Alghorithm for Map Color

MARIUS-CONSTANTIN POPESCU[1] LILIANA POPESCU[2] NIKOS MASTORAKIS[3]

[1]Faculty of Electromechanical and Environmental Engineering
University of Craiova
Decebal Bv, No.107, 200440, Craiova
[2] "Elena Cuza" College of Craiova
ROMANIA
[3]Military Institutes of University Education, Hellenic Naval Academy
GREECE
mrpopescu@em.ucv.ro lpopi2001@yahoo.com mastor@wses.org

Abstract: In this paper are followed the necessary steps for the realisation of the map's coloring, matter that stoud in the attention of many mathematicians for a long time. It is debated the matter of the four colors, but also the way of solving by implementing of an algorithm in the MAP-MAN application. Also, it is tackled the maps drawing in real time within GPS system satellites, using more colors depending on the covered route and the landforms met.

Key-Words: - The issue of the fourth colors, The MAP-MAN application, Software for GPS.

1 Introduction

The issue of the map's coloring with four colors it dates since 1852 when Francis Guthrie, while he intended to color the map of the England regions, he ascertain that four colors are enough. He asked his brother Frederick if is true that any map could be colored using just four colors, therefore the abutting regions (meaning those who has a segment of common edge, not just one point) to correspond diffrent colors. Frederick Guthrie communicate to DeMorgan this hypotesis. The first allusion at this issue was printed due to Cayley (1878).

A year later appered the first " demonstration" made by Kempe. The errors of this demonstration were punctated 11 years later by Heawood. Another demonstration who failed it is due to Tait (1880); an empty space left in this argumentation was dot by Petersen in 1891 [4], [6], [13]. Those failed demonstration didn't have any value even that Kempe discovered that was becoming known under the name of "the Kempe's handcuffs", and Tait discovered an equivalent wording of the *Four Colors Theorem* in the coloring terms of three edges [1], [2].

The next major contribution is brought by Birkhoff who's job allawed to Franklin in 1922 to ptove that the hypothesis of the for colors is true for maps with at leats 25 regions [5]. Also, she was used bz other mathematicians for improving *For Colors Issue*. It should mentioned aside Heesch who developed the two principal ingredients necessary to the last demonstration reducibility and discharge. Also, until the concept of the reductibility was studied by other researchers, it seems that the discharge idea, crucial for an unavoidable part of the demonstration, it is due to Heesch, he was the one who launched the hypothesis that a proprer development of this method would solve the Four Colors Theorem. The Heesch hypothesis was confirmed by Appel and de Haken in 1976, in that yesr they published their own demonstration of the Four Colors Theorem [3], [9], [10], [11], [12], [14].

2 The Matter of the Map Colouring with For Colours

The problem definition. Beeing given a map (more neighbouring areas among them), on demand to color the map with outside for diffrent colors, therefore any from that two neighbouring areas to be colored different [15].

The problem framing. The problem makes part from the category of the constraint problems of ("constraint – satisfaction problem", a.k.a. CSP)

Useful observation. The map can be seen like a graf in wich:
- every knot suits to an area on the map;
- every edge suits to a bond from two countries.

Fig. 1: The coloring of the Europe's map.

3 The Solution of the Coloring

The solution algorithm of the coloring problem of a map (intelligible part of the application) was realised using the CLIPS language [7].

The graphic interface of the application was developed using the orientation – C# object [8].

The coloring program of a map, realised in the CLIPS declarative language, is formed from two parts:

- the so called algorithm of coloring wich is made from rules (`rules`) – to be seen the „`mapping_engine.clp`",
- the data structures used by the algorithm wich are represented by fact (`facts`) – to be seen the „mapping_facts.clp"

The implementation in CLIPS is using three specific makers of:

- `deftemplate` – for the definition of the data structures (the definition of the fact template),
- `deffacts` – for the data definition (facts),
- `defrule` – for the rules definiton (the rules are executed actions over the facts if are observed some condition).

A rule in CLIPS it is made from two sides (sides):

- RHS (R*ight*-H*and* S*ide*) – specific for the so called *action* who is ussualy executed,
- LHS (L*eft*-H*and* S*ide*) – specific for the condition who must be fulfil (the moment) for executing the action RHS.

The MAP-MAN application is diffrent from the clasic case by the fact that it mustn't colored all the countries o the map. The graphic interface allows to the user to select the countries that will be colored by the algorithm. Follow-up, we will use the next terms:

- the active country – nominates a country wich the user select (by clicking on it) for beeing colored;
- the inactive country – nominates a country wich mustn't be colored (the user didn't clicked on it);
- the validate color – nominates a color associated to a country wich has a number smaller than 4 and who is differnt from the colors of the neighbouring countries (a color is a whole value between 0 and 3).

3.1 The Data Structures (templates, facts)

3.1.1 Templates

<`country`>: it is use for specify the fact's format (the structure) of the <`country`>, are the following fields:

<`code`> - the whole positive value, who specify the „ name" of thecountry;

<`color`> - the whole value (-1) and 3, who nominates the „color" associated to a country;

- an uncolored country has the color field is = -1 (initially, all the countries has the color is = -1);

<`flag`> - the whole positive value, between (-1) and the munber of the three active countries;

- for an inactive country we will have: `flag` = -1;

- used in the backing phase of the algorithm;
<was_first>
- can take two values: on/off;
- used in the moment when the algorithm gave back until the first country that he colored, for that the algorithm not choose again a country wich was already choose the first in the beggining (to avoid an infinit curl);
- initially, every country has was_first = -1;
- if a country was choosen the first (meaning that there aren't other active colored countries on the map), then she will have was_first=on (a complete name of this field would be <was_first_colored>) ;

< neighborhood> - a multislot field, used for specify the neighbors of a country.

3.1.2 Facts

- <no_of_active_countries> - specify the number of active countries;
- <flag_counter> - used in the backing phase of the algorithm;
- <next_country> - the country for wich we try to associate a validate country
- <next_color> -nominates the candidate's color for the country who has <code> equal with „next_country".

3.2 The Algorithm (rules)

The algorihtm uses the next nine rules:
Rule 1: <set_first_next_country>
Rule 2: <main_final>
Rule 3: <print_map_colors>
Rule 4: <main_loop_when_not_backing>
Rule 5: <main_loop_whwn_backing>
Rule 6: <color_is_valid>
Rule 7: <invalid_color>
Rule 8: <increment_color>
Rule 9: <color_is_valid>
Rule's details:

Rule 1: <set_first_next_country>
LHS (the condition / the ignition moment): doesn't exist other active colored country;
RHS (actions): - choose a country, from those with the field was_first= *off*, to be the first colored;
- set was_first = *on* for the chossen country;
- put on 0 the candidate color (next_color= 0) for this country.

Rule 2: <main_final>
LHS: after all the active countries were colored;
RHS: get the ignition of the display rule fo results.

Rule 3: <print_map_colors>
LHS: after the RHS execution from Rule 2;
RHS: result's displaing.

Rule 4: <main_loop_when_not_backing>
LHS: after on select another country that must be colored;
RHS: for the country with the field <code> equal with „next_country" put the field <color> on 0.

Rule 5: <main_loop_when_backing>
LHS: when we didn't find the country that it must be colored („next_country") a validate color (<4); this rule is not active if it isn't colored at least one country (in this case interfere the Rule 1);
RHS: add the associate color to the last olored country.

Rule 6: <color_is_valid>
LHS: when the candidate color („next_color") for the country that it must be colored („next_country") it is validate (meaning between 0 and 3 sand different from the color of any neighbour);
RHS: actualise the fields <color> and <flag> for the country „next_country"; also, actualise the fact <flag_counter> that it will be use in the backing phase of the algorithm (meaning that then when the algorithm recoil).

Rule 7: <invalid_color>
LHS: when for the country that it must be colored („next_country") the candidate color („next_color") it is not validate;
RHS: get the ignition of the Rule 8 (rather, creates a fact <inc_color> for this).

Rule 8: <increment_color>
LHS: after the RHS execution of the 7 rule (meaning after the creation of an abject <inc_color>);
RHS: add the candidate color („next_color") for the country that it must be colored at the current frequentative (meaning for „next_country").

Rule 9: <color_is_4>
LHS: when for the country that it must be colored („next_country") the candidate color („next_color") was added until it reach the 4 value (validate value; the validate values are 0, 1, 2, 3);
RHS: - actualise the fields <color> and <flag_counter> of the coutry „next_country" (color = -1; „flag=flag + 1"),
- actualise the field <flag_counter>
(„flag_counter=flag_counter -1"),
- create a fact (<back>) for the ignition of the 5 rule (<main_loop_when_backing>).

3.3 The Graphic Interface (C#)

The graphic interface of the application allows the

user to open a folder with the extension „.map". The folders with this extension are, actually, maps that respects a certain format (this format will be explained later). The pening of a amp using the menu is made in this way: `File->Open map.`

After the map was opened, the user will click (pushing on the left button of the mouse) on the countries that he wants to color by the algorithm .

To start to color the map, we will use the menu: `Compute->Start.`

Every country that the user clicked will initialy be colored orange. After the map's coloring comand, the countries on wich we clicked will be colored with red, yellow, green and blue [15]. The format description <.map>.

The MAP folders are text folders, having the next:

- Line 1: contains only ohne number that indicates
- the polygon number (countries) present on the map
- Line 2: contains 2 numbers representing the co ordinates X and Y of the map (these numbers are used at the map's scalaring);
- The next lines: on every line is discribed an polygon under the next form:
  - the first column: contains "his name" (meaning that a whole number; e.g.: 1, 2, 3, 4,… 7);
  - the second column: contains the point number from wich it is formed the polygon (it doesn't exist a relative limitation at the point's number of the polygon);
  - then it follows 2 columns representing the co-ordinates X şi Y of the polygon's points;
  - the next column states the neighbouring's number of the polygon discribed by the respective line;
  - the last columns from one line states the neighbours of the polygon described by the respective line.

Observations:

- Pay attention at the spaces and at the tabs: between any 2 elements of one line it is a space or a tab. If we don't respect this rule it is highly that the program not working correctly;
- The map it is made from polygons of any dimension. It is rather that this polygons to not be concave, because it can be problems at the polygon's detection where we clicked.

The folders of the `.map's` type allows the construction of a map very diffrent. The MAP-MAN program reads such kind of folders and on the basis of them constructs the graphic maps (we can say that the `.map's` folder contains maps in text format, wich MAP-MAN converts them in graphic format).

4 Code Listing

The Code listing supposes the following stages:

4.1 The template definiton <country> (the <mapping_engine.clp> folder).

4.2 The facts definition that discribes the map (this side of the CLIPS program differs from a map to another; further, it is presented an exemple).

4.3 The rules definition (the so called algorithm).

```
; Rule 1 (defrule set_first_next_country;
 Lights up when we don't have any active colored country
  ; (declare (salience -2));(flag_counter 0)
  ; if(all the*active* countries are *uncolored*)
  ?i<-(country(code?next_country)(color-1)(flag?flag)
  (wasfirst off)) (test(<=0?flag));if(flag>=0),
  Meaning that the country is *active*=>
  (modify?i (color 0)(wasfirst on))
  (assert (next_country ?next_country))
  (assert (next_color 0)))

; Rule 2 (defrule main_final
  ; (no_of_active_countries ?noac)
   (flag_counter ?fc) (test (eq ?fc ?noac))  =>
   (assert (print on)))

; Rule 3 (defrule print_map_colors
; (print on) (country (code ?code) (color ?c) ) =>
   (printout t "Country" ?code "has the color" ?c crlf))

; Rule 4 (defrule main_loop_when_not_backing
  ; (not (exists (back ?))) ;
  (flag_counter ?flag_counter) (test (<0?flag_counter));
We exclude the case when we don't have any colored
active country
?ic<-(country (code?next_country)(color -1)(flag ?flag))
    (test (<= 0?flag)); if(flag>=0),
Meaning that the country is *active*=>
    (assert (next_country?next_country))
    (assert (next_color 0)) (modify ?ic (color 0)))

; Rule 5 (defrule main_loop_when_backing
; ?ib<-(back on)
  (flag_counter ?flag_counter) (test (< 0?flag_counter));
We exclude the case when we don't have any colored
active country
?ic<-(country(code?next_country)(color?c)(flag?flag) )
   (test(<= 0 ?c))
   (test(and(<= 0?flag)(=?flag (+1?flag_counter))));
  if(0<=?flag = ?flag_counter - 1) =>
   (assert (next_country ?next_country))
   (assert (next_color (+1?c)))( modify ?ic (color (+1?c)));
   (retract ?ib))

; Rule 6 (defrule color_is_valid
  ; (declare (salience 7));
   ?in<- (next_country?next_country)
   ?ic<-(country(code?next_country)
   (color?candidate_color));implicit(flag<=0)din main_loop
  ( test(< ?candidate_color 4));
```

```
 (forall(country(code?next_country)
 (neighborhood$??nbour $?))
  (country(code ?nbour)(color ?color))
  (test(neq ?color ?candidate_color)))
  ?if<-(flag_counter ?flag_counter)
   ?inc<- (next_color ?)=>(retract?in)
 (modify?ic(color?candidate_color)(flag(+?flag_counter 1)))
    (retract ?if) (assert (flag_counter (+?flag_counter 1)))
    (retract ?inc))

; Rule 7 (defrule invalid_color;
; (declare (salience 2));
(exists(next_country ?next_country))
(exists (next_color ?next_color))
(exists(country(code?next_country)(color?candidate_color)));
(flag 0)))
(exists(country(code?next_country)
(neighborhood$??nbour $?)))
(exists(country (code ?nbour)(color ?candidate_color)))
  => (assert (inc_color on)))

; Rule 8 (defrule increment_color;
 Is adding the color ( of a country) if it is < 4
  ; (declare (salience 3)) ;
  ?i_inc <- (inc_color on)
   ?i_next_color<-(next_color ?next_color)
   (next_country?next_country)
   ?ic<(country(code?next_country)
    (color?candidate_color));(flag 0) )
    (test (<?candidate_color 4)) => (retract ?i_inc)
    (retract ?i_next_color)
    (assert (next_color (+ ?next_color  1)))
    (modify?ic (color  (+ ?candidate_color  1))))

; Rule 9 (defrule color_is_4;
Lights up when we didn't find an validate color for a country
(<4)
    ; (declare (salience 7));
    ?in <- (next_country  ?next_country)
    ?if <- (flag_counter  ?flag_counter)
    ?ic<- (country (code ?next_country) (color 4) ); (flag 0))
     ?inc <- (next_color ?next_color) =>; backing...
     (retract ?in)  ; 1) delete next_country
     (modify ?ic (color -1) (flag (+ 1 ?flag_counter) ));
2) restore country
        (retract ?if); 3) decrement flag_counter
        (assert  (flag_counter (- ?flag_counter 1)) )
        (retract ?inc); 4) (assert  (back on)))
```

Fig. 2: Code listing.

5 Snapshots

Further, on presents the results wich we obtain by the execution of the coloring algorithm for the map discribed by the `harta_wilensky.map.` folder

5.1 Map-Man after the folder opening `.map` (Fig. 3).

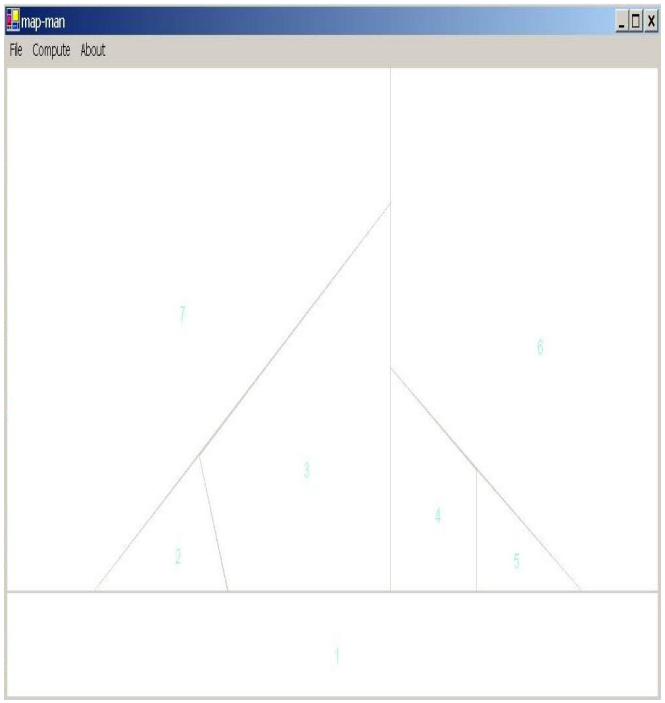

Fig. 3: The map after the folder opening.

5.2 Map-Man after the user clicked on some (Fig. 4), countries (in this case, on all of them).

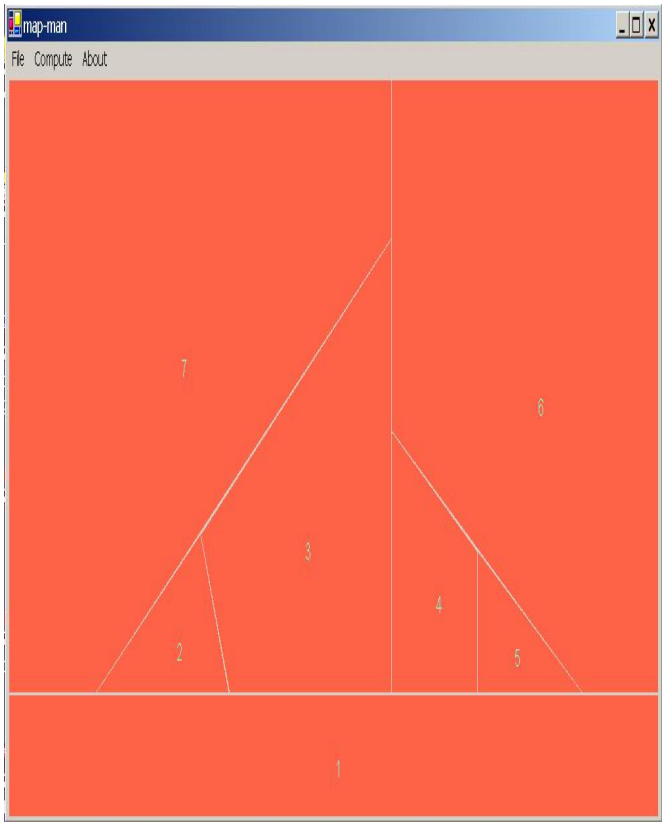

Fig. 4: The map after the user clicked on some countries.

5.3 Map-Man after the algorithm execution of the map coloring (Fig. 5).

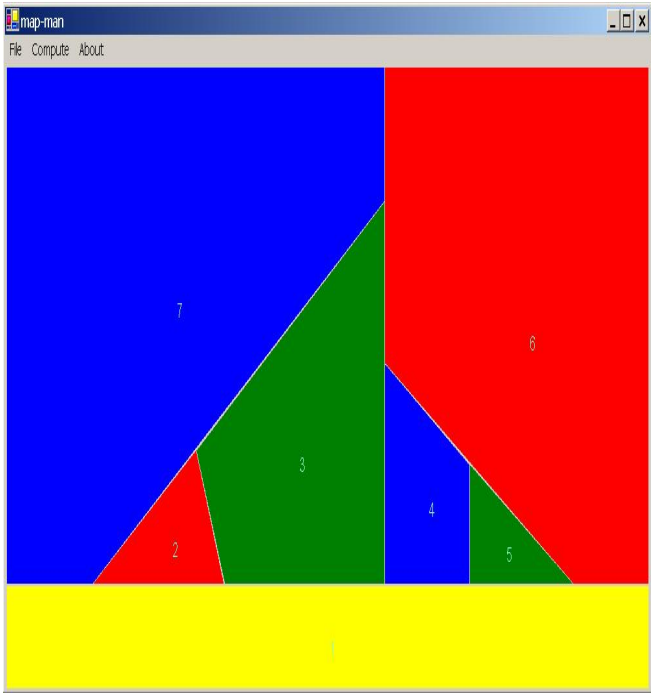

Fig. 5: The map after the algorithm execution of the map coloring.

5.4 The folder's listing `harta_wilensky.map` (Fig. 6).

```
7
680 420
1 4  0 420 0 351  680 351 680 420 6 2 3 4 5 6 7
7 5  0 350 0 0 400 0 400 90 90 350 4 1 2 3 6
6 5  401 0 680 0  680 350 601 350 401 200 51 3 4 5 7
2 3  91 350 200 259 230 350 3 1 3 7
3 4  201 259 400 91 400 350 231 350 5 1 2 4 6 7
4 4  401 350 401 201 490 269 490 350 4 1 3 5 6
5 3  491 350 491 269 600 350 3 1 4 6
```

Fig. 6: The folder's listing.

6 Romania and Greece Map

Formulating theme. The objective of this section is to implement a program addressing the problem of satisfying constraints. The chosen theme is to color a map of Romania and Greece with four colors so that the side by side surfaces to be colored differently.

How to solve. Solving the problem consists in using two programs:
- The "smart" is implemented in Clips;
- Visual interface is implemented in a high-level procedural language, Visual C#.

Communication between these programs is achieved through two dll files: *Clips.dll* and *ClipsWrapper.dll*

Detailed description. The program has in its three modules made in clips (*main.clp*, *search.clp* and *map.clp*) modules that perform the actual problem and find the solution.

- *main.clp* - contains three *deftemplates*:

1) *pb*-describe the current status of the problem - has three slots: - *state* (describe the current status of the problem and can have six values: start, in-progress, new-depth, choice-var, solved, blocked), *level* (mean the level of recursivity) and *hypothetical-var* (hypothetical variable is chosen to the current search);
2) *var*-describes each variable of the problem, has four slots: *name* (the name of the variable), *level* (clamp level), *possible-values* (the list of possible values attached to a variable) and *value* (variable value -> color of the city);
3) *solution* -used to the counting of solutions and their saving in variable - value (var-val) - contains 5 rules: *all-solutions?* (start search solution), *analysis* (search in progress), *save-solution* (saving solution in the form - var-val), *print-solution* (display solutions)

- *search.clp* - contains six rules:

1) *generate-new-depth* (generates a new level search);
2) *choice-var-val* (choose a hypothetical value for new level of search.It is chosen the variable with the fewest possible remaining values and is chosen as a possible value the first in the list);
3) *instanciation* (produce an analysis if there are any unused variables and the list of possible values only have one value, it is assigned);
4) *impasse* – is blocking the algorithm when the solution that we follow is a wrong one;
5) *backtrack-on-value* (choose using backtracking algorithm a new value for the current variable);
6) *backtrack-on-variable* (choose using backtracking algorithm, a new value from the previous step for the current variable);

- m*ap.clp* - contains the actual data of the problem (possible colors for each city, each city neighborhood and the test rule that two neighboring cities do not have the same color).

Results obtained by running the program in different situations (Fig. 7).

Table 1: Run with 4 possible values for coloring.

City	color 1st solution	color 2nd solution	color 3rd solution
0	Yellow	Blue	Cyan
1	Blue	Yellow	Yellow
2	Magenta	Magenta	Magenta
3	Cyan	Cyan	Yellow
4	Cyan	Cyan	Cyan
5	Blue	Blue	Blue
6	Blue	Blue	Blue

7	Yellow	Yellow	Yellow
8	Cyan	Cyan	Cyan
9	Cyan	Cyan	Cyan
10	Magenta	Magenta	Magenta
11	Blue	Blue	Blue
12	Cyan	Cyan	Cyan
13	Magenta	Magenta	Magenta
14	Magenta	Magenta	Magenta
15	Yellow	Yellow	Yellow
16	Yellow	Yellow	Yellow
17	Cyan	Cyan	Cyan

```
(focus PROPAG))
.....
(defrule SEARCH::backtrack-on-variable
;if problem's solve is blocked and there are no more possible-values for the hypothetical-variable, we must return to the previous search level where solving wasn't blocked and we choose another value from the possible-values of this level.
?s<- (search_level ?n)
?pb<-(pb(level?n)(blocked state|solved)(hypothetical-var ?x))
?pbbis<-(pb(levell?m&:(=?m(-?n1)))( blocked state))
?var<-(var(name?x)(level?n)(possible-values $?list&:(=(length$ $?list) 0)))
=> (retract?s)(retract?pb)(retract?var)
(modify?pbbis(blocked state)); the state of the previous level is blocked for the instanciation-var value to be changed)
map.clp
.....
(defrule PROPAG::diferent colors for neighbours
;if ?x has asociated the color ?v then we eliminate ?v from the possible-values list of ?y nodes neighboring with ?x

   (declare(salience 2))(logical(search_level?n))
   (not(search_level?n1&:(>?n1?n)))
   (var(name ?x)(value ?v&~nil)(level ?n))
   ?f <- (var (name ?y)(value nil) (level ?m)
   (possible-values $?lists&:(member$ ?v ?lists)))
   (not(var(name?y)(level?m1&:(>?m1?m))))
   (node(name?x))(node(name?y)
(neighbours$?neigh&:(member$?x$?vec)));

identify  ?x neighbours
=>(bind?poz_cul(member$?v?liste))
(if(=?m?n)then(modify?f(possible-values
(delete$?lists?poz_cul?poz_cul)))
else(duplicate?f(level?n)(possible-values
(delete$?lists?poz_cul?poz_cul)))))
.....
```

```
search.clp
.....
(defrule SEARCH::backtrack-on-value

;when state is "blocked" for the choosen hypothetical-value ?x we must choose another value from the possible values list
;we stay on the same level but we must delete all the facts reffering at the current-value of the hypothesis ?x;we are doing this thing through logical dependence

?s<-( search_level ?n)
?pb<-(pb(levell?n)(blocked state |solved)(hypothetical-var ?x))
?var<(var(name?x)(level?n)(value?v&~nil)(possible-values $?list&:(<>(length$ $?list)0)))
 => (retract?s)(assert(search-level ?n))
(modify?var(value(nth$1$?list))(possible-values(rest$ $?list))) (modify?pb(state in-progress))
```

Fig. 7: Listing important parts of the source code.

Table 2: Run with 5 possible values for coloring.

City	color 1st solution	color 2nd solution	color 3rd solution
0	Yellow	Yellow	Blue
1	Blue	Green	Yellow
2	Magenta	Magenta	Magenta
3	Cyan	Cyan	Cyan
4	Cyan	Cyan	Cyan
5	Blue	Blue	Blue
6	Blue	Blue	Blue
7	Yellow	Yellow	Yellow
8	Cyan	Cyan	Cyan
9	Cyan	Cyan	Cyan
10	Magenta	Magenta	Magenta
11	Blue	Blue	Blue
12	Cyan	Cyan	Cyan
13	Magenta	Magenta	Magenta
14	Magenta	Magenta	Magenta
15	Yellow	Yellow	Yellow
16	Yellow	Yellow	Yellow
17	Cyan	Cyan	Cyan

6.1 List of Main Classes for Impelemeting Map of Real - Time (GPS)

A. CTransfert Class - this class, common server and client, manages the transfer information (messages actions, tables of measures) between the server and the patient. His role will be to serialize the information in the form of a compressed frame that the client will decode. The main methods of this class *Class2Byte* and *Byte2Class*. The first serialized all of the information in the fields of the class (action, size and content of tables of measures,

display format, News ...) in a first stream, then this will compress flows.

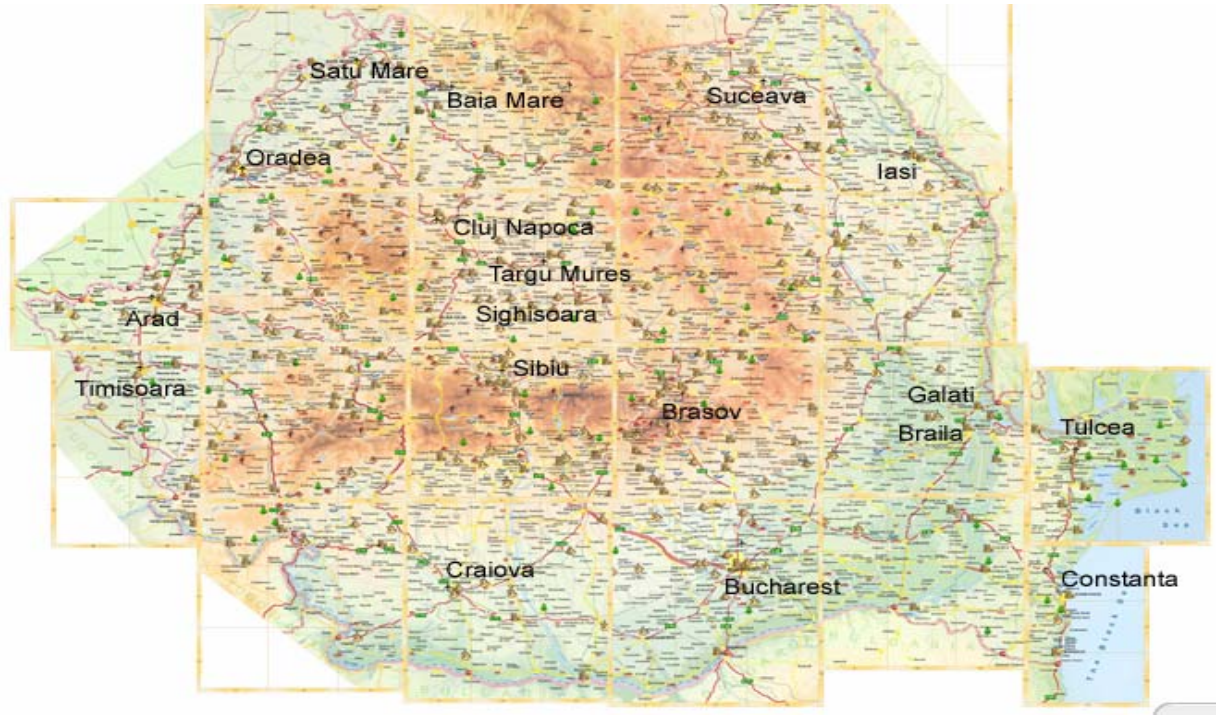

Fig. 8: The coloring of the Romania map: 4 culors and 5 coulors.

The *Byte2Class* method is the reverse of the previous one, which will decompress the flow specified before completing all fields in the class with the values in the frame. The table's measures, if any, will also be established so that the main class can access thereafter.

B. Server. Device Class. It realizes the interface between the main class and class transfer, managing the applications and by providing the class transfer relevant information to form the frame.

Fig. 9: The coloring of the Greece map: 4 culors and 5 coulors.

The main method of class, *TraiteDemande* responsible action to address sent by the client. It returns a feed store and its length, corresponding to the answer frame to be sent to the patient. If an acquisition is made of measures, the method fills an array of complex *Class CTransfert*. There are also actions settings current settings of the device, contained in the fields which will be sent to the device to match demand in real setting. Indeed, for certain input parameters, the user manually enters a value. According to the button pressed, our class will simulate the actual behavior of the device and must

accurately update the contents of current setting. Once the treated *TraiteDemande* launches flow generation, conducted in the classroom *CTransfert*, then returns the flow compressed the main class. *TmainForm Class* - class that declares the main menu of the application server, menu items and manages the events that occur in the form.

Be careful methods *AppSocketPatient-Connect, AdminSocket PatientConnect*, *AppSocketPatient-Read*, *Admin SocketPatientRead* which are methods of learning peeled event when receiving a connection from a client or receiving a string representing a client's message. It was decided to separate information about the connections, logins ... (Admin) messages of each device (App) into two sockets to facilitate management. At each new connection, the server verifies that it is a valid user with the password he compares to the list included in the file "serveur.ini." If it finds a match and if this login is not already connected, it sends a message to the opening of the connection. Otherwise, it returns a message of non-authorization and closes the socket connection. Each action received for the patient order, then we run the method of *TraiteDemande* of the Camera class: this action is decoded and the response to the motion is generated in the form of data stream. Once the action is treated on referral response sockets patient transmitter and receiver doctor. Another important method is the *ChangeEtatPatient* called in a double click on the login of a patient, and that changes the status of user choice and, if necessary, other users (because he can not have more than one user issuer at a time).

C. Patient. TMainForm Class - class that declares the main menu application main patient, menu items and manages the events that occur in the form.

7 Conclusions

MAP-MAN is an application wich propose the implementation of the coloring issue of a map, using a graphic interface. The application solves the two main problems – the reducibility and the discharge. If the four colors problem is a static one and it was solved through the aplication MAP-MAN, the problem of maps drawing in real time within GPS system satellites is a dynamic, real time and current one, and can be easyly solved by using oriented object programming as it has been shown in 6.1 paragraph.

References

[1] Appel K. and Haken W., *Every planar map is four colorable*. Part I. Discharging, Illinois J. Math. 21, pp.429-490, 1977.

[2] Appel K., Haken W. and Koch J., *Every planar map is four colorable.* Part II. Reducibility, Illinois J. Math. 21, pp.491-567, 1977.

[3] Appel K. and Haken W., *Every planar map is four colorable*, Contemporary Math. 98, 1989.

[4] Birkhoff G.D., *The reducibility of maps*, Amer. J. Math. 35, 1913, pp.114-128.

[5] Heesch H., *Untersuchungen zum Vierfarbenproblem*, Hochschulskriptum 810/a/b, Bibliographisches Institut, Mannheim 1969.

[6] Olaru O, Popescu M.C., Balas V. *A Study of Oscillation for Signal Stabilization of Nonlinear System*, Proceedings of the 10th WSEAS Int. Conf. on Automation & Information, pp.430-437, Prague, March 2009.

[9] Perescu L., Calina M.L., Popescu M.C., Calina A., *A Map-Coloring Application,* Proceedings of the 11th International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering, Published by WSEAS Press, pp.214-220, Vouliagmeni, Greece, September 2009.

[8] Popescu M.C., Olaru O., Mastorakis N., *Equilibrium Dynamic Systems Intelligence,* WSEAS Transactions on Information Science and Applications, Issue 5, Vol.6, pp.725-735, may 2009.

[9] Popescu M.C., Olaru O., Mastorakis N., *Processing Data for Colored Noise Using a Dynamic State Estimator* WSEAS Transactions on Communications, Issue 3, Vol.8, pp.321-330, March 2009.

[10] Popescu M.C., Olaru O. and Mastorakis N., *Equilibrium Dynamic Systems Intelligence*, International Journal of Mathematical Models and Methods in Applied Sciences, Issue 2, Vol.3, pp.133-142, 2009.

[11] Robertson N., Sanders D.P., Seymour P.D. and Thomas R., *A new proof of the four colour theorem*, Electron. Res. Announc. Amer. Math. Soc.2, pp.17-25 (electronic), 1996.

[12] Saaty T.L., *Thirteen colorful variations on Guthrie's four-color conjecture*, Amer. Math. Monthly 79, pp.2-43, 1972.

[13] Saaty T.L. and Kainen P.C., *The four-color problem.* Assaults and conquest, Dover Publications, New York, 1986.

[14] Tait P.G., *Note on a theorem in geometry of position*, Trans. Roy. Soc. Edinburgh 29, pp.657-660, 1880.

[15] Whitney H. and Tutte W.T., *Kempe chains and the four colour problem*, in Studies in Graph Theory, Part II, Ed. D.R. Fulkerson, Math. Assoc. of America, pp.378-413, 1975.