Artificial Intelligence for Games

Project 4 – Goal Bounding

Project Report

Prof. João Dias

Group 7
Samuel Gomes 76415
Daniel Rodrigues 65881
André Fonseca 84977

Introduction

In the 4th project our group decided to implement the goal bounding algorithm (developed by Steve Rabin in January 2015) in the navigation mesh provided in our lab 4, to reduce the number of nodes explored by the A* algorithm.

1. Precomputation of the map

The first step of the algorithm is to precompute all the bounding boxes for each edge, creating a save file to avoid the need of making the same calculations each time the program is run.

1.1 Dijkstra´s Flood fill algorithm to compute the edge bounds

In the class *NodeRecord* (that holds information about a node), we added an array of bounding boxes that represents the bonds of every edge. We also added a *startingEdge* value that is temporarily used by the flood fill to mark the most easily reachable edge of the initial node.

Then, starting on a given node, we traverse all the outgoing edges of that node by calling RAIN´s *outConnections* function. We then traverse the connections of every node until all reachable nodes have been visited. While doing so, we mark each node with the corresponding *startingEdge*. After the flood fill, we modify the Bounding boxes of each edge by allowing them to include nodes that are reached through that edge. We repeat this process for every node in the navigation mesh.

1.2 Read and Write to a file

For this step, we implemented the classes *FileReader* and *FileWriter*, to be able to save the data associated with each edge´s bounding box to a file and to read that data from the created file.

These two classes consist in:

For the *FileWriter*, we traversed all the *boundingBox* arrays in the list of *NodeRecords* present in a given *NodeRecordArray* and we write an XML hierarchy that consists in (1) tag *nodeRecord* representing an object of the class *nodeRecord*; (2) tag “bounds” representing the bounds related to that *nodeRecord* and (3) the tags *minX*, *minZ*, *maxX* and *maxZ* that contain the bound´s values.

For the *FileReader*, we traverse the file and we extract all the *nodeRecord*´s bounds value, creating a new *NodeRecordArray* that is used to replace the original one in the search algorithm class.

In this two classes, we used the *StreamReader* and *StreamWriter* classes to be able to cope with these file operations.

```
<nodeRecord>
</nodeRecord>
<nodeRecord>
  <bounds>
      <minX>-172</minX>
      <minZ>-197</minZ>
      <maxX>199.5</maxX>
      <maxZ>-164.15</maxZ>
  </bounds>
  <bounds>
      <minX>-197</minX>
      <minZ>-197</minZ>
      <maxX>-172</maxX>
      <maxZ>-170.4</maxZ>
  </bounds>
```

Fig.1. Example of the XML save file

2. Modification to the search algorithm

The difference between the original A* pathfinding algorithm and the one implemented in this project consists in the number of the processed nodes.

In order to reduce the number of processed nodes, we applied a set of restrictions that will verify if the goal node we are seeking falls within the bounding box of the edge we are traveling through; and if it doesn't, the node is not processed (and we continue directly to the next child).

So, by applying these modifications to the algorithm we decrease the overall time to compute a given path.

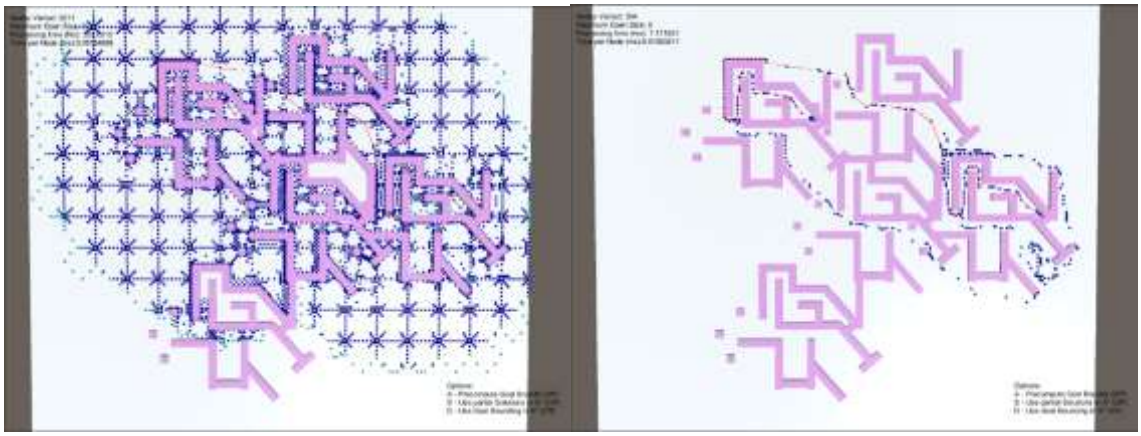

Fig.2. Comparison of path finding around the structure
between A* without (left) and with (right) Goal Bounding

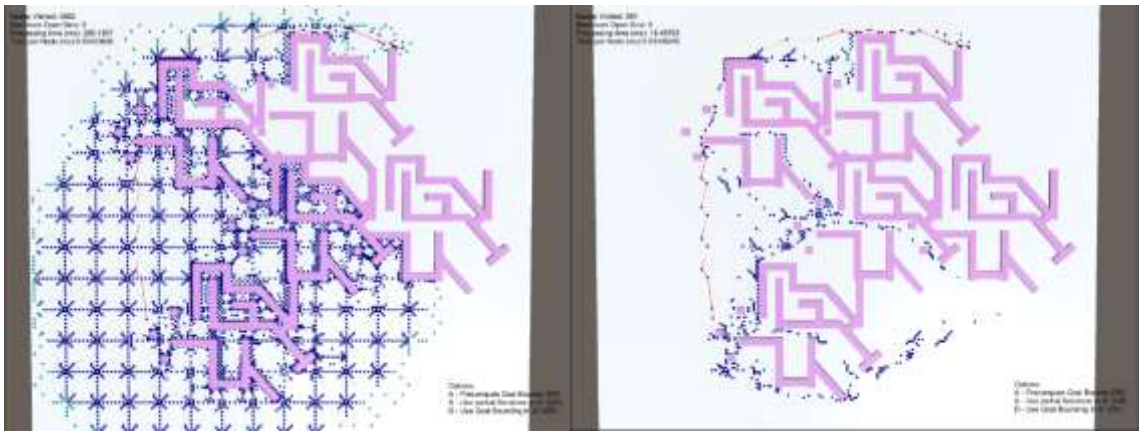

Fig.3. Comparison of path finding inside the maze
between A* without (left) and with (right) Goal Bounding

3. Fixes and Possible Optimization

As we implemented the goal bounding, several problems were fixed:

- The first problem we faced was a lack of reference to the node´s bounding boxes. We fixed this problem by applying some changes to the *NodeRecordArray´*s constructer, namely the creation of those references and the application of default values to the limits of the bounding boxes.
- The second problem we faced was an error caused by a minor bug in our *FileReader* class causing errors in the pathfinding algorithm.
- The last and most problematic issue we faced was caused by not considering the fact that a processed child node in the A* search could be the start or goal nodes, that had additional connection polys associated with them. We fixed this issue with two changes:
    - Firstly, we stopped using goal bounding in those two particular nodes;
    - And second, since start and goal nodes from previous searches were not removed from the navigation mesh, we added them to it.

As our implementation takes a long time (approximately half an hour) to pre-compute the bounding boxes, one possible optimization that could be considered consists in reducing the pre-computation time using goal bounding gates that limit the number of bounding boxes to pre-compute.

Fig.4. Goal Bounding Gates Illustration