

Searching Algorithm Project

מגיש: עמית וולף – 307854364

Introduction:

My initial approach to this project was to first create a search problem solver to experiment and try different algorithms. While analyzing the results of the solver using A* and Heuristic search, I noticed the shortcomings of each algorithm and decided that in this project I will try and take the best features of each algorithm and create a combined version.

Solver :

Even though I could have used an implemented visualization tool for the analysis, the reasoning behind wanting to create the solver myself is that implementing the entire process from start to finish would allow me to have a better understanding of the code and so it would be easier to make changes and additions accordingly.

The final result was a simple application that creates a simple NxN graph (N is changable), the user can then click on nodes to set them as Start, End or a barrier to block the path. Then the user needs to click on the appropriate key in order to solve the graph for the shortest path with the required algorithm.

Basic Info:

Yellow – Start Node

Turquoise – End Node

Purple – Path found

Green – Node in OPEN

Red – Node in Closed

Travel cost from neighboring nodes is 1.

Heuristic Function used is Manhattan distance, for node n and end node e –

$$h(n) = |x_e - x_n| + |y_e - y_n|$$

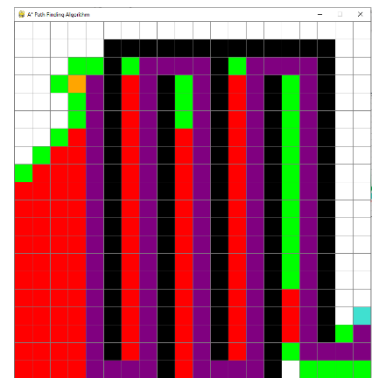
Basic Controls:

A* - Space

Heuristic Search – H

Reset Path – R

Clean Graph – C



The solver is included as an executable file named – Solver.exe, It requires python installed to run.

Searching Algorithm Project

מגיש: עמית וולף – 307854364

A*

A* is a staple searching algorithm in the field, published in a paper in 1968 it is still considered today as the best solution for many cases. Completeness, optimality and efficiency are the major properties which make A* a successful option for most cases.

A* is a best-first search algorithm which expands nodes based on a value equal to the distance from the start node plus the estimated distance to the end node. This value is called $f(n)$ and its equal to:

$$f(n) = g(n) + h(n)$$

$F(n)$ – the value used to compare node n

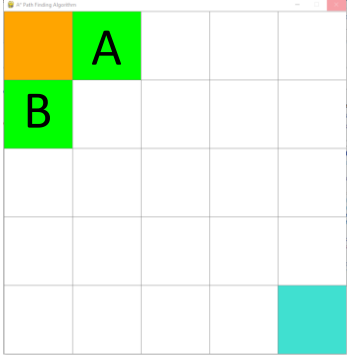
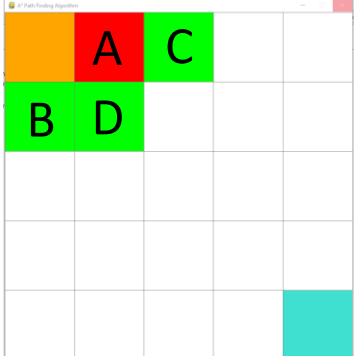
$G(n)$ - distance from start node to node n .

$H(n)$ – the estimated distance to the end node from node n , calculated using the heuristic function.

This best-first search methods expands the node from the OPEN list which has the lowest $f(n)$ value.

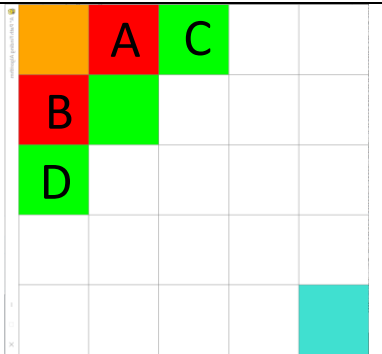
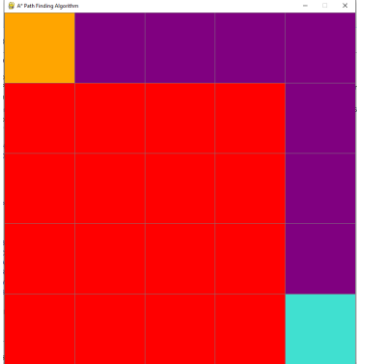
Because of the fact that A* uses the distance from the start node in its $f(n)$ value it means that even when there is a clear path to the end node, A* will still expand wide.

For example:

<p>5x5 Graph with the start node expanded, nodes A and B are in the open list –</p> $H(A) = 5-1 + 5-0 = 9$ $F(A) = 1 + H(A) = 10$ $H(B) = 5-0 + 5-1 = 9$ $F(B) = 1 + H(B) = 10$	
<p>A is expanded and added to the Closed List, C and D are added to the Open List.</p> $H(C) = 5-2 + 5-0 = 8$ $F(C) = 2 + H(C) = 10$ $H(D) = 5-1 + 5-1 = 8$ $F(D) = 2 + H(D) = 10$	

Searching Algorithm Project

מגיש: עמית וולף – 307854364

<p>Because all the $f(n)$ values in OPEN are equal to 10, the first node among them will be expanded, in this case – B.</p>	
<p>This process would continue until the entire matrix of $N \times N$ would be expanded as seen in the image to the right.</p>	

In a lot of cases this behavior can lead to searching the space more thoroughly which sometimes leads to better results, but still this is the behavior that I decided I want to try and optimize in a logical way that would allow the algorithm to use the wide search when necessary and a more heuristic approach when the problem allows it.

It's worth noting at this point that this behavior is the result of the simplicity of the search space and the heuristic function and can change depending on the travel cost and the heuristic difference between two neighboring nodes. That said, I wanted to try and create a more efficient algorithm that would know when to use the A* approach when needed, and a more straightforward approach when it's possible.

Heuristic Search

Using only the heuristic value to sort the OPEN list gives us much better results for problems where the path to the end node is straightforward.

Analyzing the Heuristic search results its clear to see the capabilities and efficiency of a straightforward approach when the search space allows it –

Searching Algorithm Project

מגיש: עמית וולף – 307854364

To the right we see the same problem used on with A* algorithm solved using Heuristic search. We can see that the nodes were expanded along the best path without expanding additional nodes. This is because every step had a lower $h(n)$ value then the one before which lead to it being the next best node:

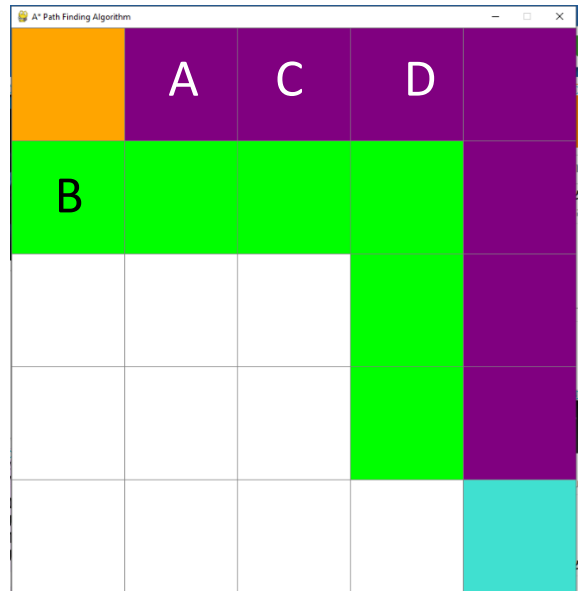
$$H(A) = |5-1| + |5-0| = 9$$

$$H(B) = |5-0| + |5-1| = 9$$

$$H(C) = |5-2| + |5-0| = 8$$

$$H(D) = |5-3| + |5-0| = 7$$

Node C has a lower $h(n)$ value than B so it would be expanded first, then D and so on.

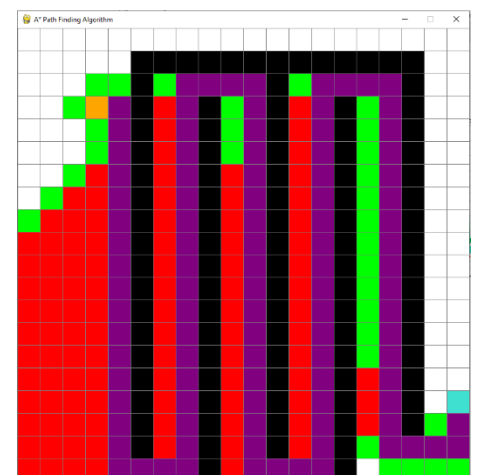
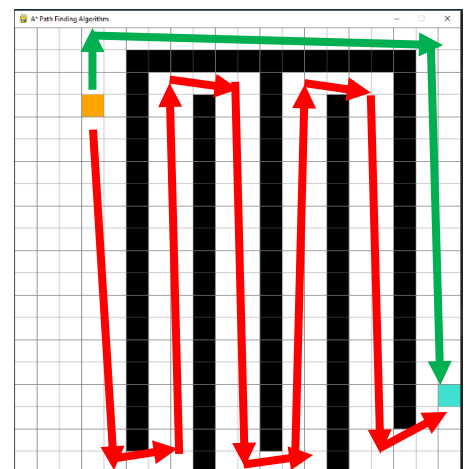


But of course, not all problems are as simple as this one. In some cases, this behavior can lead to very inefficient routes because of the constant divergence towards the end node.

For example:

In this problem we see that there are 2 possible paths from start to finish. The green path is the obvious shorter route, but it requires a few initial steps that **increase** the heuristic value (going up when the end node is below) to reach the shortcut.

The red route is the longer path, but it would suit the heuristic searches method of expanding towards the end node.

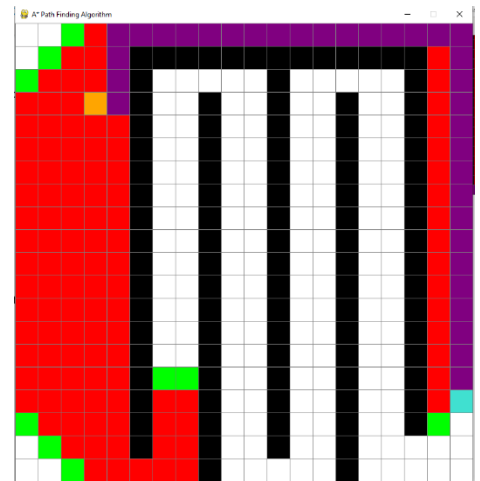


As expected the Heuristic search did follow the Red route :

Searching Algorithm Project

מגיש: עמית וולף – 307854364

A* does find the green route because it expands nodes based on the heuristic value and the distance from the start node, meaning that the node neighboring the start node are quickly expanded which reveals the shortcut.



Combining A* and Heuristic search

Naming these 2 behaviors:

Seek:

Using the A* method of sorting nodes in the Open list we can recreate the behavior shown in the example above to use a more A* oriented approach where the expansion of nodes is chosen relative to the sum of the heuristic function and the distance from the start node. This leads to the completeness of the result and the guarantee of finding the shortest path.

$$f(n) = g(n) + h(n)$$

Chase:

This Method is used by the Heuristic Search where the expansion happens towards the end node without regard to the $g(n)$ value. We will name this behavior *Chasing* because the method resembles a chase after the end node.

$$f(n) = h(n)$$

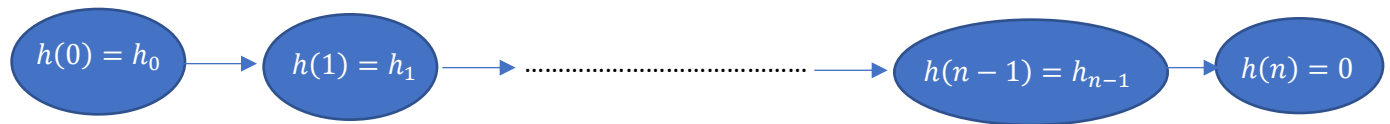
Shifting between the two behaviors:

The main challenge is to try and identify the optimal conditions for each behavior in a way that enables us to switch strategy when the conditions demand it. To force the algorithm to expand directly towards the end node we would need it to use the *Chasing* method when it is logical to do so, but when is it logical?

Searching Algorithm Project

מגיש: עמית וולף – 307854364

A *direct path* to the End node would have decreasing heuristic values for each node, like so:



$$\sum_{i=0}^{n-1} h_i > h_{i+1}$$

We will implement our algorithm so that when it finds a direct path it will *chase* along that path, and when there is no direct path, we will use the *seek* method to try and find one. In order to change between these two strategies, we will need to create two open lists so that each one sort nodes according to the wanted strategy.

The ChasingList will only hold nodes that were found to belong to a potential direct path, meaning that for each node n in ChasingList: $h(n) < h(\text{Parent of } n)$. The ChasingList is sorted using - $f(n) = h(n)$.

Any node that was found that does not meet that condition will be added to the SeekingList which is sorted using - $f(n) = g(n) + h(n)$.

Searching Algorithm Project

מגיש: עמית וולף – 307854364

Chase & Seek Algorithm

Pseudo-Code:

1	Create ChasingList
2	Create SeekingList
3	Create ClosedList
4	ChasingList.add(StartNode)
5	While !ChasingList.empty() or !SeekingList.empty():
6	If !ChasingList.empty():
7	Current = ChasingList.getBest()
8	Else:
9	Current = SeekingList.getBest()
10	If Current == Goal:
11	End
12	For Neighbor in Current.neighbors:
13	If $h(\text{Neighbor}) < h(\text{Current})$:
14	If Neighbor in SeekingList:
15	SeekingList.Remove(Neighbor)
16	ChasingList.Add(Neighbor)
17	Else:
18	If Neighbor not in ChasingList:
19	SeekingList.Add(Neighbor)
20	ClosedList.Add(Current)

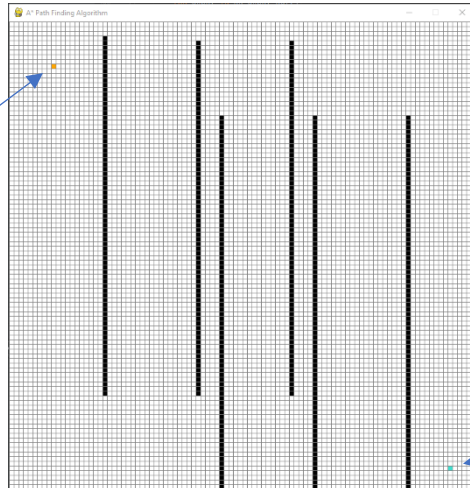
The Chase & Seek algorithm is designed to use the appropriate strategy according to the problem, lets run some experiments on different problems and see the results.

Searching Algorithm Project

מגיש: עמית וולף – 307854364

Problem 1: 100x100 Graph

Start



End

Algorithm	Result	Nodes Expanded	Runtime
A*		8396	0.125s
Heuristic Search		4042	0.14s
Chase & Seek		2442	0.515s

Searching Algorithm Project

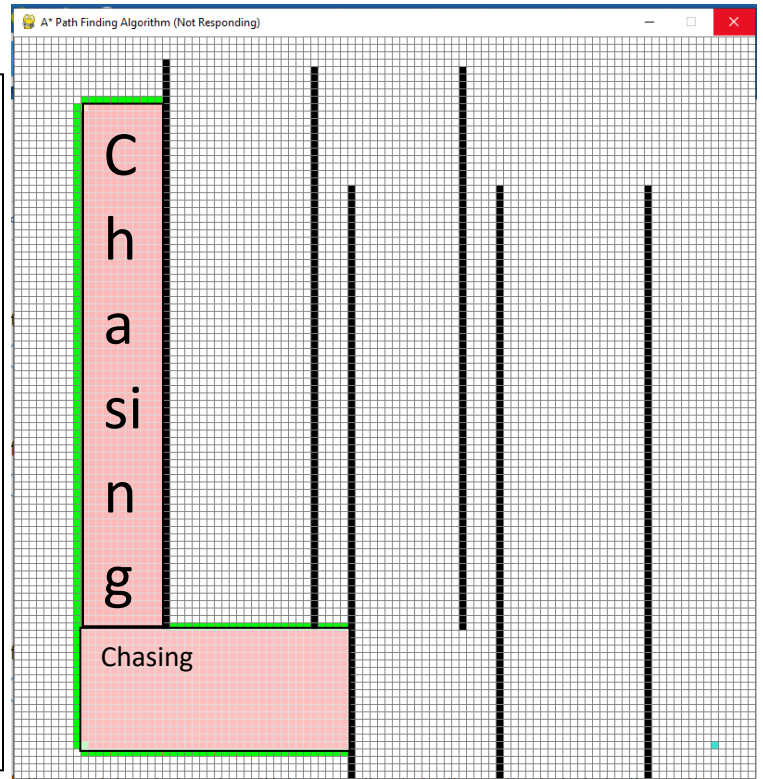
מגיש: עמית וולף – 307854364

Lets analyze the Result of the Chase & Seek algorithm:

Step 1:

Using the chasing method we expand nodes based on Heuristic value along a potential direct path.

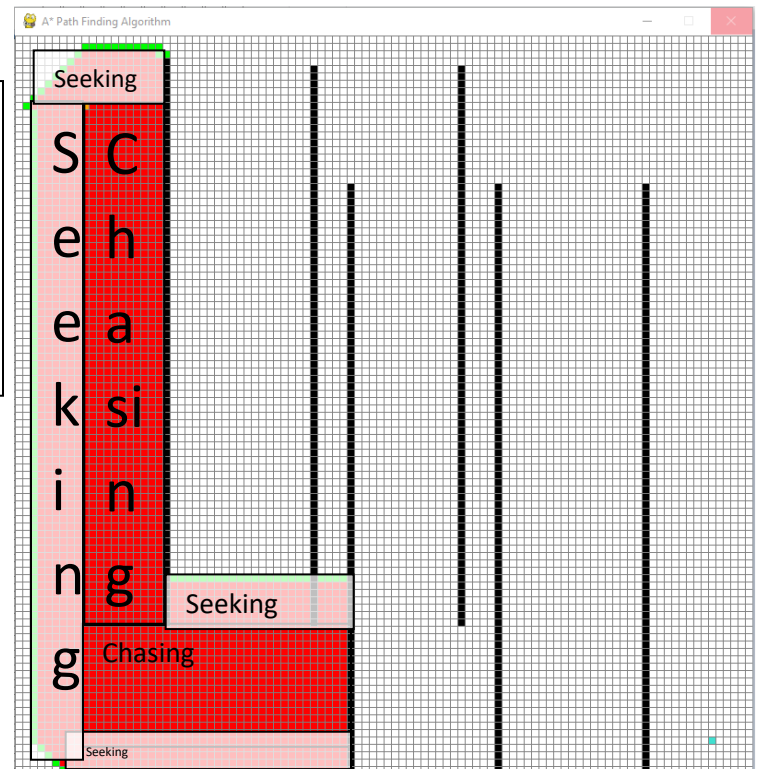
Once the path ends and there are no more nodes in the ChasingList we start using the seek method which expands node using the A* method.



Step 2:

Now that the ChasingList is empty we start expanding from the SeekingList which means we start to use the A* Method.

Once a new Direct path is found, we start Chasing.

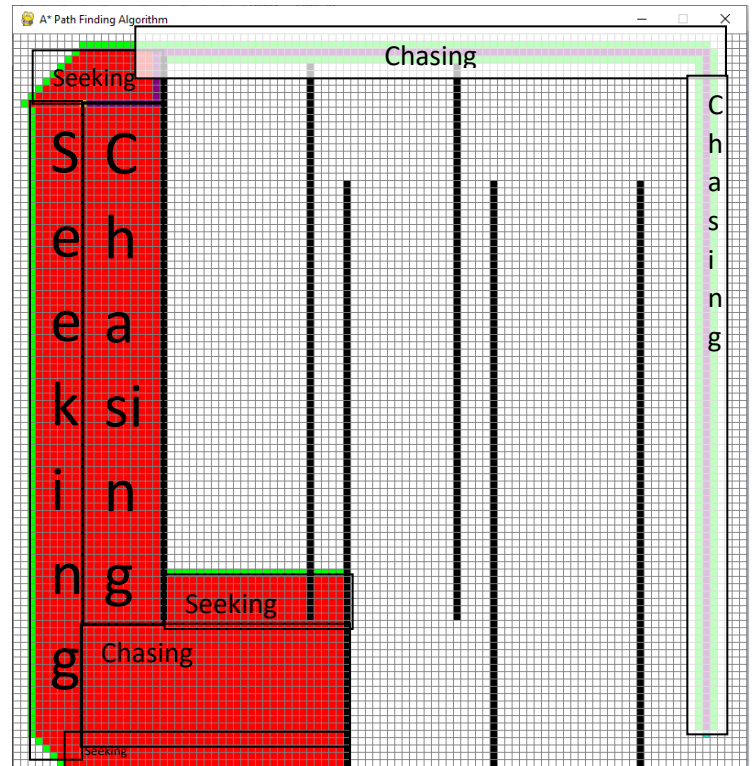


Searching Algorithm Project

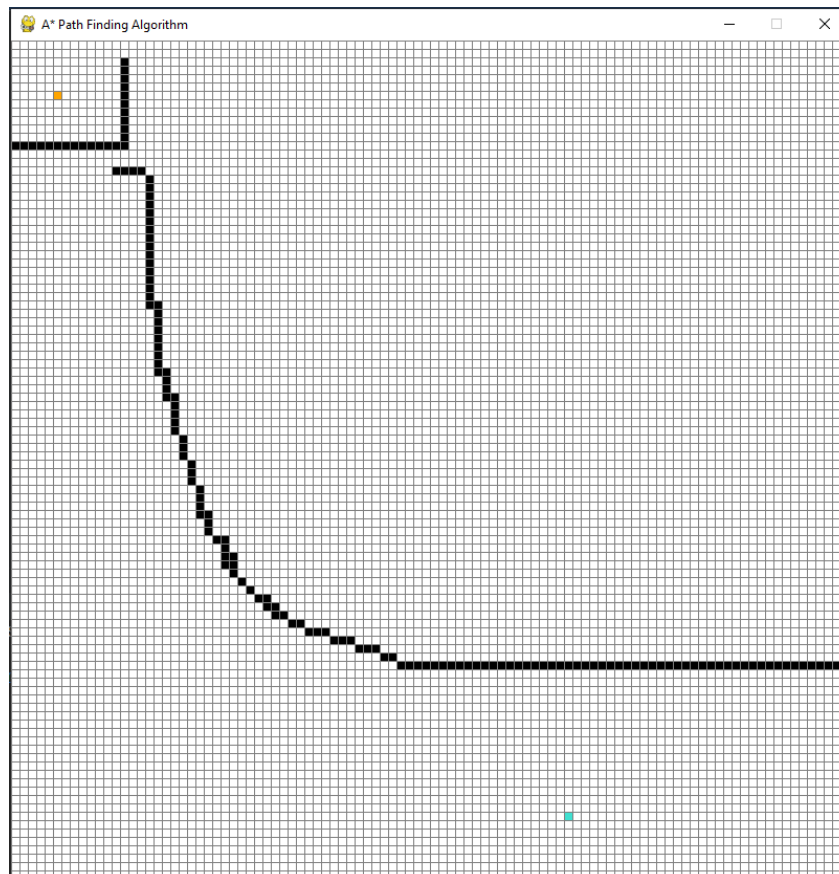
מגיש: עמית וולף – 307854364

Step 3:

The direct path is then expanded using the Chase method and finds the end node.



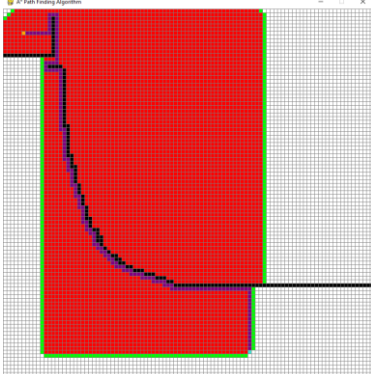
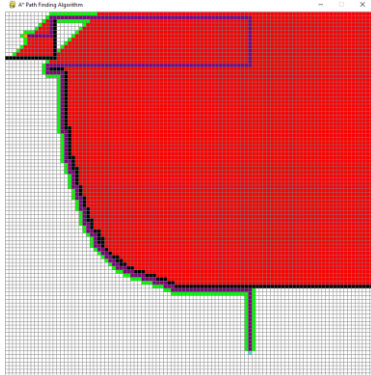
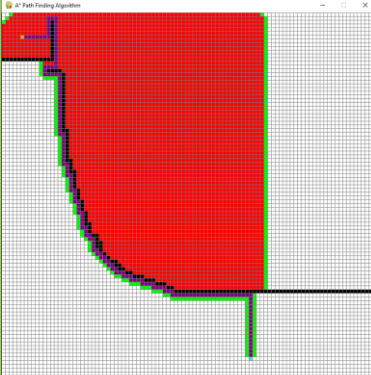
Problem 2 – 100x100 Graph



Searching Algorithm Project

מגיש: עמית וולף – 307854364

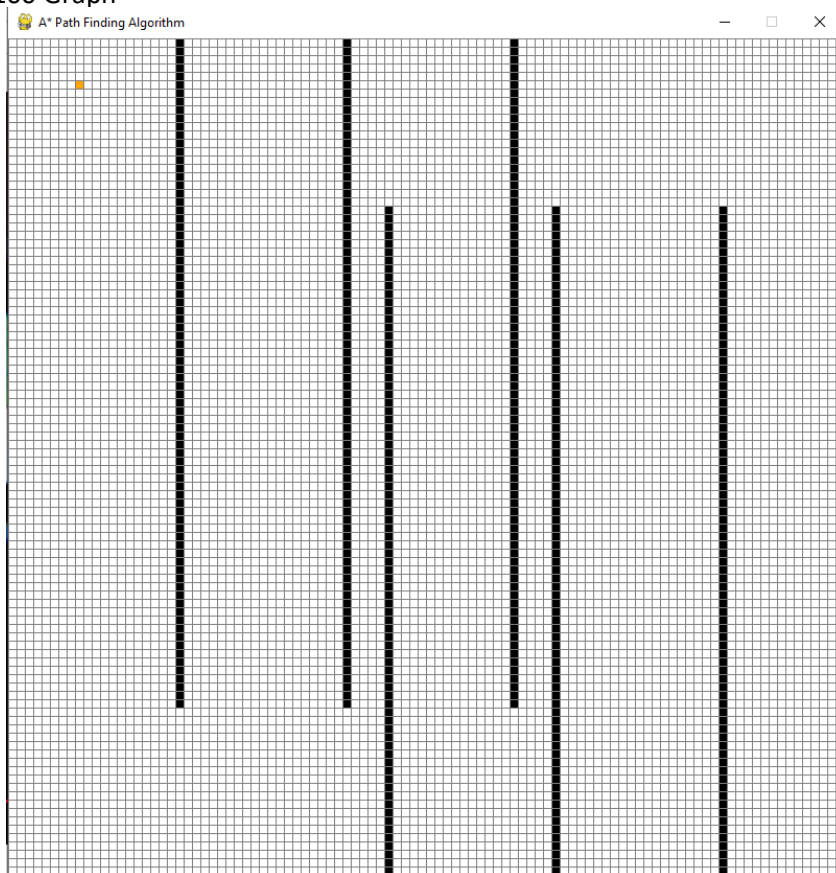
Problem 2 Results:

Algorithm	Result	Nodes Expanded	Runtime
A*		5441	0.14s
Heuristic Search		6005	0.14s
Chase & Seek		3928	1.5s

Searching Algorithm Project

מגיש: עמית וולף – 307854364

Problem 3 – 100x100 Graph

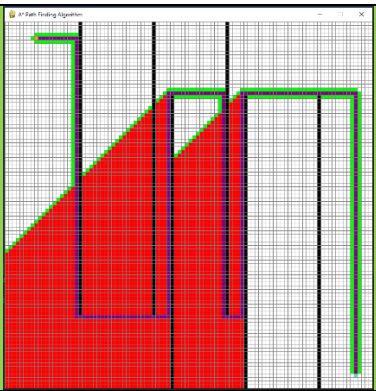
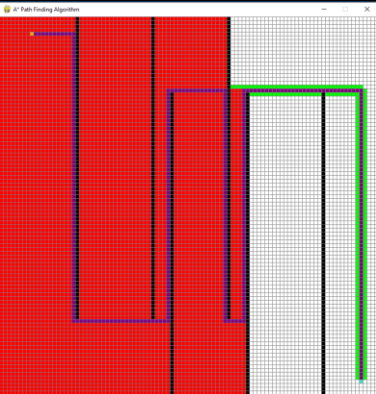


Problem 3 Results:

Algorithm	Result	Nodes Expanded	Runtime
A*	A 100x100 grid showing the result of the A* search. The explored area is shaded red, and the path from the start to the goal is highlighted with a green line.	8368	0.156s

Searching Algorithm Project

מגיש: עמית וולף – 307854364

Heuristic Search		4050	0.125s
Chase & Seek		6212	0.796s

In this case we see that the heuristic function delivers better results overall. But this is understandable given the problem's barriers and that expansion of nodes in the upper half of the search space yields no result so the convergence of the heuristic function actually is the fastest way to find the target node.

Searching Algorithm Project

מגיש: עמית וולף – 307854364

Conclusions:

Looking at the results we can see that changing strategies can lead to better results if the change is made in a logical manner. Problems 1 and 2 show that the change from chasing to seeking can save the expansion of unnecessary nodes when a potential direct path is available.

Even though the number of expansions are comparatively low, we must discuss some shortcomings of the Chase & Seek algorithm:

- Runtime – The process of handling 2 Open lists means that the potential for duplicates is higher than ever, that is why when adding a node into the ChasingList, we must first remove it from the SeekingList (if it exists there). This process of removing a certain node from the SeekingList PriorityQueue requires $O(n)$ runtime, which is the reason for the long runtimes compared to A* and Heuristic Search which have only one Open list.
- Complexity of the search space – In Problems that have weighted edges between nodes and inaccurate heuristic functions the Chase method can prove to be very inaccurate and lead to poor results, that means that the algorithm's favor towards using the chase method might prove to be problematic.

The simplicity of the search space and the accuracy of the heuristic function means that the heuristic approach will lead to the best result in a lot of cases (problem 3). In cases where the shortest path includes a sequence of nodes that **increase** in value, heuristic search would struggle to find the shortest path, and in those cases Chase & Seek would lead to the best results, as seen in problems 1 and 2.

It's safe to assume that in more complicated search spaces that use a less consistent and accurate heuristic functions, or a very large search space - the results might vary and bring to light the shortcomings of the Chase and Seek algorithm. But for this simple search space and heuristic function we can see the potential success of switching between strategies in a logical manner.