Game Data Structures - Evaluation Report
Hayden Whiteford 104001272

This short report aims to evaluate four individual data structures as a candidate for a player "inventory" for the Zorkish c++ console game.

The three areas that were evaluated were access, addition and removal. Each process was executed with each container type, but also timed using the steady_clock::now() function before and after the process was executed. This allowed me to see the exact performance for each type.

ACCESS

With regard to simply accessing a container full of 4 GameObjects, a simple class I created for this, the best recurring performers were the array and the deque. This makes each quite good candidates for situations in which the player needs all the items in the inventory displayed to them in a list. The problem with using an array however, is that it is not dynamic in size, which creates a problem with addition and removal which we'll see later on. Using duque is slightly better as it has efficient addition and removal and the front and end but not in arbitrary positions. The List structure was unfortunately far slower than the other types due to the list needing to be iterated through every time to access each node, which ended up doubling the amount of time required. Vector was slightly slower than Array and Duque, but still quite fast relative to List.

ADDITION

The fastest structures when it came to adding an additional GameObject to the back of the container were the List and the Deque. Again, Lists are extremely efficient at insertion and removal, so this makes sense here. Since we were adding to the end of the container, this also is favourable for the Deque structure. Surprisingly the Vector Structure was the slowest here, with almost double the time than the other structures. That being said the array structure should really be the slowest here if given more robust testing, as the array does not have addition and removal; to mimic such I simply created a new array. If I wanted to be more precise I would use a for loop on the original container to copy each item and the additional item over to a new array, which would have taken exponentially longer than the other structures.

REMOVAL

Excluding the Array structure which technically fails this test for the same reasons as before, the winner is the Vector class, at half the time that the List and Deque structures took. The List and Deque structures were taking up to 8.75e-07 seconds for removal.

In summary,

Vectors
Fast access and removal, slow addition

Lists
Fast addition, slow access and removal

Array
Fast access, slow addition and removal

Deque
Fast access and addition, slow removal

My personal choice comes down to between Vectors and Deques depending on whether the game requires more inventory addition or removal. My guess says that there would probably be

an equal amount of addition and removal, as most items put into the inventory should have some use of some sort that removes them from the container once used. However, there may be objects that go into the inventory that can be used more than once without being removed, or useless junk objects that go into the inventory without any use, and won't need to be removed. Therefore the Deque fits this scenario better as these outlier cases would mean there is more addition than removal for the inventory.