MAS Project: Negotiating in Tile World

Andrei Olaru

1 The project

Project description: The goal of the project is the design and implementation of a multi-agent system formed of cognitive agents, that solves a variation of the tile world problem, in a distributed manner. Agents are self-interested and their goal is to accumulate as many *points* as possible before the system is stopped.

The problem: The environment in which the agents act is a grid environment in which there are *cells*, *obstacles*, and *holes*. Holes have a *depth*, which is measured as a positive integer. Cells may contain any number of *tiles*, which are objects with the property that a tile can reduce the depth of a hole.

Agents are robots that are able to carry tiles and use them to cover (decrement the depth of) holes. Agents can move through the grid only through cells (not through obstacles or non-null-depth holes). Multiple agents can pass through the same cell simultaneously, even when carrying tiles. An agent can carry one tile at a time. Tiles can be used to cover a hole only when the agent carrying the tile is in a cell adjacent to the hole (one common edge), and if the hole did not reach null depth.

Agents, tiles, and holes have *colors*. There are as many colors as there are agents. While any tiles can be used to reduce the depth of any hole, points will be generated only when a tile of the same color as the hole is used. Moreover, the points will go to the agent of that respective color, regardless of the agent that actually covered the hole. Using a tile to cover a hole of the same color generates 10 points. Using a tile to cover a hole of the same color with the current depth of 1 (effectively covering the hole completely) generates an additional 40 points (50 points in total).

[**Bonus**] The system is dynamic. Initially there are no tiles or holes in the grid. At various moments of time (according to the specifications), tiles or holes may appear. When they do, they are assigned a period of time after which they will expire (disappear) (a tile being carried by an agent will vaporize when it expires).

System structure and specifications: The environment is a grid of size $W \times H$. Positions are 0-based. There are N agents in the system. Each agent i has an initial position (x_i, y_i) and a color c_i.

Each agent executes in its own separate thread. Agents execute in parallel. All operations that are related to the environment are sent to the environment and the environment sends confirmations to agents that the operation has been completed (or not).

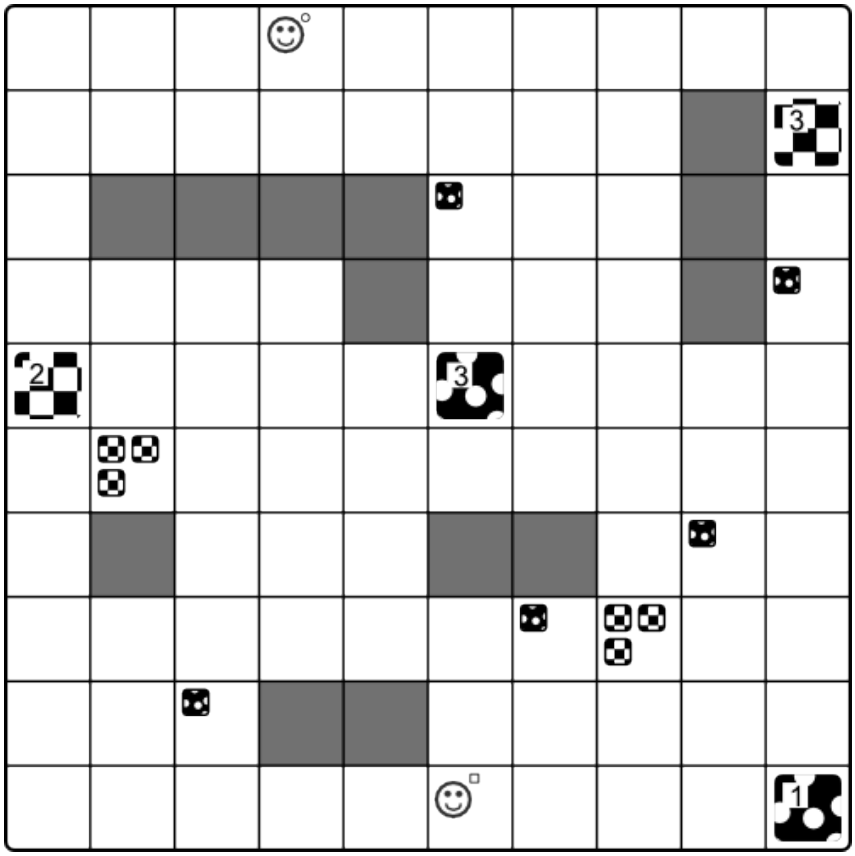

Figure 1: Example grid. There are two agents ("colored" as "circle-concerned" and "square-concerned"). For each "color" there are two holes with their respective depths and several tiles spread across the grid. This representation should not be taken as template for the output.

The environment itself runs in a different thread, which also handles the user interface. The environment needs to have a sort of message box in which agents are able to post operations. The environment will execute operations in the order in which they were received. The execution of an operation takes t milliseconds. After completion, the agent is informed of the success of the operation. In case of operations that cannot be completed, the agent is informed immediately of the failure.

The environment needs to keep the state of the whole system, comprised of the information that changes: positions and number of points of agents; color and *[bonus]* expiration of tiles carried by agents; position, color and *[bonus]* expiration of tiles in cells; position, depth, color and *[bonus]* expiration of holes.

All points are managed by the environment.

The operations available to agents (that need to be sent to the environment) are the following:

- $Pick(tile\_color)$ – picks a tile of the specified color from the current cell;
- $Drop\_tile()$ – drops the currently carried tile in the current cell;
- $Move(direction)$ – with direction in $[North, South, East, West]$;
- $Use\_tile(direction)$ – use the currently carried tile to cover the hole adjacent to the

current cell, in the specified direction;

- $Transfer\_points(agent, points)$ – transfers the specified (integer) number of points to the specified agent.

An agent is able to request from the environment the entire state of the environment (obstacles, tiles, holes, positions, points and colors of the other agents) at any time, but not more than once between two operations.

The system will run for a total time of T milliseconds, after which the system will stop and the number of points of the agents will be displayed.

Any agent is able to communicate with any other agent. Agents should be able to negotiate the realization of tasks by other agents, optionally in exchange for points. Tasks may be any sort of operation (e.g. moving a tile between any required coordinates). Feel free to use what you have learned at the lectures and apply it for this problem.

Design choices: There are a number of decisions that you need to take in the project:

- Specification of the communication primitives and protocols between agents and between agents and the environment.
- Internal representation of the environment and synchronization mechanism that allow the correct parallel access to environment state and the correct management of requests.
- Mechanisms of negotiation between agents that allow agents to request the realization of certain tasks by other agents.
- Visual representation of the environment (in text-mode or graphic mode) that allows the display of the cells, tiles (plus color), obstacles, holes (color and depth) and agents (color, points and color of carried tile), *[bonus]* plus expiration times.

Main concern: Remember that the main focus of this project is the **negotiation** between agents so that agents will may collaborate to obtain a greater number of points. Projects with no negotiation aspect will be rated very poorly. It is your choice how negotiation will work and how agents will communicate. The quality and performance of the negotiation protocol and strategies will count in the score for this assignment.

2 Input / Output / Execution

Your program will be executed on a single machine; however the execution of agents must happen in **parallel**: each agent must have its own execution thread. In order to start the program / MAS, a single command must be sufficient (this command may be the execution of a `.bat` / `.sh` / etc file), with one parameter (see next paragraph).

The format for the **input** of your program is mandatory. There will be a single input file, which will be given as first parameter of your program. The program must also support zero arguments, in which case the input will be read from the file **system.txt** . The format of this file will be the following:

```
N t T W H

color1 color2 color3 ... colorN
x1 y1 x2 y2 x3 y3 ... xN yN

OBSTACLES xo1 yo1 xo2 yo2 xo3 yo3 ...

TILES
Nt1 colorT1 xt1 yt1
Nt2 colorT2 xt2 yt2
...

HOLES
D1 colorH1 xh1 yh1
D2 colorH2 xh2 yh2
...
```

Important note: any elements in the input file may be separated by any amount of various whitespace characters (space, tab, \n, \r, etc.).

In words: the input file contains the following elements, in order:

- The number of agents (N);
- Time t that it takes to perform an operation on the environment;
- Total time T of the simulation;
- Width and height of the grid (W, H);
- N colors (strings) – the colors of the agents;
- $2 * N$ zero-based integers – the initial positions of the agents;
- The keyword `OBSTACLES`, followed by pairs of coordinates for obstacles;
- *[bonus]* The keyword `GENERATOR`, followed by 4 numbers: minimum and maximum value of the interval between the generation of two elements (groups of tiles or holes), and minimum and maximum value of the lifetime (time to the expiration time) of a generated element. Coordinates, color, and depth / number of tiles of the generated element will be generated randomly. Time to the generation of the

next element and lifetime will be generated randomly between the specified limits. Note: if after the obstacle list the `TILES` keyword follows instead of the `GENERATOR` keyword, then the program must work in "normal mode", not in "bonus mode".

- *[not in bonus]* The keyword `TILES`, followed by groups of 4 elements, characterizing groups of tiles in the grid: the number of tiles in the group, the color of the tiles in the group, and the coordinates of the cell in which the group is located. Note: there may be more groups of tiles in the same cell, if the groups have different colors;
- *[not in bonus]* The keyword `HOLES`, followed by groups of 4 elements, characterizing the holes in the grid: depth, color and coordinates of the hole.

The **output** of the application must have two elements:

The representation of the grid – either textual or graphical, comprising the display of the cells, tiles (plus color), obstacles, holes (color and depth) and agents (color, points and color of carried tile), *[bonus]* plus expiration times. In case of text mode, the grid should be displayed in the system console every *td* milliseconds, with td a constant that is easy to change in the source code. For instance it could be 2000 milliseconds. The log of operations between two displays of the grid should be buffered and output immediately before displaying the grid. The display of the grid should be accompanied by a timestamp of the simulation, rounded to seconds.

The operation log – at the system console, has no mandatory format, but should contain one line for each operation requested on the environment; information about an operation should contain the timestamp (in seconds, rounded to milliseconds), the type of the operation, and the arguments of the operation; The operation should be displayed once it has been received, also indicating if the operation fails.

The operation log should also contain information about the negotiation of tasks between agents, including proposals, acceptance and rejection, etc. It may as well contain concise indications of the agents' internal planning.

An example input file and corresponding output is given in the last section.

3 Requirements

This project is a design **and** programming assignment.

The first part of the handout must contain a document explaining the main design decisions (see first section), the communication primitives and protocol specified in a formal way (interaction diagrams, semantics, etc), an explanation of the output, the main algorithms used in the system – especially for negotiation, the results obtained for different test data, and other issues you may find interesting. Pay attention to this first part as it will be scored in your final mark for the assignment!

The second part of the handout will be the program itself, written in a language of your choice.

Bonus points will be given for the implementation of the dynamic aspect, as specified across the document indicated by *</[bonus]>* tags. Note that the application must also work with a statically specified environment (see the input specification).

Use a clean and clear programming style and comment your code.

You have to upload only one file on the lectures page on cs.curs.pub.ro – a `.zip` archive containing the sources, the design document specified above, and your own test cases.

The name of the file **must** be `Project_MAS_LastNameFirstName.zip`.

Each student will have 7-10 minutes to present his/her work, including the questions from the teaching assistant. **The project will not be scored without the presentation!** The intervals for the presentation will be announced.

You must not copy code from books, web sites, colleagues. The solution must be yours!

4 Example

system.txt:

```
2 300 10000 4 4 blue green 3 0 0 3

OBSTACLES 1 1 2 1 2 2

TILES 2 blue 2 3
1 green 2 0    1 green 1 3

HOLES 2 green 0 0   2 blue 1 2
```

Output:

```
Time: 0s

      0    1    2    3
    +    +    +    +    +
0    # 2       $1Gr @Blu
     Gree
    +    +    +    +    +
1         //// ////
          //// ////
    +    +    +    +    +
2         # 2  ////
          Blue ////
    +    +    +    +    +
3    @Gre $1Gr $2Bl

    +    +    +    +    +

Green agent:  0 points; carries nothing
Blue agent:   0 points; carries nothing
==================================================

[0.001][ENV][Green] Move East
[0.001][ENV][Blue]  Move South
[0.002][NEG][Blue -1-> Green]
     Propose: Carry 1 Blue (2,3)->(1,2) for Transfer 9 points
[0.003][NEG][Green -1-> Blue]
     Counter: Carry 1 Blue (2,3)->(1,2) for Carry 1 Green (2,0)->(0,0)
[0.004][NEG][Blue -1-> Green]
     Accept
[0.301][ENV][Green] Operation complete
```

```
[0.301][ENV][Blue]  Operation complete
[0.302][ENV][Green] Move East
[0.302][ENV][Blue]  Move North
[0.602][ENV][Green] Operation complete
[0.602][ENV][Blue]  Operation complete
[0.603][ENV][Green] Pick Blue
[0.603][ENV][Blue]  Move West
[0.903][ENV][Green] Operation complete
[0.903][ENV][Blue]  Operation complete
[0.904][ENV][Green] Move West
[0.904][ENV][Blue]  Pick Green

Time: 1s

      0    1    2    3
    +    +    +    +    +
0    # 2       @Blu
     Gree
    +    +    +    +    +
1         //// ////
          //// ////
    +    +    +    +    +
2         # 2  ////
          Blue ////
    +    +    +    +    +
3         $1Gr $1Bl
          @Gre
    +    +    +    +    +

Green agent:  0 points; carries Blue
Blue agent:   0 points; carries Green
================================================

...
```

Notes:

- the format of the output is only an indication. If you feel like it, you can have larger cells (tests will contain a maximum of 10×10 cells).
- the example above contains exchange of tasks (which can be viewed as task reallocation or as barter). It is not mandatory to implement that kind of exchange, you can have only negotiation in exchange for points.
- if agent *Green* would have accepted to carry out the task for 9 points, agent *Blue* would have had to give it 9 points although it currently had 0. However, it is OK to have negative numbers of points. So agent *Blue* would have -9 points after the exchange.