Rapport de projet: problèmes de logique

Vincent Munoz - Gaëtan Gousseaud

4 janvier 2015

Introduction et sommaire

Introduction

Voici le rapport du projet sur les problèmes de logique, il comprend l'avancée étape par étape du projet, la répartition des tâches, la liste des composantes du projet et des explications sur celui-ci.

Sommaire

Journal de bord

Etape 1

Etape 2

Etape 3

Etape 4

Etape 5

Etape 6

Etape 7

Etape 8

Etape final

Procédures mises en oeuvre

Répartition des tâches

Auto-évaluation

Mode d'emploi

Listing du programme

Les différents fichiers

Description des différentes fonctions

Chapitre 1

Journal de bord

1.1 Etape 1 (une journée et demi)

Dès le départ, nous avons eu l'idée d'énumérer toutes les possibilités de maisons et de les manipuler pour obtenir le résultat.

Mais comment matérialiser les maisons ? Nous avons choisi d'utiliser une structure comportant les différentes catégories (Nationalité, boisson ... plus une nouvelle, le rang de la maison (la maison numéro 0 est tout à gauche et la 4 tout à droite)). Une structure nommée "Quartier" comportant 5 maisons fut aussi créée. Pour faire la fonction qui affiche toutes les maisons possibles, nous avons donné un chiffre à chaque condition différente d'une même catégorie (exemples : rouge=0 bleu=2 et anglais=0).

Nous avons décidé de séparer les conditions définissant une maison grâce à une autre (ex : la maison bleue à côté du pécheur), des conditions définissant une maison elle-même (ex : Dans la maison verte, on boit du café). La liste des conditions devient donc un tableau à deux dimensions, chaque ligne étant une maison qui doit être valide. Chaque condition est complétée par des -9, correspondant à une catégorie non spécifiée. (ex : {0 (rouge),0 (anglais),-9,-9,-9,-9} pour la condition "L'Anglais habite la maison rouge"). Nous avons ensuite réflechi à la suite et avons retenu 2 idées de fonctions :

-Une fonction qui rempli un tableau (ou une structure) avec les maisons possibles.

-Une fonction qui fait défiler ce tableau et compare les maisons aux conditions.

1.2 Etape 2 (une journée)

Cette étape fut courte mais des plus embêtante. En effet, nous nous sommes lancé sur une longue réfléxion portant à s'assurer que nous utilisions les bonnes structures de données. Nous avons revu chacune d'elles et établi cela :

-Les maisons sont des structures.

-Le quartier est une structure.

-Les conditions deviennent une liste chaînée dont les noeuds sont composés de :

=> un numéro de condition.

=> un tableau correspondant à la condition (ex : {0,0,-9,-9,-9,-9}).

=> un pointeur sur la condition suivante.

Et nous avons réfléchi de nouveau aux 2 idées en suspens avec ces nouvelles structures de données.

1.3 Etape 3 (une journée et demi)

Dans cette étape nous essayons d'anticiper la modulation du projet. Dans cette optique, nous remodelons une nouvelles fois nos structures de données pour que nous puissions en changer les paramètres selon nos désires.

C'est ainsi que nos structures deviennent (ou redeviennent) des tableaux mais nous gardons la liste chaînée. Nous voyons aussi l'apparition de variables telles que le nombre de conditions (w) et le nombre de catégories d'une maison (nc). Nous pensons utiliser un tableau de taille variable (appelé panier) pour contenir les maisons vérifiées par varif_maison, fonction envisagée permettant de supprimer les maisons qui sont en contradiction avec les conditions.

1.4 Etape 4 (deux jours et demi)

C'est dans cette partie que nous avons codé les fonctions dont nous avions l'idée. Nous créons une variable indiquant le nombre de maisons dans le quartier (5 dans le problème actuel). A ce moment la notre tableau de conditions ressemble à cela :

int conditions[nc][w]={
0,0,-9,-9,-9,-9,//cond1
1,-9,1,-9,-9,-9,//cond2
...
};

On se prépare donc à faire la fonction remplir_maison qui va prendre toutes les maisons possibles une par une. Si la maison ne rentre pas en contradiction avec une des conditions ou qu'elle en vérifie une, on appelle une autre fonction qui met cette maison dans une structure (panier). Elle contient un tableau à deux dimension et une variable qui compte le nombre de maisons potables. Voici un bout de code de vérif_maison à ce moment :

if((cond[i][j]>=0) && (cond[i][j] !=maison[j]))
return 0;

i est le numéro de la condition dans le tableau 2D et j celui de la catégorie solicitée. Ces deux variables sont dans une boucle permettant de tester la maison avec toutes les catégories de toutes les conditions. Si une catégorie n'est pas -9 alors on peut regarder si elle correspond à celle de la maison. Nous faisons aussi une fonction qui affiche toutes les maisons du panier. En compilant, nous obtenons un joli segfault.

Résolution du SEGFAULT :

La valeur du malloc concernant le tableau du panier était 5 et non 6 (nombre de catégories), suite à cela nous avons remarqué que les deux valeurs nc et nm étaient inversées dans le prototype.

1.5 Etape 5 (une journée)

Après la résolution du segfault, nous avons une erreur sur vérif_maison que nous trouvons tout de suite :

Le bout de code écrit étape 4 est faux, car il sort à tout les coups (Si la condition est fausse on arrête tout. (ex : si la condition est maison rouge + anglais, si la maison testée n'est pas rouge alors on sort directement sans regarder si elle n'en vérifie pas une).

=>Problème : conditions non-concernées appliquées)

Problème résolu à l'aide de test et valid.

valid = 1 si la condition concerne la maison.
test = 0 Si la condition est fausse.
bout de code :

```
for(j=0,valid=0,test=1 ;j<nc ;j++){
if((cond[i][j]>=0) && (cond[i][j] !=maison[j]))
test=0 ;
if((cond[i][j]>=0) && (cond[i][j]==maison[j]))
valid=1 ;
}
```

Comme une condition porte toujours sur deux catégories, en avançant dans les catégories (j++), on peut différencier 3 cas de figure.

valid = 0 et test = 0, on a ici une condition qui ne conserne pas la maison (ex : cond = maison rouge + anglais, alors que maison = maison bleue + cuisinier). Il faut donc soumettre la maison à une autre condition.

Valid != test, une catégorie de la maison correspond à l'une de la condition, mais la deuxième est différente (ex : maisons = rouge + anglais et cond = bleue + anglais (ou rouge + cuisinier)).Il ne faut pas ajouter cette maison au panier.

valid = 1 et test = 1, les deux catégories de la condition correspondent à celles de la maison. Il faut donc mettre cette maison dans le panier.

Autre problème rencontré :

w = 14 alors que seulement 10 conditions concernent les maisons elles-mêmes (décidé à l'étape 1 mais toujours pas mis en place). w était un des paramettres du tableau de conditions[ ][w].

=>Les nombres de conditions internes et externes sont maintenant séparés (w et wq). Auparavant nous avions seulement mis de coté les conditions externes (ex : la maison verte est à coté de la blanche).

1.6 Etape 6 (une demi-journée)

Nous avons notre nombre de maisons potables, nous allons maintenant nous attaquer aux conditions externes (conditions de quartier, dont le nombre est wq (4 dans notre cas)). Pour cela nous faisons une fonction qui crée un tableau contenant le nombre de maisons de chaque rang (tab_rang).

Réflexion avancée sur verif_quartier :

-Idée de boucle de vérification.

-Idée du mouvement représenté dans les conditions de quartier (ex : 1 pour gauche, 0 pour droite et 2 pour voisin).

Une condition de quartier resemblerait à cela :

{info1 (catégorie validée), mouvement (Fonctions possibles : Gauche, droite et voisin),Info2 (Info à valider)}

Nous nous sommes aussi rendu compte que les maisons étaient automatiquement classées selon leurs rangs. Car la création de toutes les maisons possibles incrémente la dernière catégorie en dernier (ici, le rang), ce qui nous facilite le travail.

1.7 Etape 7 (3 jours)

Idée d'épuration et de boucle posées à forte reflexion. L'algorithme Verif quartier a été réalisé par Vincent. Commencement du code et découverte de ce que doit faire épuration, mais point d'idée sur la façon de la créer.

Epuration codée le lendemain par Vincent.

Nouvelle syntaxe pour condition quartier

=> {info1,mouvement,info2,rang de l'info1,rang de l'info2} (exemple : {0 (rouge),0 (à droite de),2 (medecin),1 (rang de la catégorie couleur),3 (metiers)})

La fonction vérif_quartier est finalisée le jour suivant.

1.8 Etape 8 (4 jours)

Nous avons terminé rérif_quartier et la fonction trace, qui trouve les maisons où il n'y a pas de -1. Nous avons compilé et après avoir résolu les érreurs, nous avons eu un segfault.

Gaëtan Gousseaud a trouvé que Mpotable (variable de la structure panier, qui compte le nombre de maisons possibles) n'était pas modifiée car épuration était un void donc le while était infini .

=> épuration retourne maintenant un panier.

Addition de reversecond, fonction qui prend une condition de quartier et en sort son reflet (Verte à droite de blanche devient blanche à gauche de verte).

1.9 Etape final (3 jours)

Notre programme tournait en boucle.

(trâce d'utilisation à ce moment-là (commentaires rajoutés après)

4 1 1 1 1 0 i==0
4 1 2 4 1 0 i==1
4 1 3 4 1 0 i==2
4 1 4 4 1 0 i==3
4 2 2 3 3 0 i==4
4 4 2 3 3 0 i==5 //cond check
4 2 3 3 3 0 i==6
4 4 3 3 3 0 i==7 //cond check
4 2 4 3 3 0 i==8
4 4 4 3 3 0 i==9 //cond check
4 2 1 1 4 0 i==10
4 4 1 1 4 0 i==11 //cond check
4 3 2 2 4 0 i==12
4 3 3 2 4 0 i==13
4 3 4 2 4 0 i==14
4 2 2 4 4 0 i==15
4 4 2 4 4 0 i==16 //cond check
4 2 3 4 4 0 i==17
4 4 3 4 4 0 i==18 //cond check
4 2 4 4 4 0 i==19
4 4 4 4 4 0 i==20 //cond check

3 2 1 1 0 1 i==21 //maison (si!=cond[2]) check
3 4 1 1 0 1 i==22
3 3 2 2 0 1 i==23 //maison (si!=cond[2]) check
3 3 3 2 0 1 i==24 //maison (si!=cond[2]) check
3 3 4 2 0 1 i==25 //maison (si!=cond[2]) check
3 2 2 4 0 1 i==26 //maison (si!=cond[2]) check
3 4 2 4 0 1 i==27
3 2 3 4 0 1 i==28 //maison (si!=cond[2]) check
3 4 3 4 0 1 i==29
3 2 4 4 0 1 i==30 //maison (si!=cond[2]) check
3 4 4 4 0 1 i==31
2 1 2 0 1 1 i==32 //maison (si!=cond[2]) check
2 1 3 0 1 1 i==33 //maison (si!=cond[2]) check
2 1 4 0 1 1 i==34 //maison (si!=cond[2]) check
1 1 0 4 1 1 i==35 //maison (si!=cond[2]) check
1 2 0 3 3 1 i==36 //maison (si!=cond[2]) check
1 4 0 3 3 1 i==37
0 0 2 3 3 1 i==38 //maison (si!=cond[2]) check
0 0 3 3 3 1 i==39 //maison (si!=cond[2]) check
0 0 4 3 3 1 i==40 //maison (si!=cond[2]) check
2 2 2 0 4 1 i==41 //maison (si!=cond[2]) check
2 4 2 0 4 1 i==42
2 2 3 0 4 1 i==43 //maison (si!=cond[2]) check
2 4 3 0 4 1 i==44
2 2 4 0 4 1 i==45 //maison (si!=cond[2]) check
2 4 4 0 4 1 i==46
0 0 1 1 4 1 i==47 //maison (si!=cond[2]) check
1 3 0 2 4 1 i==48 //maison (si!=cond[2]) check
1 2 0 4 4 1 i==49 //maison (si!=cond[2]) check
1 4 0 4 4 1 i==50
0 0 2 4 4 1 i==51 //maison (si!=cond[2]) check
0 0 3 4 4 1 i==52 //maison (si!=cond[2]) check
0 0 4 4 4 1 i==53 //maison (si!=cond[2]) check
2 2 2 0 2 2 i==54
2 4 2 0 2 2 i==55 //toutrang(==cond 0 || 2) check
2 2 3 0 2 2 i==56
2 4 3 0 2 2 i==57 //toutrang(==cond 0 || 2) check
2 2 4 0 2 2 i==58
2 4 4 0 2 2 i==59 //toutrang(==cond 0 || 2) check
0 0 1 1 2 2 i==60
1 3 0 2 2 2 i==61
1 2 0 4 2 2 i==62
1 4 0 4 2 2 i==63 //toutrang(==cond 0 || 2) check
0 0 2 4 2 2 i==64
0 0 3 4 2 2 i==65
0 0 4 4 2 2 i==66
3 2 1 1 0 3 i==67

3 4 1 1 0 3 i==68 //toutrang(==cond 0 || 2) check
3 3 2 2 0 3 i==69
3 3 3 2 0 3 i==70
3 3 4 2 0 3 i==71
3 2 2 4 0 3 i==72
3 4 2 4 0 3 i==73 //toutrang(==cond 0 || 2) check
3 2 3 4 0 3 i==74
3 4 3 4 0 3 i==75 //toutrang(==cond 0 || 2) check
3 2 4 4 0 3 i==76
3 4 4 4 0 3 i==77 //toutrang(==cond 0 || 2) check
2 1 2 0 1 3 i==78
2 1 3 0 1 3 i==79
2 1 4 0 1 3 i==80
1 1 0 4 1 3 i==81
1 2 0 3 3 3 i==82
1 4 0 3 3 3 i==83 //toutrang(==cond 0 || 2) check
0 0 2 3 3 3 i==84
0 0 3 3 3 3 i==85
0 0 4 3 3 3 i==86
2 2 2 0 4 3 i==87
2 4 2 0 4 3 i==88 //toutrang(==cond 0 || 2) check
2 2 3 0 4 3 i==89
2 4 3 0 4 3 i==90 //toutrang(==cond 0 || 2) check
2 2 4 0 4 3 i==91
2 4 4 0 4 3 i==92 //toutrang(==cond 0 || 2) check
0 0 1 1 4 3 i==93
1 3 0 2 4 3 i==94
1 2 0 4 4 3 i==95
1 4 0 4 4 3 i==96 //toutrang(==cond 0 || 2) check
0 0 2 4 4 3 i==97
0 0 3 4 4 3 i==98
0 0 4 4 4 3 i==99
3 2 1 1 0 4 i==100
3 4 1 1 0 4 i==101 //toutrang(==cond 0 || 2) check
3 3 2 2 0 4 i==102
3 3 3 2 0 4 i==103
3 3 4 2 0 4 i==104
3 2 2 4 0 4 i==105
3 4 2 4 0 4 i==106 //toutrang(==cond 0 || 2) check
3 2 3 4 0 4 i==107
3 4 3 4 0 4 i==108 //toutrang(==cond 0 || 2) check
3 2 4 4 0 4 i==109
3 4 4 4 0 4 i==110 //toutrang(==cond 0 || 2) check
2 1 2 0 1 4 i==111
2 1 3 0 1 4 i==112
2 1 4 0 1 4 i==113
1 1 0 4 1 4 i==114

1 2 0 3 3 4 i==115
1 4 0 3 3 4 i==116 //toutrang(==cond 0 || 2) check
0 0 2 3 3 4 i==117
0 0 3 3 3 4 i==118
0 0 4 3 3 4 i==119
2 2 2 0 4 4 i==120
2 4 2 0 4 4 i==121 //toutrang(==cond 0 || 2) check
2 2 3 0 4 4 i==122
2 4 3 0 4 4 i==123 //toutrang(==cond 0 || 2) check
2 2 4 0 4 4 i==124
2 4 4 0 4 4 i==125 //toutrang(==cond 0 || 2) check
0 0 1 1 4 4 i==126
1 3 0 2 4 4 i==127
1 2 0 4 4 4 i==128
1 4 0 4 4 4 i==129 //toutrang(==cond 0 || 2) check
0 0 2 4 4 4 i==130
0 0 3 4 4 4 i==131
0 0 4 4 4 4 i==132

En refléchissant un peu nous nous sommes rendu compte que lorsqu'on est voisin de la première maison, on est forcément à sa droite. Des réctifications sur la fonction épuration se sont imposées, dont la création des cas spéciaux d'être voisin de la première maison ou de la dernière. Après quelques modifications mineurs, notre programme tourne en boucle, après avoir correctement effectué l'épuration de la première condition de quartier. Nous nous rendons compte à ce moment de quelques exceptions que nous devons gérer. Si la condition est verte à droite de blanc, alors :

- Si on est premier on ne peut pas être à droite de quelque chose.
- Si on est dernier on ne peut pas être à gauche de quelque chose.
- Si à notre droite il n'y a pas de maison verte, on ne peut pas être une maison blanche.
- Si à notre gauche il n'y a pas de maison blanche, on ne peut pas être une maison verte.

Nous sommes à 33 maisons potables avant de rajouter la prochaine exeptions :

4 1 1 1 1 0 i==0 //Impossible à droite Check
4 1 2 4 1 0 i==1 //Impossible à droite Check
4 1 3 4 1 0 i==2 //Impossible à droite Check
4 1 4 4 1 0 i==3 //Impossible à droite Check
4 2 2 3 3 0 i==4 //pasblanc d check
4 4 2 3 3 0 i==5 //cond check
4 2 3 3 3 0 i==6 //pasblanc d check
4 4 3 3 3 0 i==7 //cond check
4 2 4 3 3 0 i==8 //pasblanc d check
4 4 4 3 3 0 i==9 //cond check
4 2 1 1 4 0 i==10 //pasblanc d check
4 4 1 1 4 0 i==11 //cond check
4 3 2 2 4 0 i==12
4 3 3 2 4 0 i==13 //premier cond(si == cond[2])
4 3 4 2 4 0 i==14
4 2 2 4 4 0 i==15 //pasblanc d check
4 4 2 4 4 0 i==16 //cond check
4 2 3 4 4 0 i==17 //pasblanc d check

4 4 3 4 4 0 i==18 //cond check
4 2 4 4 4 0 i==19 //pasblanc d check
4 4 4 4 4 0 i==20 //cond check
3 2 1 1 0 1 i==21 //maison (si!=cond[2]) check |pasblanc d check
3 4 1 1 0 1 i==22 //maison_premier (!= cond[2])
3 3 2 2 0 1 i==23 //maison (si!=cond[2]) check
3 3 3 2 0 1 i==24 //maison (si!=cond[2]) check
3 3 4 2 0 1 i==25 //maison (si!=cond[2]) check
3 2 2 4 0 1 i==26 //maison (si!=cond[2]) check | pasblanc d check
3 4 2 4 0 1 i==27 //maison_premier (!= cond[2])
3 2 3 4 0 1 i==28 //maison (si!=cond[2]) check | pasblanc d check
3 4 3 4 0 1 i==29
3 2 4 4 0 1 i==30 //maison (si!=cond[2]) check | pasblanc d check
3 4 4 4 0 1 i==31 //maison_premier (!= cond[2])
2 1 2 0 1 1 i==32 //maison (si!=cond[2]) check
2 1 3 0 1 1 i==33 //maison (si!=cond[2]) check
2 1 4 0 1 1 i==34 //maison (si!=cond[2]) check
1 1 0 4 1 1 i==35 //maison (si!=cond[2]) check
1 2 0 3 3 1 i==36 //maison (si!=cond[2]) check | pasblanc d check
1 4 0 3 3 1 i==37 //maison_premier (!= cond[2])
0 0 2 3 3 1 i==38 //maison (si!=cond[2]) check
0 0 3 3 3 1 i==39 //maison (si!=cond[2]) check
0 0 4 3 3 1 i==40 //maison (si!=cond[2]) check
2 2 2 0 4 1 i==41 //maison (si!=cond[2]) check | pasblanc d check
2 4 2 0 4 1 i==42 //maison_premier (!= cond[2])
2 2 3 0 4 1 i==43 //maison (si!=cond[2]) check | pasblanc d check
2 4 3 0 4 1 i==44
2 2 4 0 4 1 i==45 //maison (si!=cond[2]) check | pasblanc d check
2 4 4 0 4 1 i==46 //maison_premier (!= cond[2])
0 0 1 1 4 1 i==47 //maison (si!=cond[2]) check
1 3 0 2 4 1 i==48 //maison (si!=cond[2]) check
1 2 0 4 4 1 i==49 //maison (si!=cond[2]) check | pasblanc d check
1 4 0 4 4 1 i==50 //maison_premier (!= cond[2])
0 0 2 4 4 1 i==51 //maison (si!=cond[2]) check
0 0 3 4 4 1 i==52 //maison (si!=cond[2]) check
0 0 4 4 4 1 i==53 //maison (si!=cond[2]) check
2 2 2 0 2 2 i==54
2 4 2 0 2 2 i==55 //toutrang(==cond 0 || 2) check
2 2 3 0 2 2 i==56 //toutrang_premier(cond[0 | 2])
2 4 3 0 2 2 i==57 //toutrang(==cond 0 || 2) check
2 2 4 0 2 2 i==58
2 4 4 0 2 2 i==59 //toutrang(==cond 0 || 2) check
0 0 1 1 2 2 i==60
1 3 0 2 2 2 i==61 //toutrang_premier(cond[0 | 2])
1 2 0 4 2 2 i==62
1 4 0 4 2 2 i==63 //toutrang(==cond 0 || 2) check
0 0 2 4 2 2 i==64

0 0 3 4 2 2 i==65 //toutrang_premier(cond[0 | 2])
0 0 4 4 2 2 i==66
3 2 1 1 0 3 i==67
3 4 1 1 0 3 i==68 //toutrang(==cond 0 || 2) check
3 3 2 2 0 3 i==69 //toutrang_premier(cond[0 | 2])
3 3 3 2 0 3 i==70 //toutrang_premier(cond[0 | 2])
3 3 4 2 0 3 i==71 //toutrang_premier(cond[0 | 2])
3 2 2 4 0 3 i==72
3 4 2 4 0 3 i==73 //toutrang(==cond 0 || 2) check
3 2 3 4 0 3 i==74 //toutrang_premier(cond[0 | 2])
3 4 3 4 0 3 i==75 //toutrang(==cond 0 || 2) check
3 2 4 4 0 3 i==76
3 4 4 4 0 3 i==77 //toutrang(==cond 0 || 2) check
2 1 2 0 1 3 i==78
2 1 3 0 1 3 i==79 //toutrang_premier(cond[0 | 2])
2 1 4 0 1 3 i==80
1 1 0 4 1 3 i==81
1 2 0 3 3 3 i==82
1 4 0 3 3 3 i==83 //toutrang(==cond 0 || 2) check
0 0 2 3 3 3 i==84
0 0 3 3 3 3 i==85 //toutrang_premier(cond[0 | 2])
0 0 4 3 3 3 i==86
2 2 2 0 4 3 i==87
2 4 2 0 4 3 i==88 //toutrang(==cond 0 || 2) check
2 2 3 0 4 3 i==89 //toutrang_premier(cond[0 | 2])
2 4 3 0 4 3 i==90 //toutrang(==cond 0 || 2) check
2 2 4 0 4 3 i==91
2 4 4 0 4 3 i==92 //toutrang(==cond 0 || 2) check
0 0 1 1 4 3 i==93
1 3 0 2 4 3 i==94 //toutrang_premier(cond[0 | 2])
1 2 0 4 4 3 i==95
1 4 0 4 4 3 i==96 //toutrang(==cond 0 || 2) check
0 0 2 4 4 3 i==97
0 0 3 4 4 3 i==98 //toutrang_premier(cond[0 | 2])
0 0 4 4 4 3 i==99
3 2 1 1 0 4 i==100 //Impossible gauche check
3 4 1 1 0 4 i==101 //toutrang(==cond 0 || 2) check
3 3 2 2 0 4 i==102 //toutrang_premier(cond[0 | 2])
3 3 3 2 0 4 i==103 //toutrang_premier(cond[0 | 2])
3 3 4 2 0 4 i==104 //toutrang_premier(cond[0 | 2])
3 2 2 4 0 4 i==105 //Impossible gauche check
3 4 2 4 0 4 i==106 //toutrang(==cond 0 || 2) check
3 2 3 4 0 4 i==107 //Impossible gauche check
3 4 3 4 0 4 i==108 //toutrang(==cond 0 || 2) check
3 2 4 4 0 4 i==109 //Impossible gauche check
3 4 4 4 0 4 i==110 //toutrang(==cond 0 || 2) check
2 1 2 0 1 4 i==111

2 1 3 0 1 4 i==112 //toutrang_premier(cond[0 | 2])
2 1 4 0 1 4 i==113
1 1 0 4 1 4 i==114
1 2 0 3 3 4 i==115 //Impossible gauche check
1 4 0 3 3 4 i==116 //toutrang(==cond 0 || 2) check
0 0 2 3 3 4 i==117
0 0 3 3 3 4 i==118 //toutrang_premier(cond[0 | 2])
0 0 4 3 3 4 i==119
2 2 2 0 4 4 i==120 //Impossible gauche check
2 4 2 0 4 4 i==121 //toutrang(==cond 0 || 2) check
2 2 3 0 4 4 i==122 //Impossible gauche check
2 4 3 0 4 4 i==123 //toutrang(==cond 0 || 2) check
2 2 4 0 4 4 i==124 //Impossible gauche check
2 4 4 0 4 4 i==125 //toutrang(==cond 0 || 2) check
0 0 1 1 4 4 i==126
1 3 0 2 4 4 i==127 //toutrang_premier(cond[0 | 2])
1 2 0 4 4 4 i==128 //Impossible gauche check
1 4 0 4 4 4 i==129 //toutrang(==cond 0 || 2) check
0 0 2 4 4 4 i==130
0 0 3 4 4 4 i==131 //toutrang_premier(cond[0 | 2])
0 0 4 4 4 4 i==132
- Si tu n'as que du vin, alors personne d'autre ne doit en avoir (grâce à la fonction enterzone).
Nous récupérons bien nos 5 maisons qui vérifient toutes les conditions.
4 3 2 2 4 0
3 4 3 4 0 1
0 0 1 1 2 2
1 2 0 3 3 3
2 1 4 0 1 4

Chapitre 2

Procédures mises en oeuvre

2.1 Répartition des tâches

Durant les semaines qui précédent les vacances de noël, nous avons travaillé de concert au bocal. Le premier jour des vacances, Nous avons terminé de programmer toutes les fonctions ensemble (chez Gaëtan Gousseaud). Durant la moitier de la première semaine des vacances, Gaëtan Gousseaud s'est occupé de débuguer le programme pendant que Vincent Munoz commençait la rédaction du rapport. Puis nous avons travaillé chacun chez soi tout en informant le second membre du binôme de l'avancée du travail.

2.2 Auto-évaluation

Nous avons travaillé de façon égale sur le projet, et d'un commun accord, nous estimons que 50% du projet à été réaliser par un membre du binôme et 50% par l'autre.

Chapitre 3

Mode d'emploi

Notre projet a un makefile standard. Avec une simple compilation, vous optiendrez la solution unique au problème. Les 5 dernières lignes correspondent aux 5 maisons finales. Et les 6 collones aux catégories de celles-ci. On doit alors regarder la signification de chaque chiffre dans le fichier Scrib.

Chapitre 4

Listing du programme

4.1 Les différents fichiers

1. rapport (format éléctronique)
2. rapport (version papier)
3. affiche est un fichier en langage C contenant les fonctions qui affichent des informations sur le terminal
4. lib est le fichier header de notre projet
5. main est un fichier en langage C qui contient la fonction main.
6. corp est le fichier principale contenant la majorité de nos fonctions.
7. corps_printfs est le même fichier que corp mais les printf et les commentaires qui nous ont permi de nous repérer.
8. makefile
9. scrib est un fichier texte donnant la signification d'un chiffre.

4.2 Description des différentes fonctions

lib.h

Header standart contenant les prototypes de toutes les fonctions du programme, la structure du panier et les librairies suivantes : -stdio.h -stdlib.h -stdbool.h

corps.c

Comme son nom l'indique, c'est le corps du projet contenant les fonctions principales influants sur le cours du programme telles que :

-auxpuiss et tpuiss : Fonction pow avec des int, sa fonction terminale et son chapeau.

-remplir_maison : Utilisée dans main.c. Fonction qui ira remplir un tableau à deux dimension avec le nombre de maisons puissance le nombre de conditions (5 puissance 6) (malloc inclus) possibilités qui auront été tester par les conditions principales (non-quartier), elle renvoie la structure (panier : Tableau 2D(maisons potables) et int (nombres de ces maisons potables)).

-remplir_panier : Utilisée dans remplir_maison. Fonction remplissant le tableau 2D au fur et à mesure avec les maisons potables (épurée des conditions principales). Passe un tableau 1D dans le tableau 2D.

-remplir_cond : Utilisée dans verif_quartier. Transforme la ligne du tableau à 2D de conditions de quartier en un tableau à 1D (malloc inclus).

-verif_maison : Utilisée dans remplir_maison. Fonction qui fait tester la maison avec les conditions principales. Si la condition est applicable à la maison et que la maison ne correspond pas alors la maison n'est pas acceptée (return 0) sinon elle est bonne/potable (return 1).

-make_tabrang : Utilisée dans main.c. Crée le tableau contenant le nombre d'éléments pour chaque rang correspondant.

-triangulaire : Utilisée dans verif_quartier et épuration. Fonction "triangulaire" modifiée pour pouvoir coopérer avec le tableau de rang. Elle sert à avoir un encadrement d'un rang (utilisé dans les for()).

-track : Utilisée dans verif_quartier. Va chercher toutes les maisons valides dans le tableau de maison potable (maison sans -1) et retourne un tableau 2D.

-reversecond : Utilisée dans verif_quartier. Sert à renverser une fonction de quartier.

-enterzone : Utilisée dans verif_quartier. Cherche la ligne correspondant au premier élément valide d'un rang (utilisée pour la variable zone).

-verif_quartier : Fonction utilisée dans main.c. Fonction faisant tourner le programme (en résumé), elle permet de parcourir le tableau de maisons potables et de les tester avec :

=>Les conditions de quartier à l'endroit et renversés (à l'aide de épuration).

=>Entre elles (maisons prioritaires/uniques).

=>Cas spécifiques de à droite et à gauche.

On va aussi tester quelle caractéristique est prioritaire pour faire tourner épuration, par exemple, norvégien à côté de bleue, la condition n'est valide que si une maison d'un rang unique ne contient que norvégien ou maison bleue.

-epuration : Utilisée dans verif_quartier. Teste les cas basiques de à droite, à gauche, et voisin/à côté de. Aiguillage à l'aide des valeurs de mouvements (0 = à droite, 1 = gauche, 2 = voisin).

main.c

Contient le main qui a :

-Les tableaux statiques des conditions de quartier et principales.

-Les valeurs essentielles au bon fonctionnement du programme (nombre de conditions,de maison...).

-Ainsi que des fonctions d'affichage optionelles.

-La modulation non-achevée.

affiche.c

Contient des fonctions d'affichage plus ou moins inutiles, surtout utilisées lors du débogage :

-affiche : Fonction de départ nous ayant aidé à élaborer la base du projet (type bruteforce), non utilisée dans le programme mais laissée à des fins ludiques.

-affiche_panier : affiche les maisons potables dans le main ainsi que leur nombre.

-affichetab : Affiche un tableau à une dimension.

-affiche_res : Affiche le tableau final.

-affiche_panier_aux : Pareil que affiche_panier mais non influençable par le champ mpotable (utilise un int fixe).