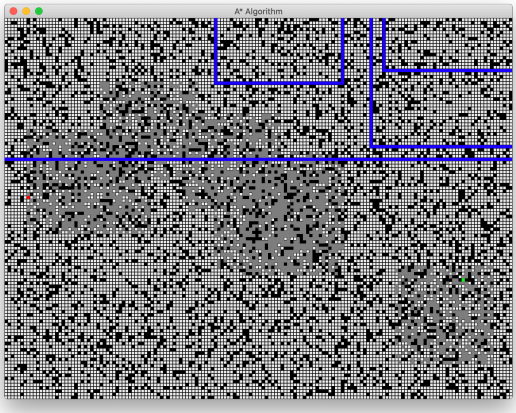
Report: Heuristic Search

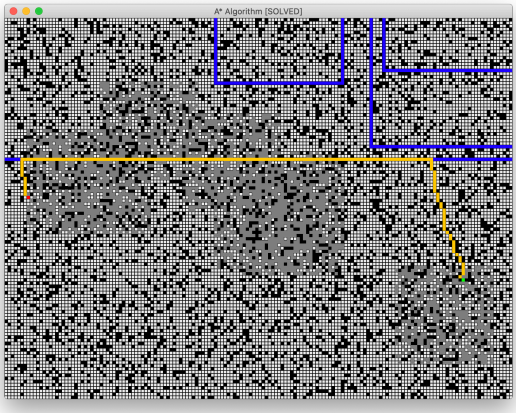
Sakib Rasul (191007914), Jacob Batista (179004590), Nick Ho (177001238), Jonathan Lu (188002803)

For this project, we created a semi-graphical interface capable of generating (and loading up previously written) eight-neighbor maps complete with a start cell, a goal cell, and a mixture of “regular”, “hard-to-traverse”, “highway”, and “blocked” cells. Users can visualize such a map in a GUI that distinguishes the start, goal, and terrain.

Users can call on variants of the A* search algorithm — uniform-cost search, admissible A*, weighted A*, and sequential A* — to compute a valid path from the start to the goal in such maps. After performing an A* search, a user may query any cell’s *h*, *g*, and *f* values.



A randomly-generated maze.



A uniform-cost solution.

To test our work, we generated 5 maps, and for each map we generated nine copies with different start-goal pairs for which a path could be computed, for a total of 50 test maps.

We optimized the implementation of our A* algorithms by using a nested `for` loop to iterate over and initialize a cell’s neighbors. This is more efficient than using an `ArrayList`, which would require tremendous amounts of space per analyzed cell.

We considered various heuristics for our search algorithms. The best admissible heuristic (*admissible* meaning barred from overestimating the cost of reaching the goal) we were able to come up with was a **Manhattan highway** heuristic that calculates the cost of traveling the Manhattan distance to the goal cell if every cell along the way happens to contain a highway. We chose this heuristic because highways are the least costly method of travel, and they never propagate diagonally.

Four inadmissible (but useful) heuristics we implemented are the following:

1. **Manhattan distance**: The Manhattan distance to the goal cell.
2. **Euclidean distance**: The Euclidean distance to the goal cell
3. **Easy Euclidean**: The cost of traversing the Euclidean distance to the goal cell given that every cell along the way is regular and unblocked

4. **Hard Euclidean**: The cost of traversing the Euclidean distance to the goal cell given that every cell along the way is hard-to-traverse

We chose these four heuristics because they rely largely on the distance to the goal cell (and so shouldn't overestimate it to an unreasonable extent) and because they aren't too difficult to compute.

We tried solving our test mazes with uniform-cost search, and 1-, 1.25-, and 2-weighted A* search with all five of the aforementioned heuristics, and found the following:

Heuristic	Runtime (ms)	% Optimality	# Expanded Nodes	Memory required (kb)
0 (Uniform-Cost Search)	34	100.00	5159.56	163.8984
Manhattan highway	28	120.05	5042.24	108.2755
1.25-Manhattan Highway	27	140.67	5112.86	54.3968
2-Manhattan Highway	22	218.06	5601.56	53.9005
Manhattan distance	3	434.88	1606.1	0
1.25-Manhattan distance	2	540.55	1120.56	0
2-Manhattan distance	2	694.24	804.96	0
Euclidean distance	7	342.02	2948.2	0
1.25-Euclidean distance	2	554.73	1159.5	0
2-Euclidean distance	2	639.06	777.32	0
Easy Euclidean	2	578.82	908.0	54.3968
1.25-Easy Euclidean	2	627.73	795.66	0
2-Easy Euclidean	2	637.52	765.86	53.9008
Hard Euclidean	2	637.52	765.86	0
1.25-Hard Euclidean	2	658.48	781.74	53.8979
2-Hard Euclidean	2	674.27	791.5	53.9011

Note: All results are averaged over our 50 test mazes.**% Optimality** is the average resulting path length as a function of the optimum length. 100% is optimal, and >100% is suboptimal.

Explain your results and discuss in detail your observations regarding the relative performance of the different methods. What impact do you perceive that different heuristic functions have on the behavior of the algorithms and why? What is the relative performance of the different algorithms and why? (10 points)

One of the biggest takeaways from our testing is that our choice of admissible heuristic ("Manhattan highway") is slightly sub-optimal, with an average path length about 20% longer than expected. It would be instructive to try and find a heuristic that brings us closer to perfect optimality.

Also a bit odd is the fact that some heuristics, like "Manhattan distance" and "Euclidean distance," seem to use up no memory. We reckon this is a resolution issue with procedures `Runtime.getRuntime().totalMemory()` and `freeMemory()`. Notice that every entry in the memory column is roughly a multiple of 54kb to see what I mean.

As expected, we find that uniform-cost search takes the longest amount of time (34ms) to complete. In contrast, nearly every inadmissible heuristic takes about 2ms to complete. This leads us to believe that 2ms is a lower bound for user programs on our computer to be able to complete A* searches (that or, akin to before, `System.currentTimeMillis()` can't record fewer than 2ms of runtime). That also suggests that, beyond about 500% optimality, it's hard to make gains in runtime efficiency.

Also as expected, weighting results in speedier completion times and less optimal paths. However, these results aren't always linked to a greater number of expanded nodes. "Manhattan highway" and "Hard Euclidean" both expand more nodes when weighted than when not, yet still enjoy non-increases in runtime like our other three heuristics do.

Of the heuristics we chose, it seems that "Manhattan distance" offers the best balance between quick runtimes and short path lengths. It's an easy heuristic to calculate and doesn't overestimate by all that much. Aiming for 100% optimality seems unreasonable at scale, but you don't have to sacrifice too much of it in order to significantly decrease solve times.

We continued our research by combining all five heuristics into a sequential A* search, anchored by our admissible heuristic, and coupled with w_1 and w_2 values of 1.25 or 2. We found the following:

	Runtime (ms)	% Optimality	# Expanded Nodes	Memory required (kb)
Manhattan Highway	28	120.05	5042.24	108.2755
(1.25, 1.25)-Sequential A*				
(1.25, 2)-Sequential A*				
(2, 1.25)-Sequential A*				

	Runtime (ms)	% Optimality	# Expanded Nodes	Memory required (kb)
(2, 2)-Sequential A*				

Note: All results are averaged over our 50 test mazes. **% Optimality** is the average resulting path length as a function of the optimum length. 100% is optimal, and >100% is suboptimal.

Describe in your report why your implementation is efficient.
Explain your results and discuss in detail your observations regarding the relative performance of the methods. Discuss the relationship with your experiments for section e. What is the relative performance of the different algorithm in terms of computation time and solution quality and why? (10 points)

Last, let's examine the upper bound of the "minimum anchor key" of a sequential A* search. By "minimum anchor key" we mean a state *s* whose key in the anchor of a sequential A* search is less than or equal to that of any and all of the other available successor states, i.e., a state s s.t. $\texttt{key(s, 0)} \leq \texttt{key(u, 0)}\ \forall u \in \texttt{OPEN}_0$. We can derive from the properties of weighted-A* search that the *g*-value of such a state *s* is upper-bounded by the product of w_1 and the optimal cost to reach state *s*, i.e., $g_0(s) \leq w_1 c(s)$, where *c(s)* is the least costly path to state *s*.

We know that given a solvable maze, it must hold that a state s_i which lies upon the least costly path to the goal always exists in $\texttt{OPEN}_0$.

Since we know that the cost of a weighted-A* solution is upper bounded by the product of *w* and the cost of the optimal solution, we can say that g

To end our discussion, we show that the anchor key of any state *s*, when it is the minimum anchor key in an iteration of sequential A* search, is also bounded by *w*_1 times the cost of the optimal path to the goal cell. In other words, we would like to show that for any state *s* where Key(*s*, 0) <= Key(*u*, 0) for all *u* in OPEN_0, it holds that Key(*s*, 0) <= *w*_1 * g(*s*_goal).

We begin by considering an OPEN_0 queue in a regular A* search. There will always exist in the queue a state *s_i* that is located along a least-cost path from *s*_start to *s*_goal. This is NOT always true when we consider an OPEN_0 queue in a weighted-A* search. This implies that our claim is true.

Furthermore, when the sequential A∗ terminates in the i-th search, it holds that g_i(s_goal) ≤ w_1 * w_2 * c(s_goal). In other words, the solution cost obtained by the algorithm is bounded by a w_1 * w_2 sub-optimality factor.