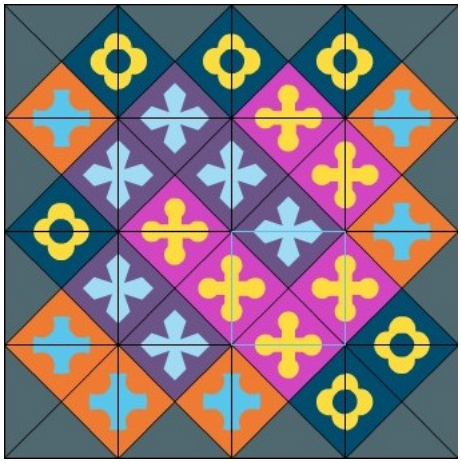

Figure 1: One example of a 4 by 4 Eternity like puzzle

INGI2261 : Artificial Intelligence
Final Project

Authors: Sébastien Mouthuy, Pierre Schaus, Yves Deville.

The final project is about solving games similar to Eternity II [1] using AI techniques. This is a very hard puzzle game. If you achieve to solve the original instance of Eternity II, you may win $2 millions! For copyright reasons, we won't work on this original instance, but on randomly generated instances. Of course you are free to try your agent on the original instance if you want.

In order to design and implement your AI program, you can use whathever techniques and programming language you want. You are completely free for this final project. You will receive a score depending on the quality of the solution found by your program and the originality of your design.

A hard copy of your report should be submitted to the secretary for Friday 19/12 at 10:30am. Before the same deadline, a pdf version of your report and a working code should be put on the svn server. No delay will be tolerated. A code that is not working automatically will lead to a nul score for the implementation part. An unclear report will also lead to a lower score. Your report can be submitted either in english or french.

1 Problem Description

This problem is about placing 256 squared pieces on a 16 by 16 grid. Each edge of the pieces are coloured. The borders of the grid are colored in gray. You must place the pieces on the grid so that the two touching edges of any two adjacent pieces (or with the border) have the same color.

One example is illustrated in Figure 1.

[1] `www.eternityii.com`, `en.wikipedia.org/wiki/Eternity_II_Puzzle`

Description of the instances The squared pieces are enumerated from 1 to 256. We provide you a function $color(p, r)$ that gives the colors of the r^{th} edge of piece p, so $p : 1 \leq p \leq 256$ and $r : 0 \leq r \leq 3$. Colors are considered as integers. 0 is the color of the borders of the grid (gray).

Description of the solution A solution to the puzzle is a pair of two-dimensional arrays (S, R). $S_{i,j}$ gives the number of the piece put at position (i, j) on the grid ($(1, 1)$ being the bottom-left corner) and R_{ij} determines how many times the piece S_{ij} has been rotated by 90° **anti-clockwise**.

Such puzzle are very difficult, so finding a solution respecting all the constraints is very hard. We will give a score to a solution of the puzzle based on the number of touching edges that do not have the same color. More precisely

$$score(S,R) = \sum_{i=1}^{n}\sum_{j=1}^{n}\begin{pmatrix} (match(color(S_{ij},R_{ij}),color(S_{i-1,j},R_{i-1,j}+2))) \\ +(match(color(S_{ij},R_{ij}+1),color(S_{i,j+1},R_{i,j+1}+3))) \\ +(match(color(S_{ij},R_{ij}+2),color(S_{i+1,j},R_{i+1,j}))) \\ +(match(color(S_{ij},R_{ij}+3),color(S_{i,j-1},R_{i,j-1}+1))) \end{pmatrix}$$

$$+\sum_{i=1}^{n}\begin{pmatrix} match(color(S_{1,i},R_{1,i}+3),0) \\ +match(color(S_{n,i},R_{n,i}+1),0) \\ +match(color(S_{i,1},R_{i,1}+2),0) \\ +match(color(S_{i,n},R_{i,n}),0) \end{pmatrix}$$

where n is the size of the grid and $match(c_1, c_2)$ is equal to 1 if $c_1 = c_2$ and 0 otherwise. This functions evaluates the number of times that two adjacent edges do not have th same color.

2 Design of an AI agent

You must design an AI agent that will solve such puzzles. You are completely free in the choice of the technique and programming paradigm used. Your main objective should be the efficiency of your agent. All agents will be run for the same number of minutes on a randomly generated instance and ranked depending on the score of the best solution they return.

Your score for the final project will depend on the rank of your agent wrt the other agents and on the originality of the design of your agent.

3 Format of the instances and of the solutions

The instances are text files defining the function *color*. The files have the following form

```
n n
1 color(1,0)  color(1,1) color(1,2) color(1,3)
2 color(2,0)  color(2,1) color(2,2) color(2,3)
...
n color(n,0)  color(n,1) color(n,2) color(n,3)
```

The first number is the size of the grid. Then the colors of all the pieces are described.

Your agent should print the best solutions it finds during its execution. It should print a solution in a file respecting the following format

```
score
S[1,1] R[1,1] S[1,2] R[1,2] ... S[1,n] R[1,n]
...
S[n,1] R[n,1] S[n,2] R[n,2] ... S[n,n] R[n,n]
```

where `score` is computed according to the function above. Your agent will be killed after the elapsed time, so write the solution to the file as soon as you find a better solution and make sure the content of the file is written.

A solution checker will be provided so that you can verify that your agent works and compute the score correctly.

4 Deliverables

You must return a report

1. describing the design of your agent: which main technique and programming language you used, your design choices, how it works, ...
2. motivating your design
3. describing the enhancements that you **would** perform on your agent to make it more efficient
4. listing precisely all the references your work is based on

The report should be as concise and precise as possible. We must be able to quickly understand what you did and why. So its length should be proprtional to the number of original ideas you implemented. Do not explain simple ideas by long texts. As we are aware of the AI course, do not explain the techniques seen during the course if you use them, only describe what is original in your design. The main objective of your report is to describe the work you did so we can judge the originality of your work.

The implementation of your agent is also very important as it will be used to partially score your work. Your code should respect **precisely** the following technical requirements.

1. Specify clearly the programming language you are using(version, libraries,...)
2. You should provide a script named `runEternity.sh` in the directory `grpN/milestoneM/`.
3. The call `runEternity.sh N instance.txt solution.txt` should run your agent on the puzzle describe in `instance.txt`. Your agent should print the solution it finds in `solution.txt` as soon as it finds some better one. `N` is the number of minutes after which your agent will be killed, so you can design a time-dependent strategy.
4. The agents will be run automatically, so check that the following execution works

```
cd //svn_server/INGI2261_TP2008/grpN/milestoneM/
./runEternity.sh N instance.txt solution.txt
```

5. The agents will be run on the computers of the Siemens room (named `volcano01,...`). So be sure everything is installed on these computers for your agent to work perfectly. Check by `ssh` whether everything is ok

6. Your agent may use the four processors available on these machines, so parallel computing is allowed. However you cannot use other computers (distributed computing is not allowed).

7. Keep in mind that a code that is not executable will lead to a nul score for the implementation part. We won't take time to debug technical problems.