Module 1
IT3105

Simon Borøy-Johnsen
MTDT

October 5, 2015

General

I implemented a general class for the basics of the A* algorithm. This general class contains three fundamental data collections;

- *Open list*: Contains all the discovered, unexpanded nodes. Nodes are added here when discovered by a neighbour node, and popped when appropriate.
- *Closed list*: Contains all the expanded nodes. Nodes are added here after being expanded.
- *Node ID cache*. Keeps track of all nodes. When a node is discovered, it is added to the cache. Not necessary for the algorithm, but is used in order to avoid iterating through the two other lists when discovering neighbours. Implemented as a dictionary.

Some functions are also used;

- f: Picks next node. Usually picks the node with the lowest $g+h$ value, but this can be overridden in order to implement for example breadth- or depth first searches using the algorithm.
- *pick_next_node*: Picks next node using f. Then closes the node and returns it. This is called in the beginning of each loop in the agenda in order to find next node to expand.

No assumptions about the specific details of the problem space are made in the general A* class. The general A* algorithm simply pops nodes from the open list, generates its neighbours, then picks a new node according to the f function. All functionality concerning the calculation of g and h values, and generating neighbour nodes is implemented in the node classes.

I implemented a general node class covering the basic functionality of a node; keeping track of parents and children is implemented here. All nodes

have states. The states are used to calculate g and h values, in addition to generating neighbour nodes. The state is also used as key in the node ID cache. The node class may be initialized with an end state, which in some problems must be used to calculate the heuristic, and to check whether a node is a goal node or not.

In order to run the algorithm, a problem specific node class must be implemented. This class should inherit the general node class and implement four abstract functions;

- *generate_neighbours*: Generates neighbours of a node.
- *generate_id*: Generates the unique ID of a node.
- *heuristic*: Calculates the node's h value.
- *is_solution*: Returns whether the node is a solution or not.

Problem specific

The state of a node is simply a tuple *(x, y)*, where x and y are the Cartesian positions in the 2D task space.

- *generate_neighbours*: Checks all adjacent states (not diagonally), and creates a new node if the state is inside the grid, and there are no obstacles there. Adds the new node to the current node's neighbour list.
- *generate_id*: Returns the string *'x.y'*, where x and y are the positions in the node's state.
- *heuristic*: In this problem, the Manhattan distance was used as the heuristic. This is admissible, so the optimal goal node will always be expanded before any other.
- *is_solution*: Returns whether the node's state is the same as the goal node's state or not.

In addition to these four functions, the function for calculating the arc cost can also be overridden. In this project, all arc costs equal 1, so the arc cost function does not raise any exceptions if not implemented.