A* algorithm pseudocode

A finds the shortest path from a starting coordinate to a goal, using heuristics. To use the algorithm, first a graph is constructed from the existing coordinates, with identical index values. Coordinates that are impassable are added to a list of inaccessible points. After the algorithm has found the shortest path, the corresponding indices are returned and the path is constructed in the actual model.*

Input:

- Grid on the origin in an integer coordinate system (*point3d , list*)
- Inaccessible points (*point3d, list*)
- Indices of the inaccessible points (*int, list*)
- Start points (*point3d, item*)
- End points (*point3d, item*)
- Number of nodes in x, y, z directions (*int, item)*

Algorithm:

```
-   Definition: Heuristic(a,b) #only search in XY direction since the corridors all lead to commonly shared climbing points (the end points)
        o   (x1, y1, z1) = a
        o   (x2, y2, z2) = b
        o   return square root ($(x1 - x2)^2 + (y1 - y2)^2$)
-   Definition: A*(graph,start,goal)
        o   Use priorityqueue to check the cost (heuristic) of the current point's neighbours to the goal, until the goal is reached:
                §  New cost = cost so far
                §  If next point has a lower cost, take this as the new point and search this points's neighbours
                §  Came_from[next] = current
        o   Return_came from
-   Class: priorityqueue #definitions to manipulate the queue of points to search next
        o   Definitions: empty #returns length, put #heappush, get #heappop
-   Definition: tuple to point3d
-   Definition: point3d to tuple
-   Definition: reconstruct path(came from,start,goal)
        o   Return the path with the lowest total value, and reverse the list since it calculates the path backwards
-   Definition: find indices(path,points)
        o   indices = []
        o   for i in length of the path:
                §  create spheres on the path points
                §  for j in length of all points:
                        ·   check if the point is inside any sphere, return the indices of the points inside spheres # this could be optimized, as
```

the time increases exponentially with increased points/path list length

- **Class:** squaregrid(number of nodes in XYZ direction, removed points)
  - **Definition:** in bounds *#check if point is in bounds*
  - **Definition:** passable *#check if point is (not) in removed points*
  - **Definition:** neighbours #find neighbours of the point *#only in XY direction*
    - Results = []
    - Append((x+1,y,z),(x-1,y,z),(x,y+1,z),(x,y-1,z))
    - Filter in bounds
    - Filter passable
    - Return results

Implementation: *calling the search and initializing the needed lists*

```
total paths =  #nr of paths
#remove the points from the graph that have been shot away by the solar envelope
for every removed point in the actual model:
        Create points on the graph that have been removed
        Append these points to a new list called Removed Points
graph = Square Grid(W,D,H,RemovedPoints) #creating the graph
start = Point to XYZ(start point)
goal = Point to XYZ(end point)

#calling the search and reconstructing the path
came from = a_star_search(graph, start, goal)
reconstructed path = reconstruct_path(came from, start, goal)
for every point in the reconstructed path:
        append the point3d in path points
Path Indices = find indices(Path Points,all points) #outputting the indices from the real pointcloud
path = Path Points #outputting the path (on the graph)
```

Output:

- Shortest path (*point3d, list*)
- Indices (*int, list*)

applied a* algorithm:
Input:
All points of the graph
Points removed from the graph
Start point for the search on the graph
End point for the search on the graph
Number of points in the X direction of the graph
Number of points in the Y direction of the graph
Number of points in the Z direction of the graph

Output:
Shortest path on the graph
Indices of points on the shortest path