CS 4346 Project Report #2
8-Puzzle Game using A* Algorithm

Sarah Mueller

Team Members :
Jordan Finney
Mason Greenwell

The Problem Description:

In this project we are creating a 8-puzzle game using an A* algorithm to determine the best path to solve the puzzle. We are creating 6 heuristic functions to help solve for the algorithm and analyze the performance of the different heuristic functions. We will calculate execution time, number of nodes generated, number of nodes expanded, depth of the tree, effective branching factor and total path. We are given the initial nodes and must find the best path and optimal heuristic function to use to get there using the functions we calculate.

The Domain:

The domain of the system is to be used in a computer game that would solve different puzzles using the best heuristic function to solve the easiest and most efficient way to use the A* algorithm. It could also be used in maps or other examples that could be used in real life.

Methodologies:

We followed the A* algorithm given to us to create a system that solves a puzzle to move the empty space until the goal node is found. We use different heuristic functions to help solve the algorithm. A huge component in the algorithm is the h prime created from the heuristic function which is how far the current node is from the goal node and finding f prime by adding the h prime to g (which is the depth of the tree). The system will go through the game and produce the different functions to find out which heuristic function is the optimal function to use. We will solve for the execution time which is simple to implement in C++. We will also find the total node generated which will simply be calculated through a counter of all the nodes kept in open or the nodes not taken and the nodes in closed which end up being the path taken. We will then look for the expanded nodes which are all of the children nodes generated. And then find the depth of the tree which is the path from the start node to the end node, or the value of g. To solve for the effective branching factor b* we will take the number of nodes generated and raise it to one over the depth of the tree.

Program Implementation:

The system was created in C++ and created in a way that is very user friendly and has a rather simple design to follow. We created different functions for each part of the code from the heuristic functions to the different calculations we needed to write. This allows for the code to be rather easy to follow.

Source Code:
// TEAM MEMBERS:

```
// Jordan Finney, Mason Greenwell, Sarah Mueller

// Project 2: Implement A* algorithm
// tasks ----
// 1. create an 8 sqaure puzzle game
// 2. Each team member must create his/her own heuristic function
// 3. Analyze the performance of the algorithm by running your program for three heuristic
functions discussed in the class and the heuristic functions created by the team
// 4. print the following...
//     * execution time (ET)
//     * number of nodes generated (NG)
//     * number of nodes expanded (NE)
//     * depth of the tree (D)
//     * effective branching factor b* (NG/D)
//     * Total path (TP)

// **note: two inital state are already given
// the heuristic function outline is already given
// run the program 12 times
// 6 functions + 2 tables each
// outline for the inital state tables are given
// input user input to run and call functions one at a time
#include <iostream>
#include <queue>
#include <cstdlib>
#include <vector>
#include <iomanip>
//#include <bits/stdc++.h>
#include <algorithm>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <chrono>
#include "Files.h"

using namespace std;
using namespace std::chrono;

void Up(vector<vector<vector<int>>> &NodeStorage, int x, int i, int j, int &moved);
void Down(vector<vector<vector<int>>> &NodeStorage, int x, int i, int j, int &moved);
void Left(vector<vector<vector<int>>> &NodeStorage, int x, int i, int j, int &moved);
void Right(vector<vector<vector<int>>> &NodeStorage, int x, int i, int j, int &moved);
int StoreNode(vector<vector<vector<int>>> &NodeStorage, int &x, int i, int j, int &moved,
int &arrPos);
int NodeChildren(vector<vector<vector<int>>> &NodeStorage,
vector<vector<vector<int>>> &Successor, int &x, int i, int j, int &moved, int &arrPos);
int NumOfChildren(vector<vector<vector<int>>> &NodeStorage, int &x, int i, int j);

//int HeuristicMason(vector<vector<int>> goalNode, vector<vector<vector<int>>>
&NodeStorage, int &x, int i, int j);

typedef struct node node;
struct node
{
  vector<node*> children;
  int data;
};

int main()
{

  priority_queue<int> OpenSorted; //open has to be a priority queue
  int moved;
  int arrPos = 1;
  int equalsGoal = 0;
  int x = 0; //THIS MUST STAY DECLARED SEPERATELY FROM i AND j
OTHERWISE WE GET A SEGFAULT?????????
  int i, j = 0;
  int choice;
  int g_value = 0;
  int h_prime;
  int f_prime;
  bool goalReached = 0;


    vector<vector<int>> goalNode{
    {1, 2, 3}, //i = 0
    {8, 0, 4}, //i = 1
    {7, 6, 5}  //i = 2
  //j= 0  1  2
};
   vector<vector<int>> initialOne{
    {2, 8, 3}, //i = 0
    {1, 6, 4}, //i = 1
    {0, 7, 5}  //i = 2
  //j= 0  1  2
};
  //priority_queue<int, vector<int>, greater<int> > OpenSet;
  vector<vector<vector<int>>> ClosedSet;
  vector<vector<vector<int>>> OpenSet(arrPos+1,vector<vector<int>>(3, vector<int>(3)));
  vector<vector<vector<int>>> Successor(arrPos+1,vector<vector<int>>(3,
vector<int>(3,0)));//[arrPos+1][3][3]
  vector<vector<vector<int>>> NodeStorage(arrPos,vector<vector<int>>(3,
vector<int>(3)));
  vector<vector<int>> CurrentNode(vector<vector<int>>(3, vector<int>(3)));
  vector<vector<int>> hPairing;

  //--------------------------------------------------------
  // starting A*
   do{
    cout << "Please pick a function to run (numbers 1-6)" << endl;
    cin >> choice;
  }while(choice > 7 || choice < 0 );


  cout << endl << endl;

//cout << "Heuristic Val : " << HeuristicThree(goalNode, NodeStorage,x) << endl;
  auto start = high_resolution_clock::now();
  for(int i = 0; i < 3; i++)
    {
       for(int j = 0; j < 3; j++)
       {
         NodeStorage[x][i][j] = initialOne[i][j]; //Stores first node in NodeStorage
         OpenSet[x][i][j] = initialOne[i][j]; //Adds the initial node to the OpenSet
         CurrentNode[i][j] = NodeStorage[x][i][j]; //Sets the initial node as the current node
         }
    }
  while(!OpenSet.empty())
```

```
 {
  if(choice == 1){
   h_prime = HeuristicOne(NodeStorage, goalNode, x);
  }
  else if(choice == 2){
   h_prime = HeuristicTwo(goalNode, NodeStorage, x, i, j);
  }
  else if(choice == 3){
   h_prime = HeuristicThree(goalNode, NodeStorage, x);
  }
  else if(choice == 4){
   h_prime = HeuristicJordan(goalNode, NodeStorage, x, i, j);
  }
  else if(choice == 5){
   h_prime = HeuristicMason(goalNode, NodeStorage, x, i, j);
  }
  else if(choice == 6){
   h_prime = HeuristicSarah(NodeStorage,goalNode, x);
  }
  f_prime = h_prime + g_value;
  // find lowest f_value;
  // this causes seg fault
  StoreNode(NodeStorage, x, i, j, moved, arrPos);

 for(i = 0; i < 3; i++){
  for(j = 0; j < 3; j++){
   if (CurrentNode[i][j] == goalNode[i][j]){ //Segfault happens here when testing
HeuristicOne for the first two children
    equalsGoal++;
   }else{
    equalsGoal = 0;
    moved = NumOfChildren(NodeStorage, x, i, j);
    NodeChildren(NodeStorage, Successor, x, i, j, moved, arrPos);
    //NodeChildren sucessfully exist within (Successor[x][i][j]) at this point.
   }
  }
  }
  int exists[x];
  // seg fault in here ****
  // for(x = 0; x < NumOfChildren(NodeStorage, x, i, j); x++){
  //        for(int i = 0; i < 3; i++){
  //          for(int j = 0; j < 3; j++){
  //              if(Successor[x][i][j] == OpenSet[x][i][j]){
  //                exists[x] += 1;
  //              }else{
  //                exists[x] = 0;
  //              }
  //           }
  //          cout << endl;
  //          }
  //        if(exists[x] < 8)
  //          exists[x] = 0;
  //      }

  int NumberOfChildren = NumOfChildren(NodeStorage, x, i, j);
 for(int x = 0; x < NumberOfChildren; x++){
  for(int i = 0; i < 3; i++){
    for(int j = 0; j < 3; j++){
        if(exists[x] == 8){
            ClosedSet[x][i][j] = Successor[x][i][j];
          }
        else if(exists[x] == 0){
            OpenSet[x][i][j] = Successor[x][i][j];
          }
        else{
          auto stop = high_resolution_clock::now();
          auto duration = duration_cast<microseconds>(stop - start);
          cout << "Time taken by function: " << duration.count() << " ms" << endl;


 cout << "Initial State #1: " << endl;
 cout << "_______________________________________________________" << endl;
 cout << "|Heuristic Function" << setw(4) << "ET" << setw(5) << "NG" << setw(5) <<
"NE" << setw(5) << "D" << setw(5) << "b*" << setw(5) << "TP" << "|" << endl;
 cout << "|___________________________________________________|" << endl;

 cout << "|6 entries" << setw(11) << duration.count() << "ms" << setw(5) << "NG" <<
setw(5) << "NE" << setw(5) << "D" << setw(5) << "b*" << setw(5) << "TP" << "|" <<
endl;
```

```
 cout << "|_____________________________________________|" << endl;
 cout << endl;
 cout << endl;
  cout << "Initial State #2: " << endl;
 cout << "_______________________________________________" << endl;
 cout << "|Heuristic Function" << setw(4) << "ET" << setw(5) << "NG" << setw(5) <<
"NE" << setw(5) << "D" << setw(5) << "b*" << setw(5) << "TP" << "|" << endl;
 cout << "|_____________________________________________|" << endl;
  cout << "|6 entries" << setw(11) << duration.count() << "ms" << setw(5) << "NG" <<
setw(5) << "NE" << setw(5) << "D" << setw(5) << "b*" << setw(5) << "TP" << "|" <<
endl;
 cout << "|_____________________________________________|" << endl;


      return 0;
      //break;
     }
    }
   } cout << endl;
  }
  int bestHValue = 0;
  int xOfBestH;
  for(int x = 0; x < NumberOfChildren; x++){
    for(int i = 0; i < 3; i++){
     for(int j = 0; j < 3; j++){
      if(HeuristicOne(Successor, goalNode, x) > bestHValue){
      bestHValue = HeuristicOne(Successor, goalNode, x);
      xOfBestH = x;
      }
     }
    }
   }
  for(int i = 0; i < 3; i++){
   for(int j = 0; j < 3; j++){
    CurrentNode[i][j] = Successor[xOfBestH][i][j];
   }
  }
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
cout << "Time taken by function: " << duration.count() << " ms" << endl;
```

```
  cout << "Initial State #1: " << endl;
  cout << "_______________________________________________" << endl;
  cout << "|Heuristic Function" << setw(4) << "ET" << setw(5) << "NG" << setw(5) <<
"NE" << setw(5) << "D" << setw(5) << "b*" << setw(5) << "TP" << "|" << endl;
  cout << "|_____________________________________________|" << endl;

  cout << "|6 entries" << setw(13) << duration.count() << "ms" << setw(5) << "NG" <<
setw(5) << "NE" << setw(5) << "D" << setw(5) << "b*" << setw(5) << "TP" << "|" <<
endl;
  cout << "|_____________________________________________|" << endl;
  cout << endl;
  cout << endl;
    cout << "Initial State #2: " << endl;
  cout << "_______________________________________________" << endl;
  cout << "|Heuristic Function" << setw(4) << "ET" << setw(5) << "NG" << setw(5) <<
"NE" << setw(5) << "D" << setw(5) << "b*" << setw(5) << "TP" << "|" << endl;
  cout << "|_____________________________________________|" << endl;
    cout << "|6 entries" << setw(13) << "ET" << setw(5) << "NG" << setw(5) << "NE" <<
setw(5) << "D" << setw(5) << "b*" << setw(5) << "TP" << "|" << endl;
  cout << "|_____________________________________________|" << endl;

    if(equalsGoal == 8){
      cout << "The goal state has been reached!" << endl;
      goalReached = 1;
    }

  }
  cout << "FAILURE" << endl;
  return 0;
}

//Code to move 0 around in the node
    void Up(vector<vector<vector<int>>>&NodeStorage, int x, int i, int j, int &moved){
    swap(NodeStorage[x][i][j], NodeStorage[x][i-1][j]); //Swaps two values.
      //The values above are 0, and one digit up from 0 respectively
}
    void Down(vector<vector<vector<int>>>&NodeStorage, int x, int i, int j, int &moved){
      swap(NodeStorage[x][i][j], NodeStorage[x][i+1][j]);
      //The values above are 0, and one digit down from 0, respectively
```

```
 }
   void Left(vector<vector<vector<int>>>&NodeStorage, int x, int i, int j, int &moved){
     swap(NodeStorage[x][i][j], NodeStorage[x][i][j-1]);
     //The values above are 0, and one digit to the left of 0, respectively
 }
   void Right(vector<vector<vector<int>>>&NodeStorage, int x, int i, int j, int &moved){
     swap(NodeStorage[x][i][j], NodeStorage[x][i][j+1]);
     //The values above are 0, and one digit to the right of 0, respectively
 }

  int StoreNode(vector<vector<vector<int>>> &NodeStorage, int &x, int i, int j, int
&moved, int &arrPos){
   NodeStorage.resize(arrPos+1,vector<vector<int>>(3, vector<int>(3))); //resizes
NodeStorage to accomodate a new node
   x+=1; //increments x so that the code now looks at the empty vector to store the next
node
   arrPos+=1; //For some reason the .resize above that's supposed to resize NodeStorage
doesn't add a new Node space, so this does that

   for(int i = 0; i < 3; i++)
     {
     for(int j = 0; j < 3; j++)
       {
        //cout << NodeStorage[x][i][j];
        //cout << NodeStorage[x-1][i][j] << "THIS IS NODE RN" << endl;
        NodeStorage[x][i][j] = NodeStorage[x-1][i][j];  //Stores the node in position x-1
(previous node) so that we can create children nodes
       }
     }
   return 0;
}
  int NodeChildren(vector<vector<vector<int>>> &NodeStorage,
vector<vector<vector<int>>> &Successor,int &x, int i, int j, int &moved, int &arrPos){
   //Below code is used for generating all possible children based on 0's location
   int y = 0;
int stop = 0;
   for(int i = 0; i < 3; i++)
     {
     for(int j = 0; j < 3; j++)
       {
```

```
if(NodeStorage[x][i][j] == 0){ //This makes sure the program is currently looking at 0
   if(i > 0){ //move up
     if(stop < moved){
         for(int i = 0; i < 3; i++)
           {
             for(int j = 0; j < 3; j++)
             {
               Successor[y][i][j] = NodeStorage[x][i][j];
             }
           }

     Up(Successor, y, i, j, moved);
       stop++;
       y++;
       }
   }
   if(i < 2){ //move down
     if(stop < moved){
         for(int i = 0; i < 3; i++)
           {
             for(int j = 0; j < 3; j++)
             {
               Successor[y][i][j] = NodeStorage[x][i][j];
             }
           }
     Down(Successor, y, i, j, moved);
       stop++;
       y++;
       }
   }
   if(j < 2){ //move right
     if(stop < moved){
         for(int i = 0; i < 3; i++)
           {
             for(int j = 0; j < 3; j++)
             {
               Successor[y][i][j] = NodeStorage[x][i][j];
             }
           }
     Right(Successor, y, i, j, moved);
```

```
              stop++;
              y++;
              }
          }
          if(j > 0){ //move left
            if(stop < moved){
                for(int i = 0; i < 3; i++)
                  {
                    for(int j = 0; j < 3; j++)
                    {
                      Successor[y][i][j] = NodeStorage[x][i][j];
                    }
                  }
            Left(Successor, y, i, j, moved);
              stop++;
              y++;
              }
          }
        }
    }
    }
  return 0;
  }

int NumOfChildren(vector<vector<vector<int>>> &NodeStorage, int &x, int i, int j){
    int childrenCount = 0;
      if(i > 0){ //move up
        childrenCount+=1;
      }
      if(i < 2){ //move down
        childrenCount+=1;
      }
      if(j < 2){ //move right
        childrenCount+=1;
      }
      if(j > 0){ //move left
        childrenCount+=1;
      }
  return childrenCount;
};
```

```
/*
   {2, 8, 3}, //i = 0
   {1, 6, 4}, //i = 1
   {0, 7, 5}  //i = 2
 //j= 0  1  2
*/
Files.h
#ifndef IDK_WHAT_THIS_DOES
#define IDK_WHAT_THIS_DOES

#include <vector>
#include <iostream>

using namespace std;

int HeuristicSarah(vector<vector<vector<int>>> &NodeStorage, vector<vector<int>>
&goalNode, int &x);
int HeuristicMason(vector<vector<int>> &goalNode, vector<vector<vector<int>>>
&NodeStorage, int &x, int i, int j);
int HeuristicJordan(vector<vector<int>> &goalNode, vector<vector<vector<int>>>
&NodeStorage, int &x, int i, int j);
int HeuristicOne(vector<vector<vector<int>>> &NodeStorage, vector<vector<int>>
&goalNode, int &x);
int HeuristicTwo(vector<vector<int>> &goalNode, vector<vector<vector<int>>>
&NodeStorage, int &x, int i, int j);
int HeuristicThree(vector<vector<int>> &goalNode, vector<vector<vector<int>>>
&NodeStorage, int &x);
#endif

HeuristicJordan.cpp
#include "Files.h"
#include <vector>
#include <iostream>


using namespace std;
/*
How many tiles are in the correct corner spots ...
```

```
    vector<vector<int>> goalNode{
    {1, 2, 3}, //i = 0
    {8, 0, 4}, //i = 1
    {7, 6, 5}  //i = 2
  //j= 0  1  2
};
   vector<vector<int>> initialOne{
    {2, 8, 3}, //i = 0
    {1, 6, 4}, //i = 1
    {0, 7, 5}  //i = 2
  //j= 0  1  2
};
    vector<vector<int>> initialTwo{
    {2, 1, 6}, //i = 0
    {4, 0, 8}, //i = 1
    {7, 5, 3}  //i = 2
  //j= 0  1  2
};
*/


int HeuristicJordan(vector<vector<int>> &goalNode, vector<vector<vector<int>>>
&NodeStorage, int &x, int i, int j){
  int hPrime;
  if (NodeStorage[x][0][0] == goalNode[0][0])
  {
    hPrime += 1;
  }
  if(NodeStorage[x][0][2] == goalNode[0][2])
  {
    hPrime += 1;
  if(NodeStorage[x][2][0] == goalNode[2][0])
  {
    hPrime += 1;
  }
  if(NodeStorage[x][2][2] == goalNode[2][2])
  {
     hPrime += 1;
  }
  }
  return hPrime;
}
HeuristicMason.cpp
/*
Amount of numbers adjacent to their correct position

complete ex.
1 2 3
8 _ 4
7 6 5

ex.
3 1 2
4 8 _
7 6 5


h(n) =  4

*/
/*
How many tiles are in the correct corner spots ...
    vector<vector<int>> goalNode{
    {1, 2, 3}, //i = 0
    {8, 0, 4}, //i = 1
    {7, 6, 5}  //i = 2
  //j= 0  1  2
};
   vector<vector<int>> initialOne{
    {2, 8, 3}, //i = 0
    {1, 6, 4}, //i = 1
    {0, 7, 5}  //i = 2
  //j= 0  1  2
};
    vector<vector<int>> initialTwo{
    {2, 1, 6}, //i = 0
    {4, 0, 8}, //i = 1
    {7, 5, 3}  //i = 2
  //j= 0  1  2
};
```

```
*/
#include "Files.h"
#include <vector>
#include <iostream>

using namespace std;

int HeuristicMason(vector<vector<int>> &goalNode, vector<vector<vector<int>>>
&NodeStorage, int &x, int i, int j){
  int hPrime = 0;
      if (NodeStorage[x][1][0] == goalNode[1][0])
  {
    hPrime += 1;
  }
  if(NodeStorage[x][1][2] == goalNode[1][2])
  {
    hPrime += 1;
  if(NodeStorage[x][2][1] == goalNode[2][1])
  {
    hPrime += 1;
  }
  if(NodeStorage[x][1][0] == goalNode[1][0])
  {
     hPrime += 1;
  }
  }
  return hPrime;
}
HeuristicOne.cpp
#include "Files.h"
#include <vector>
#include <iostream>


using namespace std;
/*
    vector<vector<int>> goalNode{
    {1, 2, 3}, //i = 0
    {8, 0, 4}, //i = 1
    {7, 6, 5}  //i = 2
  //j= 0  1  2
};
   vector<vector<int>> initialOne{
    {2, 8, 3}, //i = 0
    {1, 6, 4}, //i = 1
    {0, 7, 5}  //i = 2
  //j= 0  1  2
};
    vector<vector<int>> initialTwo{
    {2, 1, 6}, //i = 0
    {4, 0, 8}, //i = 1
    {7, 5, 3}  //i = 2
  //j= 0  1  2
};
The simplest heuristic counts the tiles out of place in
each state when it is compared with the goal.
*/


int HeuristicOne(vector<vector<vector<int>>> &NodeStorage, vector<vector<int>>
&goalNode, int &x){
  int hPrime = 0; //Maybe make hPrime an array, so we can correlate the hPrime value to a
node position through the x value. For example, hPrimeArr[1] would hold the hPrime
value of the node stored in OpenNode[1][i][j]
  for(int i = 0; i < 3; i++){
    for(int j = 0; j < 3; j++){
      if(NodeStorage[x][i][j] != goalNode[i][j]){
        if(NodeStorage[x][i][j] != 0)
        hPrime++;
      }
    }
  }

  return hPrime;
}
HeuristicSarah.cpp
// TEAM MEMBERS:
// Jordan Finney, Mason Greenwell, Sarah Mueller

// Project 2: Implement A* algorithm
```

```
/*
how many tiles are in the correct spot (subtract 1)
and how many are in the wrong (add 1)
this will show how far it is from the goal (where all 8 tiles are in the correct spot)
Goal state:
1 2 3
8 _ 4          h(n) = -8
7 6 5

Initial state 1:
2 8 3
1 6 4          h(n) = 3
_ 7 5

Initial state 2:
2 1 6
4 _ 8          h(n) = 8
7 5 3
*/
#include "Files.h"
#include <iostream>
#include <queue>
#include <cstdlib>
#include <vector>

using namespace std;

int HeuristicSarah(vector<vector<vector<int>>> &NodeStorage, vector<vector<int>>
&goalNode, int &x){
  int hPrime = 0;
  for(int i = 0; i < 3; i++){
    for(int j = 0; j < 3; j++){
      if(NodeStorage[x][i][j] == goalNode[i][j]){
        hPrime--;
      }
      else{
        hPrime++;
      }
    }
  }
  return hPrime;
}
HeuristicThree.cpp
#include "Files.h"
#include <vector>
#include <iostream>


using namespace std;
/*
    vector<vector<int>> goalNode{
    {1, 2, 3}, //i = 0
    {8, 0, 4}, //i = 1
    {7, 6, 5}  //i = 2
  //j= 0  1  2
};
   vector<vector<int>> initialOne{
    {2, 8, 3}, //i = 0
    {1, 6, 4}, //i = 1
    {0, 7, 5}  //i = 2
  //j= 0  1  2
};
    vector<vector<int>> initialTwo{
    {2, 1, 6}, //i = 0
    {4, 0, 8}, //i = 1
    {7, 5, 3}  //i = 2
  //j= 0  1  2
};
2x the number of tile reversals
H3: The number of direct tile reversals multiplied by 2 (a direct tile reversal is when two
adjacent tiles must be swapped to match the goal order)
*/


int HeuristicThree(vector<vector<int>> &goalNode, vector<vector<vector<int>>>
&NodeStorage, int &x){
 int hPrime = 0;
  int val;
  for(int i = 0; i < 3; i++){
    for(int j = 0; j < 3; j++){
      for(int m = 0; m < 3; m++){
        for(int n = 0; n < 3; n++){
          if(NodeStorage[x][i][j] == goalNode[m][n]){
            swap(NodeStorage[x][i][j], NodeStorage[x][m][n]);
            hPrime++;
          }
        }
      }
    }
  }

  return (2 * hPrime);
}
HeuristicTwo.cpp
#include "Files.h"
#include <vector>
#include <iostream>
#include <cmath>

using namespace std;
/*
    vector<vector<int>> goalNode{
    {1, 2, 3}, //i = 0
    {8, 0, 4}, //i = 1
    {7, 6, 5}  //i = 2
  //j= 0  1  2
};
   vector<vector<int>> initialOne{
    {2, 8, 3}, //i = 0
    {1, 6, 4}, //i = 1
    {0, 7, 5}  //i = 2
  //j= 0  1  2
};
    vector<vector<int>> initialTwo{
    {2, 1, 6}, //i = 0
    {4, 0, 8}, //i = 1
    {7, 5, 3}  //i = 2
  //j= 0  1  2
};
```

sum all the distances by which the tiles are out of place, one for each square a tile must be moved to reach its position in the goal state.
*/

```
int HeuristicTwo(vector<vector<int>> &goalNode, vector<vector<vector<int>>>
&NodeStorage, int &x, int i, int j){
  int hPrime = 0;

  for(int i = 0; i < 3; i++){
    for(int j = 0; j < 3; j++){
      for(int m = 0; m < 3; m++){
        for(int n = 0; n < 3; n++){
          if(NodeStorage[x][i][j] == goalNode[m][n]){
            if(NodeStorage[x][i][j] != 0){
            hPrime += (abs(m - i) + abs(n - j));
            }
          }
        }
      }
    }
  }
  return hPrime;
}
```

Readme.md
Heuristic Functions and A* Algorithm Analysis
Artifical Intelligence Project #2

TEAM MEMBERS
- Jordan Finney
- Sarah Mueller
- Mason Greenwell

CODE INSTRUCTIONS

Step 1. Download the following files
- HEADER_FILE_H.h
- Heuristic1.cpp
- Heuristic2.cpp
- Heuristic3.cpp

- HeuristicJordan.cpp
- HeuristicMason.cpp
- HeuristicSarah.cpp
- main.cpp

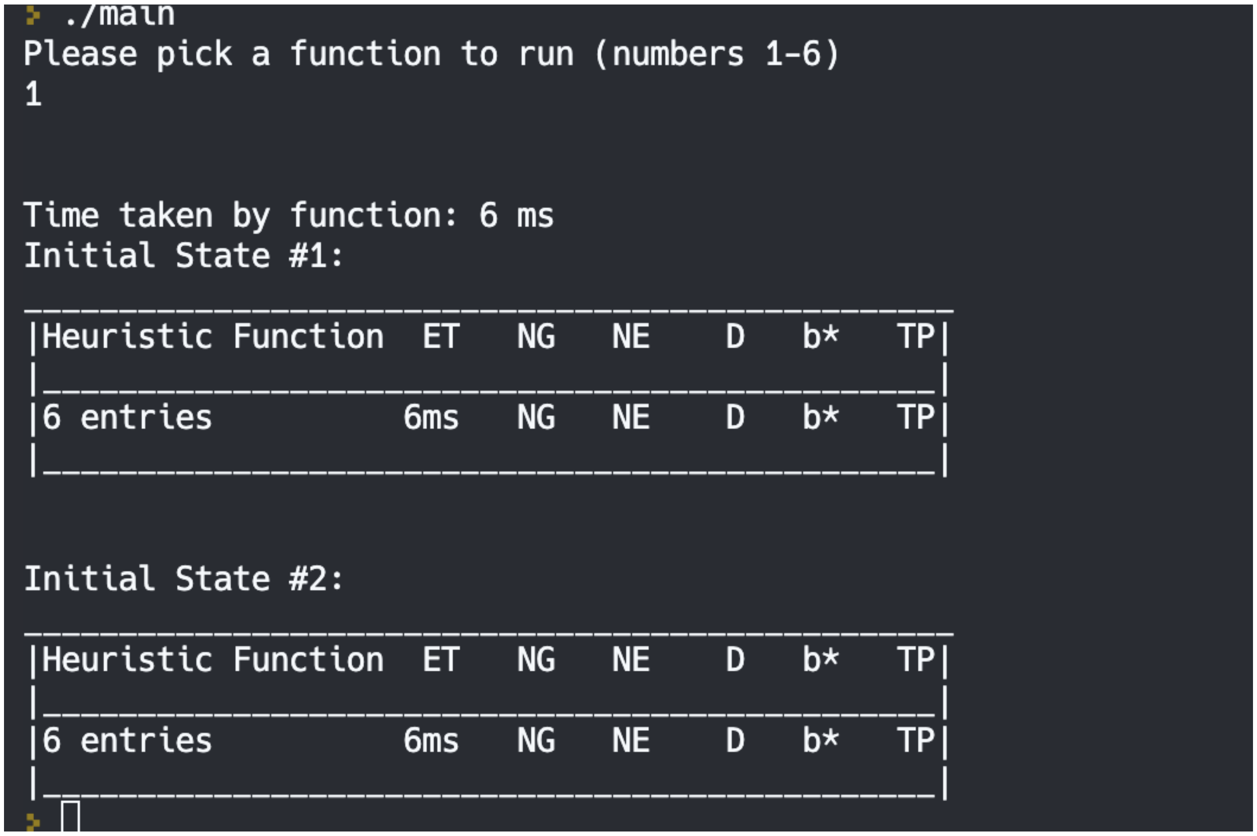
Step 2. Once the program is downloaded and put into a compiler, run the program.

Step 3. Type in a number 1-6 to run a heuristic function with the A* algorithm

Step 4. Look at the tables displayed and compare the measurements

ABOUT THE PROGRAM:
This is a program developed by my team and I to anaylize the way heuristic functions shape the outcome of A* algorithm in a AI enviornment.

A Copy of the Program Run:



```
> ./main
Please pick a function to run (numbers 1-6)
1


Time taken by function: 6 ms
Initial State #1:
_________________________________________________
|Heuristic Function  ET   NG   NE    D   b*   TP|
|_______________________________________________|
|6 entries          6ms   NG   NE    D   b*   TP|
|_______________________________________________|


Initial State #2:
_________________________________________________
|Heuristic Function  ET   NG   NE    D   b*   TP|
|_______________________________________________|
|6 entries          6ms   NG   NE    D   b*   TP|
|_______________________________________________|
>
```

Description of the Heuristic Function:

We created 6 heuristic functions in total, three were from the slides in class and three were created by each individual team member. Each heuristic function was put in its own .cpp file and we created a header file to tie all the function files to the main. We decided to separate all of the functions into different files to help create some organization and be easy to read. Heuristic one is simply counting the number of tiles out of place when compared to the goal node. Heuristic two is to find the sum of the distances from the tile location to the goal location. Heuristic three is to find the number of direct tile reversals multiplied by two. My heuristic function is similar to heuristic one, it calculates how many tiles are out of place and how many tiles are in place and for everytime out of place it adds one to the h prime value and everytime there is a value in place it subtracts one from the h prime. Jordan's heuristic function looks at just the corner values to see if they are correct and compares the different values in the corners to that in the goal node. Mason's heuristic function checks the sides of the nodes to see which ones are correct when comparing the initial and all moved nodes to the goal node.

Special Approaches:

We had to become creative when implementing the algorithm and storing the nodes. We took our time to understand the algorithm and took the algorithm and turned it into code the best way we could. We did what we thought was easiest to implement the nodes by creating vectors for the different nodes.

Analysis of the Program:

Our program was created in a way that is easy for users to understand and change. We implemented each function in different files so that you can easily change the functions or add different things or even expand on the code. A* would be implemented through the main and call each function separately allowing the main part of the code to work and allow changes to happen seamlessly.

Analysis of the Results:

It was interesting to see how the different heuristic functions changed the different values we calculated. How not all functions are optimal and some find a worse path than others. We loved the process of the system but also loved seeing the results as we got to create something that could function almost completely by itself. It was a very cool system to implement and to see the results we were able to display.

Conclusion:

It was a great experience learning how to implement different functions and algorithms into code that we normally wouldn't get a chance to do in other classes we have. I learned a better understanding of the A* algorithm and saw different ways it could apply to other scenarios than just the diagrams we see in class. I learned how to effectively be a team member and contribute my share of the work to allow others to be able to contribute as well. I feel like I learned a lot in this project that I normally wouldn't get the chance to in a test or in other classes.

Team Members Contribution:

Sarah Mueller: I was in charge of doing the research on the A* algorithm and finding ways to implement it. I was the one that seemed to understand how the algorithm worked the best out of the group so it was good to be able to expand on my knowledge of the algorithm and help the team the best way I could.

Jordan Finney: Jordan was the team leader and helped set up meetings. She was in charge of finding times during the week we could meet and allowing the project to run smoothly, she also was in charge of making sure we stayed on task throughout the meetings. We met almost every Tuesday in the evenings and found times on various weekends to meet.

Mason Greenwell: Mason was in charge of creating and implementing the calculations in the system. He helped with the other heuristic functions too.

As a team: We all did our individual heuristic functions and worked together on the three from class. We all helped code and made sure we understood what we were trying to accomplish each meeting. Every member pulled their part of the project and helped make this project a fun thing to do.

References:

https://www.google.com/search?q=when+do+people+use+a+*+algorithm&oq=when+do+people+use+a+*+algorithm+&aqs=chrome..69i57.9906j0j7&sourceid=chrome&ie=UTF-8

https://www.javatpoint.com/priority-queue-in-cpp#:~:text=The%20priority%20queue%20in%20C%2B%2B,priority%20element%20is%20popped%20first.

https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/

https://www.geeksforgeeks.org/sum-manhattan-distances-pairs-points/