**Algorithm**:

up(), left(),right(),down() function are used to move up,left,right and down respectively. copy() is used for clone.

**ModifiedHillClimb(** game , currlevel, level ):

1. **if** currlevel == level **then**

2. return score(game)

3. **end if**

4. culscore = 0

5. game.playup()

6. game.addNewTile()

7. culscore += **ModifiedHillClimb**(game.copy() , currlevel+1, level)

8. game.playup()

9. game.addNewTile()

10. culscore += **ModifiedHillClimb**(game.copy() , currlevel+1, level)

11. game.playleft()

12. game.addNewTile()

13. culscore += **ModifiedHillClimb**(game.copy() , currlevel+1, level)

14. game.playright()

15. game.addNewTile()

16. culscore += **ModifiedHillClimb**(game.copy() , currlevel+1, level)

17. game.playdown()

18. game.addNewTile()

19. culscore += **ModifiedHillClimb**(game.copy() , currlevel+1, level)

20. r**eturn** culscore

**GetMove**(game, level, iter):

1. gup = 0

2. gleft = 0

3. gright = 0

4. gdown = 0

5. n = 50

6. i = 0

7. **while** i < iter:

8. **do**

9. gamecopy = game.copy()

10. gamecopy.up()

11. gamecopy.addNewTile()

12. gup += **ModifiedHillClimb**( gamecopy, 0 , level – 1)

10. gamecopy.left()

11. gamecopy.addNewTile()

12. gleft += **ModifiedHillClimb**( gamecopy, 0 , level – 1)

10. gamecopy.right()

11. gamecopy.addNewTile()

12. gright += **ModifiedHillClimb**( gamecopy, 0 , level – 1)

10. gamecopy.down()

11. gamecopy.addNewTile()

12. gdown += **ModifiedHillClimb**( gamecopy, 0 , level – 1)

13. i = i + 1

14. **end while**

15. **return** move corresponding to max(gup,gleft,gright,gdown)

**Experimentation**:

* The most important part of the modified hill climbing algorithm is the heuristic function used.
* Lot of experimentation was done with the way by which this heuristic score is calculated.
* Some of them are shown below:

1. Score was calculated by multiplying the board by

256 225 196 169

81 100 121 144

64 49 36 25

1 4 9 16

2. Score was calculated by using (1) and adding (b-i)2  for each blank space where i is the depth of game. Many different values were used for b and b=11 gave better results

3. By multiplying each tile by its neighbouring tiles:

score = Ʃ ai,j \* ai + 1, j + ai \* ai – 1, j + ai,j \* ai , j + 1+ai \* ai, j - 1 + ai,j \* ai + 1 ,j + 1 ai,j\* ai + 1,j - 1 ai \* ai – 1, j + 1 + ai,j \* ai - 1, i + j

4. Using (2) for games with blanks > 2 and (4) for games with blanks <= 2

* Out of this (4) one gave best results as number blanks spaces were important in the early stages and later on they don’t matter we just need big numbers close to each other.
* Previously, for finding the move with max heuristic value each move was played only once ignoring the random position of new tiles added. Hence, these logic didn’t account for that randomness. In order to consider this randomness, each move was played for n number of times and then the move with max cumulative heuristic value was chosen. This gave better results.

**Results**:

Game was played for 100 times and the highest tile was recorded for each game

|  |  |  |  |
| --- | --- | --- | --- |
| **Heuristic used** | 2048 | 1024 | 512 |
| Heuristic 2 without iter | 0 | 39 | 61 |
| Heuristic 4 without iter | 3 | 55 | 42 |
| Heuristic 4 with iter | 21 | 77 | 2 |

**Conclusion:**

2048 could be solved by using a simple modified version of Hill climbing algorithm. Also, heuristic function plays very important role in determining the accuracy of the algorithm. Best heuristic function can be tuned by knowing of some basic strategies of the game like sorting the tiles in zig-zag manner and by rewarding spaces. The optimum heuristic value function changes at different time of the game.