# IT3105 - Artificial Intelligence Programming
# Project 1

Kristian Våge and Bjørn Bråthen

September 24, 2015

**Abstract**

This is a collection of the reports written for modules 1, 2 and 3 for the course IT3105.
The modules are listed with its corresponding description here:

- Module 1: Using A* to solve Navigation Problems.
- Module 2: A*-GAC, A General Constraint-Satisfaction Problem Solver.
- Module 3: Using A*-GAC to Solve Nonograms.

These have been written and completed for the demo day October 6th 2015.

# Using A* to solve Navigation Problems

Kristian Våge and Bjørn Bråthen

September 24, 2015

# 1 Central aspects of your A* program

## 1.1 The agenda and how it is managed

The agenda is awesome. list and Heapq...

## 1.2 Generality of the program

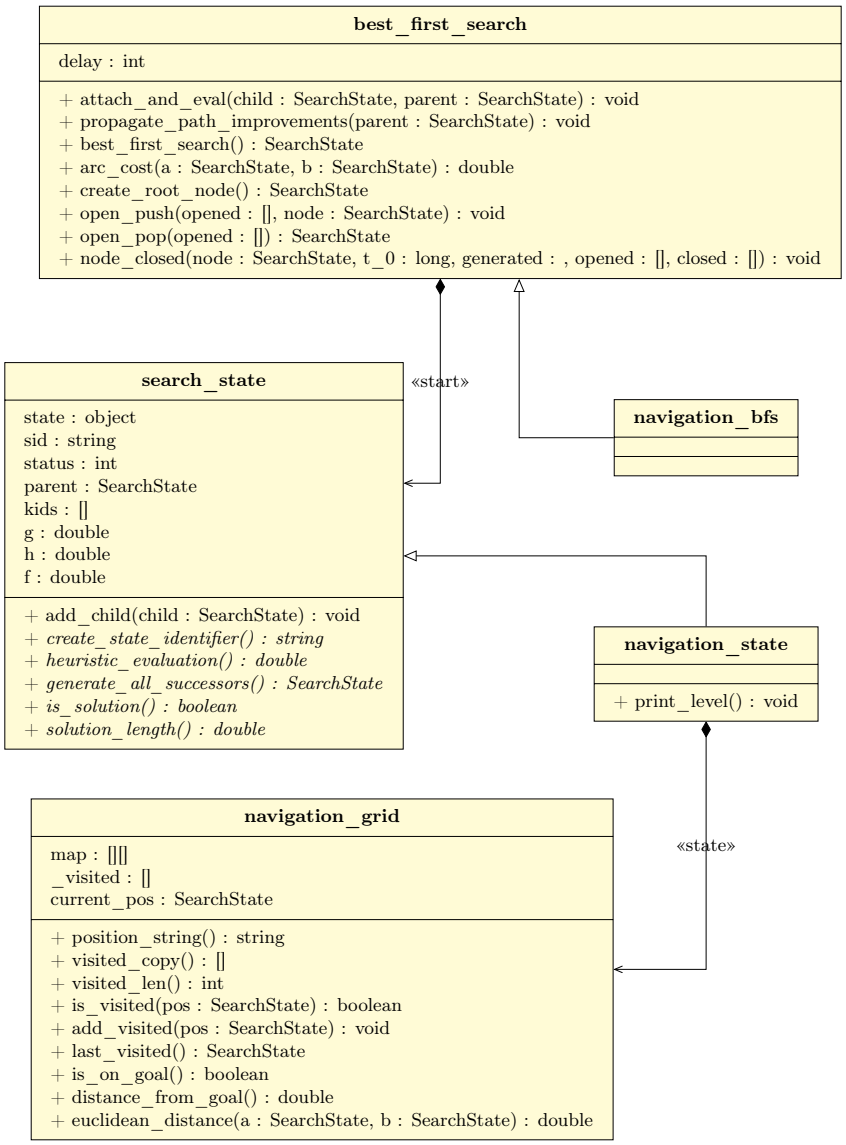


Figure 1: Class diagram for the navigation task.

Figure 3 shows the class diagram for the navigation problem.

```
class HelloWorldPrintable(object):
  def __init__(self):
    self.string = 'hello, world!'

  def __str__(self):
    return self.string
```

```
7
8 print(HelloWorldPrintable())
```

## 1.3 Heuristics

Both the *Manhattan distance* and *Euclidean distance* has been tested on the navigation problem. Each adjusted the behaviour of the agent. Using the manhattan distance the agent seem preferring to minimize the amount of $90deg$ turns before reaching the final goal. The euclidean distance seems to do the quite opposite, and the agent goes straight towards the target even though the agent cannot move diagonally. It tries to emulate the behaviour of walking diagonally and follows a diagonal line between the current position and the goal. In the example board number two (Figure 2), the manhattan distance wins by generating least amount of nodes, because it can just move right and then up without encounter an obstacle. The euclidean agent has to avoid an obstacle and therefore is generating a few more nodes. On the other hand, in example 5 (Figure 3), the euclidean agent slips through the opening between the two last obstacles, while the manhattan agent needs to explore the area before the right-most obstacle. The performance of the agent depends on the heuristics compared with the structure of the board.

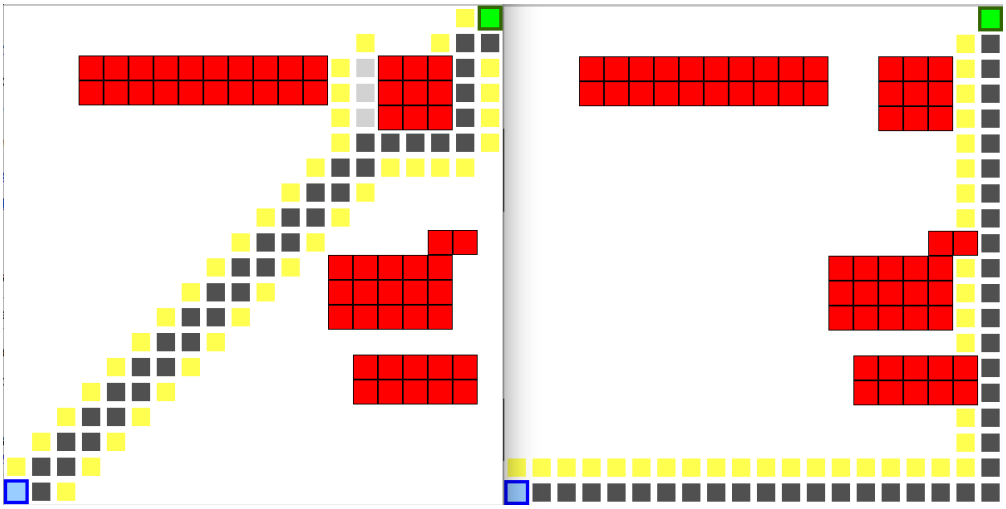

Figure 2: Comparing euclidian distance and manhattan distance in board example 2

## 1.4 Generating successor states

From each viable state, a maximum of four possible successor states can be created, each by moving in every direction from its currently held position; NORTH, SOUTH, EAST, WEST. Other considered limitations are that the new position should be inside the given dimensions of the board, and that obstacles are not walkable, together with unnecessity to visit positions that previously have been visited. Every state that is within the scope of those limitations, is created and added to the heap.

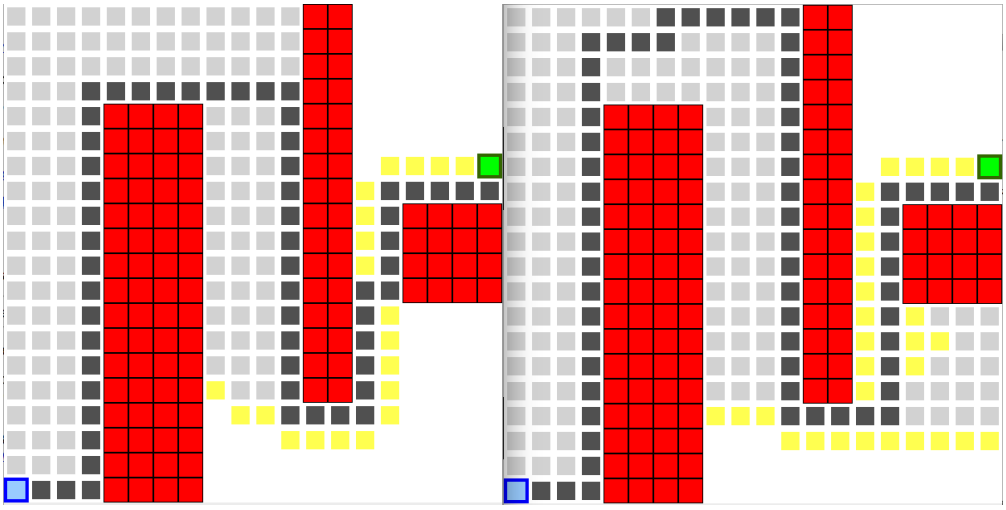

Figure 3: Comparing euclidian distance and manhattan distance in board example 5