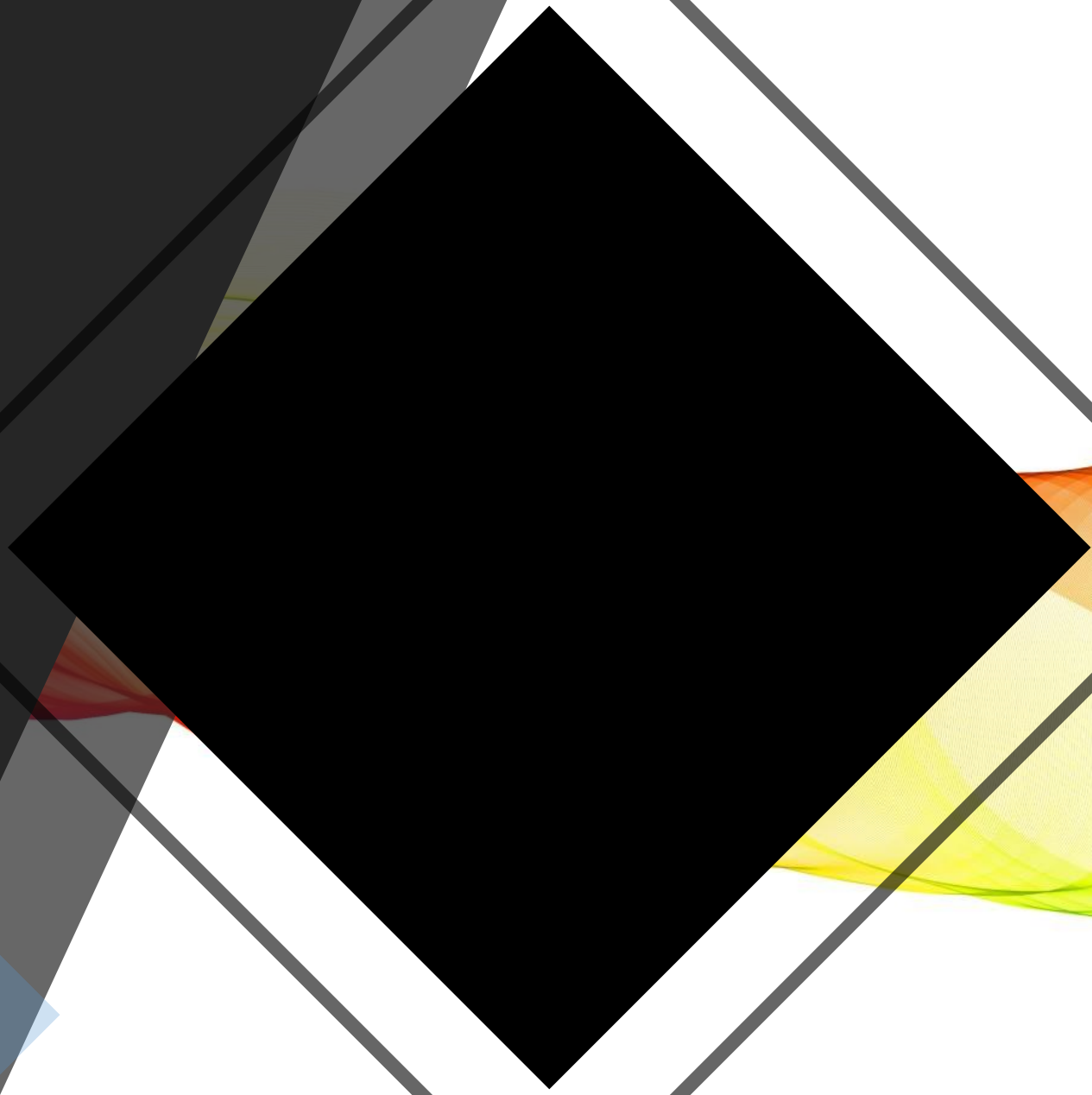
Assignment #2

- Tse-Chun Lau 29676279
- The Phi Nguyen 40015643
- Hong Phuc Nguyen 40082916



Rules and Definitions

- U -> up cost:1
- D -> down cost:1
- L -> left cost:1
- R -> right cost:1
- X, Y, Z, T -> wrapping move Up, Down, Left, Right, respectively. Cost:2
- C -> diagonal move cost:3

Uniform Cost

```
def uniformCost(self):
    search_path = []
    actual = self.start
    leaves = PriorityQueue()
    leaves.put((0, actual))
    closed = list()
    start_time = time.time()
    while True:
        if (time.time() - start_time > self.limit*2):
            return ("no solution", "no solution", self.limit*2)
        if leaves.empty():
            return None
        actual = leaves.get()[1]
        search_path.append(f"{actual.gn} {actual.gn} {0} | {str(actual.state)}\n")
        if actual.goalState():
            actual.getSolution(self.org)
            return (actual, search_path, time.time() - start_time)
        elif actual.state.puzzle not in closed:
            closed.append(actual.state.puzzle)
            succ = actual.succ()
            while not succ.empty():
                child = succ.get()
                leaves.put((child.gn, child))
```

Greedy Best First Seach (GBFS)

```
def greedyBFS(self, heuristic):
    search_path = []
    actual = self.start
    leaves = PriorityQueue()
    leaves.put((actual.costHeur(heuristic), actual))
    closed = list()
    start_time = time.time()
    while True:
        if (time.time() - start_time > self.limit):
            return ("no solution", "no solution", self.limit)
        if leaves.empty():
            return None
        actual = leaves.get()[1]
        search_path.append(f"{0 + actual.costHeur(heuristic)} {0} {actual.costHeur(heuristic)} | {str(actual.state)}\n")
        if actual.goalState():
            actual.getSolution(self.org)
            return (actual, search_path, time.time() - start_time)
        elif actual.state.puzzle not in closed:
            closed.append(actual.state.puzzle)
            succ = actual.succ()
            while not succ.empty():
                child = succ.get()
                leaves.put((child.costHeur(heuristic), child))
```

Algorithms A*

```
def aStar(self, heuristic):
    search_path = []
    actual = self.start
    leaves = PriorityQueue()
    leaves.put((actual.costHeur(heuristic), actual))
    closed = list()
    start_time = time.time()
    while True:
        if (time.time() - start_time > self.limit):
            return ("no solution", "no solution", self.limit)
        if leaves.empty():
            return None
        actual = leaves.get()[1]
        search_path.append(f"{actual.gn + actual.costHeur(heuristic)} {actual.gn} {actual.costHeur(heuristic)} | {str(actual.state)}\n")
        if actual.goalState():
            actual.getSolution(self.org)
            return (actual, search_path, time.time() - start_time)
        elif actual.state.puzzle not in closed:
            closed.append(actual.state.puzzle)
            succ = actual.succ()
            while not succ.empty():
                child = succ.get()
                leaves.put((child.costHeur(heuristic)+child.gn, child))
```

Heuristics – Hamming Distance

- Hamming Distance
  - Every time a tile is at the wrong place, we add 1 to the Heuristic

```
def hammingDistance(self):
    result = [0, 0]
    count = 1
    for i in range(0, self.state.row):
        for j in range(0, self.state.col):
            if self.state.zero != (i, j) and self.state.puzzle[i][j] != (count % (self.state.row*self.state.col)):
                result[0] += 1
            count += 1
    count = 1
    for j in range(0, self.state.col):
        for i in range(0, self.state.row):
            if self.state.zero != (i, j) and self.state.puzzle[i][j] != (count % (self.state.row*self.state.col)):
                result[1] += 1
            count += 1
    if result[0] < result[1]:
        return result[0]
    else:
        return result[1]
```

Heuristics – Manhattan Distance

- Manhattan Distance
  - The distance of incorrect tile from their current position (aka the number of move required to get the right tile)

```
def manhattanDistance(self):
    result = [0, 0]
    distance = [0, 0]
    for i in range(0, self.state.row):
        for j in range(0, self.state.col):
            index = self.state.puzzle[i][j] - 1
            if index == -1:
                distance[0] = (self.state.row-1-i)+(self.state.col-1-j)
                distance[1] = (self.state.row-1-i)+(self.state.col-1-j)
            else:
                distance[0] = abs(i-(index//self.state.col)) + \
                    abs(j-(index % self.state.col))
                distance[1] = abs(i-(index % self.state.row)) + \
                    abs(j-(index//self.state.row))
            result[0] += distance[0]
            result[1] += distance[1]
    if result[0] < result[1]:
        return result[0]
    else:
        return result[1]
```

Analysis

```
# stat1

total of 50:
 length solution length search  cost-g(n)      time(s)         %found solution
[[3.52000000e+02 3.83581000e+05 3.91000000e+02 2.77701256e+02  6.40000000e-01]          ucs
 [8.24000000e+02 3.14160000e+04 1.04900000e+03 6.33293962e+00  1.00000000e+00]          gbfs-h0
 [9.88000000e+02 1.83310000e+04 1.13100000e+03 3.58743954e+00  1.00000000e+00]          gbfs-h1
 [5.86000000e+02 8.61830000e+04 6.69000000e+02 3.17046099e+01  1.00000000e+00]          astar-h0
 [6.06000000e+02 2.67760000e+04 6.75000000e+02 6.81360173e+00  1.00000000e+00]]         astar-h1

 Average by 50:
 length solution length search  cost-g(n)      time(s)         %found solution
[[0.07040000e+02 7671.62000e+00 7.82000000e+00 5.55402512e+00  6.40000000e-01]          ucs
 [0.16780000e+02 628.320000e+00 20.9800000e+00 0.12665879e+00  1.00000000e+00]          gbfs-h0
 [0.19760000e+02 366.620000e+00 22.6200000e+00 0.07174879e+00  1.00000000e+00]          gbfs-h1
 [0.11720000e+02 1723.66000e+00 13.3800000e+00 0.06340921e+00  1.00000000e+00]          astar-h0
 [0.12120000e+02 535.520000e+00 13.5000000e+00 0.13627203e+00  1.00000000e+00]]         astar-h1
```

As you can see, ***Algorithms A* > greedy best first search > Uniform Cost Search***

- UCS
  - took **longer** to search and find a solution
  - Sometimes, it does not find a solution at all. (64%)
  - It also has the **lowest cost**
  - It has the **lowest solution** found
- GBFS
  - It was faster to search and find a solution than the UCS
  - In consequence of a faster search, It took a higher cost to find a solution than UCS
- Algorithm A*
  - It has a lower cost then GBFS but higher than UCS. (maybe because UCS does not take H(n) into consideration)
  - It was slightly slower than GBFS
  - It is faster to search and find a solution than GBFS but slower than UCS (maybe same reason mention above)

Scale Up

```
# stat2

sample[row, col] time(s)
***             sample [2, 2]           ***
LC time: 0.000997781753540039
***             sample [2, 3]           ***
RDLUCLLZ time: 0.04999351501464844
***             sample [2, 4]           ***
DCLLLDCLDLLZ time: 0.4590029716491699
***             sample [2, 5]           ***
LZLLURDLLULDZLLULDRURDR time: 10.068998336791992
***             sample [2, 6]           ***
no solution time: 300
***             sample [3, 2]           ***
UULXURD time: 0.0070037841796875
***             sample [3, 3]           ***
LZUTDCLDDR time: 0.14090228080749512
***             sample [3, 4]           ***
ULLLCLYLLDDRURRUTDRRURX time: 71.33216977119446
***             sample [3, 5]           ***
XURURRDTUZXLULDRYDRUCZYLDRD time: 138.64625692367554
***             sample [3, 6]           ***
no solution time: 300
***             sample [4, 2]           ***
RDLURDDLURDDYDDD time: 0.3549971580505371
***             sample [4, 3]           ***
LUUZDDDYCZLUURDLLZD time: 0.5550060272216797
***             sample [4, 4]           ***
XULUUXULLDYZDDDTRUUULDDDRRR time: 388.09705781936646
***             sample [4, 5]           ***
no solution time: 300
***             sample [4, 6]           ***
no solution time: 300
***             sample [5, 2]           ***
DRULDDDRDYDLDRUULDDDRD time: 3.0356969833374023
***             sample [5, 3]           ***
no solution time: 300
***             sample [5, 4]           ***
no solution time: 300
```

- We capped the scale at **6 columns MAX** and the search time to 5 minutes/300 seconds to find a solution.
- We decided to increase the column first to analyze the time complexity of a 2 x N with 1 < N <= 6 puzzle
  - The result, any puzzle with 2 x N with 1 < N < 6, a solution was found with less than 10.06 seconds' search.
  - However, once the puzzle is 2 x 6, no solution was found as we cap the time at 5 min or 300 sec.
- As follow, we increased the row by 1 and reset the column to 2. Therefore, a puzzle of 3 x N with 1 < N <= 6
  - The result was like the previous 2 x N puzzle but with a higher time to find a solution.
  - The time to find a solution was increasing exponentially at 3 x 4 -> 71.332 sec, 3 x 5 -> 138.64 sec and 3 x 6 -> No solution found.
- As we increase the number of rows and the number of columns, the time to find a solution was significantly higher.
  - No solution was found after 4 x 5
  - No solution was found after 5 x 3
    - However, we did find a solution for a 3 x 5 puzzle in which took 138.64 sec.
    - This could mean that the puzzle might be easier for the 3 x 5 than the one with a 5 x 3.
    - If we increase the time, the algorithms could have found a solution for a 5 x 3 puzzle.
  - Anything beyond either 2 x N or N x 2, is insanity because it takes a very long time.

Conclusion

- Uniform cost search is fast, but it does not count the Heuristic factor.
- Greedy best first search found a solution way quicker than the UCS but with a significant higher cost.
- Algorithms A* is better than GBFS but it is slower than UCS.
    - The reason behind this is UCS does not take the H(n) into account. Only the G(n)