**Project Report**

* User defined functions used are : replace , findlist , findandUpdateTerm, generateBelow , generateRight , updateStats , createStatsList, organizeForSorting, cmp, extract, furnishRoom3, constructSubList , helper .
* Predefined functions used are : compare , sortBy , randomZeroToX .

It will be explained later in this file how each function was used

**Training phase**

1. **findFurnitureUpdate :**

**Case 1:** if the input statsList is Empty , then create new list of pairs but first check if the given location is “right” then the 2nd element of the pair will be a list of 2 lists where the 1st list is the RightList and the 2nd one is empty . if the given location is not “right” then the 2nd element of the pair will be a list of 2 lists where the 1st list is empty and the 2nd one is the BelowList.

**Case 2:**

1. Checks if the piece of furniture “a” is found in the head of the given statsList.
2. If yes then check if c is equal to “right”.
3. If yes then get the first list which represents the RightList from the list of lists in the 2nd item of the pair contains “x” using( “findlist/2 at index 0 ).
4. Findlist: is a user defined function which takes 2 parameters , the 1st one is a piece of furniture “a” and the 2nd one is the statsList and it returns the 2nd term of the pair which contains “a”.
5. findandUpdateTerm takes “b” , “c” and the list returned from findlist function , it searches for b and c in the list and
   1. If the input list is empty or the search failed to find the elements in the given list , then a new list is created and the element b & c with frequency 1 is added to it in that form (b,c,1).
   2. If the fist element of the input list contains b and c then the frequency associated will be incremented by 1 , otherwise a recursive call of the function occurs to search for b and c in the rest of the input list.
6. Replace the old list “rl” with the updated list returned from findandUpdateTerm.
7. If the condition in 2 failed and c is not equal “right” ,then execute step 3 ,4 & 5 but with the list at index 1 in step 3 which represents the “below List“.
8. If the condition in 1 failed, make a recursive call to search for “x” in the rest of the StatsList.

Note:

l@ is the name of the input Statslist .

pair@ is the name of the first pair of the list “l”

list1@ is the name of the list of lists which represent the 2nd element of a pair.

1. **generate :**

**Case1:** if the room list is not Empty, then it updates the RightLists in the statsList first, then take the updated list and generate the below lists of the updated StatsList so far.

**Case2:** if the room List is Empty then return the same input StatsList.

1. **generateRight:** updates the RightList in the input statslist
2. If the input room list is empty, then return the same input statsList.
3. If the first subList of the input room list is empty then generateRight the rest of the list.
4. If the input room list contains sublists of one element each, then generateRight the rest of the room list.
5. If the input room list contains sublists of 2 elements at least, then update the statsList with the new entry, and apply all the previous steps on the rest of the room list and the updated statslist so far.

Since we need to consider that each element in each sublist of the room list is the element lies below the corresponding element in the previous sublist, we have created two functions to make this job:

1. generateBelow: which iterates on the rows of the given room.
2. updateStats: which iterates on the columns of the same room.

For Example consider this room :

**[ [“tv1” , ”table1”, “e1”]**

**[“e2”, ”lamp1”,”table2”]**

**[“tv2” , ”couch”, “lamp2”] ]**

**We will consider the 2nd row twice , once for saying that “e2” lies below “tv1”, “lamp1“ lies below “table1”, and “table2” lies below “e1”. And another time for saying that, “tv2” lies below “e2”, “couch” lies below “lamp1” and “lamp2” lies below “table2”.**

1. **generateBelow:** updates the BelowList in the input statslist.
2. If the input room list is empty , then return the same input statsList.
3. If the first subList of the input room List is empty then generateBelow the rest of the list.
4. If there is only one subList of the input room list , then return the same input statsList.
5. If the input room list contains at least 2 sublists then call the user defined function updateStats/3 is called, and apply all the previous steps on the rest of the room list and the updated statslist so far .
6. **Updatestats:** takes 2 rows from the room and updates the statslist accordingly.
7. **statsList :**
8. **createStatsList:** it creates the statsList from scratch .
9. createStatsList first takes list of rooms and an empty list .
10. Then it generates the statsList for each room, until the list or rooms became empty.
11. Once the list of rooms is empty and the statslist is already generated, then organizeForSorting function will be called.
12. **organizeForSorting:** iterates over the generated statsList to sort the RightList and BelowList according to their frequency using the predefined function sortBy.
13. **cmp:** is a function which compares 2 elements in the from (a,b,c) according to the value of c , which represents the frequency in our model.

**Generation Phase:**

1. **getFurnStat:** uses the user defined function findlist , to get the list which contains the right and below lists of a certain piece of furniture .
2. **getPossibleNeighbour:** 
   1. Chooses one of the two input lists randomly using the predefined function randomZeroToX , to get a possible neighbour furniture from the randomly chosen list using the user defined method extract .
   2. extract function takes an element of the form (a,b,c) and returns a .
3. **furnishRoom:**

**case 1:**  if the input integer is zero , empty list is returned .

**case 2:** if the input integer is not zero then furnishRoom3/3 is called .

1. **furnishRoom3:** takes 3 parameters, the 1st one is an integer which represents the number of columns of the room , the 2nd one is a string and the 3rd one is also an integer which represents the number of rows in the room .

**Case 1:** if the number of columns equal zero , then it means that the room is empty and an empty list is returned .

**Case 2:** if the number of columns is not equal to zero, then the input string is added as the first element in the sublist that will be created using the user defined function constructSubList/2 .

1. **constructSubList:** takes 2 parameters , the 1st one represents the number of rows in the room and the 2nd one is a string .

**Case 1:** if the number of rows equal to 1 , this means that no more possible neighbours should be returned and the room contains only one element which is the string that previously added in furnishRoom3 function.

**Case 2:** if the number of rows greater than 1, a list will be created , where it’s first element is generated using a helper function and the rest of the list will be constructed recursively by calling constructSubList function again with the number or rows decremented .

1. **helper:** it’s a user defined function which takes a piece of furniture and return one possible neighbour of it.