University of the Western Cape
Department of Computer Science

CSC311: Assignment – Artificial Intelligence (Block 3) [40 Marks]

Due Date: 9 May 2023 at 11pm

Moving Magic Square is the name of a game that is based on the concept of a magic square. A *magic square* is any square array of numbers, usually positive integers, in which the sums of the numbers in each row, each column, and both main diagonals are the same. For example, the *3* x *3* square in Table 1 is a magic square because the sum of every row, every column and the two diagonals is 15.

Table 1

6	1	8
7	5	3
2	9	4

The game, *Moving Magic Square*, is played on any *n* x *n* grid containing positive integer numbers from 1, … , n^2. The number n^2 is the *movable number*. You can move the number n^2 in one of four directions (up/down/left/right), and swap n^2 with the number that is currently occupying that cell. The player wants to move the number n^2 to reach a goal state such that the sum of the *n* numbers in every row, column, and both diagonals is equal to *k*. There are multiple states that satisfy this condition, and you can stop the game when you find the first goal state.

In a *3* x *3* game, *9* is the movable number.

The grid in Table 2 shows an initial state for a *3* x *3* game.

Table 2

6	9	8
7	1	3
2	5	4

The grid in Table 3 shows the result of a "down" move on the initial state shown in Table 2.

Table 3

6	1	8
7	9	3
2	5	4

After another "down" move on the grid in Table 3, the goal state (i.e. the magic square) shown in Table 1 results. The game has been completed.

Questions

1. Formulate the *3* x *3* game as a search problem, i.e. define the states, the moves, etc. [2]

2. Draw the state space that would be generated by the Breadth-First Search algorithm for the initial state given in Table 2. You can stop when the first goal state is reached. Perform the moves strictly in the following sequence: Up; Down; Left; Right. Do not create more than one copy of any particular state, and identify the goal state. [4]

3. Let *h(n)* be a heuristic function for a state *n* in the game where *h(n)* is the sum of all differences between *k* and the total of the entries in any row, column or diagonal:

$$h(n) = \sum_{i=1}^{n} \left(| k - \sum_{j=1}^{n} t_{ij} | \right) + \sum_{j=1}^{n} \left(| k - \sum_{i=1}^{n} t_{ij} | \right) + \left(| k - \sum_{i=1}^{n} t_{ii} | \right) + \left(| k - \sum_{i=1}^{n} t_{ip} | \right)$$

where $p = n - i + 1$

For the state depicted by the grid in Table 2, (with $k = 15$ and $n = 3$)

$h(n) = (8 + 4 + 4) + (0 + 0 + 0) + 4 + 4 = 24$

Show the heuristic function values for each of the states in the drawing of your answer to question 2. [2]

4. Write a Java program to play the *Moving Magic Square* game. You may assume $n = 3$ and $k = 15$. The initial state (i.e. a *3* x *3* grid of integer containing the integer numbers from 1 to 9) will be read from a text file. The program must use a *Greedy Best First* search algorithm to solve the game, and to show the board (and the relevant heuristic value) after each move. Use the heuristic function described in question 3. More details on the program follow in the instruction below. [32]

Follow the instructions below carefully:

1. You are required to program in Java and use object-oriented programming concepts. Save all your files in a compressed (.zip) folder with the following naming convention

XXYYZZZ_CSC212_Practical_1_Term_4_2022.zip

where XXYYZZZ corresponds to your student number. Include the following at the beginning of the Java file and in the Word document:

\\ Student surname
\\ Student name
\\ Student number
\\ CSC311 2023 AI Practical

Call your Java program `MoveMagicSquare.java`

2. Write a class called `State` with the following variables to represent a state.
   - An array with 9 entries to represent the board
   - an integer field to represent the heuristic value of the state
   - an integer field to present the position of the value "9" in the array.

   Note that the rows of the board must be from top to bottom in the array. For example, Table 2 will be presented as follows in the integer array:

Index in array:	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
	6	9	8	7	1	3	2	5	4

Write the necessary methods of the class including:
   - Accessor and mutator methods to retrieve and modify attributes of the class;
   - Methods to perform moves (i.e. generate children states)
   - A method to print the contents of a state.

3. Use the (built-in) Java Collections PriorityQueue (PQ) data structure to store the frontier list of expanded states. The PQ is of type `State` (represented in the data structure described above) and is sorted in increasing order according to the heuristic value of each state. Two states that have the same heuristic value may appear in any order in the PQ. (Note that you have to overload the comparison method in your PQ to sort the items in PQ in ascending order).

4. Test your Java program on the following initial states (the two input files, *InputA.txt* and *InputB.txt*, are provided):

   a.

6	1	8
2	7	4
9	3	5

   b.

6	9	8
7	1	3
2	5	4

Show every state (and its heuristic value) that is expanded by the algorithm (i.e. removed from the PQ) as well as the final solution (i.e. goal state) by printing it to an output (text) file (called *OutputA.txt* and *OutputB.txt*, respectively).

Also print the states on the screen. So, we want to see every move on the screen and in an output file. Simply print the contents of array in one line followed by the heuristic value of the state.

Halt the execution of your program if it has not reached a goal state after 700 moves.

5. Submit the following files on iKamva:
    - Your written answers to questions 1-3 in a Word or a .pdf file.
    - Your Java source code.
    - The two output files.

6. Demonstrate your program to a marker. Instructions will follow.

Note: You have to complete the assignment on your own and write your own code. Plagiarism in any form will result in a mark of zero.

Hint:
A partial outline of the class `State` is shown below. Add any necessary methods.

```
class State implements Comparable<State>
{
  private int hValue;
  private int pos9;
  private int[] board = new int[9];

  public State(int position9, int[] aboard )  {
    … }

  @Override
    public int compareTo(State o) {
        … }

  //  The four moves of cell 9 follow below
  public int Get_hValue( ){
     … }

  …

} //End of State class
```