CSC148, h5 Prof Mitchell 11/26/19 Due: Dec 10 UPDATES – RED a) Sec#1 (re-stock duration) b) Sec#4 (Handin logistics)

*A general note about h5 – features/behaviors of the model that are stated as “required” and are also underlined MUST BE IMPLEMENTED as described. Required features that are not implemented as described (or unimplemented) will lose proportional credit.*

**#1 Introduction**

You are to code a simpy script that extends the simulation of h4. Some parts of h4 are the same in h5 but other parts (**colored blue**) are a significant extension of h4.

h5 still has one product warehouse (refer to the warehouse instance as **W**). The DES t.u. is still 1 hour. **However, h5 has a total of 17 Home Depot stores (refer to them as HD1, HD2, … , HD17)**. The warehouse serves as a regional warehouse for the large group of HDk.

**The HDk are required to exist for an entire model execution.**

As in h4, there is only one kind of customer order at each HDk: n 90-pound bags of roofing shingles. Sales orders are at HDk’s “Pro desk”. Successive customer (general construction contractor) orders at each HDk occur 1 or 2 hours apart (uniformly distributed). Each order size n is a uniformly random int in [10,45]. Also, the time taken to perform a re-stock is uniformly-distributed in [2,4].

**#2 The model – Reorder and Stockout order processing by HDks**

Each HDk processes orders, one-by-one, and each order decrements an HDk’s remaining inventory by variable amount n. However, order processing is more complex in h5.

**As soon as a size n order decrements HDk’s inventory below level RP = 50 (RP stands for “reorder point”), HDk is required to interrupt the warehouse’s pf “dispatcher”.**

**The interrupting HDk continues processing orders and decrementing HDk’s inventory level Ik(t) (rather than stopping as in h4).**

**It might or might not happen that HDk’s Ik(t) decreases to a point that a next order’s orderSize satisfies (Ik(t)–orderSize) < 0 (the so-called “out-of-stock” state), meaning that this order cannot be fulfilled.**

**IF out-of-stock (aka stockout) occurs, an HDk is required to cease/stop fulfilling all orders, and wait synchronously (meaning HDk’s order processing pf is suspended) until the shared event (refer to as stockoutDone) happens. Shared event stockoutDone is required to be a list having an index location for each HDk. Correspondingly, the warehouse instance is required to have a pf that processes restock for an HD (it would not make sense for an HD to do its own restock). *Hint: Initialize the shared event list locations to None, and create a shared event for HDk’s reordering pf to wait on by assigning to list index position [k] the value env.event(). Each such shared event represents a future event, namely that W has delivered HDk’s restock amount. This assignment is, of course done by a pf in W.***

**#3 An h5 model execution trace – the required results content & format**

The different parts of execution results shown in the ‘Example Results’ are described next.

Some parts are the same as in h4, and are in black, non-bold, non-underlined text.

**The parts that are required and new in h5 have the notation** xxxxxxxx (blue and underlined);

**the new parts are accompanied by implementation hints/notes (see boxes).**

**All output parts are described in their required relative order of display**.

The specific display below illustrates inventory operation with “ideal restock policy”

{see HandIns requirements, part 2a)}

Running class Warehouse \_\_init\_\_ fcn at time 0 for warehouse WH1 // NumPy seed is 18334247

Creating 17 HD stores

Running HD 1 \_\_init\_\_fcn at time 0 Setting Inventory & RP values to 200 & 50

Running HD 2 \_\_init\_\_fcn at time 0 Setting Inventory & RP values to 200 & 50

:

Starting doOrders() for HD 1 at time 0

Starting doOrders() for HD 2 at time 0

:

Starting doOrders() for HD 17 at time 0

HD 3 inventory level is 180 at time 1

HD 8 inventory level is 162 at time 1

HD 11 inventory level is 187 at time 1

:

HD 17 inventory level is 144 at time 3

**Every 4 t.u.** (=reporting period) there is a 1-line display with the label shown, whose right-most part is the current list of HDk with a pending (in progress) restock

HD 14 inventory level is 134 at time 3

**RPT:: The HDks with pending restocks at time 4 :: []**

HD 9 inventory level is 177 at time 4

:

HD 12 inventory level is 54 at time 6

HD 14 inventory level is 16 at time 6

Each time an HDk’s Ik(t) decreases below RP, that HDk instance interrupts a pf in W that initiates a restock of HDk. The interrupting pf and a pf in W issue the 1-line messages in the formats shown formats.

**!! HD 14 is interrupting warehouse w at time 6**

**Start scheduling & delivering restock to HD 14 at time 6**

HD 5 inventory level is 56 at time 7

HD 6 inventory level is 97 at time 7

:

HD 9 inventory level is 127 at time 7

HD 12 inventory level is 30 at time 7

!! HD 12 is interrupting warehouse w at time 7

The first reporting period where there is at least one concurrent restock in-progress

Start scheduling & delivering restock to HD 12 at time 7

**RPT:: The HDks with pending restocks at time 8 :: ['HK12', 'HK14']**

HD 1 inventory level is 55 at time 8

:

Each cycle of inventory draw-down by an HDk either stocks out or not. Issue a message whenever an HDk is stocked out, as shown. (See stats in box below)

HD 17 inventory level is 67 at time 8

**Stockout occurred for HD 14 at time 8**

HD 15 inventory level is 73 at time 8

:

Each time an HDk restock is completed, a message is displayed. (Statistically, in a ideal MI system, approx. 2/3 of the restocks will complete **without stocking out**. This specific run had 88 stockouts total over 228 restocks total over 168 t.u. (simulates 7 continuous 24X7 operating days).

HD 15 inventory level is 58 at time 9

**HD 14 completed re-stocking, resuming w. I(t)= 200 at time 9**

HD 16 inventory level is 37 at time 10

:

HD 14 inventory level is 161 at time 11

HD 2 inventory level is 179 at time 11

HD 8 completed re-stocking, resuming w. I(t)= 200 at time 11

RPT:: The HDks with pending restocks at time 12 :: ['HK1', 'HK3', 'HK7', 'HK10', 'HK11', 'HK15', 'HK16', 'HK17']

HD 13 inventory level is 50 at time 12

One of the (somewhat rare) periodic reports having a very large number (8) of concurrently-occurring pending restocks.

HD 1 completed re-stocking, resuming w. I(t)= 200 at time 12

HD 6 inventory level is 37 at time 12

:

HD 16 inventory level is 67 at time 45

HD 11 inventory level is 165 at time 45

An interesting cluster of 3 of the HDk that all finished a restock at the same model time. Your model (and seeding) may or may not ever see a cluster this large at one time.

HD 13 completed re-stocking, resuming w. I(t)= 200 at time 45

HD 2 completed re-stocking, resuming w. I(t)= 200 at time 45

HD 7 completed re-stocking, resuming w. I(t)= 200 at time 45

HD 10 inventory level is 76 at time 46

HD 8 inventory level is 49 at time 46

!! HD 8 is interrupting warehouse w at time 46

Start scheduling & delivering restock to HD 8 at time 46

HD 17 inventory level is 45 at time 46

!! HD 17 is interrupting warehouse w at time 46

Start scheduling & delivering restock to HD 17 at time 46

HD 9 inventory level is 53 at time 46

HD 15 inventory level is 74 at time 46

HD 1 inventory level is 60 at time 46

Stockout occurred for HD 3 at time 46

HD 6 inventory level is 93 at time 46

HD 13 inventory level is 171 at time 46

HD 2 inventory level is 184 at time 46

HD 7 inventory level is 187 at time 46

HD 4 completed re-stocking, resuming w. I(t)= 200 at time 46

HD 16 inventory level is 56 at time 47

HD 5 completed re-stocking, resuming w. I(t)= 200 at time 47

:

Your simulation must continue to operate (as above, and issuing all required messages, per event type) for a total of 168 t.u. , and the last section of display is a Final Report.

:

HD 17 completed re-stocking, resuming w. I(t)= 200 at time 166

HD 8 completed re-stocking, resuming w. I(t)= 200 at time 167

HD 9 inventory level is 52 at time 167

HD 11 inventory level is 136 at time 167

Final Report section –

1. Label and final model time
2. Label and list of stockout counts, by incr. HDk number k
3. Label and grand total stockout count
4. Label and grand total number of restocks

HD 12 inventory level is 65 at time 167

HD 10 inventory level is 63 at time 167

HD 16 inventory level is 109 at time 167

HD 15 inventory level is 113 at time 167

HD 5 inventory level is 90 at time 167

**Finished run at model time 168.0**

**Total number of stockouts, by HD number [6, 6, 6, 6, 4, 9, 5, 4, 3, 6, 2, 6, 4, 4, 7, 6, 4]**

**Grand total number of stockouts over all HDk 88**

**Grand total number of restocks completed over all HDk 228**

**#4 Required HandIns –**

1. A complete copy of your source code
2. Complete output (*with exact display shown in Section #3*) **for 2 model runs** with the same numpy seed (different from the seed displayed in the sample trace in section#3).

Each model execution runs for 168 t.u. (simulates 24X7 operation for 7 simulated days).

1. Run#1: “Ideal restock policy”: Ik(t) inventory is reset to max inventory level (200) immediately before HDk resumes normal ordering immediately after a restock’s delivery to HDk has happened. This is equivalent to W being told the amount of inventory to deliver to an HDk based on Ik(t) at the time the restock arrives at HDk – which is clearly an impossible thing to know about the future.
2. Run#2: Because the policy in a) is not practically possible, many have developed restock amount algorithms in the last 70-80 years. An MI model with enough random variables is so complex that approximate policies are the best we can do.

Our non-ideal restock policy algorithm will be: W delivers a restock amount “ra” based on the model’s (fixed/constant) RP value. Then, ra = (max inventory level) – (0.5)\*RP.

This formula is based on simple averaging – assuming most restocks finish before stockout, its relatively “safe” to assume that approx. (1/2)\*RP stock will be sold in the following time interval:

[(time restock was requested of W), (time restock delivery & unloading was completed)]

*Notes: you should expect to see MORE restocks over an entire model run in case b).*

*This is because HDk’s restock amount is typically less than the same quantity in a).*

**What to hand in – there is a non-trivial amount of output in each of the 2 runs.**

**We will avoid killing trees because you will not need to submit output results from runs.**

**The only item to submit is a copy of your source code used during run#1.**

**Submit your source code as one attachment in email by NO LATER THAN 5pm, Thursday Dec 12.**

**This is a FIRM & ABSOLUTE DEADLINE.**

**To repeat, all that I need is your source file, but be sure it is an x.py formatted file.**

**A source file that I cannot translate or run will earn 0 execution credit.**

**IMPORTANT!**

**\*\*\* However, before emailing me your source code, do the following edit of Run#1 source:**

**At the place in the source where Run#1 logic re-sets the I(t) for the ‘Ideal Restock Policy’,**

**Insert the following simply Python comment statement:**

**# Restock Policy**

**(This will make it easy for me to locate and switch between Run#1 and Run#2)**

**Of course, during Run#1, Comment the Run#2 logic that changes the restock calculation in YOUR CODE that calculates: (max inventory level) – (0.5)\*RP**

**This means that after I do your Run#1, I want to simply be able to COMMENT the ‘Ideal Restock Policy’ restock amount and UN-COMMENT your Run#2 restock calculation.**

**The editing of your source code MUST BE THIS SIMPLE. I cannot get into the business of tedious editing in going from Run#1 to Run#2.**

**#5 Structure of Source code & Grading**

The element/categories graded for h5 will be similar as in h4 grading.

As in h4, there are still only 2 classes (warehouse and HD), and no sub-classing is needed or justifiable for this model. However, good OO design should suggest that W is the “driving/controlling” class in h5. It should have several (4 or 5 (?)) pfs that implement

1. periodic (every 4 t.u.) reporting
2. handling interrupts from an HD pf
3. dispatching (i.e. scheduling) a restock pf for each requested restock and the shared event relationship to HD’s ordering pf. The dispatching pf must run “forever” (so at the top level, it should be a while True: form of Python loop statement.

*(Note – there is no need for an W pf to interrupt an HD when a restock is finished because your model logic should simply have the HD ordering pf wait for the shared event to happen before HD resumes normal ordering operations)*

Concerning source code documentation, several h4 papers were non-performing/disappointing. Ordinary Python/simpy source code should have module/class/function docStrings, and almost all individual code statements should have a meaningful description. This kind of note should not be an issue for students near graduation, but unfortunately, seems to be continually ignored. (It will be a culture shock to some when future jobs require that shop coding standards, including documentation, are followed).