmpty()
- SortedList::getLength()
- SortedList::getEntry(int position, Order *& anEntry)
- Restaurant::assigncook(Cook*cook. Order *order, int currentTimeStep)
- PriorityData::getData()

Function Logic Description:

This function is responsible for assigning every waiting VIP order to a cook if possible by calling function assigncook. First it makes a loop on available VIP

cook until one is found. If no cook is found, then it loops on available normal cooks until one is found. If no cook is found, it loops on available vegan cooks until one is found. If no cook is found, this means that no cook is available or the size of the order is bigger than what the cooks need to take a break. Then it moves onto the next VIP order and does the same thing until all the orders are checked.

Function: ServiceNormalOrder

Member of: Class Restaurant

Inputs:

- current time step

Returns:

- Nothing (void)

Called by:

- Restaurant::Simulation()

Calls:

- UnsortedList::isEmpty()
- UnsortedList::getLength()
- UnsortedList::getEntry(int position, Order *&anEntry)
- SortedList::getLength()
- SortedList::getEntry(int position, Order *& anEntry)
- Restaurant::assigncook(Cook*cook. Order *order, int currentTimeStep)
- PriorityData::getData()

Function logic description:

This function is responsible for assigning every normal order to a cook if possible by calling function assigncook. First, it loops on available normal cooks until one is found. If no cook is found, then it loops on available VIP cooks until one is found. If no cook is found then this means that no normal or VIP cooks are available or that the size of the order is bigger than what any normal or VIP cook can take before taking a break and moves onto the next order until all the orders are checked.

Function: ServiceVeganOrder

Member of: Class Restaurant

Inputs:

- Current time step

Returns:

- Nothing (void)

Called by:

- Restaurant::Simulation(int mod)

Calls:

- Queue::isEmpty()
- Queue::toArray(int count)
- Queue::peekFront(Order* order)
- SortedList::getLength()
- SortedList::getEntry(int position, Order *& anEntry)
- Restaurant::assigncook(Cook*cook. Order *order, int currentTimeStep)
- PriorityData::getData()

Function logic description:

This function is responsible for assigning every waiting vegan order to a cook if possible. It loops on available vegan cooks and calls function assigncook until one is found. If no cook is found this means that there are no vegan cooks available or that that order size is bigger than what any vegan cook can take before taking a break and the function moves onto the next order until all of the vegan list is checked.

Function 6: ServicedOrders

Member of: Class Restaurant

Inputs:

- Pointer to order and current time step

Returns:

- Nothing (void)

Called by:

- Restaurant::assigncook(int mod)

Calls:

- Order::getType()
- SortedList::remove(Order* const& anEntry)
- Queue::dequeue(Order* order)
- UnsortedList::getLength()
- UnsortedList::getEntry(int position, Order *& anEntry)
- Order::getID()
- UnsortedList::remove(int position)
- Order::setStatus(ORD_STATUS s)
- Order::setWaitingTime(int WT)
- UnsortedList::insert(int newposition, Order* const& newEntry)

Function logic description:

This function is responsible for adding an order to the serviced orders list and changing what is necessary. It takes a pointer or an order. First it checks if this is order is VIP, if yes then it tries to remove it from the waiting VIP orders. If the removal process failed then this means that this order is an urgent order since they are both of type VIP and proceeds to remove it from the urgent orders list by calling function dequeue. Then it checks if this order is normal, if yes, then it loops on every waiting normal order until it finds the order with the same ID and proceeds to remove it from the list. If the order is not normal nor VIP then it is vegan so it proceeds to delete it from the vegan queue.

Then it changes the status of the order to SRV and sets its waiting time with the current time step minus arrival time of the order. Then it inserts it to the in service list at the end of the list.

Function: addAvailableCook

Member of: Class Restaurant

Inputs:

- avCook: a pointer of type Cook*, the cook to be added to the suitable list of available cooks

Returns:

- Nothing (void)

Called by:

- Restaurant :: checkBusyCooks
- Restaurant :: checkCooksInBreakAndRest
- Restaurant :: LoadingFunction

Calls:

- Cook :: getSpeed
- Cook :: getBreakDuration
- Cook :: getType
- SortedList :: insertSortedDescendingly

Function Logic Description:

This function places the available cook properly in his/her corresponding available list of cooks. First, the cook's productivity is calculated by the equation: cook speed - cook break duration, as to take more advantage of the cooks who finish a big number of dishes with taking only a small break. Therefore, we want the difference between both parameters to be large as possible. Secondly, the cook is added inside an object of PriorityData containing the cook pointer and the integer representing his/her productivity. Lastly, according to the cook type, whether vip, normal or vegan, the PriorityData object will be added to its equivalent sorted list of available cooks which are sorted descendingly according to cook productivity in order to always put the more productive cook or cooks at the front of the list as they are more recommended to use so as to increase the efficiency of the restaurant.

Function 2: assignCook

Member of: Class Restaurant

Inputs:

- cook: pointer of type Cook*, the cook to be assigned with an order
- order: order of type Order*, the order needed to be assigned to the cook
- currentTimestep: an integer indicating the current time step in which the simulation of the restaurant is working

Returns: Boolean

- true: if a suitable cook was assigned
- false: if a suitable cook was not found

Called by:

- Restaurant :: cooksOnBreakAlternative
- Restaurant :: cooksInRestAlternative
- Restaurant :: serviceUrgentOrder
- Restaurant :: serviceVipOrder
- Restaurant :: serviceNormalOrder
- Restaurant :: serviceVeganOrder

Calls:

- Order :: getSize
- Restaurant :: getBreakNumber
- Cook :: getFinishedDishesCount
- Cook :: getStatus
- Cook :: GetType
- SortedList :: remove
- Cook :: setStatus
- Cook :: setAssignedOrder
- Cook :: setFinishedDishesCount
- Restaurant :: servicedOrders
- Cook :: getSpeed
- SortedList :: insertSortedAscendingly

Function Logic Description:

This function assigns the given order to the given cook, if possible, and changes what is necessary. It first checks if the number of dishes in this order is less or equal the number of dishes this cook needs to prepare before taking a break, if it is more, this cook cannot be assigned as we cannot divide an order between two cooks and have two pointers pointing at the same object. If the given cook satisfies the condition, he/she will be removed from his/her corresponding sorted list of available cooks according to the cook type, if and only if the cook's status is actually available, neither on break nor in rest. Secondly, some of the cook's data members are changed accordingly: his/her status to busy, the pointer of the assigned order and the number of the dishes of this order will be added to the count of his/her finished orders to know when he/she will need a break. Function servicedOrders is called which changes the parameters of the order to be serviced. Finally, the time in which the cook will finish this order is calculated according to his/her speed, the number of dishes in the order and the current time step, is then put along with the cook pointer in an object of PriorityData and lastly, this object added in the sorted list of busy cooks, which is sorted ascendingly according to the time in which the cooks will finish their orders. This will facilitate the search of the cooks who finished their orders at a certain time step as they would all be accumulated at the beginning of the list.

Function: finishedOrders

Member of: Class Restaurant

Inputs:

- order: pointer of class Order*, the order whose servicing has finished or has been cooked
- currentTimestep: an integer indicating the current time step in which the simulation of the restaurant is working

Returns:

- Nothing (void)

Called by:

- Restaurant :: checkBusyCooks

Calls:

- Order :: setServiceTime
- Order :: getArrTime
- Order :: getWaitTime
- Order :: updateFinishTime
- Order :: setStatus
- UnsortedList :: getLength
- UnsortedList :: getEntry
- UnsortedList :: remove
- Queue :: enqueue

Function Logic Description:

This function changes the necessary parameters specific to the order whose cooking has been finished. Firstly, it sets the service time of the order according to the current time step, the order's arrival time and its waiting time. Secondly, it calculates and sets the order's finish time after the service time has been known and changes its status to done. Thirdly, the order has to be removed from the list orders still in service, which is a list not sorted according to a specific parameter but each order is added one after another according to the beginning of its servicing ascendingly. Therefore, its remove function removes the element at a certain given position in the list, unlike the remove function of the sorted list which removes the given element directly wherever it is in the list. The cooked order is then found and removed, and is lastly added to the queue of finished orders.

Function: checkBusyCooks

Member of: Class Restaurant

Inputs:

- currentTimestep: an integer indicating the current time step in which the simulation of the restaurant is working

Returns:

- Nothing (void)

Called by:

- Restaurant :: Simulation

Calls:

- SortedList :: isEmpty
- SortedList :: getEntry
- PriorityData :: getPriority
- PriorityData :: getData
- Sorted List :: remove
- Restaurant :: finishedOrders
- Cook :: getAssignedOrder
- Cook :: setAssignedOrder
- Cook :: getFinishedDishesCount
- Restaurant :: getBreakNumber
- Cook :: setFinishedDishesCount
- Cook :: setStatus
- Cook :: getInjuredThisTime
- Cook :: setSpeed
- Cook :: getSpeed
- Cook :: setInjuredThisTime
- Cook :: getBreakDuration
- SortedList :: insertSortedAscendingly
- Restaurant :: addAvaialbleCook
- Cook :: setInjuredThisTime

Function Logic Description:

The function generally checks the sorted list of busy cooks for any cooks who finished their orders, if it is not empty, and guides them to their destinations properly. Firstly, the PriorityData object containing the first cook in the list is copied

into an object of type PriorityData and its priority, which is the time of finishing the order, is checked if it equals the current time step. If it does, the cook will be removed from the list of busy cooks, the finished order will be sent to the finishedOrders function to change its parameters and the assignedOrder parameter of the free cook will be re-set to null. Secondly, the cook will be checked if the count of the dishes he/she finished equals the required amount before needing a break, and if it does, it is time for the cook's break whether he/she is injured or not. In that case, this counter is re-set to zero and the cook status is changed to on break, and if the cook is injured, he/she re-gains his/her original speed. After that, the time in which the cook will finish his/her break is calculated according to the current time step and his/her break duration, and the result is put as the priority along with the finished cook pointer in an object of PriorityData and is placed in the sorted list of the cooks taking a break, which is sorted ascendingly according to this priority to facilitate the search for the cooks who finished their break as they would also accumulate at the beginning of the list. If it is not time for the cook's break and he/she is not injured, his/her status will be changed to available and will be sent to the suitable list of available cooks according to his/her type through addAvailableCook function. In the case that the cook is injured and it is not time for his/her break yet, he/she will re-gain his/her original speed and have his/her status changed to in rest. Then, the time in which the cook finishes his/her rest is calculated according to the current time step and the general rest period , is put as the priority along with the injured cook in an object of PriorityData and the object will be placed in the sorted list of resting cooks to facilitate the process of the search of the cooks who finished their rest at a certain time step.

Function: injureCook

Member of: Class Restaurant

Inputs:

- Nothing (void)

Returns:

- Nothing (void)

Called by:

- Restaurant :: Simulation

Calls:

- SortedList :: isEmpty
- SortedList :: getEntry

- PriorityData :: getData
- Cook :: getInjuredThisTime
- Cook :: setInjuredThisTime
- Cook :: getInjuredBefore
- Cook :: setInjuredBefore
- SortedList :: remove
- PriorityData :: setPriority
- PriorityData :: getPriority
- Cook :: setSpeed
- Cook :: getSpeed
- SortedList :: insertSortedAscendingly

Function Logic Description:

At a certain probability, this function injures the first healthy cook in the list of busy cooks, if any, and changes what is necessary. At first, the function generates a random number and checks whether it is equal or less than the injuring probability. If it is, we copy the first cook in the list in an object of type PriorityData through getEntry function of sorted list. If this certain cook is already injured, we look for the next healthy, if found, and put him/her instead in the same object of PriorityData. If found, injure him/her by setting the injuredThisTime data member with true and if he/she wasn't injured before, count him/her in the count of injured cooks as a cook can get injured more than once throughout the simulation. The injured cook's speed then decreases to half while cooking, so the cook is removed from the list and have his/her priority doubled, which is the time step at which the cook will finish the assigned order. This which is equivalent to having his/her speed halved. Lastly, he/she is inserted again and re-positioned according to his/her new priority

Function: checksCooksInBreakAndRest

Member of: Class Restaurant

Inputs:

- currentTimestep: an integer indicating the current time step in which the simulation of the restaurant is working

Returns:

- Nothing (void)

Called by:

- Restaurant :: Simulation

Calls:

- SortedList :: isEmpty
- SortedList :: getEntry
- PriorityData :: getPriority
- PriorityData :: getData
- SortedList :: remove
- Cook :: setStatus
- Cook :: getAssignedInRest
- Cook :: setSpeed
- Cook :: getSpeed
- Cook :: setAssignedInRest
- Restaurant :: addAvailableCook

Function Logic Description:

This function checks the lists of cooks who are on break and those in rest after an injury. For those on break, if any, the first cook in the list is copied in an object of type PriorityData and his/her priority is checked, which represents here the time in which he/she will finish his break. If his/her break is over, he/she is removed from the list of on break cooks and get his/her status changed to available. In this case, if the cook was assigned before to an urgent order while in rest and had his/her speed then decreased to half, which appears in assignedInRest data member of the cook, his/her speed will be doubled and re-gained and this data member returns false again. At the end, the cook is returned to the suitable list of available cooks according to his/her type and the next cook in the list is checked. For those in rest, the first cook is also copied in an another object of type PriorityData, and have his/her priority checked, which represents here the time in which the injured cook will finish his/her rest. If his/her rest is finished, he/she is removed from the list, have his/her status changed to available and is returned to the list of available cooks. Then, the next cook in the list is checked.

Function: cooksOnBreakAlternative

Member of: Class Restaurant

Inputs:

- urgentOrder: a pointer of class Order*, the urgent order required to be assigned
- currentTimestep: an integer indicating the current time step in which the simulation of the restaurant is working

Returns: Boolean

- true: if a suitable cook from those on break was assigned with the urgent order
- false: the urgent order was not assigned because the list was empty

Called by:

- Restaurant :: serviceUrgentOrder

Calls:

- SortedList :: isEmpty
- SortedList :: getLength
- SortedList :: getEntry
- PriorityData :: getData
- Cook :: getType
- SortedList :: remove
- Restaurant :: assignCook

Function Logic Description:

This function is required to assign the urgent order to a cook from those on break, if any, if there is none available. If there are cooks in the list, the priority goes first to the VIP cooks, then the normals and lastly the vegans, according to the rules of assignment of the VIP orders. Therefore, we search for the first VIP cook in the list, if not found, we search for the first normal one and at the end for the first vegan. If one is found, we remove him/her from the list of on break cooks and assign him/her with the urgent order through assignCook function. If none was found, that means the list is empty and false is returned. It is also worthy to note that if the assigned cook was injured before and was assigned while in rest, he/she still will not regain his/her speed because he/she did not complete his/her break in this case, so he/she did not complete neither a full break nor a full rest to regain his/her speed up till now.

Function: cooksInRestAlternative

Member of: Class Restaurant

Inputs:

- urgentOrder: a pointer of class Order*, the urgent order required to be assigned
- currentTimestep: an integer indicating the current time step in which the simulation of the restaurant is working

Returns: Boolean

- true: if a suitable cook from those in rest was assigned with the urgent order
- false: the urgent order was not assigned because the list was empty

Called by:

- Restaurant :: serviceUrgentOrder

Calls:

- SortedList :: isEmpty
- SortedList :: getLength
- SortedList :: getEntry
- PriorityData :: getData
- Cook :: getType
- SortedList :: remove
- Cook :: setAssignedInRest
- Cook :: setSpeed
- Cook :: getSpeed
- Restaurant :: assignCook

Function Logic Description:

This function is required to assign the urgent order to a suitable cook from those resting after an injury, if any, and this is in case there are no available cooks and none of them are on break either. Firstly, if the list is not empty, it searches for the suitable cook according to the rules of assignment of the VIP order, as the priority goes first to the VIP cooks, then the normals and lastly the vegans. Consequently, we search for the first VIP cook, if not found, the first normal and then the first vegan. If the cook was found, he/she is removed from the list of cooks in rest, have his/her assignedInRest data member set to true, so as to regain his/her speed after the finish of his/her next break, have his/her speed halved and is sent with the urgent order to the function assignCook, which assigns the order

to the corresponding cook found. If not found, this means the list is empty and false is returned.

Function: incrementOrdersWaitTime

Member of: Class Restaurant

Inputs:

- Nothing (void)

Returns:

- Nothing (void)

Called by:

- Restaurant :: Simulation

Calls:

- SortedList :: isEmpty
- SortedList :: getLength
- SortedList :: getEntry
- PriorityData :: getData
- Order :: incrementWaitTime
- UnsortedList :: isEmpty
- UnsortedList :: getLength
- UnsortedList :: getEntry

Function Logic Description:

This function increments the waiting time counter of the VIP orders to know when they will be turned to urgent ones and of the normal orders to know when they will be promoted automatically. If there are waiting VIP orders, the function goes through the whole list and calls function incrementWaitTime of each order. For normal orders, if there are any waiting, the function also goes through the whole list of them and increments the waiting time of each one.