Assignment 2

CSC3206 Artificial Intelligence

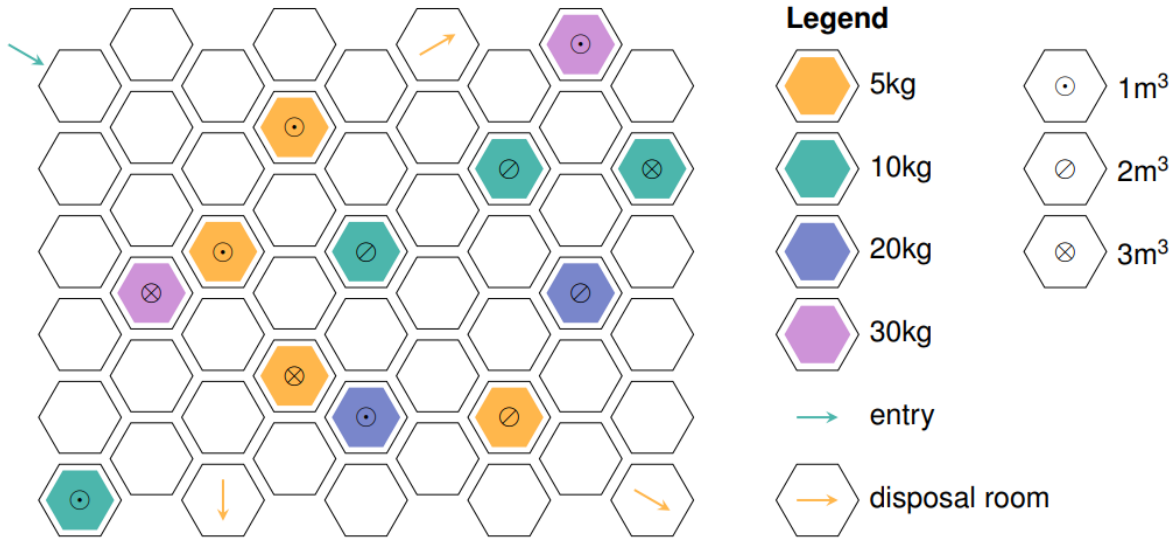
Group 13

Members:

NAME	STUDENT ID
Aisha Sofia Binti Najidi	20065231
Emily Teng Jie Qi	20054607
Justin Phang Sheng Xuan	20066502
Yap Yi Rou	20057055

Table of Contents

Problem Statement



In this report, we are to implement a search algorithm which allows Ronny to clear the rooms of rubbish in the most optimal way possible. In terms of optimality, total path cost is the main factor that is taken into consideration when deciding which search algorithm to implement.

(find solution with the shortest path, not time cause we don't know time complexity and whatnot)

(briefly explain the limitations of Ronny, just like u did before)

Before proceeding further, a few constraints are given to Ronny beforehand, which are:

1. The maximum weight of the rubbish bin is 40kg.
2. The rubbish bin can hold a total of 5 m^3 of rubbish.

When considering the algorithm that is the most optimal for the problem at hand, the term optimality is defined as the total path cost which refers to the cumulative distance or effort required for Ronny to traverse from the starting room to each rubbish location and disposal room, considering any constraints or limitations. The search algorithm should prioritize finding a path with the lowest total cost, which can be measured in terms of distance travelled or any other relevant metric.

By taking the factors listed above into consideration, we have decided to implement greedy best first search algorithm (GBFS) to solve the problem at hand.

(more assumptions pls)

Before we move on, there were a few assumptions specifically made that were not mentions in the instructions. These assumptions were made to simplify the problem and align the set characteristics of GBFS:

1. Deterministic environment: It is assumed that the environment is deterministic, which means that during the execution of the algorithm, the state of the rooms and the amount

of rubbish will not change. This presumption enables decision-making based on the status of the rooms to be consistent.

2. Locations of rubbish is fixed: It is assumed that the locations of the rubbish in every room remains consistent throughout the execution. This assumption allows for the simplification of the problem by not considering the change in rubbish locations.

Explanation on Implementation

Why is Greedy Best-First Search (GBFS) selected?

GBFS is ideal and suitable for solving this problem because it allows for a concise program to be implemented while optimizing the code for low complexity ensures optimal efficiency in terms of both time and storage utilization. Furthermore, the deterministic environment (maze) ensures that the program would not be stuck on a loopy path.

What is Greedy Best-First Search?

GFBS is an informed search algorithm that prioritizes expanding the nodes that are estimated to be closest to the goal state based on the heuristic function value. It selects the most promising nodes with minimal evaluation at each step without considering the total cost to reach the goal.

$$f(n)=h(n)$$

- *The evaluation function, f(n) is equal to the heuristic function, h(n).*

Implementation

```
class Room:
    def __init__(self, x, y, rubbish_weight=0, rubbish_size=0, disposal_room=False):
        self.x = x
        self.y = y
        self.rubbish_weight = rubbish_weight
        self.rubbish_size = rubbish_size
        self.disposal_room = disposal_room


room_coordinates = [
    (-5, 4), (-4, 4), (-3, 4), (-2, 4), (-1, 4), (0, 4),
    (-4, 3), (-3, 3), (-2, 3), (-1, 3), (0, 3), (1, 3),
    (-4, 2), (-3, 2), (-2, 2), (-1, 2), (0, 2), (1, 2),
    (-3, 1), (-2, 1), (-1, 1), (0, 1), (1, 1), (2, 1),
    (-3, 0), (-2, 0), (-1, 0), (0, 0), (1, 0), (2, 0),
    (-2, -1), (-1, -1), (0, -1), (1, -1), (2, -1), (3, -1),
    (-2, -2), (-1, -2), (0, -2), (1, -2), (2, -2), (3, -2),
    (-1, -3), (0, -3), (1, -3), (2, -3), (3, -3), (4, -3),
    (-1, -4), (0, -4), (1, -4), (2, -4), (3, -4), (4, -4)
]
```

Code 1: Class object 'Rooms' and the coordinate.

The '***Room***' object within this class represents each specific room in the maze. The '***__init__***' method initializes the variables for each room with the parameters such as x-axis, y-axis, rubbish weight, and rubbish size inside the room.

The '***room_coordinates***' represents the coordinates for all the rooms. The hexagon grids coordinate was built by using a variation of the Taxicab geometry or Manhattan geometry as shown in the following steps:

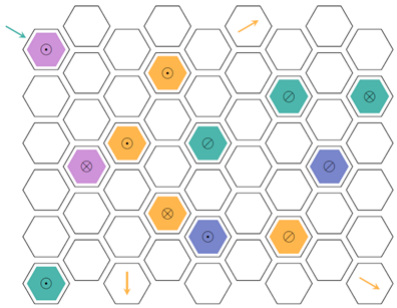


Figure 1: Original position of the grid plane.

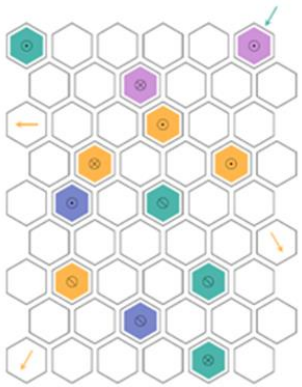


Figure 2: 90 degrees clockwise of the grid plan.

1. The plane is first rotated by 90 degrees to the right.

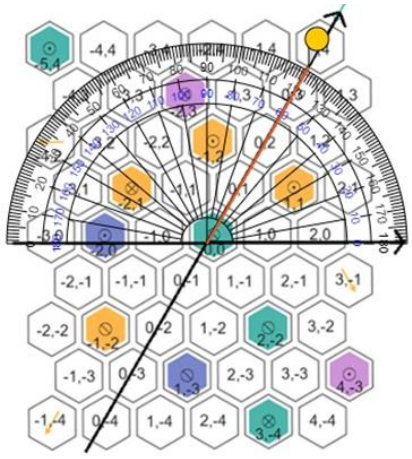


Figure 3: Cartesian plan with the protractor

2. In Euclidean geometry, a horizontal line is drawn as the x-axis while the y-axis is tilted 60-degree angle. This would prevent the odd-even row distinction problem in hexagonal grids. The coordinate of each hexagon inserts into the grids.

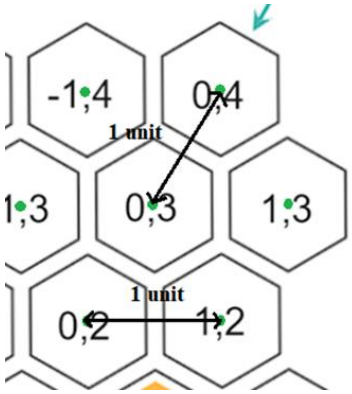


Figure 4: Horizontal spacing and vertical spacing (60 degrees).

3. The distance traveling between each node will be calculated by the spacing of hexagon centers.

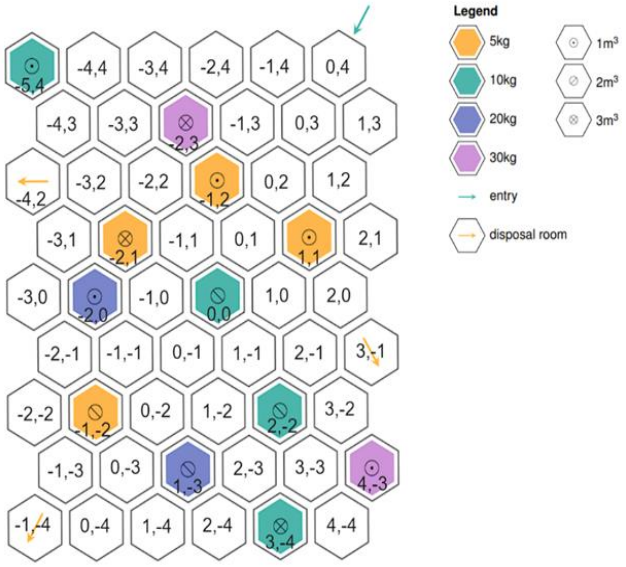


Figure 5: Complete maze coordinate map with rubbish and disposal rooms information.

```
rubbish_locations = {
    (-5, 4): (10, 1),
    (-2, 3): (30, 3),
    (-1, 2): (5, 1),
    (1, 1): (5, 1),
    (-2, 1): (5, 3),
    (0, 0): (10, 2),
    (-2, 0): (20, 1),
    (2, -2): (10, 2),
    (-1, -2): (5, 2),
    (4, -3): (30, 1),
    (1, -3): (20, 2),
    (3, -4): (10, 3),
}

disposal_rooms = [
    (-4, 2),
    (-1, -4),
    (3, -1)
]
```

Code 2: Rubbish information and disposal room's location.

The rubbish fills the rooms by following this format:

(x-axis, y-axis): (kg, m^3)

The format for the disposal rooms status is:

(x-axis, y-axis)

```
class RubbishBin:
    def __init__(self, weight_capacity, size_capacity):
        self.weight_capacity = weight_capacity
        self.size_capacity = size_capacity
        self.current_weight = 0
        self.current_size = 0

    def can_collect(self, weight, size):
        return self.current_weight + weight <= self.weight_capacity and self.current_size + size <= self.size_capacity
```

Code 3: Rubbish bin class.

The '***RubbishBin***' class represents the rubbish bin in the hand of Ronny. The parameters, '***weight_capacity***' and '***size_capacity***' initialize the maximum weight and size the rubbish bin can carry. The attributes, '***current_weight***' and '***current_size***', are initialized as 0, and it will keep track of the current weight and size of the rubbish bin.

The '***can_collect***' method is responsible to determine whether the rubbish bin can collect the rubbish without exceeding the maximum weight and size of the rubbish bin.

```
    def collect_rubbish(self, weight, size):
        self.current_weight += weight
        self.current_size += size

    def dispose_rubbish(self):
        self.current_weight = 0
        self.current_size = 0
```

Code 4: Actions (methods) apply to the rubbish bin.

To fill the rubbish in the bin, the '***collect_rubbish***' method will add the rubbish's weight and size to the bin. The '***dispose_rubbish***' method will use to clear the weight and size of the rubbish bin.

```
def find_optimal_path(start_room, rubbish_locations, disposal_rooms):
    path = []
    rubbish_bin = RubbishBin(40, 5)
    cumulative_distance = 0
```

Code 5: Find optimal path preparation.

The '***find_optimal_path***' function takes multiple values as its parameters, such as '***start_room***', '***rubbish_location***', and '***disposal_rooms***'. An empty list, '***path***' will store the optimal path found. Then, it creates and initializes a rubbish bin object with a weight capacity of 40 kg and

a size capacity of $5m^3$. It also set the cumulative distance variable to zero and start to track the total distance travelled.

```
while rubbish_locations:
    # Find the nearest room with rubbish that satisfies the weight and size constraints
    nearest_room = None
    min_distance = float('inf')

    for room_coord, rubbish in rubbish_locations.items():
        distance = distanceOf(start_room, Room(*room_coord))
        if distance < min_distance and rubbish_bin.can_collect(rubbish[0], rubbish[1]):
            min_distance = distance
            nearest_room = Room(*room_coord, rubbish_weight=rubbish[0], rubbish_size=rubbish[1])
```

Code 6: Mission to clear all the rubbish.

The '***while***' loop will end when there is no more rubbish in the rooms by referring to the '***rubbish_locations***' dictionary.

For each room, the distance to the rubbish room from the '***start_room***' is calculated by using the '***distanceOf***' function. It checks if the calculated distance is smaller than the current minimum distance and if the rubbish bin can collect the weight and size of the rubbish in that room. If these conditions are satisfied, the minimum distance will be updated, and the current room will be assigned the '***nearest_room***' with the updated rubbish weight and size.

```
    if nearest_room is not None:
        # Collect rubbish
        rubbish_bin.collect_rubbish(nearest_room.rubbish_weight, nearest_room.rubbish_size)
        cumulative_distance += min_distance
        path.append((nearest_room, cumulative_distance))
        del rubbish_locations[(nearest_room.x, nearest_room.y)]
        start_room = nearest_room
```

Code 7: When the nearest room is found.

If the '***nearest_room***' that contains a suitable weight and size of rubbish is found, it will collect the rubbish by the '***collect_rubbish***' method from the '***rubbish_bin***' object with the rubbish weight and size. It updates the '***cumulative_distance***' by adding the minimum distance found and appends the 'path' list with the room information and cumulative distance. Then, it deletes the current room from the '***rubbish_locations***' dictionary. After the rubbish has been collected and recorded its distance, the current room is assigned to the '***start_room***' to iterate the search.

```
    # Check if the rubbish bin is full or there are no more
      remaining suitable rubbish locations
    if rubbish_bin.current_weight == rubbish_bin.weight_capacity or
    rubbish_bin.current_size == rubbish_bin.size_capacity or not
    any(rubbish_bin.can_collect(weight, size) for weight, size in
    rubbish_locations.values()):
        # Dispose rubbish in the nearest disposal room
        disposal_room = get_nearest_disposal_room(start_room, disposal_rooms)
        cumulative_distance += distanceOf(start_room, disposal_room)
        path.append((disposal_room, cumulative_distance))
        rubbish_bin.dispose_rubbish()
        start_room = disposal_room

return path
```

Code 8: Time to dispose the rubbish.

After the iteration, the program will check if the rubbish bin is fully filled by checking the maximum weight or size capacity, or if there is no more suitable rubbish location available. If so, it will find the nearest disposal room by the '***get_nearest_disposal_room***' function. The

cumulative distance will update by the distance between the current room and the disposal room calculated by the '***distanceOf***' function. In addition, the '***path***' list will be updated with the room information and cumulative distance. After it calls the '***dispose_rubbish***' function to dispose of the rubbish, the disposal room will be the next '***current_room***' in the next iteration.

Once all the rubbish has been collected and disposed of, the '***while***' loop will end, and the '***path***' list that contains the optimal path and cumulative distances will be returned.

```
def distanceOf(current_room, goal_room):
    dx = goal_room.x - current_room.x
    dy = goal_room.y - current_room.y

    if (dx >= 0 and dy >= 0) or (dx < 0 and dy < 0):
        return abs(dx + dy)
    else:
        return max(abs(dx), abs(dy))
```

Code 9: Calculate the Manhattan distance between two rooms.

The '***distanceOf***' function calculates the distance between '***current_room***' and '***goal_room***' (refer to Figure 4). The '***dx***' will find the difference between the '***x***' coordinates while the '***dy***' will get the difference between the '***y***' coordinates.

The if-else statement checks the relative position of the two rooms. When both '***dx***' and '***dy***' are both positive or both negative, the distance is calculated as the sum of the absolute differences '***abs(dx + dy0)***'. Otherwise, if '***dx***' and '***dy***' have different signs, the distance will be calculated by the maximum absolute difference '***max(abs(dx), abs(dy))***'.

The function returns the Manhattan distance calculated.

```
def get_nearest_disposal_room(current_room, disposal_rooms):
    min_distance = float('inf')
    nearest_disposal_room = None

    for disposal_room_coordinates in disposal_rooms:
        disposal_room = Room(*disposal_room_coordinates, disposal_room=True)
        distance = distanceOf(current_room, disposal_room)
        if distance < min_distance:
            min_distance = distance
            nearest_disposal_room = disposal_room

    return nearest_disposal_room
```

Code 10: Find the nearest disposal room.

The '***get_nearest_disposal_room***' function finds the disposal room that is nearest to the current room from the '***disposal_rooms***' dictionary. It initializes the minimum distance to infinity and '***nearest_disposal_room***' to none to ensure that the variables can be easily overwritten.

It iterates over each disposal room and creates a '***disposal_room***' object using the '***Room***' class to pass the coordinates and setting the disposal room attribute to '***True***' for each disposal room's coordinates. The '***distanceOf***' function is used again to calculate the distance between the current room and the disposal room.

If the calculated distance is smaller than the current minimum distance, it updates the new minimum distance and assigns the current disposal room as the '***nearest_disposal_room***'.

After searching for every disposal room, the function returns the nearest disposal room.

```
# Define the starting room
start_room = Room(0, 4)

# Find the optimal path
path = find_optimal_path(start_room, rubbish_locations, disposal_rooms)
```

Code 11: Initializing the starting information.

The '***start_room***' creates with coordinates (0, 4) in the 'Room' class.

The next line calls the '***find_optimal_path***' function with the rooms' information to calculate the optimal path to complete the mission. The results are assigned to the variable '***path***'.

```
print("Start at ({}, {}) [Bin Weight: 0 kg, Bin Size: 0 m^3], Cumulative Distance: 0 units".format(start_room.x, start_room.y))
print()

# Print the optimal path with rubbish bin weight and size
rubbish_bin = RubbishBin(40, 5)
for room, cumulative_distance in path:
    if room.disposal_room:
        print("Dispose rubbish at ({}, {}) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: {} units"
              .format(room.x, room.y, cumulative_distance))
        rubbish_bin.dispose_rubbish()
        print()
    else:
        rubbish_bin.collect_rubbish(room.rubbish_weight, room.rubbish_size)
        print("Collect rubbish at ({}, {}) [Bin Weight: {} kg, Rubbish Size: {} m^3], Cumulative Distance: {} units"
              .format(room.x, room.y, rubbish_bin.current_weight, rubbish_bin.current_size, cumulative_distance))
        print()
```

Code 12: Progress and result output

The first line prints the initial state of the starting room. Next, it iterates over all the rooms in the optimal path. If the room is a disposal room, it prints a message representing the disposal action after setting the rubbish bin weight and size to 0 while keeping the cumulative distance. Otherwise, if the room is a rubbish location, it collects the rubbish by the '***collect_rubbish***' function and prints out a message representing the process.

Discussions on different configurations of the problem

With respect to the current working code we have, there are a few scenarios in which we would have to consider modifying it to better fit the changing scenarios.

Scenario 1: Adding/Removing Rubbish in Different Rooms

If we want to add or remove rubbish from different rooms in the grid, we can update the ***rubbish_locations*** dictionary to include or remove the respective room coordinates and rubbish weight/size information. In this scenario, we modify the ***rubbish_locations*** dictionary to have different rubbish locations in the maze.

Initial code:

```
22  rubbish_locations = {
23      (-5, 4): (10, 1),
24      (-2, 3): (30, 3),
25      (-1, 2): (5, 1),
26      (1, 1): (5, 1),
27      (-2, 1): (5, 3),
28      (0, 0): (10, 2),
29      (-2, 0): (20, 1),
30      (2, -2): (10, 2),
31      (-1, -2): (5, 2),
32      (4, -3): (30, 1),
33      (1, -3): (20, 2),
34      (3, -4): (10, 3),
35  }
```

Code 13: Initial ' rubbish_locations' code

Initial code results:

```
Start at (0, 4) [Bin Weight: 0 kg, Bin Size: 0 m^3], Cumulative Distance: 0 units
Collect rubbish at (-2, 3) [Bin Weight: 30 kg, Rubbish Size: 3 m^3], Cumulative Distance: 3 units
Collect rubbish at (-1, 2) [Bin Weight: 35 kg, Rubbish Size: 4 m^3], Cumulative Distance: 4 units
Collect rubbish at (1, 1) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 6 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 8 units
Collect rubbish at (2, -2) [Bin Weight: 10 kg, Rubbish Size: 2 m^3], Cumulative Distance: 10 units
Collect rubbish at (0, 0) [Bin Weight: 20 kg, Rubbish Size: 4 m^3], Cumulative Distance: 12 units
Collect rubbish at (-2, 0) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 14 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 16 units
Collect rubbish at (-5, 4) [Bin Weight: 10 kg, Rubbish Size: 1 m^3], Cumulative Distance: 18 units
Collect rubbish at (-2, 1) [Bin Weight: 15 kg, Rubbish Size: 4 m^3], Cumulative Distance: 21 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 23 units
Collect rubbish at (-1, -2) [Bin Weight: 5 kg, Rubbish Size: 2 m^3], Cumulative Distance: 27 units
Collect rubbish at (1, -3) [Bin Weight: 25 kg, Rubbish Size: 4 m^3], Cumulative Distance: 29 units
Dispose rubbish at (-1, -4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 32 units
Collect rubbish at (3, -4) [Bin Weight: 10 kg, Rubbish Size: 3 m^3], Cumulative Distance: 36 units
Collect rubbish at (4, -3) [Bin Weight: 40 kg, Rubbish Size: 4 m^3], Cumulative Distance: 38 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 40 units
```

Code 14: Initial Results

Modified code:

```
rubbish_locations = {
    (-5, 4): (10, 1),
    (-2, 3): (30, 3),
    (-1, 2): (5, 1),
    (1, 1): (5, 1),
    (-2, 1): (5, 3),
    (0, 0): (10, 2),
    (-2, 0): (20, 1),
    (2, -2): (10, 2),
    (-1, -2): (5, 2),
    (4, -3): (30, 1),
    (1, -3): (20, 2),
    (3, -4): (10, 3),
    (2, 1): (10, 1),  # New rubbish location
    (-3, -1): (5, 2)  # New rubbish location
}
```

Code 14: Modified 'rubbish_locations' code

Modified code results:

```
Start at (0, 4) [Bin Weight: 0 kg, Bin Size: 0 m^3], Cumulative Distance: 0 units
Collect rubbish at (-2, 3) [Bin Weight: 30 kg, Rubbish Size: 3 m^3], Cumulative Distance: 3 units
Collect rubbish at (-1, 2) [Bin Weight: 35 kg, Rubbish Size: 4 m^3], Cumulative Distance: 4 units
Collect rubbish at (1, 1) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 6 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 8 units
Collect rubbish at (2, -2) [Bin Weight: 10 kg, Rubbish Size: 2 m^3], Cumulative Distance: 10 units
Collect rubbish at (0, 0) [Bin Weight: 20 kg, Rubbish Size: 4 m^3], Cumulative Distance: 12 units
Collect rubbish at (-2, 0) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 14 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 16 units
Collect rubbish at (-5, 4) [Bin Weight: 10 kg, Rubbish Size: 1 m^3], Cumulative Distance: 18 units
Collect rubbish at (-2, 1) [Bin Weight: 15 kg, Rubbish Size: 4 m^3], Cumulative Distance: 21 units
Collect rubbish at (2, 1) [Bin Weight: 25 kg, Rubbish Size: 5 m^3], Cumulative Distance: 25 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 27 units
Collect rubbish at (4, -3) [Bin Weight: 30 kg, Rubbish Size: 1 m^3], Cumulative Distance: 29 units
Collect rubbish at (3, -4) [Bin Weight: 40 kg, Rubbish Size: 4 m^3], Cumulative Distance: 31 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 34 units
Collect rubbish at (1, -3) [Bin Weight: 20 kg, Rubbish Size: 2 m^3], Cumulative Distance: 38 units
Collect rubbish at (-1, -2) [Bin Weight: 25 kg, Rubbish Size: 4 m^3], Cumulative Distance: 40 units
Dispose rubbish at (-1, -4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 42 units
Collect rubbish at (-3, -1) [Bin Weight: 5 kg, Rubbish Size: 2 m^3], Cumulative Distance: 45 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 48 units
```

Code 16: Modified Results

Explanation

As seen in the modified code, we added new rubbish locations to the list, resulting in changing the path that the robot needs to take making the cumulative distance go from 40 units to 48 units.

Scenario 2: Changing Disposal Rooms' Locations

If we want to change the locations of the disposal rooms, we need to modify the ***disposal_rooms*** list to reflect the new coordinates of the disposal rooms. The code will then find the nearest disposal room based on the updated coordinates.

Initial code:

```
disposal_rooms = [
    (-4, 2),
    (-1, -4),
    (3, -1)
]
```

Code 17: Initial 'disposal_rooms' code

Initial code results:

```
Start at (0, 4) [Bin Weight: 0 kg, Bin Size: 0 m^3], Cumulative Distance: 0 units
Collect rubbish at (-2, 3) [Bin Weight: 30 kg, Rubbish Size: 3 m^3], Cumulative Distance: 3 units
Collect rubbish at (-1, 2) [Bin Weight: 35 kg, Rubbish Size: 4 m^3], Cumulative Distance: 4 units
Collect rubbish at (1, 1) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 6 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 8 units
Collect rubbish at (2, -2) [Bin Weight: 10 kg, Rubbish Size: 2 m^3], Cumulative Distance: 10 units
Collect rubbish at (0, 0) [Bin Weight: 20 kg, Rubbish Size: 4 m^3], Cumulative Distance: 12 units
Collect rubbish at (-2, 0) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 14 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 16 units
Collect rubbish at (-5, 4) [Bin Weight: 10 kg, Rubbish Size: 1 m^3], Cumulative Distance: 18 units
Collect rubbish at (-2, 1) [Bin Weight: 15 kg, Rubbish Size: 4 m^3], Cumulative Distance: 21 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 23 units
Collect rubbish at (-1, -2) [Bin Weight: 5 kg, Rubbish Size: 2 m^3], Cumulative Distance: 27 units
Collect rubbish at (1, -3) [Bin Weight: 25 kg, Rubbish Size: 4 m^3], Cumulative Distance: 29 units
Dispose rubbish at (-1, -4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 32 units
Collect rubbish at (3, -4) [Bin Weight: 10 kg, Rubbish Size: 3 m^3], Cumulative Distance: 36 units
Collect rubbish at (4, -3) [Bin Weight: 40 kg, Rubbish Size: 4 m^3], Cumulative Distance: 38 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 40 units
```

Code 18: Initial Results

Modified code:

```
37  disposal_rooms = [
38      (-4, 2),
39      (-1, -4),
40      (3, -1),
41      (-3, 3),  # New disposal room location
42      (0, -2)  # New disposal room location
43  ]
```

Code 19: Modified 'disposal_rooms' code

Modified code results:

```
Start at (0, 4) [Bin Weight: 0 kg, Bin Size: 0 m^3], Cumulative Distance: 0 units
Collect rubbish at (-2, 3) [Bin Weight: 30 kg, Rubbish Size: 3 m^3], Cumulative Distance: 3 units
Collect rubbish at (-1, 2) [Bin Weight: 35 kg, Rubbish Size: 4 m^3], Cumulative Distance: 4 units
Collect rubbish at (1, 1) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 6 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 8 units
Collect rubbish at (2, -2) [Bin Weight: 10 kg, Rubbish Size: 2 m^3], Cumulative Distance: 10 units
Collect rubbish at (0, 0) [Bin Weight: 20 kg, Rubbish Size: 4 m^3], Cumulative Distance: 12 units
Collect rubbish at (-2, 0) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 14 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 16 units
Collect rubbish at (-5, 4) [Bin Weight: 10 kg, Rubbish Size: 1 m^3], Cumulative Distance: 18 units
Collect rubbish at (-2, 1) [Bin Weight: 15 kg, Rubbish Size: 4 m^3], Cumulative Distance: 21 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 23 units
Collect rubbish at (-1, -2) [Bin Weight: 5 kg, Rubbish Size: 2 m^3], Cumulative Distance: 27 units
Collect rubbish at (1, -3) [Bin Weight: 25 kg, Rubbish Size: 4 m^3], Cumulative Distance: 29 units
Dispose rubbish at (0, -2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 30 units
Collect rubbish at (3, -4) [Bin Weight: 10 kg, Rubbish Size: 3 m^3], Cumulative Distance: 33 units
Collect rubbish at (4, -3) [Bin Weight: 40 kg, Rubbish Size: 4 m^3], Cumulative Distance: 35 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 37 units
```

Code 20: Modified Results

Explanation

As seen in the modified code, we added new disposal rooms locations to the list, resulting in changing the path that the robot needs to take making the cumulative distance go from 40 units to 37 units.

Scenario 3: Randomised Rubbish Locations

The next scenario involves dynamically changing the location of the rubbish. This involves periodically updating the '***rubbish_locations'*** dictionary with the latest positions and quantities of rubbish. Additionally, the '***find_optimal_path'*** function is modified to consider the updated rubbish locations while making decisions about the nearest room to visit.

Initial code results:

```
Start at (0, 4) [Bin Weight: 0 kg, Bin Size: 0 m^3], Cumulative Distance: 0 units
Collect rubbish at (-2, 3) [Bin Weight: 30 kg, Rubbish Size: 3 m^3], Cumulative Distance: 3 units
Collect rubbish at (-1, 2) [Bin Weight: 35 kg, Rubbish Size: 4 m^3], Cumulative Distance: 4 units
Collect rubbish at (1, 1) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 6 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 8 units
Collect rubbish at (2, -2) [Bin Weight: 10 kg, Rubbish Size: 2 m^3], Cumulative Distance: 10 units
Collect rubbish at (0, 0) [Bin Weight: 20 kg, Rubbish Size: 4 m^3], Cumulative Distance: 12 units
Collect rubbish at (-2, 0) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 14 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 16 units
Collect rubbish at (-5, 4) [Bin Weight: 10 kg, Rubbish Size: 1 m^3], Cumulative Distance: 18 units
Collect rubbish at (-2, 1) [Bin Weight: 15 kg, Rubbish Size: 4 m^3], Cumulative Distance: 21 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 23 units
Collect rubbish at (-1, -2) [Bin Weight: 5 kg, Rubbish Size: 2 m^3], Cumulative Distance: 27 units
Collect rubbish at (1, -3) [Bin Weight: 25 kg, Rubbish Size: 4 m^3], Cumulative Distance: 29 units
Dispose rubbish at (-1, -4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 32 units
Collect rubbish at (3, -4) [Bin Weight: 10 kg, Rubbish Size: 3 m^3], Cumulative Distance: 36 units
Collect rubbish at (4, -3) [Bin Weight: 40 kg, Rubbish Size: 4 m^3], Cumulative Distance: 38 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 40 units
```

Code 21: Initial Code results

Modified code:

```
1   import random
```

Code 22: Modified Code

The **import random** statement is added to import the random module from the Python standard library.

```
rubbish_locations = [
    (10, 1, -5, 4),
    (30, 3, -2, 3),
    (5, 1, -1, 2),
    (5, 1, 1, 1),
    (5, 3, -2, 1),
    (10, 2, 0, 0),
    (20, 1, -2, 0),
    (10, 2, 2, -2),
    (5, 2, -1, -2),
    (30, 1, 4, -3),
    (20, 2, 1, -3),
    (10, 3, 3, -4),
]
```

Code 23: Modified rubbish_locations code

To overcome the issue of Python dictionaries not guaranteeing specific order, meaning the randomization of keys may not result in different locations each time you run the code we updated the '***rubbish_locations'***.

In this code, the '***rubbish_locations'*** list is shuffled using the ***random.shuffle()*** function before starting the path finding. This ensures that the order of rubbish locations is randomized each time you run the code.

```
def find_optimal_path(start_room, rubbish_locations, disposal_rooms):
    random.shuffle(rubbish_locations)  # Shuffle the rubbish locations randomly
    rubbish_bin = RubbishBin(40, 5)
    path = []
    cumulative_distance = 0

    def distance_of(room1, room2):
        return abs(room1.x - room2.x) + abs(room1.y - room2.y)

    def get_nearest_room(start, locations):
        min_distance = float('inf')
        nearest_room = None

        for room in locations:
            room_object = Room(room[2], room[3], rubbish_weight=room[0], rubbish_size=room[1])
            dist = distance_of(start, room_object)
            if dist < min_distance:
                min_distance = dist
                nearest_room = room_object

        return nearest_room, min_distance

    def get_nearest_disposal_room(start, rooms):
        min_distance = float('inf')
        nearest_room = None

        for room in rooms:
            room_object = Room(room[0], room[1], disposal_room=True)
            dist = distance_of(start, room_object)
            if dist < min_distance:
                min_distance = dist
                nearest_room = room_object

        return nearest_room

```

Code 24: Modified distance_of, and get_nearest_room code

In the modified code, we made changes to the '***distance_of*** 'function and the '***get_nearest_room'*** function to accommodate the changes in the '***rubbish_location's*** data structure.

Before, the code was designed to work with the Room objects that had **x** and **y** attributes to calculate distances between rooms. However, in the updated '***rubbish_locations'*** list, the room information is stored as tuples instead of Room objects.

To make the code work with the new data structure, we made changes to the '***get_nearest_room***' function. In the modified version, it now creates Room objects using the relevant information extracted from the tuples. We adjusted the arguments passed to the Room constructor: **room[0]** is the rubbish weight, **room[1]** is the rubbish size, **room[2]** is the x-coordinate, and **room[3]** is the y-coordinate.

Modified code results:

Run 1:

```
Start at (0, 4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 0 units
Collect rubbish at (-1, 2) [Bin Weight: 5 kg, Rubbish Size: 1 m^3], Cumulative Distance: 3 units
Collect rubbish at (-2, 1) [Bin Weight: 10 kg, Rubbish Size: 4 m^3], Cumulative Distance: 5 units
Collect rubbish at (-2, 0) [Bin Weight: 30 kg, Rubbish Size: 5 m^3], Cumulative Distance: 6 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 10 units
Collect rubbish at (-2, 3) [Bin Weight: 30 kg, Rubbish Size: 3 m^3], Cumulative Distance: 13 units
Collect rubbish at (-5, 4) [Bin Weight: 40 kg, Rubbish Size: 4 m^3], Cumulative Distance: 17 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 20 units
Collect rubbish at (1, 1) [Bin Weight: 5 kg, Rubbish Size: 1 m^3], Cumulative Distance: 26 units
Collect rubbish at (0, 0) [Bin Weight: 15 kg, Rubbish Size: 3 m^3], Cumulative Distance: 28 units
Collect rubbish at (-1, -2) [Bin Weight: 20 kg, Rubbish Size: 5 m^3], Cumulative Distance: 31 units
Dispose rubbish at (-1, -4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 33 units
Collect rubbish at (1, -3) [Bin Weight: 20 kg, Rubbish Size: 2 m^3], Cumulative Distance: 36 units
Collect rubbish at (2, -2) [Bin Weight: 30 kg, Rubbish Size: 4 m^3], Cumulative Distance: 38 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 40 units
Collect rubbish at (3, -4) [Bin Weight: 10 kg, Rubbish Size: 3 m^3], Cumulative Distance: 43 units
Collect rubbish at (4, -3) [Bin Weight: 40 kg, Rubbish Size: 4 m^3], Cumulative Distance: 45 units
```

Code 25: Modified Results 1

Run 2:

```
Start at (0, 4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 0 units
Collect rubbish at (-2, 3) [Bin Weight: 30 kg, Rubbish Size: 3 m^3], Cumulative Distance: 3 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 6 units
Collect rubbish at (-2, 1) [Bin Weight: 5 kg, Rubbish Size: 3 m^3], Cumulative Distance: 9 units
Collect rubbish at (-2, 0) [Bin Weight: 25 kg, Rubbish Size: 4 m^3], Cumulative Distance: 10 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 14 units
Collect rubbish at (-5, 4) [Bin Weight: 10 kg, Rubbish Size: 1 m^3], Cumulative Distance: 17 units
Collect rubbish at (-1, 2) [Bin Weight: 15 kg, Rubbish Size: 2 m^3], Cumulative Distance: 23 units
Collect rubbish at (1, 1) [Bin Weight: 20 kg, Rubbish Size: 3 m^3], Cumulative Distance: 26 units
Collect rubbish at (0, 0) [Bin Weight: 30 kg, Rubbish Size: 5 m^3], Cumulative Distance: 28 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 32 units
Collect rubbish at (2, -2) [Bin Weight: 10 kg, Rubbish Size: 2 m^3], Cumulative Distance: 34 units
Collect rubbish at (1, -3) [Bin Weight: 30 kg, Rubbish Size: 4 m^3], Cumulative Distance: 36 units
Dispose rubbish at (-1, -4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 39 units
Collect rubbish at (-1, -2) [Bin Weight: 5 kg, Rubbish Size: 2 m^3], Cumulative Distance: 41 units
Collect rubbish at (3, -4) [Bin Weight: 15 kg, Rubbish Size: 5 m^3], Cumulative Distance: 47 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 50 units
Collect rubbish at (4, -3) [Bin Weight: 30 kg, Rubbish Size: 1 m^3], Cumulative Distance: 53 units
```

Code 26: Modified Results 2

Run 3:

```
Start at (0, 4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 0 units

Collect rubbish at (-1, 2) [Bin Weight: 5 kg, Rubbish Size: 1 m^3], Cumulative Distance: 3 units

Collect rubbish at (-2, 3) [Bin Weight: 35 kg, Rubbish Size: 4 m^3], Cumulative Distance: 5 units

Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 8 units

Collect rubbish at (-5, 4) [Bin Weight: 10 kg, Rubbish Size: 1 m^3], Cumulative Distance: 11 units

Collect rubbish at (-2, 1) [Bin Weight: 15 kg, Rubbish Size: 4 m^3], Cumulative Distance: 17 units

Collect rubbish at (-2, 0) [Bin Weight: 35 kg, Rubbish Size: 5 m^3], Cumulative Distance: 18 units

Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 22 units

Collect rubbish at (1, 1) [Bin Weight: 5 kg, Rubbish Size: 1 m^3], Cumulative Distance: 28 units

Collect rubbish at (0, 0) [Bin Weight: 15 kg, Rubbish Size: 3 m^3], Cumulative Distance: 30 units

Collect rubbish at (-1, -2) [Bin Weight: 20 kg, Rubbish Size: 5 m^3], Cumulative Distance: 33 units

Dispose rubbish at (-1, -4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 35 units

Collect rubbish at (1, -3) [Bin Weight: 20 kg, Rubbish Size: 2 m^3], Cumulative Distance: 38 units

Collect rubbish at (2, -2) [Bin Weight: 30 kg, Rubbish Size: 4 m^3], Cumulative Distance: 40 units

Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 42 units

Collect rubbish at (4, -3) [Bin Weight: 30 kg, Rubbish Size: 1 m^3], Cumulative Distance: 45 units

Collect rubbish at (3, -4) [Bin Weight: 40 kg, Rubbish Size: 4 m^3], Cumulative Distance: 47 units
```

Code 27: Modified Results 3

Explanation:

As seen in Code 24, 25, and 26, the results of this algorithm vary from 45, 53, and 47 units with each run due to the fact that the rubbish locations are being randomised, forcing the algorithm to find and calculate new paths to complete the maze.

Results:

The figure below shows the starting node, its current node, its current weight, the size, and the cumulative distance.

```
Start at (0, 4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 0 units
Collect rubbish at (-2, 3) [Bin Weight: 30 kg, Rubbish Size: 3 m^3], Cumulative Distance: 3 units
Collect rubbish at (-1, 2) [Bin Weight: 35 kg, Rubbish Size: 4 m^3], Cumulative Distance: 4 units
Collect rubbish at (1, 1) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 6 units
Collect rubbish at (1, -3) [Bin Weight: 25 kg, Rubbish Size: 4 m^3], Cumulative Distance: 29 units
Dispose rubbish at (-1, -4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 32 units
Collect rubbish at (3, -4) [Bin Weight: 10 kg, Rubbish Size: 3 m^3], Cumulative Distance: 36 units
Collect rubbish at (4, -3) [Bin Weight: 40 kg, Rubbish Size: 4 m^3], Cumulative Distance: 38 units
Collect rubbish at (-2, 0) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 14 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 16 units
Collect rubbish at (-5, 4) [Bin Weight: 10 kg, Rubbish Size: 1 m^3], Cumulative Distance: 18 units
Collect rubbish at (-2, 1) [Bin Weight: 15 kg, Rubbish Size: 4 m^3], Cumulative Distance: 21 units
Dispose rubbish at (-4, 2) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 23 units
Collect rubbish at (-1, -2) [Bin Weight: 5 kg, Rubbish Size: 2 m^3], Cumulative Distance: 27 units
Collect rubbish at (1, -3) [Bin Weight: 25 kg, Rubbish Size: 4 m^3], Cumulative Distance: 29 units
Dispose rubbish at (-1, -4) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 32 units
Collect rubbish at (3, -4) [Bin Weight: 10 kg, Rubbish Size: 3 m^3], Cumulative Distance: 36 units
Collect rubbish at (4, -3) [Bin Weight: 40 kg, Rubbish Size: 4 m^3], Cumulative Distance: 38 units
Dispose rubbish at (3, -1) [Bin Weight: 0 kg, Rubbish Size: 0 m^3], Cumulative Distance: 40 units
```

Figure 1

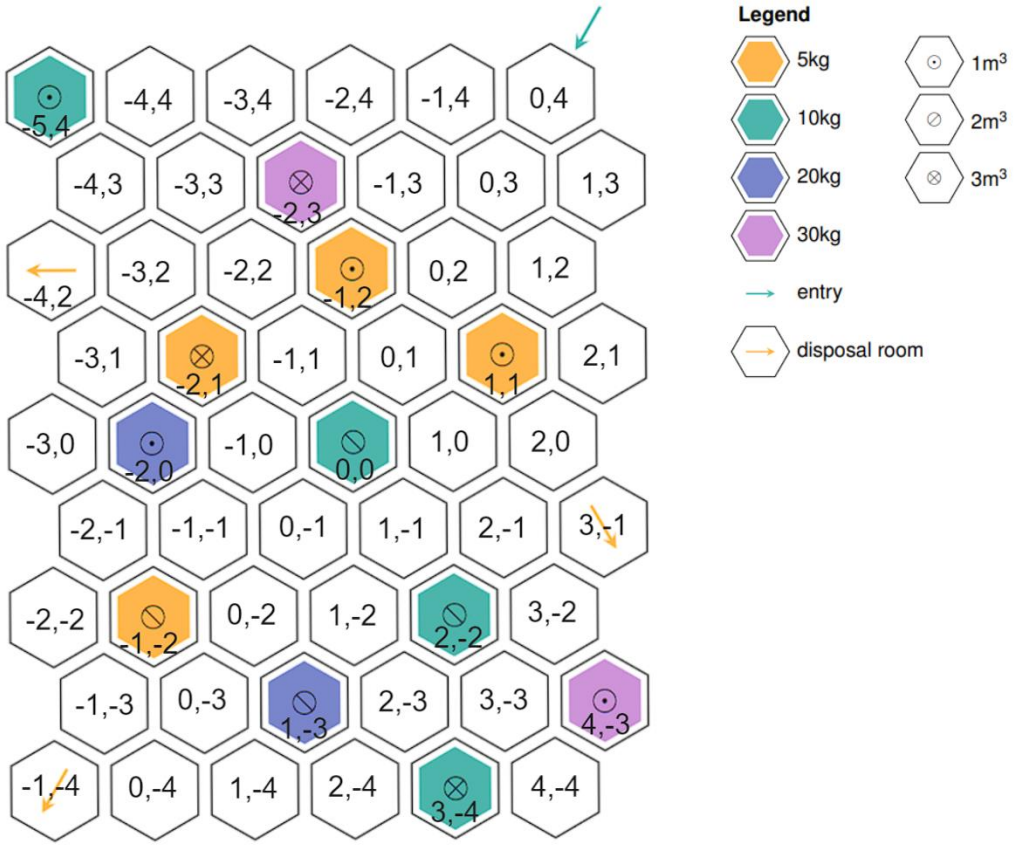


Figure 2

In Figure 1, the starting point is at (0, 4) where the initial bin weight and rubbish bin is 0 kg and 0m^3, and the cumulative distance is 0 units.

```
Collect rubbish at (-2, 3) [Bin Weight: 30 kg, Rubbish Size: 3 m^3], Cumulative Distance: 3 units

Collect rubbish at (-1, 2) [Bin Weight: 35 kg, Rubbish Size: 4 m^3], Cumulative Distance: 4 units

Collect rubbish at (1, 1) [Bin Weight: 40 kg, Rubbish Size: 5 m^3], Cumulative Distance: 6 units
```

The reason Ronny went to point (-2, 3) first rather than point (-1, 2) and (1, 1) even though they have the same travel distance from the starting point is due to the fact that in the code, point (-2, 3) is listed before (-1, 2) and (1, 1).

After collecting the rubbish at (-1, 2) [Current Bin Weight: 35kg, Rubbish Size: 3m^3], Ronny is given two options, either collect the rubbish from (1,1) which will result in a bin weight of 40kg and rubbish size of 5m^3, or (-2, 1) which will result in a bin weight of 40kg and rubbish size of 6m^3 as they have the same distance from its current point. Due to one of the constraints being that the max weight capacity of the rubbish bin is 40kg and the max rubbish size being 5m^3, the algorithm has decided that Ronny shall proceed to (1, 1) to not exceed the capacity of the rubbish bin.

Discussion:

The GBFS algorithm is best suited for Ronny's task of cleaning up rooms where speed of completion is the main concern. The benefits of implementing GBFS in this particular problem is that GBFS prioritizes the most promising paths based on the heuristic function. The prioritizing of proximity allows GBFS to speed up the completion of tasks, thus resulting in a faster turnaround time when compared to A* algorithm if the heuristic values are accurate and there are no loopy paths present.

Since GBFS is heavily dependent on the quality and accuracy of the heuristic values, issues may occur if the heuristic function is overestimated or underestimated. This will lead GBFS to produce inefficient decisions. However, such issue will not occur as there is no blocked rooms in the map which will not restrict the movement of Ronny which ultimately affects the decision-making process and the distance between each room is exactly 1 unit. Furthermore, the formula used, which is '***distanceOf***' function, allows for accurate calculations of distance, thereby allowing accurate heuristic values.

In terms of Ronny's circumstance, the most optimal solution takes precedence over the requirement for the need for quicker completion time. Ronny can move through the rooms quickly and remove trash with the fewest delays by using GBFS. Given that the project's main goal is to produce the most efficiency and optimal way to solve the issue, the trade-off in terms of speed is deemed acceptable.

In summary, by adopting GBFS as the solution for Ronny's case, Ronny can efficiently move through the rooms by utilising the heuristic guidance provided by GBFS, which ultimately leads to quicker garbage removal.