# Inteligência Artificial para Jogos

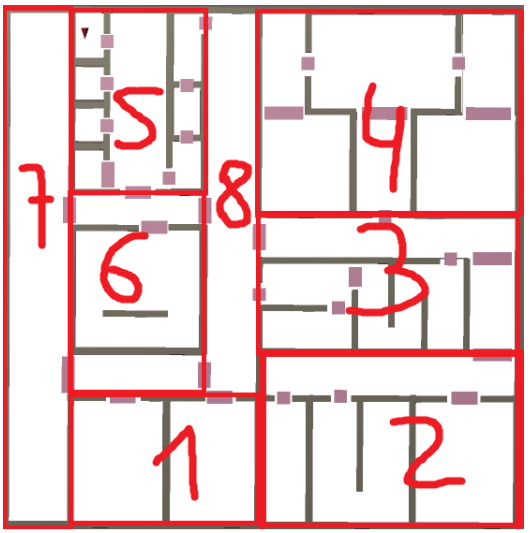
2016-2017

Ricardo Silva, 78414 – Group 4

## 2.º IAJ Project – Efficient pathfinding in Room based scenarios and Path Following

# REPORT

### Gateway Heuristic and Quantize




When I first started running searches with the GatewayHeuristic I found that some nodes were not inside of any cluster. To solve this issue I increased the size of each cluster bounding box by 2 in both directions which seems to work just fine.

To implement the ClusterGraph.Quantize() method I used the bounding box of the cluster and the local position of the node using the MathHelper.PointInsideBoundingBox() method. Later I thought about using a kind of hierarchal approach where I group clusters into a cluster of clusters to improve performance (as seen on the image). In this hierarchal approach first I check on which super cluster the node is inside and then check in which one of the clusters of that super cluster the node is. These superclusters were created manually and I added them to the clusterGraph asset.

### Following Path Movement

For the following movement I implemented the Dynamic Follow Path Movement described during the theoretical classes using the Arrive Movement to chase the target. The implementation was pretty straight forward but at first the character was getting stuck in some tight corners and random places. Upon further debugging and testing I found that that was happening because some local paths were as small as the vector (0, 0.2, 0) and in order for the character to stop successfully at the end of the path the Stop Radius of the Arrive Movement needed to be of at least 0.5. In order to solve this issue the Stop Radius is kept at 0 until the character has reached the end of the path and then it is changed to 1 to achieve a perfect stop at the end of the path without ever getting stuck on those really small paths.

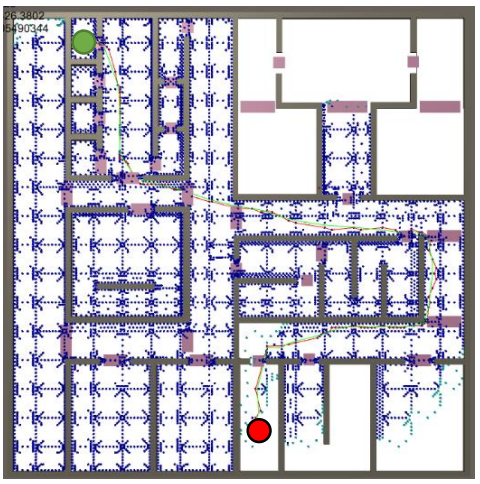
### Comparing the pathfinding algorithms

In order to get a better comparison between the three setups I hard coded a destination far from the starting position instead of using the mouse click. Also the searches were run with nodes per search = Int.MaxValue and return partial path = false so I could keep track of the total number of calls.

**Original A* Pathfinding Search**

Open: Priority heap, Closed: Dictionary, Heuristic: EuclideanDistance

| Method | Calls | Execution time (ms) |
|---|---|---|
| A*Pathfinding.Search | 1 | 1808.16 |
| GetBestAndRemove | 7767 | 163.61 |
| AddToOpen | 7903 | 32.80 |
| SearchInOpen | 41290 | 983.69 |
| RemoveFromOpen | 7766 | 229.28 |
| Replace | 2892 | 80.28 |
| AddToClosed | 7766 | 12.55 |
| SearchInClosed | 28396 | 32.42 |
| RemoveFromClosed | 2 | 0.22 |



**Nodes visited:** 7766

**Maximum Open Size:** 225

**Processing Time:** 426.3802

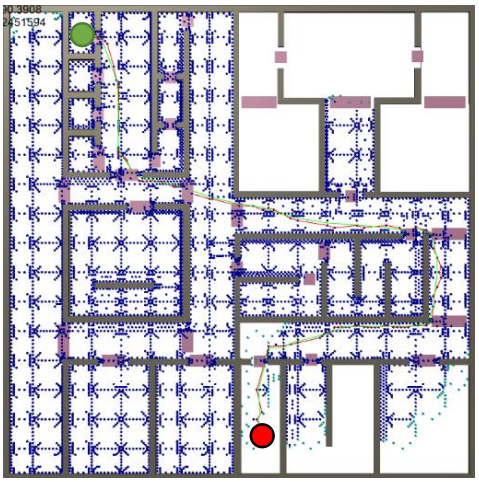
**Processing Time per node:** 0.054903

---

**NodeArray A* Pathfinding Search**

Open: Priority Heap, Closed: -, Heuristic: EuclideanDistance

| Method | Calls | Execution time (ms) |
|---|---|---|
| A*Pathfinding.Search | 1 | 784.90 |
| GetBestAndRemove | 7767 | 157.10 |
| AddToOpen | 7903 | 35.11 |
| SearchInOpen | 41290 | 14.59 |
| RemoveFromOpen | 7766 | 227.32 |
| Replace | 2892 | 78.36 |
| AddToClosed | 7766 | 2.24 |
| SearchInClosed | 28396 | 12.69 |
| RemoveFromClosed | 2 | 0.00 |



**Nodes visited:** 7766

**Maximum Open Size:** 225

**Processing Time:** 190.3908

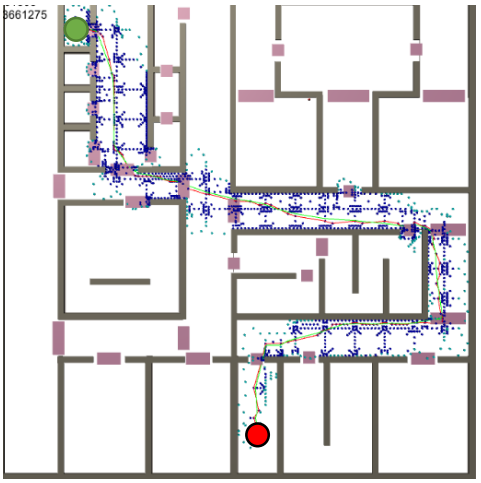
**Processing Time per node:** 0.024515

---

**NodeArray A* Pathfinding Search**

Open: Priority Heap, Closed: -, Heuristic: GatewayHeuristic

| Method | Calls | Execution time (ms) |
|---|---|---|
| A*Pathfinding.Search | 1 | 553.21 |
| GetBestAndRemove | 1886 | 43.82 |
| AddToOpen | 2174 | 15.57 |
| SearchInOpen | 10557 | 4.23 |
| RemoveFromOpen | 1885 | 86.53 |
| Replace | 964 | 36.53 |
| AddToClosed | 1885 | 0.55 |
| SearchInClosed | 6930 | 3.05 |
| RemoveFromClosed | 315 | 0.08 |



**Nodes visited:** 1885

**Maximum Open Size:** 312

**Processing Time:** 133.9016

**Processing Time per node:** 0.071035

---

## Conclusions of the comparison

As expected the NodeArrayAStar version with the Gateway Heuristic performed the best.

Comparing the original AStar with the NodeArray version is easy to understand why the second is faster. By generating all the nodes prior to starting the search we can check if a node is in open or in closed really fast by just comparing its status instead of having to search the node in some data structure and we remove the need for a closed nodes set all together. The only downside is that NodeArray uses extra memory but in this case we have no limitations so it is fine to trade memory for speed.

Comparing the EuclideanHeuristic with the GatewayHeuristic is easy to see why the second is much more efficient and produces a fill way smaller than the first one. The EuclidianHeuristic is not efficient because it relies on the distance in a straight line between two points and in a room-based map the character is almost never able to walk in a straight line. On the other hand, the GatewayHeuristic takes into account the actual path the character will need to walk from A to B and outputs a really good estimative of the cost of walking from A to B.

## Optimizations

For the optimization tests I used the same start and ending point every time.

I decided to optimize the GatewayHeuristic.H() method because this method was using the most time in a frame.

I decided to group clusters into clusters of clusters to optimize the ClusterGraph.Quantize() method. This reduces the max comparisons from 30 to 8 + 8 in the worst case scenario.
Also some additional optimizations were done, like initializing every variable that remains constant inside the cycles prior to cycle start, including getters.

| | **Before** | | **After** | |
|---|---|---|---|---|
| **Method** | **Calls** | **Time (ms)** | **Calls** | **Time (ms)** |
| GatewayHeuristic.H | 631 | 21.95 | 631 | 14.17 |
| NavMeshEdge.getLocalPosition | 3495 | 8.26 | 630 | 1.56 |
| ClusterGraph.Quantize | 1262 | 6.03 | 1262 | 5.74 |
| Vector3.getMagnitude | 6992 | 1.00 | 6992 | 1.00 |
| Vector3.Subtraction | 6992 | 0.96 | 6992 | 1.02 |
| NavMeshPoly.getLocalPosition | 3497 | 0.52 | 632 | 0.13 |

As you can see the Quantize optimization was not that successful. I found the cause of the problem to be the misalignment of the clusters bounding boxes which cause some errors (sometimes a node is inside the super cluster bounding box but it is not inside any of the clusters of that super cluster – I can explain this better during the discussion with pen and paper).

I did not notice any other methods that could be improved, maybe because I already had efficiency in mind when I was implementing the methods in the first place.

## Conclusions

In the end I think I managed to create a very efficient algorithm that is able to pathfind really well in room-based scenarios. The implementation of the levels went smoothly and I think the final result is successful. Also I think the project theme was interesting and I enjoyed solving it.