

Programmation Impérative II
Projet 2018 - 2019 : Mahjong Solitaire(n29)

Nous présentons ici le projet du jeu du mahjong solitaire en langage C, réalisé par PAN Julien et BERHOUMA Dhia Eldine.

1 Introduction

Le Mahjong solitaire est un jeu pour joueur unique contenant 144 tuiles. Les 144 tuiles sont arrangées dans des dispositions spéciales avec leurs faces tournées vers le haut. Une tuile qui peut être déplacée à gauche ou à droite sans déranger les autres tuiles est dite exposée.

1.1 But du jeu

Le but du jeu est de dégager la surface de jeu des tuiles en réalisant des paires de tuiles correspondantes. Ce n'est que là où chaque tuile a au moins un côté (gauche ou droit) libre sans avoir d'autres tuiles placées en dessus peuvent être retirées en faisant des paires.

2 Rapport

2.1 Présentation

Le but du projet était de créer un Mahjong Solitaire, et que l'ordinateur puisse résoudre le jeu en utilisant la méthode du backtracking (retour sur trace), sans mémorisation de la disposition.

2.2 Choix

Tout d'abord, pour créer la surface du Mahjong solitaire, on a créé un tableau contenant une structure avec des dimensions non variable. Avec une fonction, on initialise le tableau à "0", et à certaines cases, on leur attribue une valeur tel que "-1", pour qu'ensuite, nous puissions remplir les cases nécessaires avec une valeur aléatoire.

Les valeurs aléatoires étaient créées dans un autre tableau de 1 à 36, et pour chaque valeur, il y en avait 4. Pour l'aléatoire, on a permuté les places aléatoirement dans le tableau puis les cases du Mahjong avec "-1" reçoivent les valeurs du tableau mélangé.

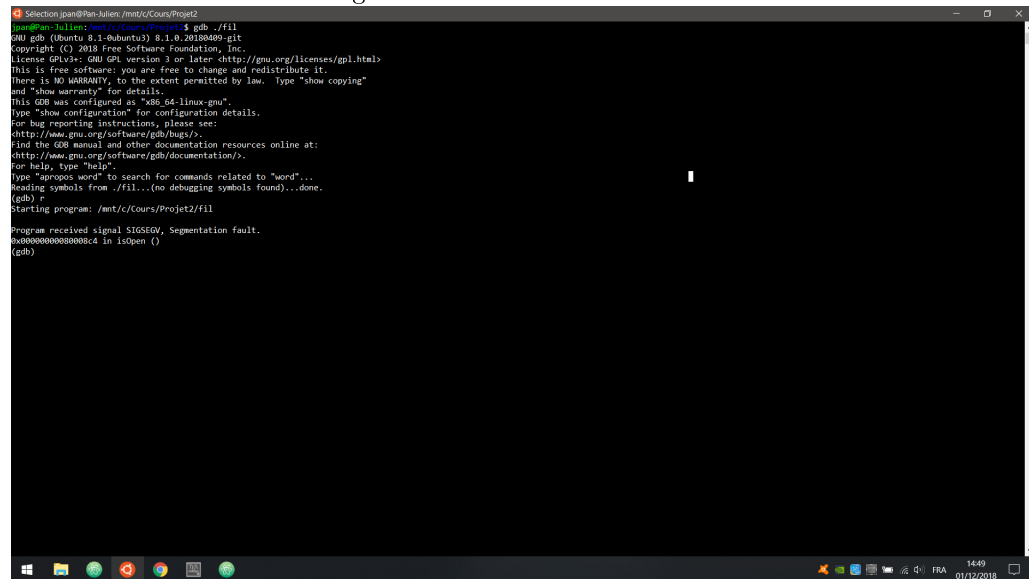
Pour faire des paires, on a créé une fonction récursive "run" qui commence à chercher les paires libres égales à 36, puis on décrémente la valeur. Les paires trouvées sont insérées dans un autre tableau (tab2) avec une structure contenant la valeur du paire, avec ses coordonnées [colonne][ligne][hauteur].

Dès que la valeur est plus petite que 1, on affiche le tableau des valeurs libres et visibles, et toutes les tuiles du mahjong restantes. On vérifie s'il reste des paires

avec une fonction qui le vérifié, si elle renvoie "true", la fonction run continuais en remettant la valeur à 36, et ainsi de suite. La fonction pouvait aussi envoyé "false", si c'est le cas, on s'arrête, on réinitialise le Mahjong. Puis on fait un autre run en essayant de ne pas faire les mêmes paires du premier essaie.

2.3 Les difficultés

On a essayé de chercher les paires sans utiliser une valeur qui décrémentait, mais cela affichait une erreur de segmentation.



```
Selection.jan@man-juken: /mnt/c/Cours/Projet2$ gdb ./fil
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Loading symbols from ./fil...(no debugging symbols found)...done.
(gdb) r
Starting program: /mnt/c/Cours/Projet2/fil

Program received signal SIGSEGV, Segmentation fault.
0x00000000000000c4 in isOpen ()
(gdb)
```

On a essayé de faire un autre fonction qui recommençait du début du jeu en essayant une autre valeur comme début de paire contenu dans le tab2. Mais cela ne fonctionnait pas comme prévu.

```
jean@Pan-Julien: /mnt/C/Cours/Projet2
Position : (0) [11][12][4] : 24
Position : (1) [12][17][0] : 32
Position : (2) [6][16][3] : 25
Position : (3) [7][15][2] : 23
Position : (4) [11][12][1] : 18
Position : (5) [13][16][0] : 18
Position : (6) [11][13][2] : 15
Position : (7) [10][13][2] : 11
Position : (8) [6][12][3] : 10
Position : (9) [2][13][2] : 5
Position : (10) [10][15][2] : 4
Position : (11) [11][15][1] : 4
Position : (12) [12][12][0] : 2
Position : (13) [6][17][1] : 1
Position : (14) [12][13][0] : 36
Position : (15) [12][12][0] : 31
Position : (16) [8][15][2] : 28
Position : (17) [11][12][0] : 28
Position : (18) [8][12][1] : 27
Position : (19) [15][14][0] : 26
Position : (20) [8][16][2] : 20
Position : (21) [10][12][2] : 17
Position : (22) [15][14][0] : 13
Position : (23) [11][16][1] : 12
Position : (24) [14][14][0] : 7
Position : (25) [7][16][1] : 3
Position : (26) [4][15][0] : 35
Position : (27) [9][12][1] : 35
Position : (28) [13][13][0] : 34
Position : (29) [10][16][2] : 30
Position : (30) [12][16][0] : 29
Position : (31) [14][13][0] : 23
Position : (32) [13][14][0] : 21
Position : (33) [13][13][0] : 9
Position : (34) [10][13][1] : 8
Position : (35) [11][16][0] : 8
Position : (36) [12][14][0] : 2
Position : (37) [10][17][1] : 1
Position : (38) [11][17][0] : 31
Position : (39) [8][15][1] : 24
Position : (40) [6][16][0] : 22
Position : (41) [6][12][0] : 22
Position : (42) [9][15][2] : 19
Position : (43) [9][15][1] : 19
Position : (44) [12][13][0] : 14
Position : (45) [9][13][1] : 18
Position : (46) [10][12][1] : 6
Position : (47) [8][13][0] : 33
Position : (48) [11][12][0] : 33
Position : (49) [8][15][1] : 30
```

3 Mode d'emploi

Après compilation et exécution du programme, l'ordinateur commence à chercher et retirer les paires de cartes du jeu jusqu'à un blocage.

4 Le code

4.1 main.c

```
#include "m.h"

int main(){
    create tab;
    create2 tab2;

    createvalp();
    shuffle();
    initialisetab(tab);
    run(tab, tab2, 0, 36);
    //affiche tab2(tab2);

    return 0;
}
```

4.2 m.h

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define ligne 20
#define colonne 20
#define hauteur 5
#define max 144

struct un{
    int value;
    int vf;
};
typedef struct un create [colonne][ligne][hauteur];

struct deux{
    int value;
    int col;
    int lig;
    int hau;
};
typedef struct deux create2 [100];

int valp[max];

void createvalp(void);
void shuffle(void);
bool isOpen(create, int, int, int);
void initialisetab(create);
void affiche(create);
void affichelibre(create);
void affichetab2(create2);
int reinitialise(create);
bool checknb(create, int);
int run(create, create2, int, int);
//int run2(create, create2, int, int);
```

4.3 mahjong.c

```
#include "m.h"

// Création des valeurs des 144 cartes
void createvalp(void){
```

```

int i, valeur, count;
valeur = 1;
count = 0;
for(i = 0; i < max; i++){
    if(count == 4){
        valeur++;
        count = 0;
    }
    valp[i] = valeur;
    count++;
}
}
// Permet d'afficher les valeurs des paires et leurs coordonnées
// faites dans la fonction run
void affichetab2(create2 tab2){
    int i;
    for(i = 0; i < 100; i++)
        printf("%d = [%d][%d][%d] : %d\n", i,
            tab2[i].col, tab2[i].lig, tab2[i].hau, tab2[i].value);
}

// Mélange le tableau valp
void shuffle(void){
    int i, x, temp;
    for(i = 1; i <= max - 1; i++){
        x = rand() % i + 1;
        temp = valp[i];
        valp[i] = valp[x];
        valp[x] = temp;
    }
}

// Vérifie si la carte est libre et visible
bool isOpen(create tab, int c, int l, int h){
    if(tab[c][l][h].vf > 1){
        if((c == 3) && (l == 13) && (h == 0))
            if(tab[c-1][l+1][h].vf > 1){
                return false;
            }

        if((c == 3) && (l == 15) && (h == 0))
            if(tab[c-1][l-1][h].vf > 1){
                return false;
            }
    }
    // Tuiles à droite
    if((c == 15) && (l == 14) && (h == 0))

```

```

        if(tab[c+1][l][h].vf > 1){
            return false;
        }

if((c == 14) && (l == 13) && (h == 0))
    if(tab[c+1][l+1][h].vf > 1){
        return false;
    }

if((c == 14) && (l == 15) && (h == 0))
    if(tab[c+1][l-1][h].vf > 1){
        return false;
    }

//Exceptions pour la couche en dessous du pion à la couche 5
if((c == 8) && (l == 13) && (h == 3))
    if(tab[c][l+1][h+1].vf > 1){
        return false;
    }

if((c == 9) && (l == 13) && (h == 3))
    if(tab[c-1][l+1][h+1].vf > 1){
        return false;
    }

if((c == 8) && (l == 15) && (h == 3))
    if(tab[c][l-1][h+1].vf > 1){
        return false;
    }

if((c == 9) && (l == 15) && (h == 3))
    if(tab[c-1][l-1][h+1].vf > 1){
        return false;
    }

if((c == 8) && (l == 14) && (h == 4))
    if(tab[c][l][h].vf > 1){
        return true;
    }

if(tab[c][l][h+1].vf > 1){
    return false;
}

```

```

    else{
        if((tab[c+1][l][h].vf > 1) && (tab[c-1][l][h].vf > 1))
            return false;

        else{
            return true;
        }
    }
}
return true;
}

```

```

// Initialise le mahjong
void initialisetab(create tab){
    int c,l,h;
    int i, x;
    x = 0;

```

```

    for(c = 0; c < colonne; c++)
    for(l = 0; l < ligne; l++)
    for(h = 0; h < hauteur; h++){
        tab[c][l][h].value = 0;
        tab[c][l][h].vf = 0;
    }

```

```

// Couche 1
for(i = 3; i <= 14; i++){
    tab[i][10][0].value = -1;
    tab[i][13][0].value = -1;
    tab[i][15][0].value = -1;
    tab[i][18][0].value = -1;
}

```

```

for(i = 5; i <= 12; i++){
    tab[i][11][0].value = -1;
    tab[i][17][0].value = -1;
}

```

```

for(i = 4; i <= 13; i++){
    tab[i][12][0].value = -1;
    tab[i][16][0].value = -1;
}

```

```

tab[2][14][0].value = -1;
tab[15][14][0].value = -1;
tab[16][14][0].value = -1;

```

```

// Couche 2

for(i = 6; i <= 11; i++){
    tab[i][11][1].value = -1;
    tab[i][12][1].value = -1;
    tab[i][13][1].value = -1;
    tab[i][15][1].value = -1;
    tab[i][16][1].value = -1;
    tab[i][17][1].value = -1;
}

// Couche 30

for(i = 7; i <= 10; i++){
    tab[i][12][2].value = -1;
    tab[i][13][2].value = -1;
    tab[i][15][2].value = -1;
    tab[i][16][2].value = -1;
}

// Couche 4

for(i = 8; i <= 9; i++){
    tab[i][13][3].value = -1;
    tab[i][15][3].value = -1;
}

// Couche 5

tab[8][14][4].value = -1;

for(c = 0; c < colonne; c++)
for(l = 0; l < ligne; l++)
for(h = 0; h < hauteur; h++)
    if(tab[c][l][h].value != 0){
        tab[c][l][h].value = valp[x];
        tab[c][l][h].vf = 2;
        x++;
    }
}

// Affiche toutes les cartes du mahjong
void affiche(create tab){
    int c,l,h;
    printf("Toutes les cartes du mahjong : \n\n");

```



```

    for(c = 0; c < colonne; c++)
    for(l = 0; l < ligne; l++)
    for(h = 0; h < hauteur; h++)
    if((tab[c][l][h].value != 0) && (tab[c][l][h].vf == 2)){
        printf("%d\t", tab[c][l][h].value);
    }
    printf("\n\n");
}

// Affiche toutes les cartes visibles et libres du mahjong
void afficheLibre(create tab){
    int c,l,h;
    printf("Toutes les cartes visibles et libres du mahjong\n\n");
    for(c = 0; c < colonne; c++)
    for(l = 0; l < ligne; l++)
    for(h = 0; h < hauteur; h++)
    if((isOpen(tab,c,l,h) == true) && (tab[c][l][h].vf == 2)){
        printf("%d\t",tab[c][l][h].value);
    }
    printf("\n\n");
}

// Réinitialise le tableau en cas de blocage
int reinitialise(create tab){
    int c, l, h;
    for(c = colonne; c > 0; c--)
    for(l = ligne; l > 0; l--)
    for(h = hauteur; h > 0; h--)
    if(tab[c][l][h].vf == 1){
        tab[c][l][h].vf = 2;
    }
}

// Vérifie s'il reste encore des paires
bool checknb(create tab, int valeur){
    int c, l ,h;
    int c2, l2, h2;
    if(valeur > 36)
        return false;

    for(c = 0; c < colonne; c++)
    for(l = 0; l < ligne; l++)
    for(h = 0; h < hauteur; h++)
    if(isOpen(tab, c, l, h) == true){
        if((tab[c][l][h].value == valeur) && (tab[c][l][h].vf == 2)){
            c2 = c;

```

```

l2 = 1;
h2 = h;
for(c = 0; c < colonne; c++)
for(l = 0; l < ligne; l++)
for(h = 0; h < hauteur; h++)
if((tab[c][l][h].value != 0) && (isOpen(tab, c, l, h) == true)
&& (tab[c][l][h].vf == 2))
    if((c != c2) || (l != l2) || (h != h2))
        if(valeur == tab[c][l][h].value){
            return true;
        }
    }
}

return checknb(tab, valeur + 1);
}
/*
// Deuxième run en choisissant une autre paire comme début

int run2(create tab, create2 tab2, int pos, int valeur){
    int c, l, h;
    int c2, l2, h2;
    int temp;
    if(tab2[pos].value == 0)
        return 0;

    temp = tab2[pos].value;
    for(c = 0; c < colonne; c++)
    for(l = 0; l < ligne; l++)
    for(h = 0; h < hauteur; h++)
    if(isOpen(tab, c, l, h) == true){
        if((tab[c][l][h].value == valeur) && (tab[c][l][h].vf == 2)
&& (tab[c][l][h].value != temp)){
            c2 = c;
            l2 = l;
            h2 = h;
            for(c = 0; c < colonne; c++)
            for(l = 0; l < ligne; l++)
            for(h = 0; h < hauteur; h++)
            if((tab[c][l][h].value != 0) && (isOpen(tab, c, l, h) == true)
&& (tab[c][l][h].vf == 2))
                if((c != c2) || (l != l2) || (h != h2))
                    if((valeur == tab[c][l][h].value) && (valeur != temp)){

                        printf("Carte 1 : [%d][%d][%d] : %d\n", c, l, h, tab[c][l][h].value);
                        printf("Carte 2 : [%d][%d][%d] : %d\n", c2, l2, h2,
                            tab[c2][l2][h2].value);

```

```

        printf("\n");
        tab[c][l][h].vf = 1;
        tab[c2][l2][h2].vf = 1;
    }
}
}
return run2(tab, tab2, pos, valeur - 1);
}
*/

// Permet de trouver les paires
int run(create tab, create2 tab2, int pos, int valeur){
    int c, l, h;
    int c2, l2, h2;
    if(valeur < 0){
        affichelibre(tab);
        affiche(tab);
        if(checknb(tab, 1) == true)
            return run(tab, tab2, pos, 36);
        else{
            reinitialise(tab);
            printf("Blocage\n");
            return 0; // run2(tab, tab2, 0, 36);
        }
    }
    for(c = 0; c < colonne; c++)
        for(l = 0; l < ligne; l++)
            for(h = 0; h < hauteur; h++)
                if(isOpen(tab, c, l, h) == true){
                    if((tab[c][l][h].value == valeur) && (tab[c][l][h].vf == 2)){
                        c2 = c;
                        l2 = l;
                        h2 = h;

                        for(c = 0; c < colonne; c++)
                            for(l = 0; l < ligne; l++)
                                for(h = 0; h < hauteur; h++)
                                    if((tab[c][l][h].value != 0) && (isOpen(tab, c, l, h) == true)
                                        && (tab[c][l][h].vf == 2))
                                        if((c != c2) || (l != l2) || (h != h2))
                                            if(valeur == tab[c][l][h].value){
                                                printf("Carte 1 : [%d][%d][%d] : %d\n", c, l, h, tab[c][l][h].value);
                                                printf("Carte 2 : [%d][%d][%d] : %d\n", c2, l2, h2,
                                                    tab[c2][l2][h2].value);
                                            }
                    }
                }
    }
}

```

```

        printf("\n");
        tab2[pos].value = tab[c][l][h].value;
        tab2[pos].col = c;
        tab2[pos].lig = l;
        tab2[pos].hau = h;
        pos++;
        tab[c][l][h].vf = 1;
        tab[c2][l2][h2].vf = 1;
    }
}
}
return run(tab, tab2, pos, valeur - 1);
}

```

4.4 makefile

```

CC=gcc
SRC=mahjong.c main.c
OBJ=$(SRC:.c=.o)
FLAGS=-Wall

```

```

fil:    $(OBJ) m.h
$(CC) -o $@ $(OBJ)
%.o: %.c
$(CC) -c $< $(CFLAGS)

```

```

clean:
rm *.o *~ core

```

5 Traces d'exécutions

The top screenshot shows the terminal output for the compilation and execution of the Mahjong program. The commands and their outputs are as follows:

```
pan@Pan-Julien:~/mnt/c/Cours/Projet2$ cd /mnt/c/Cours/La
pan@Pan-Julien:~/mnt/c/Cours/La$ ls
mahjong.tex  pairssegfault.png  tab2.png
pan@Pan-Julien:~/mnt/c/Cours/La$ cd ..
pan@Pan-Julien:~/mnt/c/Cours$ cd Projet2
pan@Pan-Julien:~/mnt/c/Cours/Projet2$ ls
fil m.h mahjong.c mahjong.o main.c main.o makefile
pan@Pan-Julien:~/mnt/c/Cours/Projet2$ make
gcc -c mahjong.c
gcc -o fil mahjong.o main.o
pan@Pan-Julien:~/mnt/c/Cours/Projet2$ ./fil
Carte 1 : [11][13][1] : 34
Carte 2 : [5][17][0] : 34
Carte 1 : [12][17][0] : 32
Carte 2 : [5][11][0] : 32
Carte 1 : [6][16][1] : 25
Carte 2 : [6][11][1] : 25
Carte 1 : [7][15][2] : 23
Carte 2 : [6][11][0] : 23
Carte 1 : [11][12][1] : 18
Carte 2 : [6][13][1] : 18
Carte 1 : [13][16][0] : 18
Carte 2 : [6][13][1] : 18
Carte 1 : [11][11][1] : 15
Carte 2 : [3][10][0] : 15
Carte 1 : [10][13][2] : 11
Carte 2 : [7][11][1] : 11
Carte 1 : [6][12][1] : 10
Carte 2 : [4][16][0] : 10
Carte 1 : [7][13][2] : 5
Carte 2 : [7][11][0] : 5
Carte 1 : [10][15][2] : 4
Carte 2 : [7][12][2] : 4
Carte 1 : [11][15][1] : 4
Carte 2 : [7][12][2] : 4
```

The bottom screenshot shows the full 14-card hand for each player, displayed as a grid of 14 columns and 2 rows. The columns are labeled with the player's name and the card's value and suit. The rows are labeled with the player's name and the card's value and suit.

Player	Card	Value	Suit
Toutes les cartes du mahjong :	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [12][11][0] : 36	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [7][13][1] : 36	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [12][12][0] : 31	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [7][12][1] : 31	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [8][11][1] : 28	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [7][16][2] : 28	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [11][12][0] : 28	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [7][16][2] : 28	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [8][12][1] : 27	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [3][15][0] : 27	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [16][14][0] : 26	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [9][12][2] : 26	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [8][16][2] : 20	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [3][18][0] : 20	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [10][12][2] : 17	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [3][13][0] : 17	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [15][14][0] : 13	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [8][14][4] : 13	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [11][16][1] : 12	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [7][17][1] : 12	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [14][18][0] : 7	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [9][15][3] : 7	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 1 : [7][16][1] : 3	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13
Carte 2 : [4][10][0] : 3	17	27	20
	36	31	29
	13	9	20
	19	7	7
	11	1	32
	21	16	23
	27	7	13

