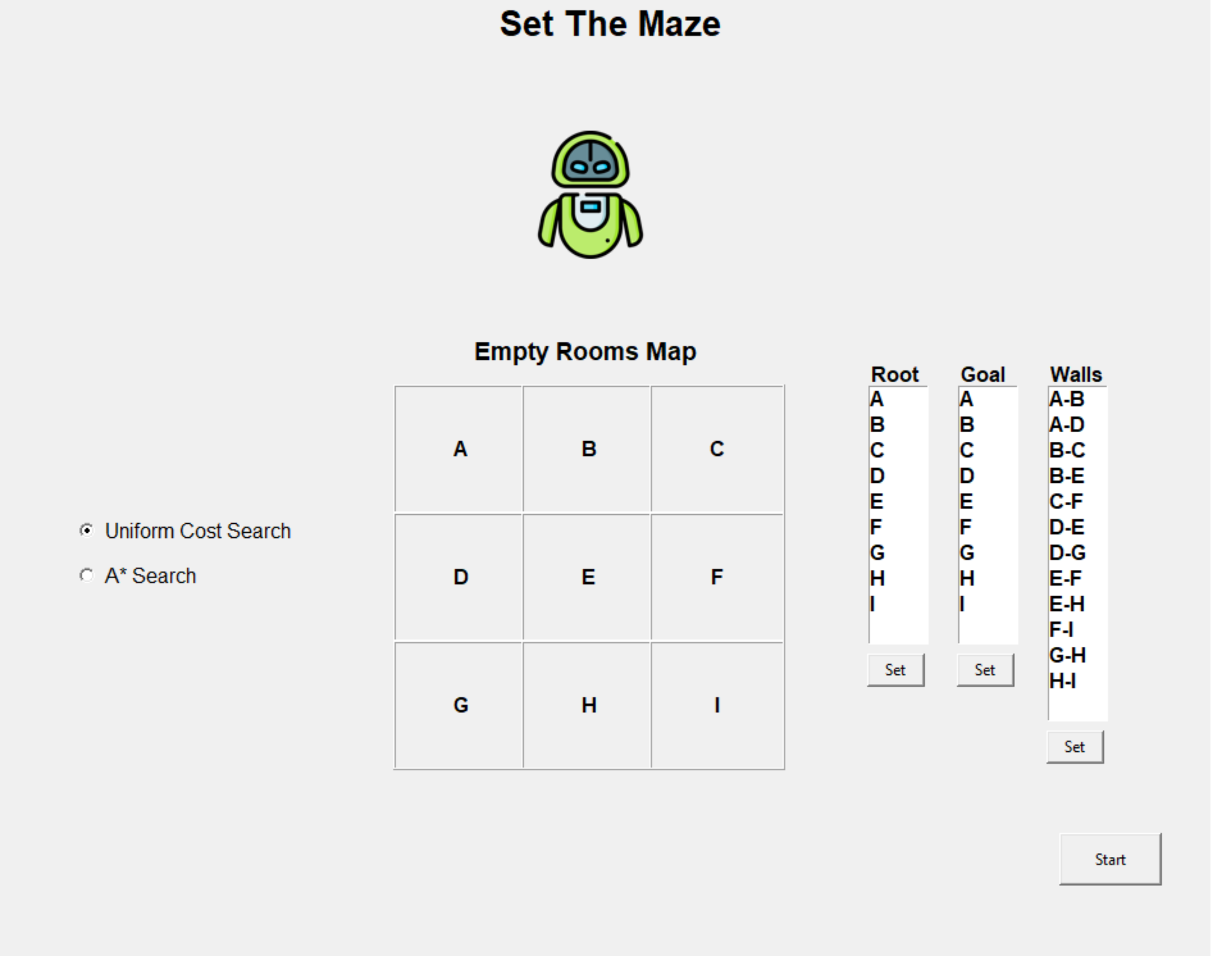
Room Maze

Game Definition

The purpose of the game is that user creates a maze for the robot by setting walls between rooms and the robot that is placed at the start room tries to reach to the goal room. There are 9 square rooms in 3x3 format, 15 available walls between rooms and 2 search algorithms which are A* Search and Uniform Cost Search. On the first page of the game, user selects the search algorithm, start room, goal room and walls. After everything is set, by clicking the next button, a new page comes which shows each stage of the search algorithm including the fringe information, expanded room and the path robot is taking currently with its cost step by step until the algorithm comes to a result. There are 3 possible results. The first one is the best case that robot reaches the goal room. The game shows the goal path and cost on the screen. The second case is that the robot is stuck in a room and cannot go anywhere. And the final one is for infinite loops. If number of steps comes to 10, the algorithm terminates itself.

YouTube Video = https://youtu.be/-L4vh-ptC8E

Game Requirements



Set The Maze

● Uniform Cost Search
○ A* Search

Empty Rooms Map

A	B	C
D	E	F
G	H	I

Root: A B C D E F G H I

Goal: A B C D E F G H I

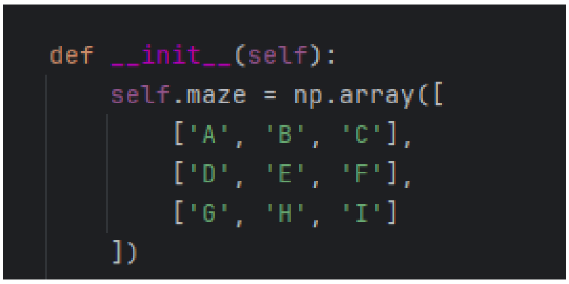
Walls: A-B A-D B-C B-E C-F D-E D-G E-F E-H F-I G-H H-I

figure 1

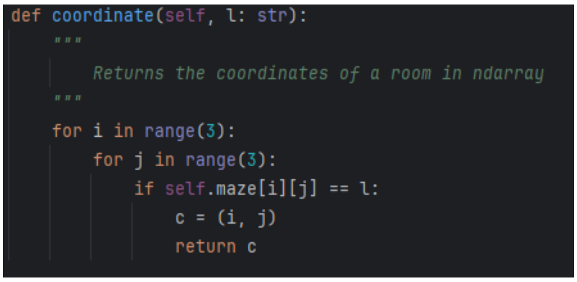
Requirement 1:

- There will be 9 rooms in the game (They will be located in 3x3 structure as given in example)

Game rooms are specified as 3x3 NumPy ndarray to perform background operations. They are also placed as in specified structure in both pages in the game as you can see it from figure 1.



```python
def __init__(self):
    self.maze = np.array([
        ['A', 'B', 'C'],
        ['D', 'E', 'F'],
        ['G', 'H', 'I']
    ])
```

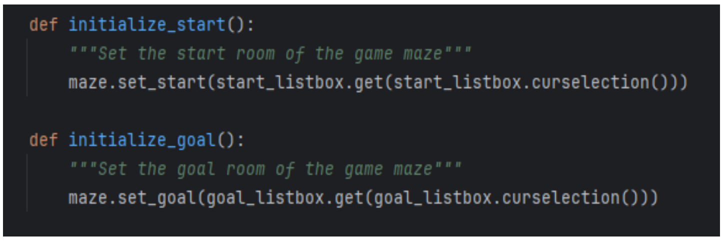


```python
def coordinate(self, l: str):
    """
        Returns the coordinates of a room in ndarray
    """
    for i in range(3):
        for j in range(3):
            if self.maze[i][j] == l:
                c = (i, j)
                return c
```

Requirement 2:

- The source and goal rooms will be given by the user.

Start and goal rooms are set in class ‘maze’ by getting the selection information from the list boxes in games maze set screen.

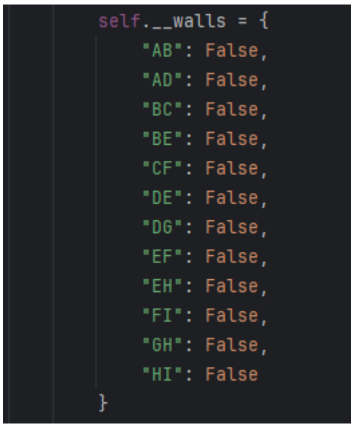


```python
def initialize_start():
    """Set the start room of the game maze"""
    maze.set_start(start_listbox.get(start_listbox.curselection()))

def initialize_goal():
    """Set the goal room of the game maze"""
    maze.set_goal(goal_listbox.get(goal_listbox.curselection()))
```

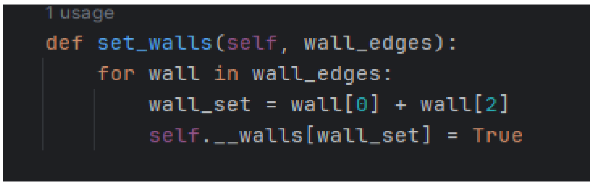
Requirement 3:

- The walls between the rooms will be given by the user.

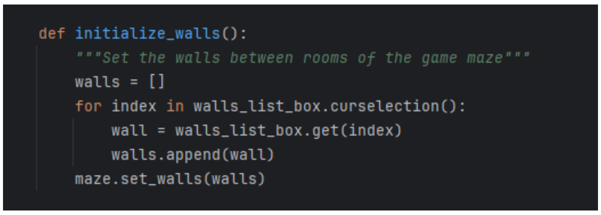
Wall information is held in a Python dictionary called walls. It stores wall names as key and boolean values as value. If the value of a key is true, then it means that the wall is active. The program gets the wall selection from user and sets the walls.



```python
self.__walls = {
    "AB": False,
    "AD": False,
    "BC": False,
    "BE": False,
    "CF": False,
    "DE": False,
    "DG": False,
    "EF": False,
    "EH": False,
    "FI": False,
    "GH": False,
    "HI": False
}
```



```python
1 usage
def set_walls(self, wall_edges):
    for wall in wall_edges:
        wall_set = wall[0] + wall[2]
        self.__walls[wall_set] = True
```



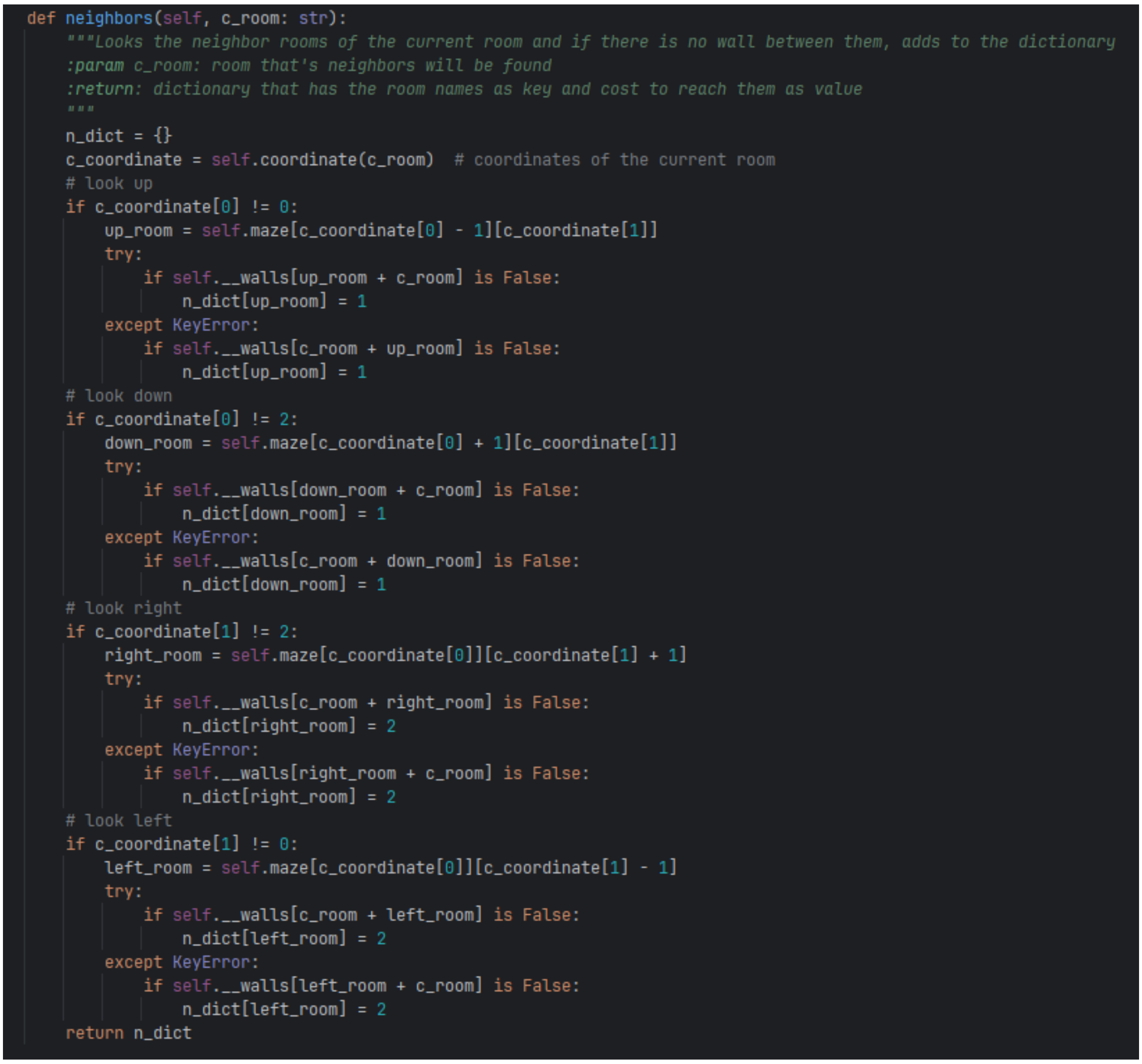
```python
def initialize_walls():
    """Set the walls between rooms of the game maze"""
    walls = []
    for index in walls_list_box.curselection():
        wall = walls_list_box.get(index)
        walls.append(wall)
    maze.set_walls(walls)
```

Requirement 4:

- The robot can be moved up, down, right, or left. The costs are:

  right or left move = 2
  up or down move = 1

At expanding room stage, program gets the available neighbor rooms considering the walls and creates a dictionary by neighbor room name as key and cost to go to that room as value. Costs are set considering the requirement information.

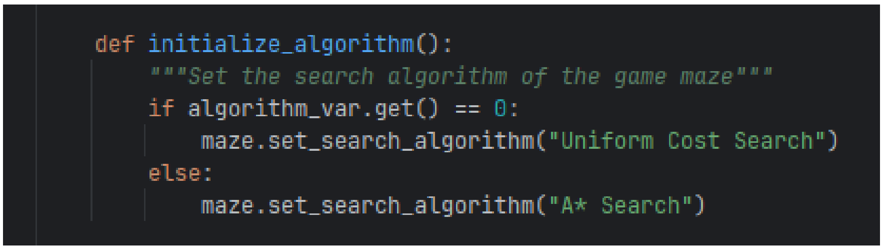


```python
def neighbors(self, c_room: str):
    """Looks the neighbor rooms of the current room and if there is no wall between them, adds to the dictionary
    :param c_room: room that's neighbors will be found
    :return: dictionary that has the room names as key and cost to reach them as value
    """
    n_dict = {}
    c_coordinate = self.coordinate(c_room)  # coordinates of the current room
    # look up
    if c_coordinate[0] != 0:
        up_room = self.maze[c_coordinate[0] - 1][c_coordinate[1]]
        try:
            if self.__walls[up_room + c_room] is False:
                n_dict[up_room] = 1
        except KeyError:
            if self.__walls[c_room + up_room] is False:
                n_dict[up_room] = 1
    # look down
    if c_coordinate[0] != 2:
        down_room = self.maze[c_coordinate[0] + 1][c_coordinate[1]]
        try:
            if self.__walls[down_room + c_room] is False:
                n_dict[down_room] = 1
        except KeyError:
            if self.__walls[c_room + down_room] is False:
                n_dict[down_room] = 1
    # look right
    if c_coordinate[1] != 2:
        right_room = self.maze[c_coordinate[0]][c_coordinate[1] + 1]
        try:
            if self.__walls[c_room + right_room] is False:
                n_dict[right_room] = 2
        except KeyError:
            if self.__walls[right_room + c_room] is False:
                n_dict[right_room] = 2
    # look left
    if c_coordinate[1] != 0:
        left_room = self.maze[c_coordinate[0]][c_coordinate[1] - 1]
        try:
            if self.__walls[c_room + left_room] is False:
                n_dict[left_room] = 2
        except KeyError:
            if self.__walls[left_room + c_room] is False:
                n_dict[left_room] = 2
    return n_dict
```

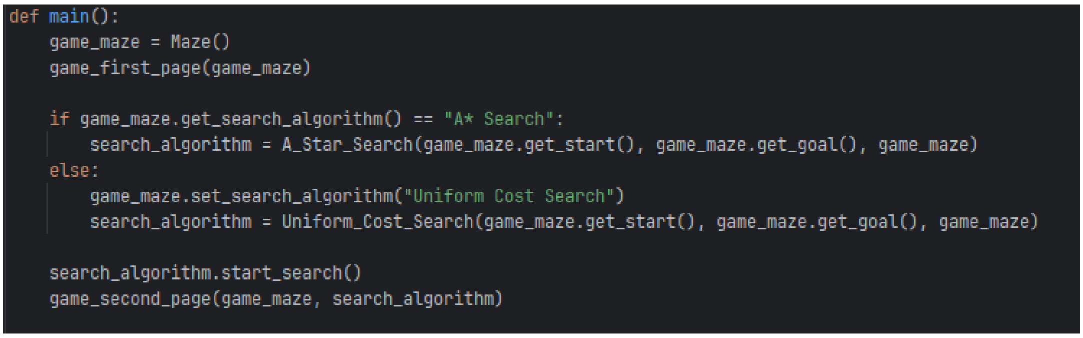
Requirement 5:

- The user will choose one of the search strategies: uniform cost and A* search (use Hamming distance as heuristics).

Selection information is got from the user and the “search algorithm” attribute is set considering this information. In the main method, object of the search algorithm is created by the information from “maze” class.



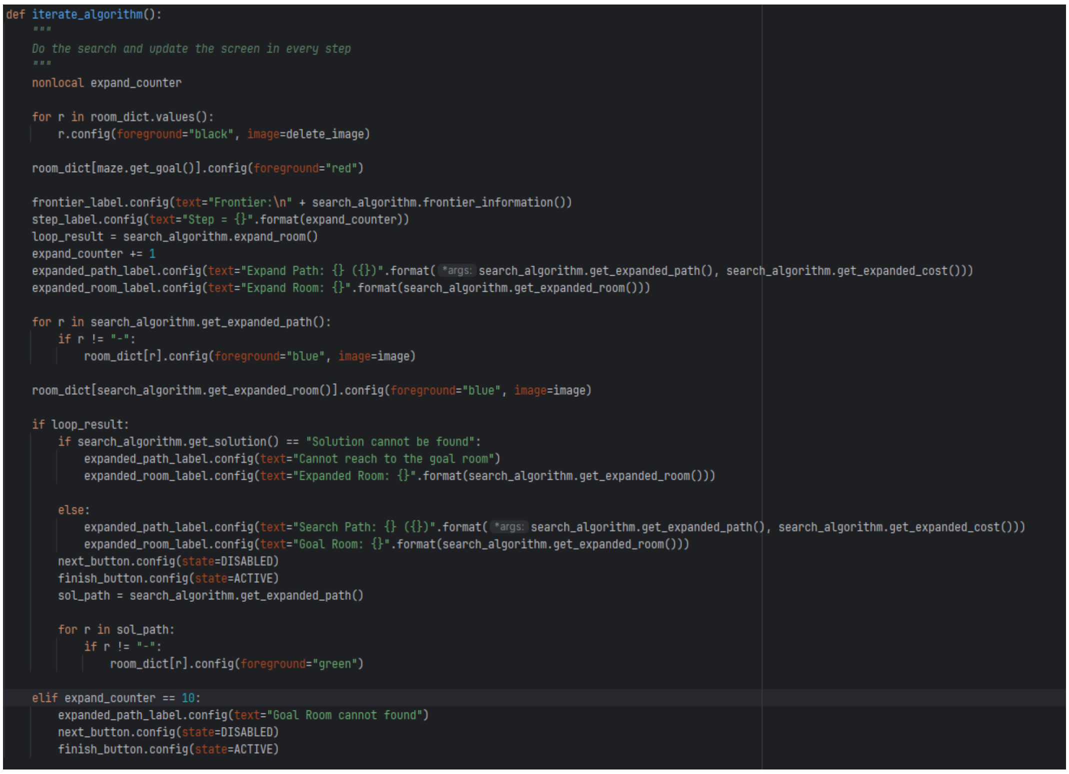
```
def initialize_algorithm():
    """Set the search algorithm of the game maze"""
    if algorithm_var.get() == 0:
        maze.set_search_algorithm("Uniform Cost Search")
    else:
        maze.set_search_algorithm("A* Search")
```



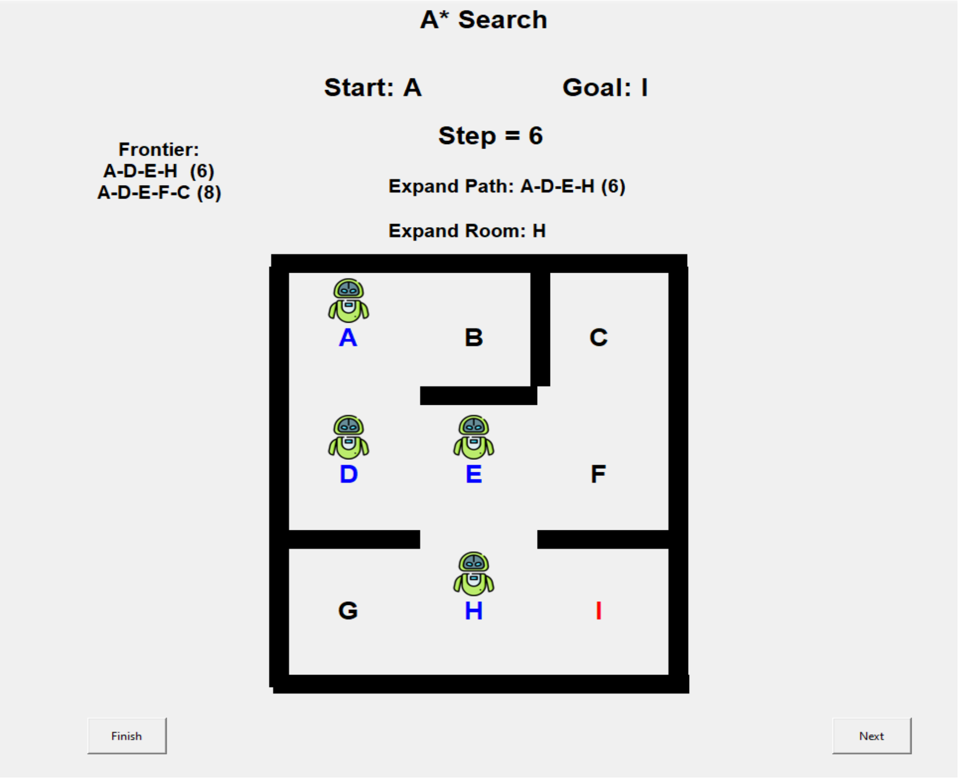
```
def main():
    game_maze = Maze()
    game_first_page(game_maze)

    if game_maze.get_search_algorithm() == "A* Search":
        search_algorithm = A_Star_Search(game_maze.get_start(), game_maze.get_goal(), game_maze)
    else:
        game_maze.set_search_algorithm("Uniform Cost Search")
        search_algorithm = Uniform_Cost_Search(game_maze.get_start(), game_maze.get_goal(), game_maze)

    search_algorithm.start_search()
    game_second_page(game_maze, search_algorithm)
```

Requirement 6:

- The searching will go on till the 10th expanded node. The program will print out each expanded state and compare it with the given goal state.



```
def iterate_algorithm():
    """
    Do the search and update the screen in every step
    """
    nonlocal expand_counter

    for r in room_dict.values():
        r.config(foreground="black", image=delete_image)

    room_dict[maze.get_goal()].config(foreground="red")

    frontier_label.config(text="Frontier:\n" + search_algorithm.frontier_information())
    step_label.config(text="Step = {}".format(expand_counter))
    loop_result = search_algorithm.expand_room()
    expand_counter += 1
    expanded_path_label.config(text="Expand Path: {} ({})".format( *args: search_algorithm.get_expanded_path(), search_algorithm.get_expanded_cost()))
    expanded_room_label.config(text="Expand Room: {}".format(search_algorithm.get_expanded_room()))

    for r in search_algorithm.get_expanded_path():
        if r != "-":
            room_dict[r].config(foreground="blue", image=image)

    room_dict[search_algorithm.get_expanded_room()].config(foreground="blue", image=image)

    if loop_result:
        if search_algorithm.get_solution() == "Solution cannot be found":
            expanded_path_label.config(text="Cannot reach to the goal room")
            expanded_room_label.config(text="Expanded Room: {}".format(search_algorithm.get_expanded_room()))

        else:
            expanded_path_label.config(text="Search Path: {} ({})".format( *args: search_algorithm.get_expanded_path(), search_algorithm.get_expanded_cost()))
            expanded_room_label.config(text="Goal Room: {}".format(search_algorithm.get_expanded_room()))
        next_button.config(state=DISABLED)
        finish_button.config(state=ACTIVE)
        sol_path = search_algorithm.get_expanded_path()

        for r in sol_path:
            if r != "-":
                room_dict[r].config(foreground="green")

    elif expand_counter == 10:
        expanded_path_label.config(text="Goal Room cannot found")
        next_button.config(state=DISABLED)
        finish_button.config(state=ACTIVE)
```



A* Search

Start: A

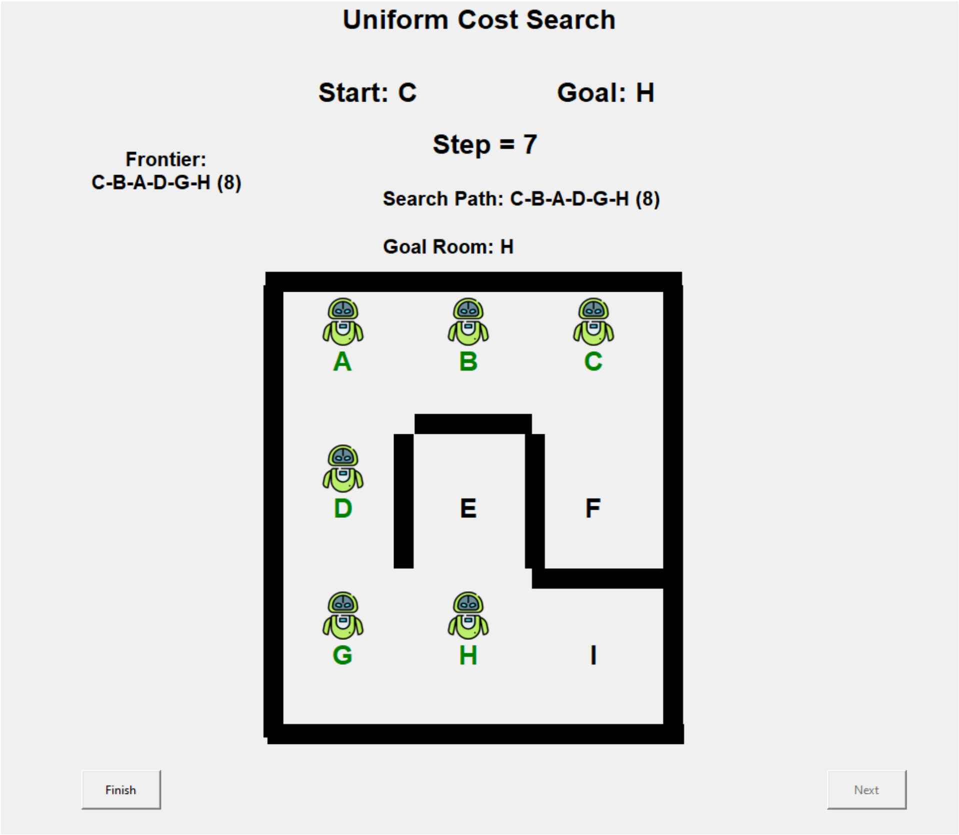
Goal: I

Step = 6

Frontier:
A-D-E-H (6)
A-D-E-F-C (8)

Expand Path: A-D-E-H (6)

Expand Room: H



Uniform Cost Search

Start: C

Goal: H

Step = 7

Frontier:
C-B-A-D-G-H (8)

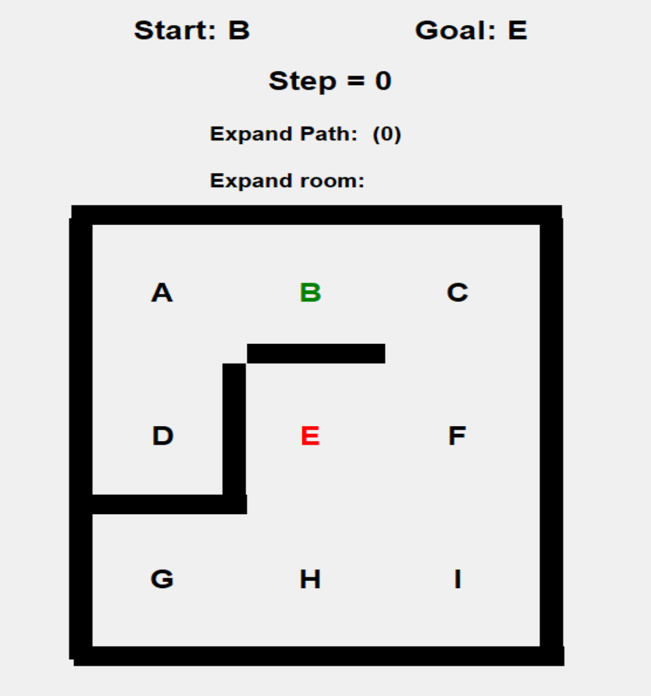
Search Path: C-B-A-D-G-H (8)

Goal Room: H

Algorithm Execution

Uniform Cost Search

Step 1:



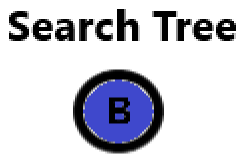
Start: B Goal: E

Step = 0

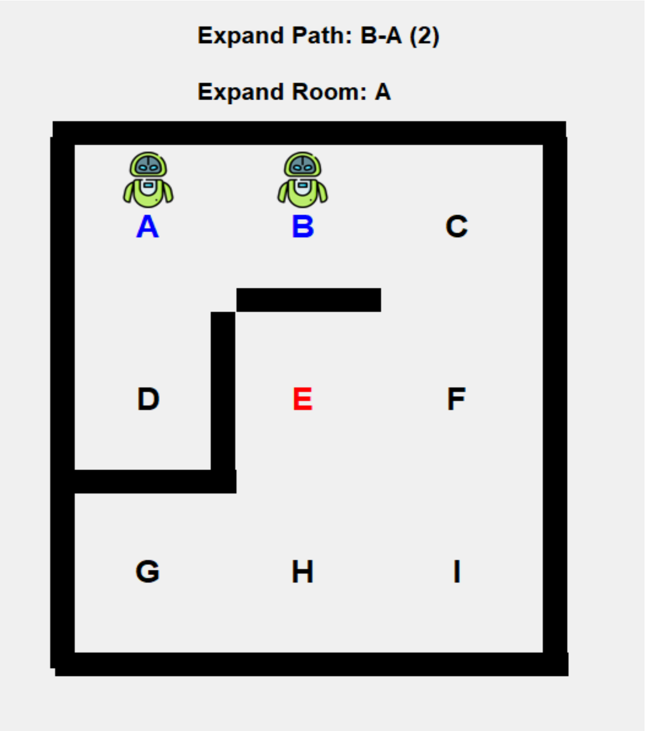
Expand Path: (0)

Expand room:

Frontier	Expand
B (0)	B



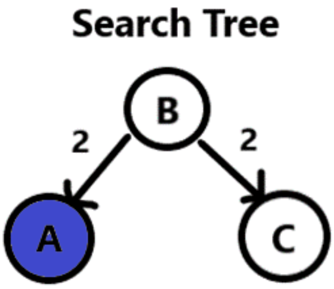
Step 2:



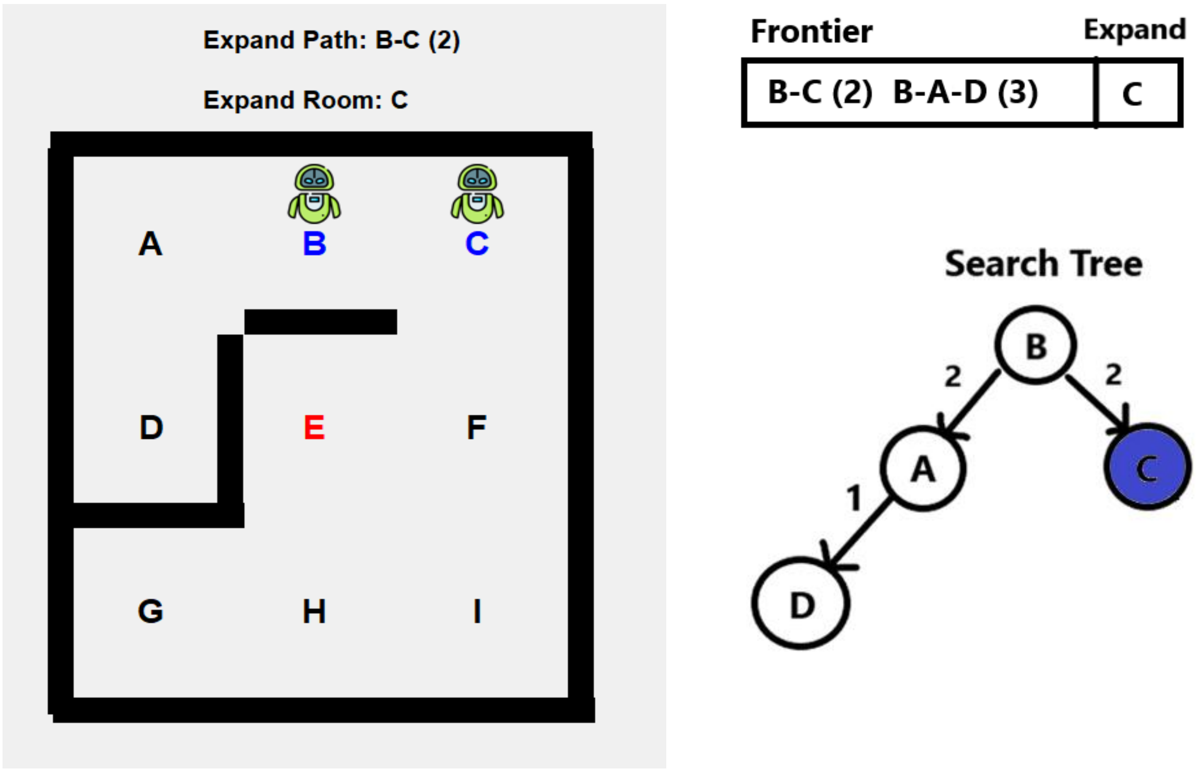
Expand Path: B-A (2)

Expand Room: A

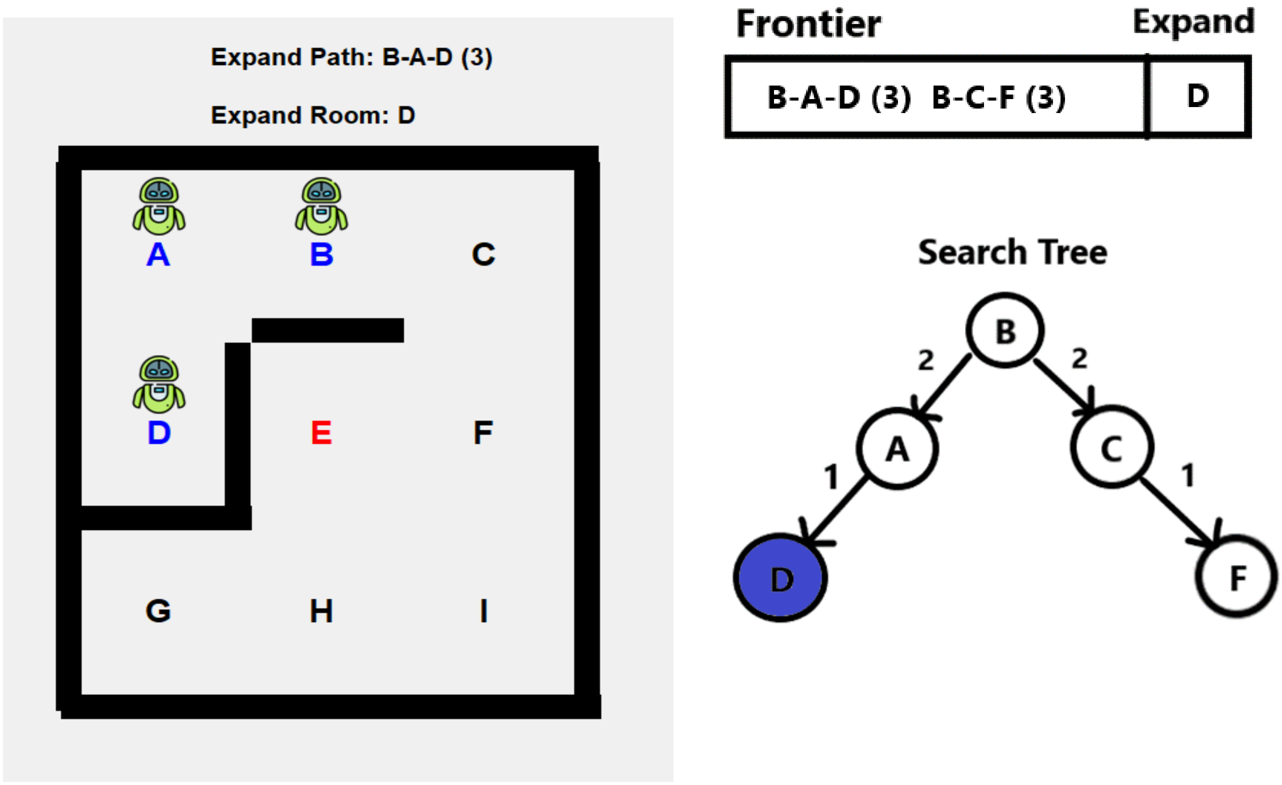
Frontier	Expand
B-A (2) B-C(2)	A



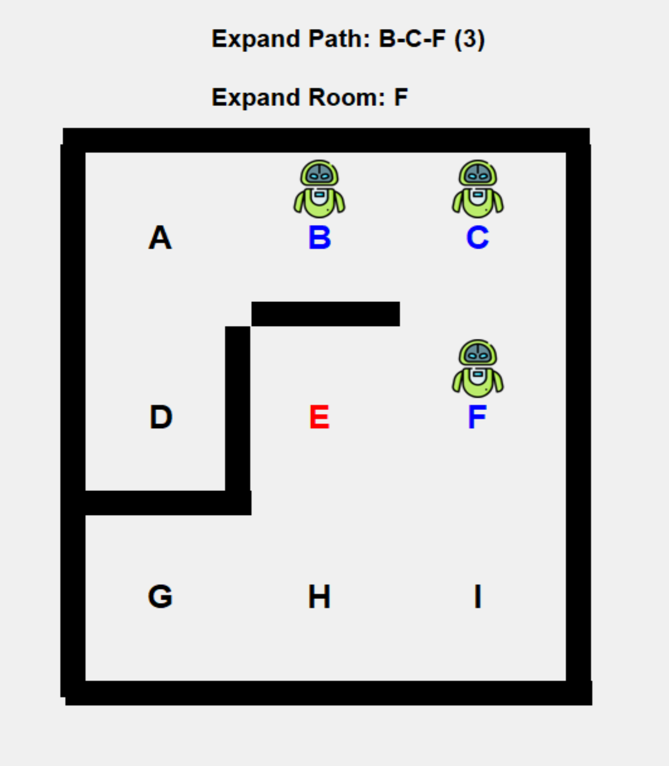
Step 3:


Expand Path: B-C (2)
Expand Room: C
A
B
C
D
E
F
G
H
I
Frontier
Expand
B-C (2) B-A-D (3)
C
Search Tree
B
2
2
A
C
1
D

Step 4:


Expand Path: B-A-D (3)
Expand Room: D
A
B
C
D
E
F
G
H
I
Frontier
Expand
B-A-D (3) B-C-F (3)
D
Search Tree
B
2
2
A
C
1
1
D
F

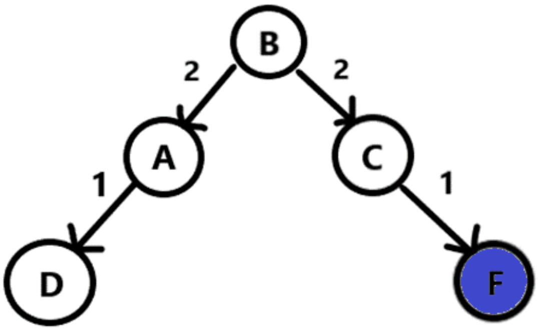
Step 5:


Expand Path: B-C-F (3)

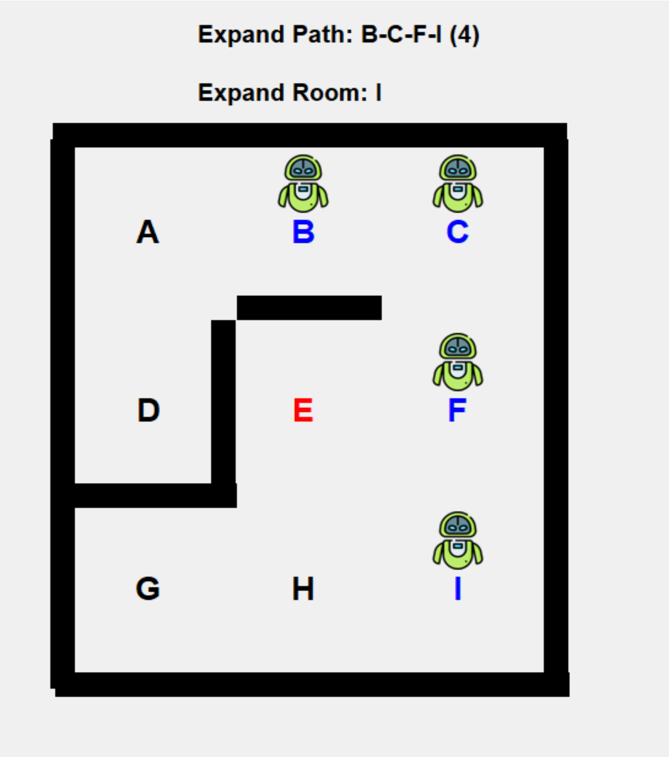
Expand Room: F

Frontier	Expand
B-C-F (3)	F

Search Tree



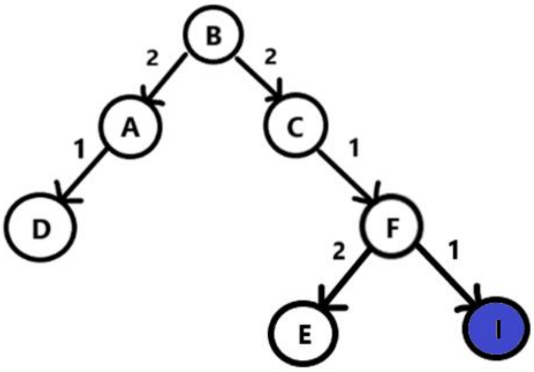
Step 6:


Expand Path: B-C-F-I (4)

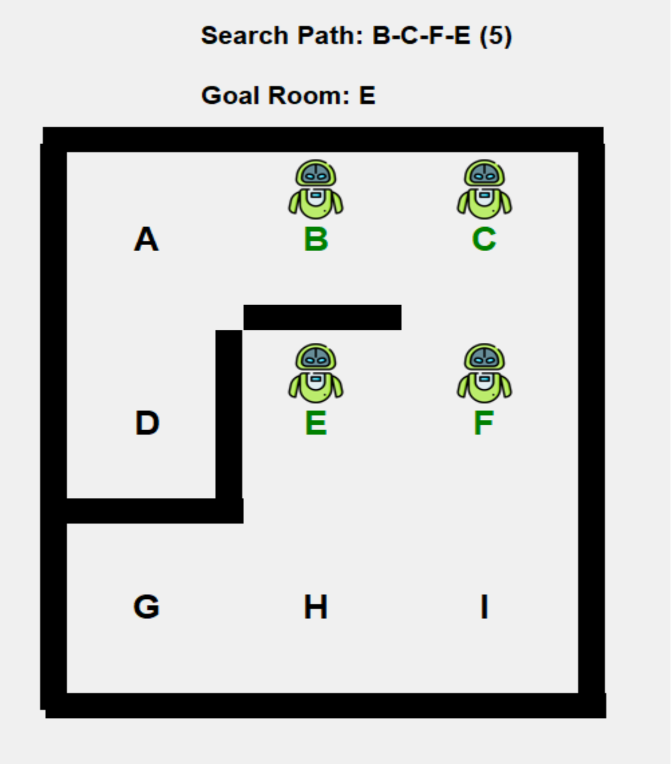
Expand Room: I

Frontier	Expand
B-C-F-I (4) B-C-F-E (5)	I

Search Tree



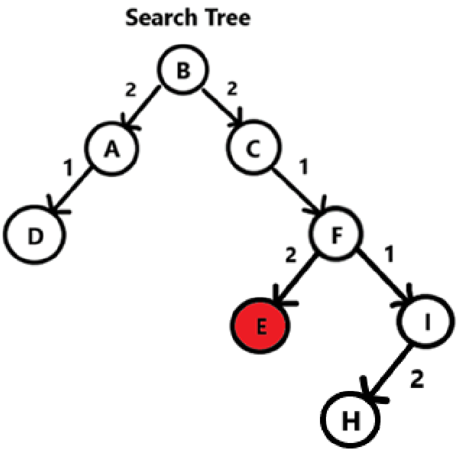
Step 7:



Search Path: B-C-F-E (5)

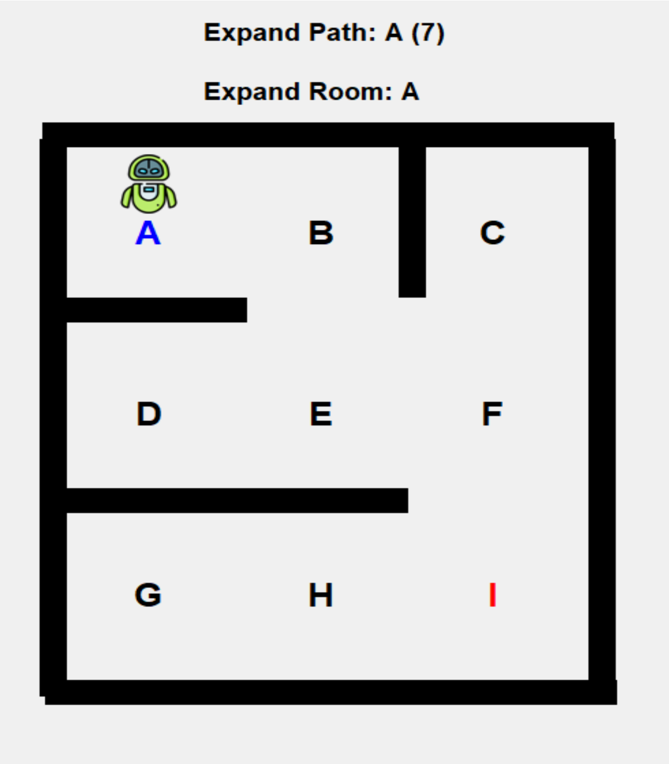
Goal Room: E

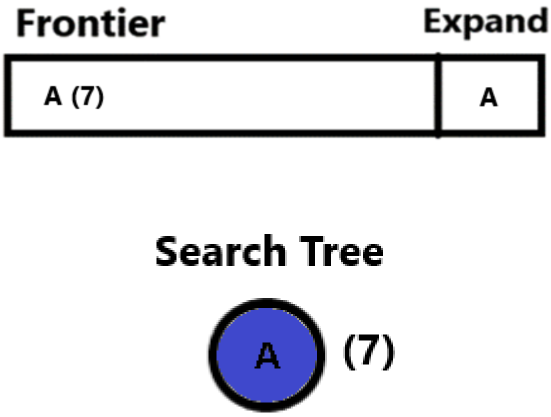
Frontier	Expand
B-C-F-E (5) B-C-F-I-H (6)	E



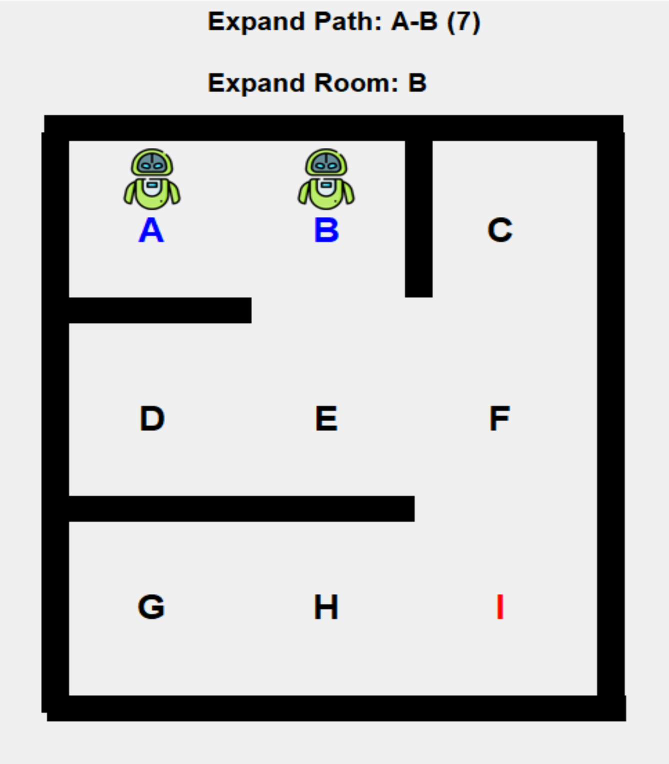
A* Search

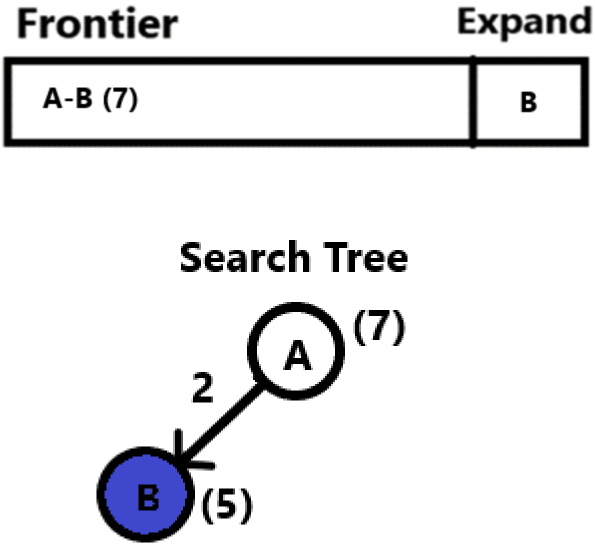
Step 1:





Step 2:

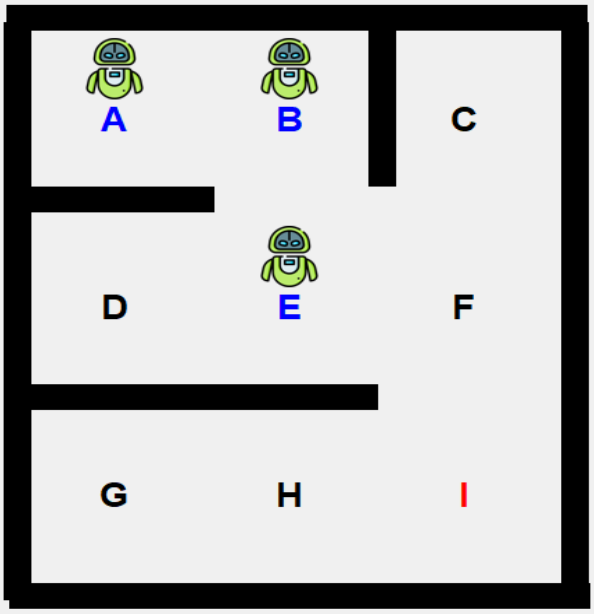




Step 3:

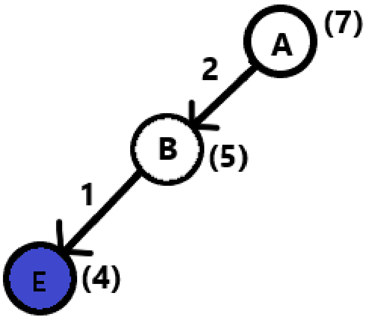
Expand Path: A-B-E (7)

Expand Room: E



Frontier	Expand
A-B-E (7)	E

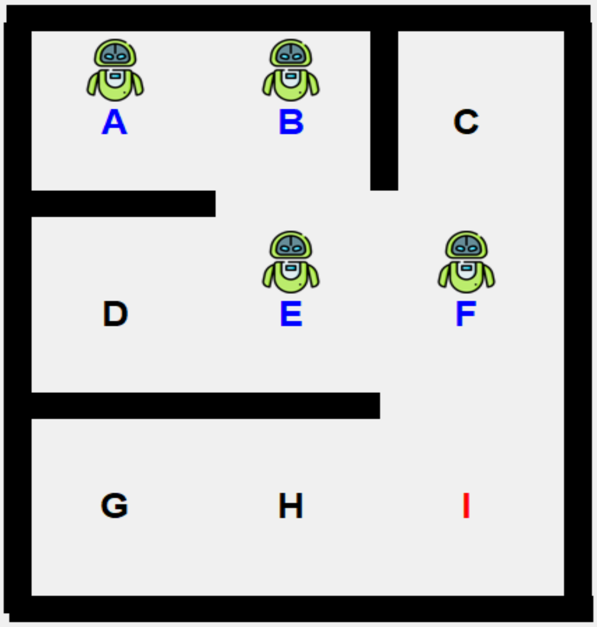
Search Tree



Step 4:

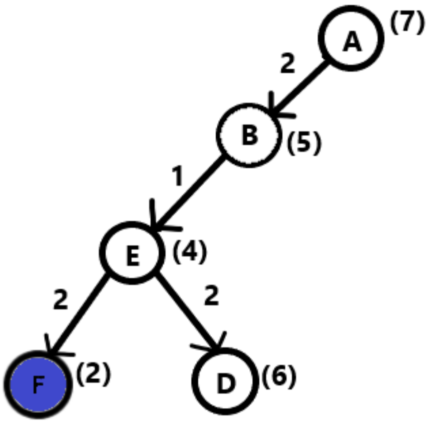
Expand Path: A-B-E-F (7)

Expand Room: F

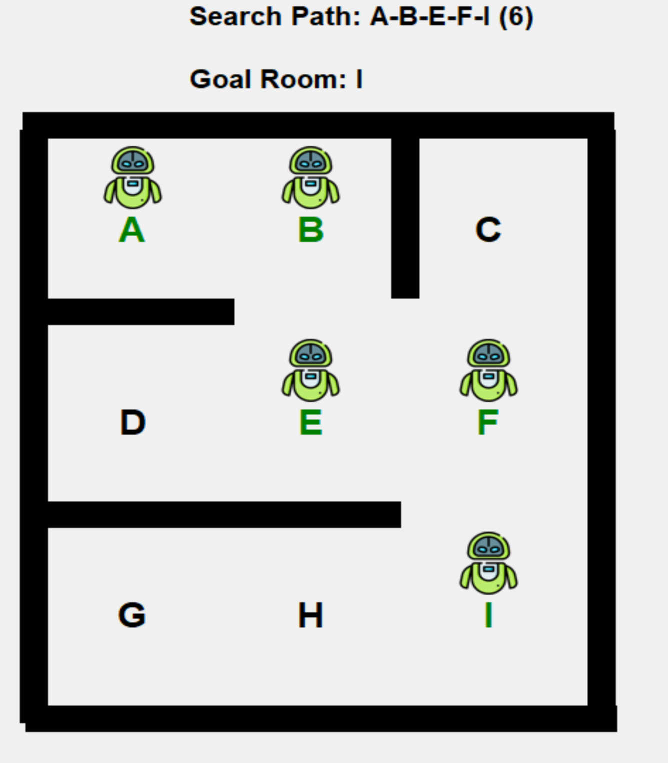


Frontier	Expand
A-B-E-F (7) A-B-E-D (11)	F

Search Tree



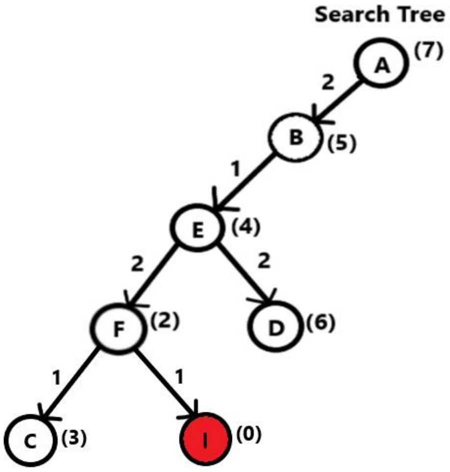
Step 5:



Search Path: A-B-E-F-I (6)

Goal Room: I

Frontier	Expand
A-B-E-F-I (6) A-B-E-F-C (9) A-B-E-D (11)	I



Search Tree
A
(7)
2
B
(5)
1
E
(4)
2
2
F
(2)
D
(6)
1
1
C
(3)
I
(0)