Assignment 2 - The A*-algorithm

Morgan Heggland & Patrik Kjærran - MTIØT
TDT4136 - Introduction to Artificial Intelligence

September 16, 2019

1 Introduction

Assignment 2 in TDT4136 revolves around the A* algorithm, and asks us to implement and use the algorithm in order to find our way through Samfundet. This document serves as an assignment report to complement and explain our solutions as they appear in the submitted code. In the following sections we will describe our approaches and designs to the different exercises as well as the results we obtained.

2 File structure

Apart from the files supplied for this assignment, we have added the following files and folders:

- **solutions:** folder containing the solutions to all parts of the assignment, formatted as `.png` files.
- `min_heap.py`: array implementation of a priority queue in the form of a min heap, see below for details.
- `part_[1|2|3].py`: script for generating a solution to the corresponding part of the assignment.
- `a_star.py`: generalized implementation of the *A*-algorithm*, see below for details.

3 Data structures

The A* algorithm requires a way to store discovered states as well as retrieving the state from the discovered state space which has the lowest value, evaluated by a specific attribute. For this, we have chosen to implement a *priority queue* in the form of a *min-heap*. This implementation is based on the psuedocode described in *Introduction to Algorithms* [1].

4 A* implementation

We have developed a generic A* algorithm based on the pseudocode provided in the supplementary document regarding A*. The implementation is generic in the sense that it can be used in any situation where A* may be used, as it takes all situation-specific functionality as inputs to the function. This includes

- `start_state`: the starting state
- `heuristic_func`: the heuristic function to use
- `succesors_gen`: a successor function to generate successors
- `goal_predicate`: a function that determines whether the goal is reached
- `cost_func`: the cost function to use

The code should be pretty straight forward and can be found in `a_star.py`.

5 Assignment tasks

In the assignment tasks, the objective is to traverse through the halls and bars of Samfundet in order to reach a goal position from a starting position, with varying degrees of complications. For the tasks, we need to implement all the input functions of the A*-algorithm. The successor generator, goal predicate and cost function are the same for all 3 parts (except for a minor adjustment to the goal predicate in part 3).

The **successor generator** simply returns the adjacent positions to the current postion that are not obstacles. The **goal predicate** simply checks if the current position is the goal position (known *a priori*). The **cost function** simply checks the cell value of the target position in order to evaluate the cost of transitioning to that position/state. It should be noted that this is redundant for part 1, but yields the same solution as all valid positions have the same cell value.

5.1 Part 1

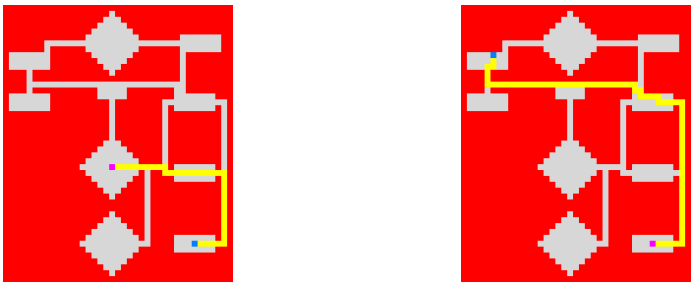
In part 1, we simply need to traverse from a starting position to a target position without any further complication.

5.1.1 Implementation

In order to solve the task, we need to define the remaining input: a **heuristic function**. For this task we chose to use the **manhattan distance** from the current position to the goal position as the heuristic. As we can only move in a 2D grid pattern, this heuristic will produce a reasonable estimate while never overshooting. Compared to euclidean distance, the manhattan distance will be closer to the actual distance we must travel and is therefore preferrable over euclidean distance as a heuristic.

5.1.2 Results

A visualisation of the paths our algorithm produces as the shortest path from start to goal can be seen in Figure 1a and 1b below.



(a) Task 1 (b) Task 2

Figure 1: The solutions to part 1.

5.2 Part 2

The objective for part 2 is the same as in the previous part, but with costs associated with walking to different cells. A higher cost reflects that the area is more difficult to traverse, either due to the area being crowded or when climbing stairs.

5.2.1 Implementation

Seeing as the only difference between part 1 and 2 is the addition of different costs, the only input function we would need to change is the cost function. However, as we were able to use the cost function for part 1 as well, the implementation for part 2 is exactly the same as described in section 5.1.

5.2.2 Results

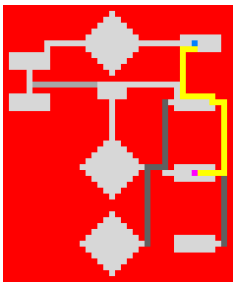
A visualisation of the paths our algorithm produces as the shortest path from start to goal can be seen in Figure 2a and 2b below.

5.3 Part 3

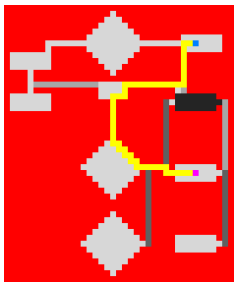
For part 3, additional complexity is added by introducing a moving goal, moving from the right end of the map towards the left, in a straight line. The different cells are still associated to different costs.

5.3.1 Implementation

In order to simulate the moving goal, a `tick()` function is provided, moving the target every fourth call. To incorporate this in the algorithm, we modified



(a) Task 3



(b) Task 4

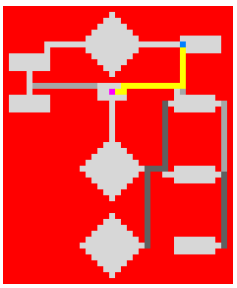
Figure 2: The solutions to part 2.

the `goal_predicate` function which is called once per search iteration. Here, we simply call the tick function to realize the moving goal.

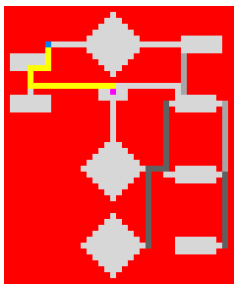
The **heuristic function** has to be slightly modified as the goal is no longer stationary. In order to make a better prediction of the estimated distance to the goal state, we made a linear approximation by calculating the displacement that will occur in the number of ticks it will take for us to get to the current goal position. We then calculate the **manhattan distance** to the corrected goal cell, using this as the heuristic value.

5.3.2 Results

In addition to running the algorithm with a goal moving at one fourth of the "search speed", we also ran an experiment with the goal moving at half the "search speed" to see if we would take a different path. A visualisation of the paths produced as the shortest path from start to goal can be seen in Figure 3a and 3b below.



(a) Task 5



(b) Task 5, friend moving twice as fast

Figure 3: The solution to part 3 with a bonus solution where the friend runs faster.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2009.