CM2602 Artificial Intelligence

Coursework Report

Module Leader – Mr. Nipuna Senanakaye

Name : Lisara Gajaweera

RGU ID : 2118813

IIT ID : 20211029

**Question 1**

**a)1.** Elaborate the functionality of the Naïve Auction Algorithm using an appropriate flow chart / pseudocode.

Diagram

Description automatically generated

**a)2.** What are the issues associated with “Naïve Auction Algorithm”?

* The algorithm does not take into account any of the additional information and constraints except the predetermined prices of the items and the bids submitted by the bidders.
* The algorithm is not optimal to allocate items based on the overall values of the items which means to the bidders/ the bidders’ budget. This causes,
  + Bidders may be willing to pay more for an item than the value it actually has to them.
  + Some of the bidders may not be able to afford the items they value a lot.

**a)3.**How do you rectify those issues? Again, elaborate the introduced modifications and their impacts to the algorithm via using an appropriate flow chart / pseudocode

* by making the agents bid more than the cost spend to get the item

#Initialize the list of items

items = [“1”,”2”,”3”,..]

#Initialize list of agents

agents = [“d”,”c”,”b”…]

#Initialize a dictionary to hold each items along with the bidder

itemHolderDict = {}

#initialize a dictionary to hold the cost of the item for each bidder

costs = {}

def naïve\_auction(agents, item, costs, bids, itemHolderDict):

#Initialize the list of agents and the item

agents = agents

item = item

#for each agent

for agent in agents:

#add bid to dict, if bid is greater than cost

If agent:amount > cost[item]

Bids.update(agent:amount)

# sort the bids dictionary in descending order based on the bid value

#allocate the item to the agent in bid dictionary in 0th index

itemHolder = bids[0]

# update itemHolderDict and return

return itemHolderDict.update({item:itemHolder})

# iterate through the items

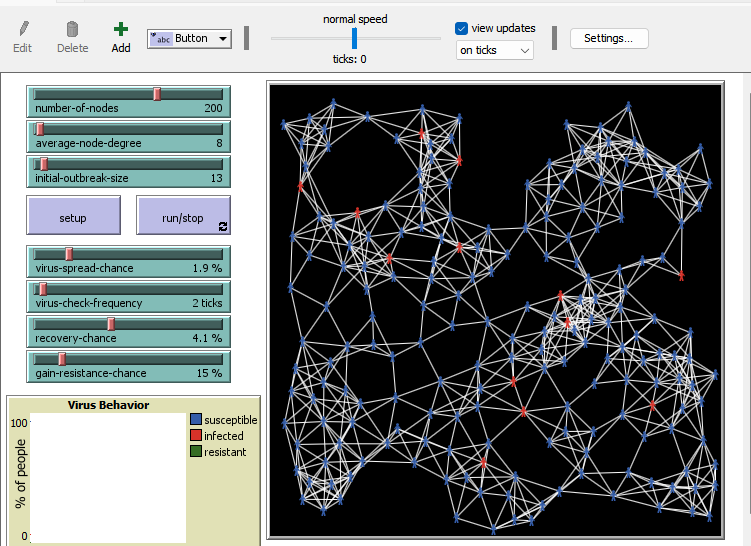
for item in items:

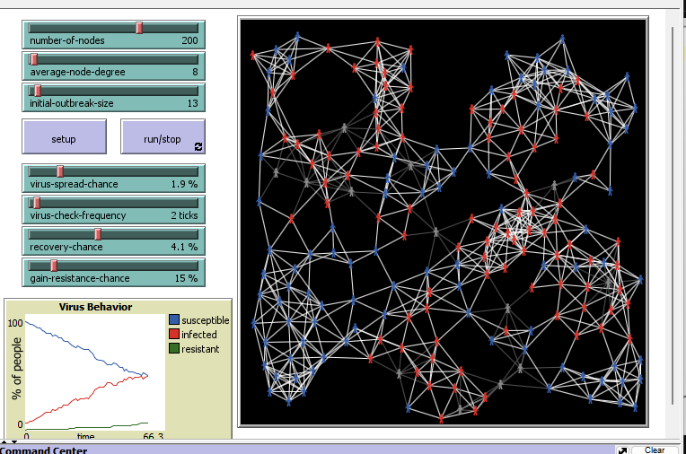
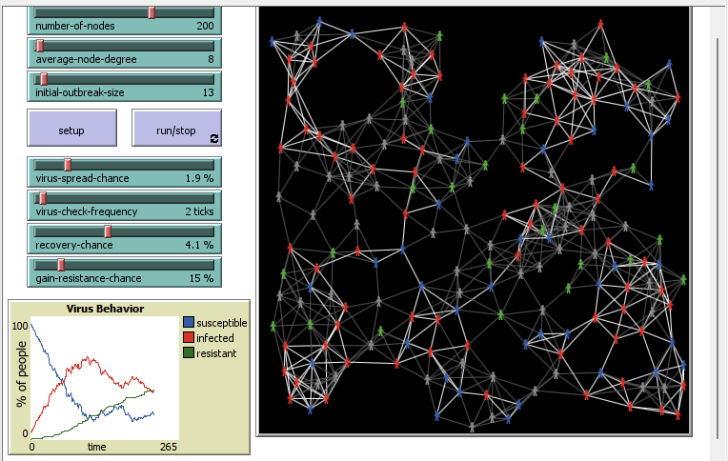
# calling the naiveAuction function

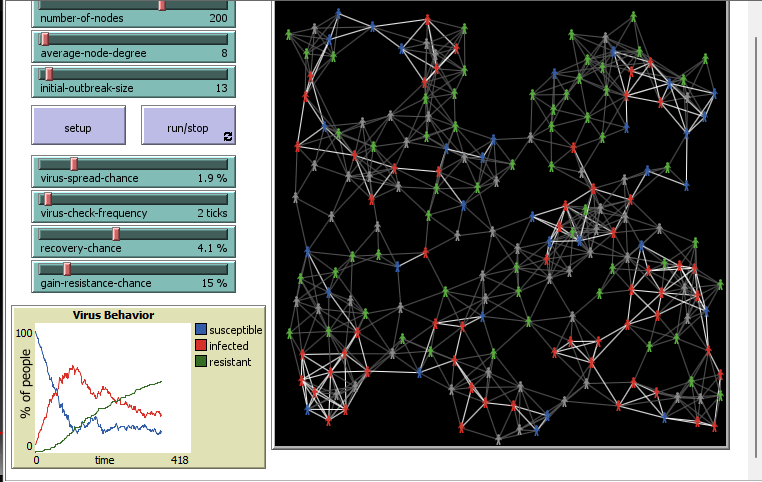
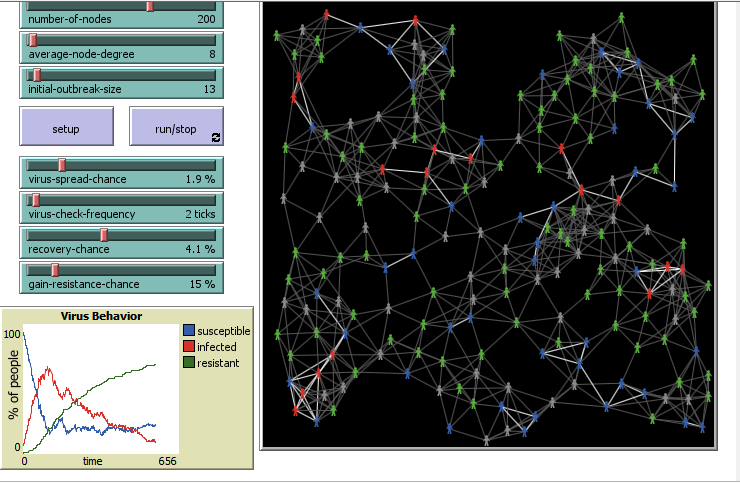
naiveAuction(agents, item, costs, bids, resourceHolderDict)

**b)1.**Refer to the self-study research article uploaded in the Moodle on Virus modelling

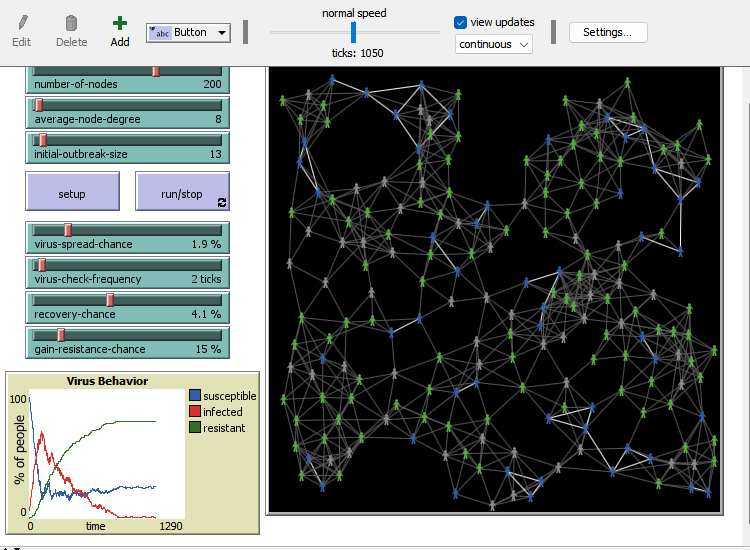
**b)2.**Create a Netlogo Model to visualize the virus behavior.

Initial stage





Final stage



**b)3.** Elaborate the code snippets functionalities of your proposed model with appropriate screenshots and explanations

turtles-own

[

infected? ;; if true, - infectious

resistant? ;; if true, - can't be infected

virus-check-timer ;; number of ticks since this person's last virus-check

]

to setup ;setup button

clear-all

setup-nodes

setup-spatially-clustered-network

ask n-of initial-outbreak-size turtles

[ become-infected ]

ask links [ set color white ]

reset-ticks

end

to run/stop ;go button

if all? turtles [not infected?]

[ stop ]

ask turtles

[

set virus-check-timer virus-check-timer + 1

if virus-check-timer >= virus-check-frequency

[ set virus-check-timer 0 ]

]

spread-virus

do-virus-checks

tick

end

to become-infected

set infected? true

set resistant? false

set color red

end

to become-susceptible

set infected? false

set resistant? false

set color blue

end

to become-resistant

set infected? false

set resistant? true

set color green

ask my-links [ set color gray - 2 ]

end

to spread-virus

ask turtles with [infected?]

[ ask link-neighbors with [not resistant?]

[ if random-float 100 < virus-spread-chance

[ become-infected ] ] ]

end

to do-virus-checks

ask turtles with [infected? and virus-check-timer = 0]

[

if random 100 < recovery-chance

[

ifelse random 100 < gain-resistance-chance

[ become-resistant ]

[ become-susceptible ]

]

]

end

to setup-nodes

set-default-shape turtles "person"

create-turtles number-of-nodes

[

; for visual reasons,

; don't put any nodes \*too\* close to the edges of the window

setxy (random-xcor \* 0.94) (random-ycor \* 0.94)

become-susceptible

set virus-check-timer random virus-check-frequency

]

end

to setup-spatially-clustered-network

let num-links (average-node-degree \* number-of-nodes) / 2

while [count links < num-links ]

[

ask one-of turtles

[

let choice (min-one-of (other turtles with [not link-neighbor? myself])

[distance myself])

if choice != nobody [ create-link-with choice ]

]

]

; make the network look a little prettier

repeat 10

[

layout-spring turtles links 0.3 (world-width / (sqrt number-of-nodes)) 1

]

end

**Question 2**

**1.** Define the scope you attempt to cover from your knowledge model/ontology and propose relevant competency questions (at least 05 aspects). List all the sources you referred to for the information gathering in the references section.

There are few sections that I’m willing to cover in the ontology of covid virus. Mainly those are the background of the covid virus, infection possibilities, the necessary precautions, treatment methods, symptoms, types and severity of the symptoms.

Possible competency questions relevant to the knowledge base are,

1. What are the variants of the Covid-19 virus.
2. What are the treatments available for the virus.
3. How can we stop/ reduce the spread of the virus.
4. What are the symptoms of the Covid-19 virus.
5. What are the types of symptoms that are possible in Covid-19

**2.** Suggest an appropriate concept graph (i.e., taxonomy) to link the information fragments you identified above

A picture containing diagram

Description automatically generated

**3.** Design a suitable domain ontology (i.e. RDF / OWL) to make above proposed concept graph machine-readable .Introduce at least 05 individuals for each knowledge branch

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

        xmlns:owl="http://www.w3.org/2002/07/owl#"

        xmlns:dc="http://purl.org/dc/elements/1.1/">

<!-- Class, Subclass are coming from OWL -->

<!-- Title, Description - ontology used to build ontology -->

<!-- <rdf:Description rdf:about="subject">

    <predicate rdf:resource="object" />

    <predicate>literal value</predicate>

</rdf:Description> -->

<!-- OWL header -->

<owl:Ontology rdf:about= "http://www.linkeddatatools.com/corona">

    <dc:title>Covid-19 Ontology</dc:title>

    <dc:description>An ontology of SARS Novel Corona Virus or Covid-19</dc:description>

</owl:Ontology>

<!-- Classes.............................................................. -->

<!-- OWL Class- Corona -->

<owl:Class rdf:about= "http://www.linkeddatatools.com/corona#covid19">

    <rdfs: label>The Novel corona Virus</rdfs:label>

    <rdfs: comment>the novel corona virus/covid-19 virus</rdfs:comment>

</owl:Class>

<!-- Define Class- Variant -->

<owl:Class rdf:about= "http://www.linkeddatatools.com/cov19#variant">

    <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/corona#covid19"/>

    <rdfs: label>The Covid-19 virus variants</rdfs:label>

    <rdfs: comment>The SARs covid-19 have many straints that mutated from the initial version of the virus</rdfs:comment>

</owl:Class>

<!-- Define Class- Treatments -->

<owl:Class rdf:about= "http://www.linkeddatatools.com/corona#treatment">

    <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/corona#covid19"/>

    <rdfs: label>treatments</rdfs:label>

    <rdfs: comment>Treatements to the covid-19 patients</rdfs:comment>

</owl:Class>

<!-- Define class-symptoms -->

<owl:Class rdf:about= "http://www.linkeddatatools.com/corona#symptoms">

    <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/corona#covid19"/>

    <rdfs: label>symptoms</rdfs:label>

    <rdfs: comment>Symptoms of the covid 19 virus</rdfs:comment>

</owl:Class>

<!-- DefineClass - prevention measures -->

<owl:Class rdf:about= "http://www.linkeddatatools.com/corona#prevent">

    <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/corona#covid19"/>

    <rdfs: label>Prevention and controlling</rdfs:label>

    <rdfs: comment>The prevention and control methods</rdfs:comment>

</owl:Class>

<owl:Class rdf:about= "http://www.linkeddatatools.com/corona">

    <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/corona#covid19"/>

    <rdfs: label></rdfs:label>

    <rdfs: comment></rdfs:comment>

</owl:Class>

<!-- Symptoms sub classes -->

<!-- muscle-skeleton -->

<owl:Class rdf:about= "http://www.linkeddatatools.com/corona#muscoskeletal">

    <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/corona#symptoms"/>

    <rdfs: label>Muscoskeletal Symptoms</rdfs:label>

    <rdfs: comment>Symptoms related to muscle and bone systems</rdfs:comment>

</owl:Class>

<!-- Digestive Symptoms -->

<owl:Class rdf:about= "http://www.linkeddatatools.com/corona#digestive">

    <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/corona#symptoms"/>

    <rdfs: label>Digestive Symptoms</rdfs:label>

    <rdfs: comment>Systems related to the digestive system</rdfs:comment>

</owl:Class>

<!-- Respiratory Symptoms -->

<owl:Class rdf:about= "http://www.linkeddatatools.com/corona#respiratory">

    <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/corona#symptoms"/>

    <rdfs: label>Respiratory Symptoms</rdfs:label>

    <rdfs: comment>Systems related to respiratory system</rdfs:comment>

</owl:Class>

<!-- End of defining Classes.............................................................. -->

<!-- Defining data properties -->

<!-- Define the severity property -->

<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/corona#severity"/>

<rdf:Description rdf:about="http://www.linkeddatatools.com/corona#fatigue">

    <!-- fatigue is a type (instance) of the muscoskeletal class -->

    <rdf:type rdf:resource="http://www.linkeddatatools.com/corona#muscoskeletal"/>

    <!-- The fatigue  has a  'mild' severity -->

    <corona:severity>mild</corona:severity>

</rdf:Description>

<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/corona#severity"/>

<rdf:Description rdf:about="http://www.linkeddatatools.com/corona#musclepain">

    <!-- muscle Pain is a type (instance) of the muscoskeletal class -->

    <rdf:type rdf:resource="http://www.linkeddatatools.com/corona#muscoskeletal"/>

    <!-- The muscle Pain  has a  'mild' severity -->

    <corona:severity>mild</corona:severity>

</rdf:Description>

<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/corona#severity"/>

<rdf:Description rdf:about="http://www.linkeddatatools.com/corona#headache">

    <!-- headache is a type (instance) of the muscoskeletal class -->

    <rdf:type rdf:resource="http://www.linkeddatatools.com/corona#muscoskeletal"/>

    <!-- The headache  has a  'mild' severity -->

    <corona:severity>mild</corona:severity>

</rdf:Description>

<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/corona#severity"/>

<rdf:Description rdf:about="http://www.linkeddatatools.com/corona#abdominalpain">

    <!-- abdominal pain is a type (instance) of the digestive class -->

    <rdf:type rdf:resource="http://www.linkeddatatools.com/corona#digestive"/>

    <!-- The  abdominal pain  has a  'mild' severity -->

    <corona:severity>mild</corona:severity>

</rdf:Description>

<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/corona#severity"/>

<rdf:Description rdf:about="http://www.linkeddatatools.com/corona#diarrhea">

    <!-- diarrhea is a type (instance) of the digestive class -->

    <rdf:type rdf:resource="http://www.linkeddatatools.com/corona#digestive"/>

    <!-- The diarrhea  has a  'mild' severity -->

    <corona:severity>mild</corona:severity>

</rdf:Description>

<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/corona#severity"/>

<rdf:Description rdf:about="http://www.linkeddatatools.com/corona#vomitting">

    <!--vomitting  is a type (instance) of the digestive class -->

    <rdf:type rdf:resource="http://www.linkeddatatools.com/corona#digestive"/>

    <!-- The vomitting  has a  'mild' severity -->

    <corona:severity>mild</corona:severity>

</rdf:Description>

<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/corona#severity"/>

<rdf:Description rdf:about="http://www.linkeddatatools.com/corona#dyspnea">

    <!-- dyspnea is a type (instance) of the respiratory class -->

    <rdf:type rdf:resource="http://www.linkeddatatools.com/corona#respiratory"/>

    <!-- The dyspnea  has a  'high' severity -->

    <corona:severity>high</corona:severity>

</rdf:Description>

<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/corona#severity"/>

<rdf:Description rdf:about="http://www.linkeddatatools.com/corona#hypoxia">

    <!-- hypoxia is a type (instance) of the respiratory class -->

    <rdf:type rdf:resource="http://www.linkeddatatools.com/corona#respiratory"/>

    <!-- The hypoxia  has a  'high'' severity -->

    <corona:severity>high</corona:severity>

</rdf:Description>

<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/corona#severity"/>

<rdf:Description rdf:about="http://www.linkeddatatools.com/corona#respfailure">

    <!-- repiratory failure is a type (instance) of the respiratory class -->

    <rdf:type rdf:resource="http://www.linkeddatatools.com/corona#respiratory"/>

    <!-- The repiratory failure has a 'high' severity -->

    <corona:severity>high</corona:severity>

</rdf:Description>

<!-- Pharmocological type -->

</rdf:RDF>

**Question 3**

**Task 1**

Code

def draw\_grid(window, rows, width):

    gap = width // rows

    for i in range(rows):

        pygame.draw.line(window, GREY, (0, i \* gap), (width, i \* gap))

        for j in range(rows):

            pygame.draw.line(window, GREY, (j \* gap, 0), (j \* gap, width))

def draw(window, grid, rows, width):

    window.fill(WHITE)

    for row in grid:

        for node in row:

            node.draw(window)

    draw\_grid(window, rows, width)

    pygame.display.update()

def make\_grid(rows, width):

    grid = []

    gap = width // rows

    for i in range(rows):

        grid.append([])

        for j in range(rows):

            spot = Spot(i, j, gap, rows)

            grid[i].append(spot)

    return grid

def main\_f(window, width):

    ROW = 6

    grid = make\_grid(ROW, width)

    start = None

    end = None

    #START NODE

    s\_row = random.randint(0,ROW-1)

    s\_col = random.randint(0,1)

    print(s\_row , s\_col)

    # Goal node

    g\_row = random.randint(0,ROW-1)

    g\_col = random.randint(ROW-2,ROW-1)

    print(g\_row , g\_col)

    s\_node = grid[s\_col][s\_row]

    s\_node.make\_start()

    g\_node = grid[g\_col][g\_row]

    g\_node.make\_end()

    noOfBarriers = 4

    barriers\_made = 0

    while barriers\_made<noOfBarriers:

      barrier = grid[random.randint(0,ROW-1)][random.randint(0,ROW-1)]

      if barrier == s\_node or barrier ==g\_node:

        pass

      else:

        barrier.make\_barrier()

        barriers\_made=barriers\_made+1

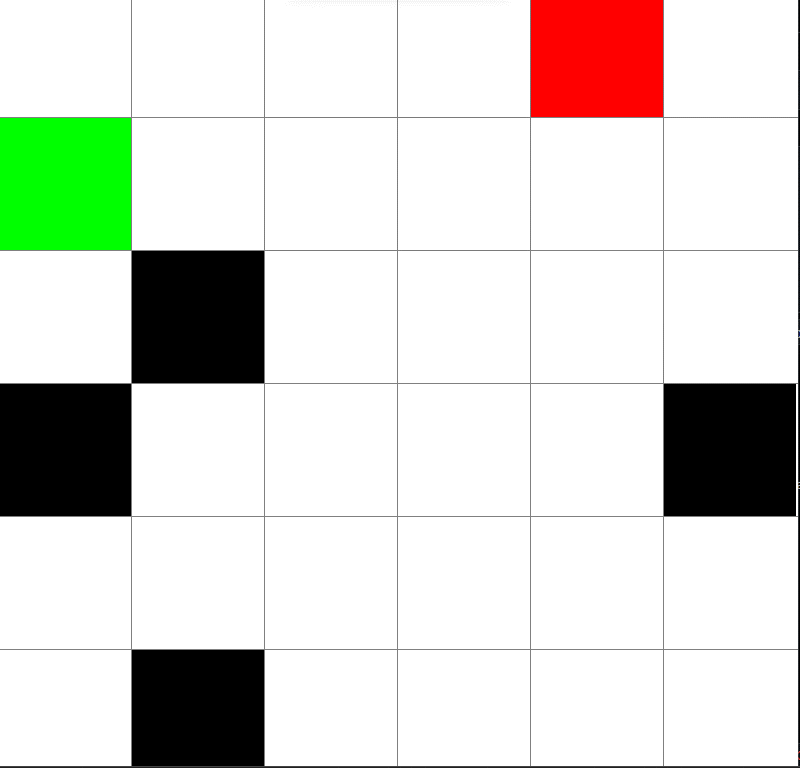
    run = True

    while run:

        draw(window, grid, ROW, width)

    pygame.quit()

Output



**Task 2**

**Chart

Description automatically generated**

**Task 3**

**A star**

**Chart

Description automatically generated**

**Task 4**

**Task 5**

|  |  |  |
| --- | --- | --- |
| **DFS** | **A star** |  |
|  |  |  |
|  |  |  |
|  |  |  |

Completeness- Both are complete. They both reach the goal

Optimality- A\* is more optimized as it finds the shortest path to the goal. DFS does not find the shortest path always.

Time Complexity

The time complexity of the A\* depends on the heuristic function. Usually the A\* algorithm is more fast than the DFS algorithm

Appendix

Code of the Question5

import time

import pygame

import random

import math

from queue import PriorityQueue

WIDTH = 800

WINDOW = pygame.display.set\_mode((WIDTH, WIDTH))

pygame.display.set\_caption("Search Algorithms")

RED = (255, 0, 0)

PURPLE = (24, 16, 46)

MAGENTA = (48, 20, 40)

GREY = (128, 128, 128)

TURQU = (20, 91, 201)

GREEN = (0, 255, 0)

BLUE = (0, 255, 0)

YELLOW = (255, 255, 0)

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

class Node:

    def \_\_init\_\_(self, row, column, width, total\_rows):

        self.x = row \* width

        self.y = column \* width

        self.color = WHITE

        self.neighbors = []

        self.row = row

        self.column = column

        self.width = width

        self.total\_rows = total\_rows

    def get\_position(self):

        return self.row, self.column

    def is\_closed(self):

        return self.color == PURPLE

    def is\_open(self):

        return self.color == YELLOW

    def is\_barrier(self):

        return self.color == BLACK

    def is\_start(self):

        return self.color == GREEN

    def is\_goal(self):

        return self.color == RED

    def reset(self):

        self.color = WHITE

    def make\_close(self):

        self.color = PURPLE

    def make\_open(self):

        self.color = YELLOW

    def make\_barrier(self):

        self.color = BLACK

    def make\_start(self):

        self.color = GREEN

    def make\_goal(self):

        self.color = RED

    def make\_path(self):

        self.color = MAGENTA

    def draw(self, win):

        pygame.draw.rect(win, self.color, (self.x, self.y, self.width, self.width))

    def update\_neighbours(self, grid):

        self.neighbors = []

        # NW,W,SW,N,S,NE,E,SE

        #North West

        if self.row - 1 >= 0 and self.column - 1 >= 0 and not grid[self.row - 1][self.column - 1].is\_barrier() and not grid[self.row - 1][self.column - 1].is\_open():  # LEFT UP

            self.neighbors.append(grid[self.row - 1][self.column - 1])

        #West

        if self.column - 1 >= 0 and not grid[self.row][self.column - 1].is\_barrier() and not grid[self.row][self.column - 1].is\_open():  # LEFT

            self.neighbors.append(grid[self.row][self.column - 1])

        #SouthWest

        if self.row + 1 < self.total\_rows and self.column - 1 >= 0 and not grid[self.row + 1][self.column - 1].is\_barrier() and not grid[self.row + 1][self.column - 1].is\_open():  # LEFT DOWN

            self.neighbors.append(grid[self.row + 1][self.column - 1])

        #North

        if self.row - 1 >= 0 and not grid[self.row - 1][self.column].is\_barrier() and not grid[self.row - 1][self.column].is\_open():  # UP

            self.neighbors.append(grid[self.row - 1][self.column])

        #South

        if self.row + 1 < self.total\_rows and not grid[self.row + 1][self.column].is\_barrier() and not grid[self.row + 1][self.column].is\_open():  # DOWN

            self.neighbors.append(grid[self.row + 1][self.column])

        #NorthEast

        if self.row - 1 >= 0 and self.column + 1 < self.total\_rows and not grid[self.row - 1][self.column + 1].is\_barrier() and not grid[self.row - 1][self.column + 1].is\_open():  # RIGHT UP

            self.neighbors.append(grid[self.row - 1][self.column + 1])

        #East

        if self.column + 1 < self.total\_rows and not grid[self.row][self.column + 1].is\_barrier() and not grid[self.row][self.column + 1].is\_open():  # RIGHT

            self.neighbors.append(grid[self.row][self.column + 1])

        #South East

        if self.row + 1 < self.total\_rows and self.column + 1 < self.total\_rows and not grid[self.row + 1][self.column + 1].is\_barrier() and not grid[self.row + 1][self.column + 1].is\_open():  # RIGHT DOWN

            self.neighbors.append(grid[self.row + 1][self.column + 1])

    def \_\_lt\_\_(self, other):

        return False

def final\_path(came\_from, curr\_node, draw):

    while curr\_node in came\_from:

        curr\_node = came\_from[curr\_node]

        curr\_node.make\_path()

        draw()

def dfs\_final\_path(draw, path, start, goal):

    cell = goal

    while cell != start:

        node = path[cell]

        node.make\_path()

        cell = node

        draw()

    goal.make\_goal()

    start.make\_start()

def heuristic\_cost(curr\_node, goal\_node):

    curr\_node\_y, curr\_node\_x = curr\_node

    goal\_node\_y, goal\_node\_x = goal\_node

    return max(abs(curr\_node\_x - goal\_node\_x), abs(curr\_node\_y - goal\_node\_y))

def DFS(draw, start, goal):

    exploredList = [start]

    frontierList = [start]

    visited = []

    dfs\_path = {}

    while len(frontierList) > 0: # while the nodes list that should be discovered is not empty

        current\_node = frontierList.pop()

        visited.append(current\_node)

        if current\_node == goal:

            dfs\_final\_path(draw, dfs\_path, start, goal)

            return len(visited), True

        for neighbor in reversed(current\_node.neighbors):

            if neighbor not in exploredList:

                frontierList.append(neighbor)

                exploredList.append(neighbor)

                dfs\_path[neighbor] = current\_node

                neighbor.make\_open()

        if current\_node != start:

            current\_node.make\_close()

        goal.make\_goal()

        draw()

        time.sleep(0.5)

    return len(visited), False

def A\_star(draw, grid, start, goal):

    count = 0

    visited\_count = 0

    frontierQ = PriorityQueue()

    frontierQ.put((0, count, start))

    came\_from = {}

    g\_score = {node: float("inf") for row in grid for node in row}

    g\_score[start] = 0

    f\_score = {node: float("inf") for row in grid for node in row}

    f\_score[start] = heuristic\_cost(start.get\_position(), goal.get\_position())

    frontier\_list = {start}

    while not frontierQ.empty():

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

        current = frontierQ.get()[2]

        frontier\_list.remove(current)

        visited\_count += 1

        if current == goal:

            final\_path(came\_from, goal, draw)

            start.make\_start()

            goal.make\_goal()

            return visited\_count - 1, True

        for neighbor in current.neighbors:

            temp\_g\_score = g\_score[current] + 1

            if temp\_g\_score < g\_score[neighbor]:

                came\_from[neighbor] = current

                g\_score[neighbor] = temp\_g\_score

                f\_score[neighbor] = temp\_g\_score + heuristic\_cost(neighbor.get\_position(), goal.get\_position())

                if neighbor not in frontier\_list:

                    count += 1

                    frontierQ.put((f\_score[neighbor], count, neighbor))

                    frontier\_list.add(neighbor)

                    neighbor.make\_open()

        goal.make\_goal()

        draw()

        time.sleep(0.6)

        if current != start:

            current.make\_close()

    return 0, False

def draw\_maze(window, rows, width):

    gap = width // rows

    for i in range(rows):

        pygame.draw.line(window, GREY, (0, i \* gap), (width, i \* gap))

        for j in range(rows):

            pygame.draw.line(window, GREY, (j \* gap, 0), (j \* gap, width))

def create\_maze(rows, width):

    maze = []

    gap = width // rows

    for i in range(rows):

        maze.append([])

        for j in range(rows):

            node = Node(i, j, gap, rows)

            maze[i].append(node)

    return maze

def draw(window, grid, rows, width):

    window.fill(WHITE)

    for row in grid:

        for node in row:

            node.draw(window)

    draw\_maze(window, rows, width)

    pygame.display.update()

def get\_random\_coordinates(minx, maxx, miny, maxy):

    x\_coordinate = random.randint(minx, maxx)

    y\_coordinate = random.randint(miny, maxy)

    return x\_coordinate, y\_coordinate

def initialize\_nodes(maze, ROWS):

    start\_row, start\_column = get\_random\_coordinates(0, 1, 0, ROWS - 1)

    goal\_row, goal\_column = get\_random\_coordinates(ROWS - 2, ROWS - 1, 0, ROWS - 1)

    start\_node = maze[start\_row][start\_column]

    start = start\_node

    start.make\_start()

    goal\_node = maze[goal\_row][goal\_column]

    goal = goal\_node

    goal.make\_goal()

    no\_of\_barriers = 4

    for i in range(no\_of\_barriers):

        barrier\_column, barrier\_row = get\_random\_coordinates(0, ROWS - 1, 0, ROWS - 1)

        barrier\_node = maze[barrier\_row][barrier\_column]

        while True:

            if barrier\_node != start and barrier\_node != goal and not barrier\_node.is\_barrier():

                barrier\_node.make\_barrier()

                break

            barrier\_column, barrier\_row = get\_random\_coordinates(0, ROWS - 1, 0, ROWS - 1)

            barrier\_node = maze[barrier\_row][barrier\_column]

    return start, goal

def reset\_maze(maze, start, goal):

    for row in maze:

        for node in row:

            if node.get\_position() != start and node.get\_position() != goal and not node.is\_barrier():

                node.reset()

    start.make\_start()

    goal.make\_goal()

    return maze

def main(window, width):

    a\_star\_visited\_count = 0

    dfs\_visited\_count = 0

    ROWS = 6

    grid = create\_maze(ROWS, width)

    start, goal = initialize\_nodes(grid, ROWS)

    run = True

    while run:

        draw(window, grid, ROWS, width)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                run = False

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_SPACE:

                    for row in grid:

                        for node in row:

                            node.update\_neighbours(grid)

                    dfs\_visited\_count, status = DFS(lambda: draw(window, grid, ROWS, width), start, goal)

    grid = reset\_maze(grid, start, goal)

    run = True

    while run:

        draw(window, grid, ROWS, width)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                run = False

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_SPACE:

                    for row in grid:

                        for node in row:

                            node.update\_neighbours(grid)

                    a\_star\_visited\_count, status = A\_star(lambda: draw(window, grid, ROWS, width), grid, start, goal)

    print("DFS search algorithm visited " + str(dfs\_visited\_count) + " nodes")

    print("A\* search algorithm visited " + str(a\_star\_visited\_count) + " nodes")

    pygame.quit()

if \_\_name\_\_ == '\_\_main\_\_':

    main(WINDOW, WIDTH)

References

<https://jamanetwork.com/journals/jama/pages/coronavirus-alert>

<https://www.bmj.com/coronavirus>

<https://www.elsevier.com/connect/coronavirus-information-center>

<https://www.springernature.com/gp/researchers/campaigns/coronavirus>

<https://www.youtube.com/watch?v=zzr6aJudoRI>

<https://novel-coronavirus.onlinelibrary.wiley.com/>

https://www.who.int/publications/i/item/therapeutics-and-covid-19-living-guideline