If you see this, check discord

Functions:

**Function** roundState(continuous state):

Responsibility: Herman

Use: Rounds a continuous state to a cell in the search space.

**Function** exists(continuous state, Set):

Responsibility:

Use: Returns true if roundState(continuous state) coincides with any roundState(x) for x in Set.

**Function** applyMotion(continuous state, i):

Responsibility: Yuxin

Use: Generates the i:th child of the continuous state based on the i:th motion in the motion set. It is possible to directly assign a cost based on the taken motion.

**Function** euclidean(continuous state, goal continuous state):

Responsibility: Herman

Use: Estimates the cost-to-go from the continuous state to the goal continuous state using the euclidean distance.

**Function** gridIndex(roundState(continuous state), terrainManager):

Responsibility: Herman

Use: Computes the grid index of the rounded state. Useful for pruning the tree.

**Function** obstacleDetection(gridIndex(roundState(continuous state), terrainManager):

Responsibility: Herman

Use: Checks the gridIndex of the state against the traversability grid to ensure that the cell is reachable. Returns true if cell is reachable.

Hybrid A\* Algorithm

Start algorithm(start continuous state xs, goal continuous state xg):

1. Initialize open list O
2. Initialize closed list C
3. Set the predecessor of the start node to null
4. Push start continuous state onto open list
5. **While** open list is not empty
   1. Pop least cost node x of open list
   2. Push the node into the closed list
   3. **If** the roundState of the node x is the same as the roundState of the goal node xg:
      1. Return node (this node will contain pointers back to the start)
   4. **Else**
      1. **For** every possible motion u in U
         1. Add motion to node to generate a child xsucc
         2. **If** the child is not in the closed list
            1. Calculate the cost-so-far (g) of the child as the sum of the cost to the predecessor and the cost from the predecessor to the child
            2. **If** the child is not in the open list **or** the cost of the child is lower than the cost of a vertex in the same cell

Assign the previous node x as the predecessor of the child xsucc

Assign the cost-so-far of the child g to the cost of the cell g(x\_succ)

Calculate the heuristic cost-to-go h of the child based on the distance between the child and the goal

**If** the child is not in the open list

Add it to the open list

**Else**

Update the priority of xsucc in the open list

1. **Return** null

Note: Same cell expansion (Algorithm 6, KTH paper) should be added before line 5.d.i.2.a.