Module 1

By: Andreas Lønes and Ole Halvor Dahl

Agenda Loop

The agenda loop is a while loop with end clauses:

1. if current_node == end_node: return True
2. if len(open_list) < 1: return False

If the algorithm reaches a node that is equal to the goal node, then the algorithm has found a solution. If the open list is empty, the algorithm will not find any solution.

The loop pops the first element from the open_list. The open_list is a list of all nodes that have been generated and not been chosen as the current_state. It is sorted by the F-value(heuristic + g) of each node, which means that the assumed best state is always chosen. When a node is popped from the open_list it is removed from the open_list and appended to the closed_list. The closed list is a list containing all nodes that have been chosen by the algorithm.

Generality

For other search problems the changes that would have to be made are that the loop as of now assumes that the distance between a node and its children always is 1. We realized that the distance between a node and its child should be stored inside the node.

This algorithm takes the goal_node as an argument. There might be other search problems where one has to calculate whether one has reached the goal differently. For example if one does not know where the goal is beforehand.

Heuristic function

The heuristic function calculates the minimal distance from a node to the goal node. This is the formula used by the algorithm:

node = the popped node, goal_node = the node one is trying to reach

abs(goal_node.y_pos - node.y_pos) + abs(goal_node.x_pos - node.x_pos)

It calculates how many steps one have to move vertically and horizontally to reach the goal node, given that there are no obstacles in the way. This is an admissible heuristic because it is not possible to find a path from a node to the goal that is shorter than this.

Generate successor states

When a node is chosen and it is not the goal_node, the algorithm generates the nodes' successor states. This is done by iterating over its children. A child of a node is another node that is either directly to the left, right, above or under the chosen node(not diagonally).

When the node is added to the open_list its f-, h- and g-value is calculated by the method "attach_and_eval(child, parent, end_node)".
After all children is added the open_list, it is sorted again.

If the child has already been found, we check to see if the new way to the node is shorter than the previous route. This ensures that when you finally choose a node, you are guaranteed to have found the shortest path to it.

```
elif ((current_node.g + 1) < child.g):
        attach_and_eval(child, current_node, end_node)
        if child in closed:
            prop_path_imp(child, end_node)
```

If the child node is in the closed_list or the open_list and the childs g-value is more than the parents g-value+1, then the g-value of the child is updated because a shorter path to that node has been found.