Testing the Intelligence of the A* Algorithm

Research Assistant: Aleem Anula, Barindra Narinesingh, Willy William
Project Director: Minhua Huang
Department of Computer Science and Mathematics

YORK College

CUNY The City University of New York

INTRODUCTION

This research is to build an 8 puzzle solver that solves itself and outputs the process as it transitions between choices. Normally can build a starting puzzle and an ending puzzle then compare them as a simple game however we intend on building a puzzle that solves itself fast and efficiently. Our method of going about this is by the direct method of designing a matrix to hold the 8 puzzle then design a data structure to hold the matrix and use the A* algorithm to solve our puzzle, if it is smart it will be done fast.

The 8 Puzzle

- The 8 puzzle is a 3 by 3 grid with one blank and every other spot is number between 1 through 8.
- The goal is to get the numbers to go from 1 to 8 and having the last be the blank.

START		
2	6	1
	7	8
3	5	4

GOAL		
1	2	3
4	5	6
7	8	

OBJECTIVES

Our main objective is to solve the puzzle.
Our secondary objective is to solve it fast.
Our last objective is to test its intelligence.

PROCEDURE

1st We will use the direct approach, and made a 3x3 matrix.
2nd We will then generate a goal state 3x3 matrix. And then compared it to our matrix.
3rd We Will then figure out how to make sure it was solvable, inversion will be used.
4th The A* algorithm will be used to generates children. Heuristics will be used as costs and means of measurement. Children will then be put into objects and then compared.
5th We will then out put to a .txt file the series of moves made to the goal state
6Th We will then test the series of moves for existence of intelligence against an inactive algorithm.

```
typedef struct node_object
{
     int matrix[3][3];          // with number 0 as blank tile

     int g_n;
     int h_n;                   //used function for herustic value
                                //and manhatten distance

     int f_n;                   //Cost Function f(n)=g(n)+h(n)

     string move_d;             //move directions
     string move_c;             //move choice

     node_object * parent;  //pointer to the parent node

} node;
```

MATERIALS & METHODS

Dev++, Xcode, Visual Studio and Arduino were used as coders and compilers.
Arduino and Servos motor were used with an Xbox One and controller.

Our code running

```
Welcome to the 8 puzzle solver!

Enter your matrix to solve :

1 2 3 4 5 6 0 7 8
```

```
: 1 : 2 : 3 :
: 4 : 5 : 6 :
:   : 7 : 8 :
```

```
Total moves made : 2
Total states (start state+ states
traversed + goal state): 3
Total nodes traversed looking for the
above?: 200

Total websites looked at trying to
figure out how to get this code to work
 = 21 a day * 7 days a week * 7 weeks
 = 1029 websites

Time taken for process to complete:
0.00337 seconds.
```

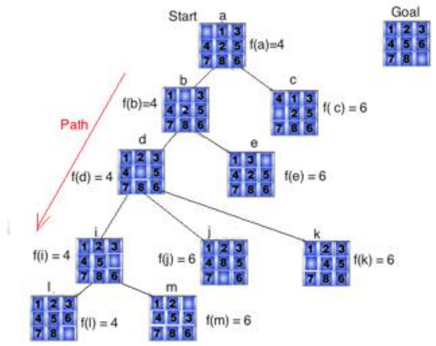
A simple visualization

- G = {1,2,3,4,5,6,7,8,0}
- S2= {1,2,3,4,5,6,7,8,0}
- S1={1,2,3,4,5,6,7,0,8}
- Given:
- S0={1,2,3,4,5,6,0,7,8}

This guy uses the A* Algorithm

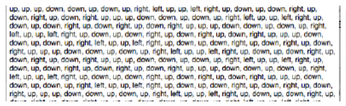
- There are many paths The knight chooses the one that directly gets to the goal state where Link is.
- Copyright Nintendo, please don't sue us.

An example of the A* Algorithm

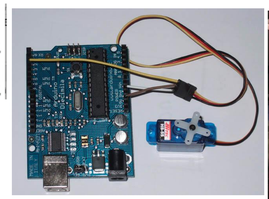


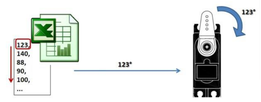
The A* algorithm will expand possible paths and then choose the one with the least cost then take that choice as the next state until the goal state is reached. The further away we get from the goal state we end up with high cost states which we will close and ignore. Each state change corresponds with a movement of the empty tile. Our code moves the empty tile and keeps track of movements. Outputting up, down, left, right, etc. until the goal state is reached. We spent time wondering on how to test intelligence and while taking a break from coding we developed the idea below.

1) Generate Ouput from 8 Puzzle → 2) Program Arduino to move servos based on input→ 3) Move Xbox Controller tied to servos → 4) Test Inactive algorithm in Halo 5 against A* movements



```
#include <Servo.h>

int servoPin = 9;

Servo servo;

int angle = 0;  // servo position in degrees

void setup()
{
 servo.attach(servoPin);
}


void loop()
{
 // scan from 0 to 180 degrees
 for(angle = 0; angle < 180; angle++)
 {
  servo.write(angle);
  delay(15);
 }
 // now scan back from 180 to 0 degrees
 for(angle = 180; angle > 0; angle--)
 {
  servo.write(angle);
  delay(15);
 }
}
```





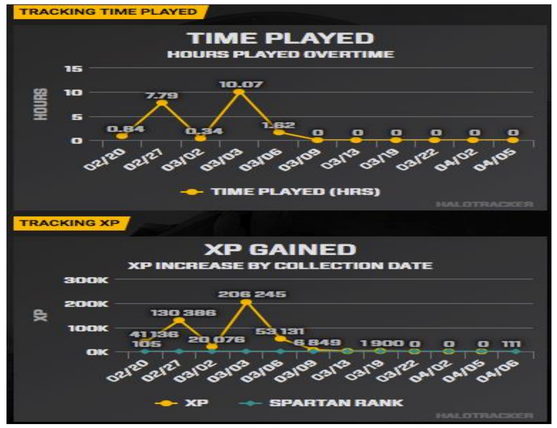
Our 8 puzzle will output to a .txt file which will then be read by our Arduino system which will then make degree movements based on the states. Example:
Up=30 up
Down=30 down
Right= 60 up
Left = 60 down

TESTING AND CONCLUSIONS

The first few times after we manage to get the code to run, it would expand nodes to infinity, this was resolved by putting a limit on expanding.

STRING	NODES EXPANDED	MOVES MADE	TIME TAKEN IN SECONDS
MAX	181440		
876543210	181428	30	8.18869
120345678	131385	22	4.13896
132457680	127252	22	3.73363
412357680	100904	20	2.77018
134586702	75522	19	1.85557
132465780	63187	18	1.41702
123405678	25695	14	0.513529
123456078	200	2	0.005631

The more complicated the puzzle the longer it took to be solved. More nodes had to be expanded to find the solution even though amount of moves made were the same.



The inactive algorithm is a algorithm in video games that tests if the player is there or not and if they are not there the player is kicked from the game.

Testing the 8 puzzle output against the inactive algorithm proved to be successful, the code ran on loop for 7 hours on 2/27 and for 10 hours on 3/3 proving the output was considered intelligent since the inactive algorithm did not detect that there was anything wrong.

FUTURE GOALS

Implementation of A Graphical User Interface.

Our program only displays the states as it solves the puzzle. We would want it to say all possible directions.

An alternative input method, perhaps reading from a .txt file or configuring it to read in strings instead of having to space each number.

Test the A* algorithm against other games to see how their inactive algorithm measures up.

REFERENCES

"Introduction to A* From Amit's Thoughts on Pathfinding." *Introduction to A**. Web. 17 Apr. 2016.

"8-Puzzle Programming." *8-Puzzle Programming*. Web. 17 Apr. 2016.

Inspiration from

"CIG 2012 (Granada)." *CIG 2012 (Granada)*. Web. 17 Apr. 2016.