CENG317 – Principles of Artificial Intelligence

Homework #2

Due: 05.11.2023 @16:59

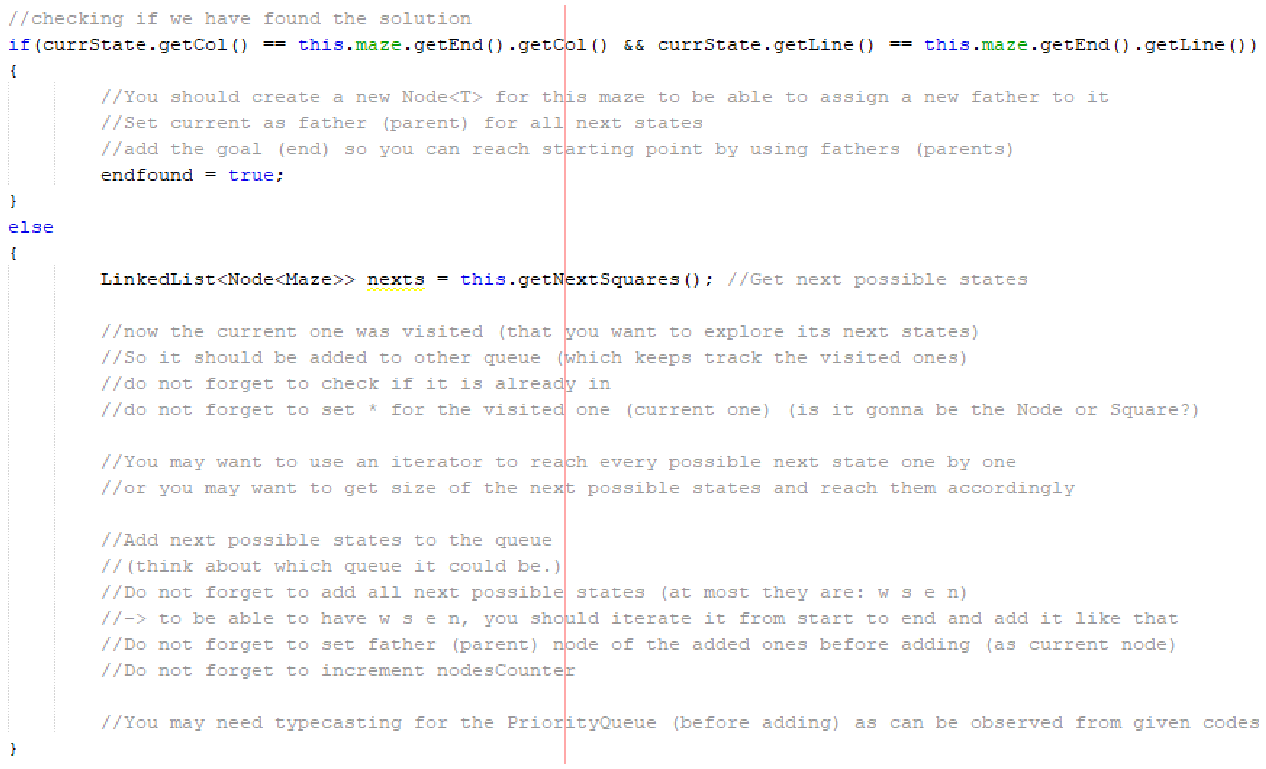
ATTENTION:

- Programs which contain **errors** will **NOT** be graded.
- In all of the programs, use **COMMENTS** where necessary.
- Code blocks inside comments will **NOT** be graded.
- Send only .text or .java file and the .pdf file in a .zip file named as **"hw2_yourName_yourID.zip"**.

Disciplinary action to be taken against cheating is arranged by the Rules and Regulations Governing Student Disciplinary Actions in Institutions of Higher Education.

QUESTIONS (100pts)

1. (15pts) Complete the Java program that will read the given text files (maze.txt and maze2.txt) and prints the output text files (astar.txt, astar_euclid.txt, astar2.txt and astar2_euclid.txt). You will only implement the following "if-else" sections (inside AStarSolver class under solve() method):

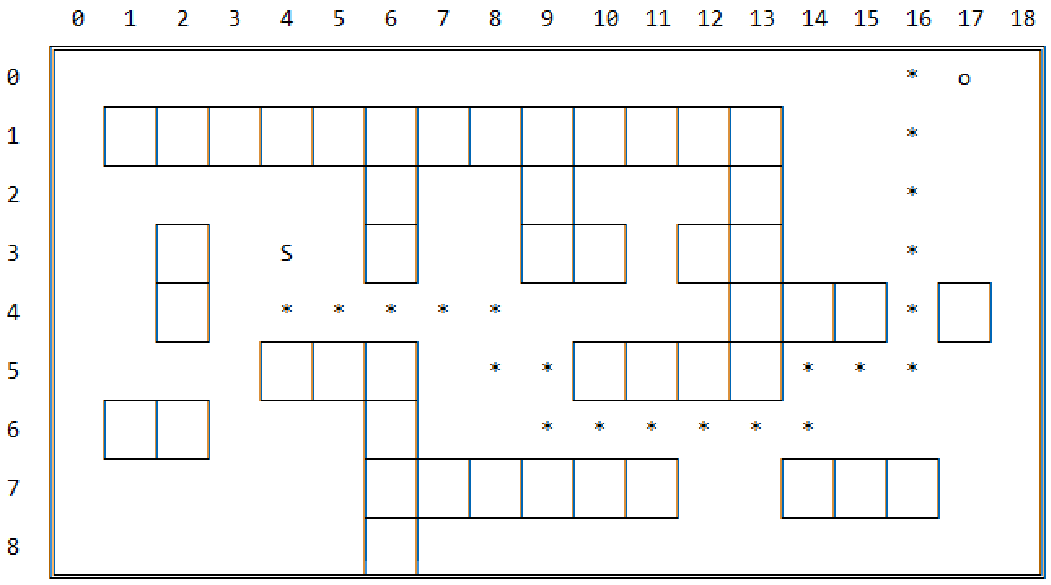


```java
//checking if we have found the solution
if(currState.getCol() == this.maze.getEnd().getCol() && currState.getLine() == this.maze.getEnd().getLine())
{
        //You should create a new Node<T> for this maze to be able to assign a new father to it
        //Set current as father (parent) for all next states
        //add the goal (end) so you can reach starting point by using fathers (parents)
        endfound = true;
}
else
{
        LinkedList<Node<Maze>> nexts = this.getNextSquares(); //Get next possible states

        //now the current one was visited (that you want to explore its next states)
        //So it should be added to other queue (which keeps track the visited ones)
        //do not forget to check if it is already in
        //do not forget to set * for the visited one (current one) (is it gonna be the Node or Square?)

        //You may want to use an iterator to reach every possible next state one by one
        //or you may want to get size of the next possible states and reach them accordingly

        //Add next possible states to the queue
        //(think about which queue it could be.)
        //Do not forget to add all next possible states (at most they are: w s e n)
        //-> to be able to have w s e n, you should iterate it from start to end and add it like that
        //Do not forget to set father (parent) node of the added ones before adding (as current node)
        //Do not forget to increment nodesCounter

        //You may need typecasting for the PriorityQueue (before adding) as can be observed from given codes
}
```

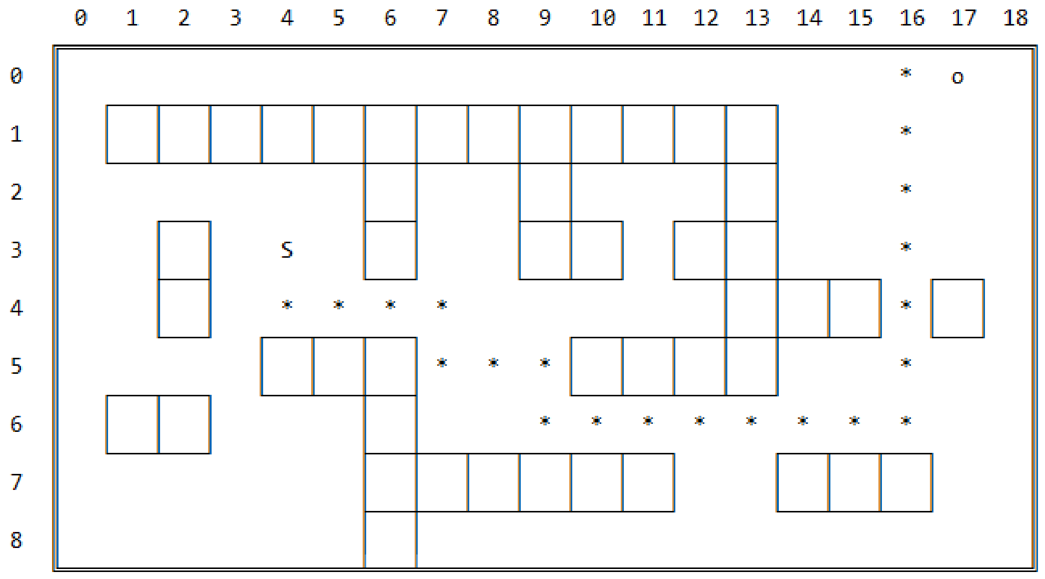
(-15pts) You will send **ONLY** the code of "if-else" sections in a .txt or .java file (not the code of the entire class).

Sample Outputs:

```
    ___                   __  ___            __          __  __
   /   | __/|_           /  |/  /___ _____  / /_  ____ _/ /_/ /_____ _____
  / /| ||    /  ______  / /|_/ / __ `/ __ \/ __ \/ __ `/ __/ __/ __ `/ __ \
 / ___ /_ __|  /_____/ / /  / / /_/ / / / / / / / /_/ / /_/ /_/ /_/ / / / /
/_/  |_||/            /_/  /_/\__,_/_/ /_/_/ /_/\__,_/\__/\__/\__,_/_/ /_/
Path: [0, 17](22.0)(End) <- [0, 16](22.0) <- [1, 16](22.0) <- [2, 16](22.0) <- [3
Path length: 22
Number of nodes created: 68
Execution time: 0.0 seconds
```



```
    ___                   ______           ___     __
   /   | __/|_           / ____/_  _______/ (_)___/ /
  / /| ||    /  ______  / __/ / / / / ___/ / / __  /
 / ___ /_ __|  /_____/ / /___/ /_/ / /__/ / / /_/ /
/_/  |_||/            /_____/\__,_/\___/_/_/\__,_/
Path: [0, 17](22.0)(End) <- [0, 16](22.0) <- [1, 16](21.414213562373096) <- [2,
Path length: 22
Number of nodes created: 94
Execution time: 0.0 seconds
```



2. (60pts) (For all, give short answers only, no extra explanations needed)

2.1. What are the states for this problem? **(5pts)**

2.2. What is the branching factor (maximum) for the agent? **(5pts)**

2.3. Compare the A* search results using Manhattan distance and Euclidean distance. **(10pts)**

- **a.** Why are they different?
- **b.** Which one is better? **c.** Why?

2.4. Compare one of the A* search results (Manhattan or Euclidean) with the BFS and DFS results you worked with before (the ones you got in hw1). (Compare the three) **(10pts)**

- **a.** What is different?
- **b.** Why did this happen?
- **c.** Which one is better? **d.** Why?

2.5. Answer separately for both heuristic functions. **(15pts)**

- **a.** Are these heuristic functions useful?
- **b.** Are they optimistic enough? Or, are they pessimistic?
- **c.** Do they provide optimal results for A* search? **d.** Why?

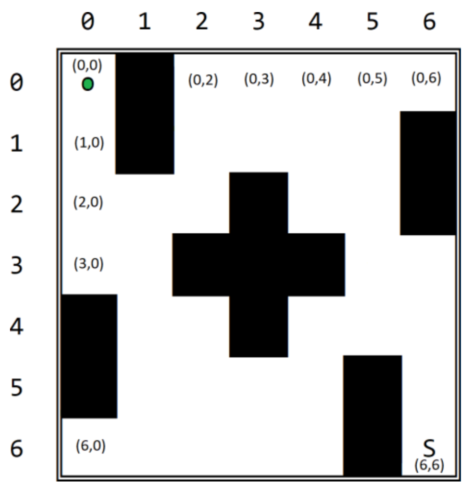
2.6. Answer separately for both heuristic functions. **(15pts)**

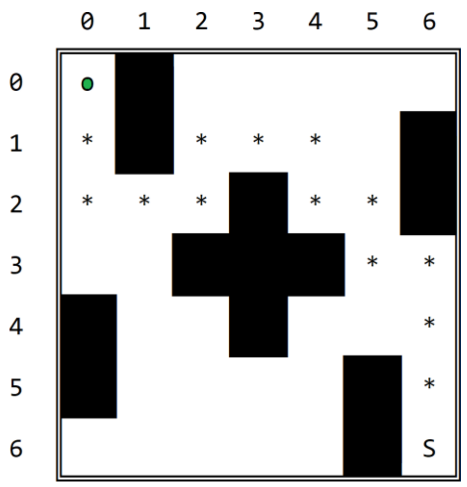
- **a.** Are they admissible heuristics? Why? Justify your answer.
- **b.** Are they consistent heuristics? Why? Justify your answer.

3. (25pts)

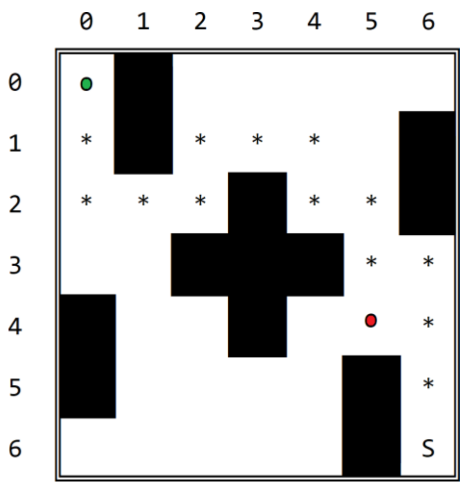
As you can observe from this GIF (click on the link), the A* algorithm expands more nodes than the nodes in the resulting path as it searches (also check out the sample outputs given to you, where the path has 22 nodes but A* expanded 68 or 94 nodes).

Examine "astar2.txt", which is the output of the A* search using the Manhattan distance, it expanded 30 nodes before declaring success and finding the path to the goal.





Task: Find out which nodes are expanded. Mark them using a markup tool (like Paint) to get a result like the example below (there is only 1 mark for the expanded node):



Your answer in the pdf should have the same structure as the one on the left, with all expanded nodes marked **(-10pts)**.

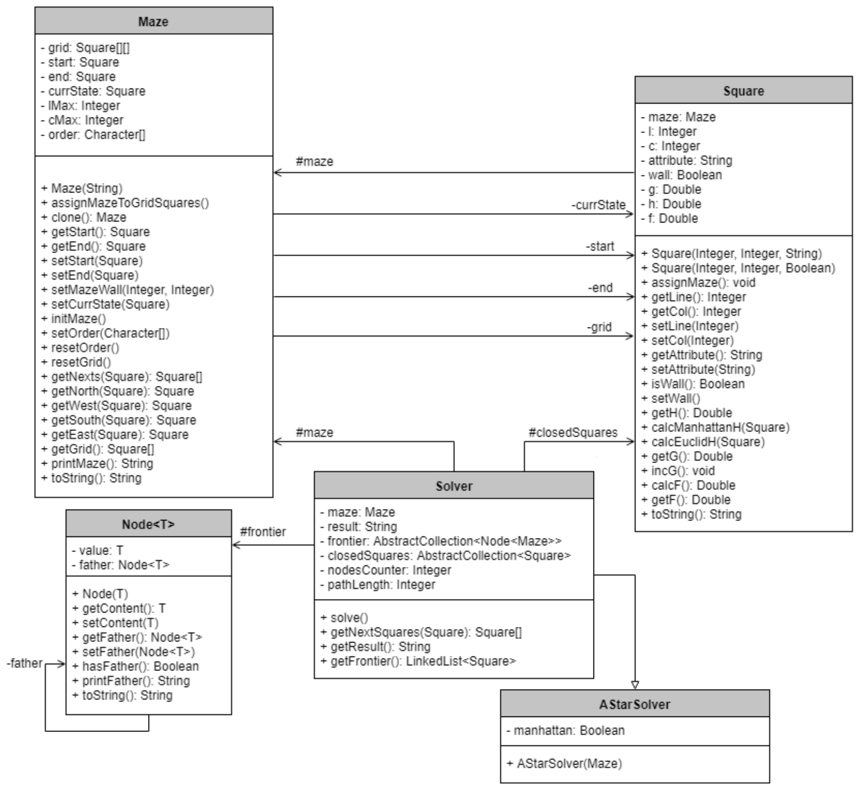
Use this as a marker for expanded ones: ●

(You can choose to use the images in the hw2.zip file, or type 'o' in the expanded ones in the .txt file and get the screenshot then fill o's in red [don't forget to fill the goal node with green and the walls with black])

You should prepare a simple answer sheet in a .pdf file format **(-20pts)** that answers all the questions asked above. It should only contain "key" information about questions and nothing more. Try not to exceed 1 page. Also include your name and ID info to the pdf.

At the end of your pdf, rate the difficulty of this assignment on a scale of 1 to 10 (1 being the easiest homework ever, 10 being the hardest homework ever). **(-10pts)**

This is the class diagram of the system:


Maze
- grid: Square[][]
- start: Square
- end: Square
- currState: Square
- lMax: Integer
- cMax: Integer
- order: Character[]
+ Maze(String)
+ assignMazeToGridSquares()
+ clone(): Maze
+ getStart(): Square
+ getEnd(): Square
+ setStart(Square)
+ setEnd(Square)
+ setMazeWall(Integer, Integer)
+ setCurrState(Square)
+ initMaze()
+ setOrder(Character[])
+ resetOrder()
+ resetGrid()
+ getNexts(Square): Square[]
+ getNorth(Square): Square
+ getWest(Square): Square
+ getSouth(Square): Square
+ getEast(Square): Square
+ getGrid(): Square[]
+ printMaze(): String
+ toString(): String
#maze
-currState
-start
-end
-grid
#maze
#closedSquares
Square
- maze: Maze
- l: Integer
- c: Integer
- attribute: String
- wall: Boolean
- g: Double
- h: Double
- f: Double
+ Square(Integer, Integer, String)
+ Square(Integer, Integer, Boolean)
+ assignMaze(): void
+ getLine(): Integer
+ getCol(): Integer
+ setLine(Integer)
+ setCol(Integer)
+ getAttribute(): String
+ setAttribute(String)
+ isWall(): Boolean
+ setWall()
+ getH(): Double
+ calcManhattanH(Square)
+ calcEuclidH(Square)
+ getG(): Double
+ incG(): void
+ calcF(): Double
+ getF(): Double
+ toString(): String
Solver
- maze: Maze
- result: String
- frontier: AbstractCollection<Node<Maze>>
- closedSquares: AbstractCollection<Square>
- nodesCounter: Integer
- pathLength: Integer
+ solve()
+ getNextSquares(Square): Square[]
+ getResult(): String
+ getFrontier(): LinkedList<Square>
#frontier
Node<T>
- value: T
- father: Node<T>
+ Node(T)
+ getContent(): T
+ setContent(T)
+ getFather(): Node<T>
+ setFather(Node<T>)
+ hasFather(): Boolean
+ printFather(): String
+ toString(): String
-father
AStarSolver
- manhattan: Boolean
+ AStarSolver(Maze)