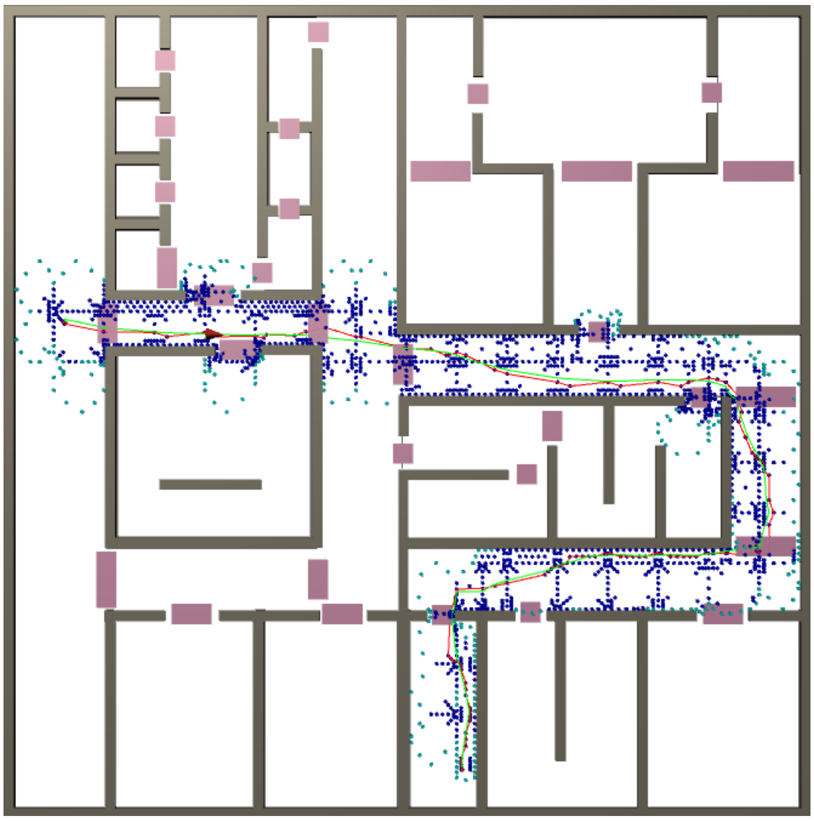
2.º IAJ Project – Efficient and smoothed pathfinding



In the second practical project of the IAJ Course, we want you to create a very efficient pathfinding algorithm that uses the goal bounding technique. The second goal is then to create a path smoothing algorithm to smooth the path returned by the pathfinding algorithm. You can (and should) reuse part of your work from the Lab classes. For this second project, you will need to write a report up to 4 pages, explaining and justifying the decisions made by your group in terms of implementation.

You should submit a zip file with both Unity source code and with the report (a pdf/doc file) via Fenix, **until 24:00 of October, 29**. We will accept submissions after the deadline, but with a 0.5 value penalty for each hour after the deadline.

Browse the Assets/Editor folder. You will find the file *IAJMenuItem.cs.* This file will create a new menu item called IAJ in your Unity editor. In that menu a new option will appear, *Create Goal Bounds.* When selected, this option will run the code that calculates the goal bounds for each edge by running a variation of Dijkstra. At the end of the process, a table with all goal bounds will be saved using Unity's serialization mechanism. A new Asset named GoalBoundingTable will be created (usually inside the Assets folder). **Move the created GoalBoundingTable Asset to the Resources folder.** For this to work you will have to implement the Dijkstra variant (as part of

Level3). Once the Goal Bounding Table is calculated, it can be used in runtime to improve the efficiency of the pathfinding algorithm.

Level 1– Traditional A*.

- Implement the A* search algorithm.

Level 2– Node Array A*

- Implement the Node Array A* search algorithm.

Level 3 – Goal Bounding

- This level will focus on the development of the Goal Bounding technique. Start by exploring and analyzing the folders provided. The DataStructures folder will contain the classes used to store information about GoalBounding. The Pathfinding/GoalBounding folder will contain the two algorithms that need to be implemented.
- Implement the GoalBoundingDijkstraMapFlooding algorithm. Starting from an initial node, this algorithm must calculate the GoalBounds for each of the initial outgoing connections. To do that, it runs a Dijkstra search, but when a new node is selected it detects the initial outgoing connection that was used to get to the selected node, and updates the bounding box for that connection accordingly[1]. Calculating the whole GoalBoundingTable for the scenario provided will take a large amount of time[2]. If necessary, you can remove part of the scenario, and generate a smaller navmesh.
- Implement the GoalBoundingPathfinding Algorithm. It should be similar to the NodeArrayA* with the difference that when processing a child node, the algorithm tests if the goal position is inside the bounds of the connection towards the child node. If the goal position is not inside the goal bounds, the child node is discarded immediately.

Level 4 – Comparing the pathfinding algorithms

- Analyse the differences in performance between the original A* algorithm (w euclidean distance), the NodeArray A* algorithm (w Euclidean distance), and the GoalBoundingPathfinding. Consider aspects such as fill, nodes visited, total processing time and processing time per node (and other aspects if relevant). Which version has the best performance and why? Include this analysis in the report. Also Include the tables calculated in Lab 4 (comparing the different variations of datastructures) in the final report.

Level 5 – Path Smoothing

- This level can be implemented in parallel with levels 2-4. The goal is to implement a path smoothing algorithm to smooth the path returned by the pathfinding algorithm. The path smoothing algorithm must be based on the path smoothing technique for grids lectured in the theoretical classes.
- Integrate the path smoothing in the Pathfinding Manager.
- Print the smoothed path.

[1] Using a polygon-as-node representation, this would correspond to using the polygon's vertices (or min and max) to update the bounding box. However, RAIN AI uses an edge-as-node representation, so we will have to use the edge for the update.

[2] It corresponds to performing around 10 000 Dijkstra runs.

Level 6 – Optimizations

- Select 2 relevant optimizations that can be made in your code and implement them. Justify why they are important or not, create a table with the method’s execution time and memory and number of calls before the optimization and after. Finally, briefly discuss the results obtained.
    - One of the optimizations can be a memory optimization. For instance, you can try to optimize the memory used by the GoalBoundingTable.