Artificial Intelligence Project 2 Report

By: Tyler Gates
Group members: Marcos Izaguirre& Daisey Jaramillo

Problem

The goal of this project was to create and implement the A* algorithm to solve the eight puzzle the quickest way possible. We also test five different heuristic functions for the A* algorithm to use, as well as two different initial states.

Methodology

Our method for solving this problem and completing the project was creating an AStar class that stores the structure of the algorithm, and another class for the structure of the nodes. Inside the node class is a parent pointer, a struct called children that stores the potential four different children of each position of the 8 puzzle as pointers. It has getter and setters for the F value, G Value, H value, the puzzle position. It also has functions to generate the parent nodes and children nodes, and calculate the H and G values. There is also a 2D integer array to store the puzzle position of each node.

The AStar class stores a pointer to the 2D int of the goal state, and the starting node, ‘root’. It also stores and calculates all the needed variables for the table (ET, NG, NE, d, b*, and MO). Inside of the AStar class is also the priority queue for the open/close list. A function for the algorithm titled algorithm that runs the entire program until the goal state is reached, A constructor for the class, a stack of Nodes that stores the best path. A function to check if the goal state has been reached. And lastly a function to process the children or successors of the current node.

Inside of main we first define the first two initial states, and the goal state. Then we run all the algorithms by initializing separate AStar classes for each initial state and each heuristic function and print the first one with intermediary steps to show how it works, then print out the two tables for the two different initial states. We also put the five separate functions for the heuristic values, H1, H2, H, H4, and H5 inside of main.

Group Participation

Marcos and I both wrote our own versions of AStar to perfect it in a friendly competitive manner, helping improve the end result. Daisey wrote the heuristic functions, and I later transferred all of her work to work with the finished AStar. I also created the fifth heuristic function that calculates the euclidean distance.

Source Code Implementation

Node.h

```
#ifndef NODE_H_INCLUDED
#define NODE_H_INCLUDED
```

```
#include <iostream>
#include<list>
#include<vector>
#include<deque>
using namespace std;

class Node{

public:
    Node * parent = nullptr;
    float f_value;
    float g_value;
    float h_value;

    struct Children{
    Node * left;
    Node * right;
    Node * up;
    Node * down;
    } Children_default = {nullptr,nullptr,nullptr,nullptr};
    typedef struct Children Children;

    Children children = Children_default;
    int number_of_childs;
    static const int ROW = 3, COL = 3;
                        //   c0 c1 c2
    int puzzle[ROW][COL];// r0 xx xx xx
                       //  r1 xx xx xx
                      //   r2 xx xx xx

    int blank[2] = {0,0};
    Node(int val[3][3]);
    Node();
    Node(Node *p);

    float getFVal();
    void setFval(float val);

    float getGVal();
    void setGval(float val);

    float getHVal();
    void setHval(float val);

    int** getPuzzle();
    void setPuzzle(int val[][COL]);

    int getNumOfChilds();
    void addToNumChilds();

    Node* getParent();
    void setParent(Node* val);
```

```
    Children getchildren();
    void generateChildren();

    void calculateGvalue();
    void calculateHvalue(Node * val, int func(Node*));

    Node*  moveLeft(Node* a);
    Node* moveRight(Node* a);
    Node* moveUp(Node* a);
    Node* moveDown(Node* a);

    friend bool operator <(const Node &n1, const Node &n2){
        return n1.f_value < n2.f_value;
    }

    friend bool operator >(const Node &n1, const Node &n2){
        return n1.f_value > n2.f_value;
    }


    void print();

    ~Node();
};




#endif // NODE_H_INCLUDED
```

Node.cpp

```
#include "Node.h"
int puzzle_default[][3] = {{0,0,0}, {0,0,0}, {0,0,0}};

Node::Node(int val[3][3] = puzzle_default){
    this->setPuzzle(val);
    this->number_of_childs = 0;
}

Node::Node(Node *p){
    this->setPuzzle(p->puzzle);
    this->g_value = p->g_value;
    this->parent = p->parent;
}

float Node:: getFVal(){
    return this->f_value;
}
void Node:: setFval(float val){
```

```
    this->f_value = val;
}

float Node:: getGVal(){
    return this->g_value;
}
void Node:: setGval(float val){
    this->g_value = val;
}
float Node:: getHVal(){
    return this->h_value;
}
void Node:: setHval(float val){
    this->h_value = val;
}
int** Node:: getPuzzle(){
    int** temp;
    temp = new int*[3];
      for (int r = 0; r < 3; r++)
      {
            temp[r] = new int[3];

            for (int c = 0; c < 3; c++)
            {
                  temp[r][c] = this->puzzle[r][c];
            }
      }
    return temp;
}
void Node:: setPuzzle(int val[][COL]){
    for(int r = 0; r < ROW; r++ ){
        for(int c = 0; c < COL; c++ ){
            this->puzzle[r][c] = val[r][c];
            if(val[r][c]==0){
                this->blank[0] = r;
                this->blank[1] = c;
            }
        }
    }
}
int Node:: getNumOfChilds(){
    return this->number_of_childs;
}
void Node:: addToNumChilds(){
    this->number_of_childs++;
}
Node* Node:: getParent(){
    return this->parent;
}
void Node:: setParent(Node* val){
    Node *p = new Node(val);
    this->parent = p;
}
```

```
Node::Children Node:: getchildren(){
   return this->children;
}

void Node:: generateChildren(){
   this->children.left = moveLeft(this);
   this->children.right = moveRight(this);
   this->children.up = moveUp(this);
   this->children.down = moveDown(this);
}

void Node:: print(){
   cout << "--------" << endl;
   for(int r = 0; r < ROW; r++ ){
      for(int c = 0; c < COL; c++ ){
         cout << "|" << puzzle[r][c];
      }
      cout << "|" << endl;
   }
   cout << "--------" << endl;
}

Node* Node:: moveLeft(Node *a) {
   if(a->blank[1]==0)
      return nullptr;
   else{
      Node* temp = new Node(a->puzzle);
      //moving blank space left
      temp->puzzle[a->blank[0]][a->blank[1]]=
      temp->puzzle[a->blank[0]][a->blank[1]-1];
      temp->puzzle[a->blank[0]][a->blank[1]-1]=0;
      a->addToNumChilds();
      temp->blank[1]--;
      return temp;
   }
}

Node* Node::moveRight(Node *a) {
   if(a->blank[1]==2)
      return nullptr;
   else{
      Node* temp = new Node(a->puzzle);
      //moving blank space left
      temp->puzzle[a->blank[0]][a->blank[1]]=
      temp->puzzle[a->blank[0]][a->blank[1]+1];
      temp->puzzle[a->blank[0]][a->blank[1]+1]=0;
      a->addToNumChilds();
      temp->blank[1]++;
      return temp;
   }

}
```

```
Node* Node:: moveUp(Node *a) {
   if(a->blank[0]==0)
      return nullptr;
   else{
      Node* temp = new Node(a->puzzle);
      //moving blank space left
      temp->puzzle[a->blank[0]][a->blank[1]]=
      temp->puzzle[a->blank[0]-1][a->blank[1]];
      temp->puzzle[a->blank[0]-1][a->blank[1]]=0;
      a->addToNumChilds();
      temp->blank[0]--;
      return temp;
   }

}

Node* Node:: moveDown(Node *a) {
   if(a->blank[0]==2)
      return nullptr;
   else{
      Node* temp = new Node(a->puzzle);
      //moving blank space left
      temp->puzzle[a->blank[0]][a->blank[1]]=
      temp->puzzle[a->blank[0]+1][a->blank[1]];
      temp->puzzle[a->blank[0]+1][a->blank[1]]=0;
      a->addToNumChilds();
      temp->blank[0]++;
      return temp;
   }
}

Node::~Node() {
         if(this->children.left != nullptr){
      delete this->children.left;
      this->children.left = nullptr;
         }
         if(this->children.right != nullptr){
      delete this->children.right;
      this->children.right = nullptr;
         }
         if(this->children.up != nullptr){
      delete this->children.up;
      this->children.up = nullptr;
         }
         if(this->children.down != nullptr){
      delete this->children.down;
      this->children.down = nullptr;
         }
}
```

AStar.h

```
#ifndef ASTAR_H_INCLUDED
```

```
#define ASTAR_H_INCLUDED

#include <iostream>
#include <queue>
#include "Node.h"
#include <vector>
#include<stack>
using namespace std;

class AStar{

public:
    int **goal;
    Node *qp;
    Node *goal_state = nullptr;
    Node *root;
    int NG = 0;
    int NE = 0;
    int d = 0;
    float ebf = 0;

    priority_queue<Node, vector<Node>, greater<vector<Node>::value_type>> Open_List;
    priority_queue<Node, vector<Node>, greater<vector<Node>::value_type>> Close_List;

    AStar(int **g,int go[][3], Node * r){
        goal = g;
        root = new Node(go);
        root->f_value = 0;
        root->g_value = 0;
        root->parent = nullptr;
        Open_List.push(*root);


    }

    stack<Node*> algorithm(){
        while(!Open_List.empty()){
            NE++;
            Node q = Open_List.top();
            qp = &q;

            if(qp->g_value>d)
                d = qp->g_value;

            int ** currentpuzzle = qp->getPuzzle();
            if(checkPuzzleEqual(currentpuzzle, goal)){
                goal_state = qp;
                break;
            }
            Open_List.pop();
            Close_List.push(q);
            qp->generateChildren();
```

```
        if(qp->children.down!=nullptr)
            qp->children.down->setParent(qp);

        if(qp->children.up!=nullptr)
            qp->children.up->setParent(qp);

        if(qp->children.left!=nullptr)
            qp->children.left->setParent(qp);

        if(qp->children.right!=nullptr)
            qp->children.right->setParent(qp);

        processChildren(qp->children.down);
        processChildren(qp->children.up);
        processChildren(qp->children.left);
        processChildren(qp->children.right);


    }
stack<Node*> best_path;
while(goal_state->parent !=nullptr){
    goal_state = goal_state->parent;
    best_path.push(goal_state);
}
ebf = NG/d;
return best_path;
}

void processChildren(Node *s){
    if(s == nullptr)
        return;
    NG++;

    int ** currentpuzzle = s->getPuzzle();

    s->g_value = s->parent->g_value + 1;
    s->h_value = HeuristicFunction(s,this);

    s->f_value = s->g_value + s->h_value;
    if(checkPuzzleEqual(currentpuzzle, goal)){
        goal_state = s;
        }
    bool skip = false;
    vector<Node> open;

    while(!Open_List.empty()){
        Node it = Open_List.top();
        Open_List.pop();
        open.push_back(it);
        if(checkPuzzleEqual(currentpuzzle,it.getPuzzle())){
            if(it.getFVal() < s->getFVal()){
                skip = true;
                break;
```

```
            }
        }
    }
    for (std::vector<Node>::iterator it = open.begin() ; it != open.end(); ++it){
        Open_List.push(*it);
    }
    if(!skip){
        vector<Node> close;
        while(!Close_List.empty()){
            Node it = Close_List.top();
            Close_List.pop();
            close.push_back(it);
            if(checkPuzzleEqual(currentpuzzle,it.getPuzzle())){
                if(it.getFVal() < s->getFVal()){
                    skip = true;
                    break;
                }
            }
        }
        for (std::vector<Node>::iterator it = close.begin() ; it != close.end(); ++it){
            Close_List.push(*it);
        }
    }

    if(!skip){
        Node temp = *s;
        Open_List.push(temp);
    }
}

float (*HeuristicFunction)(Node *s, AStar * t);


bool checkPuzzleEqual(int** current_puzzle, int** check){
    for(int r = 0; r < 3; r++ ){
        for(int c = 0; c < 3; c++ ){
            if(check[r][c]!=current_puzzle[r][c])
                return false;
        }
    }
    return true;
}

};


#endif // ASTAR_H_INCLUDED
```

AStar.cpp

```
#include "AStar.h"
int puzzle_dd[][3] = {{0,0,0}, {0,0,0}, {0,0,0}};
```

```
#include <iostream>
#include "Node.h"
#include <stack>
#include "AStar.h"
#include <math.h>
using namespace std;

float h1(Node *s, AStar *t);//h1
float h2(Node* s, AStar* t);//h2
float S(Node* s, AStar* t);//S
float H(Node* s, AStar* t);//H
float h4(Node* s, AStar* t);//h4
float h5(Node* s, AStar* t);//h5
```

main.cpp

```
int main()
{
    int state_one[][3] = {{2,8,3}, {1,6,4}, {0,7,5}};
    int state_two[][3] = {{2,1,6}, {4,0,8}, {7,5,3}};
    int goal[][3] = {{1,2,3}, {8,0,4}, {7,6,5}};
    int **goal_p;
    goal_p = new int*[3];
    for (int r = 0; r < 3; r++)
    {
        goal_p[r] = new int[3];
        for (int c = 0; c < 3; c++)
        {
            goal_p[r][c] = goal[r][c];
        }
    }
    Node *root = new Node(state_one);// pass current state of game array
    AStar A = AStar(goal_p,state_one,root);//pass root, current state of game
                            //and the goal pointer
    A.HeuristicFunction = h1; //pass heuristic function
    stack<Node*> a=A.algorithm();//receive a stack of the best path


    cout<<"the best path"<<endl;
    while(!a.empty()){
        Node *nod = a.top();
        a.pop();
        nod->print(); //print path
    }

    cout<<"GOAL STATE REACHED"<<endl;//print goal
    cout << "--------" << endl;
    for(int r = 0; r < 3; r++ ){
        for(int c = 0; c < 3; c++ ){
            cout << "|" << goal[r][c];
        }
```

```
    cout << "|" << endl;
}
AStar B = AStar(goal_p,state_one,root);
B.HeuristicFunction = h2;
stack<Node*> b=B.algorithm();
cout << "test"<< endl;
AStar C = AStar(goal_p,state_one, root);
C.HeuristicFunction = H; //pass heuristic function
stack<Node*> c=C.algorithm();
cout << "test"<< endl;
AStar D = AStar(goal_p,state_one,root);
D.HeuristicFunction = h4;
stack<Node*> d=D.algorithm();
cout << "test"<< endl;
AStar E = AStar(goal_p,state_one,root);
E.HeuristicFunction = h5;
stack<Node*> e=E.algorithm();
cout << "test"<< endl;
cout << "***Initial State #1***" << endl;
cout << "heuristic/ET/NG/NE/d/b*/MO" << endl;
cout<<  "h1"<< "/x/"<<A.NG<<"/"<<A.NE<<"/"<<A.d<<"/"<<A.ebf<<"/"<<"x  " <<endl;
cout<<  "h2"<< "/x/"<<B.NG<<"/"<<B.NE<<"/"<<B.d<<"/"<<B.ebf<<"/"<<"x  " <<endl;
cout<<  "H"<< "/x/"<<C.NG<<"/"<<C.NE<<"/"<<C.d<<"/"<<C.ebf<<"/"<<"x  " <<endl;
cout<<  "h4"<< "/x/"<<D.NG<<"/"<<D.NE<<"/"<<D.d<<"/"<<D.ebf<<"/"<<"x  " <<endl;
cout<<  "h5"<< "/x/"<<E.NG<<"/"<<E.NE<<"/"<<E.d<<"/"<<E.ebf<<"/"<<"x  " <<endl<<endl;
//delete &A, &B, &C, &D, &E;

*root = new Node(state_two);
AStar G = AStar(goal_p,state_two, root);
G.HeuristicFunction = h1;
stack<Node*> g=G.algorithm();
// << "test"<< endl;
AStar HH = AStar(goal_p,state_two, root);
HH.HeuristicFunction = h2;
stack<Node*> hh=HH.algorithm();
//cout << "test"<< endl;
AStar F = AStar(goal_p,state_two, root);
F.HeuristicFunction = H;
stack<Node*> f=F.algorithm();
//cout << "test"<< endl;
AStar I = AStar(goal_p,state_two, root);
I.HeuristicFunction = h4;
stack<Node*> i=I.algorithm();
//cout << "test"<< endl;
AStar J = AStar(goal_p,state_two, root);
J.HeuristicFunction = h5;
stack<Node*> j=J.algorithm();
//cout << "test"<< endl;
cout << "***Initial State #2***" << endl;
cout << "heuristic/ET/NG/NE/d/b*/MO" << endl;
cout<<  "h1"<< "/x/"<<G.NG<<"/"<<G.NE<<"/"<<G.d<<"/"<<G.ebf<<"/"<<"x  " <<endl;
cout<<  "h2"<< "/x/"<<HH.NG<<"/"<<HH.NE<<"/"<<HH.d<<"/"<<HH.ebf<<"/"<<"x  " <<endl;
cout<<  "H"<< "/x/"<<F.NG<<"/"<<F.NE<<"/"<<F.d<<"/"<<F.ebf<<"/"<<"x  " <<endl;
```

```
    cout<<  "h4"<< "/x/"<<I.NG<<"/"<<I.NE<<"/"<<I.d<<"/"<<I.ebf<<"/"<<"x  " <<endl;
    cout<<  "h5"<< "/x/"<<J.NG<<"/"<<J.NE<<"/"<<J.d<<"/"<<J.ebf<<"/"<<"x  " <<endl;

    return 0;
}

float h1(Node *s, AStar *t){
    float sum = 0;
    int ** currentpuzzle = s->getPuzzle();
    for(int r = 0; r < 3; r++ ){
        for(int c = 0; c < 3; c++ ){
            if(t->goal[r][c]!=currentpuzzle[r][c])
                sum++;
        }
    }
    return sum;
}

float h2(Node* s, AStar* t) {
    float h2 = 0;
    int** currentpuzzle = s->getPuzzle();
    if (currentpuzzle[0][0] != 1)
    {
        if (currentpuzzle[0][0] == 2)
            h2 += 1;
        if (currentpuzzle[0][0] == 3)
            h2 += 2;
        if (currentpuzzle[0][0] == 4)
            h2 += 3;
        if (currentpuzzle[0][0] == 5)
            h2 += 4;
        if (currentpuzzle[0][0] == 6)
            h2 += 3;
        if (currentpuzzle[0][0] == 7)
            h2 += 2;
        if (currentpuzzle[0][0] == 8)
            h2 += 1;
    }
    if (currentpuzzle[0][1] != 2)
    {
        if (currentpuzzle[0][1] == 1)
            h2 += 1;
        if (currentpuzzle[0][1] == 3)
            h2 += 1;
        if (currentpuzzle[0][1] == 4)
            h2 += 2;
        if (currentpuzzle[0][1] == 5)
            h2 += 3;
        if (currentpuzzle[0][1] == 6)
            h2 += 2;
        if (currentpuzzle[0][1] == 7)
            h2 += 3;
        if (currentpuzzle[0][1] == 8)
```

```
        h2 += 2;

}
if (currentpuzzle[0][2] != 3)
{
    if (currentpuzzle[0][2] == 1)
        h2 += 2;
    if (currentpuzzle[0][2] == 2)
        h2 += 1;
    if (currentpuzzle[0][2] == 4)
        h2 += 1;
    if (currentpuzzle[0][2] == 5)
        h2 += 2;
    if (currentpuzzle[0][2] == 6)
        h2 += 3;
    if (currentpuzzle[0][2] == 7)
        h2 += 4;
    if (currentpuzzle[0][2] == 8)
        h2 += 3;
}
if (currentpuzzle[1][0] != 8)
{
    if (currentpuzzle[1][0] == 1)
        h2 += 1;
    if (currentpuzzle[1][0] == 2)
        h2 += 2;
    if (currentpuzzle[1][0] == 3)
        h2 += 3;
    if (currentpuzzle[1][0] == 4)
        h2 += 2;
    if (currentpuzzle[1][0] == 5)
        h2 += 3;
    if (currentpuzzle[1][0] == 6)
        h2 += 2;
    if (currentpuzzle[1][0] == 7)
        h2 += 1;
}
if (currentpuzzle[1][1] != 0)
{
    if (currentpuzzle[1][1] == 1)
        h2 += 2;
    if (currentpuzzle[1][1] == 2)
        h2 += 1;
    if (currentpuzzle[1][1] == 3)
        h2 += 2;
    if (currentpuzzle[1][1] == 4)
        h2 += 1;
    if (currentpuzzle[1][1] == 5)
        h2 += 2;
    if (currentpuzzle[1][1] == 6)
        h2 += 1;
    if (currentpuzzle[1][1] == 7)
        h2 += 2;
```

```
    if (currentpuzzle[1][1] == 8)
        h2 += 1;
}
if (currentpuzzle[1][2] != 4)
{
    if (currentpuzzle[1][2] == 1)
        h2 += 3;
    if (currentpuzzle[1][2] == 2)
        h2 += 2;
    if (currentpuzzle[1][2] == 3)
        h2 += 1;
    if (currentpuzzle[1][2] == 5)
        h2 += 1;
    if (currentpuzzle[1][2] == 6)
        h2 += 2;
    if (currentpuzzle[1][2] == 7)
        h2 += 3;
    if (currentpuzzle[1][2] == 8)
        h2 += 2;
}
if (currentpuzzle[2][0] != 7)
{
    if (currentpuzzle[2][0] == 1)
        h2 += 2;
    if (currentpuzzle[2][0] == 2)
        h2 += 3;
    if (currentpuzzle[2][0] == 3)
        h2 += 4;
    if (currentpuzzle[2][0] == 4)
        h2 += 3;
    if (currentpuzzle[2][0] == 5)
        h2 += 2;
    if (currentpuzzle[2][0] == 6)
        h2 += 1;
    if (currentpuzzle[2][0] == 8)
        h2 += 1;
}
if (currentpuzzle[2][1] != 6)
{
    if (currentpuzzle[2][1] == 1)
        h2 += 3;
    if (currentpuzzle[2][1] == 2)
        h2 += 2;
    if (currentpuzzle[2][1] == 3)
        h2 += 3;
    if (currentpuzzle[2][1] == 4)
        h2 += 2;
    if (currentpuzzle[2][1] == 5)
        h2 += 1;
    if (currentpuzzle[2][1] == 7)
        h2 += 1;
    if (currentpuzzle[2][1] == 8)
        h2 += 2;
```

```
    }
    if (currentpuzzle[2][2] != 5)
    {
        if (currentpuzzle[2][2] == 1)
            h2 += 4;
        if (currentpuzzle[2][2] == 2)
            h2 += 3;
        if (currentpuzzle[2][2] == 3)
            h2 += 2;
        if (currentpuzzle[2][2] == 4)
            h2 += 1;
        if (currentpuzzle[2][2] == 6)
            h2 += 1;
        if (currentpuzzle[2][2] == 7)
            h2 += 2;
        if (currentpuzzle[2][2] == 8)
            h2 += 3;
    }

    return h2;
}


float S(Node* current, AStar* t)
{
    //a sequence score obtained by checking around the non-central squares in turn,
   // allotting 2 for every tile not followed by its proper successor and allotting
     //   0 for every other tile; a piece in the center scores one
    float s = 0;
    int** currentpuzzle = current->getPuzzle();
    if (currentpuzzle[1][1] != 0)
    {
        s += 1;
    }
    if (currentpuzzle[1][0] != (currentpuzzle[0][0] + 1))
    {
        s += 2;
    }
    if (currentpuzzle[0][2] != (currentpuzzle[0][1] + 1))
    {
        s += 2;
    }
    if (currentpuzzle[1][2] != (currentpuzzle[0][2] + 1))
    {
        s += 2;
    }
    if (currentpuzzle[2][2] != (currentpuzzle[1][2] + 1))
    {
        s += 2;
    }
    if (currentpuzzle[2][1] != (currentpuzzle[2][2] + 1))
    {
        s += 2;
```

```
    }
    if (currentpuzzle[2][0] != (currentpuzzle[2][1] + 1))
    {
        s += 2;
    }
    if (currentpuzzle[1][0] != (currentpuzzle[2][0] + 1))
    {
        s += 2;
    }
    return s;
}

float H(Node* current, AStar* t) {

    float H = h2(current, t) + (3 * S(current, t));
    return H;
}


float h4(Node* s, AStar* t)
{
    //h4(n) = # tiles misplaced in terms of row + # tiles misplaced in terms of column
    float misplacedRow = 0;
    float misplacedColum = 0;

    int** currentpuzzle = s->getPuzzle();
    if(currentpuzzle[0][0] != 1 && currentpuzzle[0][0] != 2 && currentpuzzle[0][0] != 3)
        misplacedRow += 1;
    if(currentpuzzle[0][1] != 1 && currentpuzzle[0][1] != 2 && currentpuzzle[0][1] != 3)
        misplacedRow += 1;
    if(currentpuzzle[0][2] != 1 && currentpuzzle[0][2] != 2 && currentpuzzle[0][2] != 3)
        misplacedRow += 1;
    if(currentpuzzle[1][0] != 8 && currentpuzzle[1][0] != 0 && currentpuzzle[1][0] != 4)
        misplacedRow += 1;
    if(currentpuzzle[1][1] != 8 && currentpuzzle[1][1] != 0 && currentpuzzle[1][1] != 4)
        misplacedRow += 1;
    if(currentpuzzle[1][2] != 8 && currentpuzzle[1][2] != 0 && currentpuzzle[1][2] != 4)
        misplacedRow += 1;
    if(currentpuzzle[2][0] != 7 && currentpuzzle[2][0] != 6 && currentpuzzle[2][0] != 5)
        misplacedRow += 1;
    if(currentpuzzle[2][1] != 7 && currentpuzzle[2][1] != 6 && currentpuzzle[2][1] != 5)
        misplacedRow += 1;
    if(currentpuzzle[2][2] != 7 && currentpuzzle[2][2] != 6 && currentpuzzle[2][2] != 5)
        misplacedRow += 1;


    if(currentpuzzle[0][0] != 1 && currentpuzzle[0][0] != 8 && currentpuzzle[0][0] != 7)
        misplacedColum += 1;
    if(currentpuzzle[1][0] != 1 && currentpuzzle[1][0] != 8 && currentpuzzle[1][0] != 7)
        misplacedColum += 1;
    if(currentpuzzle[2][0] != 1 && currentpuzzle[2][0] != 8 && currentpuzzle[2][0] != 7)
        misplacedColum += 1;
    if(currentpuzzle[0][1] != 2 && currentpuzzle[0][1] != 0 && currentpuzzle[0][1] != 6)
```

```
        misplacedColum += 1;
    if(currentpuzzle[1][1] != 2 && currentpuzzle[1][1] != 0 && currentpuzzle[1][1] != 6)
        misplacedColum += 1;
    if(currentpuzzle[2][1] != 2 && currentpuzzle[2][1] != 0 && currentpuzzle[2][1] != 6)
        misplacedColum += 1;
    if(currentpuzzle[0][2] != 3 && currentpuzzle[0][2] != 4 && currentpuzzle[0][2] != 5)
        misplacedColum += 1;
    if(currentpuzzle[1][2] != 3 && currentpuzzle[1][2] != 4 && currentpuzzle[1][2] != 5)
        misplacedColum += 1;
    if(currentpuzzle[2][2] != 3 && currentpuzzle[2][2] != 4 && currentpuzzle[2][2] != 5)
        misplacedColum += 1;

    return (misplacedColum + misplacedRow);
}

float h5(Node* s, AStar* t)
{
    //h5(n) =  Sum of Eucledian distances of the tiles from their
    //goal positions

    float h5 = 0;
    int** currentpuzzle = s->getPuzzle();

    if (currentpuzzle[0][0] != 1)
    {
        if (currentpuzzle[0][0] == 2)
            h5 += 1;
        if (currentpuzzle[0][0] == 3)
            h5 += 2;
        if (currentpuzzle[0][0] == 4)
            h5 += sqrt(5);
        if (currentpuzzle[0][0] == 5)
            h5 += sqrt(8);
        if (currentpuzzle[0][0] == 6)
            h5 += sqrt(5);
        if (currentpuzzle[0][0] == 7)
            h5 += 2;
        if (currentpuzzle[0][0] == 8)
            h5 += 1;
    }
    if (currentpuzzle[0][1] != 2)
    {
        if (currentpuzzle[0][1] == 1)
            h5 += 1;
        if (currentpuzzle[0][1] == 3)
            h5 += 1;
        if (currentpuzzle[0][1] == 4)
            h5 += sqrt(2);
        if (currentpuzzle[0][1] == 5)
            h5 += sqrt(5);
        if (currentpuzzle[0][1] == 6)
            h5 += 2;
        if (currentpuzzle[0][1] == 7)
```

```
        h5 += sqrt(5);
    if (currentpuzzle[0][1] == 8)
        h5 += sqrt(2);

}
if (currentpuzzle[0][2] != 3)
{
    if (currentpuzzle[0][2] == 1)
        h5 += 2;
    if (currentpuzzle[0][2] == 2)
        h5 += 1;
    if (currentpuzzle[0][2] == 4)
        h5 += 1;
    if (currentpuzzle[0][2] == 5)
        h5 += 2;
    if (currentpuzzle[0][2] == 6)
        h5 += sqrt(5);
    if (currentpuzzle[0][2] == 7)
        h5 += sqrt(8);
    if (currentpuzzle[0][2] == 8)
        h5 += sqrt(5);
}
if (currentpuzzle[1][0] != 8)
{
    if (currentpuzzle[1][0] == 1)
        h5 += 1;
    if (currentpuzzle[1][0] == 2)
        h5 += sqrt(2);
    if (currentpuzzle[1][0] == 3)
        h5 += sqrt(5);
    if (currentpuzzle[1][0] == 4)
        h5 += 2;
    if (currentpuzzle[1][0] == 5)
        h5 += sqrt(5);
    if (currentpuzzle[1][0] == 6)
        h5 += sqrt(2);
    if (currentpuzzle[1][0] == 7)
        h5 += 1;
}
if (currentpuzzle[1][1] != 0)
{
    if (currentpuzzle[1][1] == 1)
        h5 += sqrt(2);
    if (currentpuzzle[1][1] == 2)
        h5 += 1;
    if (currentpuzzle[1][1] == 3)
        h5 += sqrt(2);
    if (currentpuzzle[1][1] == 4)
        h5 += 1;
    if (currentpuzzle[1][1] == 5)
        h5 += sqrt(2);
    if (currentpuzzle[1][1] == 6)
        h5 += 1;
```

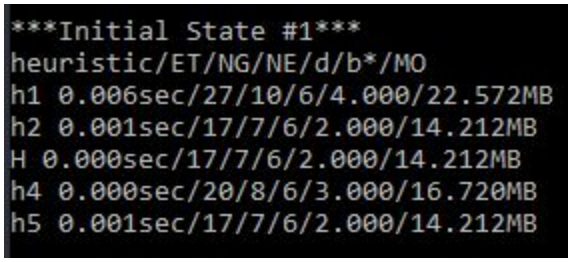
```
    if (currentpuzzle[1][1] == 7)
        h5 += sqrt(2);
    if (currentpuzzle[1][1] == 8)
        h5 += 1;
}
if (currentpuzzle[1][2] != 4)
{
    if (currentpuzzle[1][2] == 1)
        h5 += sqrt(5);
    if (currentpuzzle[1][2] == 2)
        h5 += sqrt(2);
    if (currentpuzzle[1][2] == 3)
        h5 += 1;
    if (currentpuzzle[1][2] == 5)
        h5 += 1;
    if (currentpuzzle[1][2] == 6)
        h5 += sqrt(2);
    if (currentpuzzle[1][2] == 7)
        h5 += sqrt(5);
    if (currentpuzzle[1][2] == 8)
        h5 += 2;
}
if (currentpuzzle[2][0] != 7)
{
    if (currentpuzzle[2][0] == 1)
        h5 += 2;
    if (currentpuzzle[2][0] == 2)
        h5 += sqrt(5);
    if (currentpuzzle[2][0] == 3)
        h5 += sqrt(8);
    if (currentpuzzle[2][0] == 4)
        h5 += sqrt(5);
    if (currentpuzzle[2][0] == 5)
        h5 += 2;
    if (currentpuzzle[2][0] == 6)
        h5 += 1;
    if (currentpuzzle[2][0] == 8)
        h5 += 1;
}
if (currentpuzzle[2][1] != 6)
{
    if (currentpuzzle[2][1] == 1)
        h5 += sqrt(5);
    if (currentpuzzle[2][1] == 2)
        h5 += 2;
    if (currentpuzzle[2][1] == 3)
        h5 += sqrt(5);
    if (currentpuzzle[2][1] == 4)
        h5 += sqrt(2);
    if (currentpuzzle[2][1] == 5)
        h5 += 1;
    if (currentpuzzle[2][1] == 7)
        h5 += 1;
```

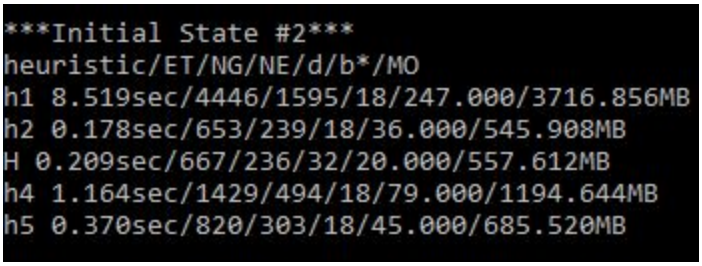
```
        if (currentpuzzle[2][1] == 8)
            h5 += sqrt(2);
    }
    if (currentpuzzle[2][2] != 5)
    {
        if (currentpuzzle[2][2] == 1)
            h5 += sqrt(8);
        if (currentpuzzle[2][2] == 2)
            h5 += sqrt(5);
        if (currentpuzzle[2][2] == 3)
            h5 += 2;
        if (currentpuzzle[2][2] == 4)
            h5 += 1;
        if (currentpuzzle[2][2] == 6)
            h5 += 1;
        if (currentpuzzle[2][2] == 7)
            h5 += 2;
        if (currentpuzzle[2][2] == 8)
            h5 += sqrt(5);
    }


 return  h5;
}
```

Description of Heuristic Functions

H1: H1 is a very simple heuristic function, if a square is out of place, it will increment by 1. If it is in the right place, nothing will be added to its overall value. This function is admissible.
H2: H2 checks each square, if it is the right number, nothing is added. If it is in the wrong square, it adds as many squares away the number currently in the square is away from its goal square. This function is admissible.
H: Is a combination of two functions. It is H2(n) + 3*S(n). Where S(n) is a sequence score obtained by checking around the non-central squares in turn allotting 2 for every square not followed by its proper successor and allotting 0 for every other tile; a piece in the center scores one. This function is non admissible
H4: H4(n) = (number of squares misplaced in terms of row) + (number of squares misplaced in terms of column). If the square's number is in the wrong row, add one to the first, If the square's number is in the wrong column, add to the latter. This function is admissible
H5: Sum of Euclidean distances of the squares from their goal positions. The Euclidean Distance can be calculated by root((x2-x1)^2 + (y2-y1)^2). This gives a more exact value than H2 does. This function is admissible.

Analysis of Results





Based upon the results shown above, the best performing heuristic function on both initial state 1 and 2, is h2. Followed by H, which went far further down in depth than the others, but also didn't generate nearly as many nodes as h4 and h1.. Third place is taken by h5. Fourth is h4 which generated almost double the amount of nodes as h5. And last is h1 which generated and expanded an incredibly high number of nodes.

Conclusion

This project helped me build my teamwork skills, we were all able to work together and create a very good product in the end. This project also helped me better understand the A star algorithm and how smart choosing works. It was a very informative and learning experience. By the end we learned that our h2 function was the most efficient out of all of them, and h1 was the least efficient.