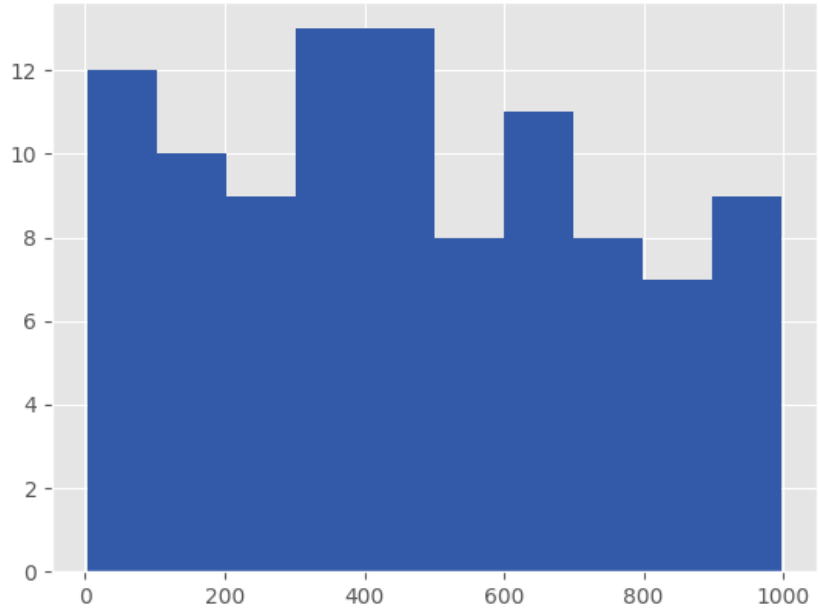
Trevor Bright
HW 2
CSE 622

Homework 2

1. For this problem, I created a set of uniform numbers using Python's uniform number generator. The graph shows that while there are some spikes of occurrence in a few places, it does a good job of being relatively even and uniform.

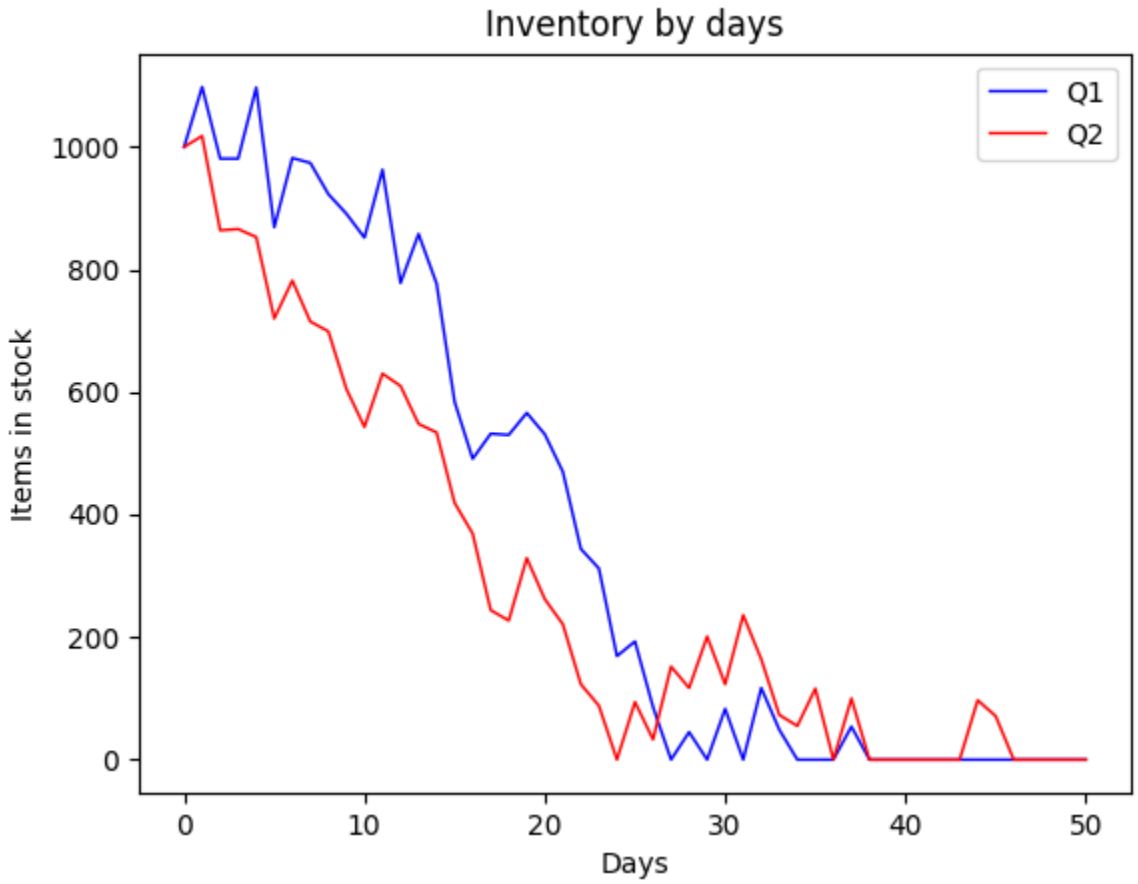


The next section was using these numbers to calculate the area of a circle, its bounding square, and eventually calculating the value of pi for each. I learned that while the value of pi is the same for the first 14 numbers behind the decimal, the numbers lower than that are not always the same. The random numbers and the calculation of the areas affected the value of pi.

```
num:853.6770995065295          pi:3.1415926535897927
num:299.75060122808605         pi:3.1415926535897927
num:878.3548683416466          pi:3.141592653589793
num:105.76538719495043         pi:3.141592653589793
num:601.9435694370095          pi:3.141592653589793
num:838.4667856276072          pi:3.1415926535897936
num:39.3566436909234           pi:3.1415926535897936
num:207.75835022078581         pi:3.1415926535897927
num:377.0041084332204          pi:3.141592653589793
num:409.3131297430721          pi:3.1415926535897936
num:674.1123410514386          pi:3.141592653589793
num:833.7143633119205          pi:3.1415926535897936
num:585.4513837358538          pi:3.141592653589793
num:412.86044018601143         pi:3.141592653589793
num:995.8694233114321          pi:3.141592653589793
num:416.8905865893383          pi:3.1415926535897936
num:413.9937405864882          pi:3.141592653589793
```

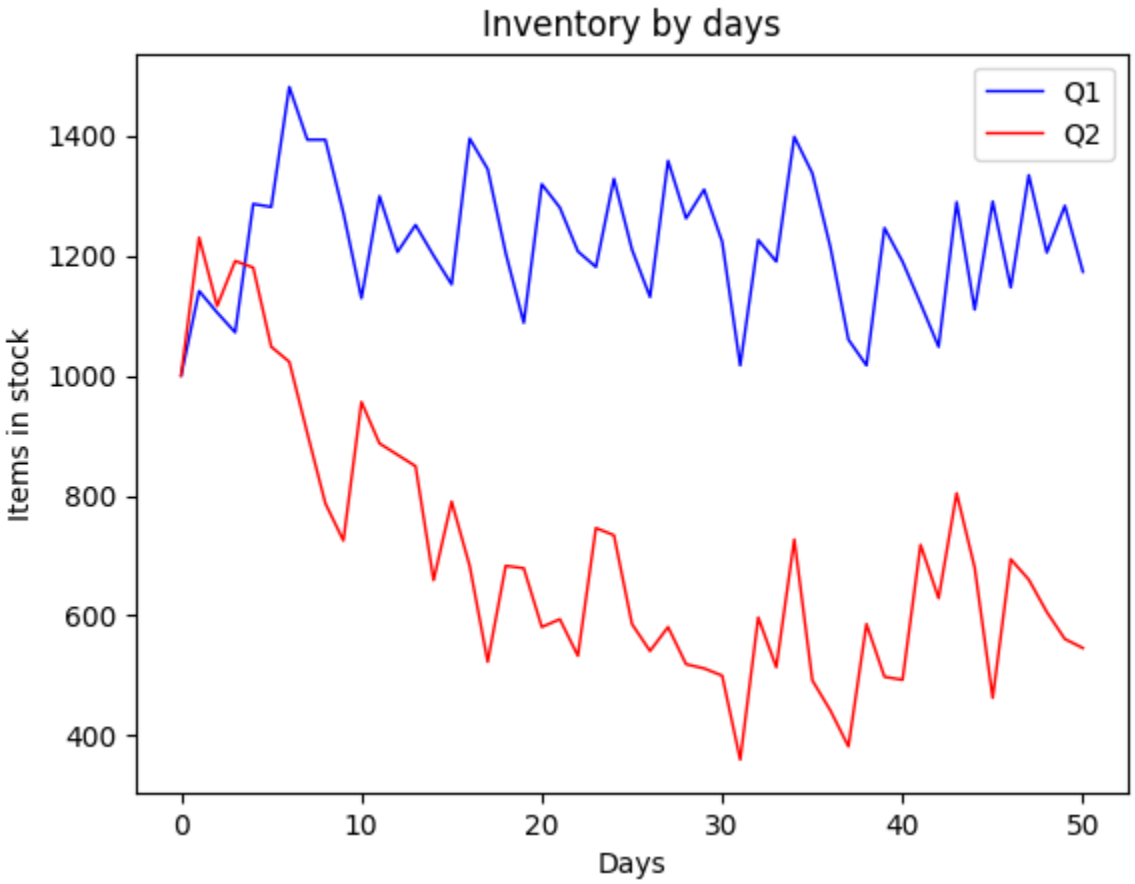
Trevor Bright
HW 2
CSE 622

2. The program I created was a simple Python command line program that accepts the initial quantity of item 1 and item 2, and the number of days you want to run the simulation. For the order sizes of each, P1 and P2, I toyed around with those a lot to see the different results. Before I explain my results of different order sizes, I must go over the basics of the customer portion. Since my testing Q1 and Q2 was 100, I decided to pick the 100 as the mean number of daily customers for each item. With no restocking orders, this means it would only last 10 days. To add variety, I created a list of 1000 numbers of normal (Gaussian) distribution with the mean of 100 and the standard deviation of 100. I then randomly picked a number from that list, and that would be the purchases for the day. I did that with both items. While playing around with the order size for each, I decided instead of the user defining it, it would be a portion of the initial quantity. I first started off with just having 1/8 of the initial inventory, which when using 1000 would come out to be 125 per order. This proved to be not sustainable for keeping a consistent stock as shown in the graph.



I first tried to use 1/6 of the initial inventory, but after running it a few times, the results were very similar to 1/8. I then tried to go to 1/4 of the initial inventory, which came out to be 250 per order. This proved to be the most consistent without having too much in stock or not having enough. While the order size could be increased, it wouldn't be

economically efficient. You would have more in stock than you are selling, leading to losing money from buying more than selling. If this occurs, orders will have to stop to collect the lost profits from the overstock.



3. If I had more time, I would have improved the tracking of the attacking and defending ship. While I think it turned out good for the time I had, it was often weird at times and counterproductive. The defending was a lot trickier than the attacking, so it didn't work as well. I would also create bounds in the scene to keep the ships in a contained area. In the initial phase of testing, the defending ship would go straight into the abys due to its running away algorithm and the attacking ship would infinitely follow it. I'd also like to add variability in speed, giving both ships an advantage to either catch up or avoid the other ship.