

Search Game

Definition

The aim of the game is to detect the walls chosen by the user and go to the target point in the shortest way from the given starting point without touching or passing through them. The user also chooses the starting and destination point and the search algorithm. There are two types of searching algorithms in this game, one is Uniform-Cost Search and the other is A* Search. The game starts when the user clicks the start button after entering the values. At each step, we can see the costs of the paths and the opened node. The results appear on the right side of the screen and the path is indicated in yellow.

Video: <https://www.youtube.com/watch?v=jjC2IWRDey4>

The screenshot shows a web application titled "Search Game". It features a 3x3 grid with cells labeled A through I. Below the grid, there are input fields for "Initial State:" and "Goal State:", a "Search Type:" dropdown menu currently set to "UCS", and a "Define Walls" section with checkboxes for edges AB, AD, BC, BE, CF, DE, DG, EF, EH, GH, HI, and FI. A "Start" button is located at the bottom of the input section. To the right of the main interface is a large empty box labeled "Algorithm Steps:".

A	B	C
D	E	F
G	H	I

Initial State:

Goal State:

Search Type:

Define Walls

☐ AB ☐ AD ☐ BC

☐ BE ☐ CF ☐ DE

☐ DG ☐ EF ☐ EH

☐ GH ☐ HI ☐ FI

Algorithm Steps:

Requirement 1:

We defined the rooms and their connections using a dictionary.

```
room_dict = {  
    'A': (1, 1), 'B': (1, 2), 'C': (1, 3),  
    'D': (2, 1), 'E': (2, 2), 'F': (2, 3),  
    'G': (3, 1), 'H': (3, 2), 'I': (3, 3)  
}
```

```
connections = {  
    'A': ['B', 'D'],  
    'B': ['A', 'C', 'E'],  
    'C': ['B', 'F'],  
    'D': ['A', 'E', 'G'],  
    'E': ['B', 'D', 'F', 'H'],  
    'F': ['C', 'E', 'I'],  
    'G': ['D', 'H'],  
    'H': ['E', 'G', 'I'],  
    'I': ['F', 'H']  
}
```

Requirement 2:

We received the source and goal room from the user.

```
def start_game():  
    initial_state = initial_state_entry.get().upper()  
    goal_state = goal_state_entry.get().upper()  
    search_type = search_type_var.get().upper()  
  
    if initial_state not in room_dict:  
        messagebox.showerror("Error", "Invalid initial state. Please enter a valid letter (A-I).")  
    elif goal_state not in room_dict:  
        messagebox.showerror("Error", "Invalid goal state. Please enter a valid letter (A-I).")  
    else:  
        if search_type == 'UCS':  
            uniform_cost_search(initial_state, goal_state, room_dict, search_type)  
        elif search_type == 'A*':  
            a_star_search(initial_state, goal_state, room_dict, search_type)  
        else:  
            messagebox.showerror("Error", "Invalid search type. Please enter 'UCS' or 'A*'.")
```

Requirement 3:

In this part of the code, we created separate check buttons for all wall possibilities, that is, twelve. Whichever walls the user clicks on, it calls the `define_walls` function and deletes those nodes from possible connections.

```
v1 = tk.IntVar()
wall1_check = tk.Checkbutton(walls_frame, text="AB", variable=v1, command=lambda: define_walls(v1))
wall1_check.grid(row=0, column=0, padx=10, pady=10)
```

```
def define_walls(var):
    walls_dict = {
        'AB': v1, 'AD': v2, 'BC': v3,
        'BE': v4, 'CF': v5, 'DE': v6,
        'DG': v7, 'EF': v8, 'EH': v9,
        'GH': v10, 'HI': v11, 'FI': v12
    }

    walls = [key for key, value in walls_dict.items() if value.get() == 1]

    for wall in walls:
        room1, room2 = wall[0], wall[1]
        if room1 in connections and room2 in connections[room1]:
            connections[room1].remove(room2)
            connections[room2].remove(room1)

    return connections
```

Requirement 4:

We wrote such an algorithm to establish the connection between direction and cost.

```
if room_dict[current[-1]][0] == room_dict[i][0]:
    new_cost += 2
elif room_dict[current[-1]][1] == room_dict[i][1]:
    new_cost += 1
```

Requirement 5:

We used OptionMenu widget for search type selection.

```
search_type_label = tk.Label(root, text="Search Type:", font=('Arial', 12))
search_type_label.grid(row=5, column=0, padx=10, pady=10)

search_type_var = tk.StringVar(root)
search_type_var.set("UCS")

search_type_dropdown = tk.OptionMenu(root, search_type_var, "UCS", "A*")
search_type_dropdown.grid(row=5, column=1, padx=10, pady=10)
```

```
search_type = search_type_var.get().upper()
```

```
if search_type == 'UCS':
    uniform_cost_search(initial_state, goal_state, room_dict, search_type)
elif search_type == 'A*':
    a_star_search(initial_state, goal_state, room_dict, search_type)
```

A*:

```
if search_type == 'A*':
    if room_dict[current[-1]][0] == room_dict[i][0]:
        new_cost = new_cost + 2 + manhattan_distance(i, goal_state)
    elif room_dict[current[-1]][1] == room_dict[i][1]:
        new_cost = new_cost + 1 + manhattan_distance(i, goal_state)
```

UCS:

```
if search_type == 'UCS':
    if room_dict[current[-1]][0] == room_dict[i][0]:
        new_cost += 2
    elif room_dict[current[-1]][1] == room_dict[i][1]:
        new_cost += 1
```

Requirement 6:

If the algorithm cannot reach the target in 10 nodes, it gives such an error message. But it still shows the stages and opened nodes.

Search Game

A	B	C
D	E	F
G	H	I

Initial State:
Goal State:
Search Type:

Define Walls:

<input type="checkbox"/> AB	<input type="checkbox"/> AD	<input checked="" type="checkbox"/> BC
<input type="checkbox"/> BE	<input type="checkbox"/> CF	<input type="checkbox"/> DE
<input type="checkbox"/> DG	<input type="checkbox"/> EF	<input type="checkbox"/> EH
<input type="checkbox"/> GH	<input type="checkbox"/> HI	<input type="checkbox"/> FI

Algorithm Steps:
Step 1: Fringe - [[0, 'A']]
Expanded Node: A
Step 2: Fringe - [[1, 'AD'], [2, 'AB']]
Expanded Node: D
Step 3: Fringe - [[2, 'AB'], [2, 'ADG'], [3, 'ADE']]
Expanded Node: B
Step 4: Fringe - [[2, 'ADG'], [3, 'ABE'], [3, 'ADE']]
Expanded Node: G
Step 5: Fringe - [[3, 'ABE'], [3, 'ADE'], [4, 'ADGH']]
Expanded Node: E
Step 6: Fringe - [[3, 'ADE'], [4, 'ABEH'], [4, 'ADGH'], [5, 'ABEF']]
Expanded Node: E
Step 7: Fringe - [[4, 'ABEH'], [4, 'ADEH'], [4, 'ADGH'], [5, 'ABEF'], [5, 'ADEF']]
Expanded Node: H
Step 8: Fringe - [[4, 'ADEH'], [4, 'ADGH'], [5, 'ABEF'], [5, 'ADEF'], [6, 'ABEHI']]
Expanded Node: H
Step 9: Fringe - [[4, 'ADGH'], [5, 'ABEF'], [5, 'ADEF'], [6, 'ABEHI'], [6, 'ADEHI']]
Expanded Node: H
Step 10: Fringe - [[5, 'ABEF'], [5, 'ADEF'], [6, 'ABEHI'], [6, 'ADEHI'], [6, 'ADGHI']]
Expanded Node: F

Goal Not Reached

i Goal can't be reached! You reached the 10th node.

Goal Not Reached

i Goal can't be reached! You reached the 10th node.

This is the output when the goal is reached. Also, if it reaches the destination, the shortest path is indicated in yellow.

Search Game

A	B	C
D	E	F
G	H	I

Initial State:

A

Goal State:

C

Search Type:

A* —

Define Walls

☐ AB

☐ AD

☒ BC

☐ BE

☐ CF

☐ DE

☐ DG

☐ EF

☐ EH

☐ GH

☐ HI

☐ FI

Start

Algorithm Steps:

Step 1: Fringe - [[2, 'A']]
Expanded Node: A
Step 2: Fringe - [[3, 'AB'], [4, 'AD']]
Expanded Node: B
Step 3: Fringe - [[4, 'AD'], [5, 'ABE']]
Expanded Node: D
Step 4: Fringe - [[5, 'ABE'], [5, 'ADE'], [6, 'ADG']]
Expanded Node: E
Step 5: Fringe - [[5, 'ADE'], [6, 'ABEF'], [6, 'ADG'], [7, 'ABEH']]
Expanded Node: E
Step 6: Fringe - [[6, 'ABEF'], [6, 'ADEF'], [6, 'ADG'], [7, 'ABEH'], [7, 'ADEH']]
Expanded Node: F
Step 7: Fringe - [[6, 'ABEFC'], [6, 'ADEF'], [6, 'ADG'], [7, 'ABEH'], [7, 'ADEH'], [8, 'ABEFI']]
Expanded Node: C
Path: ABEFC
Cost: 6

Algorithm Execution

Uniform-Cost Search:

Initial State: A

Goal State: H

Walls: AD – CF – DG – EF

STEP 1:

Fringe: A(0)

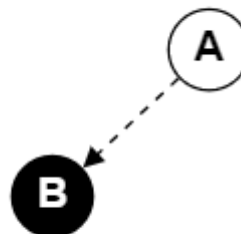
Expanded Node: A



STEP 2:

Fringe: AB(2)

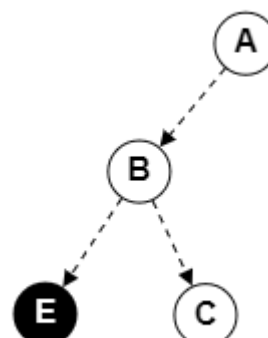
Expanded Node: B



STEP 3:

Fringe: ABE(3), ABC(4)

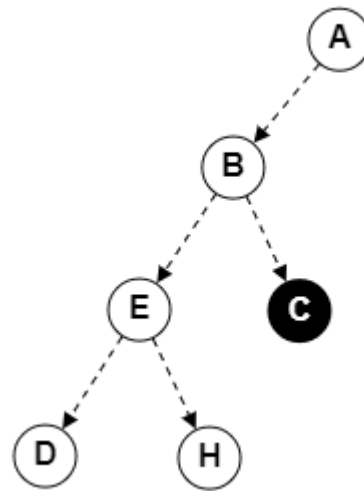
Expanded Node: E



STEP 4:

Fringe: ABC(4), ABEH(4), ABED(5)

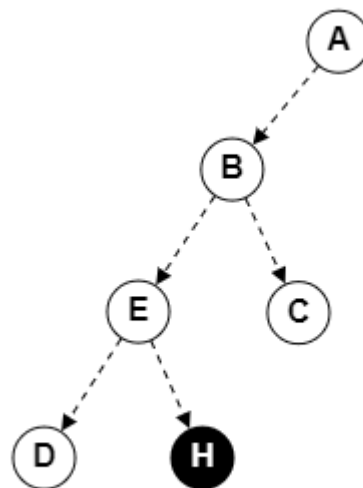
Expanded Node: C



STEP 5:

Fringe: ABEH(4), ABED(5)

Expanded Node: H



PATH = ABEH

COST = 4

A* Search:

Initial State: I

Goal State: D

Walls: AB – CF – FI – EH

STEP 1:

Fringe: I(3)

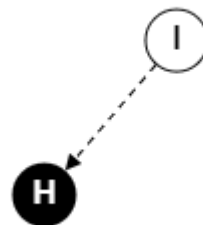
Expanded Node: I



STEP 2:

Fringe: IH(4)

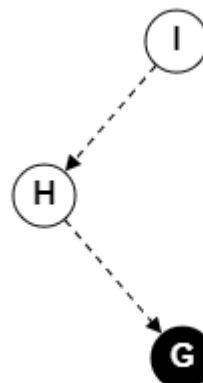
Expanded Node: H



STEP 3:

Fringe: IHG(5)

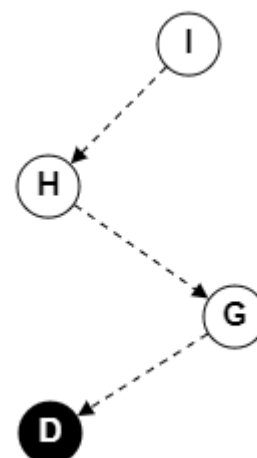
Expanded Node: G



STEP 4:

Fringe: IHGD(5)

Expanded Node: D



PATH = IHGD

COST = 5