



A* Search Algorithm

Isaac Jennings

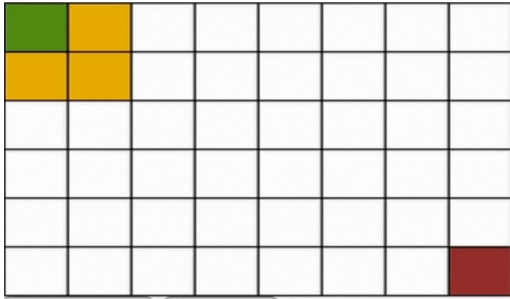


What is A*

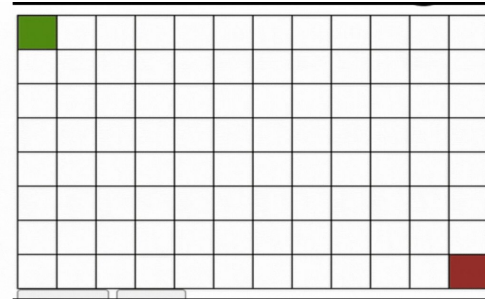
- Graph search algorithm for computer data structures
- Originally designed for map traversal via the shortest possible path
- It optimizes the shortest paths first
- Optimal and Comprehensive search method
- Uses weighted traversal to establish the shortest path
- Space and time will be very important factors when it comes to implementing A*, so it is important to keep them in mind.

How A* works

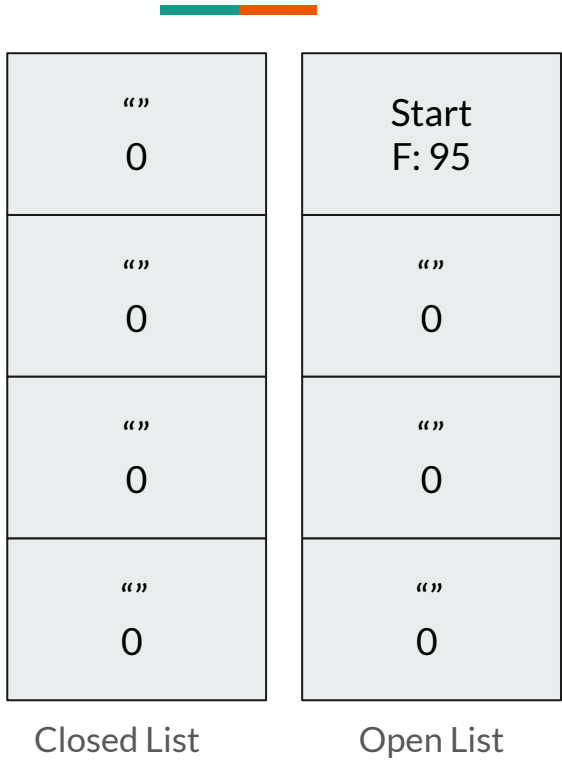
- A* Uses a grid structure and a breadth-first search to kick off the pathfinding algorithm
 - Upon the first iteration of the breadth first (the grid squares surrounding the start, A* begins to compute)
 - g: distance/cost to get to the cell
 - h: the heuristic (estimation of absolute path)
 - f: $g + h$




Dijkstra's Search



A* Search





“” 0
“” 0
“” 0
“” 0

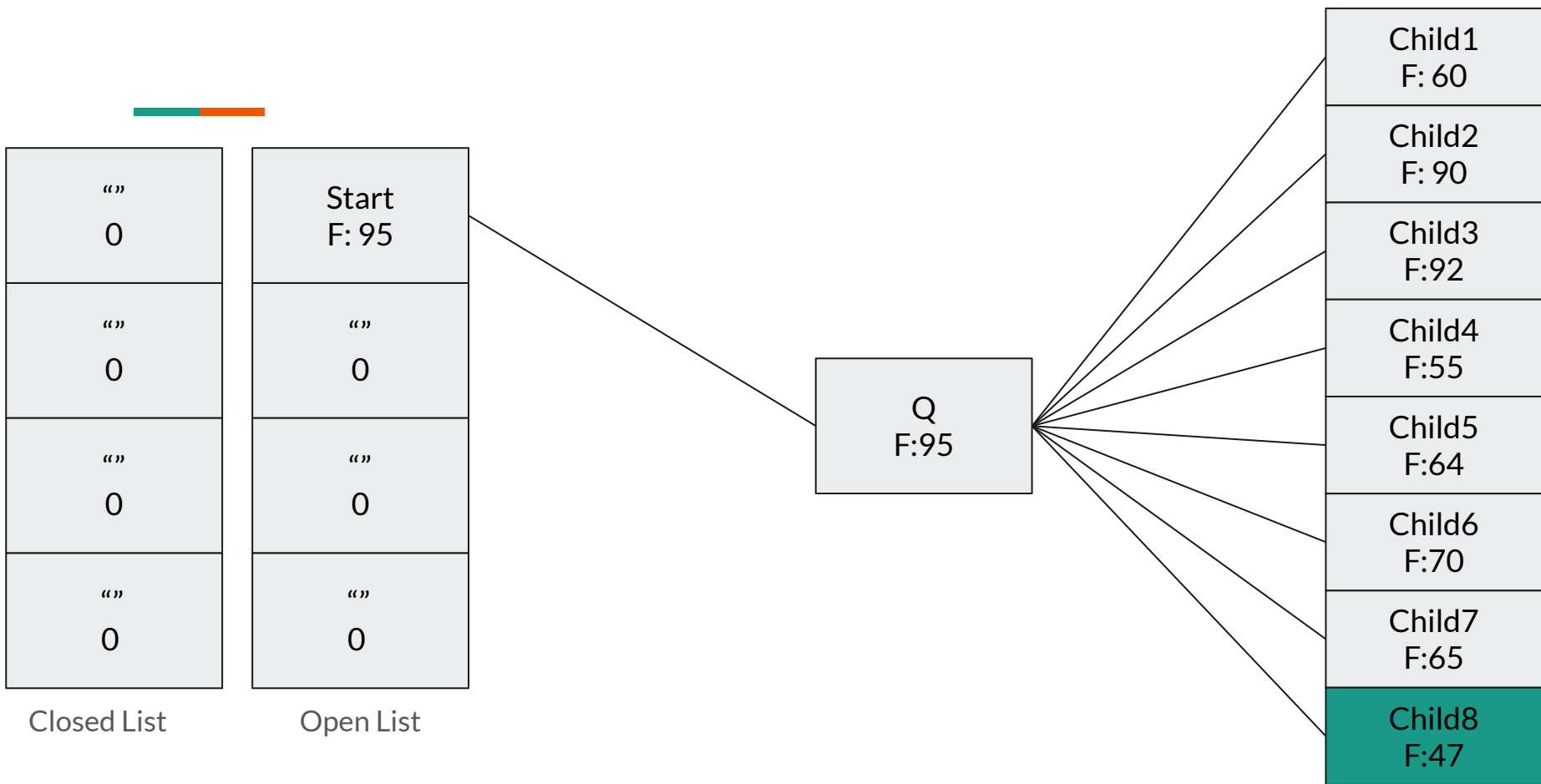
Closed List

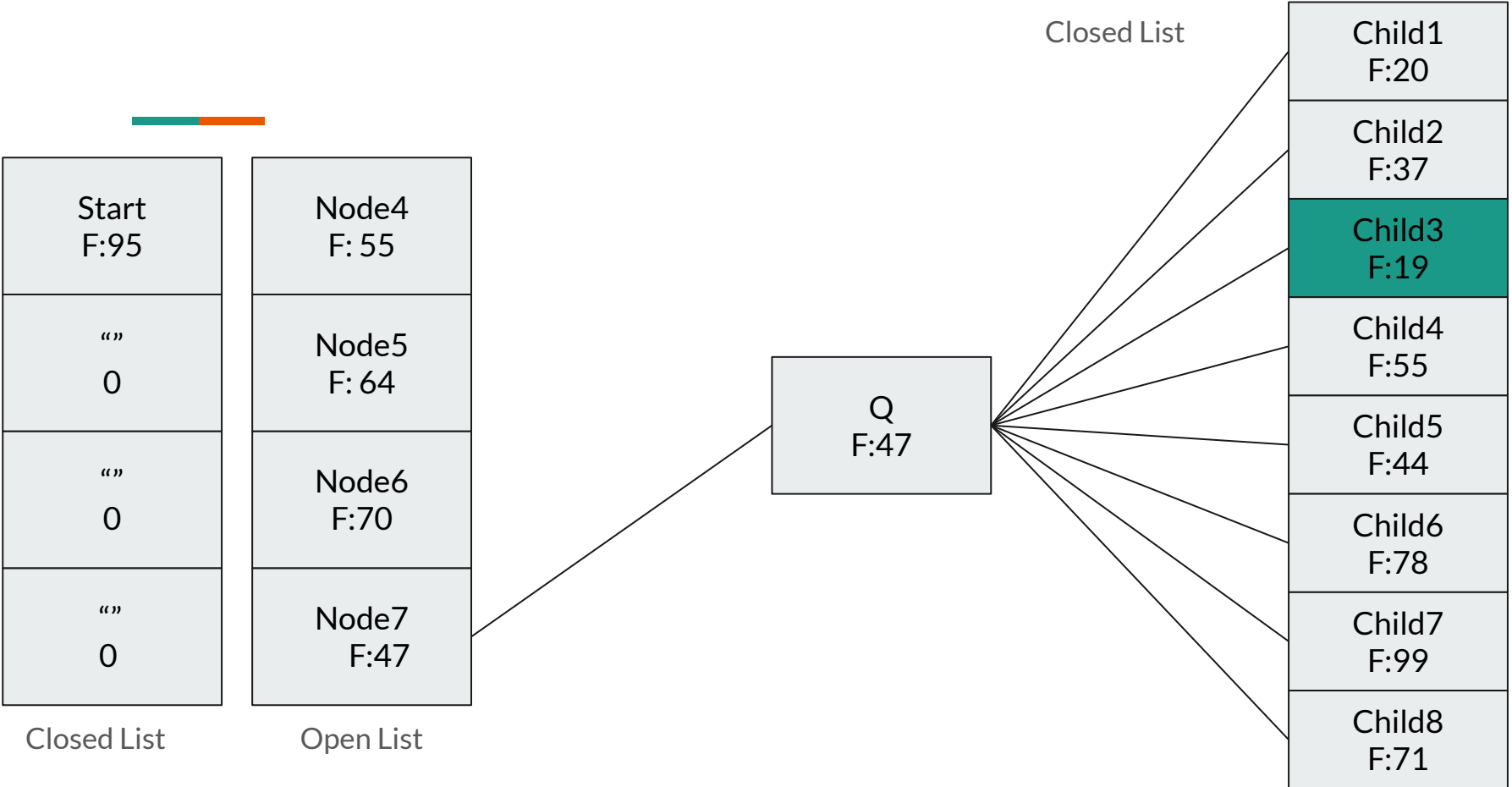
Start F: 95
“” 0
“” 0
“” 0

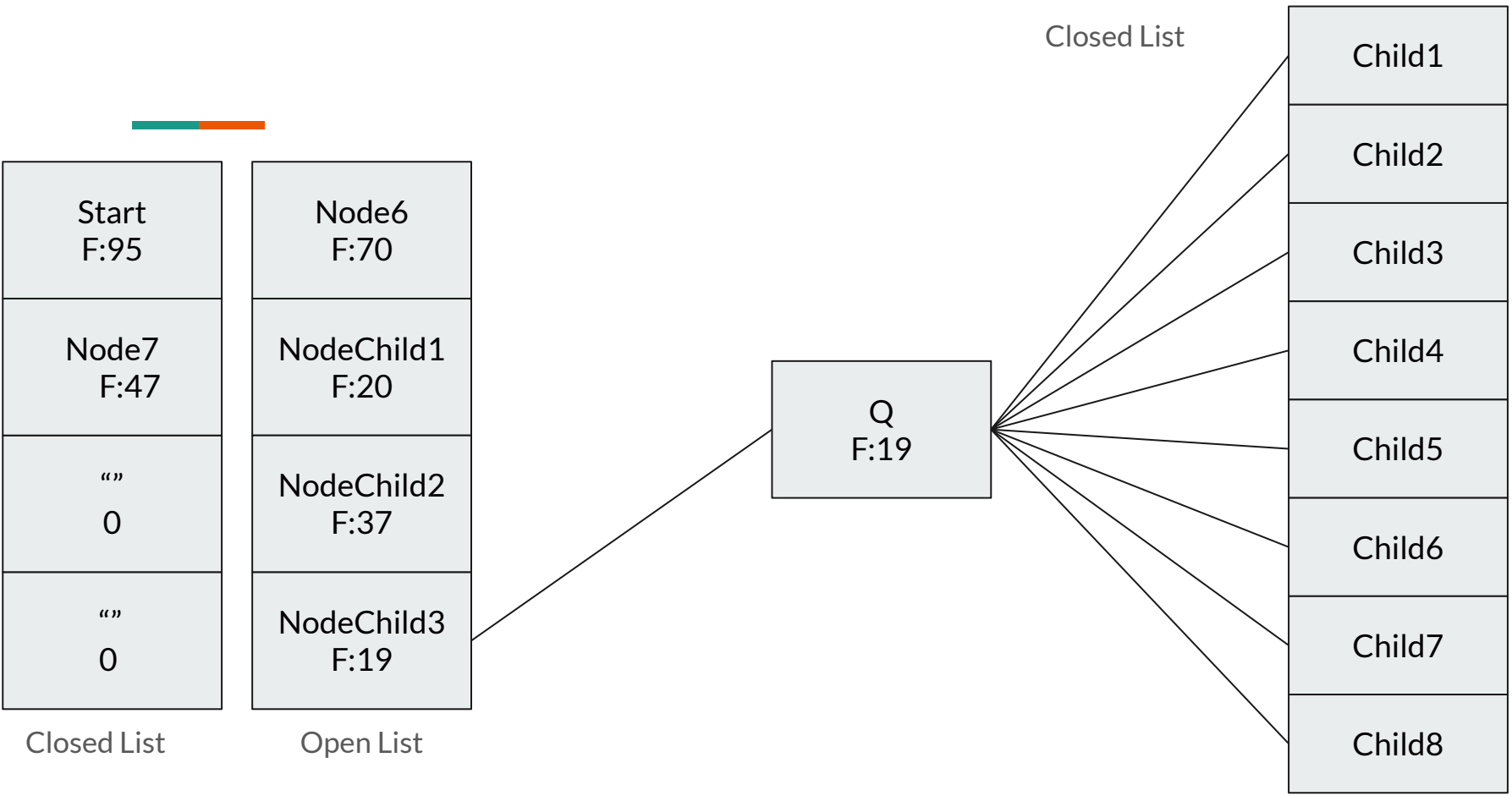
Open List

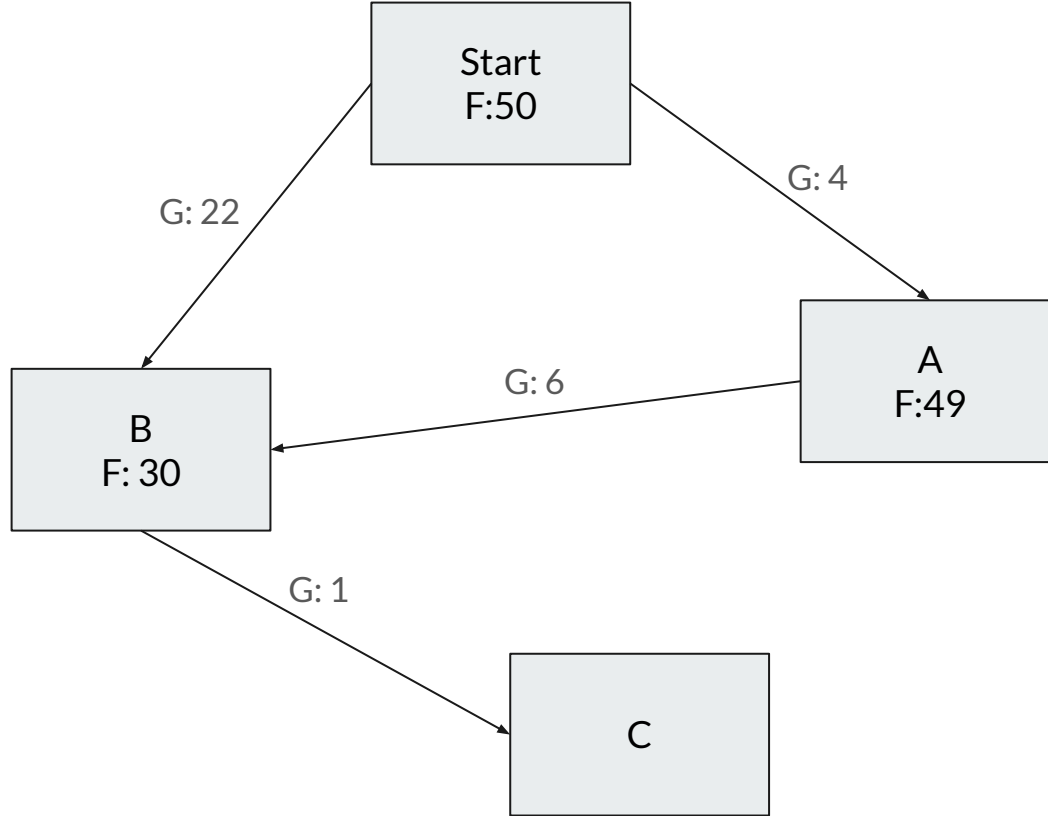
Q
F:95

Child1 F: 60
Child2 F: 90
Child3 F:92
Child4 F:55
Child5 F:64
Child6 F:70
Child7 F:65
Child8 F:47





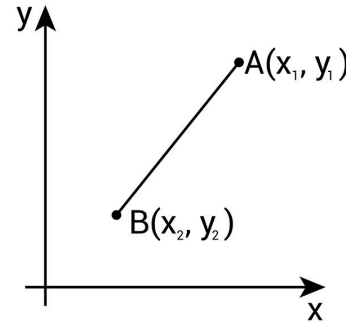




Exact Heuristics

- The direct weights (G) are very easy to compute, but what about heuristics
- 2 methods:
 - Calculate the exact value
 - Approximate the absolute path
- In terms of the exact heuristics, the distance between every single pair needs to be calculated so that the distance formula can be used from every node to the final goal.
- Approximations require significantly less space
- 3 Methods to Approximate

Distance Formula



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



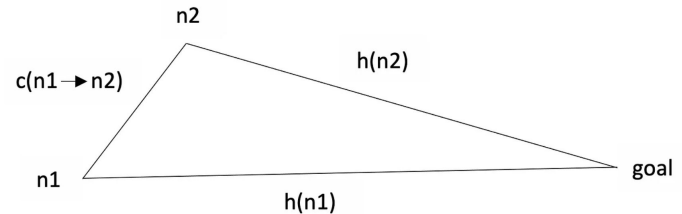
Approximate Heuristics

- **Manhattan distance** is calculated based on movement in the xy plane
 - $H = \text{abs}(\text{current_node.x} - \text{goal.x}) + \text{abs}(\text{current_node.y} - \text{goal.y})$
- **Diagonal distance** is the greatest difference between x and y from current node to the goal
 - $dx = \text{abs}(\text{current_node.x} - \text{goal.x}), dy = \text{abs}(\text{current_node.y} - \text{goal.y})$
 - $H = D * (dx + dy) + (D2 - 2 * D) * \min(dx, dy)$; D is the length of each node and D2 is the length of the diagonal
 - Allows movement in eight directions (compass movement)
- **Euclidean distance** uses the aforementioned distance formula to approximate the distance from the current node to the end goal.
 - No movement constrictions

Making Good Approximations

- Heuristics should be admissible
 - Value of the given heuristic should never exceed the actual value
 - $h(n) \leq h^*(n)$; $h^*(n)$ is the actual value of $f(n)$
- Heuristics should be Monotone
 - $h(n_1) \leq c(n_1 \rightarrow n_2) + h(n_2)$
 - If the two smaller sides are not always greater than the direct path, the path follower may visit the same node twice, thinking it is another step in the path

Monotonicity





Real World Implementations

- A* was originally designed for robot mobility
 - Currently, it is used in many SLAM algorithms (Simultaneous localization and mapping)
 - A great example of this is the Navigation2 architecture in ROS2 (robot operating system)
 - Slam_toolbox utilizes A* for differential drive robots, and the package has a modified version for legged and ackerman steer robots.
 - What was mostly explained here would work for many robotic applications, but A* has many other modifications that make it useful in other environments

Real World Implementations

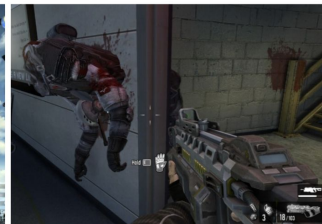
- A^* is also heavily used in Non Playable Character (NPC) and Playable Character (PC) pathfinding in video games
 - Often, static A^* is not enough in these environments, and a localization planner would take up too much processing power, so the navigation planner must get more sophisticated
 - Weighted A^* (Biased towards destination)
 - ARA^* (finds an initial solution quickly and begins optimizing with remaining time)
 - D^* (Dynamic A^* : allows characters to maneuver in a constantly changing environment)
 - Multi Heuristic A^* (Uses an admissible heuristic and inadmissible heuristic(s) to search the environment)



(a) The Elder Scrolls V: Skyrim



(b) The Witcher 3: Wild Hunt



(c) F.E.A.R. 3

Figure 1. Inappropriate behavior of non-playable characters (NPCs) in various games.



Sources

V. Hwang, M. Likhachev, S. Swaminathan, and V. Narayanan, Multi-heuristic A* - CMU school of computer science, https://www.cs.cmu.edu/~maxim/files/mha_ijrr15.pdf (accessed Mar. 21, 2024).

R. A. S, "A* algorithm in artificial intelligence you must know in 2024: Simplilearn," Simplilearn.com, [https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm#:~:text=A*%20Search%20Algorithm%20is%20a,path%20algorithm%20\(Dijkstra's%20Algorithm\)](https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm#:~:text=A*%20Search%20Algorithm%20is%20a,path%20algorithm%20(Dijkstra's%20Algorithm).). (accessed Mar. 21, 2024).

A. Stoll, "A* pathfinding animation," A Star Pathfinding Algorithm Animation, <https://adrianstoll.com/post/a-star-pathfinding-algorithm-animation/> (accessed Mar. 21, 2024).

Kim J-H, Lee J, Kim S-J. Navigating Non-Playable Characters Based on User Trajectories with Accumulation Map and Path Similarity. *Symmetry*. 2020; 12(10):1592. <https://doi.org/10.3390/sym12101592>