Faculty of Computing & Information Technology
INDUS UNIVERSITY

“Artificial Intelligence (Lab)” (0+1)

Project Report

Title:
“Searches Using Several AI Algorithms”

Student Name	Student ID
Abdul Basit Khan	2551 – 2018
Mubashir Qamar	2356 – 2018

Submitted to: Sir Abid Ali

ACKNOWLEDGMENT

We are really grateful because we managed to complete our E-project within the time given by our teacher **Sir Abid Ali**. This assignment cannot be completed without the effort and co-operation from our group members, Group members; **Abdul Basit Khan** and **Mubashir Qamar.**

We also sincerely thank our teacher **Sir Abid Ali** for the guidance and encouragement in finishing this project and also for teaching us in this course.

Last but not the least, we would like to express our gratitude to our friends and respondents for the support and willingness directly or indirectly to spend some times with us to fill in the questionnaires.

Table of Contents

1- Introduction

In Artificial Intelligence, Search techniques are universal problem-solving methods. Rational agents or Problem-solving agents in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result. Problem-solving agents are the goal-based agents and use atomic representation.

Searching problems can be solved using trees and graphs and to be be specific, trees and graphs works differently. For graphs, they are considered to be more accurate in searches but they can either be directed or undirected. Directed means that they contain a specific direction for the search flow whereas undirected does not have any direction which means the graph can go either way.

Generally, searches are of two types; the uninformed search (blind search) and the informed search. These searches works differently depending on the method chosen like blind search with Breadth-first search will work differently as compared to informed search with Depth-first search. Uninformed search will take a bit more time to reach to the goal node compared to the informed search as it takes a fairly lesser time but if we have opted for A* algorithm then we cannot say much on the time taken as these methods involves weights of the edge and the heuristics associated with them so time can vary.

There are two categories for graphs to be drawn:

- The Directed Search
- The Undirected Search

The two kinds of searches are as follows:

- The Informed Search
- The Uninformed Search (Blind Search)

Search techniques inside uninformed search are:

- Breadth-First Search
- Depth-First Search
- Depth Limit Search
- Iterative Deepening Search
- Uniform Cost Search

Search techniques inside informed search are:

- Greedy Search
- A* Search

2- Objectives of the Project

The objective of this project is to make a problem-solving agent that can perform various searches using some AI techniques. Problem-solving, at times, can be difficult because the user will be willing to get as much efficiency as he can get, therefore this model will be providing us with several search techniques based on different kinds of problem.

Moreover, this project is useful to study different AI algorithms and to have a good hand on them as it enables us to choose nodes (initial node or the goal node) by ourselves in order to recognize how searches are being performed based on different techniques.

Furthermore, the goal is to have an understanding on the problems which involves several heuristics as they are the complex ones, but this agent is actually useful here as it provides us the way to choose the branching factor, heuristic value and the edges by ourselves so that we can find a way about how the searches are actually being done.

3- Topic of our Project

The topic of our project is “Searches using several AI Algorithms”. Basically, this project is totally about searches and different techniques to be used in searches.

4- Problem Statement

- Searching problems are becoming more complex
- No such problem-solving agent to deal with problems involving heuristics
- Fewer AI models with uninformed search techniques as they consume some time to get to the solution

5- Hardware/Software Requirements

5.1- Hardware Requirements

- A Computer System
- 2 GB RAM or above

5.2- Software Requirements

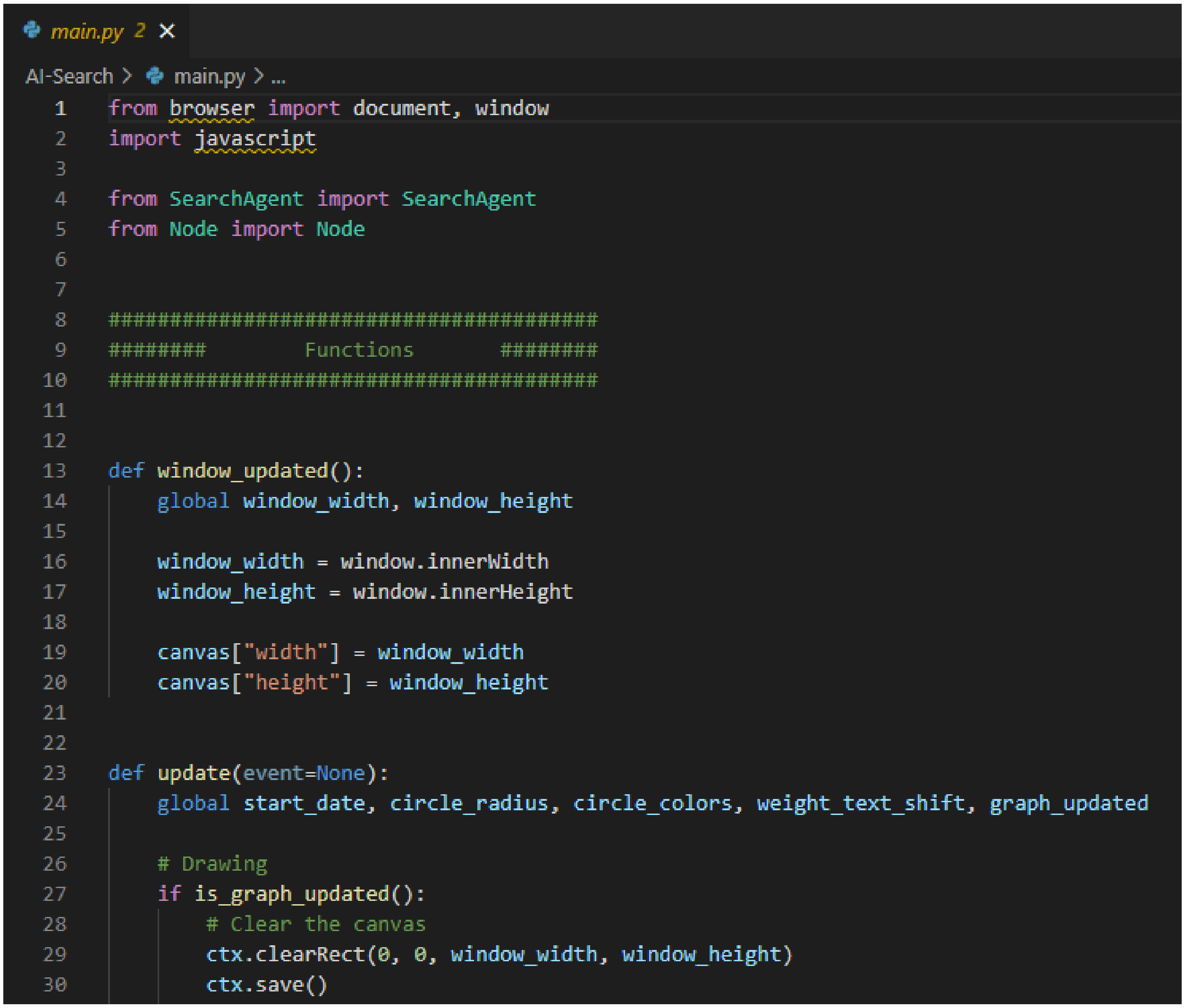
- Visual Studio Code (Extensions: live server, code runner and python)
- Any Browser (Google Chrome and Microsoft Edge etc.)

6- Work Analysis

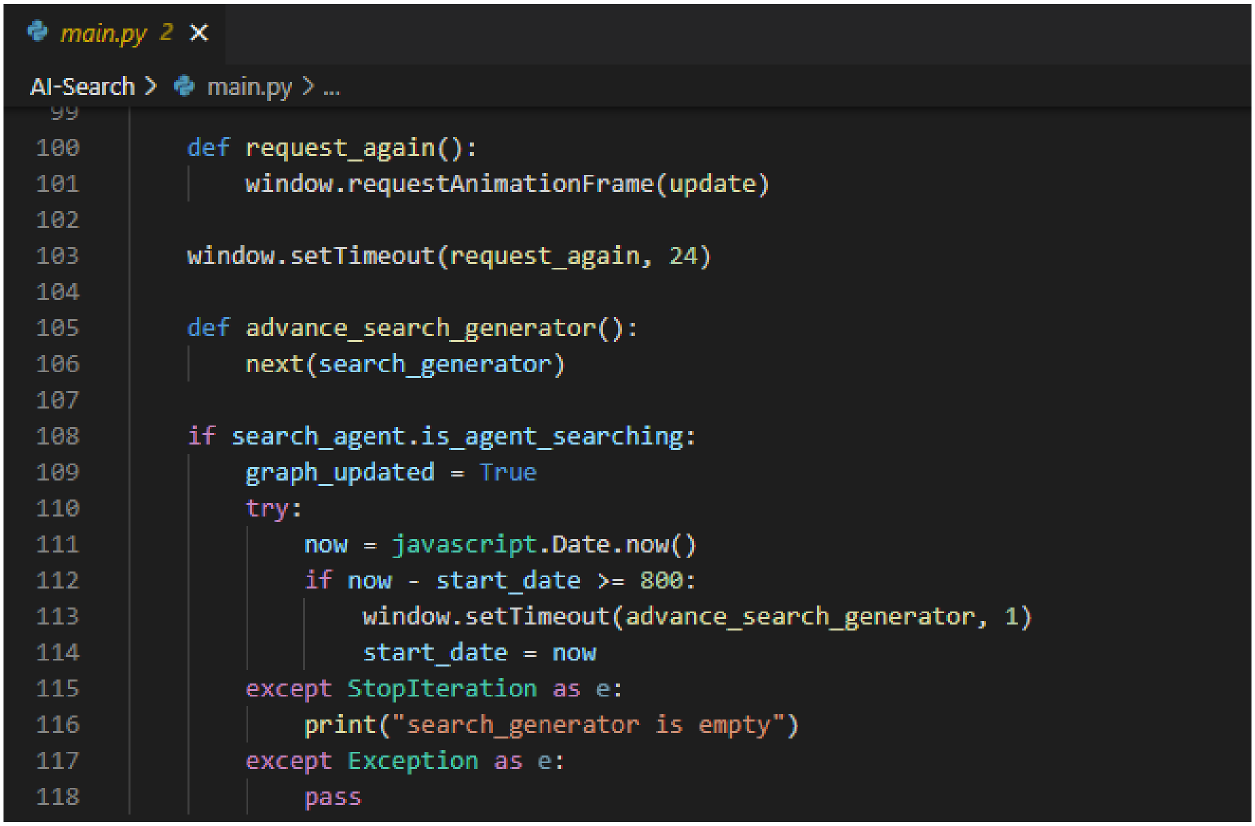
Tasks	Abdul Basit Khan	Mubashir Qamar
Analysis	✓	✓
Design	✓	✓
Coding	✓	✗
Testing	✗	✓
Documentation	✓	✓

7- Code Snippets

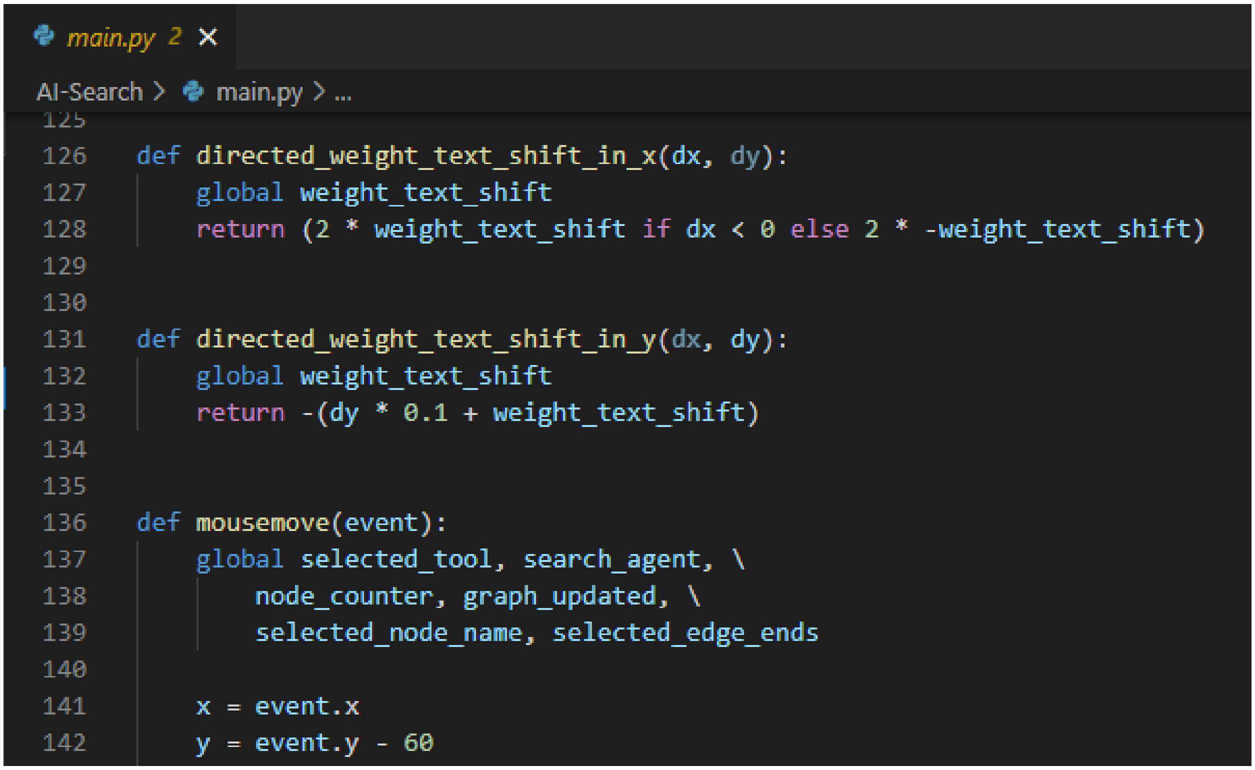
7.1- Main.py



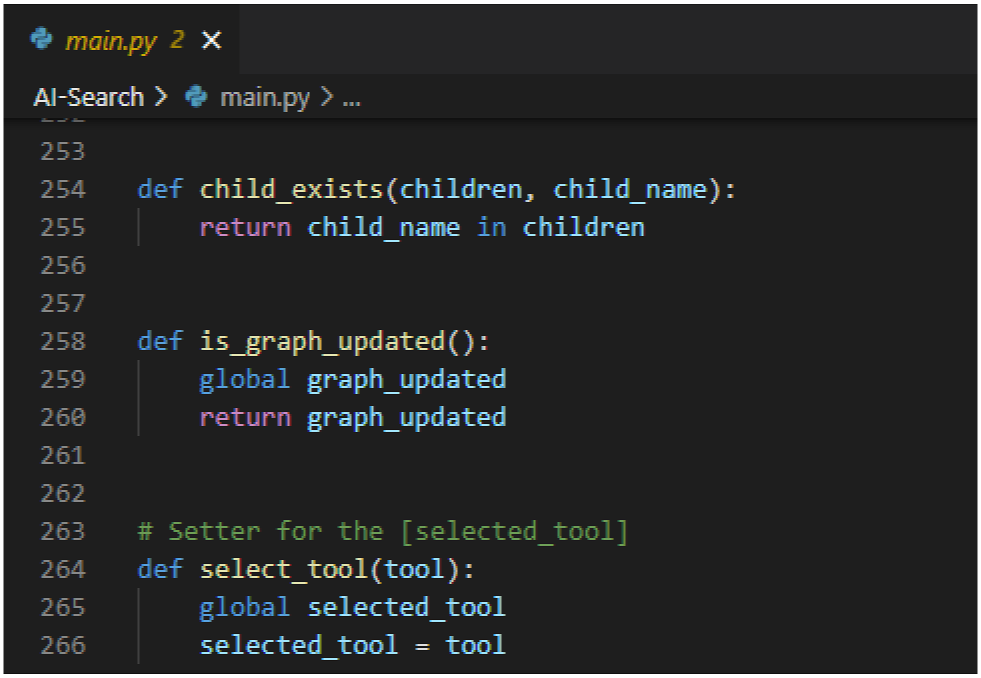
```python
from browser import document, window
import javascript

from SearchAgent import SearchAgent
from Node import Node


########################################
########        Functions       ########
########################################


def window_updated():
    global window_width, window_height

    window_width = window.innerWidth
    window_height = window.innerHeight

    canvas["width"] = window_width
    canvas["height"] = window_height


def update(event=None):
    global start_date, circle_radius, circle_colors, weight_text_shift, graph_updated

    # Drawing
    if is_graph_updated():
        # Clear the canvas
        ctx.clearRect(0, 0, window_width, window_height)
        ctx.save()
```



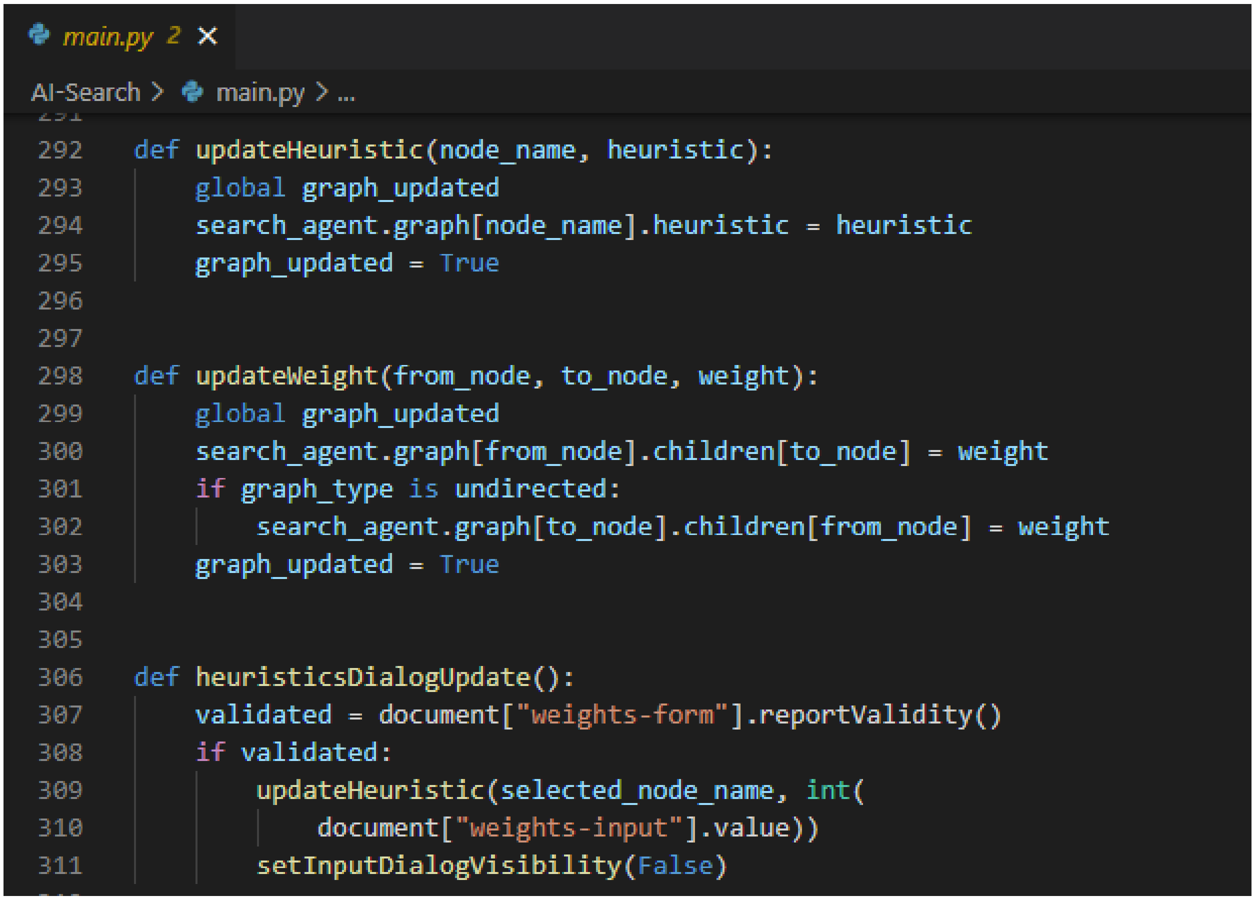
```python
    def request_again():
        window.requestAnimationFrame(update)

    window.setTimeout(request_again, 24)

    def advance_search_generator():
        next(search_generator)

    if search_agent.is_agent_searching:
        graph_updated = True
        try:
            now = javascript.Date.now()
            if now - start_date >= 800:
                window.setTimeout(advance_search_generator, 1)
                start_date = now
        except StopIteration as e:
            print("search_generator is empty")
        except Exception as e:
            pass
```



```python
def directed_weight_text_shift_in_x(dx, dy):
    global weight_text_shift
    return (2 * weight_text_shift if dx < 0 else 2 * -weight_text_shift)


def directed_weight_text_shift_in_y(dx, dy):
    global weight_text_shift
    return -(dy * 0.1 + weight_text_shift)


def mousemove(event):
    global selected_tool, search_agent, \
        node_counter, graph_updated, \
        selected_node_name, selected_edge_ends

    x = event.x
    y = event.y - 60
```

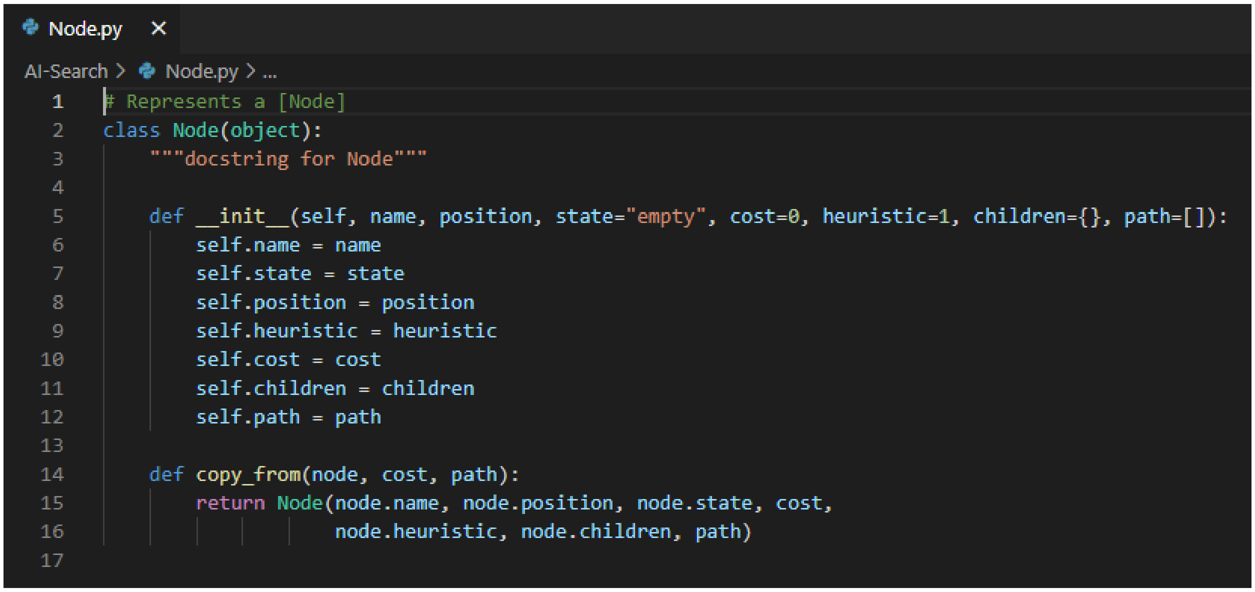


```python

def child_exists(children, child_name):
    return child_name in children


def is_graph_updated():
    global graph_updated
    return graph_updated


# Setter for the [selected_tool]
def select_tool(tool):
    global selected_tool
    selected_tool = tool
```



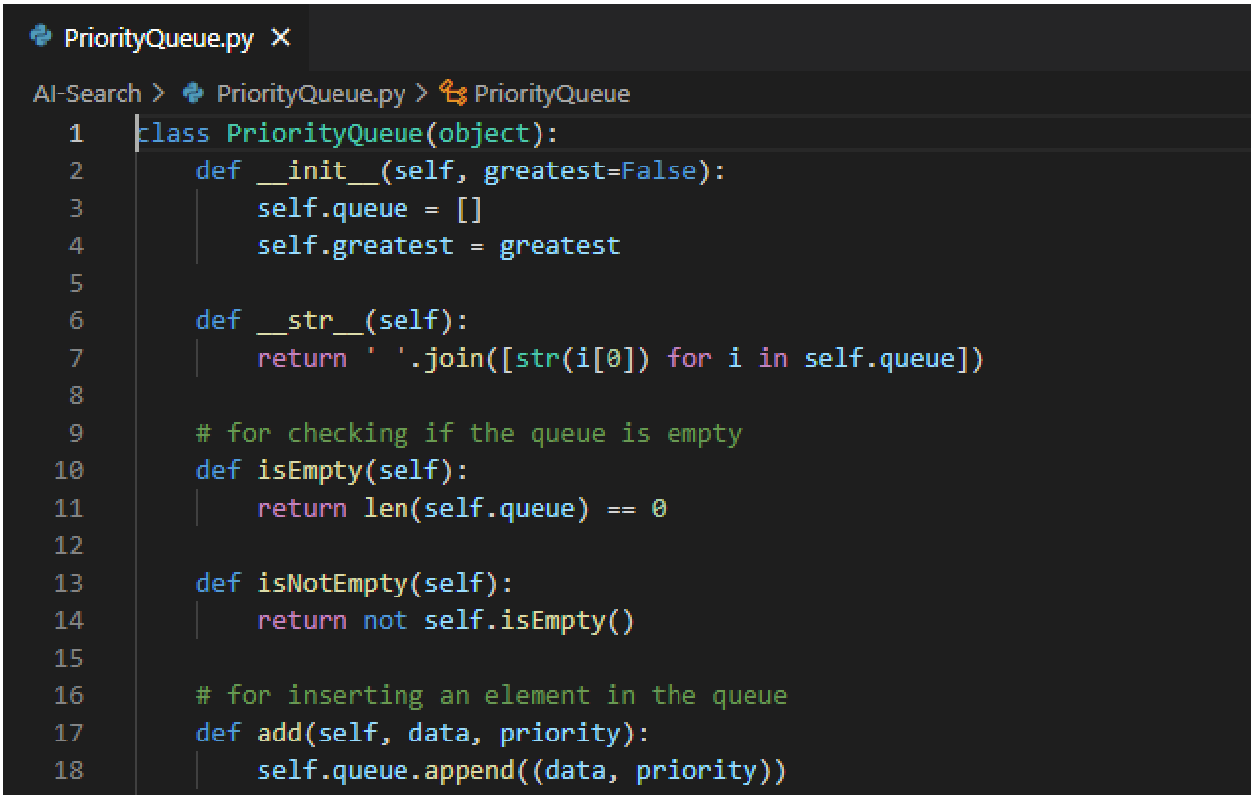
```python
def updateHeuristic(node_name, heuristic):
    global graph_updated
    search_agent.graph[node_name].heuristic = heuristic
    graph_updated = True


def updateWeight(from_node, to_node, weight):
    global graph_updated
    search_agent.graph[from_node].children[to_node] = weight
    if graph_type is undirected:
        search_agent.graph[to_node].children[from_node] = weight
    graph_updated = True


def heuristicsDialogUpdate():
    validated = document["weights-form"].reportValidity()
    if validated:
        updateHeuristic(selected_node_name, int(
            document["weights-input"].value))
        setInputDialogVisibility(False)
```

7.2- Node.py

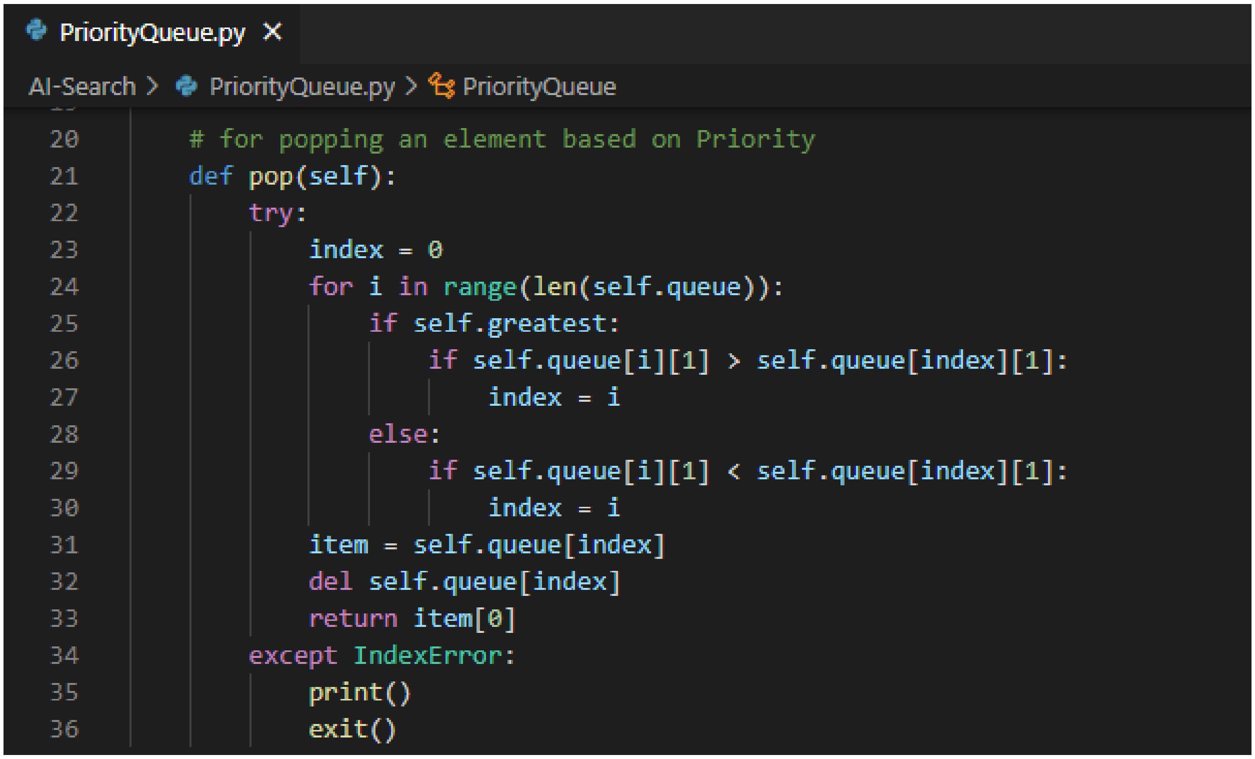


```python
# Represents a [Node]
class Node(object):
    """docstring for Node"""

    def __init__(self, name, position, state="empty", cost=0, heuristic=1, children={}, path=[]):
        self.name = name
        self.state = state
        self.position = position
        self.heuristic = heuristic
        self.cost = cost
        self.children = children
        self.path = path

    def copy_from(node, cost, path):
        return Node(node.name, node.position, node.state, cost,
                    node.heuristic, node.children, path)

```

7.3- PriorityQueue.py

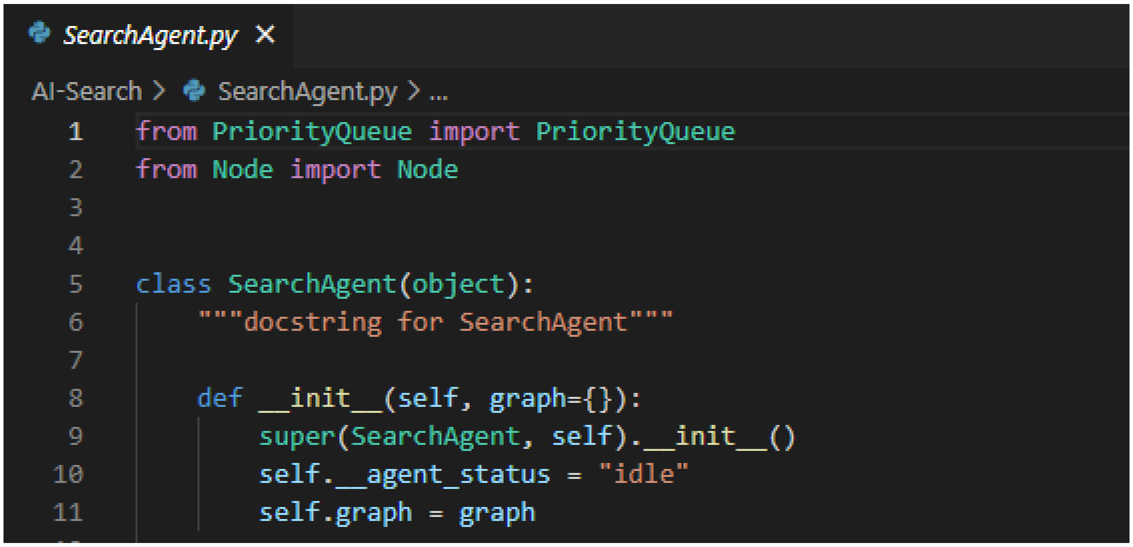


```python
class PriorityQueue(object):
    def __init__(self, greatest=False):
        self.queue = []
        self.greatest = greatest

    def __str__(self):
        return ' '.join([str(i[0]) for i in self.queue])

    # for checking if the queue is empty
    def isEmpty(self):
        return len(self.queue) == 0

    def isNotEmpty(self):
        return not self.isEmpty()

    # for inserting an element in the queue
    def add(self, data, priority):
        self.queue.append((data, priority))
```

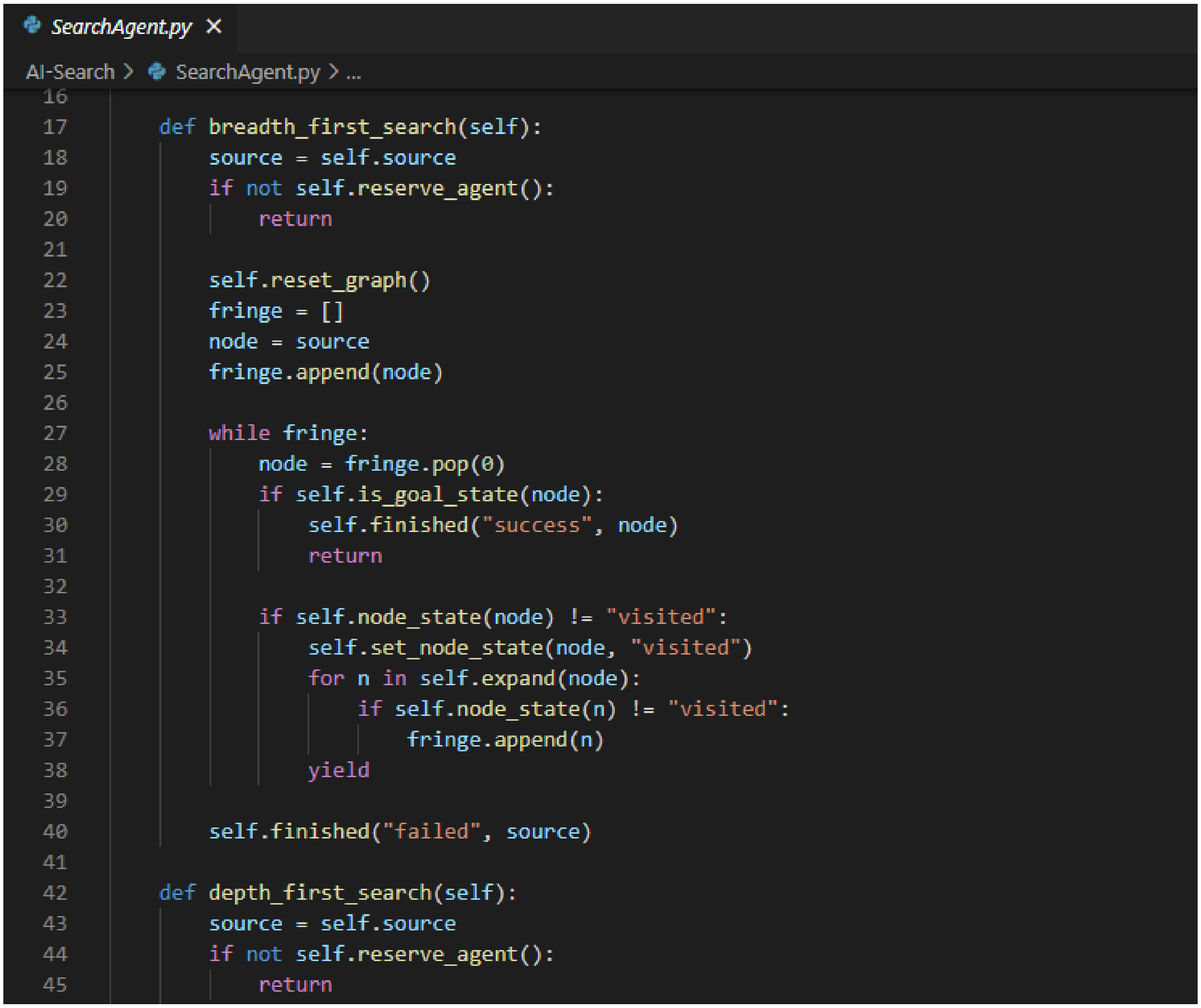


```python
    # for popping an element based on Priority
    def pop(self):
        try:
            index = 0
            for i in range(len(self.queue)):
                if self.greatest:
                    if self.queue[i][1] > self.queue[index][1]:
                        index = i
                else:
                    if self.queue[i][1] < self.queue[index][1]:
                        index = i
            item = self.queue[index]
            del self.queue[index]
            return item[0]
        except IndexError:
            print()
            exit()
```

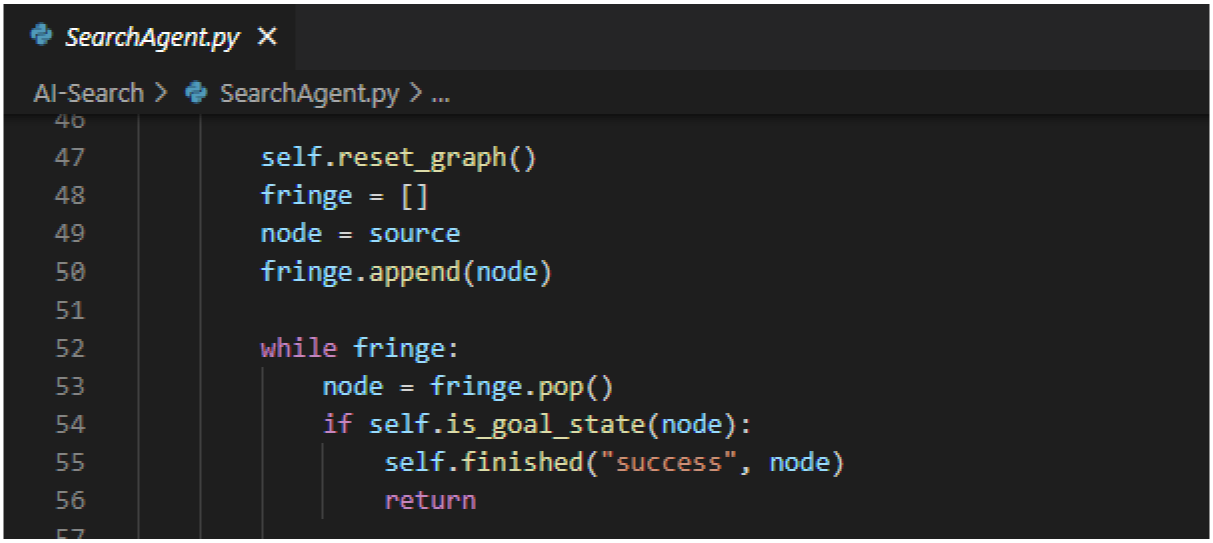
7.4- SearchAgent.py



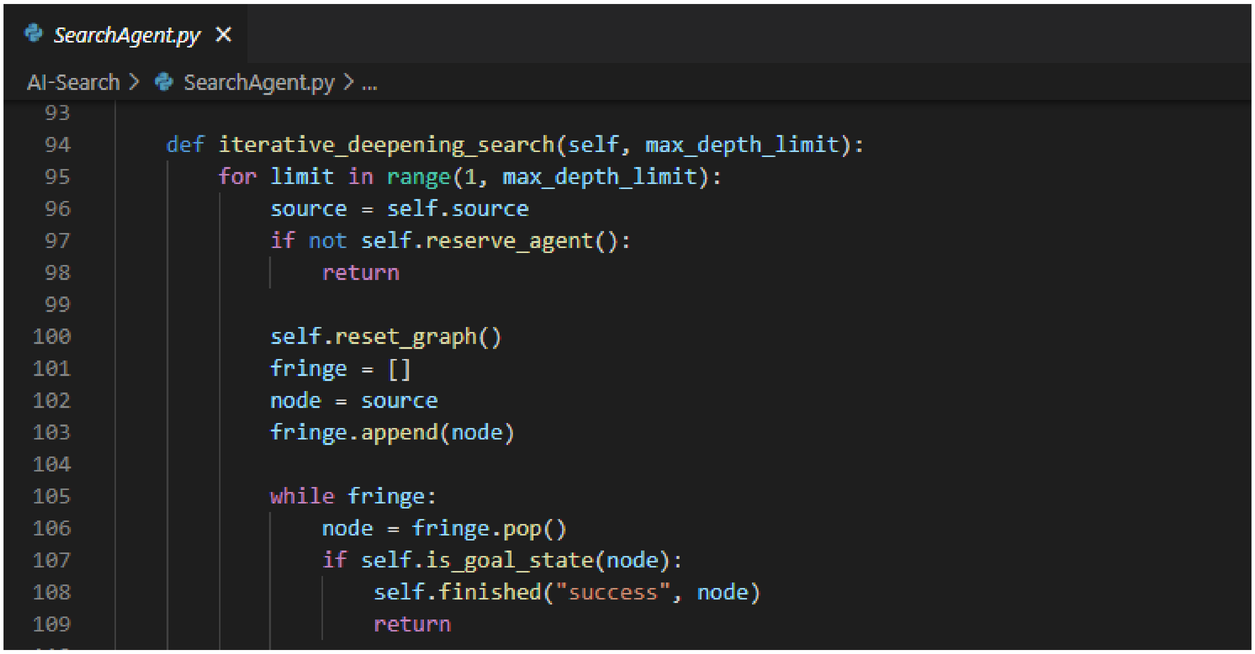
```python
from PriorityQueue import PriorityQueue
from Node import Node


class SearchAgent(object):
    """docstring for SearchAgent"""

    def __init__(self, graph={}):
        super(SearchAgent, self).__init__()
        self.__agent_status = "idle"
        self.graph = graph
```



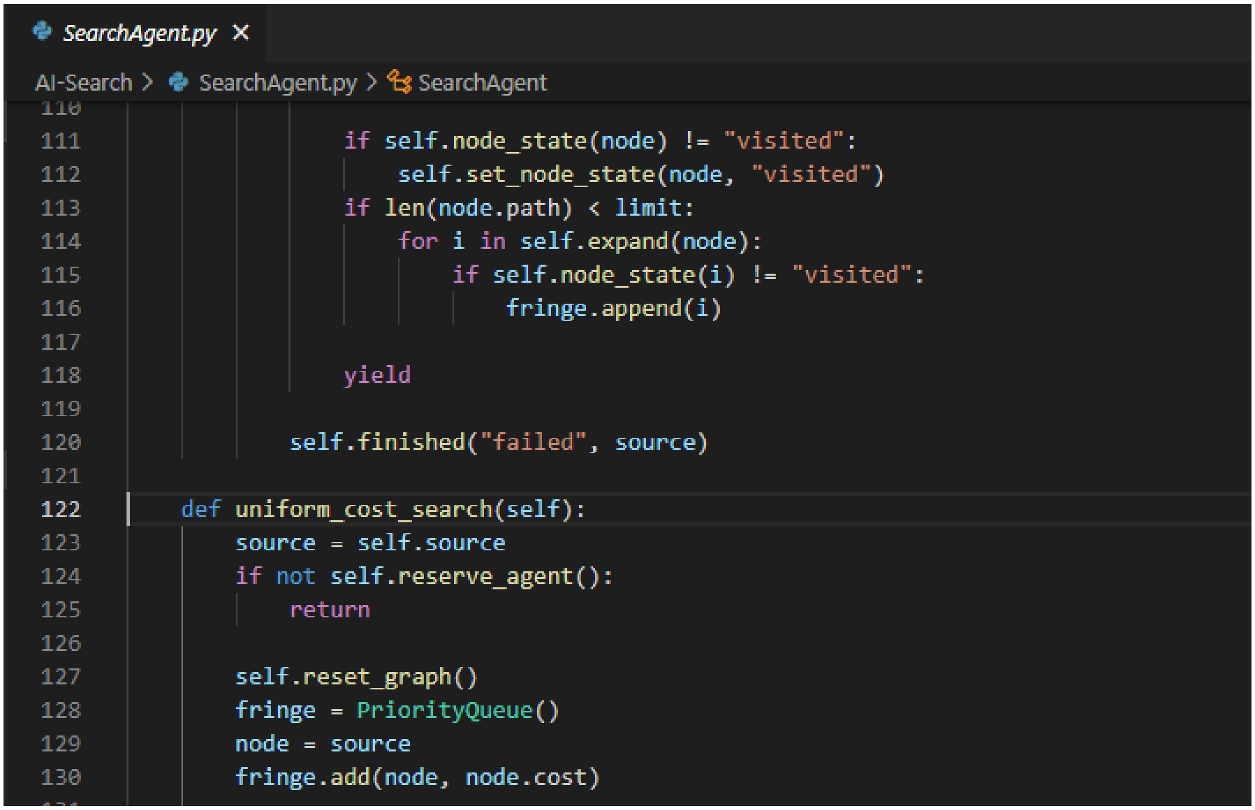
```
    def breadth_first_search(self):
        source = self.source
        if not self.reserve_agent():
            return

        self.reset_graph()
        fringe = []
        node = source
        fringe.append(node)

        while fringe:
            node = fringe.pop(0)
            if self.is_goal_state(node):
                self.finished("success", node)
                return

            if self.node_state(node) != "visited":
                self.set_node_state(node, "visited")
                for n in self.expand(node):
                    if self.node_state(n) != "visited":
                        fringe.append(n)
                yield

        self.finished("failed", source)

    def depth_first_search(self):
        source = self.source
        if not self.reserve_agent():
            return
```



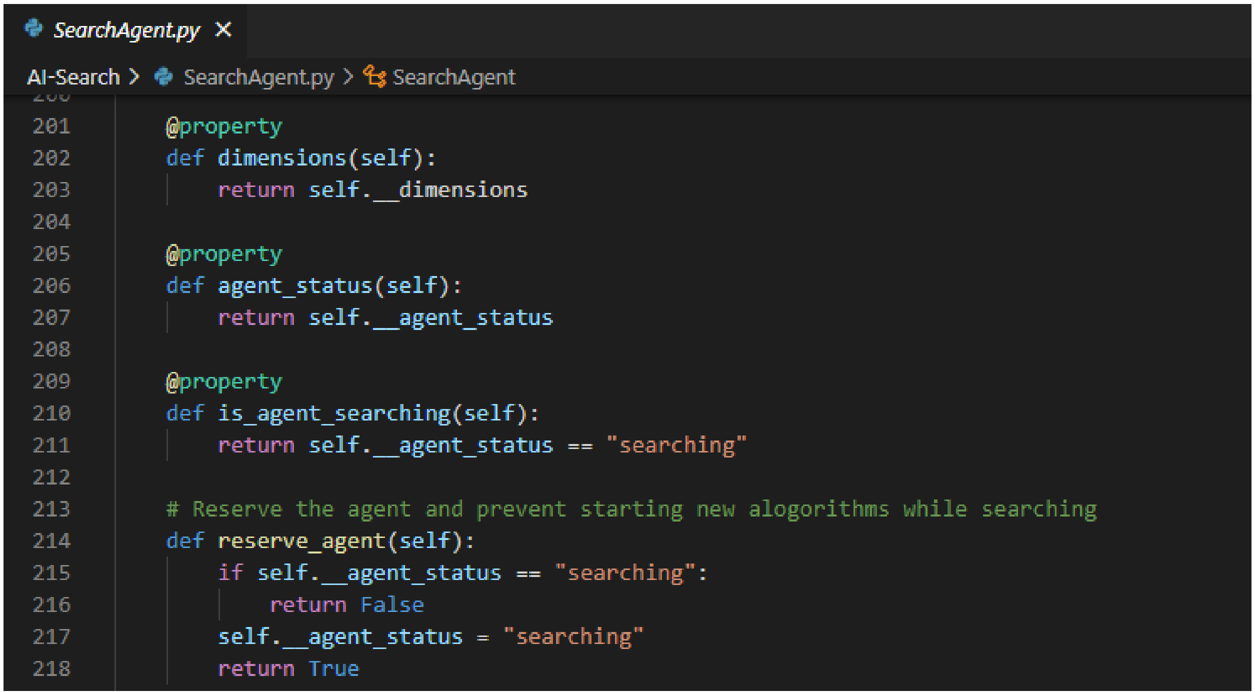
```
        self.reset_graph()
        fringe = []
        node = source
        fringe.append(node)

        while fringe:
            node = fringe.pop()
            if self.is_goal_state(node):
                self.finished("success", node)
                return
```



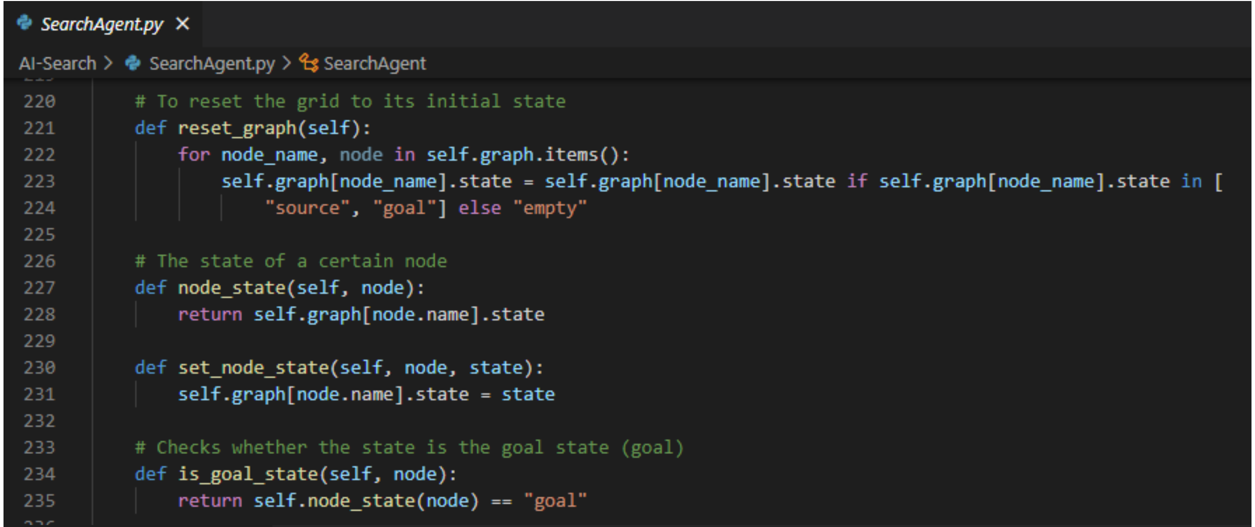
```python

    def iterative_deepening_search(self, max_depth_limit):
        for limit in range(1, max_depth_limit):
            source = self.source
            if not self.reserve_agent():
                return

            self.reset_graph()
            fringe = []
            node = source
            fringe.append(node)

            while fringe:
                node = fringe.pop()
                if self.is_goal_state(node):
                    self.finished("success", node)
                    return
```



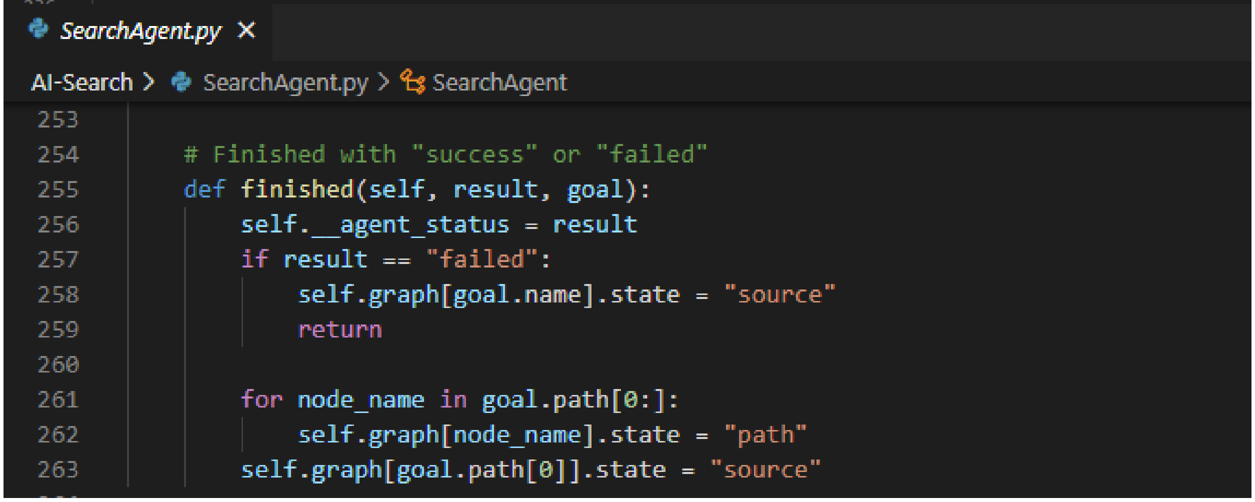
```python
                if self.node_state(node) != "visited":
                    self.set_node_state(node, "visited")
                if len(node.path) < limit:
                    for i in self.expand(node):
                        if self.node_state(i) != "visited":
                            fringe.append(i)

                yield

            self.finished("failed", source)

    def uniform_cost_search(self):
        source = self.source
        if not self.reserve_agent():
            return

        self.reset_graph()
        fringe = PriorityQueue()
        node = source
        fringe.add(node, node.cost)
```



```python
    @property
    def dimensions(self):
        return self.__dimensions

    @property
    def agent_status(self):
        return self.__agent_status

    @property
    def is_agent_searching(self):
        return self.__agent_status == "searching"

    # Reserve the agent and prevent starting new alogorithms while searching
    def reserve_agent(self):
        if self.__agent_status == "searching":
            return False
        self.__agent_status = "searching"
        return True
```

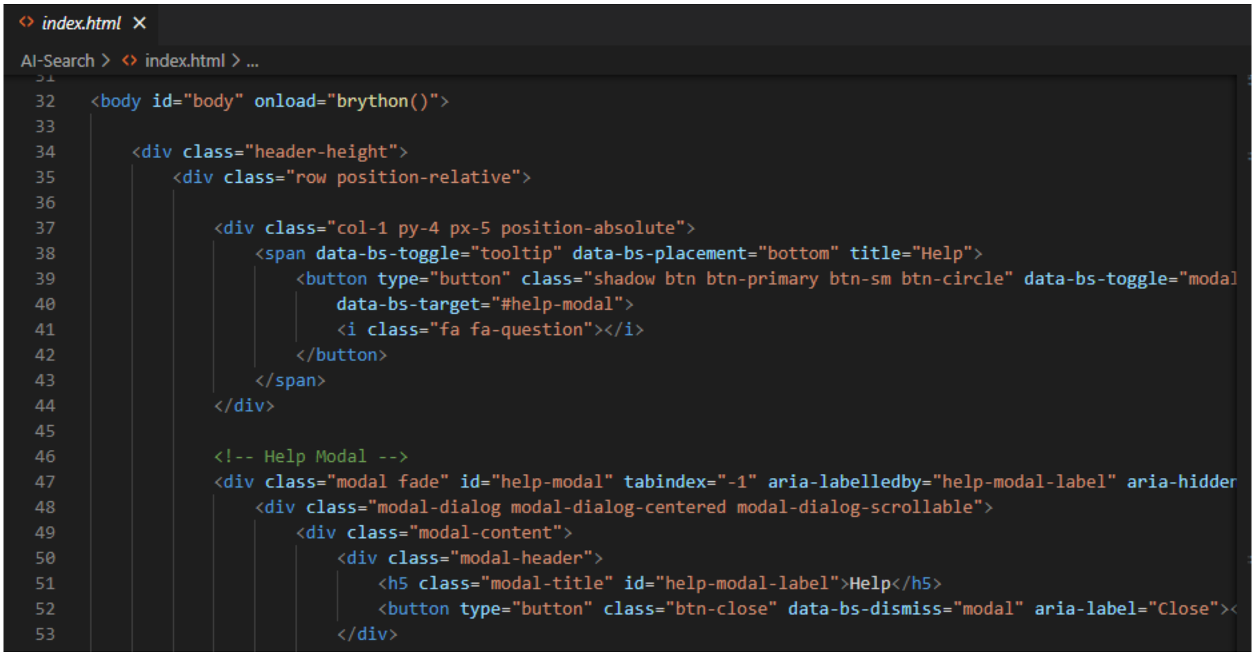


```python
    # To reset the grid to its initial state
    def reset_graph(self):
        for node_name, node in self.graph.items():
            self.graph[node_name].state = self.graph[node_name].state if self.graph[node_name].state in [
                "source", "goal"] else "empty"

    # The state of a certain node
    def node_state(self, node):
        return self.graph[node.name].state

    def set_node_state(self, node, state):
        self.graph[node.name].state = state

    # Checks whether the state is the goal state (goal)
    def is_goal_state(self, node):
        return self.node_state(node) == "goal"
```

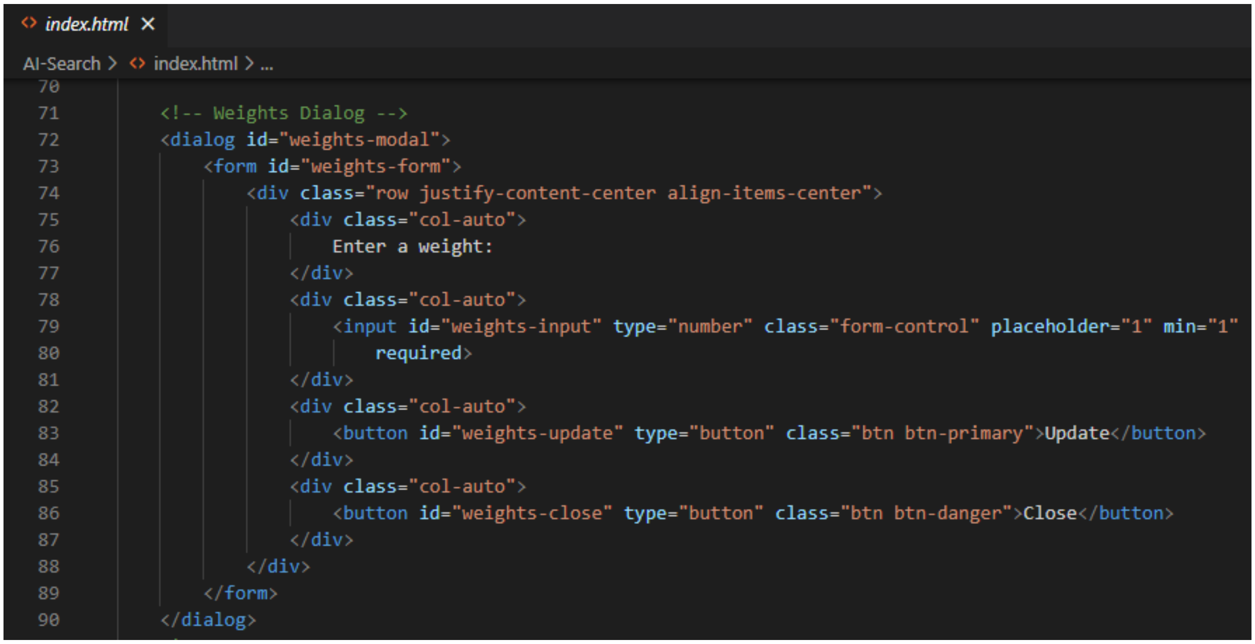


```python

    # Finished with "success" or "failed"
    def finished(self, result, goal):
        self.__agent_status = result
        if result == "failed":
            self.graph[goal.name].state = "source"
            return

        for node_name in goal.path[0:]:
            self.graph[node_name].state = "path"
        self.graph[goal.path[0]].state = "source"
```

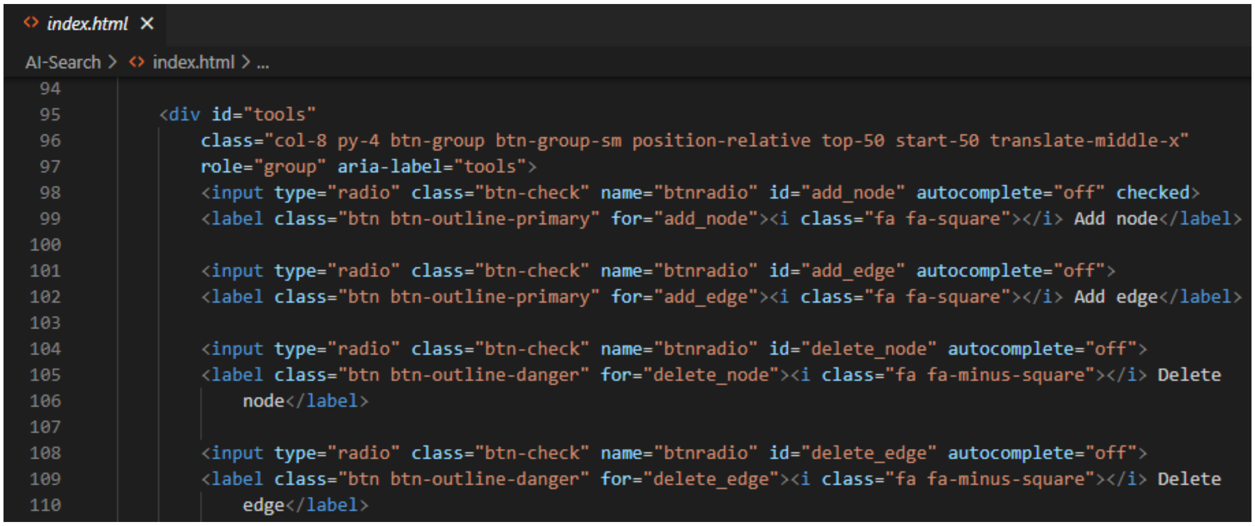
7.5- Index.html



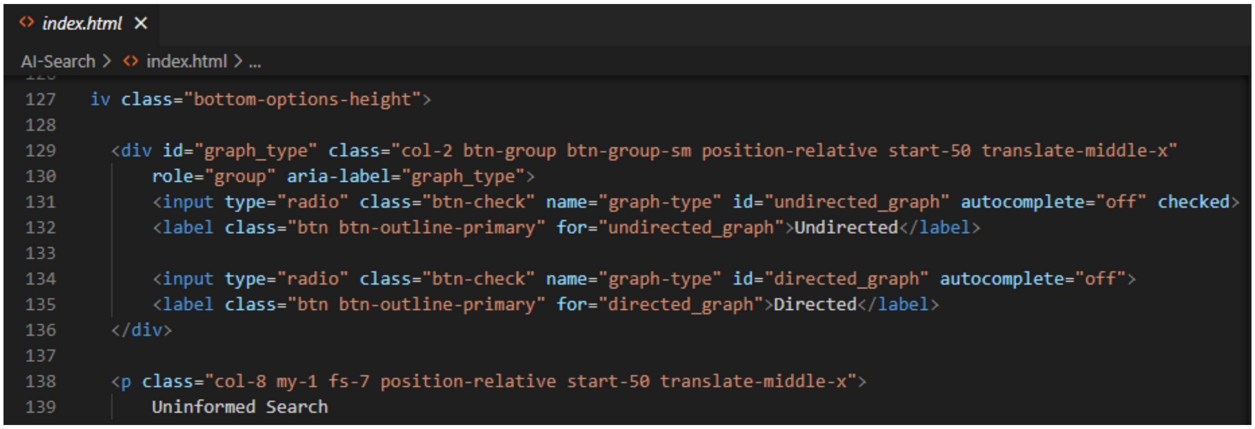
```html
<body id="body" onload="brython()">

    <div class="header-height">
        <div class="row position-relative">

            <div class="col-1 py-4 px-5 position-absolute">
                <span data-bs-toggle="tooltip" data-bs-placement="bottom" title="Help">
                    <button type="button" class="shadow btn btn-primary btn-sm btn-circle" data-bs-toggle="modal
                        data-bs-target="#help-modal">
                        <i class="fa fa-question"></i>
                    </button>
                </span>
            </div>

            <!-- Help Modal -->
            <div class="modal fade" id="help-modal" tabindex="-1" aria-labelledby="help-modal-label" aria-hidder
                <div class="modal-dialog modal-dialog-centered modal-dialog-scrollable">
                    <div class="modal-content">
                        <div class="modal-header">
                            <h5 class="modal-title" id="help-modal-label">Help</h5>
                            <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"><
                        </div>
```



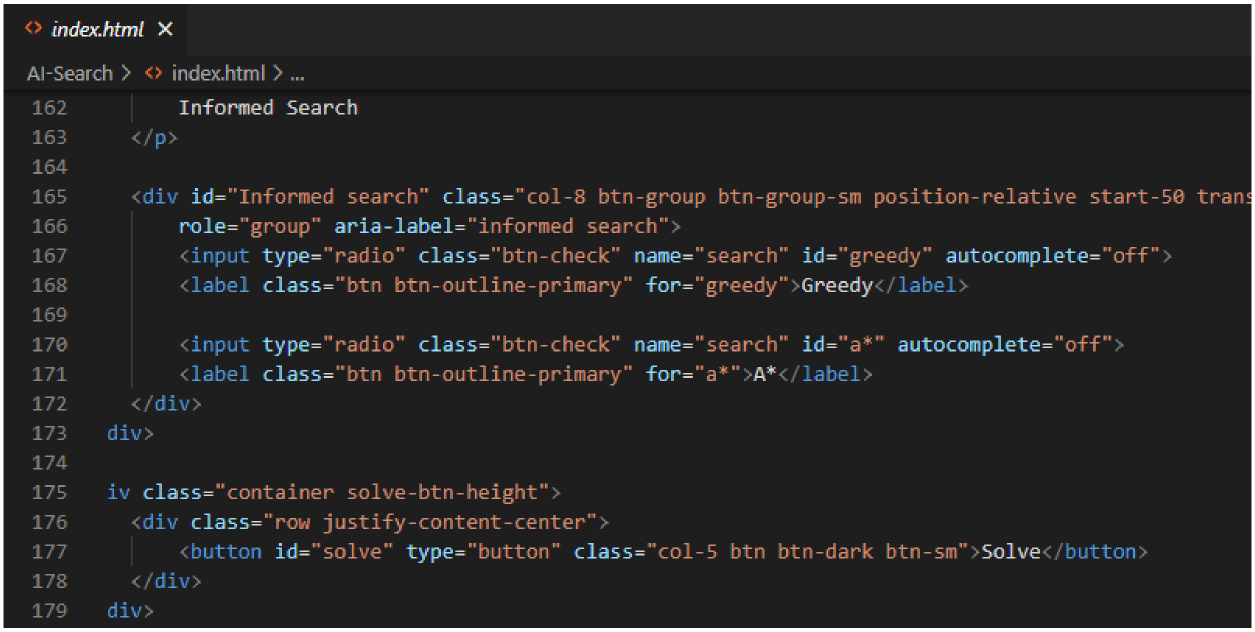
```html
        <!-- Weights Dialog -->
        <dialog id="weights-modal">
            <form id="weights-form">
                <div class="row justify-content-center align-items-center">
                    <div class="col-auto">
                        Enter a weight:
                    </div>
                    <div class="col-auto">
                        <input id="weights-input" type="number" class="form-control" placeholder="1" min="1"
                            required>
                    </div>
                    <div class="col-auto">
                        <button id="weights-update" type="button" class="btn btn-primary">Update</button>
                    </div>
                    <div class="col-auto">
                        <button id="weights-close" type="button" class="btn btn-danger">Close</button>
                    </div>
                </div>
            </form>
        </dialog>
```



```
    <div id="tools"
        class="col-8 py-4 btn-group btn-group-sm position-relative top-50 start-50 translate-middle-x"
        role="group" aria-label="tools">
        <input type="radio" class="btn-check" name="btnradio" id="add_node" autocomplete="off" checked>
        <label class="btn btn-outline-primary" for="add_node"><i class="fa fa-square"></i> Add node</label>

        <input type="radio" class="btn-check" name="btnradio" id="add_edge" autocomplete="off">
        <label class="btn btn-outline-primary" for="add_edge"><i class="fa fa-square"></i> Add edge</label>

        <input type="radio" class="btn-check" name="btnradio" id="delete_node" autocomplete="off">
        <label class="btn btn-outline-danger" for="delete_node"><i class="fa fa-minus-square"></i> Delete
            node</label>

        <input type="radio" class="btn-check" name="btnradio" id="delete_edge" autocomplete="off">
        <label class="btn btn-outline-danger" for="delete_edge"><i class="fa fa-minus-square"></i> Delete
            edge</label>
```

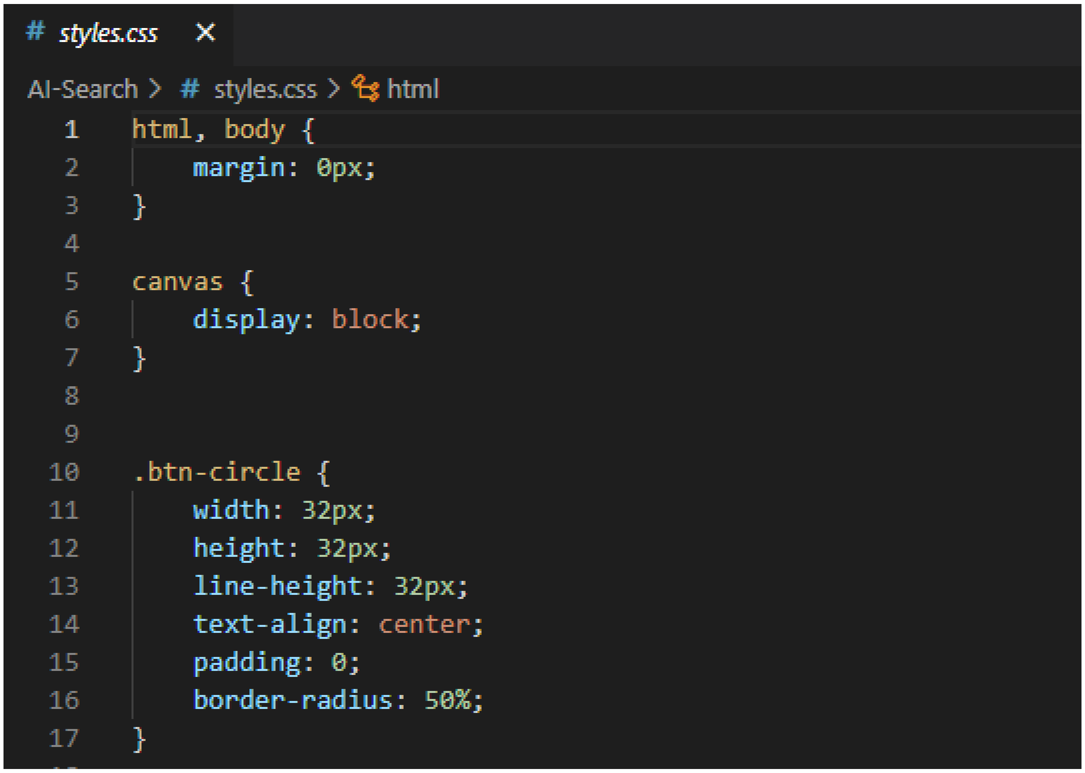


```
iv class="bottom-options-height">

  <div id="graph_type" class="col-2 btn-group btn-group-sm position-relative start-50 translate-middle-x"
      role="group" aria-label="graph_type">
      <input type="radio" class="btn-check" name="graph-type" id="undirected_graph" autocomplete="off" checked>
      <label class="btn btn-outline-primary" for="undirected_graph">Undirected</label>

      <input type="radio" class="btn-check" name="graph-type" id="directed_graph" autocomplete="off">
      <label class="btn btn-outline-primary" for="directed_graph">Directed</label>
  </div>

  <p class="col-8 my-1 fs-7 position-relative start-50 translate-middle-x">
      Uninformed Search
```

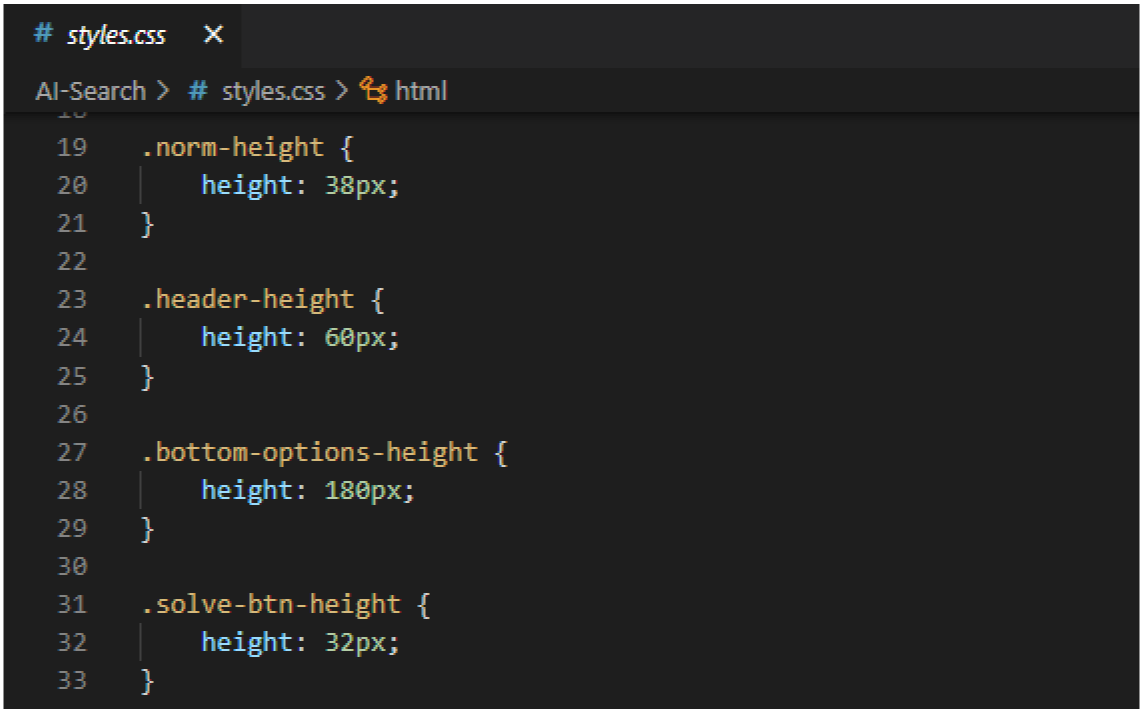


```
      Informed Search
  </p>

  <div id="Informed search" class="col-8 btn-group btn-group-sm position-relative start-50 trans
      role="group" aria-label="informed search">
      <input type="radio" class="btn-check" name="search" id="greedy" autocomplete="off">
      <label class="btn btn-outline-primary" for="greedy">Greedy</label>

      <input type="radio" class="btn-check" name="search" id="a*" autocomplete="off">
      <label class="btn btn-outline-primary" for="a*">A*</label>
  </div>
div>

iv class="container solve-btn-height">
  <div class="row justify-content-center">
      <button id="solve" type="button" class="col-5 btn btn-dark btn-sm">Solve</button>
  </div>
div>
```

7.6- Style.css

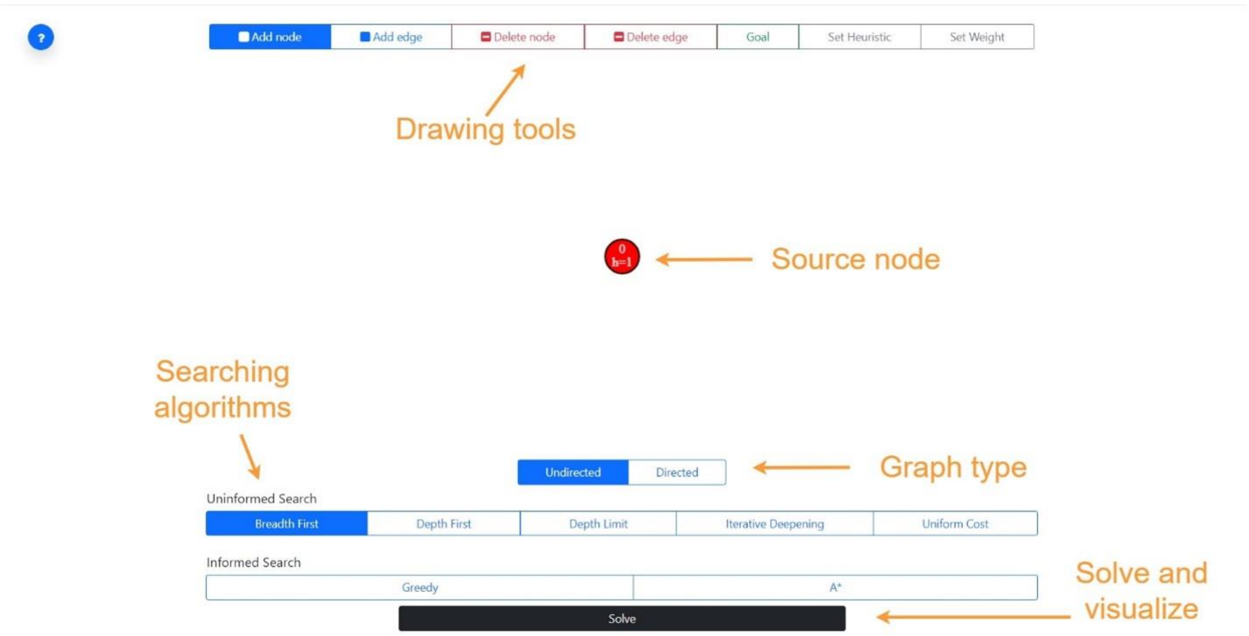


```css
html, body {
    margin: 0px;
}

canvas {
    display: block;
}


.btn-circle {
    width: 32px;
    height: 32px;
    line-height: 32px;
    text-align: center;
    padding: 0;
    border-radius: 50%;
}
```



```css
.norm-height {
    height: 38px;
}

.header-height {
    height: 60px;
}

.bottom-options-height {
    height: 180px;
}

.solve-btn-height {
    height: 32px;
}
```

8- Result/Output



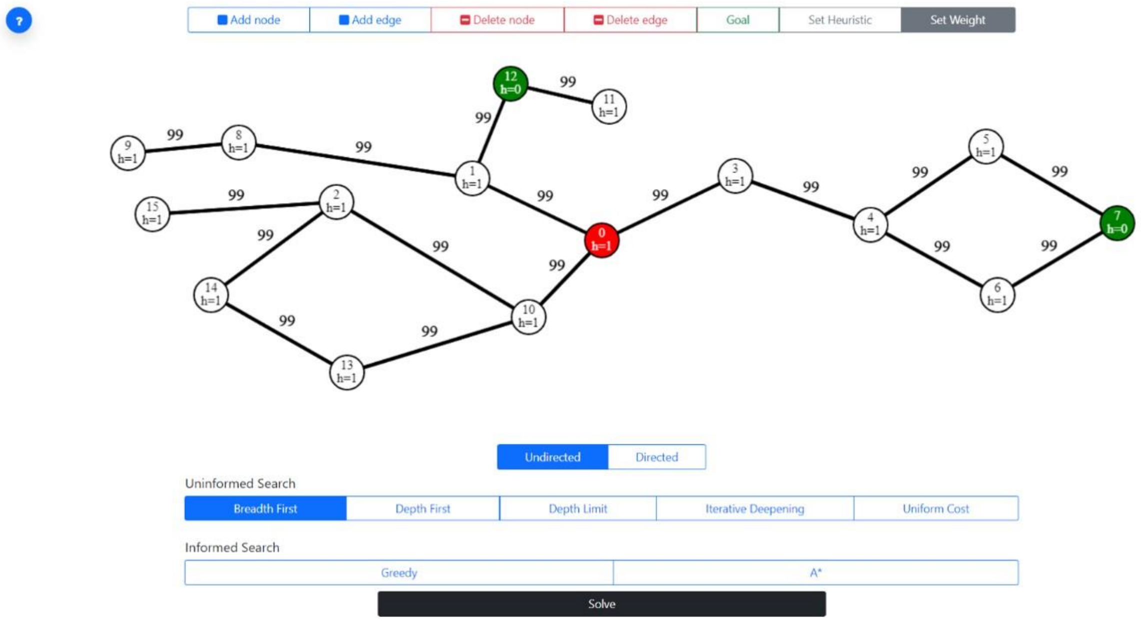
Drawing tools

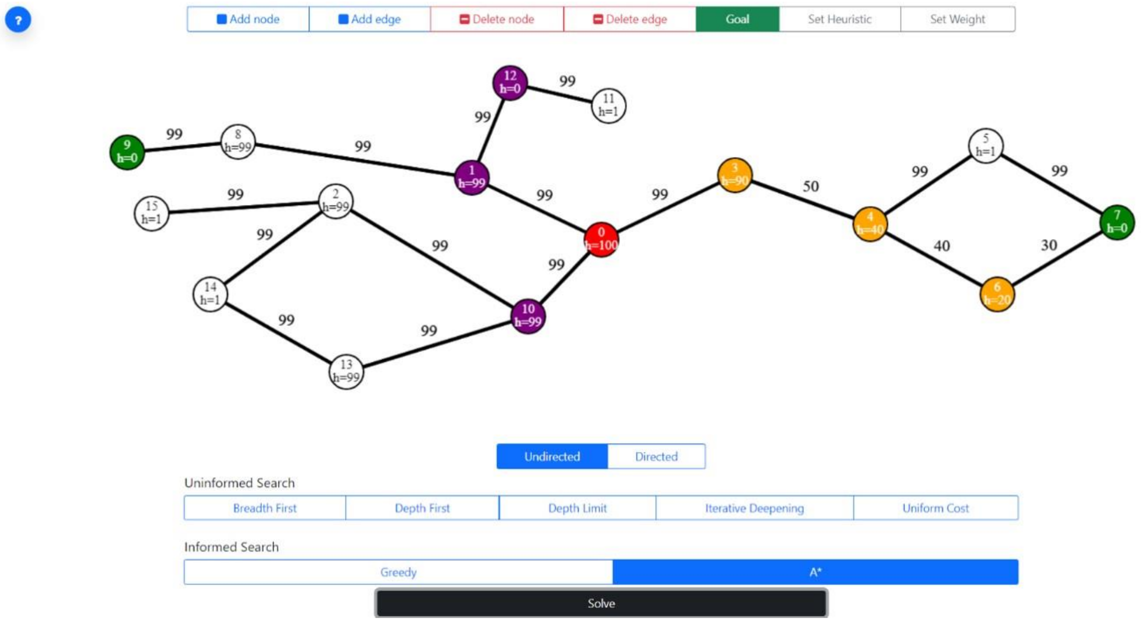
Source node

Searching algorithms

Graph type

Solve and visualize


?
Add node
Add edge
Delete node
Delete edge
Goal
Set Heuristic
Set Weight
12 h=0
11 h=1
9 h=1
8 h=1
1 h=1
15 h=1
2 h=1
0 h=1
3 h=1
4 h=1
5 h=1
6 h=1
7 h=0
14 h=1
10 h=1
13 h=1
99
Undirected
Directed
Uninformed Search
Breadth First
Depth First
Depth Limit
Iterative Deepening
Uniform Cost
Informed Search
Greedy
A*
Solve


?
Add node
Add edge
Delete node
Delete edge
Goal
Set Heuristic
Set Weight
12 h=0
11 h=1
9 h=0
8 h=99
1 h=99
15 h=1
2 h=99
0 h=100
3 h=90
4 h=40
5 h=1
6 h=20
7 h=0
14 h=1
10 h=99
13 h=99
99
50
40
30
Undirected
Directed
Uninformed Search
Breadth First
Depth First
Depth Limit
Iterative Deepening
Uniform Cost
Informed Search
Greedy
A*
Solve

9- Conclusion

Since, we know that searches are a common task these days and it is to be done efficiently in order to achieve a wholesome output. Searches in Artificial Intelligence corresponds the nature of work like it can done in many ways but the target is to choose the path which takes us to the goal. Goal state can be reached in many ways but the challenge is to pick up the most appropriate algorithm.

However, we know the difference between LIFO (Stack) and FIFO (Queue) and these terms are essential in implementing the algorithms because it depends on the requirements of the problem. For instance, if we are using the Breadth-First Search (BFS) technique then the entire algorithm is to be based on the technique involved in LIFO (Stack) and vice versa with FIFO (Queue).

- Basic and complex searching problems are now easy
- Problem solving agent must be the rational
- Uninformed searches take a bit more time as compared to informed searches
- Uninformed searches have information of the goal node
- Directed graphs can go either way whereas undirected graphs have only one direction
- A* search method is the efficient one among all if branching factor is to be taken into consideration