--Direct solver--

---Using 4*4 for demonstration---

Parameters:

rowDone=-1 //assume none of the row or col are done, if all row are doen, rowDone=dimension-1

colDone=-1

goSolveRow=true // why not solving the row first?

goSolveCol=false //I hate column

boundDBTask=dimension-2; // for 4*4, boundDBTask= 4-2=2; so we start solving 2 tiles at once when workingIndex=2

workingIndex=0; // we start working on the 0th element of the row

cocernTile[0]=rowToDoList.remove(); // now cocernTile[0]=1, cocernTile[1] will be updated before we go to state 2

HashSet<Integer> doneTile //A set to store the tiles we have done.

tileLeft=dimension*dimension //to track how many tiles left, if>9, go state1, else solve it with searching algo

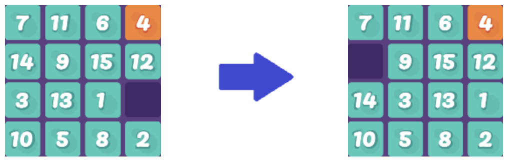
----

State 1: Solve a single tile.

<u>*Condition: If concernTile[0] has a wrong pos go P1, else go P5*</u>

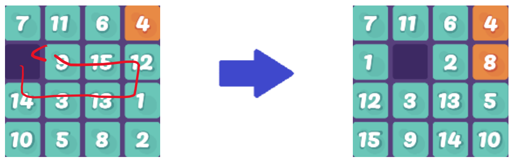
→P1: Move the zero, without considering anything,

- ➔ Use: zeroPusher
- ➔ If goSolveRow:
  - o push zero to the pos under the correct pos of concernTile[0]
- → if goSolveCol:
  - o push zero to the pos right to the correct pos of concernTile[0]



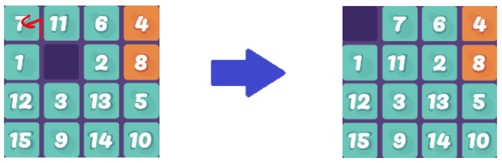
→P2: Rotate concernTile[0] to the pos of zero

- ➔ use rotate(concernTile[0])

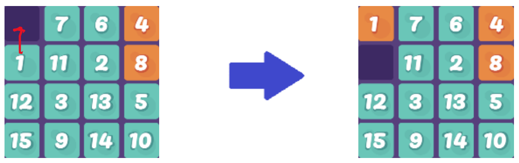


→P3: Move the zero to the correct pos of concernTile[0]

- ➔ If goSolveRow
  - o If the zero on the left of concernTile[0]: ULLDDR
  - o If the zero on the right of concernTile[0]: DR
  - o If the zero on the bottom of concernTile[0]: LDDR
  - o If the zero on the top of concernTile[0]: P4
- ➔ If goSolveCol
  - o If the zero on the right of concernTile[0]: URRD
  - o If the zero on the top pf concernTile[0]: LUURRD
  - o If the zero on the bottom of concernTile[0]: RD
  - o If the zero on the left of concernTile[0]: P4



→P4: Push the conernTile[0] to its correct pos



- ➔ If goSolveRow, singleMove('U'), else singleMove('L')
- ➔ doneTile.add(the concernTile[0])

→P5: Update task:

- ➔ if goSolveRow: cocernTile[0]=rowToDoList.remove(); // Now 2
- ➔ if goSolveCol: cocernTile[0]=colToDoList.remove();
- ➔ tileLeft- -; //now 15
- ➔ workingIndex++; //Now it is 1, so we are working on the index 1 of row 0
- ➔ if workingIndex< boundDBTask, do state 1 again
- ➔ if workingIndex== boundDBTask,
    - ○ if goSolveRow: cocernTile[1]= rowToDoList.remove();
    - ○ if goSolveCol: cocernTile[1]=colToDoList.remove();
    - ○ Go to state2

State 2: Solve two tiles at once.

Reminder: for this demonstration, we have run state 1 twice, so that concernTile[0]=3, concernTile[1]=4, workingIndex=2,doneTile={1,2}, tileLeft=14

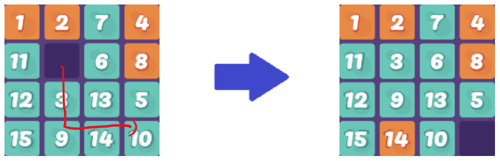
Conditions: if both the concernTile[0] and concerTile[1] have been placed in the correct pos go to P9

else If concernTile[1] is not in the bottom right corner, go to P1, else go to P3

→P1: Move the zero, without considering anything,

to the pos of bottom right corner

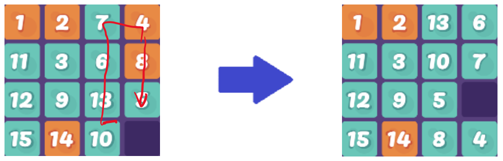
- ➔ Use: zeroPusher



→ P2: Rotate concernTile[1] to the pos of zero without

moving any of the doneTile

- ➔ use the method rotate(concernTile[1])
- ➔ Temporarily: doneTile.add(the concernTile[1])



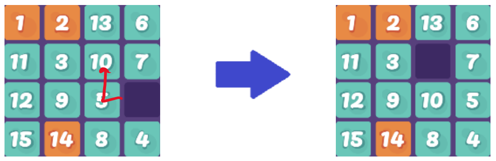
→P3: move the zero to the target without considering anything

→If goSolveRow:

- ➔ int tempRow=this.rowDone+2;   //-1+2=1
- ➔ int tempCol=this.dimention-2;   //4*4: 4-2=2

→if goSolveCol:

- ➔ int tempRow=this.dimention-2 //4*4: 4-2=2
- ➔ int tempCol=this.colDone+2;



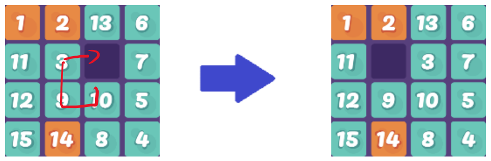
target index will be (tempRow, tempCol)

Use the method: freeZeroPusher

Conditions: if concernTile[0] has been placed in the correct pos of cocernTile[1], if goSolveRow: push zero to the pos under the correct pos of cocernTile[1]: L; Else: push zero to the pos right to the correct pos of cocernTile[1]: U; then go to P7

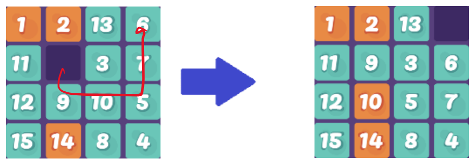
→P4: Rotate concernTile[0] to the pos of zero

- ➔ use: rotate(concernTile[0])
- ➔ doneTile.remove(concernTile[1])



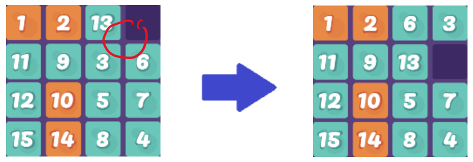
→P5: Move the zero into the correct pos of concernTile[1]   //now 4

- doneTile.remove(concernTile[0])
- If goSolveRow
  - If the zero on the left of concernTile[0]: ULLDD
  - If the zero on the bottom of concernTile[0]: LDD
  - Else: zeroPusher(correctIndex of Tile[1])
- If goSolveCol
  - If the zero on the right of concernTile[0]: URR
  - If the zero on the top pf concernTile[0]: LUURR
  - Else: zeroPusher(correctIndex of Tile[1])



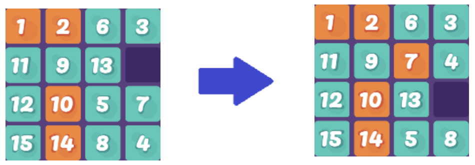
→P6: Rotate concernTile[0] to the pos of zero

- if this.goSolveRow: URDLU
- If this.goSolveCol: LDRUL



→P7: Rotate concernTile[1] to the pos of zero

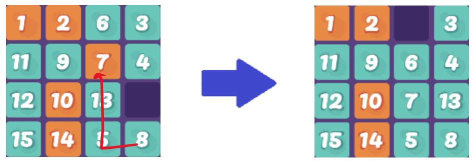
- Use: rotate(concernTile[1])



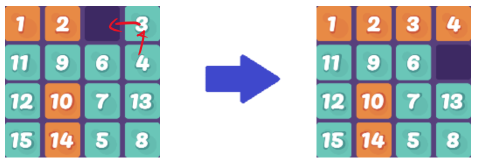
→P8: Move the zero to the correct pos of concernTile[0]   //now concernTile[0] is 3

and Move concernTile[0] and concernTile[1] to their correct pos

- goSolveRow: RDDLU
- goSolveCol: DRRUL



→P9: Update parameters



- doneTile.add(the concernTile[0])
- doneTile.add(the concernTile[1])
- tileLeft-=2; //now 12
- if tileLeft>9
  - If goSolveRow:
    - goSolveRow=false
    - goSolveCol=true
    - rowDone++;   //now 0
    - workingIndex=rowDone+1;   //now 1, so we are working on the element 1 in the col, btw 0 to n-1
    - while(!doneTIle.contains(concernTile[0]){concernTile[0]=colToDoQueue.remove();}
  - goSolveCol:
    - goSolveRow=true
    - goSolveCol=false
    - colDone++;
    - workingIndex=colDone+1;
    - while(!doneTIle.contains(concernTile[0]){concernTile[0]=rowToDoQueue.remove();}
  - if tileLeft>9, will go back to state 1 for the col or row,
- Else if tileLeft=9: 3*3 on the bottom right corner, solve it by any searching algo (or develop an algo for 3*3)