IZMIR UNIVERSITY OF ECONOMICS

DEPARTMENT OF SOFTWARE ENGINEERING

SE 420

Artificial Intelligence and Expert Systems

Project Report

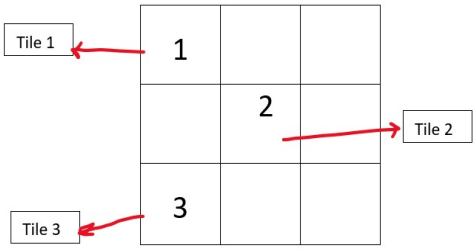
3*3 BOARD GAME

Team Members

Ercan ACAR - 20180601001
Mustafa ALAN - 20180601003
Fidan ÇELENK - 20180601015
Ayşegül MERCAN - 20180601029
Cem ÖZCAN - 20170601024

1. Short Definition of the Game

The aim of the 3*3 board game is to reach the goal state where Tile #1, #2, and #3 are located on the board.



2. Rules of the Game

- The initial and goal states will be given by the user.
- There will be three tiles to move: Tile #1, Tile #2, Tile #3, and their goal tiles.
- The tiles can be moved up, down, right, or left until it reaches their goal tiles' locations.
- The game will begin with the move of Tile #1 (if required) and go on with the moves of other tiles in order.
  For example:
  1st step: move Tile #1
  2nd step: move Tile #2
  3rd step: move Tile #3
  4th step: move Tile #1
  5th step: move Tile #2
  6th step: move Tile #3
  ……..

- Distance (cost) between two neighboring states will be measured based on the move costs which are given by the user as given below:
  Tile #1:
  right or left move → cost = 2
  up or down move → cost =1
  Tile #2:
  Right or left move → cost =1
  up or down move → cost =2
  Tile #3:
  right or left move → cost =3
  up or down move → cost =4

- The user can select one of two searching strategies: Uniform Cost and A* search (use Manhattan distance as heuristics).
- The expansion will go on till the 20th expanded node. The program will print out each expanded state and compare it with the given goal state.
- If we reach the goal state, the program will stop expanding nodes.

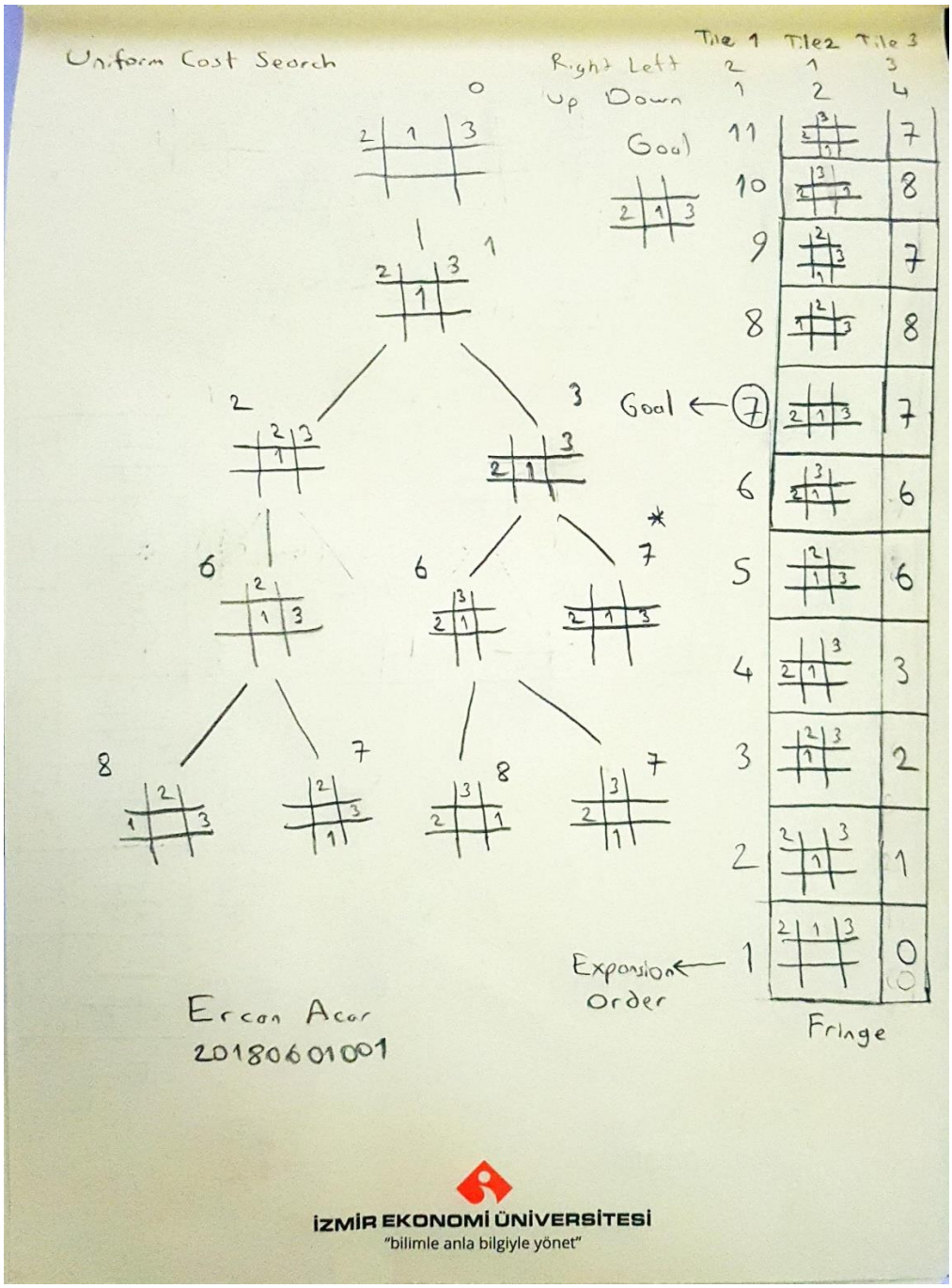
3. Implementation of the Game

- We used Java as a programming language for the implementation of the game.
- We used 3 different classes for implementation which are Main, State and Tile. States will be the nodes in our program and each state will have Tile1, Tile2, and Tile3 on it. Each tile will have coordinates on a 3x3 matrix.
- We have our main method in the Main class.
- We create our priority queue which has the comparator up to the total cost number in the states.
- We add the initial state into our priority queue.
- Then we go into the loop of expansion.
- In our loop firstly, we expand the node (State) from our priority queue.
- Then we check if we reach the goal state. If not, then we start by moving the first tile which is Tile1.
- When we want to move Tile1 we get Current State (Expanded node)'s new version in which the Tile1 moves up, down, right, or left, we add each of these versions in the priority queue if possible.
- New versions that I mentioned in the previous part will keep the record of the tile that its parent was moving so that it can continue from the next tile.
- Also these each new version state which is added into the three will calculate its cost in the move function by adding the parent state's cost, its own moves cost, and heuristic (which is 0 in uniform cost search but will be Manhattan Distance in A*).
- We will keep on going like this until we reach the goal state and know that our Tile1's location is the same as GoalTile1, Tile2's location is the same as GoalTile2, and Tile3's location is the same as GoalTile3.

3.1 Searching Strategies Examples with **Outputs** and **Hand Calculations :**

3.1.1- An Example for **Uniform cost** search :

a) Hand Calculations



b) Execution Output

Results

Order #1

2	1	3	Cost:	0
0	0	0		
0	0	0		

Order #2

2	0	3	Cost:	1
0	1	0		
0	0	0		

Order #3

0	2	3	Cost:	2
0	1	0		
0	0	0		

Order #4

0	0	3	Cost:	3
2	1	0		
0	0	0		

Order #5

0	2	0	Cost:	6
0	1	3		
0	0	0		

Order #6

0	3	0	Cost:	6
2	1	0		
0	0	0		

Order #7

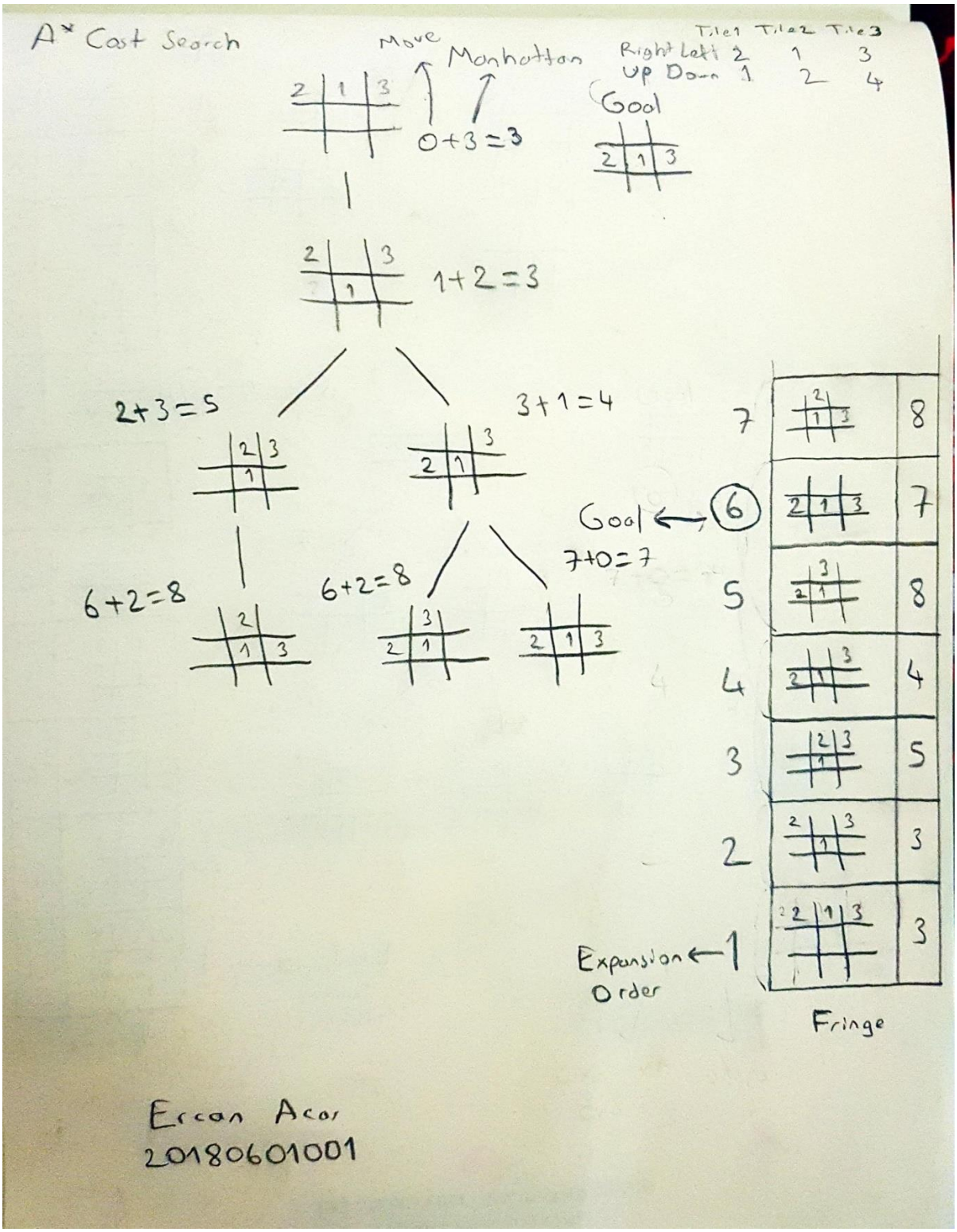
0	2	0	Cost:	7
0	0	3		
0	1	0		

Goal!

0	0	0	Cost:	7
2	1	3		
0	0	0		

3.1.2- An example for **A*** search (used Manhattan distance as heuristics) :

a) Hand Calculations



b) Execution Output

Results

Order #1

2	1	3	Cost:	3
0	0	0		
0	0	0		

Order #2

2	0	3	Cost:	3
0	1	0		
0	0	0		

Order #3

0	0	3	Cost:	4
2	1	0		
0	0	0		

Order #4

0	2	3	Cost:	5
0	1	0		
0	0	0		

Goal!

0	0	0	Cost:	7
2	1	3		
0	0	0		