COMP2209 Programming III CW2 Report

Name: Beh Shu Ao

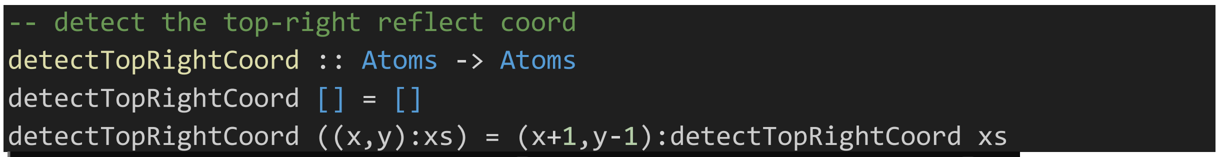
Student ID: 33354723

Student Email: sab1e22@soton.ac.uk

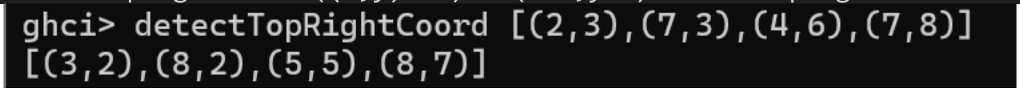
1. Identifying Coordinates

"detectTopRightCoord" is defined as the points where will disturb the path by the top right corner of the atoms.

`detectTopRightCoord`: Takes a list of coordinates and returns a new list where each coordinate has been adjusted by adding 1 to the x-coordinate and subtracting 1 from the y-coordinate.



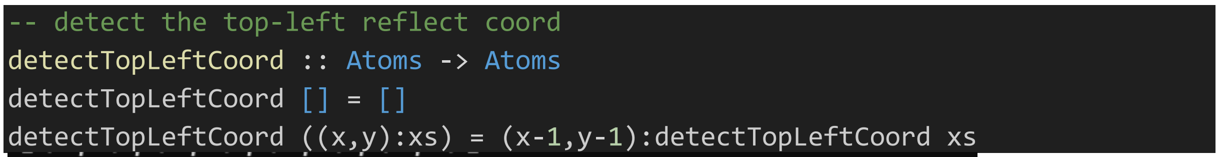
```
-- detect the top-right reflect coord
detectTopRightCoord :: Atoms -> Atoms
detectTopRightCoord [] = []
detectTopRightCoord ((x,y):xs) = (x+1,y-1):detectTopRightCoord xs
```



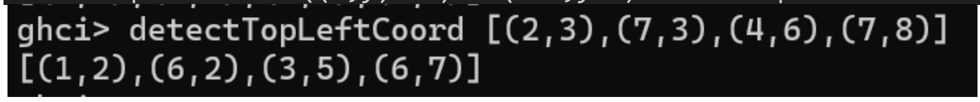
```
ghci> detectTopRightCoord [(2,3),(7,3),(4,6),(7,8)]
[(3,2),(8,2),(5,5),(8,7)]
```

"detectTopLeftCoord" is defined as the points where will disturb the path by the top left corner of the atoms.

`detectTopLeftCoord`: Takes a list of coordinates and returns a new list where each coordinate has been adjusted by subtracting 1 from both the x-coordinate and the y-coordinate.



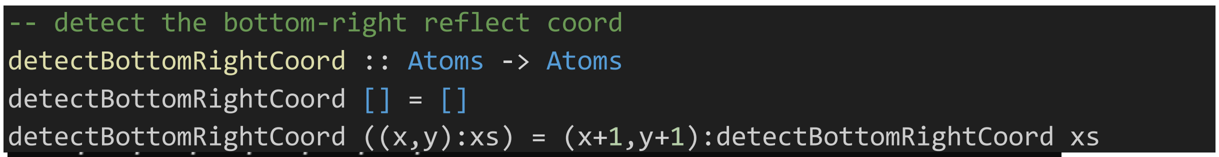
```
-- detect the top-left reflect coord
detectTopLeftCoord :: Atoms -> Atoms
detectTopLeftCoord [] = []
detectTopLeftCoord ((x,y):xs) = (x-1,y-1):detectTopLeftCoord xs
```



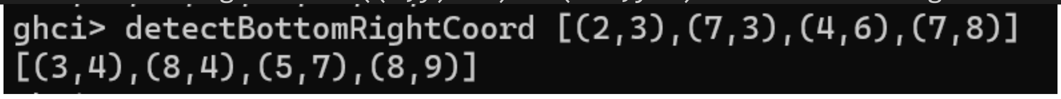
```
ghci> detectTopLeftCoord [(2,3),(7,3),(4,6),(7,8)]
[(1,2),(6,2),(3,5),(6,7)]
```

"detectBottomRightCoord" is defined as the points where will disturb the path by the bottom right corner of the atoms.

`detectBottomRightCoord`: Takes a list of coordinates and returns a new list where each coordinate has been adjusted by adding 1 to both the x-coordinate and the y-coordinate.



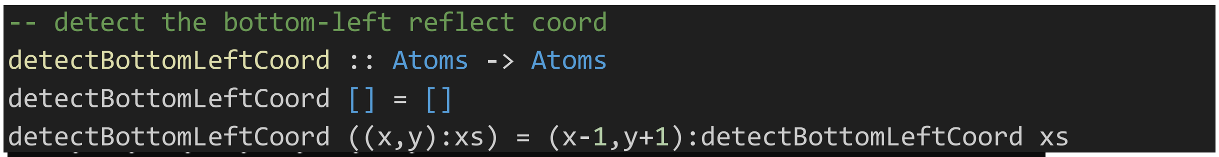
```
-- detect the bottom-right reflect coord
detectBottomRightCoord :: Atoms -> Atoms
detectBottomRightCoord [] = []
detectBottomRightCoord ((x,y):xs) = (x+1,y+1):detectBottomRightCoord xs
```



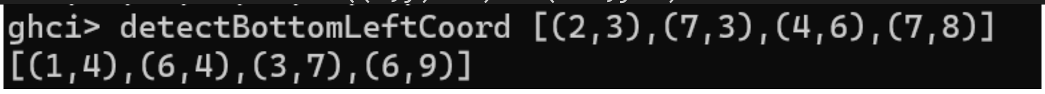
```
ghci> detectBottomRightCoord [(2,3),(7,3),(4,6),(7,8)]
[(3,4),(8,4),(5,7),(8,9)]
```

“detectBottomLeftCoord” is defined as the points where will disturb the path by the bottom left corner of the atoms.

`**detectBottomLeftCoord**`: Takes a list of coordinates and returns a new list where each coordinate has been adjusted by subtracting 1 from the x-coordinate and adding 1 to the y-coordinate.



```haskell
-- detect the bottom-left reflect coord
detectBottomLeftCoord :: Atoms -> Atoms
detectBottomLeftCoord [] = []
detectBottomLeftCoord ((x,y):xs) = (x-1,y+1):detectBottomLeftCoord xs
```



```
ghci> detectBottomLeftCoord [(2,3),(7,3),(4,6),(7,8)]
[(1,4),(6,4),(3,7),(6,9)]
```

type Atoms = [Pos]

North West	1	2	3	4	5	6	7	8	
1									1
2	(1,2)		(3,2)			(6,2)		(8,2)	2
3		(2,3)					(7,3)		3
4	(1,4)		(3,4)			(6,4)		(8,4)	4
5			(3,5)		(5,5)				5
6				(4,6)					6
7			(3,7)		(5,7)	(6,7)		(8,7)	7
8							(7,8)		8
						(6,9)		(8,9)	East South

2. Movement of Coordinates

`**goVerticalPath**` Function:

Parameters:

- **start:** Represents the starting edge position. Eg:(North,1)
- **num**: Represents the size of the grid.
- **pos**: Represents the current position on the grid. Eg: (1,0)
- **direction:** Represents the direction of movement (either up or down). Eg: 1 for down, -1 for up
- **atomsCoord**: Represents a list of atom coordinates.

Guard Clauses:

- Checks if the current position is in `**atomsCoord**`**.** If true, it absorbs the particle **(Absorb**).
- Checks if the particle is at the top or bottom corners of the grid and reflects it **(Reflect).**
- Checks if the particle is at the top or bottom edge of the grid and returns a **Path** indicating reaching the **North** or **South** side.

Recursion:

- If the above conditions are not met, the function recursively calls itself with an updated position based on the specified direction.

```
goVerticalPath :: EdgePos -> Int -> Pos ->Int -> Atoms -> (EdgePos,Marking)
goVerticalPath start num (x,y) direction atomsCoord
    | elem (x,y) topRightCornerAtom && direction==1 && y==0 = (start,Reflect)
    | elem (x,y) bottomRightCornerAtom && direction==(-1) && y==num+1 =
(start,Reflect)
    | elem (x,y) topLeftCornerAtom && direction==1 && y==0 = (start,Reflect)
    | elem (x,y) bottomLeftCornerAtom && direction==(-1) && y==num+1 =
(start,Reflect)

    | direction==(-1) && y<=0 = (start, Path (North,x)) -- reach the North
Side
    | direction==1 && y>=num+1 = (start, Path (South,x)) -- reach the South
Side

    | elem (x,y) atomsCoord =(start, Absorb)

    | elem (x,y) topRightCornerAtom = goHorizontalPath start num (x+1,y) 1
atomsCoord --go to west
    | elem (x,y) bottomRightCornerAtom = goHorizontalPath start num (x+1,y) 1
atomsCoord -- go to west
    | elem (x,y) topLeftCornerAtom = goHorizontalPath start num (x-1,y) (-1)
atomsCoord -- go to east
```

```
    | elem (x,y) bottomLeftCornerAtom = goHorizontalPath start num (x-1,y) (-
1) atomsCoord -- go to east

    | otherwise = goVerticalPath start num (x,y+direction) direction
atomsCoord -- based on the direction to go up or down
        where
            topLeftCornerAtom = detectTopLeftCoord atomsCoord
            topRightCornerAtom = detectTopRightCoord atomsCoord
            bottomLeftCornerAtom = detectBottomLeftCoord atomsCoord
            bottomRightCornerAtom = detectBottomRightCoord atomsCoord
```

`**goHorizontalPath**` Function:

Parameters:

- **start**: Represents the starting edge position.
- **num**: Represents the size of the grid.
- **pos**: Represents the current position on the grid.
- **direction**: Represents the direction of movement (either left or right).
- **atomsCoord**: Represents a list of atom coordinates.

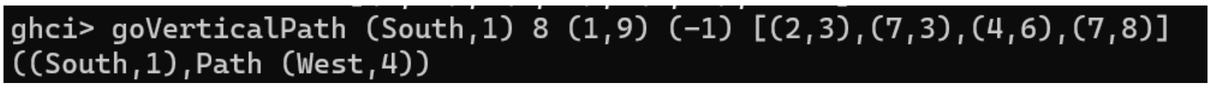
Guard Clauses:

- Checks if the current position is in `**atomsCoord**`. If true, it absorbs the particle **(Absorb**).
- Checks if the particle is at the left or right corners of the grid and reflects it (**Reflect**).
- Checks if the particle is at the left or right edge of the grid and returns a **Path** indicating reaching the **West** or **East** side.

Recursion:

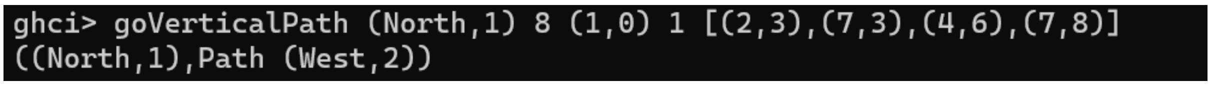
- If the above conditions are not met, the function recursively calls itself with an updated position based on the specified direction.

```
goHorizontalPath :: EdgePos -> Int -> Pos ->Int -> Atoms -> (EdgePos,Marking)
goHorizontalPath start num (x,y) direction atomsCoord
    | elem (x,y) topRightCornerAtom && direction==1 && x==0 = (start,Reflect)
    | elem (x,y) bottomRightCornerAtom && direction==1 && x==0 =
(start,Reflect)
    | elem (x,y) topLeftCornerAtom && direction==(-1) && x==num+1 =
(start,Reflect)
    | elem (x,y) bottomLeftCornerAtom && direction==(-1) && x==num+1 =
(start,Reflect)

    | direction==1 && x>=num+1 = (start, Path (East,y))
    | direction==(-1) && x<=0 = (start, Path (West,y))
    | elem (x,y) atomsCoord =(start, Absorb)

    | elem (x,y) topRightCornerAtom = goVerticalPath start num (x,y-1) (-1)
atomsCoord
    | elem (x,y) bottomRightCornerAtom = goVerticalPath start num (x,y+1) 1
atomsCoord
    | elem (x,y) topLeftCornerAtom = goVerticalPath start num (x,y-1) (-1)
atomsCoord
    | elem (x,y) bottomLeftCornerAtom = goVerticalPath start num (x,y+1) 1
atomsCoord

    | otherwise = goHorizontalPath start num (x+direction,y) direction
atomsCoord
        where
            topLeftCornerAtom = detectTopLeftCoord atomsCoord
            topRightCornerAtom = detectTopRightCoord atomsCoord
            bottomLeftCornerAtom = detectBottomLeftCoord atomsCoord
            bottomRightCornerAtom = detectBottomRightCoord atomsCoord
```



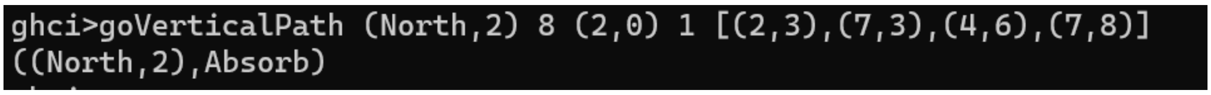
```
ghci> goVerticalPath (South,1) 8 (1,9) (-1) [(2,3),(7,3),(4,6),(7,8)]
((South,1),Path (West,4))
```

The start is (South,1) and the size of the grid is 8. Based on its edgePos, its coordinate is (1,9) and its direction is -1, moves up to north direction. Its atoms coordinates are (2,3), (7,3), (4,6) and (7,8). It goes up towards North direction until it detects the bottom-left corner of the atom (2,3) which is (1,4). Then, it will change direction towards West, finally reach (0,4) which is (West,4).



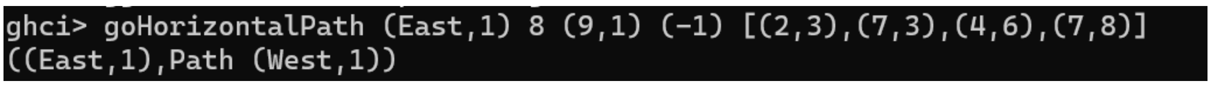
```
ghci> goVerticalPath (North,1) 8 (1,0) 1 [(2,3),(7,3),(4,6),(7,8)]
((North,1),Path (West,2))
```

The start is (North,1) and the size of the grid is 8. Based on its edgePos, its coordinate is (1,0) and its direction is +1, moves down to south direction. Its atoms coordinates are (2,3), (7,3), (4,6) and (7,8). It goes down towards South direction until it detects the top-left corner of the atom (2,3) which is (1,2). Then, it will change direction towards West, finally reach (0,2) which is (West,2).



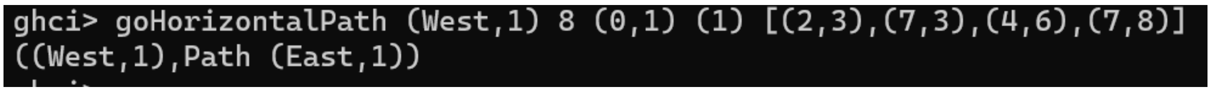
```
ghci>goVerticalPath (North,2) 8 (2,0) 1 [(2,3),(7,3),(4,6),(7,8)]
((North,2),Absorb)
```

The start is (North,2) and the size of the grid is 8. Based on its edgePos, its coordinate is (2,0) and its direction is +1, moves down to south direction. Its atoms coordinates are (2,3), (7,3), (4,6) and (7,8). It goes down towards South direction until it reaches the atom (2,3). Then, it will be absorbed.



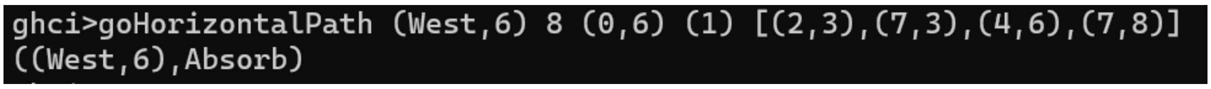
```
ghci> goHorizontalPath (East,1) 8 (9,1) (-1) [(2,3),(7,3),(4,6),(7,8)]
((East,1),Path (West,1))
```

The start is (East,1) and the size of the grid is 8. Based on its edgePos, its coordinate is (9,1) and its direction is -1, moves left to west direction. Its atoms coordinates are (2,3), (7,3), (4,6) and (7,8). It goes left towards West without reflection by any corner of the atoms or absorbed by atoms, reached (0,1) which is (East,1).



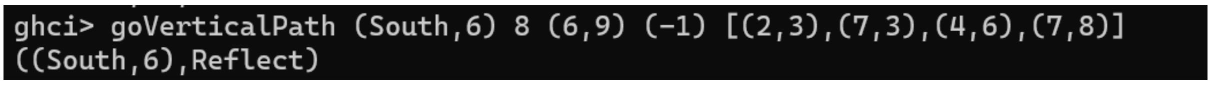
```
ghci> goHorizontalPath (West,1) 8 (0,1) (1) [(2,3),(7,3),(4,6),(7,8)]
((West,1),Path (East,1))
```

The start is (West,1) and the size of the grid is 8. Based on its edgePos, its coordinate is (0,1) and its direction is +1, moves right to east direction. Its atoms coordinates are (2,3), (7,3), (4,6) and (7,8). It goes right towards East without reflection by any corner of the atoms or absorbed by atoms, reached (9,1) which is (West,1).



```
ghci>goHorizontalPath (West,6) 8 (0,6) (1) [(2,3),(7,3),(4,6),(7,8)]
((West,6),Absorb)
```

The start is (West,2) and the size of the grid is 8. Based on its edgePos, its coordinate is (0,6) and its direction is +1, moves right to east direction. Its atoms coordinates are (2,3), (7,3), (4,6) and (7,8). It goes down towards East direction until it meets the atom (4,6). Then, it will be absorbed.



```
ghci> goVerticalPath (South,6) 8 (6,9) (-1) [(2,3),(7,3),(4,6),(7,8)]
((South,6),Reflect)
```

The start is (South,6) and the size of the grid is 8. Based on its edgePos, its coordinate is (6,9) and its direction is -1, moves right to north direction. Its atoms coordinates are (2,3), (7,3), (4,6) and (7,8). It goes up towards South direction, but it will be reflected by bottom-left corner of the atom (7,8). Then, it will be reflected since it is on the edge of South.

3. Generate List of the Interactions

It is separated into four lists which is generated by four functions starting from specific directions which are North, East, West and South based on the grid size to generate the interactions.

`**pathFromNorthCoord**` Function:

Parameters:

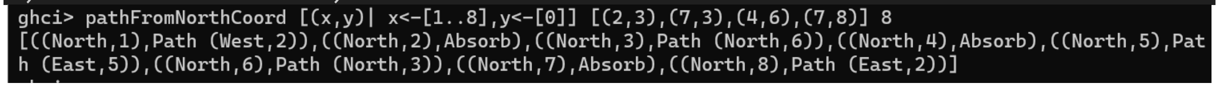
- **xs**: List of initial coordinates.
- **atomsCoord**: List of atom coordinates.
- **num**: Size of the grid.

Functionality:

- Maps over the initial coordinates (**xs**) and applies **goVerticalPath** for each coordinate, starting from the North edge (**North**) with an initial direction of **1** (upwards).
- The result is a list of interactions (**Interactions**) representing the paths of particles starting from the North edge.

For north coordinate, its y-coordinate is **0**.

```
pathFromNorthCoord :: Atoms-> Atoms -> Int -> Interactions
pathFromNorthCoord xs atomsCoord num = map (\ x -> goVerticalPath (North,fst
x) num x 1 atomsCoord) xs
```



```
ghci> pathFromNorthCoord [(x,y)| x<-[1..8],y<-[0]] [(2,3),(7,3),(4,6),(7,8)] 8
[((North,1),Path (West,2)),((North,2),Absorb),((North,3),Path (North,6)),((North,4),Absorb),((North,5),Pat
h (East,5)),((North,6),Path (North,3)),((North,7),Absorb),((North,8),Path (East,2))]
```

`**pathFromEastCoord**` Function:

Parameters:

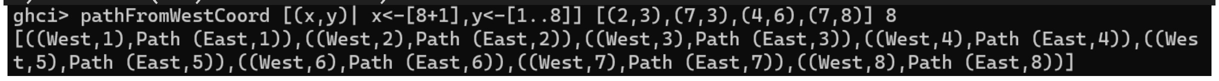
- **xs**: List of initial coordinates.
- **atomsCoord**: List of atom coordinates.
- **num**: Size of the grid.

Functionality:

- Maps over the initial coordinates (**xs**) and applies `**goHorizontalPath**` for each coordinate, starting from the East edge (**East**) with an initial direction of **-1** (leftwards).
- The result is a list of interactions (**Interactions**) representing the paths of particles starting from the East edge.

For east edge, its x-coordinate is num+1 which is the **gridsize+1.**

```
pathFromEastCoord :: Atoms-> Atoms -> Int -> Interactions
pathFromEastCoord xs atomsCoord num = map (\ x -> goHorizontalPath (East,snd
x) num x (-1) atomsCoord) xs
```



```
ghci> pathFromWestCoord [(x,y)| x<-[8+1],y<-[1..8]] [(2,3),(7,3),(4,6),(7,8)] 8
[((West,1),Path (East,1)),((West,2),Path (East,2)),((West,3),Path (East,3)),((West,4),Path (East,4)),((Wes
t,5),Path (East,5)),((West,6),Path (East,6)),((West,7),Path (East,7)),((West,8),Path (East,8))]
```

`**pathFromSouthCoord**` Function:

Parameters:

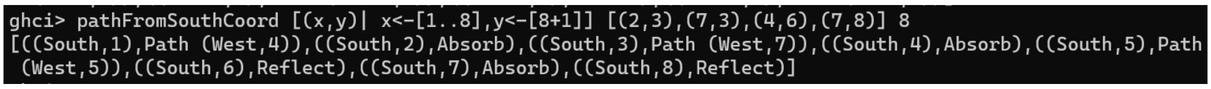
- **xs**: List of initial coordinates.
- **atomsCoord**: List of atom coordinates.
- **num**: Size of the grid.

Functionality:

- Maps over the initial coordinates (**xs**) and applies `**goVerticalPath**` for each coordinate, starting from the South edge (**South**) with an initial direction of **-1** (downwards).
- The result is a list of interactions (**Interactions**) representing the paths of particles starting from the South edge.

For south coordinate, its y-coordinate is num+1 which is **gridSize+1**.

```
pathFromSouthCoord :: Atoms-> Atoms -> Int -> Interactions
pathFromSouthCoord xs atomsCoord num = map (\ x -> goVerticalPath (South,fst
x) num x (-1) atomsCoord) xs
```



```
ghci> pathFromSouthCoord [(x,y)| x<-[1..8],y<-[8+1]] [(2,3),(7,3),(4,6),(7,8)] 8
[((South,1),Path (West,4)),((South,2),Absorb),((South,3),Path (West,7)),((South,4),Absorb),((South,5),Path
 (West,5)),((South,6),Reflect),((South,7),Absorb),((South,8),Reflect)]
```

`**pathFromWestCoord**` Function:

Parameters:

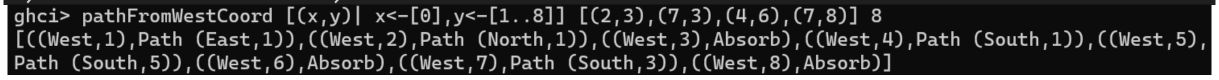
- **xs**: List of initial coordinates.
- **atomsCoord**: List of atom coordinates.
- **num**: Size of the grid.

Functionality:

- Maps over the initial coordinates (**xs**) and applies `**goHorizontalPath**` for each coordinate, starting from the West edge (**West**) with an initial direction of **1** (rightwards).
- The result is a list of interactions (**Interactions**) representing the paths of particles starting from the West edge.

For east edge, its y-coordinate is **0.**

```
pathFromWestCoord :: Atoms-> Atoms -> Int -> Interactions
pathFromWestCoord xs atomsCoord num = map (\ x -> goHorizontalPath (West,snd
x) num x 1 atomsCoord) xs
```



```
ghci> pathFromWestCoord [(x,y)| x<-[0],y<-[1..8]] [(2,3),(7,3),(4,6),(7,8)] 8
[((West,1),Path (East,1)),((West,2),Path (North,1)),((West,3),Absorb),((West,4),Path (South,1)),((West,5),
Path (South,5)),((West,6),Absorb),((West,7),Path (South,3)),((West,8),Absorb)]
```

<u>4. Generate All the Possible Interactions From Four Edges.</u>

The "calcBBInteractions" function is used to generate all the possible interactions from four edges which is North, East, South and West and concatenates them into a single list.

`**calcBBInteractions**` Function:

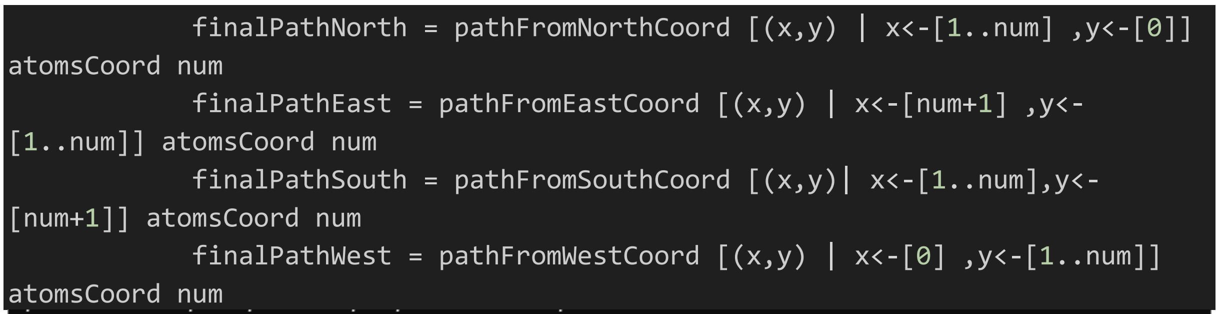
Parameters:

- **num**: Size of the grid.
- **atomsCoord**: List of atom coordinates.

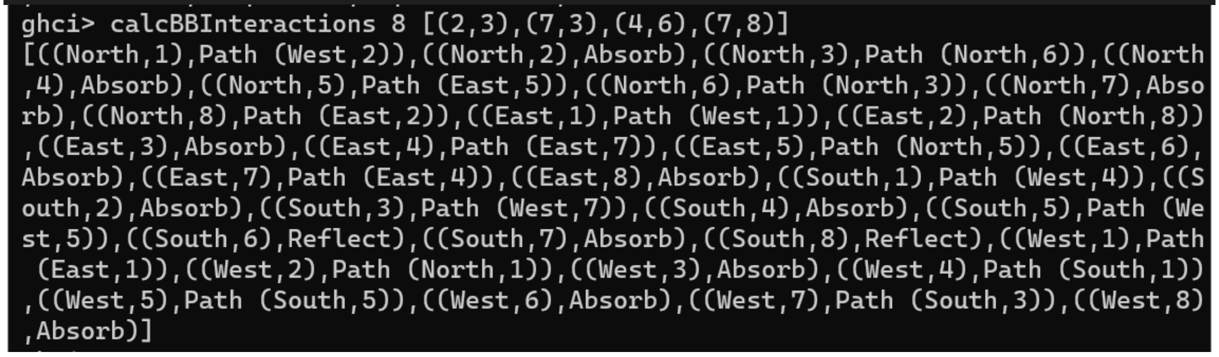
Functionality:

- Creates four lists of interactions by invoking specific functions for particles starting from different edges.
  - **finalPathNorth**: Interactions for particles starting from the North edge.
  - **finalPathEast**: Interactions for particles starting from the East edge.
  - **finalPathSouth**: Interactions for particles starting from the South edge.
  - **finalPathWest**: Interactions for particles starting from the West edge.
- Concatenates these four lists into a single list, representing all interactions.

```
-- return the set of interactions from all possible edge entry point
calcBBInteractions :: Int -> Atoms -> Interactions
calcBBInteractions num atomsCoord =
finalPathNorth++finalPathEast++finalPathSouth++finalPathWest
        where
```



```
            finalPathNorth = pathFromNorthCoord [(x,y) | x<-[1..num] ,y<-[0]]
atomsCoord num
            finalPathEast = pathFromEastCoord [(x,y) | x<-[num+1] ,y<-
[1..num]] atomsCoord num
            finalPathSouth = pathFromSouthCoord [(x,y)| x<-[1..num],y<-
[num+1]] atomsCoord num
            finalPathWest = pathFromWestCoord [(x,y) | x<-[0] ,y<-[1..num]]
atomsCoord num
```



```
ghci> calcBBInteractions 8 [(2,3),(7,3),(4,6),(7,8)]
[((North,1),Path (West,2)),((North,2),Absorb),((North,3),Path (North,6)),((North
,4),Absorb),((North,5),Path (East,5)),((North,6),Path (North,3)),((North,7),Abso
rb),((North,8),Path (East,2)),((East,1),Path (West,1)),((East,2),Path (North,8))
,((East,3),Absorb),((East,4),Path (East,7)),((East,5),Path (North,5)),((East,6),
Absorb),((East,7),Path (East,4)),((East,8),Absorb),((South,1),Path (West,4)),((S
outh,2),Absorb),((South,3),Path (West,7)),((South,4),Absorb),((South,5),Path (We
st,5)),((South,6),Reflect),((South,7),Absorb),((South,8),Reflect),((West,1),Path
 (East,1)),((West,2),Path (North,1)),((West,3),Absorb),((West,4),Path (South,1))
,((West,5),Path (South,5)),((West,6),Absorb),((West,7),Path (South,3)),((West,8)
,Absorb)]
```

The below is the table for me to check the interactions between all edge points.

Start from North Side

North West	1 In	2 In	3 In/Out	4 In	5 In	6 Out/In	7 In	8 In	
1	Path	Absorb	Path	Absorb	Path	Path	Absorb	Path	1
2 out	Path	Absorb	Path	Path	Path	Path	Absorb	Path	2 Out
3		(2,3)		Absorb	Path		(7,3)		3
4				Absorb	Path				4
5				Absorb	Path	Path	Path	Path	5 out
6				(4,6)					6
7									7
8							(7,8)		8
									East South

(North,1) -> (1,0) -> (2,0) -> (West,2)

(North,2) -> (2,0) -> (2,3) -> Absorb

(North,3) -> (3,0) -> (6,0) -> (North,6)

(North,4) -> (4,0) -> (4,6) -> Absorb

(North,5) -> (5,0) -> (9,5) -> (East,5)

(North,6) -> (6,0) -> (3,0) -> (North,3)

(North,7) -> (7,0) -> (7,3) -> Absorb

(North,8) -> (8,0) -> (9,2) -> (East,2)

Start from East Side

North West	1	2	3	4	5 Out	6	7	8 Out	North East
1 Out	Path	Path	Path	Path	Path	Path	Path	Path	1 In
2					Path			Path	2 In
3		(2,3)			Path		(7,3)	Absorb	3 In
4					Path			Path	4 In/Out
5					Path	Path	Path	Path	5 In
6				(4,6)	Absorb	Absorb	Absorb	Path	6 In
7								Path	7 Out/In
8							(7,8)	Absorb	8 In
									East South

(East,1) -> (9,1) -> (0,1) -> (West,1)

(East,2) -> (9,2) -> (8,0) -> (North,8)

(East,3) -> (9,3) -> (7,3) -> Absorb

(East,4) -> (9,4) -> (9,7) -> (West,7)

(East,5) -> (9,5) -> (5,0) -> (North,5)

(East,6) -> (9,6) -> (4,6) -> Absorb

(East,7) -> (9,7) -> (9,4) -> (East,4)

(East,8) -> (9,8) -> (7,8) -> Absorb

Start from South Side

North West	1	2	3	4	5	6	7	8	
1									1
2									2
3		(2,3)					(7,3)		3
4 out	Path	Absorb	Path	Path	Path	Path			4
5 out	Path	Absorb	Path			Path			5
6	Path	Absorb		(4,6)		Path			6
7 Out	Path	Path	Path	Absorb	Path	Path			7
8	Path	Absorb	Path	Absorb	Path		(7,8)		8
West South	1 In	2 In	3 In	4 In	5 In	6 Reflect In/Out	7 Absorb In	8 Reflect In/Out	East South

(South,1) -> (1,9) -> (0,4) -> (West,4)

(South,2) -> (2,9) -> (2,3) -> Absorb

(South,3) -> (3,9) -> (0,7) -> (West,7)

(South,4) -> (4,9) -> (4,6) -> Absorb

(South,5) -> (5,9) -> (0,5) -> (West,5)

(South,6) -> (6,9) -> (6,9) -> Reflect

(South,7) -> (7,9) -> (7,8) -> Absorb

(South,8) -> (8,9) -> (8,9) -> Reflect

Start from west side

North West	1 Out	2	3	4	5	6	7	8	
1 In	Path	Path	Path	Path	Path	Path	Path	Path	1 Out
2 In	Path								2
3 In	Absorb	(2,3)					(7,3)		3
4 In	Path		Path	Path	Path	Path			4
5 In	Path	Path	Path			Path			5
6 In	Path	Absorb	Absorb	(4,6)		Path			6
7 In	Path	Path	Path		Path	Path			7
8 In	Path	Absorb	Path	Absorb	Absorb	Absorb	(7,8)		8
West South	1 Out	2	3 Out	4	5 out	6	7	8	East South

(West,1) -> (0,1) -> (9,1) -> (East,1)

(West,2) -> (0,2) -> (1,0) -> (North,1)

(West,3) -> (0,3) -> (2,3) -> Absorb

(West,4) -> (0,4) -> (1,9) -> (South,1)

(West,5) -> (0,5) -> (5,9) -> (South,5)

(West,6) -> (0,6) -> (4,6) -> Absorb

(West,7) -> (0,7) -> (3,9) -> (South,3)

(West,8) -> (0,8) -> (7,8) -> Absorb

5. Finding Position of Atoms

`**findMapSize**` Function:

Parameters:

- **xs**: List of interactions.
- **maxSize**: Initial maximum size.

Functionality:

- Uses `**foldl**` to iterate through the list of interactions.
- For each interaction (x), it compares the size (second component of the first component of the interaction: `**snd (fst x)**`) with the current maximum size.
- The maximum of the current size and the maximum size encountered so far is used for the next iteration.
- The final result is the maximum size encountered during the iteration.

```
-- return the max number of the map
findMapSize :: Interactions -> Int -> Int
findMapSize xs maxSize = foldl (\ maxSize x -> max maxSize (snd (fst x)))
maxSize xs
```

This function is taken from https://stackoverflow.com/questions/52602474/function-to-generate-the-unique-combinations-of-a-list-in-haskell

`**subsets**` Function:

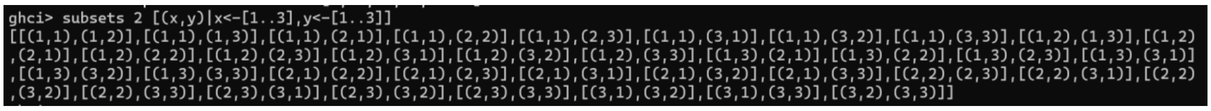
Parameters:

- **n**: the size of the subsets to generate
- **atoms**: a list of atoms

Functionality:

- **Base** case of the recursion: When n is **0**, it returns a list containing an empty list, representing the only subset of size 0.
- **Another** base case: When the list of atoms is empty, there are no subsets to generate, so an empty list is returned.
- **Recursive** case: It generates subsets by taking the first element `**x**` and recursively generating subsets of size **(n - 1)** from the remaining elements `**xs**`. The **map (x :)** part adds the current element `**x**` to each subset generated from the recursive call. The "**++**" operator concatenates the subsets obtained from the recursive call with subsets generated without including the current element. This process continues until the base cases are reached.

```
-- generate all combinations of atoms
--for grid size 8 and wants 4 atoms it needs 635,376 combinations (8C3)
subsets :: Int-> Atoms -> [Atoms]
subsets 0 _ = [[]]
subsets _ [] = []
subsets n (x:xs) =map (x :) (subsets (n - 1) xs) ++ subsets n xs
```



```
ghci> subsets 2 [(x,y)|x<-[1..3],y<-[1..3]]
[[(1,1),(1,2)],[(1,1),(1,3)],[(1,1),(2,1)],[(1,1),(2,2)],[(1,1),(2,3)],[(1,1),(3,1)],[(1,1),(3,2)],[(1,1),(3,3)],[(1,2),(1,3)],[(1,2)
,(2,1)],[(1,2),(2,2)],[(1,2),(2,3)],[(1,2),(3,1)],[(1,2),(3,2)],[(1,2),(3,3)],[(1,3),(2,1)],[(1,3),(2,2)],[(1,3),(2,3)],[(1,3),(3,1)]
,[(1,3),(3,2)],[(1,3),(3,3)],[(2,1),(2,2)],[(2,1),(2,3)],[(2,1),(3,1)],[(2,1),(3,2)],[(2,1),(3,3)],[(2,2),(2,3)],[(2,2),(3,1)],[(2,2)
,(3,2)],[(2,2),(3,3)],[(2,3),(3,1)],[(2,3),(3,2)],[(2,3),(3,3)],[(3,1),(3,2)],[(3,1),(3,3)],[(3,2),(3,3)]]
```

`**generateAtomCombinations**` Function:

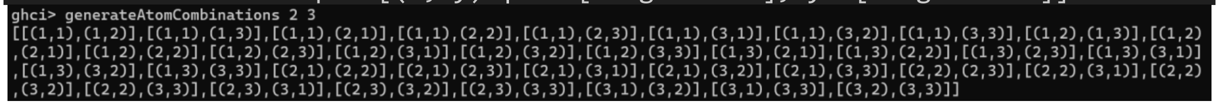
Parameter:

- **numOfAtoms**: the number of atoms in each combination.
- **gridSize**: the size of the 2D grid.

Functionality:

- Uses the `**subsets**` function to generate all combinations of atoms from the list of **allCoordInMap**.
- **allCoordInMap** generates all possible pairs of **(x,y)** where x and y range from 1 to **gridSize**. This represent all possible atoms in a grid of size `**gridSize**`

```haskell
--generate possible atom combinations based on a grid size
generateAtomCombinations::Int->Int->[Atoms]
generateAtomCombinations numOfAtoms gridSize = subsets numOfAtoms
allCoordInMap
  where allCoordInMap = [(x, y) | x<-[1..gridSize], y<-[1..gridSize]]
```



```
ghci> generateAtomCombinations 2 3
[[(1,1),(1,2)],[(1,1),(1,3)],[(1,1),(2,1)],[(1,1),(2,2)],[(1,1),(2,3)],[(1,1),(3,1)],[(1,1),(3,2)],[(1,1),(3,3)],[(1,2),(1,3)],[(1,2)
,(2,1)],[(1,2),(2,2)],[(1,2),(2,3)],[(1,2),(3,1)],[(1,2),(3,2)],[(1,2),(3,3)],[(1,3),(2,1)],[(1,3),(2,2)],[(1,3),(2,3)],[(1,3),(3,1)]
,[(1,3),(3,2)],[(1,3),(3,3)],[(2,1),(2,2)],[(2,1),(2,3)],[(2,1),(3,1)],[(2,1),(3,2)],[(2,1),(3,3)],[(2,2),(2,3)],[(2,2),(3,1)],[(2,2)
,(3,2)],[(2,2),(3,3)],[(2,3),(3,1)],[(2,3),(3,2)],[(2,3),(3,3)],[(3,1),(3,2)],[(3,1),(3,3)],[(3,2),(3,3)]]
```

`**filterAtoms**` Functin:

Parameter:

- **numOfAtoms**: the number of atoms
- **gridSize**: the size of the 2D grid

Functionality:

- **atoms <- possibleAtoms**: Iterates over all possible combinations of atoms generated by generateAtomCombinations.
- **let testCase = calcBBInteractions gridSize atoms**: Calculates the interactions using the function **calcBBInteractions** for the current set of atoms.
- **contains interactions testCase**: Checks if the calculated interactions contain the given interactions.
- **length testCase > 0**: Ensures that the calculated interactions are non-empty.
- The filtered atoms are returned, and **getFirstElement** is assumed to be a function that extracts the first element from the list of filtered atoms.

```haskell
--filter the atoms that doesn't give the same interactions
filterAtoms :: Int -> Int -> Interactions -> Atoms
filterAtoms gridSize n interactions = getFirstElement
```

```
  [atoms | atoms<-possibleAtoms, let testCase=(calcBBInteractions gridSize
atoms),  contains interactions result, (length testCase)>0]
    where possibleAtoms = generateAtomCombinations n gridSize
```

`**getFirstElement**` Function:

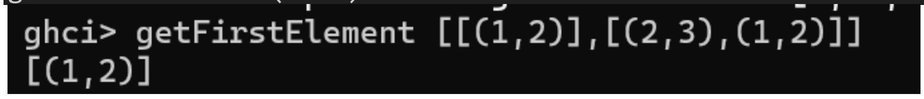
Parameter:

- **List**: separate it into x and xs

Functionality:

- For **empty** list, we just return empty list
- For **non-empty** list, pattern-matches on the list, binding the first element to **x**, and returned that element.

```
--same function as head
getFirstElement :: [Atoms] ->Atoms
getFirstElement [] = []
getFirstElement  (x:xs) = x
```



```
ghci> getFirstElement [[(1,2)],[(2,3),(1,2)]]
[(1,2)]
```

`**contains**` Function:

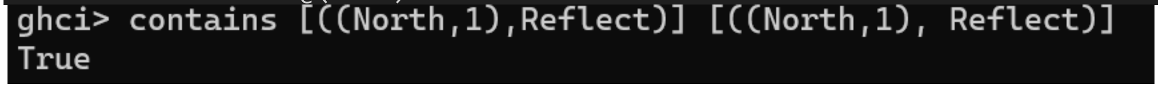
Parameter:

- **Set**: the list of the interactions which needs to solve
- **subset@(x:xs)**: subset of test case list

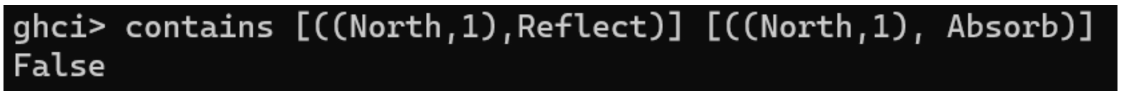
Functionality:

- For **empty** subset, it returns `**True**` because from the view of Maths, an empty subset is always contained in any set.
- For **non-empty** subset, pattern-matches on the list, binding the first element of the subset to **x** and the rest of the subset to **xs**. Then, it will checks if x is an element of the set using `**elem**` function. If it is, the function is called recursively on the remaining subset (`xs`). If `x` is not in the set, it returns `**False**`.

```
-- to check element exists in this list or not
contains :: Interactions -> Interactions -> Bool
contains set [] = True
contains set subset@(x:xs) = if x `elem` set then contains set xs else False
```



```
ghci> contains [((North,1),Reflect)] [((North,1), Reflect)]
True
```



```
ghci> contains [((North,1),Reflect)] [((North,1), Absorb)]
False
```

`**solveBB**` function:

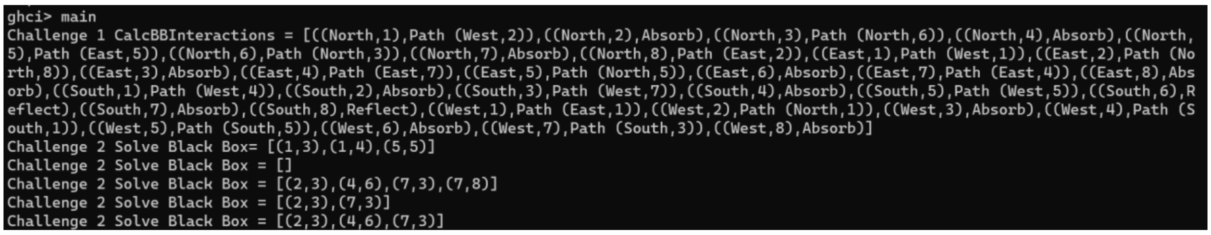
Parameters:

- **numOfAtoms**: the number of atoms
- **interactions**: the interactions which need to be solved

Functionality:

- **findMapSize interactions 0**: This part seems to calculate a map size based on the interactions. The specific implementation is not provided, but it seems to take the interactions and some initial value (0 in this case).
- **filterAtoms (findMapSize interactions 0) numOfAtoms interactions**: This part calls the filterAtoms function with the map size, numOfAtoms, and the interactions.

```
solveBB :: Int -> Interactions -> Atoms
solveBB numOfAtoms interactions = filterAtoms (findMapSize interactions 0)
numOfAtoms interactions
```

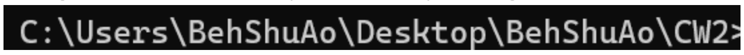


```
ghci> main
Challenge 1 CalcBBInteractions = [((North,1),Path (West,2)),((North,2),Absorb),((North,3),Path (North,6)),((North,4),Absorb),((North,
5),Path (East,5)),((North,6),Path (North,3)),((North,7),Absorb),((North,8),Path (East,2)),((East,1),Path (West,1)),((East,2),Path (No
rth,8)),((East,3),Absorb),((East,4),Path (East,7)),((East,5),Path (North,5)),((East,6),Absorb),((East,7),Path (East,4)),((East,8),Abs
orb),((South,1),Path (West,4)),((South,2),Absorb),((South,3),Path (West,7)),((South,4),Absorb),((South,5),Path (West,5)),((South,6),R
eflect),((South,7),Absorb),((South,8),Reflect),((West,1),Path (East,1)),((West,2),Path (North,1)),((West,3),Absorb),((West,4),Path (S
outh,1)),((West,5),Path (South,5)),((West,6),Absorb),((West,7),Path (South,3)),((West,8),Absorb)]
Challenge 2 Solve Black Box= [(1,3),(1,4),(5,5)]
Challenge 2 Solve Black Box = []
Challenge 2 Solve Black Box = [(2,3),(4,6),(7,3),(7,8)]
Challenge 2 Solve Black Box = [(2,3),(7,3)]
Challenge 2 Solve Black Box = [(2,3),(4,6),(7,3)]
```

6.Running GHCI

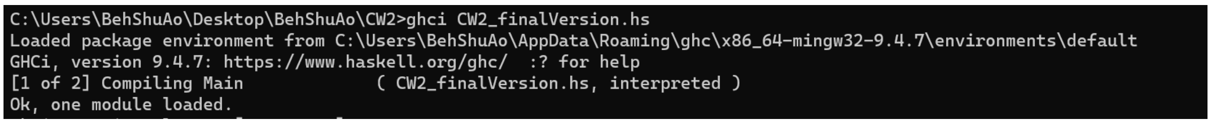
To run the program on GHCI , the following steps must be taken:

1. Launch the command prompt
2. Navigate to the directory of the script using `cd`



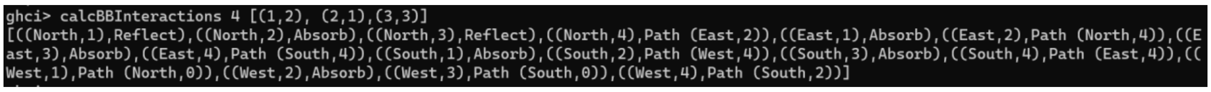
```
C:\Users\BehShuAo\Desktop\BehShuAo\CW2>
```

3. Load the script by typing `ghci CW2_finalVersion.hs`



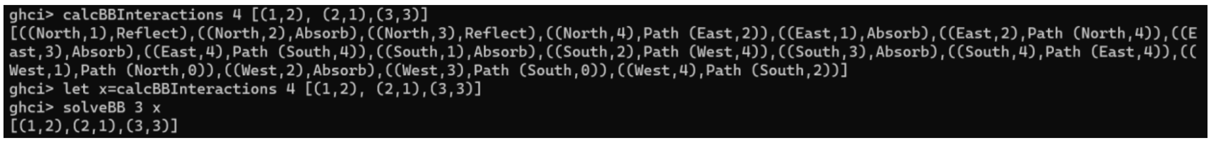
```
C:\Users\BehShuAo\Desktop\BehShuAo\CW2>ghci CW2_finalVersion.hs
Loaded package environment from C:\Users\BehShuAo\AppData\Roaming\ghc\x86_64-mingw32-9.4.7\environments\default
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[1 of 2] Compiling Main             ( CW2_finalVersion.hs, interpreted )
Ok, one module loaded.
```

4. Run "calcBBInteractions"



```
ghci> calcBBInteractions 4 [(1,2), (2,1),(3,3)]
[((North,1),Reflect),((North,2),Absorb),((North,3),Reflect),((North,4),Path (East,2)),((East,1),Absorb),((East,2),Path (North,4)),((E
ast,3),Absorb),((East,4),Path (South,4)),((South,1),Absorb),((South,2),Path (West,4)),((South,3),Absorb),((South,4),Path (East,4)),((
West,1),Path (North,0)),((West,2),Absorb),((West,3),Path (South,0)),((West,4),Path (South,2))]
```

5. Run "solveBB"



```
ghci> calcBBInteractions 4 [(1,2), (2,1),(3,3)]
[((North,1),Reflect),((North,2),Absorb),((North,3),Reflect),((North,4),Path (East,2)),((East,1),Absorb),((East,2),Path (North,4)),((E
ast,3),Absorb),((East,4),Path (South,4)),((South,1),Absorb),((South,2),Path (West,4)),((South,3),Absorb),((South,4),Path (East,4)),((
West,1),Path (North,0)),((West,2),Absorb),((West,3),Path (South,0)),((West,4),Path (South,2))]
ghci> let x=calcBBInteractions 4 [(1,2), (2,1),(3,3)]
ghci> solveBB 3 x
[(1,2),(2,1),(3,3)]
```