PATH FINDING VISUALIZER

Using A* Algorithm.

Aditya Rajesh Nair

Roll No: 01 ; TyBSc:CS

NATIONAL EDUCATION SOCIETY'S

Ratnam College of Arts, Science & Commerce
Bhandup, Mumbai – 400 078.

Department of Computer Science

CERTIFICATE

Class: *Roll / Seat No.:* *Year:*

This is to certify that Mr./Ms. __________________________ has satisfactorily completed the project work ______________________for partial fulfillment of the 3 years Degree Course Bachelor in Computer Science for the University of Mumbai for the year ________ to ________.

Place: _______________

Date: _______________

Project Guide

In-charge,
Dept. of Computer Science

Signature of Examiner with Date

Contents:

Contents:

Introduction:

Path finding Visualizer helps the user to visualizer the A*Algorithm by using many hindrances in 2D grid board/canvas & walls.

A* Algorithm:

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

Why A* Algorithm?

Informally speaking, A* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms. This fact is cleared in detail in below sections.
And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation).

Project Language:

Python:

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented and functional programming

Tkinter:

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications.

import random:

The Python import random module in Python defines a series of functions for generating or manipulating random integers

import time:

The Python time module provides many ways of representing time in code, such as objects, numbers, and strings. It also provides functionality other than representing time, like waiting during code execution and measuring the efficiency of your code.

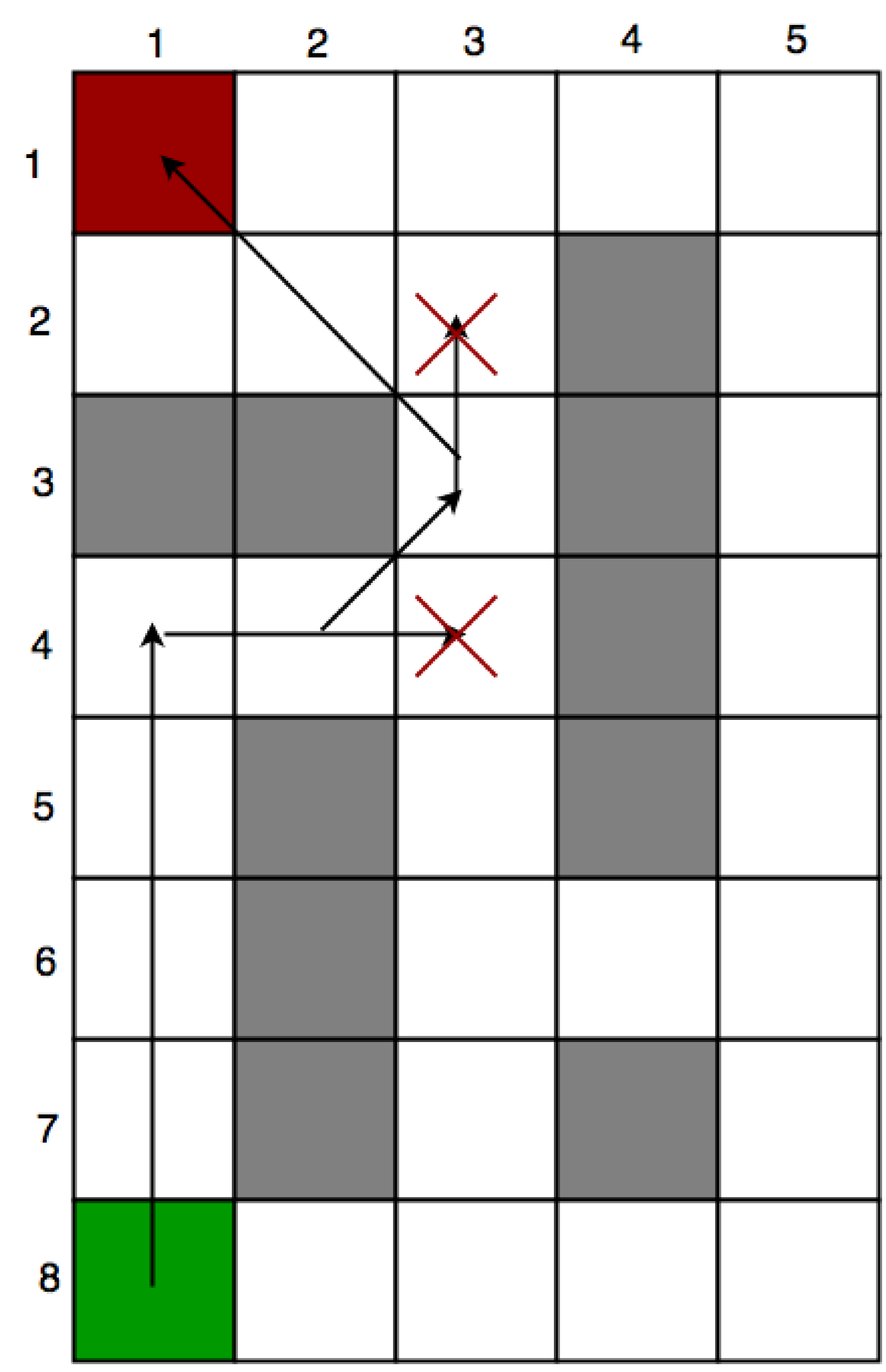
Path Finding Visualizer:

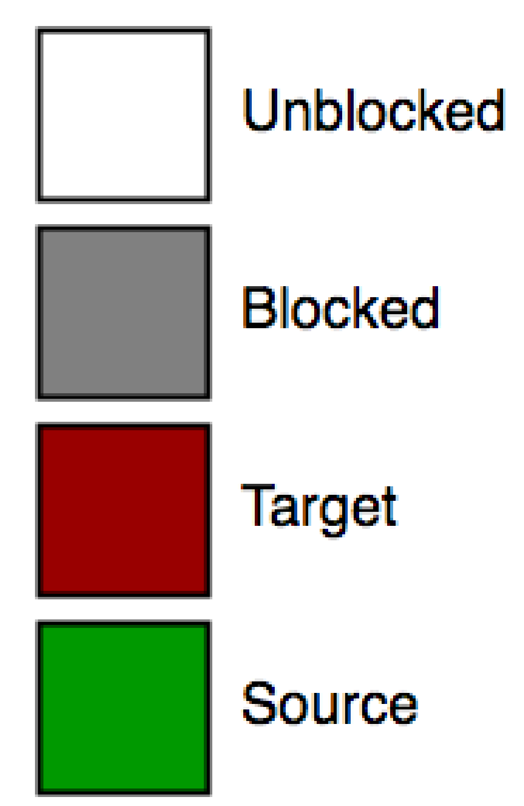
Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A* Search Algorithm comes to the rescue.
What A* Search Algorithm does is that at each step it picks the node according to a value-‘f’ which is a parameter equal to the sum of two other parameters – ‘g’ and ‘h’. At each step it picks the node/cell having the lowest ‘f’, and process that node/cell.

We define ‘g’ and ‘h’ as simply as possible below

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.

h = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don’t know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.). There can be many ways to calculate this ‘h’ which are discussed in the later sections.





A* Search Algorithm makes the most intelligent choice at each step. Hence you can see that algorithm goes from (4,2) to (3,3) and not (4,3) (shown by cross).

Similarly the algorithm goes from (3,3) to (2,2) and not (2,3) (shown by cross).

Code:

```
## Find Shortest Path ##
from tkinter import *
import random
import time

window = Tk()
window.title("Shortest Path")
window.maxsize(800, 800)
window.config(bg="black")

canvas = Canvas(window,
height="700", width="700",
bg="black")
canvas.grid(row=1, column=0,
padx=5, pady=5)

frame = Frame(window, height=50,
width=700, bg='pink')
frame.grid(row=0, column=0,
padx=5, pady=5)

xy = 25
h, w = 700, 700
maxBdr = 100 # [0 < maxBar <=
1225]
queue = []
temp = []
```

```
def rect(x1, y1, x2, y2, color):
    canvas.create_rectangle(x1, y1,
x2, y2, fill=color)

def generateWindow():
    queue.clear()
    temp.clear()
    canvas.delete("all")
    global list
    list = []
    for x in range(0, h, xy):
        for y in range(0, w, xy):
            list.append([x, y+xy, x+xy,
y])
            rect(x, y+xy, x+xy, y,
"white")
# generateWindow()

#For Border
def generateBorder():
    global borderList, pathList
    borderList = []
    pathList = []
    for n in range(maxBdr):
        r=random.randrange(len(list))
        borderList.append(list[r])
        list.pop(r)
```

```
for b in range(len(borderList)):
        x, y_xy, x_xy, y = borderList[b]
[0], borderList[b][1], borderList[b]
[2], borderList[b][3]
        rect(x, y_xy, x_xy, y, "black")

    for l in list:
        if(l not in borderList):
            pathList.append(l)
# generateBorder()

def generateTargets():
    global rs, re
    rs =
random.randrange(len(pathList))
    canvas.create_oval(pathList[rs]
[0], pathList[rs][1], pathList[rs][2],
pathList[rs][3], fill="green")
    re =
random.randrange(len(pathList))
    canvas.create_oval(pathList[re]
[0], pathList[re][1], pathList[re][2],
pathList[re][3], fill="red")
# generateTargets()
# print((pathList))

generateWindow()
generateBorder()
generateTargets()
```

```
pp = []
n = 0
def setInQ(x1, y1, x2, y2):
    c = "blue"
    p = []
    p.append([x1, y1, x2, y2])
    if [x1+xy, y1, x2+xy, y2] in pathList
and [x1+xy, y1, x2+xy, y2] not in temp
and [x1+xy, y1, x2+xy, y2] not in queue:
        queue.append([x1+xy, y1, x2+xy,
y2])
        p.append([x1+xy, y1, x2+xy, y2])
        canvas.create_oval(x1+xy+(xy/3),
y1-(xy/3), x2+xy-(xy/3), y2+(xy/3),
fill=c)
    if [x1, y1+xy, x2, y2+xy] in pathList
and [x1, y1+xy, x2, y2+xy] not in temp
and [x1, y1+xy, x2, y2+xy] not in queue:
        queue.append([x1, y1+xy, x2,
y2+xy])
        p.append([x1, y1+xy, x2, y2+xy])
        canvas.create_oval(x1+(xy/3),
y1+xy-(xy/3), x2-(xy/3), y2+xy+(xy/3),
fill=c)
```

```
 if [x1-xy, y1, x2-xy, y2] in pathList and
[x1-xy, y1, x2-xy, y2] not in temp and
[x1-xy, y1, x2-xy, y2] not in queue:
 queue.append([x1-xy, y1, x2-xy, y2])
 p.append([x1-xy, y1, x2-xy, y2])
 canvas.create_oval(x1-xy+(xy/3), y1-
(xy/3), x2-xy-(xy/3), y2+(xy/3), fill=c)
 if [x1, y1-xy, x2, y2-xy] in pathList and
[x1, y1-xy, x2, y2-xy] not in temp and
[x1, y1-xy, x2, y2-xy] not in queue:
 queue.append([x1, y1-xy, x2, y2-xy])
 p.append([x1, y1-xy, x2, y2-xy])
 canvas.create_oval(x1+(xy/3), y1-xy-
(xy/3), x2-(xy/3), y2-xy+(xy/3), fill=c)
 pp.append(p)
```

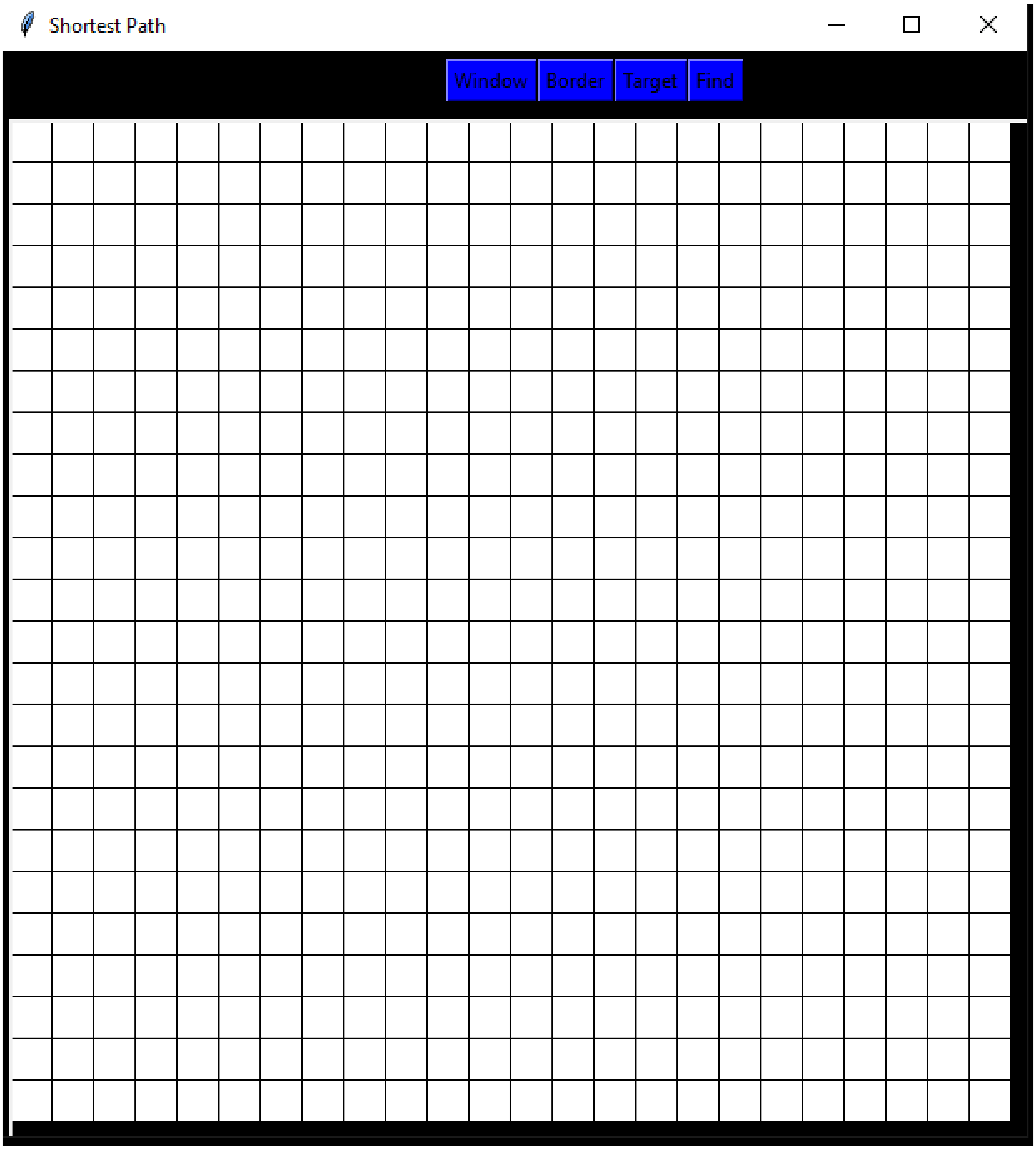
```
def find():
    global fond
    fond = False
    setInQ(pathList[rs][0], pathList[rs][1],
pathList[rs][2], pathList[rs][3])
    while len(queue) != 0:
        q = queue.pop(0)
        if([q[0], q[1], q[2], q[3]] ==
[pathList[re][0], pathList[re][1],
pathList[re][2], pathList[re][3]]):
            rect(q[0], q[1], q[2], q[3], "blue")
            fond = True
            break
        elif [q[0], q[1], q[2], q[3]] ==
[pathList[rs][0], pathList[rs][1],
pathList[rs][2], pathList[rs][3]]:
            rect(q[0], q[1], q[2], q[3], "green")
        else:
            setInQ(q[0], q[1], q[2], q[3])
            rect(q[0], q[1], q[2], q[3], "aqua")
            canvas.update()
            # time.sleep(0.1)
        temp.append(q)
```

```
# Sotest Path
    point = [pathList[re][0], pathList[re][1],
pathList[re][2], pathList[re][3]]
    sotestPath = []
    if fond == True:
        for n in range(len(pp), 0, -1):
            if point in pp[n-1] and point not in
sotestPath:
                sotestPath.append(point)
                point = pp[n-1][0]

 canvas.create_rectangle(point[0],
point[1], point[2], point[3], fill="orange")
                canvas.update()
    canvas.create_oval(pathList[rs][0],
pathList[rs][1], pathList[rs][2], pathList[rs]
[3], fill="green")
    canvas.create_oval(pathList[re][0],
pathList[re][1], pathList[re][2], pathList[re]
[3], fill="red")
```

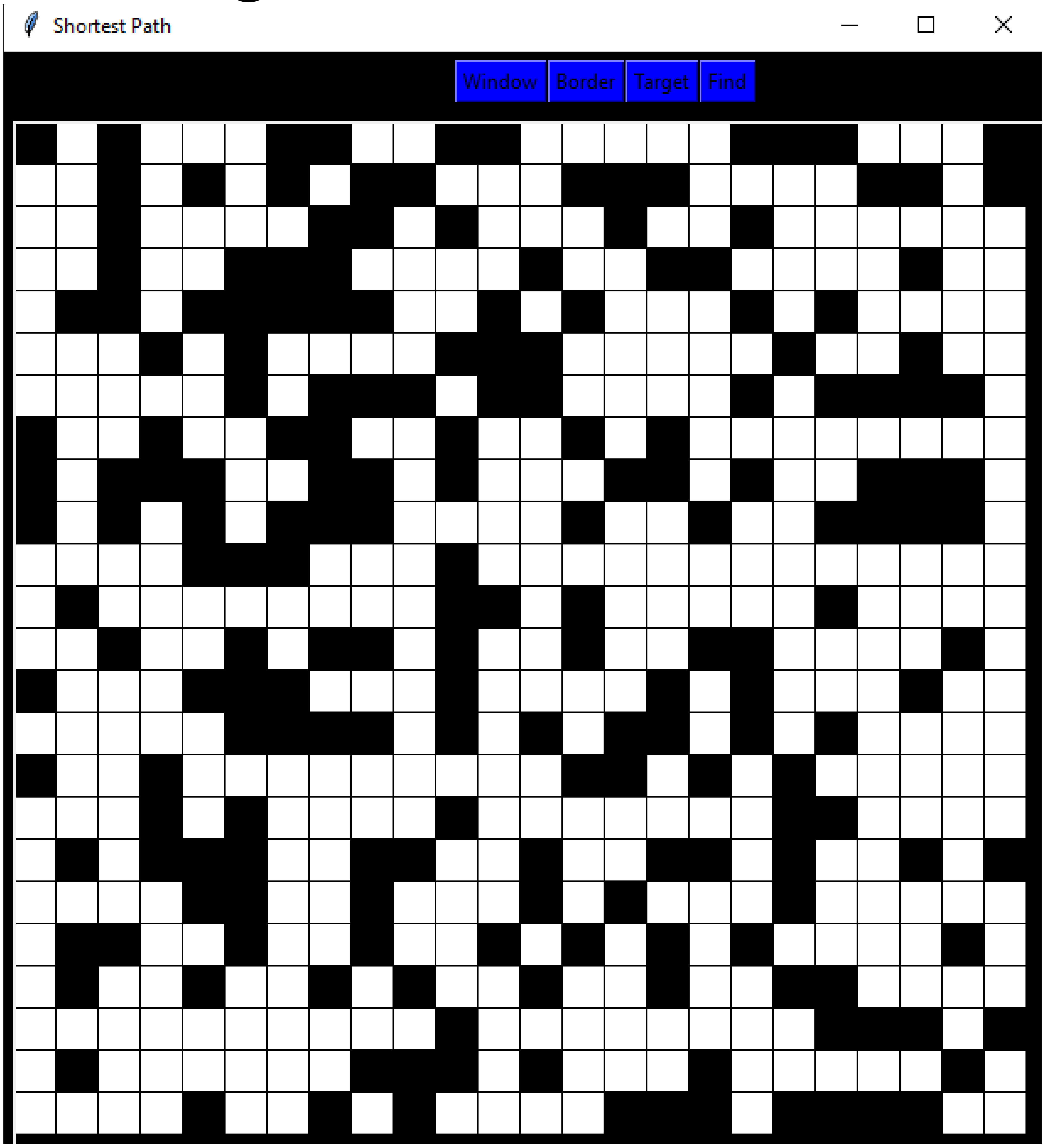
```
btn1 = Button(frame, text='Window',
bg='blue', command=generateWindow)
btn1.grid(row=0, column=0)
btn2 = Button(frame, text='Border',
bg='blue', command=generateBorder)
btn2.grid(row=0, column=1)
btn3 = Button(frame, text='Target',
bg='blue', command=generateTargets)
btn3.grid(row=0, column=2)
btn4 = Button(frame, text='Find',
bg='blue', command=find)
btn4.grid(row=0, column=3)

window.mainloop()
```

ScreenShots:

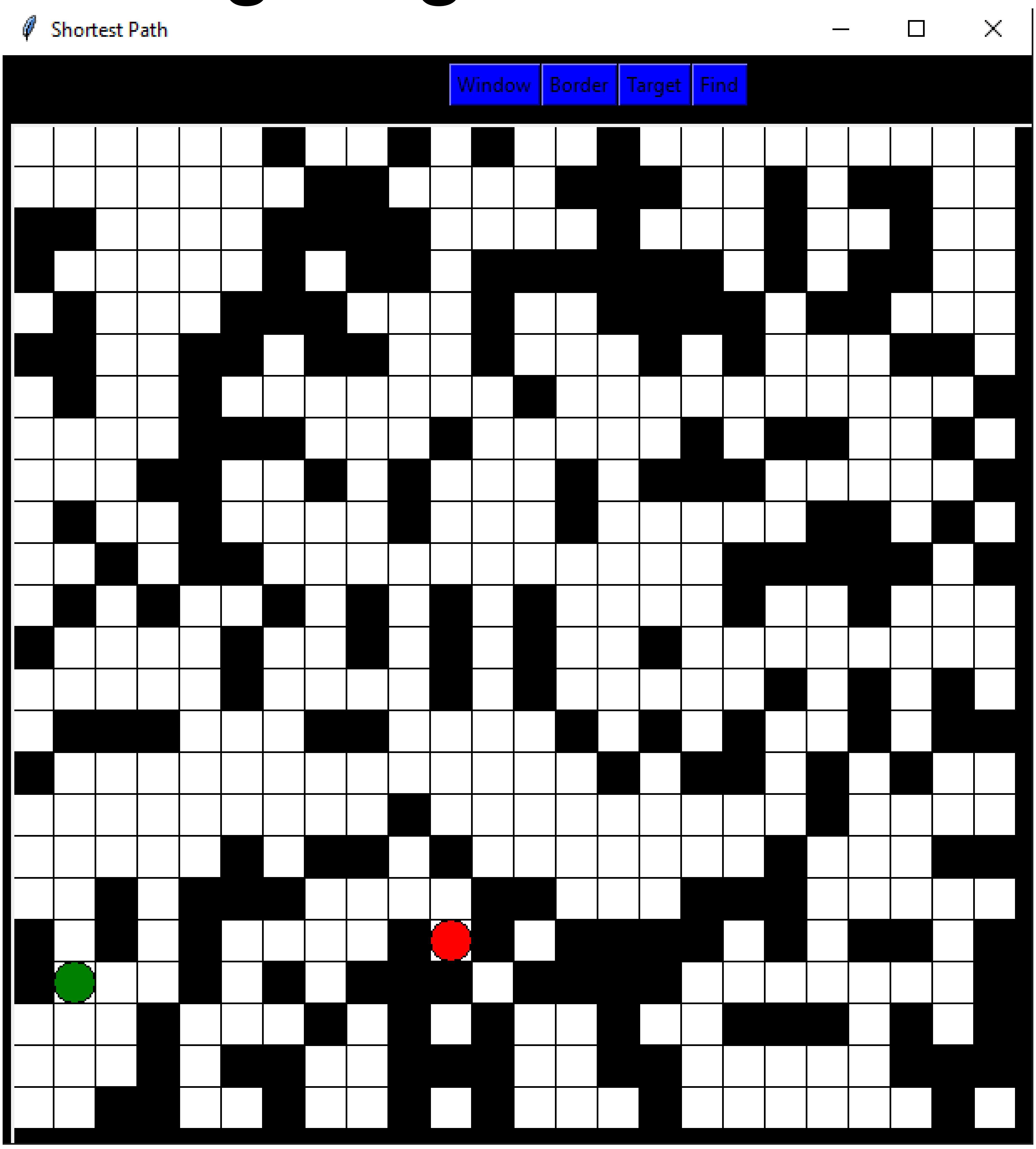
Main Screen:



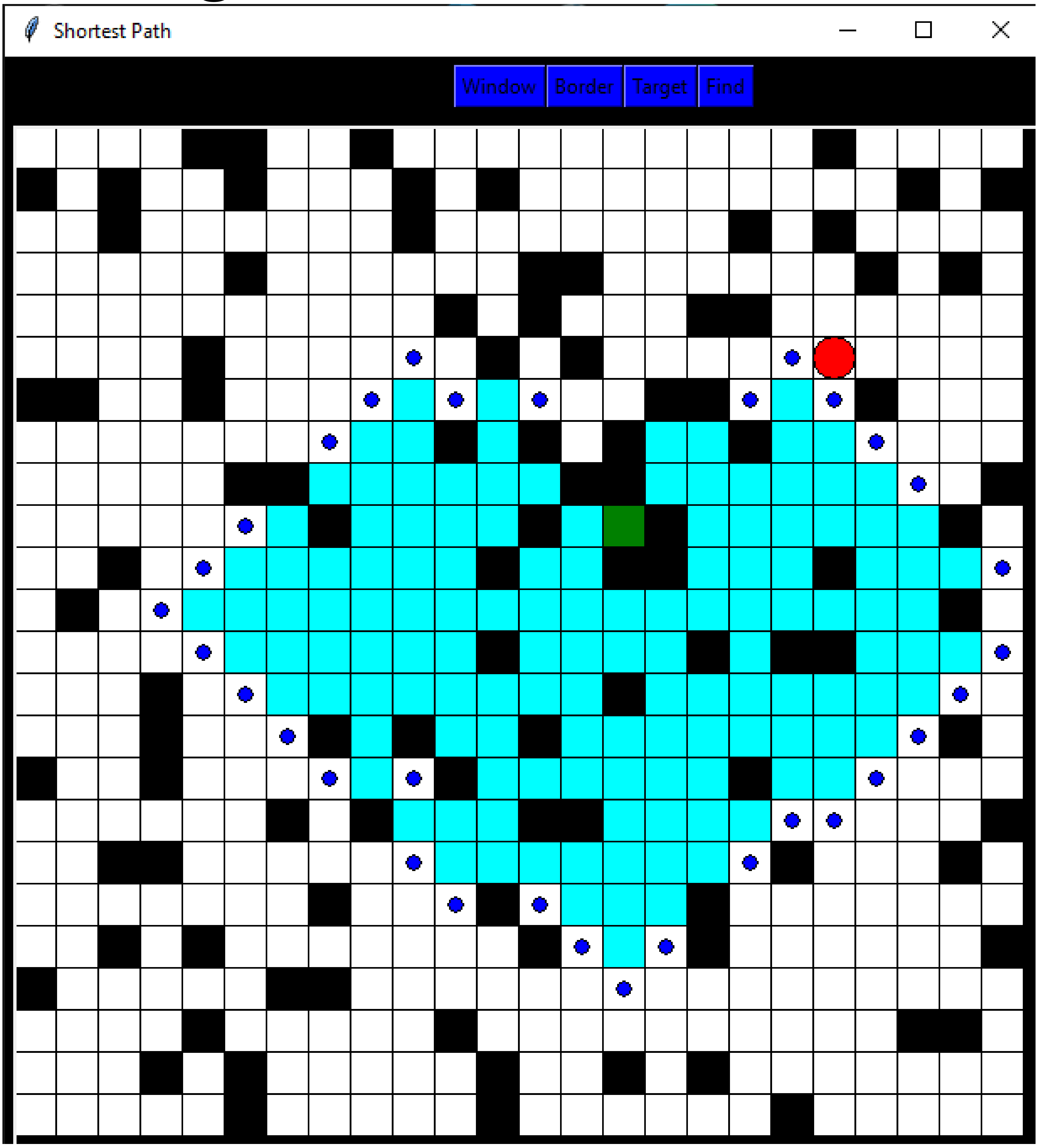
Adding Walls:



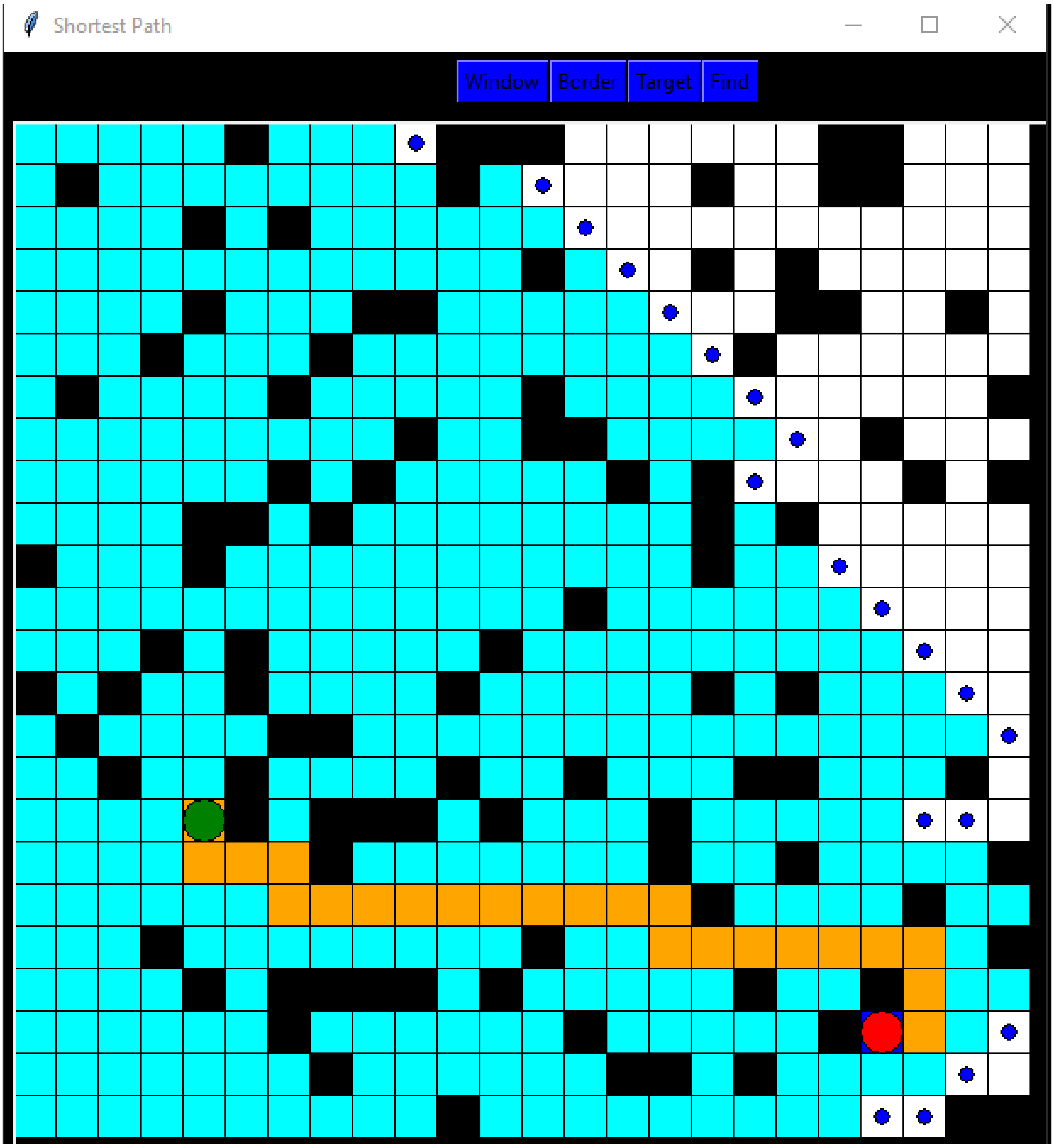
Adding Targets:



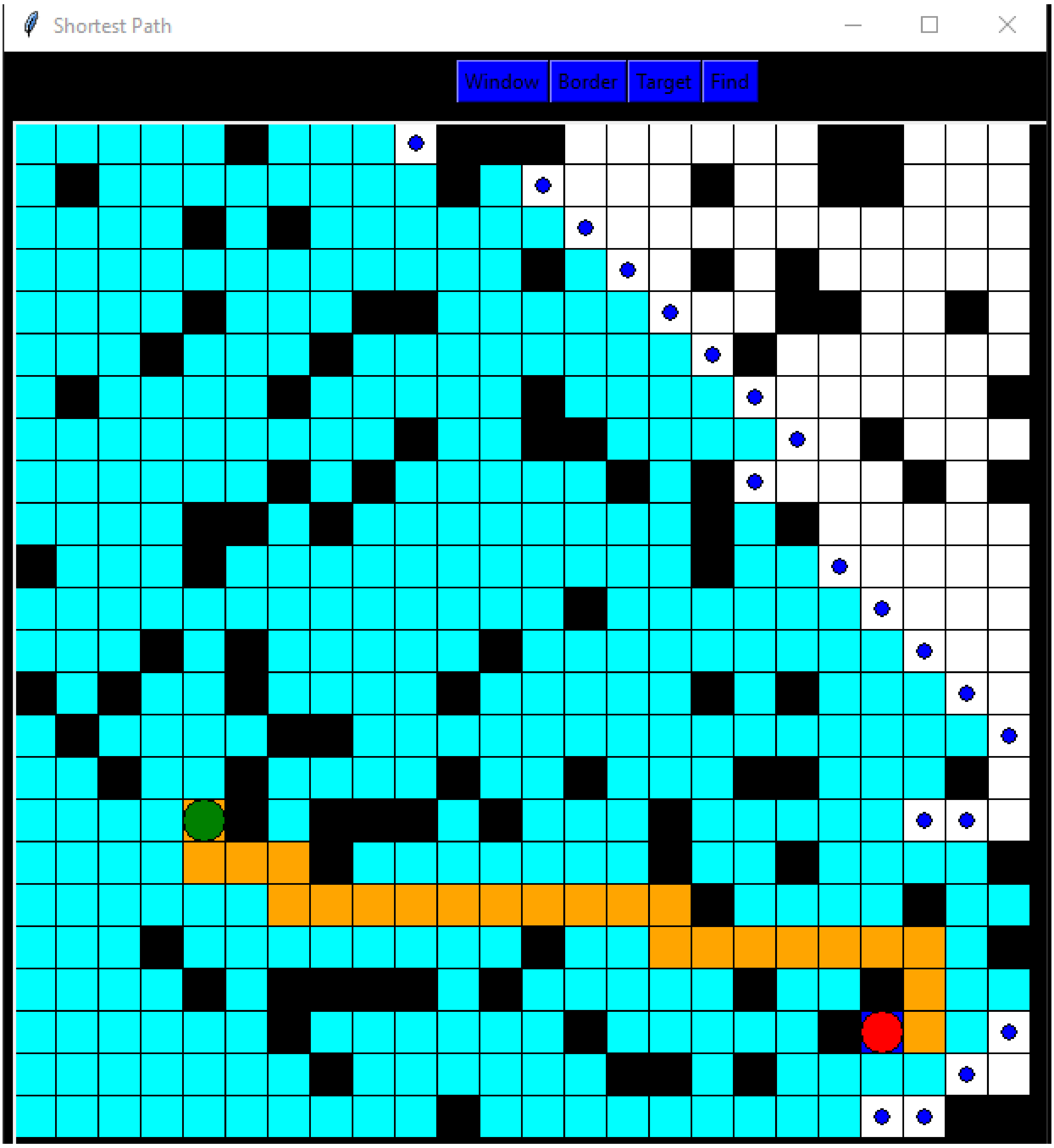
During Process:



End Result:



End Result:



Conclusion:

Path finding Visualizer helps the user to visualize and understand the A*Algorithm by using many hindrances in 2D grid board/canvas & walls.

References:

Youtube

https://www.youtube.com/watch?v=msttfIHHkak

geeksforgeeks

https://www.geeksforgeeks.org/a-search-algorithm/

My Project Github LInk

https://github.com/Aditya-bot28/PFV.git