


















Lab 4 – Optimizing A* for Pathfinding

1) Explore the provided scene and the source code














- a) In this laboratory we will analyze differences in efficiency by using different data structures for the Open and Closed sets, and you will also implement the NodeArray A* algorithm.

2) Compare the Unordered List for Closed set vs Dictionary for Closed Set

- a) In Lab 3, we implemented a Dictionary for the ClosedSet. We will compare the efficiency between using the original unordered list and the dictionary. It is very important that you select more or less the same path when comparing the two data structures. Select a path that is complex enough (i.e. with a large number of explored nodes: 2000-3000 would be great) and that your computer can handle. Specify a number of 50-100 nodes explored per frame (the most your computer can handle without stalling). Your search algorithm will then run over a set of frames. When running the profiler, we will have many frames where the search algorithm is doing the search. Select one where the time spent is average (i.e. avoid the frames with the lowest and with the highest values).
- b) Start by running the A* pathfinding algorithm with the unordered list for closed, and fill the next table with the corresponding information.

Unordered list for Open and Closed		
Method	Calls	Execution time
A*Pathfinding.Search	1	
GetBestAndRemove		
AddToOpen		
SearchInOpen		
RemoveFromOpen		
Replace		
AddToClosed		
SearchInClosed		
RemoveFromClosed		














- c) Now do the same, but using the dictionary for the closed set, filling the next table. Try to explain why optimizing the closed set is so important (hint: look at the size of the closed set). Which of the three methods for the closed set is the most relevant?

Dictionary for Closed		
Method	Calls	Execution time
A*Pathfinding.Search	1	
GetBestAndRemove		
AddToOpen		
SearchInOpen		
RemoveFromOpen		
Replace		
AddToClosed		

SearchInClosed		
RemoveFromClosed		












3) Comparing Priority Heap for Open Set

- a) Run the A* pathfinding algorithm with the provided NodePriorityHeap data structure. Fill the table below and compare with the other data sets.

NodePriorityHeap for Open		
Method	Calls	Execution time
A*Pathfinding.Search	1	
GetBestAndRemove		
AddToOpen		
SearchInOpen		
RemoveFromOpen		
Replace		
AddToClosed		
SearchInClosed		
RemoveFromClosed		

4) Implement and Compare Node Array A*

- a) Complete the implementation of the NodeArrayAStarPathfinding. You will also need to complete the implementation of the NodeRecordArray. The NodeRecordArray is the data structure used for both Open Set and Closed Set in Node Array A*. Take a look at the theoretical slides in order to know what you need to do. The basic idea is that node records for all nodes are initially created, and you can use the node Index to access the corresponding node record almost immediately from the Node Record Array. The node record has a flag that specifies the current state of the Node (unvisited, open, closed). When a new search is performed all the NodeRecords have to be initialized to unvisited. We still need an Open data structure because we need to order the nodes to get the best ones (and you don't want to do this for the whole array). But for the Open you can just use a PriorityHeap. **Note:** to make this algorithm as fast as possible you should avoid creating new NodeRecords. Reuse the ones in the array. Also take that into consideration when implementing the NodeRecordArray.
- b) Once you manage to implement the new algorithm, change the Pathfinding Manager to use the new search algorithm and run the same path as for the previous versions of the algorithm. Fill out the last table. Compare this version of the algorithm with the previous one. Compare this version with your initial implementation (i.e. the one in table 1). How much faster is this one? Which methods benefited the most? You will see that choosing the most appropriate data structure can have a huge impact in the algorithm efficiency.

NodeRecordArray A*		
Method	Calls	Execution time
NodeArrayA*.Search	1	
GetBestAndRemove		
AddToOpen		
SearchInOpen		
RemoveFromOpen		
Replace		
AddToClosed		
SearchInClosed		
RemoveFromClosed		