Inteligência Artificial Para Jogos
Projecto - Pathfinding
Group 26

Henrique Fernandes - 66994
Fábio Almeida - 70227
Pedro Rosa - 73841

1. Comparing the Pathfinding Algorithms

	A* (Zero)	A* (Euclidean)	Node Array A* (Zero)	Node Array A* (Euclidean)	Node Array A* (Gateway)
Max Open	154	166	155	259	247
Processing Time	195,3964	93,8911	75,9735	127,5406	43,7813
Avg. Time/Node	0.0310	0,0494	0,0085	0,0252	0,0256

2. Optimizations

As requested we profiled our code and tried to find places where we could make improvements. We realized there were two places that we could improve, namely, in the Cluster Graph class and DynamicFollowPath class. We'll briefly describe our attempts to improve our code. In the following table we show the differences between the algorithms prior and after optimizations:

	Max Open (Before)	Max Open (After)	Processing Time (Before)	Processing Time (After)	Avg Time/Node (Before)	Avg Time/Node (After)
A* (Euclidean)	150	166	371,974	93,891	0,3523	0,0494
Node A* (Euclidean)	177	259	147,276	127,540	0,1647	0,0252
Node A* (Gateway)	157	247	101,944	43,7813	0,31659	0,0256

2.1. Gateway Distance Heuristic

Upon profiling the completed project it was clear that the most time consuming part of our code was the Quantize method inside the Cluster Graph class, as we can see in Figure 1. We decided to direct our efforts into improving this method since it was used a lot by the Gateway Distance Heuristic.

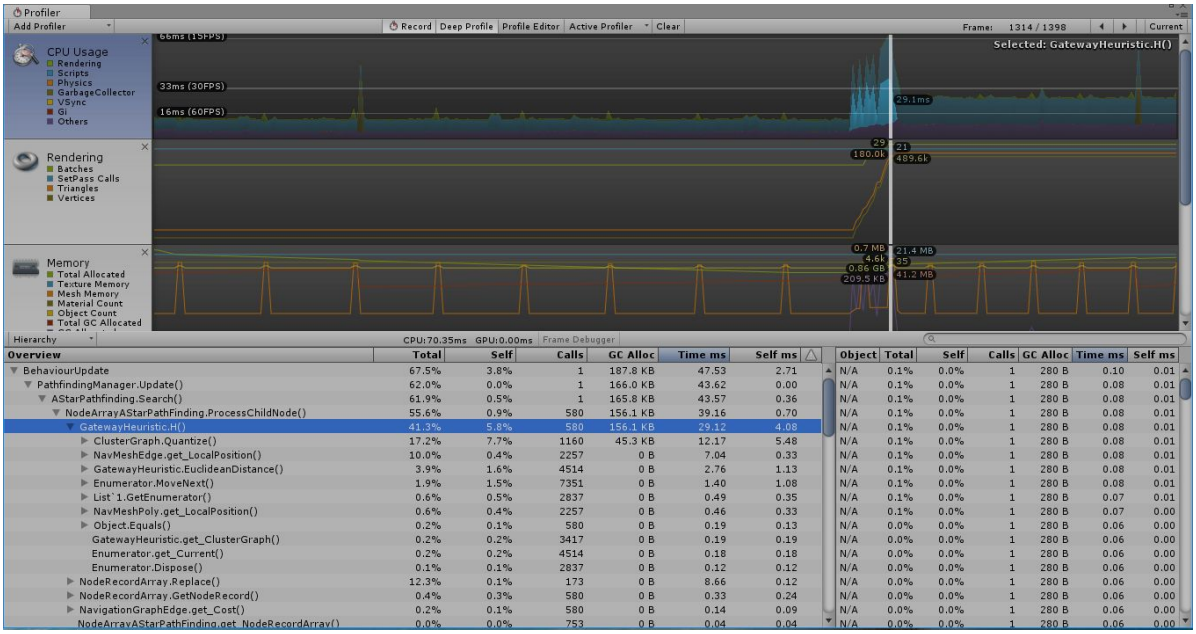

Figure 1 - Profiler screenshot for the Quantize method before optimization

After identifying the critical section, where we would transverse all clusters in order to check if a given node was inside that cluster. It was obvious that we could improve this by storing this information in order to have a better performance at run time sacrificing some memory. Therefore we decided to add a new array to the ClusterGraph that is calculated “offline”. This way we gained a significant improve in speed at runtime.

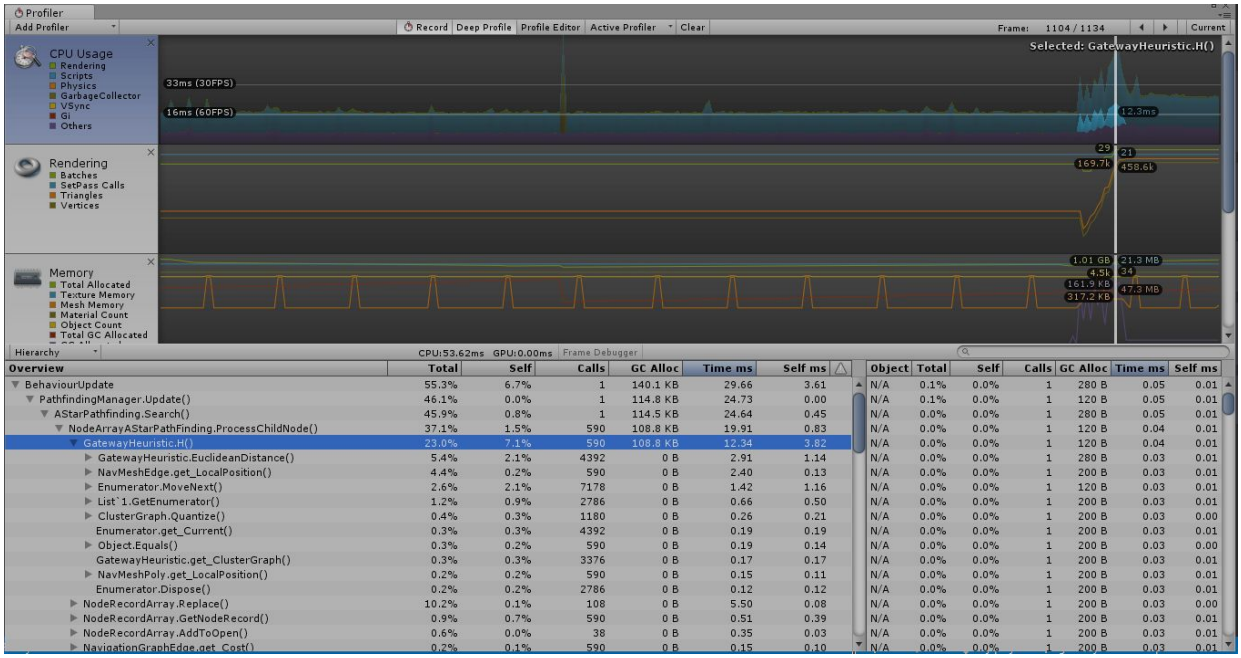

Figure 2 - Profiler screenshot for the Quantize method after optimization

2.2. Dynamic Follow Path

In this section the optimization was plain and simple, the reuse of the dynamic follow path, since when the code was given to us, every time there was a new path to be followed by the character it was create a new instance of the DynamicFollowPath class. That creation was done in the update method, although this method is only called when we have a new point to follow, if the user clicked several times to test performance, there was a needless creation of N objects of type DynamicFollowPath when we could just reset the class and set the new parameters.

This is the basis of recycle whenever you can, and we could this time.