Programming Assignment: Ambulance Simulator

CS21120 Assignment, 2019-2020

Wojciech Sowinski

wos2@aber.ac.uk

Task 1: Job, the IJob implementation

As some method were overridden in a way I haven't had to do anything with them (getters) I will not describe their implementation. Method tick() and isDone() are strictly connected and one specifies the other, both its returns depend on same variable. I have decided to create a variable called ticks_left which is initiliazed in constructor of Job and replace its value with a value of duration value of particular job. So in tick() I decrement the variable by 1. Tick() updates job, which behaviour is dependent on variable ticks_left in the method. Variable is needed to maintain some internal state of the Job. Tick() is called from Simulator class only then when there's any ambulance available and there is job to do. When it's called it means that ambulance has started its travel to patient and back to station. Each time when time 'ticks' it means the same as duration of job (I mean its way to patient and back) reduces by one 'ticks'. Note that we can assume that a duration (I.e. duration == 4) is equal to ticks == 4. In this case we would have to call tick() four times to finish the job. Here we get to method isDone(), which also uses variable ticks_left. The method cointains nothing else than just IF statement, which returns true if ticks_left equals 0 or less than 0 (both cases means that job is Done), else returns false.

SetSubmitTime method takes a parameter: time, which I have decided to assign to a new-made variable setSubTime and then pass new variable to getTimeSinceSubmit(explanation later). To setSubTime in this case is just assigned value of time to this variable.

GetTimeSinceSubmit() uses parameter "now" specified in Simulator class and it provides information how many ticks has been called since the moment when setSubmitTime's been called when Job is added to Simulator. "Now" equals current internal state (Time/Ticks in Simulator). The method returns difference now – setSubmitTime which is the ticks passed since Job was added to Simulator for a particular job.

CompareTo() method is used to compare two different jobs. Job class implements Comparable class which already includes inside method compare(). So I have decided to limit lines of code as much as I can, so I used the method compare() to compare objects but actually to compare priorities of jobs. According to Oracle method compare() "Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object." Method returns "a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object." It's used for proper inserting jobs in priority queue in order specified by this function.

Task 2: Simulator, the ISimulator implementation

At the beginning of the Class Simulator I initialize two int variable: ambulances (for keeping track of number available of ambulances) and currentTime (to maintain internal state). Then I create collection of priority queue called waitingQueue, in which are stored new-created jobs waiting for its turn to run. I create also Abstract Set CopyOnWriteArraySet called runningSet (CopyOnWriteArraySet was the choice to avoid ConcurrentModificationException error which pops up in case of using HashSet or TreeSet).

In method add() which takes an object of IJob as parameter, I add this object to the waitingQueue and set setSubmitTime of object to current tick, internal state (currentTime).

Method tick() cointains two loops: for and while. According to assignment in loop For, for each of job in runningSet I call a method from Job Class: tick() so that duration decrements (ticks_left). Then, but still in the loop, I checks using IF statement for each job if it's done (job.isDone()), if so I use completionTimeMapping() and numsOfJobMapping() methods. Those I'm explaining later. Also I increment number of ambulances, as done job means that ambulance is again available for doing next job. I remove done job from runningSet, close IF statement and for loop. Next step in tick() is while loop, which runs until waitingQueue has any jobs waiting inside && number of ambulances is bigger then 0. If condition of loop is true it retrieves head of waitingQueue and adds it to runningSet, removes head from waitingQueue and decreases number of ambulances by 1, as the ambulance has job to do now. I close loop and increment currentTime by 1 maintaining internal state and doing "tick" of time.

Methods mentioned above works(completionTimeMapping and numOfJobMapping) almost the same, both takes parameter of object IJob, both has same amount of keys and we could say that their keys are "the same". Only values for the keys are different. First, methods check if they contains key which is job's priority. If not – associates with the specified key (priority) the specified value = job's timeSinceSubmit(currentTime) for completionTimeMap and 1 for NumsOfJobsAtEachLvl (as it means it's first job done for the priority). Else if maps contains mentioned above keys, values for those keys are replaced. For CompletionTimeMap new value equals old value + new Job's timeSinceSubmit(currentTime), and for NumsOfJobsAtEachLvl value for the key is increased by 1 (as job for specified key = priority has been done).

Getter getTime() only returns currentTime variable, initialized at beginning of class and increasing in tick() method.

Method allDone() contains only IF statement with two condition. If condition 1: runningSet is empty && condition 2: waitingQueue is Empty, both must be true to return true, returns true. Otherwise it returns false.

Method of type Set<Integer> getRunningJobs() creates a substutional HashSet for runningSet called returnSet. For each of job in runningSet it copy the value of the set and adds to new-made HashSet. After loop it returns returnSet.

Method getAverageJobCompletionTime() takes as parameter int priority, which corresponds to priority in completionTimeMap and numOfJobsAtEachLvl. It has only return statement where for both maps it searches for specifed priority (key) and returns value associated with key. Values are divided (completion time / number of jobs).

Task 3: Experimenting with the simulator

TASK3()

I have run task3() for many times, although I put only two diagrams of two runs, as I don't see the point of putting more than that, because each of them will show the same result. If we run the task with number of ambulances equals 4, we can clearly see that between priority 0, 1, 2 there isn't so much difference of average completion time. If we look at priority 3 (no. 4 in the diagram) we can see enormous uptick in completion time. Such uptick is due to the fact that the jobs with priority 3 are performed as late as it can, although they can be added before other jobs with "smaller" priority.

On left side of diagrams we can see average time for completion time, while number 1,2,3,4 shows priority in order 0,1,2,3.

DIAGRAM 1:

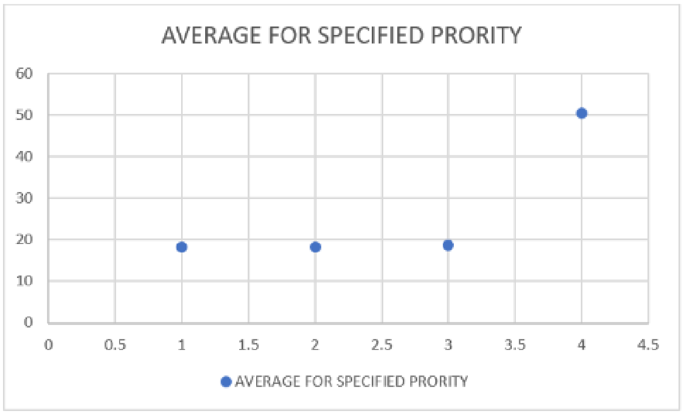
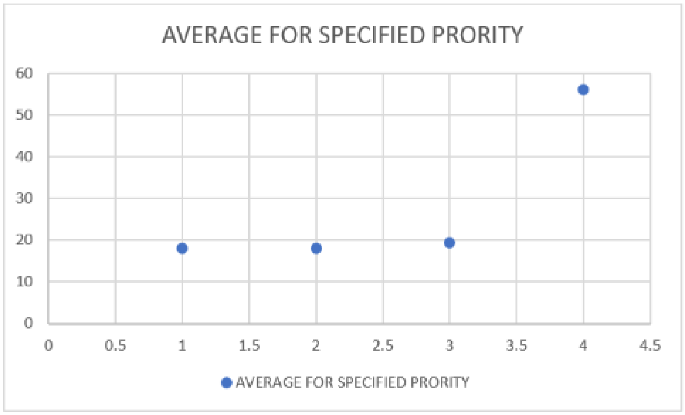


DIAGRAM 2:

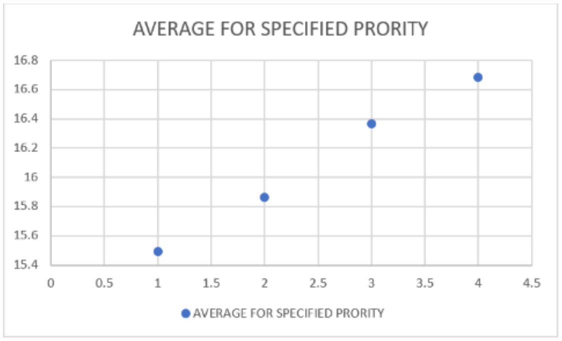


TASK4();

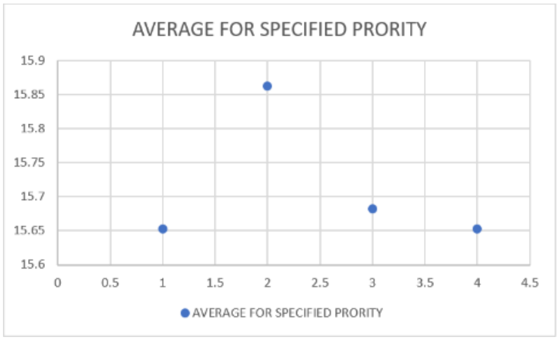
Task4() I have run many, many times. Results showed me the best number of ambulances and how best to schedule them. If we run task4() with number of ambulances equals 4, we can see the result above. There's enormous difference between priorities 0,1,2 and priority 3. However, if we run task3() with number of ambulances equals 5, we can see great improvement since average for each priority, including priority 3, are evening up. And the difference then between priorities 0,1,2 are less than 1, and the difference between 0,1,2 is approximately can be around 1. Diagrams below shows that.

While analysing task4() and diagrams, we need to specify our requirements about "best number of ambulances". If we use 5 ambulances, differences between averages as I mentioned above vary between 0.5-1.5, but if we run simulator with number of ambulances 6, the difference between priorities averages Is less 0.2. Because ambulances saves peoples' life I would say the best number of ambulances for this task is 6, as the difference (even such small as) 1 could make too big damage. Number of ambulances more than 6 is just waste of ambulances as their don't cause any visible difference.

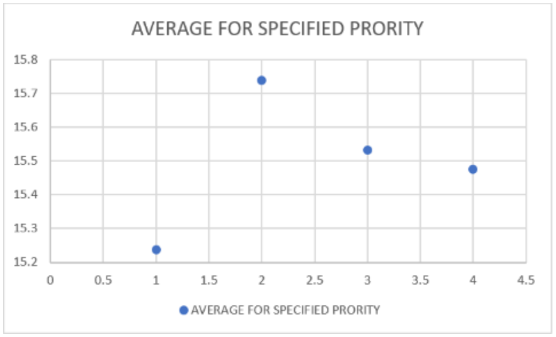
NUMBER OF AMBULANCES: 5



NUMBER OF AMBULANCES: 6



NUMBER OF AMBULANCES: 20



Self-evaluation

I can honestly admit that I spent more than 40hours for that assignment. I have struggled with it basically at every stage, but the most difficult part I think was method getAverageTime that includes creating two maps with certain ADT, and if wrongly picked it causes many errors. Also, the way of putting values into maps was a bit tricky, and I had to think about the algorithm few times and changing it over. Report I think was not that hard, as I understand now the whole assignment perfectly, as I learned a lot while doing this, so it was just formality. I'm just afraid that my analysing and describing is not the way it should look like. However, I hope it's not that bad. I think I should get the mark around 70-80%.

Thanks for your attention :)