Faculty of Media Engineering Technology

AI Project 1 Report

1- Heejin Yun (43-9876)
2- Elhossin Rehan(46-8761)
3- Hossam Mohamed (46-17614)

In project 1, we are a member of the coast guard force in charge of a rescue boat that goes into the sea to rescue other sinking ships. The area in the sea that we navigate is an m x n grid of cells where m corresponds to the number of columns and n corresponds to the number of rows. We as coast guard can perform 4 actions.

1. Pick-up the passengers from the sinking ship as much as possible depending on the capacity of the coast guard boat.
2. Drop all the passengers that have been picked up from the sinking ship to the nearby station.
3. Retrieve the black box.
4. Movement in any of the 4 directions (up, down, left, right) within the grid boundaries.

Our code consists of two different classes. 1. tNode class 2. sProblem.

The goal is achieved when we are on a station and have no more ships to save. And no more blackboxes to retrieve.

Each node represents the grid with the current location of the agent and changes in the number of the passengers and the blackbox. Number of passengers carried in the initial node (root) is set to zero and the initial blackbox carried is also zero.

Number of passengers and blackbox carried by initial node (root) is set to zero and also the number of deaths is initially zero.

Each node has a hashtable for ships , hash table for stations and a hashtable for blackboxes.
With blackboxes initially set to 0.
And each has a tuple as their key being their x and y coordinates.

The constructor takes a string s and splits it by the character “;” and with the 0th element in the array created from the split being the number of columns and number of rows of the grid.

The 1st element is the maximum passengers our agent can carry. And the 2nd being the co-ordinates of the agent's start location.
3rd being the station's location. And the 4th is the ship's location and their number of passengers.
Then there was a fifth and a 6th we created / added where it stores information about the blackboxes and their health and their locations.
That was the 6th as for the 5th it carried information about the number of passengers carried thus far.

We had a couple of helper functions ,

```
public synchronized  void retrieve(){
    //if the coordinates are return a blackbox then remove it from the blackbox hashmap
    String coordinates = this.co_ordinates[0] + "," + this.co_ordinates[1];
    if (this.blackboxes.containsKey(coordinates)) {
        this.blackboxes.remove(coordinates);
        this.boxescarried++;
        this.sequence += "retrieve,";
    }
}
```

Where retrieval was supposed to be a part of the action method where it allows us to carry a box if we were on a box location and if we are to remove it , increment the number of boxes carried.
Add retrieve to the output sequence of the node. And remove the blackbox from the hashmap representing the blackboxes.

```
public void performAction()
{
    //check if I am in a station
    this.drop();
    //Check if the node is containing a ship
    this.pickup();
    //Check if the node is in a blackbox cell
    this.retrieve();
}
```

Where this was the performance action class. Where we call first drop to check if we are on a station then we drop passengers and rest the number of passengers using the drop method

```
public void drop(){
    String stationString = this.co_ordinates[0] + "," + this.co_ordinates[1];
    //if I am on a station then drop off the passengers
    if(this.stations.containsKey(stationString) == true){
```

```
            this.passengers_carried = 0;
            this.sequence += "drop,";
        }
}
```

Where it simply checks if the position we are in right now is going to return a value in the hashtable of stations if it is then simply rest the passengers carried variable and then increment the sequence string by drop

Next was the pickup function

```
public void pickup(){
    String shipString = this.co_ordinates[0] + "," + this.co_ordinates[1];
    //if I am on a ship then carry the passengers
    if(this.ships.containsKey(shipString) == true){
            this.carrypassengers(shipString);
            this.sequence += "pickup,";
    }
}
```

Where it checks if our location returns a ship from the ship hashtable. If we are in one call the method carrypassengers to carry passengers from that ship.
Then add to the sequence that it picked up passengers.

Where carry passengers is the following

```
    public void carrypassengers(String shipString){
        int vall = this.ships.get(shipString);
        int diff = this.max_passengers - this.passengers_carried;
        if(diff >= vall){
            this.passengers_carried += vall;
            //remove this ship from the hashmap
            this.ships.remove(shipString);
            //add a blackbox to the blackbox hashmap
            this.blackboxes.put(shipString, 19);
        }
        else{
            //replace the value by value - the diff
            this.ships.put(shipString, vall - diff);
            this.passengers_carried = this.max_passengers;
        }
}
```

This is the function carrypassengers where it takes a string representing the location of a ship and it gets the value from the hashtable then it saves the value of number

of passengers in a variable called vall , then it checks if there is a space in the number of passengers the boat can carry by calculating the difference between the number of passengers carried and the max number of passengers.
If the difference is higher than the vall then we will take all the passengers and remove the ship from the hashtable. Then add a blackbox with its health already decreased by 1 to the hashtable.

Else we will take as much as we can from the ship and continue by adding the difference between vall and the amount we can carry right now into the ship and then making the carried passengers max value.

This is for the helpers pertaining to action.
Now the next section will talk about the moves the agent can make , it can move up down left right depending on its position on the board right now.

```
    public ArrayList expand() {
        //create a new arraylist to store the expanded nodes
        ArrayList<tNode> expanded = new ArrayList<tNode>();
        //up , Check that the y co-ordinate is greater than 0
        if (this.co_ordinates[0] > 0) {
            //create a new node
            tNode up = new tNode(this.toString());
            //decrement the y co-ordinate
            up.co_ordinates[0]--;
            up.sequence = this.sequence + "up,";
            up.reduceMap();
            up.reduceBox();
            up.performAction();
            //add the node to the expanded nodes
            //System.out.println(up.co_ordinates[0] + ""+ up.sequence);
            expanded.add(up);
        }
        //down , Check that the y co-ordinate is less than the height
        if (this.co_ordinates[0] < this.length_width[0] - 1) {
            //create a new node
            tNode down = new tNode(this.toString());
            //increment the y co-ordinate
```

```
    down.co_ordinates[0]++;
    down.sequence =  this.sequence +"down,";
    down.reduceMap();
    down.reduceBox();
    down.performAction();
    //add the node to the expanded nodes
    //System.out.println(down.co_ordinates[0] + ""+ down.sequence);
    //down.number = down.ships.get("3,2");
    //add the node to the expanded nodes
    //System.out.println(down.co_ordinates[0] + ""+ down.sequence);
    expanded.add(down);
}
//Check left , right , up , down
//left , Check that the x co-ordinate is greater than 0
if (this.co_ordinates[1] > 0) {
    //create a new node
    tNode left = new tNode(this.toString());
    //decrement the x co-ordinate
    left.co_ordinates[1]--;
    left.sequence = this.sequence + "left,";
    //if passengers are dead. then cancel going in that direction.
    left.reduceMap();
    left.reduceBox();
    left.performAction();
    //System.out.println(left.co_ordinates[1] + ""+ left.sequence);
    //add the node to the expanded nodes
    expanded.add(left);
}

//right , Check that the x co-ordinate is less than the width
if (this.co_ordinates[1] < this.length_width[1] - 1) {
    //create a new node
    tNode right = new tNode(this.toString());
    //increment the x co-ordinate
    right.co_ordinates[1]++;
    right.sequence = this.sequence + "right,";
    right.reduceMap();
```

```
        right.reduceBox();
        right.performAction();
        //add the node to the expanded nodes
        //System.out.println(right.co_ordinates[1] + ""+ right.sequence);
        expanded.add(right);
    }
    //check if expanded is empty
    if (expanded.isEmpty()) {
        //return null
        return null;
    }
        //return the expanded nodes
        return expanded;

    }
```

It basically expands the node in all 4 directions if it can, based on the same logic so we will only go through one direction here and the rest is similar.

We first create an arraylist called expanded that we will return to the search algorithms when it needs expansions for a given node.

We will check first if the move in our case is up the same as on all other nodes.
So if we can move up by making sure we are having a value greater than 0.

We can create a tNode instance and pass to it the mother node converted to string using the toString method that will be discussed below.
We then decrease the y co-ordinate by 1 and then we add to the sequence of new child nodes up.

Next we call the method reduce box and the method reducemap on it
up.reduceMap()
up.reduceBox()

Where both if true cause an effect and if false are not executed.

Then we call on the new child node performAction discussed above. Then after that we can add the new node to the expanded arraylist.

```
public synchronized void reduceMap() {
    if (!(this.ships.isEmpty())) {
    Iterator<String> it = this.ships.keySet().iterator();
    while (it.hasNext()) {
        String key = it.next();
        int value = this.ships.get(key);
        if (value == 1) {
            this.blackboxes.put(key, 19);
            this.dead++;
            it.remove();
        } else {
            this.ships.replace(key, value - 1);
        }
    }
}
}
```

Where we check if the ships hashmap is not empty then we will iterate through the ships keys that is because we read online that the iterator is the way to loop through a hashtable without running into the heap issue as we ran into it before.
We store the key into a string and then we store the value of that key into an int value and then from there if the value is 1 then we will add the key into blackboxes hashtable with a value of 19 and will increase the number of dead people by.

And else we will replace the ship's key by the same value - 1.

```
public synchronized void reduceBox()
{
    if(!(this.blackboxes.isEmpty()))
    {
    Iterator<String> it = this.blackboxes.keySet().iterator();
    while(it.hasNext())
    {
        String key = it.next();
        int value = this.blackboxes.get(key);
        if(value == 1)
        {
            it.remove();
        }
        else
        {
            this.blackboxes.replace(key, value - 1);
        }
    }
    }
```

```
}
```

If the blackboxes hashtable is not empty then what we will do is we will loop through it using an iterator and then we will store its value reduce it by 1 , however if the value is already 1 then this means that it is already damaged by the next move so we just remove it

Then there is method to string which transforms the node to a string

```
    public String toString(){
//3,4 CoastGuardCarry=97 Boat=1,2 Station=0,1 Ship=3,2,Passengers=65;
String s = "";
s += this.length_width[1] + "," + this.length_width[0] + ";";
s  += this.Split[1] + ";";
s += this.co_ordinates[0] + "," + this.co_ordinates[1] + ";";
s   += this.Split[3] + ";";
    if(this.ships.size() > 0){
for (String ship : this.ships.keySet()) {
    s += ship.charAt(0) + "," + ship.charAt(2) + "," + this.ships.get(ship) + ",";
    }
    }
    else{
    s += " ;";
    }
//s += Blackboxes <<==
    //Check if split is more than 5 before adding passengers carried
    if(this.Split.length > 5){
s += this.Split[5] + ";";
    }
return s;

}
```