German University in Cairo
Department of Computer Science
Assoc. Prof. Haythem O. Ismail
Ammar Yasser
Ahmed Hashem

Introduction to Artificial Intelligence, Winter Term 2021
Project 1
The Matrix: Escaped[1]

Due: November 29^{th}

1. Project Description

The year is 2200 and the machines took over the world after a long and grueling battle the likes of which were never seen before. Most of humanity was defeated and they got their memories erased and then put into deep sleep. In this state, their brains are used as a huge neural network by the machines. To harness the power of the humans' brains, the machines created a simulation called *the matrix* where humans who are asleep think that in fact they are alive in. After the war, a very few number of humans were able to escape the wrath of the machines and they started planning how to avenge their loss and save the human race. A prophecy was told that there will come someone from within the matrix who will be the chosen savior of humanity. This person is called *Neo*. The living were able to get to the matrix and they were able to locate Neo and get him finally awake. Now, Neo get in and out of the matrix when he can to save other humans.

The machines started noticing a weird pattern of behavior in the matrix and hence they created a virus called agent *Smith* that should track Neo and kill him whenever he sees him in the matrix. In order to do so, agent Smith made a lot of copies of himself and decided to set a trap for Neo. They (the copies of agent Smith) took some of the humans Neo would save as hostages and injected them with a slow spreading chemical that would eventually kill them and turn them into agents. If a human dies in the matrix their brain stops and they die in real life. Hence, Neo, on sensing that some humans are dying, rushed to the matrix in order to save the humans. The agents do not know that Neo has special skills and that he foresaw the agents' plan. As such, Neo planted a number of pills that would restore his health and the health of the hostages once activated and a number of launching pads that will let him fly from one to the other. Neo's goal is to take as many humans as possible alive to a special location called the *telephone booth* where they can get out of the matrix.

The area the hostages are held in can be thought of as an $m \times n$ grid of cells where $5 \leq m, n \leq 15$. Initially, a grid cell is either free or contains one of the following: Neo, a hostage, a pill, a pad, an agent, or the telephone booth. In this project, you will use *search* to help Neo complete the mission by finding all the hostages and bringing them back to the telephone booth. Neo can:

- Move in all four directions as long as there are no agents in the cell Neo is heading towards.
- Carry a hostage only if both of them are in the same cell.

[1] This project's theme is based on the Matrix move franchise.

- Drop a hostage at the telephone booth only if he is in the same cell where the telephone booth lies.
- kill an agent if Neo is at a neighboring cell.
- Take a pill to increase the health of Neo and all current hostages.
- Fly from one launching pad to the next.

Neo can only carry up to c hostages at a time. Accordingly, Neo might have to make multiple trips to the telephone booth to transport all the hostages. The hostages, who have been poisoned will continue to incur damage with every passing time step. A time step is the duration taken by Neo to complete one action. With every time step, the damage of any hostage increases by 2. If the damage of a hostage reaches 100, they die and turn into agents. Every time Neo kills an agent, Neo's damage increases by 20. Every time Neo takes a pill the damage of Neo and all living hostages decreases by 20 (but damage should not get below zero). If Neo's damage reaches 100 he dies and the game is over.

Using search you should formulate a plan that Neo can follow to complete the mission. An optimal plan is one where the deaths are at a minimum as a first condition. Given two plans with the same number of deaths, the more optimal plan is the one where the total number of agents killed is minimal. The following search strategies will be implemented and each will be used to help Neo:

a) Breadth-first search.

b) Depth-first search.

c) Iterative deepening search.

d) Uniform-cost search.

e) Greedy search with at least two heuristics.

f) A* search with at least two *admissible* heuristics.

Each of the aforementioned strategies should be tested and compared in terms of RAM usage, CPU utilization, and the number of search tree nodes expanded. **You must only use Java 8 to implement this project.**

Your implementation should have two main functions `genGrid` and `solve`:

- `genGrid()` generates a random grid. The dimensions of the grid, the starting position of Neo and the telephone booth, as well as the number and locations of the hostages, agents, pills, and pads are to be randomly generated. You need to make sure that the dimensions of the generated grid is between 5×5 and 15×15 and the number of generated hostages is between 5 and 10. For every hostage, a random starting damage between 1 and 99 should also be generated. Also, pads come in pairs. Meaning, pad $p1$ would be generated with another pad $p2$ where Neo can fly from $p1$ to $p2$ and vice versa. The number of hostages Neo can carry c should be randomly generated as well where $c \leq 4$. The number of pills should be the same as the number of hostages.
- `solve(String grid, String strategy, boolean visualize)` uses search to try to formulate a winning plan:
  - *grid* is a string representing the grid to perform the search on. This string should be in the following format:

`M,N; C; NeoX,NeoY; TelephoneX,TelehoneY;`
`AgentX`$_1$`,AgentY`$_1$`, ...,AgentX`$_k$`,AgentY`$_k$`;`
`PillX`$_1$`,PillY`$_1$`, ...,PillX`$_k$`,PillY`$_k$`;`
`StartPadX`$_1$`,StartPadY`$_1$`,FinishPadX`$_1$`,FinishPadY`$_1$`,...,`
`StartPadX`$_k$`,StartPadY`$_k$`,FinishPadX`$_k$`,FinishPadY`$_k$`;`
`HostageX`$_1$`,HostageY`$_1$`,HostageDamage`$_1$`, ...,HostageX`$_k$`,HostageY`$_k$`,HostageDamage`$_k$

where:

* `M` and `N` represent the width and height of the grid respectively.
* `C` is the maximum number of members Neo can carry at a time.
* `NeoX` and `NeoY` represent the x and y starting positions of Neo.
* `TelephoneX` and `TelephoneY` represent the x and y positions of the telephone booth.
* `AgentX`$_i$`,AgentY`$_i$ represent the x and y position of agent i where $1 \leq i \leq k$ and k is the total number of agents.
* `PillX`$_i$`,PillY`$_i$ represent the x and y position of pill i where $1 \leq i \leq k$ and k is the total number of pills.
* `StartPadX`$_i$`,StartPadY`$_i$ represent the x and y position of pad i where $1 \leq i \leq k$ and k is the total number of pads. Moreover `FinishPadX`$_i$`,FinishPadY`$_i$ represent the x and y position of the target pad stated by `StartPadX`$_i$ and `StartPadY`$_i$. For example, if `StartPadX` $= 1$, `StartPadY` $= 2$, `FinishPadX` $= 3$, and `FinishPadY` $= 4$, this means that Neo can fly directly from cell $(1, 2)$ to cell $(3, 4)$. Further, if $1, 2, 3, 4$ is in the string, then the string must also contain $3, 4, 1, 2$. That is, Neo could fly from cell $(3, 4)$ to cell $(1, 2)$ instantly.
* `HostageX`$_i$`,HostageY`$_i$`,HostageDamage`$_i$ represent the x and y position and current damage of hostage i where $1 \leq i \leq k$ and k is the total number of hostages.

Note that the string representing the grid does not contain any spaces or new lines. It just formatted this way above to make it more readable. All x and y positions are assuming 0-indexing.

– *strategy* is a symbol indicating the search strategy to be applied:
  * `BF` for breadth-first search,
  * `DF` for depth-first search,
  * `ID` for iterative deepening search,
  * `UC` for uniform cost search,
  * `GR`i for greedy search, with $i \in \{1, 2\}$ distinguishing the two heuristics, and
  * `AS`i for A* search, with $i \in \{1, 2\}$ distinguishing the two heuristics.
– *visualize* is a boolean parameter which, when set to `true`, results in your program's side-effecting a visual presentation of the grid as it undergoes the different steps of the discovered solution (if one was discovered).

The function returns a `String` of the following format: `plan;deaths;kills;nodes` where

– `plan` is a string representing the operators Neo needs to follow separated by commas. The possible operator names are: `up, down, left, right, carry`, `drop`, `takePill`, `kill`, and `fly`.
– `deaths` is a number representing the number of dead hostages in the found goal state.

- `kills` is a number representing the number of killed agents in the found goal state.
- `nodes` is the number of nodes chosen for expansion during the search.

2. Sample Input/Output:

Example Input Grid: 5,5**;**2**;**0,4**;**1,4**;**0,1,1,1,2,1,3,1,3,3,3,4**;**1,0,2,4**;**0,3,4,3,4,3,0,3**;** 0,0,30,3,0,80,4,4,80

H (30)	A		Pad (4,3)	Neo
P	A			TB
	A			P
H (80)	A		A	A
			Pad (0,3)	H(80)

TB represents the telephone booth, H (X) represents a hostage with initial damage X, Pad (X,Y) represents a pad that lets Neo fly towards cell (X,Y), A represents an agent and P represents a pill.

Example Output:
left,fly,right,carry,left,fly,down,right,drop,left,left,kill,left,left
up,carry,down,right,right,right,right,drop**;**1**;**1**;**1165836

3. Groups: You may work in groups of *at most* four.

4. Deliverables

a) Source code.

- You should implement a data type for a search-tree node as presented in class.
- You should implement an abstract data type for a generic search problem, as presented in class.
- You should implement the generic search procedure presented in class, taking a problem and a search strategy as inputs. *You should make sure that your implementation of search does not allow repeated states and that all your search strategies terminate as per the time constraints of the automated public test cases that will be posted.*
- You should implement a `Matrix` subclass of the generic search problem. This class must contain `genGrid()` and `solve` as *static* methods.
- You should implement all of the search strategies indicated above together with the required heuristics. A trivial heuristic (e.g. $h(n) = 1$) is *not* acceptable. You should make sure that your heuristic function runs in maximum *polynomial* time in the size of the state representation.
- Your program will be subject to both public and private test cases.
- Your source doe should have two packages. A package called *tests* and a package called *code*. All your code should be located in the *code* package and the test cases (when posted) should be imported in the *tests* package.
- Part of the grade will be on how readable your code is. Use explanatory comments whenever possible.

b) Project Report, including the following.

- A discussion of your implementation of the search-tree node ADT.

- A discussion of your implementation of the Matrix problem.
- A description of the main functions you implemented.
- A discussion of how you implemented the various search algorithms.
- A discussion of the heuristic functions you employed and, in the case of A*, an argument for their admissibility.
- At least two running examples from your implementation.
- A comparison of the performance of the implemented search strategies on your running examples in terms of completeness, optimality, RAM usage, CPU utilization, and the number of expanded nodes. You should comment on the differences in the RAM usage, CPU utilization, and the number of expanded nodes between the implemented search strategies.
- Proper citation of any sources you might have consulted in the course of completing the project. *Under no condition*, you may use on-line code as part of your implementation.
- If your program does not run, your report should include a discussion of what you think the problem is and any suggestions you might have for solving it.

5. Important Dates

Teams. Make sure you submit your team members' details by October 30^{th} at 23:59 using the following link `https://forms.gle/tm72z5BeSLQN41dm8`. Only one team member should submit this for the whole team. After this deadline, we will be posting on the MET/CMS a team ID for each submitted team. You will be using this team ID for submission.

Source code and report. Online submission by November 29^{th} at 23:59. The submission details will be announced after the team submission deadline.

Brainstorming session. In tutorials.