Assignment 2: Applying the A* search

Jakob Keller and Eivind Dalholt

Task 1:

In the first task we had to actually implement the A* algorithm. We started by making a “Node” object, which is able to represent every walkable square of the board, and search the adjacent nodes in the x- and y-axis.

We then implemented A_star which keeps track of the discovered nodes ready to be traversed (activeList), and already visited nodes (by knowing the parent of the last visited node). The implementation uses the heuristic (difference in x-position plus difference in y-position, from current node to end position as an absolute value) and adds this with the cost of visiting the node. The node with the lowest total cost in the activeList is chosen as the next node in every iteration. If we no longer can find a better cost, we go back to the parent and try the second best node. This will eventually give us the shortest path.

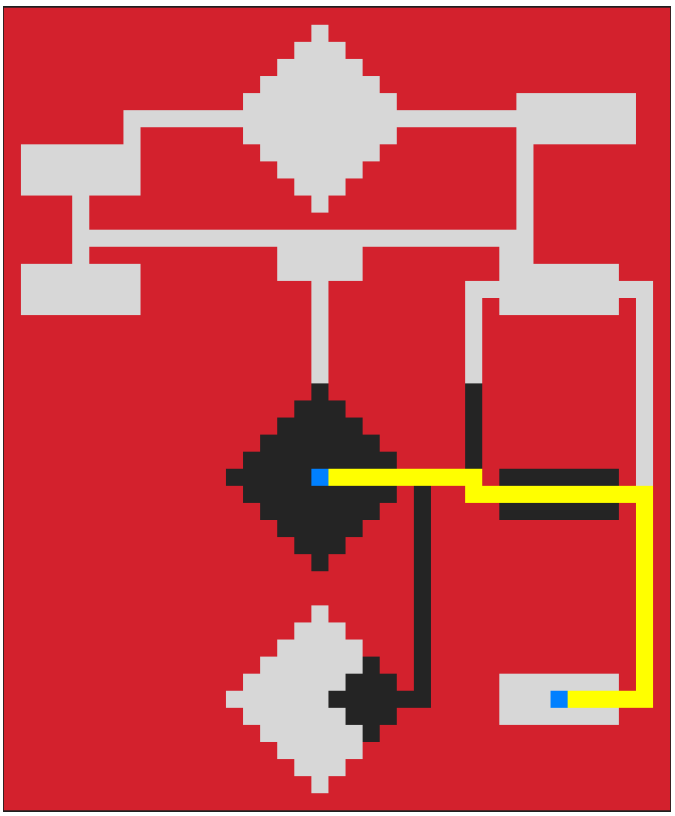

Figure 1 - Result of task 1

Task 2:
When we calculated the heuristic in task 1 we thought of calculating it by the difference in x- and y positions because we only are able to move four ways (not diagonally). Coincidentally this was the same way as the method for Manhattan distance, and task 2 was therefore also solved.

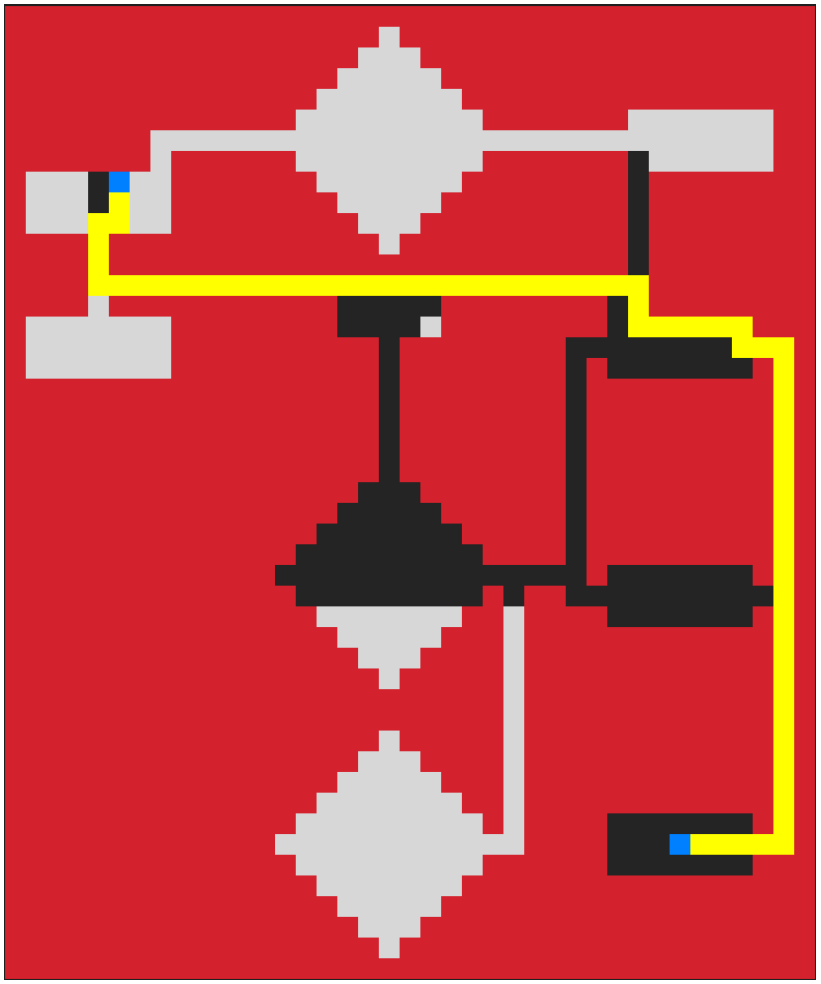

Figure 2 - Result of task 2

Task 3:

In this task we had to account for variable cost. We are anyway considering all opportunities, and append all adjacent nodes with a cost over zero, as negative is non walkable surface (code example under, figure 3). The algorithm is sorting the candidates by the total cost afterwards, and the best next node is chosen. By doing this A* is considering both the positive and negative aspects of visiting stairs or crowded areas, versus a slightly longer, less crowded or a path without stairs.

```
def find_inherit(self):
    x = self.current.pos[0]
    y = self.current.pos[1]
    self.inherit = []

    for i in range(4):
        if i == 0:
            if (map_obj.get_cell_value([x+1, y]) > 0):
                self.inherit.append([x+1,y])
        elif i == 1:
            if (map_obj.get_cell_value([x, y+1]) > 0):
                self.inherit.append([x, y+1])
        elif i == 2:
            if (map_obj.get_cell_value([x-1, y]) > 0):
                self.inherit.append([x-1, y])
        else:
            if (map_obj.get_cell_value([x, y-1]) > 0):
                self.inherit.append([x, y-1])
        i += 1

    self.activeList.remove(self.current)        You, 28 mi
```

Figure 3, code example

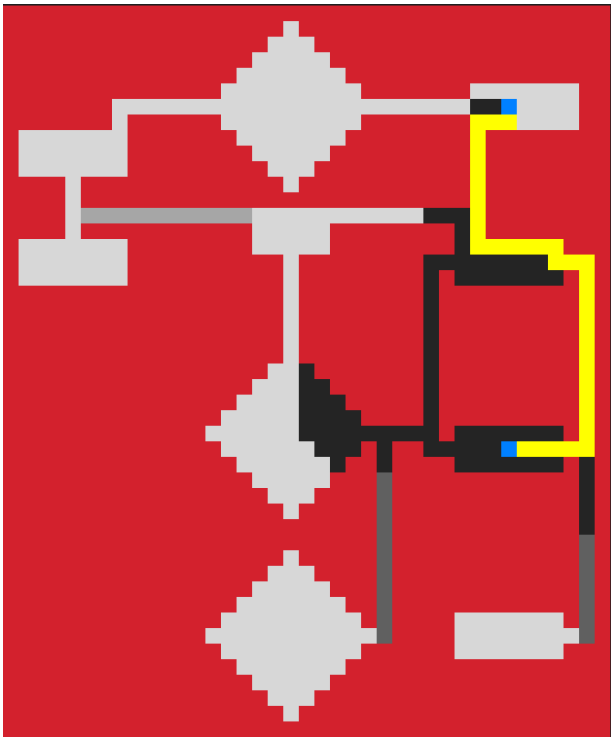

Figure 4 - Result of task 3

Task 4:
Implemented by the method described in task 3. Result below, A* is now changing path because of the cake party.

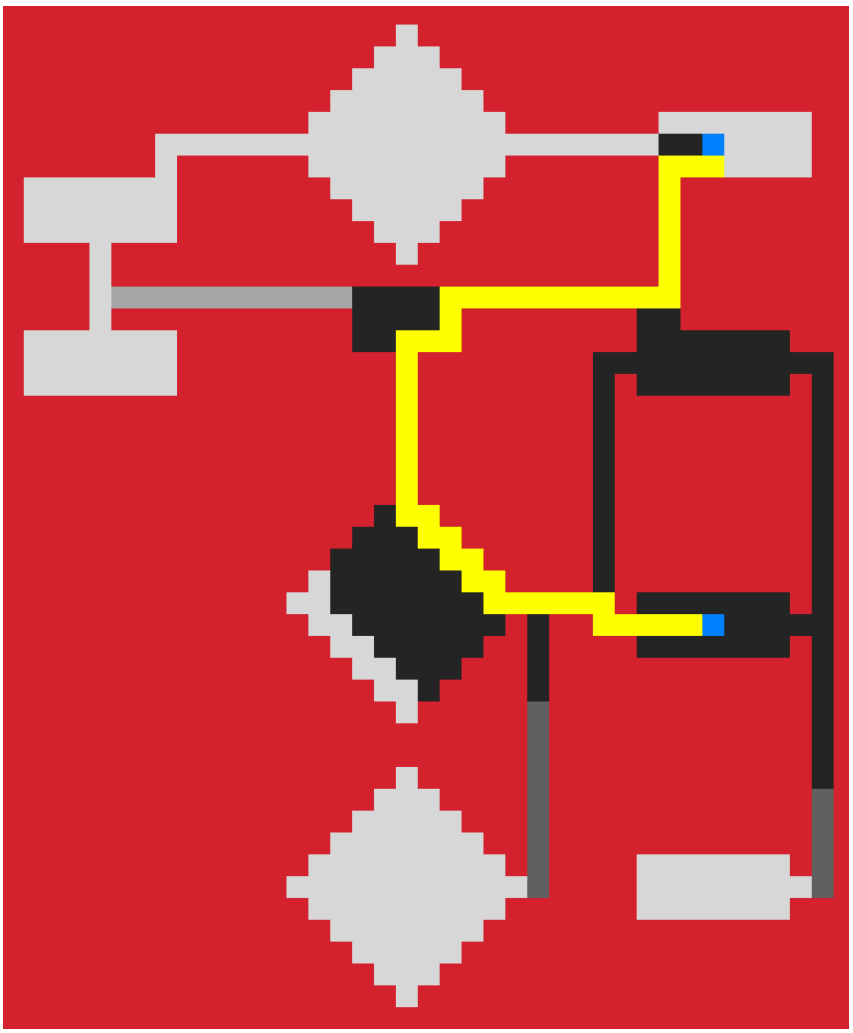

Figure 5 - Result of task 4