Deadline: 2019-09-20, 23:59
Recitation (øvingsforelesning): 2019-09-11, 16:15

Applying the A* Algorithm

Purpose: Gain hands-on experience with best-first search using the A* algorithm.

1 Overview:

In this assignment, you will become familiar with the A* algorithm by applying it to a classical use case for A*, namely that of finding the shortest path in a two-dimensional grid-like world.

The assignment consists of three parts. You need to make a passable effort on the two first parts in order to get this assignment approved.

Each of the three parts specifies a set of required deliverables. All parts include both programming and report writing.

2 A* Implementation

In order to solve the problems in this assignment, you first need to obtain an implementation of the A* algorithm by either (a) writing it from scratch in the language of your choice, or (b) downloading it from the internet. It is strongly recommended that you write the code yourself, since this will provide you with an in-depth understanding of this important AI algorithm. The accompanying document entitled "Essentials of the A* Algorithm" can be of assistance.

Should you choose to download a version of A*, then you will not receive much (if any) assistance from the course assistants, unless you fortuitously download code with which they are familiar. You cannot expect them to spend a lot of time trying to learn it.

Whether you implement or download the code, you must include it (along with appropriate comments) in your hand-in on Blackboard.

3 Path finding in 2D Games

A common demonstration problem for A* is that of finding shortest paths in two-dimensional square-grid boards. Imagine for example a two-dimensional top-down perspective video game—in such games, you might move your character around by clicking on locations in the game board to which you want to move.

The game would then calculate a specific path — a sequence of adjacent cells — for the character to follow to get to that location. This is called *path finding*, and is a common use case for the A* algorithm.

See Figure 1 for an example of such a 2D game board. The goal is to find the shortest path from the marked A (top left) to the square marked B (bottom right). The gray cells represent obstacles/walls which have to be avoided by the algorithm. Assuming the game character can only move in the four cardinal directions—north, south, west, east—the shortest path will be as shown in Figure 2.

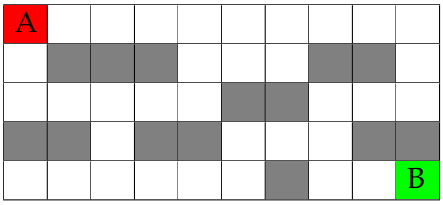

Figure 1: Example of pathfinding problem in a 2D game world. The goal is to get from A to B while avoiding the gray obstacles. The character can move north, south, west and east.

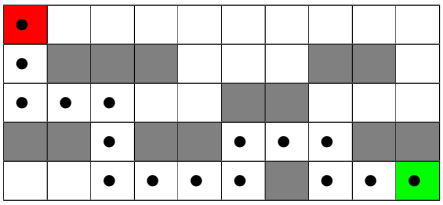

Figure 2: A solution to the pathfinding problem in Figure 1.

For this problem, you will implement a program that reads in a board configuration csv file describing where the obstacles are located and where the start (A) and goal (B) squares are. You are supplied an example of such code in Map.py. You will then find the shortest path from the start to the goal using the A* algorithm.

This problem consists of two mandatory parts and a third optional part. In the first part (Part 1), the game board will only consist of open cells or blocked cells as seen in the example above. In this task, all open cells have the same cost attached to them - the cost of movement. In the second part (Part 2), the cells will have different costs attached to them. In other words, the shortest path will no longer be determined merely by the number of cells traversed, but by the sum of the costs of the cells traversed. In the third part of the problem (Part 3 - optional), the goal cell will move as time goes on, requiring you to account for this movement in your algorithm.

In all three parts of the problem, you will be asked to visualize your program's results. The specifics of the visualization are up to you, but the board configuration and the calculated path should be clearly visible. The visualization can for example be in the form of tables (as above), text or simple 2D images. A possible textual representation of the path found in Figure 2 is shown in Figure 3, while an image-based representation is shown in Figure 4. (The image was drawn using the Python Imaging Library, which is a good choice if you're using Python for this assignment.). The assignment handouts contains some code for visualizing the grid-world that you can use as a starting point.

```
o . . . . . .
o ###...##.
ooo . .##...
##o##ooo##
 . . oooo#ooo
```

Figure 3: A textual representation of the path found in Figure 2.

Figure 4: An image-based representation of the path found in Figure 2.

4 Part 1 - Grid with obstacles:

In this assignment, you take on the (familiar?) role of a student trying to navigate Samfundet, see Figure 5.

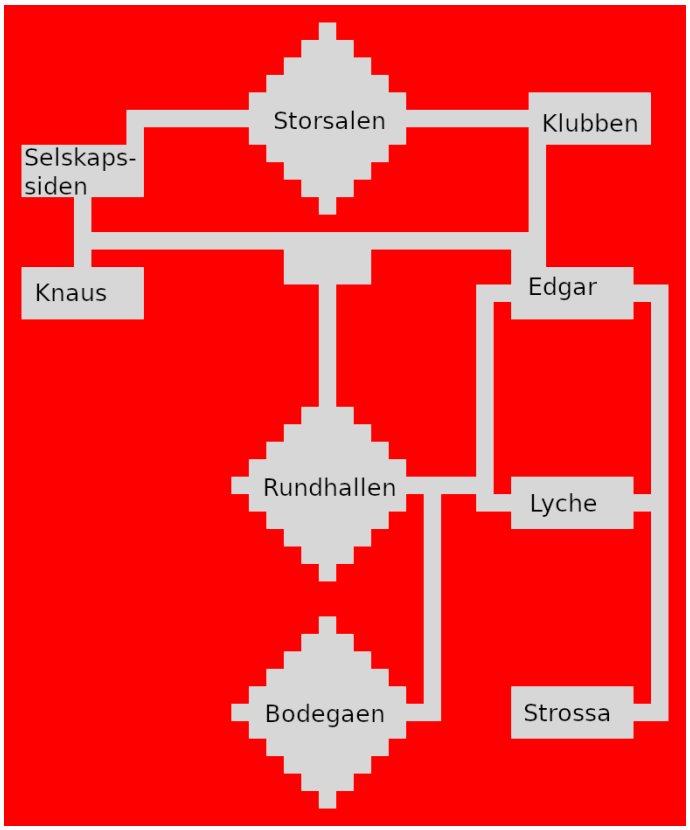

Figure 5: A 2D-world representation of Samfundet.

4.1 Task 1:

You and your friend arrived at Samfundet only five minutes ago, but you've already managed to get separated. Being the resourceful person that you are, you call your friend, which tells you that they went looking for you and is currently located at Strossa. Your task is therefore to find the shortest path from Rundhallen (your location) to Strossa as seen in Figure 6 using your implementation of the A* algorithm.

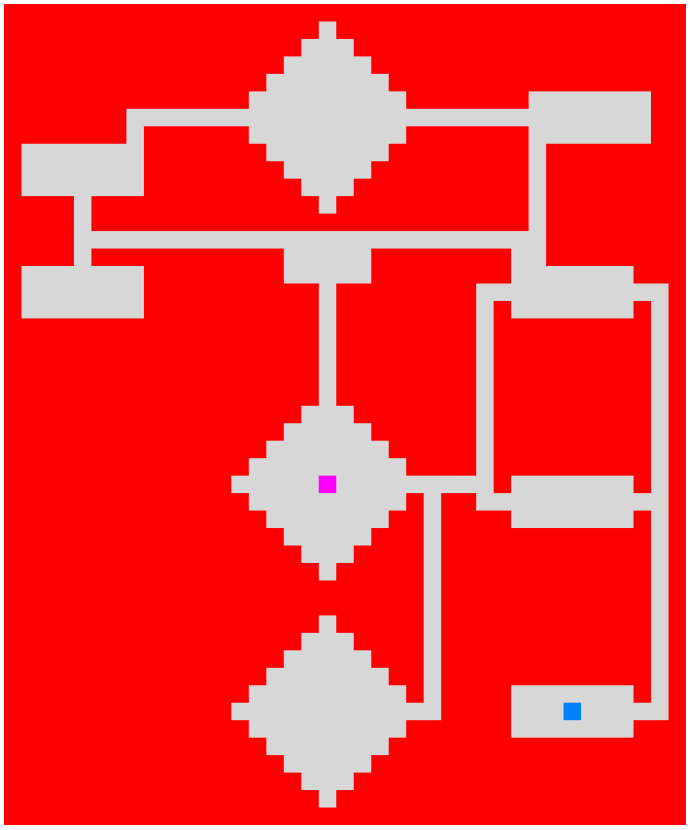

Figure 6: Task 1

4.2 Task 2:

When you arrive at Strossa, your friend is nowhere to be found. You deduce that they have probably moved on and you missed them in the stairs. You call your friend again and find out that they are now at Selskapssiden. Your task is now to use your A* implementation to find the shortest path from Strossa to Selskapssiden as shown in Figure 7.

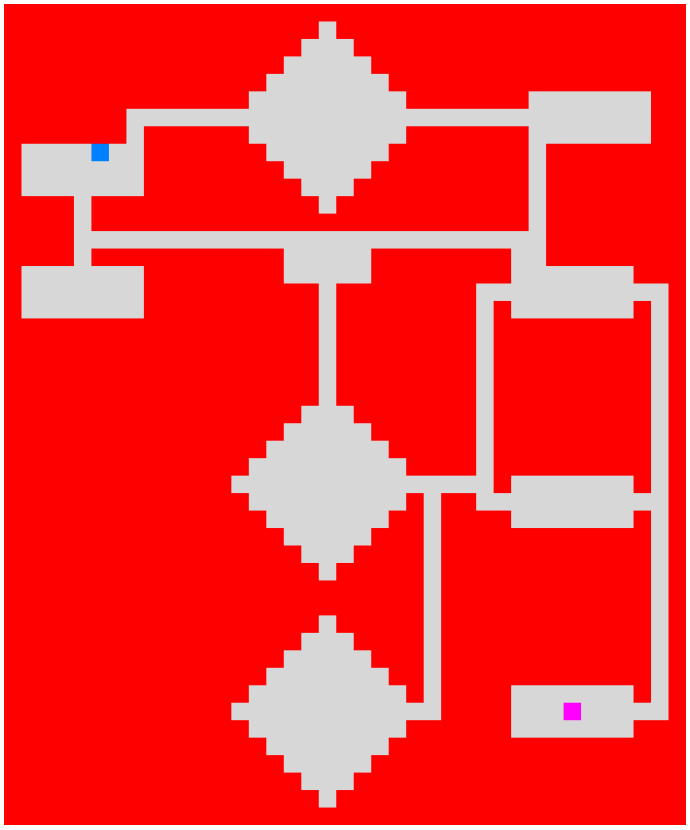

Figure 7: Task 2

For a given state in the A* search, i.e. a given cell on the board, the successor states will be the adjacent cells to the north, south, west and east that are within he boundaries of the board and that do not contain obstacles. Suggestions for the heuristic function $h()$ for this problem are to calculate either the Manhattan distance or the Euclidean distance between the current cell and the goal cell.

In the code provided in Map.py you can generate the two scenarios by instantiating a Map_Obj with the corresponding task number i.e map_obj = Map_Obj(task=1) for Task 1 above. Further information about the use of the map_obj is available in the Appendix.

Deliverables:

- Well-commented source code for a program that finds shortest paths for boards with obstacles and that visualizes the results. If you did not write the A* implementation yourself, specify where you got the code from.
- Visualizations of the shortest path for the two tasks above.

5 Part 2 - Grids with different cell costs:

In part 1 we modelled Samfundet as if it was a building consisting of a single level i.e all rooms are on the same plane. In reality it is a multi-leveled building with stairs connecting the different rooms. Using stairs is usually more time consuming than walking on flat ground, especially if it is a popular night at

Samfundet. To reflect this reality an additional cost has been added to the stair cells.

In this part of the assignment, your code from Part 1 will be extended to take different cell costs into account for the path finding process. Table 1 specifies the cell types that should be supported, while Figure 8 shows an example of Samfundet where these cell types are used.

Table 1: Cell types and their associated costs.

	CHAR.	DESCRIPTION	COST
	.	Flat Ground	1
	,	Stairs	2
	:	Packed Stairs	3
	;	Packed Room	4

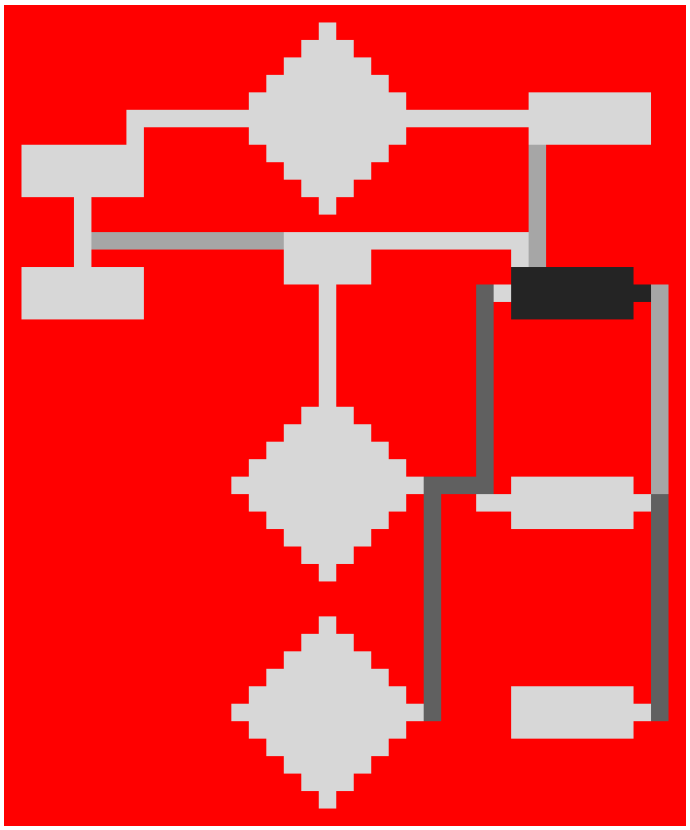

Figure 8: Samfundet modeled with all the different cell types in use.

5.1 Task 3:

Tonight you are going to a concert at Samfundet. The concert is held at Klubben and will start at 21. You arrived early to enjoy a Lyche-Burger with some friends before going to the concert. The time is 20:45 and you should get going. The stairs from Rundhallen to Edgar have unfortunately become packed with all the concert goers arriving. Use your A* implementation to find the path from Lyche to Klubben with the lowest cost using the map shown in Figure 9.

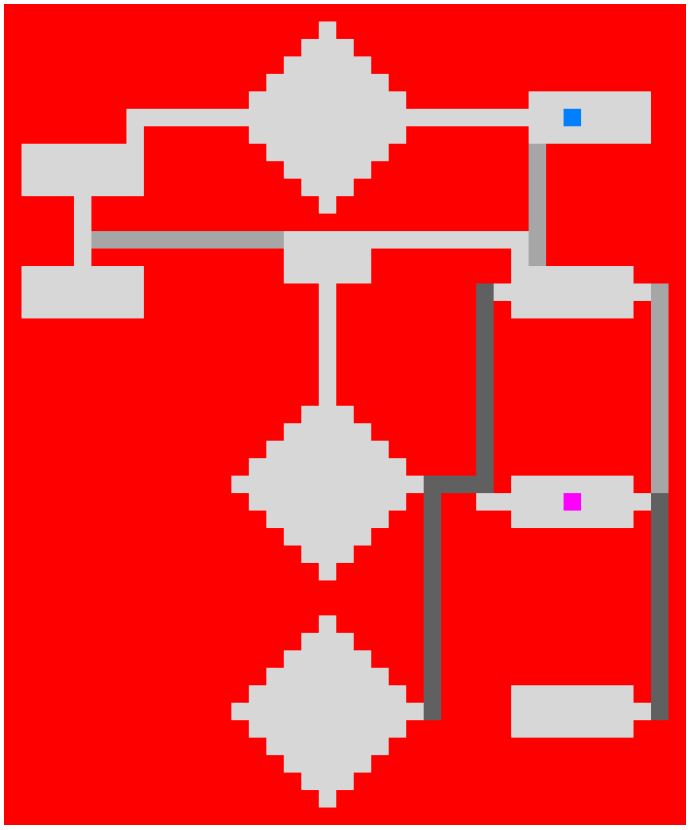

Figure 9: Task 3

5.2 Task 4:

You remember seeing a poster announcing a free chocolate cake party at Edgar this very evening. Edgar is therefore filled with hungry students scrambling to eat as much cake as possible. Use your A* implementation to find the new least-cost path from Lyche to Klubben, now considering the cake party at Edgar. The map is shown in Figure 10.

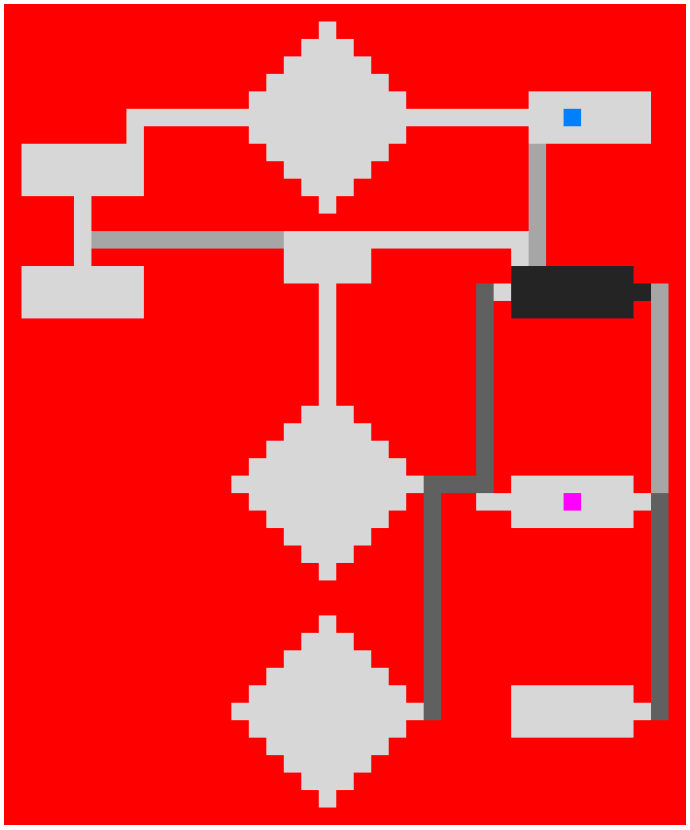

Figure 10: Task 4

Again, you can generate the two scenarios by instantiating a Map_Obj with the corresponding task number i.e map_obj = Map_Obj(task=3) for Task 3 above. Further information about the use of the map_obj is available in the Appendix.

Deliverables:

- Well-commented source code for a program that finds least-cost paths for boards with different cell costs (as specified in Table 1) and that visualizes the results.
- Visualizations of the least-cost paths for the two tasks above.

6 Part 3 - Moving Goal (Optional):

In the previous parts, we have only used static goals. In this final part we will use a moving goal. Your algorithm will therefore need to take this movement into account. We have limited the movement of our goal to a straight line, with a constant speed of $1/4$ your speed and a constant direction. You can therefore use this information to make some assumptions when designing your heuristic function $h()$.

6.1 Task 5:

After having finished the concert you feel the need for sugar! You rush down to Edgar to get some of that free chocolate cake. After having satisfied your sugar cravings you notice that you have separated

from your friend again. You head to the top of Rytterhallen and call to find that they are still at Klubben. Having had one beer too many they tell you that they will be heading towards Selskapssiden and can meet you outside. Having just consumed four chocolate cakes, you experience a sugar high and you quickly estimate that with your sugar high and your friend's reduced motor skills, you can move at four times their speed. Use this information along with the knowledge that your friend will be starting out at Klubben and moving in a straight line towards Selskapssiden to improve your A* implementation to handle a moving target.

A tick() function is provided with the code in Map_Obj. This function moves the goal one step towards Selskapssiden every fourth call. You should call this function every iteration of your A* algorithm for the goal to move properly.

Deliverables:

- Well-commented source code for a program that finds the least-cost path for the board with a moving goal.
- Visualization of the least-cost path.

7 Appendix:

This appendix contains information about the source code accompanying this assignment. The source code contains one class, Map_Obj, which provides functions for reading the .csv files containing the maps, placing the start and goal positions, moving goal positions as well as some code for printing the maps.

To start one of the above tasks instantiate a Map_Obj i.e map_obj = Map_Obj(task=1). You can then call map_obj.get_maps() to receive an integer and a string representation of the current map. The integer map contains the cost of each cell, while the string map can for instance be used for printing.

The map_obj.show_map() function can be called to show the current version of the string map in a human readable format.

The map_obj.print_map(map) function will print the provided map in the terminal (not as human readable).

The map_obj.tick() function will move the goal position every 4th call and returns the current goal position.

There are also a number of getters for:

- cell values given a coordinate
- goal position
- start position
- final goal position (used with moving goal)

You are free to modify the code as you like.