Maze Solver Program

Mazes can be solved using a variety of algorithms. Here I wrote a program which solves the maze using A\* algorithm

A\* algorithm

* Similar to Dijkstra but A\* considers priority nodes and Dijkstra explore all possible paths
* A\* is an extension to Dijkstra but with priority nodes so it is more optimal
* A\* follows a formula f(x) = g(x) + h(x), where f(x) is the next step the program decides to take, g(x) is how far is x from the start and h(x) is how far the node is to the end. The program does this with all the surrounding nodes around it, puts all the surrounding nodes in a queue, with the smallest f(x) placed in priority first. Finally the program chooses the smallest f(x) as the next step, and repeats
* If we took a step then we pop it out of our stack
* Heuristic: a technique designed for solving a problem more quickly when classic methods are too slow. H(x) is our heuristic, because its an additional step from Dijkstra

A\* algorithm, which also uses Artificial Intelligence to find the end of the maze(How?)

When the maze is found it indicates how many steps it took (Which is the g(x) value of the last node), how many items were still in the queue (All the steps that we could’ve taken, but didn’t)

MazeSolverToo

* Class with the A\* algorithm implemented. The Main class
* Checks if there is a file, if not throws an exception error
* Else there is a file, setTimeDelay sets the speed of the maze, identify the start tile in the maze then change the colour of the start hexagon. Repaint refreshes the window to paint the colours in the maze where we currently are (uses java swing)

PriorityNode (basically a node class with a priority attribute)

* Basically each item in a queue is a node and we define the nodes here with a priority attribute. Not an ADT because we both define and add implementation
* Basically priority node will get the value and details of that hexagon and store that information in there, such as instantly getting the priority node, getting the node next to it,
* Element represents the name of the node

PriorityQueueADT

* What is an ADT? A class for a data type, where only behavior is **defined** but not implementation.
* So in priority queues we defined what it was and what methods a priority queue is capable of doing but did not define how they go about doing it
* LinkedPriorityQueue is the class where we define its implementation and how we go about doing it
* PriorityQueueADT and LinkedPriorityQueue gives the nodes capabilities to enqueue, dequeue, get the first item in the list, checks if list is empty, or returns a string list of the queue

LinkedPriorityQueue

Hexagon

* There are different hexagon types, the hexagons are displayed in different colours so you can track the algorithm’s progress
* stepsToMe attribute (attribute which determines g(x), steps it takes to get to the current hexagon).
* distanceToEnd() determines h(x) the number of steps needed till end of the maze