

DEPARTMENT OF MECHATRONICS ENGINEERING

Course Name:

Course No: CSE

Experiment Name: Software development.

SUBMITTED BY:

1. Azim Miya – 1808028
2. Bishowjit Paul -1808029

SUBMITTED TO:

SUBMITTED DATE:

Software Development: Software development is the process of conceiving, specifying, designing, [programming](https://en.wikipedia.org/wiki/Computer_programmer), [documenting](https://en.wikipedia.org/wiki/Software_documentation), [testing](https://en.wikipedia.org/wiki/Software_testing), and [bug fixing](https://en.wikipedia.org/wiki/Software_bugs) involved in creating and maintaining [applications](https://en.wikipedia.org/wiki/Application_software), [frameworks](https://en.wikipedia.org/wiki/Software_framework), or other software components. Software development is a process of writing and [maintaining](https://en.wikipedia.org/wiki/Software_maintenance) the [source code](https://en.wikipedia.org/wiki/Source_code), but in a broader sense, it includes all that is involved between the conception of the desired software through to the final manifestation of the software, sometimes in a planned and [structured](https://en.wikipedia.org/wiki/Software_development_process) process. Therefore, software development may include research, new development, prototyping, modification, reuse, re-engineering, maintenance, or any other activities that result in software products.

Software Development Activities:

1. Identification of need
2. Planning
3. Designing
4. Implementation, testing and documenting
5. Deployment and maintenance.

Software Development project:

1. Audio Palyer: [Audio player (software)](https://en.wikipedia.org/wiki/Audio_player_(software)), a piece of computer software for playing audio files.
2. Student Management System:

Path Finding visualizer:Pathfinding or pathing is the plotting, by a computer application, of the shortest route between two points. It is a more practical variant on [solving mazes](https://en.wikipedia.org/wiki/Maze#Solving_mazes). This field of research is based heavily on [Dijkstra's algorithm](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm" \o "Dijkstra's algorithm) for finding the shortest path on a [weighted graph](https://en.wikipedia.org/wiki/Glossary_of_graph_theory#Weighted_graphs_and_networks).

Pathfinding is closely related to the [shortest path problem](https://en.wikipedia.org/wiki/Shortest_path_problem), within [graph theory](https://en.wikipedia.org/wiki/Graph_theory), which examines how to identify the path that best meets some criteria (shortest, cheapest, fastest, etc) between two points in a large network.

## **Algorithms used in pathfinding[[edit](https://en.wikipedia.org/w/index.php?title=Pathfinding&action=edit&section=8" \o "Edit section: Algorithms used in pathfinding)]**

* [A\* search algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)
* [Dijkstra's algorithm](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm), a special case of the A\* search algorithm
* [D\*](https://en.wikipedia.org/wiki/D*) a family of [incremental heuristic search](https://en.wikipedia.org/wiki/Incremental_heuristic_search) algorithms for problems in which constraints vary over time or are not completely known when the [agent](https://en.wikipedia.org/wiki/Intelligent_agent) first plans its path
* [Any-angle path planning](https://en.wikipedia.org/wiki/Any-angle_path_planning) algorithms, a family of algorithms for planning paths that are not restricted to move along the edges in the search graph, designed to be able to take on any angle and thus find shorter and straighter paths

**Dijkstra's algorithm**[[edit](https://en.wikipedia.org/w/index.php?title=Pathfinding&action=edit&section=2" \o "Edit section: Dijkstra's algorithm)]

A common example of a graph-based pathfinding algorithm is [Dijkstra's algorithm](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm" \o "Dijkstra's algorithm). This algorithm begins with a start node and an "open set" of candidate nodes. At each step, the node in the open set with the lowest distance from the start is examined. The node is marked "closed", and all nodes adjacent to it are added to the open set if they have not already been examined. This process repeats until a path to the destination has been found. Since the lowest distance nodes are examined first, the first time the destination is found, the path to it will be the shortest path.[[3]](https://en.wikipedia.org/wiki/Pathfinding#cite_note-3)

Dijkstra's algorithm fails if there is a negative [edge](https://en.wikipedia.org/wiki/Edge_(graph_theory)) weight. In the hypothetical situation where Nodes A, B, and C form a connected undirected graph with edges AB = 3, AC = 4, and BC = −2, the optimal path from A to C costs 1, and the optimal path from A to B costs 2. Dijkstra's Algorithm starting from A will first examine B, as that is the closest. It will assign a cost of 3 to it, and mark it closed, meaning that its cost will never be reevaluated. Therefore, Dijkstra's cannot evaluate negative edge weights. However, since for many practical purposes there will never be a negative edgeweight, Dijkstra's algorithm is largely suitable for the purpose of pathfinding.

**A\* algorithm**[[edit](https://en.wikipedia.org/w/index.php?title=Pathfinding&action=edit&section=3" \o "Edit section: A* algorithm)]

[A\*](https://en.wikipedia.org/wiki/A*_search_algorithm) is a variant of Dijkstra's algorithm commonly used in games. A\* assigns a weight to each open node equal to the weight of the edge to that node plus the approximate distance between that node and the finish. This approximate distance is found by the [heuristic](https://en.wikipedia.org/wiki/Heuristic_(computer_science)), and represents a minimum possible distance between that node and the end. This allows it to eliminate longer paths once an initial path is found. If there is a path of length x between the start and finish, and the minimum distance between a node and the finish is greater than x, that node need not be examined.[[4]](https://en.wikipedia.org/wiki/Pathfinding#cite_note-4)

A\* uses this heuristic to improve on the behavior relative to Dijkstra's algorithm. When the heuristic evaluates to zero, A\* is equivalent to Dijkstra's algorithm. As the heuristic estimate increases and gets closer to the true distance, A\* continues to find optimal paths, but runs faster (by virtue of examining fewer nodes). When the value of the heuristic is exactly the true distance, A\* examines the fewest nodes. (However, it is generally impractical to write a heuristic function that always computes the true distance.) As the value of the heuristic increases, A\* examines fewer nodes but no longer guarantees an optimal path. In many applications (such as video games) this is acceptable and even desirable, in order to keep the algorithm running quickly.

**Sample algorithm**[[edit](https://en.wikipedia.org/w/index.php?title=Pathfinding&action=edit&section=4" \o "Edit section: Sample algorithm)]

This is a fairly simple and easy-to-understand pathfinding algorithm for tile-based maps. To start off, you have a map, a start coordinate and a destination coordinate. The map will look like this, X being walls, S being the start, O being the finish and \_ being open spaces, the numbers along the top and right edges are the column and row numbers:

1 2 3 4 5 6 7 8

X X X X X X X X X X

X \_ \_ \_ X X \_ X \_ X 1

X \_ X \_ \_ X \_ \_ \_ X 2

X S X X \_ \_ \_ X \_ X 3

X \_ X \_ \_ X \_ \_ \_ X 4

X \_ \_ \_ X X \_ X \_ X 5

X \_ X \_ \_ X \_ X \_ X 6

X \_ X X \_ \_ \_ X \_ X 7

X \_ \_ O \_ X \_ \_ \_ X 8

X X X X X X X X X X

First, create a list of coordinates, which we will use as a queue. The queue will be initialized with one coordinate, the end coordinate. Each coordinate will also have a counter variable attached (the purpose of this will soon become evident). Thus, the queue starts off as ((3,8,0)).

Then, go through every element in the queue, including elements added to the end over the course of the algorithm, and to each element, do the following:

1. Create a list of the four adjacent cells, with a counter variable of the current element's counter variable + 1 (in our example, the four cells are ((2,8,1),(3,7,1),(4,8,1),(3,9,1)))
2. Check all cells in each list for the following two conditions:
   1. If the cell is a wall, remove it from the list
   2. If there is an element in the main list with the same coordinate and a less than or equal counter, remove it from the cells list
3. Add all remaining cells in the list to the end of the main list
4. Go to the next item in the list

Thus, after turn 1, the list of elements is this: ((3,8,0),(2,8,1),(4,8,1))

* After 2 turns: ((3,8,0),(2,8,1),(4,8,1),(1,8,2),(4,7,2))
* After 3 turns: (...(1,7,3),(4,6,3),(5,7,3))
* After 4 turns: (...(1,6,4),(3,6,4),(6,7,4))
* After 5 turns: (...(1,5,5),(3,5,5),(6,6,5),(6,8,5))
* After 6 turns: (...(1,4,6),(2,5,6),(3,4,6),(6,5,6),(7,8,6))
* After 7 turns: (...(1,3,7)) – problem solved, end this stage of the algorithm – note that if you have multiple units chasing the same target (as in many games – the finish to start approach of the algorithm is intended to make this easier), you can continue until the entire map is taken up, all units are reached or a set counter limit is reached

Now, map the counters onto the map, getting this:

1 2 3 4 5 6 7 8

X X X X X X X X X X

X \_ \_ \_ X X \_ X \_ X 1

X \_ X \_ \_ X \_ \_ \_ X 2

X S X X \_ \_ \_ X \_ X 3

X 6 X 6 \_ X \_ \_ \_ X 4

X 5 6 5 X X 6 X \_ X 5

X 4 X 4 3 X 5 X \_ X 6

X 3 X X 2 3 4 X \_ X 7

X 2 1 0 1 X 5 6 \_ X 8

X X X X X X X X X X

Now, start at S (7) and go to the nearby cell with the lowest number (unchecked cells cannot be moved to). The path traced is (1,3,7) -> (1,4,6) -> (1,5,5) -> (1,6,4) -> (1,7,3) -> (1,8,2) -> (2,8,1) -> (3,8,0). In the event that two numbers are equally low (for example, if S was at (2,5)), pick a random direction – the lengths are the same. The algorithm is now complete.