TIPE

Fichier Code

Olivier Caffier

```c
#include "structure3.h"

#include <time.h>


typedef int** cell; // De taille 4, 0 pour top, on poursuit dans le sens des aiguilles d'une montre



struct plateau_s{
    int len; // Donne le nombre de slots, pour un plateau de n*n cases, len = n
    cell* components;
    bool completed;
};
typedef struct plateau_s plateau;

int nb_liaisons(cell* compo, int num_card){
    int compteur = 0;
    for (int i=0; i<4; i++){
        compteur += (*compo)[num_card][0] ;
    }
    return compteur % 2;
}

void trouve_slots_prio(int** conditions, int taille_plateau, MaxPriorityQueue* file_prio){
    int* compteurs_concern = malloc(taille_plateau * sizeof(int));
    for (int i=0; i< taille_plateau; i++){
        compteurs_concern[i] = 0;
    }

    int nb_conditions = conditions[0][0];

    for (int i=1; i<nb_conditions+1; i++){
        int slot_A = conditions[i][0];
        int slot_B = conditions[i][1];
        compteurs_concern[slot_A] +=1;
        compteurs_concern[slot_B] +=1;
    }

    for (int id_case=0 ; id_case<taille_plateau; id_case++){
        insert(file_prio,compteurs_concern[id_case],id_case);
    }

}


int* compte_nb_slots_cards(cell* cards, int nb_cases){
    int* res = malloc(nb_cases * sizeof(int));
    for (int id_card=0; id_card<nb_cases; id_card++){
        int tmp =0;
        for (int id_side=0; id_side<4; id_side++){
            if (cards[id_side][0] >0){
                tmp+=1;
            }
        }
        res[id_card]=tmp;
    }
    return res ;
}




void trouve_cards_useful(int** conditions, int taille_plateau, bool** cards_usefuls, cell* cards){

    int* compteurs_concern = malloc(taille_plateau * sizeof(int));
```

```
    for (int i=0; i< taille_plateau; i++){
        compteurs_concern[i] = 0;
    }

    int nb_conditions = conditions[0][0];

    for (int i=1; i<nb_conditions+1; i++){
        int slot_A = conditions[i][0];
        int slot_B = conditions[i][1];
        compteurs_concern[slot_A] +=1;
        compteurs_concern[slot_B] +=1;

    }


    int* nb_slots_cards = compte_nb_slots_cards(cards,taille_plateau);


    for (int id_case=0; id_case<taille_plateau; id_case++){
        for (int id_card=0; id_card<taille_plateau; id_card++){
            // La carte id_card pour l'emplacement id_case
            if (compteurs_concern[id_case] <= nb_slots_cards[id_card]){
                // La carte a assez de slots pour convenir aux contraintes imposees a id_case
                cards_usefuls[id_case][id_card] = true;
            }
            else{
                cards_usefuls[id_case][id_case] = false;
            }
        }
    }




}

void rotate_right(cell card){
    // 1ere etape
    for (int i=0; i<4; i++){
        int nb_neighbors = card[i][0];
        for (int j=1; j< nb_neighbors +1; j++){
            card[i][j] = (card[i][j] + 1)%4;
        }
    }

    int* tmp = malloc(4*sizeof(int));
    for (int i=0; i<card[0][0]+1; i++){
        tmp[i]=card[0][i];
    }

    for (int i=0; i<card[3][0]+1; i++){
        card[0][i]=card[3][i];
    }
    for (int i=0; i<card[2][0]+1; i++){
        card[3][i]=card[2][i];
    }
    for (int i=0; i<card[1][0]+1; i++){
        card[2][i]=card[1][i];
    }
    for (int i=0; i<tmp[0]+1; i++){
        card[1][i]=tmp[i];
    }
    free(tmp);


```

```
}

bool in_bornes(int n,int nb){
    return (nb >= 0 && nb < n);
}

bool accessible(plateau* board, int case_start, int case_end, int side_start, int side_end, bool* vus){
    printf("appel accessible \n");
    if (case_start == case_end){
        if (side_start == side_end){
            return true;
        }
        else{
            int nb_neighbor = (board->components)[case_start][side_start][0];
            for (int i=1; i<nb_neighbor+1; i++){
                if ((board->components)[case_start][side_start][i]==side_end){
                    return true;
                }
            }
            return false;
        }
    }
    else{
        if (vus[case_start]){
            // Si la case est deja parcourue, on ne la parcourt pas une deuxieme fois
            return false;
        }
        else{
            int nb_neighbor = (board->components)[case_start][side_start][0];
            int len = board->len;
            vus[case_start]=true;
            for (int i=1; i<nb_neighbor+1;i++){
                // On parcourt chaque voisin
                if ((board->components)[case_start][side_start][i]==0 && in_bornes(board->len*board->len,
                    case_start-len)){
                    if ( accessible(board,case_start-len,case_end,2,side_end,vus)){
                        return true;
                    }
                }
                else if ((board->components)[case_start][side_start][i]==1 && in_bornes(board->len*board->
                    len,case_start+1) && (case_start %len!=len-1)){
                    // A CORRIGER
                    if (accessible(board,case_start+1,case_end,3,side_end,vus)){
                        return true;
                    }
                }
                else if ((board->components)[case_start][side_start][i]==2 && in_bornes(board->len*board->
                    len,case_start+len)){
                    if (accessible(board,case_start+len,case_end,0,side_end,vus)){
                        return true;
                    }
                }
                else if ((board->components)[case_start][side_start][i]==3 && in_bornes(board->len*board->
                    len,case_start-1) && (case_start%len !=0)){
                    // A CORRIGER
                    if (accessible(board,case_start-1,case_end,1,side_end,vus)){
                        return true;
                    }
                }
            }
            return false;
        }



    }
```

```
}


bool is_border(plateau* board, int ind_case, int side_case){
    if (!in_bornes(board->len*board->len, ind_case-board->len)){
      // Partie haute du plateau
      if (ind_case % board->len == 0){
        // Position 2 : tester sur le cote gauche
        return side_case == 3;
      }
      else if (ind_case % board->len == board->len-1){
        // Position 4 : tester en bas et a gauche
            return side_case == 1;
      }
      else{
        // Position 3 : tester en bas, a droite et a gauche
                return side_case == 0;
    }
    }
  else if (!in_bornes(board->len*board->len, ind_case + board->len)){
      // Partie basse du plateau


      if (ind_case % board->len == 0){
        // Position 8 : tester la gauche et en bas

                return side_case == 3 || side_case == 2;

      }
      else if (ind_case % board->len == board->len-1){
        // Position 6 : tester en bas et a droite

                return side_case == 1 || side_case == 2;
      }
      else{
        // Position 7 : tester en bas
                return side_case == 2;
      }

  }
  else{
      // Pos 9,1,5
      if (ind_case % board->len == 0){
        // Position 9 : tester a droite, en haut et en bas
                return side_case == 3;
      }
      else if (ind_case % board->len< board->len-1){
                // Case du milieu
        return false;
      }
      else{
        // Position 5 : tester a gauche, en bas et en haut
                return side_case == 1;
      }
  }
}



void affiche_case(cell card){
    for (int i=0; i<4; i++){
        printf("Sommet %d : ", i);
        for (int j=1; j< card[i][0]+1; j++){
            printf("%d ", card[i][j]);
        }
        printf("\n");
```

```
    }
    printf("\n");
}

bool est_valide_board(plateau* board, int** conditions){

    int len = board->len;
    const int NB_COND = conditions[0][0];
    bool* vus = malloc(len*len*sizeof(bool));

    for (int j=0;j<len*len; j++){
        vus[j]=false;

    }


    for (int i=1; i<NB_COND+1; i++){

        int case_start = conditions[i][0];
        int case_end = conditions[i][1];
        int side_start = conditions[i][2];
        int side_end = conditions[i][3];


        bool res = accessible(board,case_start,case_end,side_start,side_end,vus);



        if (!accessible(board,case_start,case_end,side_start,side_end,vus)){

            return false;
        }

        for (int j=0;j<len*len; j++){
            vus[j]=false;
        }
    }
    return true;
}

void reset_board(plateau* board){
    int len = board->len;
    for (int i=0; i<len*len; i++){
        board->components[i]=NULL;
    }
    board->completed = false;
}
void reset_u_cards(bool* tab, int len){
    for (int i=0; i<len; i++){
        tab[i]= false;
    }
}
void reset_rotations_and_ordre(int* tab, int len){
    for (int i=0;i<len;i++){
        tab[i]=-1;
    }
}
void affiche_case_fichier(int* ordre_cards, int* rotations_cards, int len, FILE* fichier){
    // On suppose que le fichier est deja open en ecriture
    fprintf(fichier,"%d \n",len);
    for (int i=0; i<len;i++){
        fprintf(fichier,"%d \n",ordre_cards[i]);
    }
    for (int i=0; i<len;i++){
        fprintf(fichier,"%d \n",rotations_cards[i]);
    }
```

```
}

void transfer_files(MaxPriorityQueue* f1, MaxPriorityQueue* f2){
    // On transfere tout le contenu de f2 dans f1

    while (f2->size != 0){
        Element elt = extractMax(f2);
        insert(f1,elt.weight,elt.identifier);
    }
}

bool bruteforce(int** conditions, plateau* board, cell* cards, bool* used_cards, int indice_case_start, int
    current_slot, int* ordre_cards, int* rotations_cards,FILE* fichier){
    if (board->completed || current_slot >= board->len* board->len){
        if (est_valide_board(board,conditions)){
            printf("SUCCESS ! \n");

            affiche_case_fichier(ordre_cards,rotations_cards,board->len*board->len,fichier);
            // Afficher le board dans le fichier de retour
            return true;
        }
        return false;
    }
    else if (current_slot == 0){
        if (indice_case_start== board->len * board->len){
            return false;
        }
        else{
            for (int i = 0; i < 4; i++){
                rotate_right(cards[indice_case_start]);
                board->components[0]= cards[indice_case_start];
                used_cards[indice_case_start] = true;
                ordre_cards[0]=indice_case_start+1;
                rotations_cards[0]=i+1;
                if (bruteforce(conditions,board,cards,used_cards,indice_case_start,current_slot+1,
                    ordre_cards,rotations_cards,fichier)){
                    return true;
                }
                else{
                    reset_board(board);
                    int LEN = board->len * board->len;
                    reset_u_cards(used_cards,LEN);
                    reset_rotations_and_ordre(ordre_cards,LEN);
                    reset_rotations_and_ordre(rotations_cards,LEN);

                }
            }
            // Si on a teste sur toutes les positions de la carte au depart, il faut changer de carte de
                depart

            // On teste sur la compo suivante
            return bruteforce(conditions,board,cards,used_cards,indice_case_start+1,0,ordre_cards,
                rotations_cards,fichier);
        }
    }
    else{
        // Le board n'est ni completed ni en position de depart
        for (int i=0; i<board->len*board->len; i++){
            if (!used_cards[i]){
                // Pour toutes les cartes non-utilisees
                for (int j=0; j<4; j++){
                    // On teste toutes les orientations possibles
                    bool* copy_tmp = malloc( board->len * board->len * sizeof(bool));
                    int* copy_ordre = malloc(board->len * board->len * sizeof(int));
                    int* copy_rotation = malloc(board->len * board->len * sizeof(int));
                    for (int w = 0; w< board->len * board->len; w++){
```

```
                        copy_tmp[w] = used_cards[w];
                        copy_ordre[w]=ordre_cards[w];
                        copy_rotation[w]=rotations_cards[w];
                    }


                    rotate_right(cards[i]);
                    board->components[current_slot]=cards[i];
                    used_cards[i]=true;
                    ordre_cards[current_slot]=i+1;
                    rotations_cards[current_slot]=j+1;

                    if (bruteforce(conditions,board,cards,used_cards,indice_case_start,current_slot+1,
                        ordre_cards,rotations_cards,fichier)){
                        free(copy_ordre);
                        free(copy_rotation);
                        free(copy_tmp);
                        return true;
                    }
                    else{
                        // Reset la partie modifiee sur le board
                        for (int z =current_slot; z < board->len * board->len ; z++){
                            board->components[z]=NULL;
                        }
                        // Reset le tableau used_cards
                        for (int w = 0; w< board->len * board->len; w++){
                            used_cards[w] = copy_tmp[w] ;
                            rotations_cards[w]=copy_rotation[w];
                            ordre_cards[w]=copy_ordre[w];
                        }
                        free(copy_ordre);
                        free(copy_rotation);
                        free(copy_tmp);
                    }
                }
            }
        }
        return false;
    }
}


bool bruteforce_opti(int** conditions, plateau* board, cell* cards,int taille_plateau, bool start_situation,
     bool* used_cards, int* ordre_cards, int* rotations_cards, MaxPriorityQueue* file_prio, MaxPriorityQueue
    * file_preservation,bool** cards_useful, FILE* fichier, int* nb_appels){

    *nb_appels +=1;
    printf("taille file_prio : %d | ", file_prio->size);
    if (board->completed || file_prio->size <= 0){
        // bool res = est_valide_board(board,conditions);
        printf("\n test final \n");


        if (est_valide_board(board,conditions)){
            printf("SUCCESS %d ! \n", *nb_appels);

            // Renvoyer le plateau dans un fichier de retour
            affiche_case_fichier(ordre_cards,rotations_cards,board->len * board->len, fichier);

            //Renvoyer un booleen pour le signaler aux appels d'au dessus
            return true;
        }
        printf("echec");
        return false;
    }
    else if (start_situation){
```

```
        printf("start");
        // Le plateau est vide

        // On identifie la case du plateau avec le plus de priorites
        Element case_start = extractMax(file_prio);
        int indice_case_start = case_start.identifier;
        int prio_case_start = case_start.weight;

        // On enfile la case et sa valeur de prio dans la file de preservation
        // insert(file_preservation, prio_case_start, indice_case_start);


        // On va essayer les cases utiles pour sa position

        for (int id_card=0; id_card < taille_plateau; id_card++){
            if (cards_useful[indice_case_start][id_card]){
                // On a trouve un carte qui etait jugee comme utile pour cet emplacement et qui n'a pas ete
                    utilisee avant sur celui-ci vu qu'on procede iterativement

                for (int rot=0; rot<4; rot++){
                    rotate_right(cards[id_card]);

                    board->components[id_card] = cards[id_card];
                    used_cards[id_card] = true;
                    ordre_cards[indice_case_start]= id_card;
                    rotations_cards[indice_case_start]=rot+1;
                    if (bruteforce_opti(conditions,board,cards,taille_plateau,false,used_cards,ordre_cards,
                        rotations_cards, file_prio, file_preservation, cards_useful,fichier,nb_appels)){
                        return true;
                    }
                    else{
                        // On reinitialise toutes les donnees modifiees
                        reset_board(board);
                        reset_u_cards(used_cards,taille_plateau);
                        reset_rotations_and_ordre(ordre_cards,taille_plateau);
                        reset_rotations_and_ordre(rotations_cards, taille_plateau);

                        // IL FAUT REMETTRE EN PLACE LA FILE BON DIEU
                        transfer_files(file_prio,file_preservation);
                    }


                }
                // La carte a ete testee dans toutes les rotations

            }
            // On teste donc la carte utile suivante en bouclant
        }

        // On a teste toutes les cartes utiles pour cet emplacement et rien de concluant, cette situation
            est impossible mais je renvoie false au cas ou
        return false;
    }
    else {
        // Le plateau n'est ni en position de depart, ni complete
        // On identifie la case a traiter en premier

        Element current_case = extractMax(file_prio);
        int indice_current_case= current_case.identifier;
        int prio_current_case= current_case.weight;
        printf("current_case : %d \n", indice_current_case);



        // On va essayer les cartes utiles pour cet emplacement

```

```
        for (int id_card = 0; id_card < taille_plateau; id_card++){
            if (cards_useful[indice_current_case][id_card] && !used_cards[id_card]){

                    // On a trouve une carte qui etait jugee comme utile pour cet emplacement et qui n'a pas ete utilisee avant sur celui-ci vu qu'on procede iterativement et qui n'est pas activement utilisee

                    for (int rot=0; rot<4; rot++){
                        // On sauvegarde notre progression
                    bool* copy_tmp = malloc( taille_plateau * sizeof(bool));
                    int* copy_ordre = malloc(taille_plateau * sizeof(int));
                    int* copy_rotation = malloc(taille_plateau * sizeof(int));
                    for (int w = 0; w< taille_plateau; w++){
                        copy_tmp[w] = used_cards[w];
                        copy_ordre[w]=ordre_cards[w];
                        copy_rotation[w]=rotations_cards[w];

                    }
                    rotate_right(cards[id_card]);
                    board->components[id_card] = cards[id_card];

                    used_cards[id_card] = true;

                    ordre_cards[indice_current_case]= id_card;
                    rotations_cards[indice_current_case] = rot+1;
                    if (bruteforce_opti(conditions,board,cards,taille_plateau,false,used_cards,ordre_cards, rotations_cards, file_prio, file_preservation, cards_useful,fichier,nb_appels)){
                        free(copy_ordre);
                        free(copy_rotation);
                        free(copy_tmp);
                        return true;
                    }
                    else{
                        // On remet en place la sauvegarde

// ATTENTION : a MODIFIER -> il faut remettre en place la sauvegarde des cartes (i.e mettre a NULL les modifs des appels d'en dessous)

                        // Reset le tableau used_cards
                        for (int w = 0; w< taille_plateau; w++){
                            used_cards[w] = copy_tmp[w] ;
                            rotations_cards[w]=copy_rotation[w];
                            ordre_cards[w]=copy_ordre[w];
                        }
                        free(copy_ordre);
                        free(copy_rotation);
                        free(copy_tmp);

                        for (int w = 0; w<taille_plateau; w++){
                            if (!used_cards[w]){
                                board->components[w]=NULL;
                            }
                        }

                        // Il faut remettre la file en place
                        // IL FAUT REMETTRE EN PLACE LA FILE BON DIEU
                        transfer_files(file_prio,file_preservation);
                    }

                }
                // La carte a ete testee dans toutes les rotations

            }
        }
        // On enfile la case et sa valeur de prio dans la file de preservation
        insert(file_preservation, prio_current_case, indice_current_case);
```

```
577
578         // On a teste toutes les cartes dispos sur cet emplacement et ... rien de concluant
579         return false;
580     }
581 }
582
583 void detruire_file_prio(MaxPriorityQueue* pq){
584     while (pq->size != 0){
585         Element tmp = extractMax(pq);
586         printf("element detruit : %d \n", tmp.identifier);
587     }
588     free(pq);
589 }
590 /*
591     MAIN
592 */
593
594 int main(){
595
596     /*
597         INITIALISATION DU PLATEAU
598     */
599
600
601     // Initialisation des cases
602     cell p1 = malloc(4*sizeof(int*));
603     p1[0]=malloc(5*sizeof(int));
604     p1[1]=malloc(5*sizeof(int));
605     p1[2]=malloc(5*sizeof(int));
606     p1[3]=malloc(5*sizeof(int));
607     p1[0][0] = 2;
608     p1[0][1] = 1;
609     p1[0][2] = 2;
610     p1[1][0] = 2;
611     p1[1][1] = 0;
612     p1[1][2] = 2;
613     p1[2][0] = 2;
614     p1[2][1] = 0;
615     p1[2][2] = 1;
616     p1[3][0] = 0;
617
618
619
620     cell p2 = malloc(4*sizeof(int*));
621     p2[0]=malloc(5*sizeof(int));
622     p2[1]=malloc(5*sizeof(int));
623     p2[2]=malloc(5*sizeof(int));
624     p2[3]=malloc(5*sizeof(int));
625     for (int i=0;i<4;i++){
626         p2[i][0]=1;
627     }
628     p2[0][1] = 3;
629     p2[1][1] = 2;
630     p2[2][1] = 1;
631     p2[3][1] = 0;
632
633     cell p3 = malloc(4*sizeof(int*));
634     p3[0]=malloc(5*sizeof(int));
635     p3[1]=malloc(5*sizeof(int));
636     p3[2]=malloc(5*sizeof(int));
637     p3[3]=malloc(5*sizeof(int));
638     for (int i=0;i<4;i++){
639         p3[i][0]=1;
640     }
641     p3[0][1]=2;
642     p3[1][1]=3;
```

```
643     p3[2][1]=0;
644     p3[3][1]=1;
645
646
647     cell p4 = malloc(4*sizeof(int*));
648     p4[0]=malloc(5*sizeof(int));
649     p4[1]=malloc(5*sizeof(int));
650     p4[2]=malloc(5*sizeof(int));
651     p4[3]=malloc(5*sizeof(int));
652     p4[0][0] = 2;
653     p4[0][1] = 2;
654     p4[0][2] = 3;
655     p4[3][0] = 2;
656     p4[3][1] = 0;
657     p4[3][2] = 2;
658     p4[2][0] = 2;
659     p4[2][1] = 0;
660     p4[2][2] = 3;
661     p4[1][0] = 0;
662
663
664     cell p5 = malloc(4* sizeof(int*));
665     p5[0]=malloc(5*sizeof(int));
666     p5[1]=malloc(5*sizeof(int));
667     p5[2]=malloc(5*sizeof(int));
668     p5[3]=malloc(5*sizeof(int));
669     p5[0][0]=0;
670     p5[1][0]=1;
671     p5[1][1]=2;
672     p5[2][0]=1;
673     p5[2][1]=1;
674     p5[3][0]=0;
675
676     cell p6 = malloc(4* sizeof(int*));
677     p6[0]=malloc(5*sizeof(int));
678     p6[1]=malloc(5*sizeof(int));
679     p6[2]=malloc(5*sizeof(int));
680     p6[3]=malloc(5*sizeof(int));
681     p6[0][0]=3;
682     p6[0][1]=1;
683     p6[0][2]=2;
684     p6[0][3]=3;
685     p6[1][0]=3;
686     p6[1][1]=0;
687     p6[1][2]=2;
688     p6[1][3]=3;
689     p6[2][0]=3;
690     p6[2][1]=0;
691     p6[2][2]=1;
692     p6[2][3]=3;
693     p6[3][0]=3;
694     p6[3][1]=0;
695     p6[3][2]=1;
696     p6[3][3]=2;
697
698
699     cell p7 = malloc(4* sizeof(int*));
700     p7[0]=malloc(5*sizeof(int));
701     p7[1]=malloc(5*sizeof(int));
702     p7[2]=malloc(5*sizeof(int));
703     p7[3]=malloc(5*sizeof(int));
704     p7[0][0]=0;
705     p7[1][0]=1;
706     p7[1][1]=2;
707     p7[2][0]=1;
708     p7[2][1]=1;
```

```
    p7[3][0]=0;

    cell p8 = malloc(4* sizeof(int*));
    p8[0]=malloc(5*sizeof(int));
    p8[1]=malloc(5*sizeof(int));
    p8[2]=malloc(5*sizeof(int));
    p8[3]=malloc(5*sizeof(int));
    p8[0][0]=1;
    p8[0][1]=3;
    p8[1][0]=0;
    p8[2][0]=0;
    p8[3][0]=1;
    p8[3][1]=0;

    cell p9 = malloc(4* sizeof(int*));
    p9[0]=malloc(5*sizeof(int));
    p9[1]=malloc(5*sizeof(int));
    p9[2]=malloc(5*sizeof(int));
    p9[3]=malloc(5*sizeof(int));
    for (int i=0;i<4;i++){
        p9[i][0]=1;
    }
    p9[0][1]=1;
    p9[1][1]=0;
    p9[2][1]=3;
    p9[3][1]=2;









    plateau* main_board = malloc(sizeof(plateau));
    const int LEN = 3;
    const int NB_CASES = 9;
    main_board->len=LEN;
    main_board->completed=false;
    main_board->components= malloc(NB_CASES * sizeof(cell));
    cell* compo = malloc(NB_CASES * sizeof(cell));
    compo[0]=p1;
    compo[1]=p2;
    compo[2]=p3;
    compo[3]=p4;
    compo[4]=p5;
    compo[5]=p6;
    compo[6]=p7;
    compo[7]=p8;
    compo[8]=p9;

    bool* used_cards = malloc(NB_CASES * sizeof(bool));

    int* ordre = malloc(NB_CASES * sizeof(int));
    int* rotation = malloc(NB_CASES * sizeof(int));
    for (int i=0; i<NB_CASES;i++){
        used_cards[i]=false;
        ordre[i]=-1;
        rotation[i]=-1;
    }

    // File de priorites

    MaxPriorityQueue file_prio;
    file_prio.size=0;
    MaxPriorityQueue file_preservation;
    file_preservation.size=0;
    MaxPriorityQueue* file_prio_main = malloc(sizeof(MaxPriorityQueue));
    MaxPriorityQueue* file_preservation_main = malloc(sizeof(MaxPriorityQueue));
```

```
    *file_prio_main = file_prio;
    *file_preservation_main= file_preservation;


    // Cards useful



    /*
        Initialisation des conditions
    */
    int** conditions = malloc(6*sizeof(int*));
    conditions[0]=malloc(sizeof(int));
    conditions[1]=malloc(4*sizeof(int));
    conditions[2]=malloc(4*sizeof(int));
    conditions[3]=malloc(4*sizeof(int));




    conditions[0][0]=3;

    conditions[1][0]=5;
    conditions[1][1]=1;
    conditions[1][2]=1;
    conditions[1][3]=0;

    conditions[2][0]=1;
    conditions[2][1]=0;
    conditions[2][2]=0;
    conditions[2][3]=3;

    conditions[3][0]=0;
    conditions[3][1]=8;
    conditions[3][2]=3;
    conditions[3][3]=1;

    bool** cards_useful = malloc(NB_CASES * sizeof(bool*));
    for (int i=0; i<NB_CASES; i++){
        cards_useful[i]=malloc(NB_CASES * sizeof(bool));
    }
    trouve_cards_useful(conditions,NB_CASES,cards_useful,compo);

    trouve_slots_prio(conditions,NB_CASES,file_prio_main);

    /*
        MISC.
    */

    //FILE* fichier = fopen("test.txt","w+");

    clock_t t;

    /*
    if (bruteforce(conditions,main_board,compo,used_cards,0,0,ordre,rotation,fichier)){
        printf("BRUTEFORCE -> OK \n");
    }

    bool* vus = malloc(NB_CASES *sizeof(bool));
    for (int i=0;i<NB_CASES;i++){
        vus[i]=false;
    }

    if (accessible(main_board,6,8,2,1,vus)){
        printf("accessible");
    }*/
```

```
841
842
843
844     printf(" \n Bruteforce opti : \n");
845     FILE* fichier2 = fopen("bruteforce_opti.txt","w+");
846     t = clock();
847     int* nb_appels = malloc(sizeof(int));
848     *nb_appels=1;
849
850     if (bruteforce_opti(conditions,main_board,compo,NB_CASES,true,used_cards,ordre,rotation,file_prio_main,
            file_preservation_main,cards_useful,fichier2,nb_appels)){
851         printf("BRUTEFORCE OPTI -> OK \n");
852          t = clock() - t;
853         printf(" \n =====> Perf = %f seconds", ((float)t) / CLOCKS_PER_SEC);
854     }
855
856
857
858
859
860
861
862
863     /*
864         LIBERATION DE MEMOIRE
865     */
866
867     for (int i=0;i<4;i++){
868         free(p1[i]);
869         free(p2[i]);
870         free(p3[i]);
871         free(p4[i]);
872         free(p5[i]);
873         free(p6[i]);
874         free(p7[i]);
875         free(p8[i]);
876         free(p9[i]);
877     }
878
879     free(p1);
880     free(p2);
881     free(p3);
882     free(p4);
883     free(p5);
884     free(p6);
885     free(p7);
886     free(p8);
887     free(p9);
888
889     free(main_board->components);
890     free(main_board);
891
892     for (int i=0; i<NB_CASES; i++){
893         free(cards_useful[i]);
894     }
895     free(cards_useful);
896     detruire_file_prio(file_prio_main);
897     detruire_file_prio(file_preservation_main);
898
899
900
901
902
903     //fclose(fichier);
904     //fclose(fichier2);
905 }
```