CS-461
Artificial Intelligence
Homework #4

Group Name: *Enigma*

Group Members:
1. Kerem Ayöz - 21501569 / CS
2. Ege Yosunkaya - 21402025 / CS
3. Erkut Alakuş - 21101413 / CS

Sample Problem

1	2	4	8
5	6	7	3
13	11	-1	12
10	9	14	15

Path to goal using heuristic 1, 15 Steps

1	2	4	8
5	6	7	3
13	11	-1	12
10	9	14	15

1	2	4	8
5	6	-1	3
13	11	7	12
10	9	14	15

1	2	4	8
5	6	3	-1
13	11	7	12
10	9	14	15

1	2	4	-1
5	6	3	8
13	11	7	12
10	9	14	15

1	2	-1	4
5	6	3	8
13	11	7	12
10	9	14	15

1	2	3	4
5	6	-1	8
13	11	7	12
10	9	14	15

1	2	3	4
5	6	7	8
13	11	-1	12
10	9	14	15

1	2	3	4
5	6	7	8
13	-1	11	12
10	9	14	15

1	2	3	4
5	6	7	8
13	9	11	12
10	-1	14	15

1	2	3	4
5	6	7	8
13	9	11	12

−1	10	14	15

1	2	3	4
5	6	7	8
−1	9	11	12
13	10	14	15

1	2	3	4
5	6	7	8
9	−1	11	12
13	10	14	15

1	2	3	4
5	6	7	8
9	10	11	12
13	−1	14	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	−1	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	−1

Path to goal using heuristic 1, 15 Steps

```
--------------------
|  1 |  2 |  4 |  8 |
--------------------
|  5 |  6 |  7 |  3 |
--------------------
| 13 | 11 | -1 | 12 |
--------------------
| 10 |  9 | 14 | 15 |
--------------------

--------------------
|  1 |  2 |  4 |  8 |
--------------------
|  5 |  6 | -1 |  3 |
--------------------
| 13 | 11 |  7 | 12 |
--------------------
| 10 |  9 | 14 | 15 |
--------------------

--------------------
|  1 |  2 |  4 |  8 |
--------------------
|  5 |  6 |  3 | -1 |
--------------------
| 13 | 11 |  7 | 12 |
--------------------
| 10 |  9 | 14 | 15 |
--------------------

--------------------
|  1 |  2 |  4 | -1 |
--------------------
|  5 |  6 |  3 |  8 |
--------------------
| 13 | 11 |  7 | 12 |
--------------------
| 10 |  9 | 14 | 15 |
--------------------

--------------------
|  1 |  2 | -1 |  4 |
--------------------
|  5 |  6 |  3 |  8 |
--------------------
| 13 | 11 |  7 | 12 |
--------------------
| 10 |  9 | 14 | 15 |
--------------------

--------------------
|  1 |  2 |  3 |  4 |
--------------------
|  5 |  6 | -1 |  8 |
```

```
--------------------
| 13 | 11 |  7 | 12 |
--------------------
| 10 |  9 | 14 | 15 |
--------------------
```

```
--------------------
|  1 |  2 |  3 |  4 |
--------------------
|  5 |  6 |  7 |  8 |
--------------------
| 13 | 11 | -1 | 12 |
--------------------
| 10 |  9 | 14 | 15 |
--------------------
```

```
--------------------
|  1 |  2 |  3 |  4 |
--------------------
|  5 |  6 |  7 |  8 |
--------------------
| 13 | -1 | 11 | 12 |
--------------------
| 10 |  9 | 14 | 15 |
--------------------
```

```
--------------------
|  1 |  2 |  3 |  4 |
--------------------
|  5 |  6 |  7 |  8 |
--------------------
| 13 |  9 | 11 | 12 |
--------------------
| 10 | -1 | 14 | 15 |
--------------------
```

```
--------------------
|  1 |  2 |  3 |  4 |
--------------------
|  5 |  6 |  7 |  8 |
--------------------
| 13 |  9 | 11 | 12 |
--------------------
| -1 | 10 | 14 | 15 |
--------------------
```

```
--------------------
|  1 |  2 |  3 |  4 |
--------------------
|  5 |  6 |  7 |  8 |
--------------------
| -1 |  9 | 11 | 12 |
--------------------
| 13 | 10 | 14 | 15 |
--------------------
```

1	2	3	4
5	6	7	8
9	-1	11	12
13	10	14	15

1	2	3	4
5	6	7	8
9	10	11	12
13	-1	14	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	-1	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	-1

Distinct Puzzles

Puzzle 1

1	2	-1	3
5	6	7	4
13	9	11	8
10	14	15	12

Steps to achieve goal using heuristic 1: 13

Steps to achieve goal using heuristic 2: 13

Puzzle 2

1	2	3	4
6	7	14	8
5	9	-1	10
13	11	15	12

Steps to achieve goal using heuristic 1: 13

Steps to achieve goal using heuristic 2: 13

Puzzle 3

1	6	2	8
5	-1	4	3
9	10	7	12
13	14	11	15

Steps to achieve goal using heuristic 1: 11

Steps to achieve goal using heuristic 2: 11

Puzzle 4

1	11	4	7
5	3	2	8
10	6	15	12
9	13	14	−1

Steps to achieve goal using heuristic 1: 11

Steps to achieve goal using heuristic 2: 11

Puzzle 5

1	2	3	4
5	15	6	8
10	7	14	−1
9	13	12	11

Steps to achieve goal using heuristic 1: 11

Steps to achieve goal using heuristic 2: 11

Puzzle 6

6	1	3	4
5	2	7	8
−1	9	11	12
13	10	14	15

Steps to achieve goal using heuristic 1: 11

Steps to achieve goal using heuristic 2: 11

Puzzle 7

1	2	3	4
9	5	7	8
6	-1	11	12
13	10	14	15

Steps to achieve goal using heuristic 1: 8

Steps to achieve goal using heuristic 2: 8

Puzzle 8

1	2	7	3
5	6	8	-1
9	14	4	12
13	11	10	15

Steps to achieve goal using heuristic 1: 8

Steps to achieve goal using heuristic 2: 8

Puzzle 9

2	5	3	4
1	6	7	-1
9	10	11	8
13	14	15	12

Steps to achieve goal using heuristic 1: 8

Steps to achieve goal using heuristic 2: 8

Puzzle 10

2	6	8	3
1	7	4	12
5	10	11	-1
9	13	14	15

Steps to achieve goal using heuristic 1: 8

Steps to achieve goal using heuristic 2: 8

Puzzle 11

1	2	4	7
5	6	11	3
9	10	-1	8
13	14	15	12

Steps to achieve goal using heuristic 1: 4

Steps to achieve goal using heuristic 2: 4

Puzzle 12

5	1	2	4
9	3	7	8
10	6	-1	11
13	14	15	12

Steps to achieve goal using heuristic 1: 13

Steps to achieve goal using heuristic 2: 13

Puzzle 13

1	2	4	-1
9	7	3	8
6	5	10	15
13	14	12	11

Steps to achieve goal using heuristic 1: 16

Steps to achieve goal using heuristic 2: 16

Puzzle 14

2	3	8	7
1	6	4	12
5	-1	11	15
9	10	13	14

Steps to achieve goal using heuristic 1: 20

Steps to achieve goal using heuristic 2: 20

Puzzle 15

5	-1	3	4
2	1	6	8
9	10	7	12
13	14	11	15

Steps to achieve goal using heuristic 1: 5

Steps to achieve goal using heuristic 2: 5

Puzzle 16

1	2	3	-1
6	10	11	4
9	15	8	7
13	5	14	12

Steps to achieve goal using heuristic 1: 12

Steps to achieve goal using heuristic 2: 12

Puzzle 17

5	2	3	4
6	-1	11	7
9	1	10	8
13	14	15	12

Steps to achieve goal using heuristic 1: 12

Steps to achieve goal using heuristic 2: 12

Puzzle 18

5	-1	2	3
9	1	7	4
14	6	11	8
10	13	15	12

Steps to achieve goal using heuristic 1: 16

Steps to achieve goal using heuristic 2: 16

Puzzle 19

2	3	7	4
1	6	12	8
5	10	-1	15
9	13	14	11

Steps to achieve goal using heuristic 1: 7

Steps to achieve goal using heuristic 2: 7

Puzzle 20

1	2	4	8
5	6	12	-1
9	10	3	7
13	14	11	15

Steps to achieve goal using heuristic 1: 7

Steps to achieve goal using heuristic 2: 7

Puzzle 21

6	1	3	4
2	9	7	8
10	15	14	11
5	13	12	-1

Steps to achieve goal using heuristic 1: 10

Steps to achieve goal using heuristic 2: 10

Puzzle 22

1	2	3	4
5	6	11	7
-1	10	13	8
9	14	15	12

Steps to achieve goal using heuristic 1: 10

Steps to achieve goal using heuristic 2: 10

Puzzle 23

1	2	11	7
5	6	8	3
9	14	10	12
13	-1	4	15

Steps to achieve goal using heuristic 1: 10

Steps to achieve goal using heuristic 2: 10

Puzzle 24

1	7	4	8
5	3	10	2
9	14	6	12
13	-1	11	15

Steps to achieve goal using heuristic 1: 9

Steps to achieve goal using heuristic 2: 9

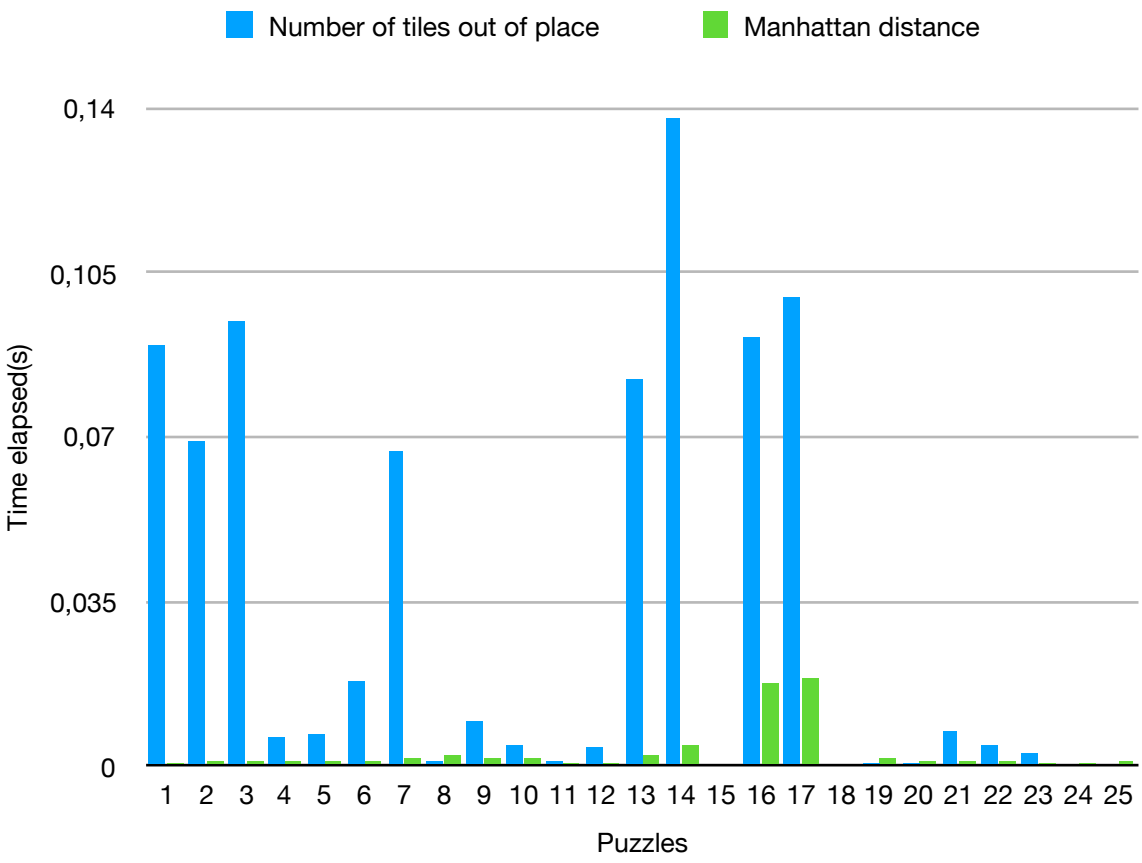
Puzzle 25

−1	2	3	4
1	6	10	7
5	11	14	8
9	13	15	12

Steps to achieve goal using heuristic 1: 9

Steps to achieve goal using heuristic 2: 9

Time vs Puzzles Plot of different heuristic functions



We used two different heuristic functions while implementing A* Search. Both of these heuristics are admissible heuristics which means they never overestimates the remaining path from to goal from current state.

We used different number of backward moves while shuffling the puzzle. Shuffling started from the goal state and program makes random number of moves from that state. After generations 25 such puzzles we measured the CPU time of the execution of the program with 2 distinct heuristics.

As it could be seen from the graph, CPU times really inconsistent because of the computer conditions(compiler optimizations, external programs, memory organization). However, as a result we found out that heuristic 2 which calculates the sum of the Manhattan distances of tiles give better results on the average in terms of CPU time while solving 15-Puzzle with A* search. Because the count of the misplaced tiles sometimes directs the program to irrelevant parts of the solution tree. Manhattan distance is better in terms of that situation.

CODE

A_Star.py

```
from State import State
from puzzleGenerator import PuzzleGenerator
from random import randint
import copy


class A_Star:

    # Store all paths visited in queue
    queue = []

    def a_star_search(self, start, goal, heuristic_id,):
        # Form a one element queue consisting of start
        self.queue.append([start])
        # While queue is not empty
        while(self.queue):

            # Select the min score path to extend its last node
            if heuristic_id == 1:
                minIndexInQueue = self.queue.index(min(self.queue,
key=lambda o: len(o) + State.heuristic1(o[-1])))
            else:
                minIndexInQueue = self.queue.index(min(self.queue,
key=lambda o: len(o) + State.heuristic2(o[-1])))
            stateWillBeExpanded = self.queue[minIndexInQueue][-1]

            # If goal node is found in front of the queue,
announce success
            if stateWillBeExpanded == goal:
                print("Success: " +
str(len(self.queue[minIndexInQueue])) + "\n")
                '''
                i = 1
                for goalPathStates in self.queue[minIndexInQueue]:
                    print("Step: " + str(i))
                    i += 1
                    print(goalPathStates)
                '''
                return

            # Expand the state with min score
            newStates = stateWillBeExpanded.makeMove()

            # Remove the cycling paths
            for a in newStates:
```

```
                if a in self.queue[minIndexInQueue]:
                    newStates.remove(a)

            # Add the newly expanded paths
            for states in newStates:

                # Build the new paths with newStates in the
terminal position
                expandedPath =
copy.deepcopy(self.queue[minIndexInQueue])
                expandedPath.append(states)
                exists = False

                # If queue currently consists a path with same
terminal node and least cost, do not add new extended path
                for i in range(len(self.queue)):
                    for j in range(len(self.queue[i])):
                        if states == self.queue[i][j]:
                            exists = True
                            if j + 1 > len(expandedPath):
                                self.queue[i] = expandedPath
                                break

                # If it is not in the queue, means not expanded or
added before, simply add it to queue
                if not exists:
                    self.queue.append(expandedPath)

            # Delete the first path in queue
            del self.queue[minIndexInQueue]
```

puzzleGenerator.py

```
from State import State
from random import randint
import time


class PuzzleGenerator:

    def generatePuzzles(puzzleCount):
        newPuzzles = []
        for i in range(puzzleCount):
            start = State(State.goalState)
            newStates = []
            expandeds = [start]
            count = 0
            for i in range(randint(15, 20)):
                newStates = start.makeMove()
                start = newStates[randint(0, len(newStates) - 1)]
```

```
                while start in expandeds:
                    start = newStates[randint(0, len(newStates) -
1)]
                expandeds.append(start)
                count += 1
            newPuzzles.append(start)
        return newPuzzles
```

State.py

```
class State:

    grid = []

    goalState = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13,
14, 15, -1]]

    def __init__(self, grid):
        self.grid = grid

    def __eq__(self, other):
        for i in range(0, 4):
            for j in range(0, 4):
                if self.grid[i][j] != other.grid[i][j]:
                    return False
        return True
    # Emege saygı..

    def __str__(self):
        string = "--------------------\n"
        string += "| " + ((1 - abs((self.grid[0][0] // 10))) * "
") + str(self.grid[0][0]) + " | " + ((1 - abs((self.grid[0][1] //
10))) * " ") + str(self.grid[0][1])
        string += " | " + ((1 - abs((self.grid[0][2] // 10))) * "
") + str(self.grid[0][2]) + " | " + ((1 - abs((self.grid[0][3] //
10))) * " ") + str(self.grid[0][3]) + " |\n"
        string += "--------------------\n"
        string += "| " + ((1 - abs((self.grid[1][0] // 10))) * "
") + str(self.grid[1][0]) + " | " + ((1 - abs((self.grid[1][1] //
10))) * " ") + str(self.grid[1][1])
        string += " | " + ((1 - abs((self.grid[1][2] // 10))) * "
") + str(self.grid[1][2]) + " | " + ((1 - abs((self.grid[1][3] //
10))) * " ") + str(self.grid[1][3]) + " |\n"
        string += "--------------------\n"
        string += "| " + ((1 - abs((self.grid[2][0] // 10))) * "
") + str(self.grid[2][0]) + " | " + ((1 - abs((self.grid[2][1] //
10))) * " ") + str(self.grid[2][1])
        string += " | " + ((1 - abs((self.grid[2][2] // 10))) * "
") + str(self.grid[2][2]) + " | " + ((1 - abs((self.grid[2][3] //
10))) * " ") + str(self.grid[2][3]) + " |\n"
        string += "--------------------\n"
```

```
        string += "| " + ((1 - abs((self.grid[3][0] // 10))) * "
") + str(self.grid[3][0]) + " | " + ((1 - abs((self.grid[3][1] //
10))) * " ") + str(self.grid[3][1])
        string += " | " + ((1 - abs((self.grid[3][2] // 10))) * "
") + str(self.grid[3][2]) + " | " + ((1 - abs((self.grid[3][3] //
10))) * " ") + str(self.grid[3][3]) + " |\n"
        string += "--------------------\n"
        return string

    def heuristic1(self):
        count = 0
        for i in range(0, 4):
            for j in range(0, 4):
                if self.grid[i][j] != self.goalState[i][j]:
                    count += 1
        return count

    def heuristic2(self):
        count = 0
        manhattan_map = {
            1: (0, 0),
            2: (0, 1),
            3: (0, 2),
            4: (0, 3),
            5: (1, 0),
            6: (1, 1),
            7: (1, 2),
            8: (1, 3),
            9: (2, 0),
            10: (2, 1),
            11: (2, 2),
            12: (2, 3),
            13: (3, 0),
            14: (3, 1),
            15: (3, 2),
            -1: (3, 3),
        }

        for i in range(0, 4):
            for j in range(0, 4):
                count += abs(i - manhattan_map[self.grid[i][j]]
[0])
                count += abs(j - manhattan_map[self.grid[i][j]]
[1])
        return count

    def makeMove(self):

        x = -1
        y = -1

        for i in range(0, 4):
```

```
    for j in range(0, 4):
        if self.grid[i][j] == -1:
            x = i
            y = j
            break

newStates = []
tempGrid = [row[:] for row in self.grid]
tempGrid2 = [row[:] for row in self.grid]
tempGrid3 = [row[:] for row in self.grid]
tempGrid4 = [row[:] for row in self.grid]

if x == 0 and y == 0:
    tempGrid[0][0] = tempGrid[0][1]
    tempGrid[0][1] = -1
    newStates.append(State(tempGrid))

    tempGrid2[0][0] = tempGrid2[1][0]
    tempGrid2[1][0] = -1
    newStates.append(State(tempGrid2))
elif x == 0 and y == 1:
    tempGrid[0][1] = tempGrid[0][0]
    tempGrid[0][0] = -1
    newStates.append(State(tempGrid))

    tempGrid2[0][1] = tempGrid2[1][1]
    tempGrid2[1][1] = -1
    newStates.append(State(tempGrid2))

    tempGrid3[0][1] = tempGrid3[0][2]
    tempGrid3[0][2] = -1
    newStates.append(State(tempGrid3))
elif x == 0 and y == 2:
    tempGrid[0][2] = tempGrid[0][1]
    tempGrid[0][1] = -1
    newStates.append(State(tempGrid))

    tempGrid2[0][2] = tempGrid2[0][3]
    tempGrid2[0][3] = -1
    newStates.append(State(tempGrid2))

    tempGrid3[0][2] = tempGrid3[1][2]
    tempGrid3[1][2] = -1
    newStates.append(State(tempGrid3))
elif x == 0 and y == 3:
    tempGrid[0][3] = tempGrid[0][2]
    tempGrid[0][2] = -1
    newStates.append(State(tempGrid))

    tempGrid2[0][3] = tempGrid2[1][3]
    tempGrid2[1][3] = -1
    newStates.append(State(tempGrid2))
```

```
elif x == 1 and y == 0:
    tempGrid[1][0] = tempGrid[0][0]
    tempGrid[0][0] = -1
    newStates.append(State(tempGrid))

    tempGrid2[1][0] = tempGrid2[2][0]
    tempGrid2[2][0] = -1
    newStates.append(State(tempGrid2))

    tempGrid3[1][0] = tempGrid3[1][1]
    tempGrid3[1][1] = -1
    newStates.append(State(tempGrid3))

elif x == 1 and y == 1:
    tempGrid[1][1] = tempGrid[0][1]
    tempGrid[0][1] = -1
    newStates.append(State(tempGrid))

    tempGrid2[1][1] = tempGrid2[1][2]
    tempGrid2[1][2] = -1
    newStates.append(State(tempGrid2))

    tempGrid3[1][1] = tempGrid3[2][1]
    tempGrid3[2][1] = -1
    newStates.append(State(tempGrid3))

    tempGrid4[1][1] = tempGrid4[1][0]
    tempGrid4[1][0] = -1
    newStates.append(State(tempGrid4))

elif x == 1 and y == 2:
    tempGrid[1][2] = tempGrid[0][2]
    tempGrid[0][2] = -1
    newStates.append(State(tempGrid))

    tempGrid2[1][2] = tempGrid2[1][3]
    tempGrid2[1][3] = -1
    newStates.append(State(tempGrid2))

    tempGrid3[1][2] = tempGrid3[2][2]
    tempGrid3[2][2] = -1
    newStates.append(State(tempGrid3))

    tempGrid4[1][2] = tempGrid4[1][1]
    tempGrid4[1][1] = -1
    newStates.append(State(tempGrid4))

elif x == 1 and y == 3:
    tempGrid[1][3] = tempGrid[0][3]
    tempGrid[0][3] = -1
    newStates.append(State(tempGrid))
```

```
        tempGrid2[1][3] = tempGrid2[2][3]
        tempGrid2[2][3] = -1
        newStates.append(State(tempGrid2))

        tempGrid3[1][3] = tempGrid3[1][2]
        tempGrid3[1][2] = -1
        newStates.append(State(tempGrid3))

    elif x == 2 and y == 0:
        tempGrid[2][0] = tempGrid[1][0]
        tempGrid[1][0] = -1
        newStates.append(State(tempGrid))

        tempGrid2[2][0] = tempGrid2[3][0]
        tempGrid2[3][0] = -1
        newStates.append(State(tempGrid2))

        tempGrid3[2][0] = tempGrid3[2][1]
        tempGrid3[2][1] = -1
        newStates.append(State(tempGrid3))

    elif x == 2 and y == 1:
        tempGrid[2][1] = tempGrid[2][0]
        tempGrid[2][0] = -1
        newStates.append(State(tempGrid))

        tempGrid2[2][1] = tempGrid2[2][2]
        tempGrid2[2][2] = -1
        newStates.append(State(tempGrid2))

        tempGrid3[2][1] = tempGrid3[1][1]
        tempGrid3[1][1] = -1
        newStates.append(State(tempGrid3))

        tempGrid4[2][1] = tempGrid4[3][1]
        tempGrid4[3][1] = -1
        newStates.append(State(tempGrid4))

    elif x == 2 and y == 2:
        tempGrid[2][2] = tempGrid[2][1]
        tempGrid[2][1] = -1
        newStates.append(State(tempGrid))

        tempGrid2[2][2] = tempGrid2[2][3]
        tempGrid2[2][3] = -1
        newStates.append(State(tempGrid2))

        tempGrid3[2][2] = tempGrid3[1][2]
        tempGrid3[1][2] = -1
        newStates.append(State(tempGrid3))
```

```
            tempGrid4[2][2] = tempGrid4[3][2]
            tempGrid4[3][2] = -1
            newStates.append(State(tempGrid4))

        elif x == 2 and y == 3:
            tempGrid[2][3] = tempGrid[2][2]
            tempGrid[2][2] = -1
            newStates.append(State(tempGrid))

            tempGrid2[2][3] = tempGrid2[1][3]
            tempGrid2[1][3] = -1
            newStates.append(State(tempGrid2))

            tempGrid3[2][3] = tempGrid3[3][3]
            tempGrid3[3][3] = -1
            newStates.append(State(tempGrid3))

        elif x == 3 and y == 0:
            tempGrid[3][0] = tempGrid[3][1]
            tempGrid[3][1] = -1
            newStates.append(State(tempGrid))

            tempGrid2[3][0] = tempGrid2[2][0]
            tempGrid2[2][0] = -1
            newStates.append(State(tempGrid2))

        elif x == 3 and y == 1:
            tempGrid[3][1] = tempGrid[3][2]
            tempGrid[3][2] = -1
            newStates.append(State(tempGrid))

            tempGrid2[3][1] = tempGrid2[3][0]
            tempGrid2[3][0] = -1
            newStates.append(State(tempGrid2))

            tempGrid3[3][1] = tempGrid3[2][1]
            tempGrid3[2][1] = -1
            newStates.append(State(tempGrid3))

        elif x == 3 and y == 2:

            tempGrid[3][2] = tempGrid[3][3]
            tempGrid[3][3] = -1
            newStates.append(State(tempGrid))

            tempGrid2[3][2] = tempGrid2[3][1]
            tempGrid2[3][1] = -1
            newStates.append(State(tempGrid2))

            tempGrid3[3][2] = tempGrid3[2][2]
            tempGrid3[2][2] = -1
            newStates.append(State(tempGrid3))
```

```
        elif x == 3 and y == 3:
            tempGrid[3][3] = tempGrid[2][3]
            tempGrid[2][3] = -1
            newStates.append(State(tempGrid))

            tempGrid2[3][3] = tempGrid2[3][2]
            tempGrid2[3][2] = -1
            newStates.append(State(tempGrid2))

        return newStates
```

Solve.py

```
from A_Star import A_Star
from puzzleGenerator import PuzzleGenerator
from random import randint
from State import State
import time
import copy

roots = PuzzleGenerator.generatePuzzles(25)
timesh1 = []
timesh2 = []

for i in range(len(roots)):
    solver = A_Star()
    print("Puzzle " + str(i + 1))
    print(roots[i])
    goalState = State([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11,
12], [13, 14, 15, -1]])

    t1 = time.time()
    A_Star.a_star_search(solver, roots[i], goalState, 1)
    t2 = time.time()
    timesh1.append(t2 - t1)

    t1 = time.time()
    A_Star.a_star_search(solver, roots[i], goalState, 2)
    t2 = time.time()
    timesh2.append(t2 - t1)

print("HEURISTIC 1")
for i in range(len(timesh1)):
    print("Puzzle : " + str(i) + ": " + str(timesh1[i]))

print("HEURISTIC 2")
for i in range(len(timesh2)):
    print("Puzzle : " + str(i) + ": " + str(timesh2[i]))
```