Name: Nguyen Khanh Toan

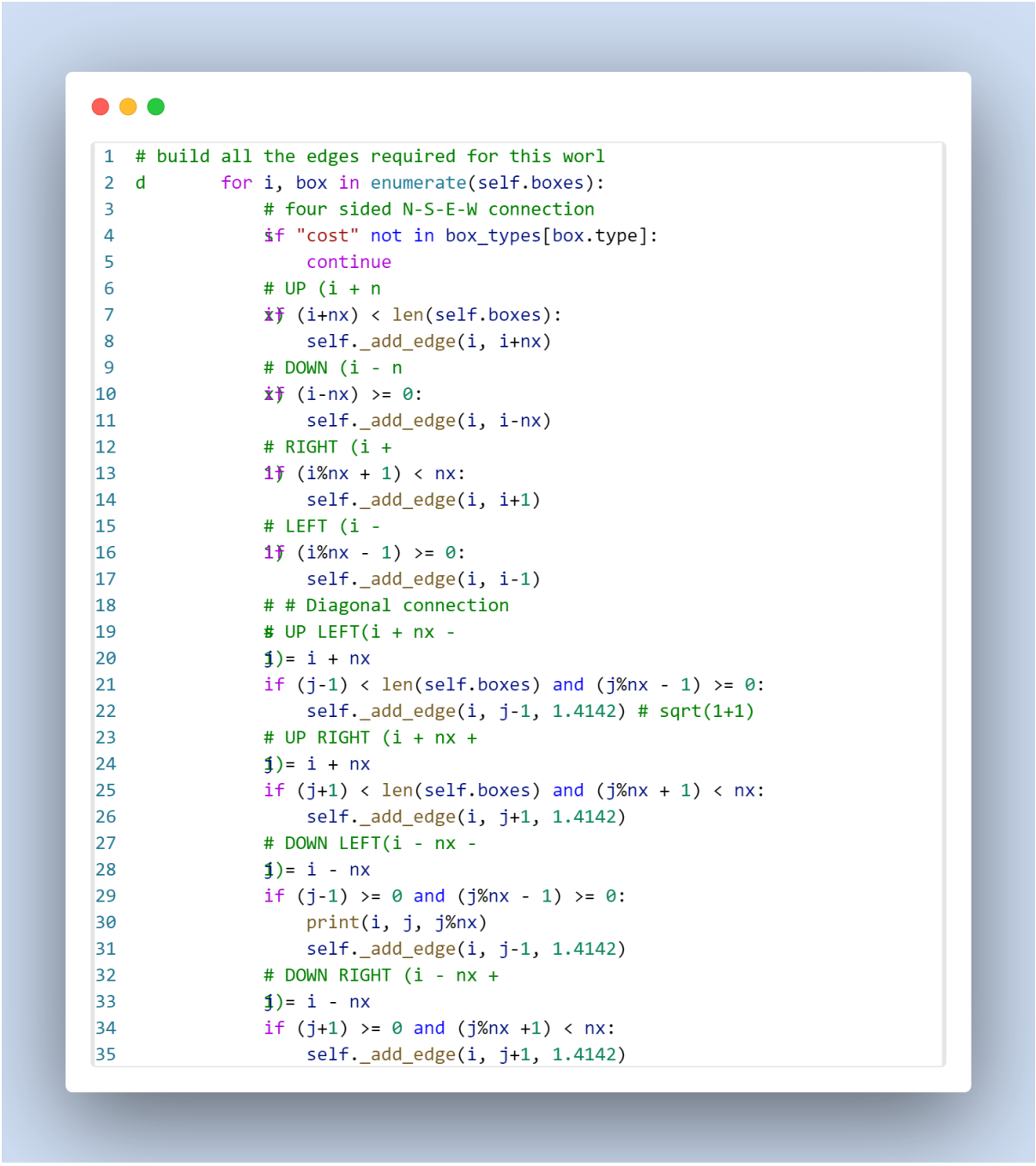
StudentID: 104180605

Lab 5:

The min_edge_cost will be change to 1 to be a min value because the Manhattan function will choose the lowest value of estimate distance to the goal.

```
min_edge_cos  = 1 # must be min value for heuristic cost to wor
t                 k
    def _manhatta (self, idx1, idx2):
        ''' Manhattan distance between two nodes in boxworld, assuming th
        minimal edge cost so that we don't overestimate the cost).
'''     x1, y1 = self.boxes[idx1].pos
        x2, y2 = self.boxes[idx2].pos
        return (abs(x1-x2) + abs(y1-y2)) * min_edge_cos
                                           t
    def _hypot(self, idx1, idx2):
        '''Return the straight line distance between two points on a 2-
        Cartesian plane. Argh, Pythagoras... trouble maker.
'''     x1, y1 = self.boxes[idx1].pos
        x2, y2 = self.boxes[idx2].pos
        return hypot(x1-x2, y1-y2) * min_edge_cos
                                     t
    def _max(self, idx1, idx2):
        '''Return the straight line distance between two points on a 2-
        Cartesian plane. Argh, Pythagoras... trouble maker.
'''     x1, y1 = self.boxes[idx1].pos
        x2, y2 = self.boxes[idx2].pos
        return max(abs(x1-x2),abs(y1-y2)) * min_edge_cos
                                            t
```

These lines of code will allow the connection of each node horizontally, vertically and diagonally:



```
# build all the edges required for this worl
d       for i, box in enumerate(self.boxes):
            # four sided N-S-E-W connection
            if "cost" not in box_types[box.type]:
                continue
            # UP (i + n
            if (i+nx) < len(self.boxes):
                self._add_edge(i, i+nx)
            # DOWN (i - n
            if (i-nx) >= 0:
                self._add_edge(i, i-nx)
            # RIGHT (i +
            if (i%nx + 1) < nx:
                self._add_edge(i, i+1)
            # LEFT (i -
            if (i%nx - 1) >= 0:
                self._add_edge(i, i-1)
            # # Diagonal connection
            # UP LEFT(i + nx -
            1)= i + nx
            if (j-1) < len(self.boxes) and (j%nx - 1) >= 0:
                self._add_edge(i, j-1, 1.4142) # sqrt(1+1)
            # UP RIGHT (i + nx +
            1)= i + nx
            if (j+1) < len(self.boxes) and (j%nx + 1) < nx:
                self._add_edge(i, j+1, 1.4142)
            # DOWN LEFT(i - nx -
            1)= i - nx
            if (j-1) >= 0 and (j%nx - 1) >= 0:
                print(i, j, j%nx)
                self._add_edge(i, j-1, 1.4142)
            # DOWN RIGHT (i - nx +
            1)= i - nx
            if (j+1) >= 0 and (j%nx +1) < nx:
                self._add_edge(i, j+1, 1.4142)
```

Explanation of 4 algorithms:

All algorithms have 4 same components and code flows but its different between each other is the use of stack frontier.

- The node will always try to explore its neighbour node if the goal node is not reached neither in closed (explored node) or in open (frontier that contains node it is going next).
- It will continue to search when the open is still contain any elements in it.

BFS:

- The open (frontier stack) used is FIFO, which mean it pop the first element that added to the frontier to move to. This allow BFS to search in breadth by checking for every neighbour of the single node.

DFS:

- The open (frontier stack) used is LIFO, which mean it pop the last element that added to the frontier to move to. This allow DFS to search in depth by checking every child node of the single node.

Dijkstra:

- The open stack used is LIFO, the functionality is same with DFS explained above.
- However, it will check the cost of the next neighbour node and choose the node with less cost calculated from the initial state to that.

A star:

- The open stack used is LIFO, the functionality is same with DFS explained above.
- The different of A star with DFS and Dijkstra is it will check the cost of path from initial state + estimated cost to the goal and choose the path with lower value.

Explanation of distance function:

Heuristic function:

- The heuristic function measure the distance of start state and goal state by the distance in X-Axis and Y-axis of abs(goal_xAxis – start_xAxis) + abs(goal_yAxis – goal_yAxis)

Hypot function:

- The Hypot function measure the distance from the start state to the goal state diagonally.
- It used Euclid method of calculating distances of two point in 2-dimentional.
  Distance of d(start,goal) = sqrt[(goal_xAxis – start_xAxis)^2 + (goal_yAxis – goal_yAxis)^2]

Max function:

- This function will measure choose a higher straight-line distance horizontally or vertically to the goal.
- It choose the higher value from the calculation of X = abs(goal_xAxis – start_xAxis) and Y = abs(goal_yAxis – goal_yAxis)