Alissa Niewiadomski

CS 32 Winter 2018

1. In mazestack.cpp
2. The first 12 coordinates popped from the stack are:

1. (4,3)

2. (3,3)

3. (5,3)

4. (5,2)

5. (5,1)

6. (6,1)

7. (7,1)

8. (8,1)

9. (8,2)

10. (6,3)

11. (4,4)

12. (4,5)

1. In mazequeue.cpp
2. The first 12 coordinates popped from the queue are:

1. (4,3)

2. (4,4)

3. (5,3)

4. (3,3)

5. (4,5)

6. (6,3)

7. (5,2)

8. (4,6)

9. (5,5)

10. (5,1)

11. (4,7)

12. (6,5)

In the stack implementation of pathExists, paths are searched out less “radially” than in the queue implementation. Because stacks implement a last-in, first-out method of searching, this particular implementation will search northward first, then westward, then southward, then eastward (as the implementation first checks a point’s neighbors in the order of east, south, west, north). If one is to look at the resultant searched maze in the given example for the first 12 coordinates visited, there is a clear trend that the south/western side of the maze was explored thoroughly before it started to expand eastward (as there were very early dead-ends from the starting point both directly northward and directly southward):

XXXXXXXXXX

X...X..X.X

X..XX....X

X.XeXXXX.X

XXXeee...X // Starting point at 4,3 in blue. We start seeing

XeeeX.XX.X // a search proceeding eastward, as the north,

XeXeX..X.X // west, and south searches have reached dead

XeXXXX.X.X // ends.

XeeX...X.X

XXXXXXXXXX

From my discrete structures class, I learned that this is a method of *depth-first* searching. The stack thoroughly searches the last branch it’s given (north) until it reaches a dead end, popping off values until it begins to search westward, southward, then eastward. This explains why the maze has been thoroughly searched through one path.

On the other hand, the queue implementation of pathExists explores the maze more “radially.” Since queues use a first in, first out method, each branch of the maze from the starting point is searched one point at a time per iteration through the while loop. Looking at the coordinates explored after the first 12 pops with this implementation, we see that the searched coordinates are less skewed towards the west/southern path and more evenly outward from the center:

XXXXXXXXXX

X...X..X.X

X..XX....X

X.XeXXXX.X

XXXeeeeeeX // Starting point at 4,3 in blue. We start seeing

XeeeXeXX.X // a search that explores points in succession

XeXeXeeX.X // from east, south, west, to north, repeating

X.XXXX.X.X // this cycle until the end point is found.

X..X...X.X

XXXXXXXXXX

Again, from my discrete structures class, I learned that this is a method of *breadth-first* searching. The queue, unlike the stack, searches each “level” in succession (“level” meaning coordinates stratified by degrees of separation from the start point). Once a path reaches a dead end, the queue will not search that path anymore (as there are no coordinates to add). Each path stemming from the start is searched through each iteration of the while loop, whereas only one path is searched at a time in the depth-first method. Interestingly, breadth-first algorithms let one find the shortest path.

1. In eval.cpp