Abstract

„Implementation of a Pathfinding-Algorithm for a voxel-based 3D-space in Unity“
„Practical project as an example of a prototype in a Minecraft style cubic grid “

The idea behind the project is, to implement a fully working 3D-pathfinding-Algorithm in Unity. The said Algorithm is supposed to be based on the A*-Algorithm. The Algorithm should be optimized speed and performance. The project is created and will be presented as a prototype in Unity.

The Focus of this project will be on the Search Space Representation and the A*-Algorithm. At the beginning of the paper there will be some information about Search Space Representations like: Waypoint Graphs, Grids, Navigation Meshes, Octrees and Sparse Voxel Octrees. Aside from that there will be information given about Hierarchical Pathfinding.

The next part focuses on the Algorithm itself, how it works and how to manipulate it. The Algorithm can be manipulated by overestimating and underestimating the heuristics, in addition to that Octile and Euclidean Distance are presented as possible heuristics. The paper is also referencing similar Projects like Mercuna 3D and Warframe to show the relevance. Furthermore, trends are shown which suggest that 3D-Pathfinding in Video Games and other areas will be much more relevant in the future.

The third Chapter shows a plan created for this Paper to evaluate and compare a common Octree to a Sparse Voxel Octree. The criteria to Compare the two Search Space Representations are: speed, accuracy and use of resources. The comparison and evaluation will take place with a fitting Algorithm, because it can’t be compared or evaluated without and Algorithm. The Algorithm will run on both Search Space Representations.
The A*-Algorithm will be compared with the Dijkstra-Algorithm, the will both be analysed first and compared afterwards. The criteria to analyse and compare the Algorithms can be differed, there are criteria to compare and there are exclusion criteria, when the exclusion criteria are not fulfilled the Algorithm is rendered useless for the purpose of this paper. The two exclusion criteria are: the Algorithm has to be working on graphs otherwise it cannot be implemented for the Search Space Representation and the second criterion is that the Algorithm has to be influenced by heuristics so we can manipulate it for our purpose. The three criteria to Compare the algorithms are: speed, accuracy and complexity, speed and accuracy have to be put in relation, but speed is more important for the purpose of this project. The complexity becomes important when accuracy and speed of both Algorithms are similar, if one Algorithm is way more complex than the other one but speed and accuracy are similar the less complex Algorithm will be chosen, furthermore an Algorithm could be too complex to be implemented in the bounds of this project.

Chapter four describes how the practical project is actually created, it shows the creation and optimization of the octree and the implementation and optimization of the A*-Algorithm onto the Sparse Voxel Octree. It also shows how Search Space Representation and Algorithm are evaluated and compared. The last part of the chapter describes how the Project is visualised.

The first thing described is, how the common Octree is created and how many nodes are needed to create a 6-layer Octree. A common Octree with more than 6 layers was not possible because Unity took to long to create it, a 6-layer Octree needs around 300.000 nodes to be created. That's why the common Octree was optimized into a Sparse Voxel Octree, the Paper describes how it was optimized and differs from the common Octree.
The next part is a step-by-step description of the implementation of the A*-Algorithm. It is possible to implement the Algorithm with the description given in the paper. Afterwards the evaluation of the Algorithm and the Search Space Representation is described, it shows that the Dijkstra-Algorithm cannot be used in the Project because it has no Heuristic but it will be used to Compare A* and Dijkstra.
Common Octree and Sparse Voxel Octree are compared and the optimized Sparse Voxel Octree is way better and used in the final project. At the end of this chapter the visualisation is described, you can see all the nodes as outlined cubes every node with children is red and the ones without are green. The found path is visualised as solid blue cubes.

The last Chapter Contains the Results of the Project and shows whether the goals were met or not. The three aspects are Algorithm, Search Space Representation and visualisation.
The optimized A*-Algorithm turned out to be a thousand times faster than the common A*-Algorithm, furthermore the optimized A* is 24.000 times faster than the Dijkstra Algorithm, all numbers are for the given test environment, it can differ for more dense environments. All previous set goals for the Algorithm are met.
The optimized Octree with 8 layers and a thousand obstacles in the represented space contains about 40 thousand nodes, while the common Octree with 8 layers would need about 19 million nodes, which means that the common Octree needs about 450 times the space to store the nodes. The Sparse Voxel Octree would need more nodes the more obstacles there are, the common Octree always has the same number of nodes as long as the layer doesn't change. Every set goal was met for the Search Space Representation.
Every goal previously set for the visualisation was met, the found path is shown as solid blue cubes, the division of the space into the Sparse Voxel Octree is shown as outlined Voxels, where the ones with children are red and the ones without are green. Additionally, an NPC was created which travels along the found path from start to finish.

The Project was a success every goal could be met and besides a few set backs everything went great.