```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : strucutres.h
    DerniÃ¨re Modification : 16/05/2010

    Description :
        Header
        Contient une bonne partie des structures de donnÃ©es ainsi que les constantes utiles au jeu
*/

#ifndef STRUCTURES
        #define STRUCTURES

        #define WINDOWS

        #ifdef MAC
                #define IS_MAC 1
                #define IS_WINDOWS 0
        #endif

        #ifdef WINDOWS
                #define IS_MAC 0
                #define IS_WINDOWS 1
        #endif


        //PLATEAU DE JEU
        #define TAILLE_PLAT 30

        #define COUPS_CONTROLE 3
        #define COUPS_VICTOIRE_GB 20
        #define NB_EVOL 1
        #define TOURS_GB 3
        #define TOURS_VIR 6

        #define VS 0
        #define IA 1

        //TABLEAU DES LISTES
        /*
                De faÃ§on Ã  pouvoir utiliser les mÃªmes constantes,
                on surdimensionne le tableau
        */
        #define DIM_TAB_LISTES 4

        //TABLEAU DES POINTS CLEFS
        #define DIM_TAB_POINTS_CLEFS 5

        //TYPE DE PION A PLACER
        #define CASE 0
        #define GLOBULE 1
        #define VIRUS 2
        #define GR 3

        //TYPE DE CASE
        #define NORMALE 0
        #define BLOQUEE 1
        #define CONTROLE 2
        #define ENTREE_VIRUS 3
        #define ENTREE_GR 4

        //VITESSES
        #define VITESSE_INI_VIRUS 4
        #define VITESSE_INI_GB 4
        #define VITESSE_INI_GR 4

        //TAILLE DU MASQUE
```

```c
#define VITESSE_MAX 4
#define MASQUE_MAX (VITESSE_MAX * 2 + 1)


struct coord
{
        int x;
        int y;
}; typedef struct coord coord;

struct S_pion
{
        int type_pion;
        int niveau;
        int vitesse;
        coord pos;
        struct S_pion* suiv;
}; typedef struct S_pion S_pion;

struct S_case
{
        S_pion* pion;
        int type;
}; typedef struct S_case S_case;

struct S_masque
{
    int occupe ; // 0 si libre , 1 si ami ou bloque , 2 si ennemi
    int poids ;
}; typedef struct S_masque S_masque;
#endif
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : main.cpp
    Dernière Modification : 16/05/2010

    Description :
        Le main n'a pour but que de lancer partie() ou editeur_prgm(). Ainsi, il a ses propres surfaces qui ne sont utilisés que
pour lui et lui seul.

*/




#include <stdlib.h>
#include <stdio.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h> // Bibliotheque qui prend en charge tout les types d'images
#include <SDL/SDL_ttf.h> // Bibliotheque de texte
#include <SDL/SDL_getenv.h>// Bibliotheque de positionement de fenetre
#include <FMOD/fmod.h> // Bibliotheque de lecture de sons

#include "partie.h"
#include "editeur.h"
#include "outils_main.h"



int main(int argc, char *argv[])
{
/* CREATION DES SURFACES POUR LA SDL */
    SDL_Surface* tableau_surface_main[NB_SURFACE_MAIN];
    SDL_Rect tableau_position_main[NB_POS_MAIN];


/* DECLARATIONS DES VARIABLES */
    int menu=-1, son=1;
    FSOUND_STREAM *musique = NULL;

/* CHARGEMENT DES SURFACES */
    init_main(tableau_surface_main, tableau_position_main);

/* CHARGEMENT ET DEFINITION DE LA MUSIQUE */
    musique = FSOUND_Stream_Open("music/musique.mp3", FSOUND_LOOP_NORMAL, 0, 0);
    FSOUND_Stream_SetLoopCount(musique, -1);// Boucle à l'infini
    FSOUND_SetVolume(FSOUND_ALL, 255);// Reglage du volume


/* MENU PRINCIPAL */
    while(menu!=QUITTER)
    {
        FSOUND_Stream_Play(FSOUND_FREE, musique);// Lecture de la musique

        afficher_menu(tableau_surface_main, tableau_position_main, &son);

        choix_menu(tableau_surface_main, tableau_position_main, &menu, &son);

        if(menu==JOUER)
            {
                FSOUND_Stream_Stop(musique);// On arrete la musique
                partie();       //game.h
            }
        else if(menu==EDITEUR)
            {
                FSOUND_Stream_Stop(musique);
```

```
                editeur_prg(); // editeur.h
                SDL_ShowCursor(SDL_ENABLE); // Au cas où l'utilisateur a quitté l'éditeur en ayant le curseur désactivé.
            }

    }

    liberer_sdl_main(tableau_surface_main); // Libération des surfaces utiles au main.

    FSOUND_Stream_Close(musique);// Libération de l'espace mémoire
    FSOUND_Close(); // Quitte la bibliotheque de gestion de la musique

    return EXIT_SUCCESS;
}
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : outils_main.h
    Dernière Modification : 16/05/2010

    Description :
       Header
*/

#ifndef OUTILS_MAIN_H_INCLUDED
#define OUTILS_MAIN_H_INCLUDED

#define LARGEUR_FENETRE 620
#define HAUTEUR_FENETRE 730

#define NB_SURFACE_MAIN 8
#define NB_POS_MAIN 7

#define MENU_PRINCIPAL 0
#define TITRE_MENU 1
#define MENU1 2
#define MENU2 3
#define MENU3 4
#define IMAGE1 5
#define SON_ON 6
#define SON_OFF 7


#define QUITTER 0
#define JOUER 1
#define EDITEUR 2
#define SON 3

void init_main(SDL_Surface* tableau_surface_main[], SDL_Rect tableau_position_main[]);
void choix_menu(SDL_Surface* tableau_surface_main[], SDL_Rect tableau_position_main[], int* menu,int *son);
void afficher_menu(SDL_Surface* tableau_surface_main[], SDL_Rect tableau_position_main[], int* son);
void liberer_sdl_main(SDL_Surface* tableau_surface_main[]);


#endif // OUTILS_MAIN_H_INCLUDED
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : outils_main.cpp
    Dernière Modification : 16/05/2010

    Description :
        Contient les fonctions utiles au main. Ce ne sont que des fonctions d'affichage,
        d'initialisation, de libération ou d'aquisition du choix du menu principal.
*/

#include <stdlib.h>
#include <stdio.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_ttf.h>
#include <SDL/SDL_getenv.h>
#include <FMOD/fmod.h>

#include "outils_main.h"


void init_main(SDL_Surface* tableau_surface_main[], SDL_Rect tableau_position_main[])
/* PS : /
    Initialise et créé les surfaces et les positions.
*/
{
    int i;

    /* INITIALISATION DES SURFACES */
    for(i=0;i<=NB_SURFACE_MAIN;i++)
        tableau_surface_main[i]=NULL;

    /* INITIALISATION DES DIFFERENTS GESTIONNAIRES */
    SDL_Init(SDL_INIT_VIDEO); // chargement de la SDL
    TTF_Init();               // chargement du gestionnaire de police
    FSOUND_Init(44100, 32, 0);//chargement du gestionnaire de son

    /* OUVERTURE D'UNE FENETRE */
    putenv("SDL_VIDEO_WINDOW_POS=center"); //Centre la fenêtre par rapport à l'écran
    tableau_surface_main[MENU_PRINCIPAL] = SDL_SetVideoMode(LARGEUR_FENETRE, HAUTEUR_FENETRE, 32, SDL_HWSURFACE);
    SDL_WM_SetIcon(IMG_Load('pic/icones/icone_fenetre.png'), NULL); // Change l'icone de la fenêtre
    SDL_WM_SetCaption("Immuno Wars", NULL); //Renome la fenetre



    /* CHARGEMENT DES IMAGES */
    tableau_surface_main[IMAGE1] = SDL_LoadBMP('pic/fonds/acceuil1.bmp');
        tableau_position_main[IMAGE1].x=0;
        tableau_position_main[IMAGE1].y=HAUTEUR_FENETRE-tableau_surface_main[IMAGE1]->h;
    tableau_surface_main[SON_ON] = IMG_Load('pic/icones/son_on.png');
    tableau_surface_main[SON_OFF] = IMG_Load('pic/icones/son_off.png');
        tableau_position_main[SON_ON].x=LARGEUR_FENETRE-tableau_surface_main[SON_ON]->w;
        tableau_position_main[SON_ON].y=HAUTEUR_FENETRE-tableau_surface_main[SON_ON]->h;


    /* CREATION DES POLICES */
    TTF_Font *police_titre = NULL, *police = NULL; // creation des polices
    police = TTF_OpenFont("polices/police_menu.ttf", 30);  // definition des polices
    police_titre = TTF_OpenFont("polices/police_menu_titre.ttf", 100);
    SDL_Color noir = {0,0,0};// creation d'une couleur


/* DEFINITION DES ZONES DE TEXTE */
    tableau_surface_main[TITRE_MENU] = TTF_RenderText_Blended(police_titre,"IMMUNO WARS",noir);
        tableau_position_main[TITRE_MENU].x=(LARGEUR_FENETRE-tableau_surface_main[TITRE_MENU]->w)/2;// centrage du texte
        tableau_position_main[TITRE_MENU].y=20;
```

```
    tableau_surface_main[MENU1] = TTF_RenderText_Blended(police, "JOUER", noir);
        tableau_position_main[MENU1].x=(LARGEUR_FENETRE-tableau_surface_main[MENU1]->w)/2;// centrage du texte
        tableau_position_main[MENU1].y=200;
    tableau_surface_main[MENU2] = TTF_RenderText_Blended(police, "EDITEUR", noir);
        tableau_position_main[MENU2].x=(LARGEUR_FENETRE-tableau_surface_main[MENU2]->w)/2;// centrage du texte
        tableau_position_main[MENU2].y=300;
    tableau_surface_main[MENU3] = TTF_RenderText_Blended(police, "QUITTER", noir);
        tableau_position_main[MENU3].x=(LARGEUR_FENETRE-tableau_surface_main[MENU3]->w)/2;// centrage du texte
        tableau_position_main[MENU3].y=400;

    TTF_CloseFont(police); // liberation des polices
    TTF_CloseFont(police_titre);
}

void choix_menu(SDL_Surface* tableau_surface_main[], SDL_Rect tableau_position_main[], int* menu,int *son)
/* PS: menu, son
    Attend le choix de l'utilisateur (clic de souris) pour redéfinir menu ou son.
*/
{
    *menu=-1;
    SDL_Event event;


    while(*menu==-1)
        {
            SDL_WaitEvent(&event);
            switch(event.type)
            {
                case SDL_QUIT: // Si l'utilisateur clic sur la croix de fermeture de fenetre
                    *menu=QUITTER;
                    break;
                case SDL_MOUSEBUTTONDOWN :
                    if (event.button.button == SDL_BUTTON_LEFT)
                    {
                        if((event.button.x>=tableau_position_main[MENU1].x && event.button.x<=tableau_position_main[MENU1].x
+tableau_surface_main[MENU1]->w) && (event.button.y>=tableau_position_main[MENU1].y &&
event.button.y<=tableau_position_main[MENU1].y+tableau_surface_main[MENU1]->h))// Si le joueur clique sur 'jouer'
                            *menu=JOUER;
                        else if((event.button.x>=tableau_position_main[MENU2].x &&
event.button.x<=tableau_position_main[MENU2].x+tableau_surface_main[MENU2]->w) &&
(event.button.y>=tableau_position_main[MENU2].y && event.button.y<=tableau_position_main[MENU2].y+tableau_surface_main[MENU2]-
>h)) // Si le joueur clique sur 'editeur'
                            *menu=EDITEUR;
                        else if((event.button.x>=tableau_position_main[MENU3].x &&
event.button.x<=tableau_position_main[MENU3].x+tableau_surface_main[MENU3]->w) &&
(event.button.y>=tableau_position_main[MENU3].y && event.button.y<=tableau_position_main[MENU3].y+tableau_surface_main[MENU3]-
>h)) // Si le joueur clique sur 'quitter'
                            *menu=QUITTER;
                        else if((event.button.x>=tableau_position_main[SON_ON].x &&
event.button.x<=tableau_position_main[SON_ON].x+tableau_surface_main[SON_ON]->w) &&
(event.button.y>=tableau_position_main[SON_ON].y && event.button.y<=tableau_position_main[SON_ON].y
+tableau_surface_main[SON_ON]->h))// Si le joueur clique sur l'icone du son
                            {
                                FSOUND_SetPaused(FSOUND_ALL, *son);
                                if(*son) // Si le son est actif
                                {
                                    *son=0; // Alors on l'arrete
                                    *menu=SON;
                                }
                                else
                                {
                                    *son=1; // Sinon on l'active
                                    *menu=SON;
                                }
                            }
                    }
                    break;
            }
        }
}
```

```c
void afficher_menu(SDL_Surface* tab_s[], SDL_Rect tab_p[], int* son)
/* PS : /
    Affiche le menu principal
*/
{
        SDL_FillRect(tab_s[MENU_PRINCIPAL], NULL, SDL_MapRGB(tab_s[MENU_PRINCIPAL]->format, 255, 255, 255));

        SDL_BlitSurface(tab_s[IMAGE1], NULL, tab_s[MENU_PRINCIPAL], &tab_p[IMAGE1]);

        SDL_BlitSurface(tab_s[TITRE_MENU], NULL, tab_s[MENU_PRINCIPAL], &tab_p[TITRE_MENU]);

        SDL_BlitSurface(tab_s[MENU1], NULL, tab_s[MENU_PRINCIPAL], &tab_p[MENU1]);
        SDL_BlitSurface(tab_s[MENU2], NULL, tab_s[MENU_PRINCIPAL], &tab_p[MENU2]);
        SDL_BlitSurface(tab_s[MENU3], NULL, tab_s[MENU_PRINCIPAL], &tab_p[MENU3]);
        if(*son)
            SDL_BlitSurface(tab_s[SON_ON], NULL, tab_s[MENU_PRINCIPAL], &tab_p[SON_ON]);
        else
            SDL_BlitSurface(tab_s[SON_OFF], NULL, tab_s[MENU_PRINCIPAL], &tab_p[SON_ON]);

        SDL_Flip(tab_s[MENU_PRINCIPAL]);
}

void liberer_sdl_main(SDL_Surface* tableau_surface_main[])
/* PS : /
    Libère les surfaces en vue de quitter le programme
*/
{
    int i;
    for(i=0;i<NB_SURFACE_MAIN;i++)
       SDL_FreeSurface(tableau_surface_main[i]);

    TTF_Quit(); // Quitte la bibliotheque de gestion du texte
    SDL_Quit(); // Quitte la SDL

}
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : partie.h
    Dernière Modification : 16/05/2010

    Description :
        Header
*/

#ifndef PARTIE_H_INCLUDED
#define PARTIE_H_INCLUDED

#include "structures.h"
#include "extraction_fichier.h"
#include "affichage_sdl.h"
#include "masque.h"
#include "jeu.h"
#include "outils_sdl.h"
#include "ia.h"

int partie();


#endif // PARTIE_H_INCLUDED
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : partie.cpp
    Dernière Modification : 16/05/2010

    Description :
        Il contient le programme de jeu. Noyau du programme, il est le 'main' de la partie.
*/

#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>


#include "partie.h"

int partie()
{

        /* CREATION DES VARIABLES */
        int joueur, i = 0, encombrement_pion, coups = 0, quitter=0;
        int mode_jeu;  //IA ou VS
        coord case_ini, case_fin; //structures.h

    /* CREATION DES DONNEES DE LA SDL */
        SDL_Surface* tableau_surface[NB_SURFACE]; // Tableau de surfaces
        SDL_Rect tableau_position[NB_POS]; // Tableau de positions
        initialiser_SDL(tableau_surface, tableau_position);


        /*
                La grille de jeu, sous forme de tableau bi-dimensionnel
                S_case est défini dans structures.h
        */
        S_case plateau_jeu[TAILLE_PLAT][TAILLE_PLAT];
        S_masque masque[MASQUE_MAX][MASQUE_MAX];
        coord tab_points_clefs[DIM_TAB_POINTS_CLEFS];


        /*
                La liste des globules. Elle doit servir pour l'IA.
                S_pion est défini dans structures.h
        */

        S_pion* tab_listes[DIM_TAB_LISTES];
                tab_listes[GLOBULE] = NULL;
                tab_listes[VIRUS] = NULL;
                tab_listes[GR] = NULL;


        /*
                Génère le plateau et la liste
                Le plateau est rempli à partir d'un fichier texte selon des règles prédéfinies
                La liste est remplie au même moment.
        */
    generer_plateau(plateau_jeu, tab_listes, tab_points_clefs);//extraction_fichier.h

        /*
                Le premier joueur est la défense
        */
        joueur = GLOBULE;
    /*
        On demande à l'utilisateur quel type de partie veut-il faire ? Humain VS Humain ou Humain VS IA.
    */
        mode_jeu=selection_mode_jeu(tableau_surface);
```

```
        do
        {
                /*
                        Introduire les changements:
                                -faire entrer des virus
                                -gonfler les globules
                                -injecter les globules rouges
                */


                afficher_joueur(joueur, tableau_surface, tableau_position);
                afficher_toto(tableau_surface, tableau_position);



                if(tab_listes[joueur] != NULL)
                {
                    if(mode_jeu == IA && joueur == VIRUS)
                 /*
                        On laisse l'IA gérer les déplacements de ses pions.
                */
                {
                    meilleur_coup(plateau_jeu, tab_listes, tab_points_clefs, &case_ini, &case_fin);
                    masque_prgm(masque, plateau_jeu[case_ini.x][case_ini.y].pion, plateau_jeu);// Utile au chemin.

                    SDL_Delay(400); // Pour séparer les coups de l'IA
                }
                        else
                /*
                        Demander les coordonnées
                        Vérifier que les coordonnées saisies sont valides
                        Demander la destination
                        Vérifier que les coordonnées de destination saisies sont valides
                */

                        {
                    do{
                        afficher_plateau(plateau_jeu, tableau_surface, tableau_position, case_ini, 0);
                        SDL_Flip(tableau_surface[ECRAN]);

                        do demander_coord(&case_ini, 1, tableau_surface, tableau_position, &quitter);// L'utilisateur doit cliquer
sur le pion qu'il souhaite déplacer
                        while(verif_coord(plateau_jeu, case_ini, joueur)==0);

                        //Attention, comme un globule peut occuper plusieurs cases,
                        //il faut bien sélectionner la case caractéristique (la plus proche de l'origine)
                        if(joueur==GLOBULE)
                        {
                            case_ini = case_caract(plateau_jeu, case_ini);
                            encombrement_pion = plateau_jeu[case_ini.x][case_ini.y].pion->niveau;
                        }
                        else
                            encombrement_pion = 1;

                        masque_prgm(masque, plateau_jeu[case_ini.x][case_ini.y].pion, plateau_jeu);
                        afficher_masque(masque,tableau_surface, tableau_position, case_ini);

                        demander_coord(&case_fin, 2, tableau_surface, tableau_position, &quitter);// L'utilisateur doit cliquer sur
l'emplacement où il souhaite déplacer le pion
                    }while(is_valid(plateau_jeu, masque, case_ini, case_fin, joueur, encombrement_pion,
tab_points_clefs[CONTROLE])==0);

                        }

            afficher_chemin(masque, plateau_jeu, case_ini, case_fin, tableau_surface, tableau_position);// Affiche pixel par
pixel la progression du pion déplacé
                        deplacement(plateau_jeu, tab_listes, case_ini, case_fin, tab_points_clefs[CONTROLE]);/Déplacer le pion,
gère les attaques, les divisions de globules blancs etc.
                }
```

```
        if(joueur == GLOBULE)
            entree_pions(plateau_jeu, tab_listes, tab_points_clefs[ENTREE_VIRUS], VIRUS);// On fait entrer des virus

        deplacer_gr(plateau_jeu, tab_listes[GR]); // Deplacer le globule rouge de façon aléatoire

        i++;

                if((joueur == GLOBULE && i%5 == 0) || (joueur == VIRUS && i%9 == 0))// On rentre dans la condition tout les 5
tours pour les globules blancs et tout les 9 tours pour les virus
                    {
            i=0;
            coups++;
                    evolution_globules(plateau_jeu, tab_listes[GLOBULE]);
            changer_joueur(&joueur);
            entree_pions(plateau_jeu, tab_listes, tab_points_clefs[ENTREE_GR], GR);// On fait entrer des globules rouges
                }

    /*
        On continue la partie tant qu'il reste des globules ou des virus, que l'utilisateur ne veut pas quitter, qu'il n'y a pas
assez de virus dans
        le point de contrôle et que le nombre de tours défini pour la victoire des globules blancs n'est pas atteint!
    */
        }while(quitter==0 && tab_listes[GLOBULE] != NULL && tab_listes[VIRUS] != NULL && cpt_controle(0)<COUPS_CONTROLE &&
coups<COUPS_VICTOIRE_GB);

        afficher_vainqueur(tab_listes, tableau_surface, coups);
        cpt_controle(2); // Remise à zéro du compteur

    liberer_SDL(tableau_surface);
    return 0;
}
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : jeu.h
    DerniÃ¨re Modification : 16/05/2010

    Description :
        Header
*/
#ifndef JEU
        #define JEU

        #include "structures.h"
        #include "tableau_jeu.h"
        #include "listes.h"

        void changer_joueur(int* joueur);

        void deplacement(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], const coord case_ini,
const coord case_fin, const coord pt_ctrl);

        void evolution_globules(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* L_gb);
        void entree_pions(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion** L_deb, const coord pt_entree, const int type_pion);

        coord case_caract(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], coord case_jeu);

        int verif_coord(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], const coord case_jeu, const int joueur);
        int is_valid(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_masque masque[MASQUE_MAX][MASQUE_MAX], coord case_ini, coord
case_jeu, const int joueur, int encombrement_pion, coord pt_ctrl);

        void maj_glob(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion** L_deb_glob, coord case_jeu);
        int cpt_controle(int add);
        void deplacer_gr(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* Grouge);
#endif
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : jeu.cpp
    Dernière Modification : 16/05/2010

    Description :
        Contient les fonctions appellées depuis
        partie.cpp qui gèrent le moteur du jeu, toute
        la logique et les actions réalisables.
                Certaines d'entre elles nécessitent des
        fonctions secondaires qui se trouvent alors
        dans tableau_jeu.cpp
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "jeu.h"

void changer_joueur(int* joueur)
{
        //Passe au joueur suivant

        switch(*joueur)
        {
                case GLOBULE: *joueur = VIRUS; break;
                default: *joueur = GLOBULE; break;
        }
}

void deplacement(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], const coord case_ini, const coord
case_fin, const coord pt_ctrl)
{
        /*
                Cette fonction déplace un virus ou un globule en gérant les attaques


                case_ini & case_fin ont été vérifiées sont valides pour
                le déplacement

                Récupérer le pion

                Si c'est un globule
                        Supprimer le globule de la case de départ
                        Ajouter le globule sur la case d'arrivée, en gérant
                        les phagocytoses (mange un autre pion si présent)

                Si c'est un Virus
                        Si la case d'arrivée est la point de contrôle
                                Supprimer le virus
                                Incrémenter le point de contrôle
                        Si la case d'arrivée contient un globule
                                Attaquer le globule
                                Mettre à jour le globule
                                Supprimer le virus
                        Sinon
                                Si la case d'arrivée contient un globule rouge
                                        Supprimer le globule rouge
                                Déplacer le virus

        */

        S_pion* pion_ini = plateau[case_ini.x][case_ini.y].pion;
        S_pion* pion_fin = plateau[case_fin.x][case_fin.y].pion;

        if(pion_ini->type_pion == GLOBULE) //Ménage dans les trois cases autour
```

```
        {
                del_glob(plateau, pion_ini, case_ini);
                add_glob(plateau, tab_listes, pion_ini, case_fin);
        }
        else
        {
                if(coord_eq(case_fin, pt_ctrl))
                {
                        supprimer_element(&tab_listes[VIRUS], pion_ini);
                        cpt_controle(1); //Incrémente le compteur de virus ayant atteint le point de contrôle

                }
                else if( pion_fin == NULL || pion_fin->type_pion != GLOBULE )
                {
                    if(pion_fin !=NULL && pion_fin->type_pion == GR)
                supprimer_element(&tab_listes[GR], pion_fin);

                        //Déplace le pion

                        plateau[case_fin.x][case_fin.y].pion = pion_ini;              //Dans le tableau

                        plateau[case_fin.x][case_fin.y].pion->pos = case_fin;  //Dans la liste

                }
                else //-> atteindre GB

                {
                        attaque_virus(pion_ini, pion_fin);
                        maj_glob(plateau, &tab_listes[GLOBULE], case_fin);
                        supprimer_element(&tab_listes[VIRUS], pion_ini);
                }

                plateau[case_ini.x][case_ini.y].pion =NULL;
        }
}

void evolution_globules(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* L_gb)
{
        /*
                Cette fonction permet de faire évoluer un certain nombre de globules
        */
        S_pion* glob = NULL;
        int i = 0;
        coord A, B;

        //Parcours de la liste des globules

        while(i<NB_EVOL && L_gb != NULL)
        {
        switch(L_gb->niveau)
        {
                //Si c'est un globule de niveau 1

            case 1:
                //On vérifie que le globule a la place pour grossir

                A = L_gb->pos; A.x++;
                if(case_est_libre(plateau, A)==1)
                {
                    L_gb->niveau++;                                     //Le globule passe au niveau 2

                    plateau[A.x][A.y].pion = glob;      //On màj le tableau en conséquence

                    i++;        //On a fait évoluer un globule : le compteur augmente

                }
                break;
            case 2:
                //Même logique que le niveau deux, mais il faut rajouter deux pointeurs dans le tableau
```

```
                A = L_gb->pos; A.x++; A.y++;
                B = L_gb->pos; B.y++;
                if(case_est_libre(plateau, A)==1 && case_est_libre(plateau, B)==1)
                {
                    L_gb->niveau++;
                    plateau[A.x][A.y].pion = L_gb;
                    plateau[B.x][B.y].pion = L_gb;
                    i++;
                }
                break;
            default: break;
        }
        L_gb = L_gb->suiv;
        }
}

void entree_pions(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], const coord pt_entree, const int
type_pion)
{
        /*
                Cette fonction crée un pion du type voulu aléatoirement autour de son point d'entrée
        */
        int i,j;
        int creer = 0;
        coord sortie;

        srand(time(NULL)); //seed

        // On vérifie d'abord qu'une case est disponible autour du point d'entrée

        for(i=-1; i<2; i++)
        {
                for(j=-1; j<2; j++)
                {
                        sortie.x = i + pt_entree.x;
                        sortie.y = j + pt_entree.y;
                        if(case_est_libre(plateau, sortie)==1)
                                creer=1;
                }
        }

        //On crée le pion si cela est possible

        if(creer==1)
        {
                //Recherche de coordonnées aléatoires

                do
                {
                        sortie.x=rand()%3; sortie.x = sortie.x == 2 ? -1 : sortie.x;   //Après cette ligne, sortie.x appartient
à { -1 ; 0 ; 1 }

                        sortie.y=rand()%3; sortie.y = sortie.y == 2 ? -1 : sortie.y;   //Idem

                        sortie.x += pt_entree.x;               //On ajoute cela aux coordonnées du point d'entrée, et on
obtient alors l'une des cases

                        sortie.y += pt_entree.y;               //autour dudit point d'entrée

                }while(coord_eq(sortie, pt_entree)==1 || case_est_libre(plateau, sortie)!=1);//Reste à vérifier que la case est
disponible


                //Création du pion

                plateau[sortie.x][sortie.y].pion = nouveau_pion(type_pion, tab_listes);
                plateau[sortie.x][sortie.y].pion->pos = sortie;

                //Niveau aléatoire pour les virus
```

```
                if(type_pion == VIRUS)
            plateau[sortie.x][sortie.y].pion->niveau = rand()%3 + 1;
        }

}

coord case_caract(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], coord case_jeu)
{
        /*
                Renvoie la case caractéristique d'un pion. Pour les globules de niveau
                supérieur à 1, c'est la case du globule la plus proche de l'origine
        */
        if(plateau[case_jeu.x][case_jeu.y].pion!=NULL && plateau[case_jeu.x][case_jeu.y].pion->type_pion==GLOBULE)
                case_jeu = plateau[case_jeu.x][case_jeu.y].pion->pos;

        return case_jeu;
}

int verif_coord(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], const coord case_jeu, const int joueur)
{
        /*
                Vérifie qu'une case contient bien un pion appartenant au joueur voulu
                Retourne 1 si la condition précédente est vraie, 0 sinon
        */
        S_pion* pion = plateau[case_jeu.x][case_jeu.y].pion;
        int retour = 0;

        if( pion != NULL)
        {
                if(pion->type_pion == joueur)
                        retour = 1;
        }

        return retour;
}

int is_valid(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_masque masque[MASQUE_MAX][MASQUE_MAX], coord case_ini, coord case_jeu,
const int joueur, int encombrement_pion, coord pt_ctrl)
{
        /*
                Vérifie que la case d'arrivée est effectivement disponible pour le pion
                de la case de départ, nécessite le masque pondéré de ce pion
                1 si vrai, 0 si faux
        */
        int x_masque = case_jeu.x - (case_ini.x - VITESSE_MAX);
        int y_masque = case_jeu.y - (case_ini.y - VITESSE_MAX);

        int retour = 0;

        if(x_masque>=0 && x_masque <MASQUE_MAX && y_masque>=0 && y_masque < MASQUE_MAX)
        {
                //On vérifie que la case est disponible pour le pion (info donnée par le masque)

        retour = masque[x_masque][y_masque].poids>0 ? 1 : 0;

                //Si le joueur est un globule de niveau supérieur à 1, il faut vérifier les cases concernées

        if(joueur==GLOBULE && retour==1 && encombrement_pion > 1)
        {
            case_jeu.x++;
            retour = case_est_prenable(plateau, case_ini, case_jeu, joueur);

            if(retour==1 && encombrement_pion==3)
            {
                case_jeu.y++;
                retour = case_est_prenable(plateau, case_ini, case_jeu, joueur);
                case_jeu.x--;
                retour += case_est_prenable(plateau, case_ini, case_jeu, joueur);

                retour = retour==2 ? 1 : 0;
```

```
            }
        }
        }

        return retour;
}

void maj_glob(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion** L_deb_glob, coord case_jeu)
{
        //Met à jour un globule blanc lorsque celui-ci a été attaqué par un virus

        coord caract = case_caract(plateau, case_jeu);
        S_pion* glob = plateau[caract.x][caract.y].pion;

        if(glob->niveau == 0)
        {
                supprimer_element(L_deb_glob, glob);
                plateau[caract.x][caract.y].pion =NULL;
        }
        else if(glob->niveau == 1)
        {
                plateau[caract.x+1][caract.y].pion =NULL;
        }
        else if(glob->niveau == 2)
        {
                plateau[caract.x][caract.y+1].pion =NULL;
                plateau[caract.x+1][caract.y+1].pion =NULL;
        }
}

int cpt_controle(int add)
{
        /*
                Renseigne, incrémente ou remet à zéro le compteur
                de virus ayant atteint le point de contrôle
        */
        static int cpt = 0;
        cpt = add == 1 ? ++cpt : cpt; //Incrémentation préfixée de rigueur

        cpt = add == 2 ? 0 : cpt;
        return cpt;
}

void deplacer_gr(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* Grouge)
{
        /*
                Déplace aléatoirement les globules rouges autour de leur position
        */
        int j=-1, i=-1, creer = 0;
        coord sortie, pt_entree;

        srand(time(NULL));

        while(Grouge != NULL)
        {
                pt_entree = Grouge->pos;

                //Le code est tout à fait similaire au code d'entrée de pions


                while(creer != 1 && !(i==1 && j==1))
                {
                        if(i==2){
                                i=-1;
                                j++;
                        }

                        sortie.x = i + pt_entree.x;
                        sortie.y = j + pt_entree.y;
                        if(case_est_libre(plateau, sortie)==1)
```

```
                        creer=1;
                i++;
        }

        if(creer==1)
        {
                do
                {
                        sortie.x=rand()%3; sortie.x = sortie.x == 2 ? -1 : sortie.x;
                        sortie.y=rand()%3; sortie.y = sortie.y == 2 ? -1 : sortie.y;
                        sortie.x += pt_entree.x;
                        sortie.y += pt_entree.y;
                }while(coord_eq(sortie, pt_entree)==1 || case_est_libre(plateau, sortie)!=1);

                plateau[pt_entree.x][pt_entree.y].pion =NULL;
                plateau[sortie.x][sortie.y].pion = Grouge;
                Grouge->pos = sortie;
        }

        Grouge = Grouge->suiv;
    }
}
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : tableau_jeu.h
    Dernière Modification : 16/05/2010

    Description :
        Header
*/

#ifndef TAB_JEU
        #define TAB_JEU

        #include "structures.h"
        #include "listes.h"

        int case_est_prenable(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], const coord case_ini, const coord case_jeu, const int
joueur);
        void del_glob(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* pion_temp, coord case_ini);
        void add_glob(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], S_pion* pion_temp, coord
case_fin);

        void mitose(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], coord caract);

        int case_est_libre(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], coord case_jeu);
        int case_existe(coord case_jeu);
        int coord_eq(coord a, coord b);

        void attaque_virus(S_pion* virus, S_pion* globule);

#endif
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : tableau_jeu.cpp
    Dernière Modification : 16/05/2010

    Description :
        Contient les fonctions appellées depuis
        jeu.cpp, les noms des fonctions sont explicites
        dans leur grande majorité
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "tableau_jeu.h"

int case_est_prenable(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], const coord case_ini, const coord case_jeu, const int joueur)
{
        S_pion* pion_ini = plateau[case_ini.x][case_ini.y].pion;
        S_pion* pion_fin = plateau[case_jeu.x][case_jeu.y].pion;
        int retour = 1;

        /*
                Une case n'est prenable que si celle si
                        -n'est pas une case bloquée (elle est NORMALE)
                        -ne contient pas un pion de la meme famille

        */

        if(plateau[case_jeu.x][case_jeu.y].type != NORMALE)
                retour = 0;
    else if(pion_fin!=NULL)
        if(pion_fin!=pion_ini && pion_fin->type_pion==joueur)
            retour = 0;

        return retour;
}

void del_glob(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* pion_temp, coord case_ini)
{
        //Supprime un grand pion

        int x,y;
        for(x=0; x<=1 ; x++)
                for(y=0; y<=1; y++)
                        if(plateau[case_ini.x + x][case_ini.y + y].pion == pion_temp)
                                plateau[case_ini.x + x][case_ini.y + y].pion = NULL;
}

void add_glob(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], S_pion* pion_temp, coord case_fin)
{
        //Place un grand pion en fonction de son niveau (GLOBULE)

        int x_p, y_p, x_max, y_max;
        int diviser = 0;
        x_max = pion_temp->niveau > 1 ? 1 : 0;
        y_max = pion_temp->niveau > 2 ? 1 : 0;

        for(y_p=0; y_p <= y_max; y_p++)
                for(x_p=0; x_p <= x_max; x_p++){
                        if(plateau[ case_fin.x + x_p ][ case_fin.y + y_p ].pion!=NULL)
                        {
                                if(plateau[ case_fin.x + x_p ][ case_fin.y + y_p ].pion->type_pion == GR)
                                {
                                        if(pion_temp->niveau == 3)
                                                diviser = 1;
                                        supprimer_element(&tab_listes[GR], plateau[ case_fin.x + x_p ][ case_fin.y +
```

```
y_p ].pion);
                                }
                                //Supprimer de la liste des virus

                                if(plateau[ case_fin.x + x_p ][ case_fin.y + y_p ].pion->type_pion == VIRUS)
                                        supprimer_element(&tab_listes[VIRUS], plateau[ case_fin.x + x_p ][ case_fin.y +
y_p ].pion);
                        }
                        plateau[ case_fin.x + x_p ][ case_fin.y + y_p ].pion = pion_temp;
                }

        plateau[case_fin.x][case_fin.y].pion->pos = case_fin;

        if( diviser == 1 )
                mitose(plateau, tab_listes, case_fin);
}

void mitose(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], coord caract)
{
        plateau[caract.x][caract.y].pion->niveau = 1;

        plateau[caract.x+1][caract.y+1].pion = nouveau_pion(GLOBULE, tab_listes);
        plateau[caract.x+1][caract.y+1].pion->pos.x = caract.x+1;
        plateau[caract.x+1][caract.y+1].pion->pos.y = caract.y+1;

        plateau[caract.x][caract.y+1].pion =NULL;
        plateau[caract.x+1][caract.y].pion =NULL;
}

int case_est_libre(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], coord case_jeu)
{
        if(case_existe(case_jeu)==1 && plateau[case_jeu.x][case_jeu.y].type==NORMALE && plateau[case_jeu.x][case_jeu.y].pion ==
NULL)
                return 1;
        else
                return 0;
}

int case_existe(coord case_jeu)
{
        if(case_jeu.x>=0 && case_jeu.x<TAILLE_PLAT && case_jeu.y>=0 && case_jeu.y<TAILLE_PLAT)
                return 1;
        else
                return 0;
}

int coord_eq(coord a, coord b)
{
        if(a.x==b.x && a.y==b.y)
                return 1;
        else
                return 0;
}

void attaque_virus(S_pion* virus, S_pion* globule)
{
        switch(virus->niveau)
        {
                case 1 :
                        globule->vitesse = (int) ceil(0.25*globule->vitesse); //math.h

                        break;
                case 2 :
                        globule->vitesse = 0;
                        break;
                default :
                        globule->niveau--;
                        break;
        }
}
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : listes.h
    DerniÃ¨re Modification : 16/05/2010

    Description :
        Header
*/
#ifndef LISTES
        #define LISTES

        #include "structures.h"

        S_pion* nouveau_pion(int type_pion, S_pion* tab_listes[DIM_TAB_LISTES]);
        void enfiler_element(S_pion** L_deb, S_pion* element);
        void supprimer_element(S_pion** L_deb, S_pion*element);
        void dupliquer_liste(S_pion** L_deb_2, S_pion* L_deb);
        void supprimer_liste(S_pion* element);

#endif
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : listes.cpp
    Dernière Modification : 16/05/2010

    Description :
        Contient toutes les fonctions permettant
        de manipuler les listes de pions, qui vont de
        pair avec le tableau de jeu
*/
#include <stdio.h>
#include <stdlib.h>

#include "listes.h"

S_pion* nouveau_pion(int type_pion, S_pion* tab_listes[DIM_TAB_LISTES])
{
        /*
                Cette fonction crée un pion et l'ajoute à la liste qui lui
                correspond.
        */
        S_pion* element = (S_pion*) malloc(sizeof(S_pion)); //Allocation


        /*---------CARACTERISTIQUES DU PION--------*/
        element->type_pion = type_pion;
        element->niveau = 1;
        element->suiv = NULL;

        switch(type_pion)
        {
                case GLOBULE:
                        element->vitesse = VITESSE_INI_GB;
                        break;
                case VIRUS :
                        element->vitesse = VITESSE_INI_VIRUS;
                        break;
                default :
                        element->vitesse = VITESSE_INI_GR;
                        break;
        }

        /*---------AJOUT DU PION À LA LISTE--------*/
        enfiler_element(&tab_listes[type_pion], element);

        return element;
}

void enfiler_element(S_pion** L_deb, S_pion* element)
{
        /*
                Chaîne un élément à la fin d'une liste
        */
        S_pion* actuel;

        if(*L_deb == NULL)
                *L_deb = element;
        else
        {
                actuel = *L_deb;
                while(actuel->suiv != NULL)
                        actuel = actuel->suiv;
                actuel->suiv = element;
        }
}

void supprimer_element(S_pion** L_deb, S_pion* element)
```

```
{
        /*
                Supprime un élément d'une liste
        */
        S_pion* prec = NULL;
        S_pion* actuel = *L_deb;

        //On recherche l'élement dans la liste

        while( actuel != element )
        {
        prec = actuel;
        actuel = prec->suiv;
        }

        //On modifie le chaînage dans la liste

    if(prec==NULL)
        *L_deb = actuel->suiv;
    else
        prec->suiv = actuel->suiv;

        //On peut alors supprimer le pion et libérer la mémoire

    free(actuel);
}

void dupliquer_liste(S_pion** L_deb_2, S_pion* L_deb)
{
        /*
                Cette fonction crée une copie de la liste L_deb
                dans L_deb_2. Les deux sont strictement identiques,
                mais les actions sur l'une n'auront pas d'effet sur l'autre.
        */
        S_pion* act1 = L_deb;
        S_pion* nouveau = NULL;

        //Parcours de la liste en entier

        while(act1 != NULL)
        {
                //on crée une copie de chaque élément act1

                nouveau = (S_pion*) malloc(sizeof(S_pion));
                *nouveau = *act1;
                nouveau->suiv = NULL;

                //On le chaîne

                enfiler_element(L_deb_2, nouveau);

                act1 = act1->suiv;
        }
}

void supprimer_liste(S_pion* element)
{
        /*
                Supprime tous les éléments d'une liste
                en passant le premier element en argument
                Attention, le pointeur de début de liste
                n'est pas mis à NULL.
        */
        if(element != NULL)
        {
                supprimer_liste(element->suiv);
                free(element);
        }
}
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : masque.h
    DerniÃ¨re Modification : 16/05/2010

    Description :
        Header.
*/
#ifndef MASQUE_H_INCLUDED
        #define MASQUE_H_INCLUDED

                #include "structures.h"

                //Types de cases occupees
                #define BLOK 1
                #define NEUTRE 2
                #define TARG 3
                #define OPPOSANT 4

                struct S_chemin
                {
                    coord etape;
                struct S_chemin* suiv;
                };typedef struct S_chemin S_chemin;

        void masque_prgm(S_masque masque[MASQUE_MAX][MASQUE_MAX], S_pion* pion_jeu, S_case plateau[TAILLE_PLAT][TAILLE_PLAT]);
        void parcours(S_masque masque[MASQUE_MAX][MASQUE_MAX], coord pos, int cpt, int longueur);
        coord get_pos(coord pos, int pos_rel);
        int poids(S_masque masque[MASQUE_MAX][MASQUE_MAX], coord pos);
        coord origine_masque(coord centre_masque);
        coord masque2tab(coord convert, coord centre_masque);
        coord tab2masque(coord convert, coord centre_masque);
        void chemin(S_masque masque[MASQUE_MAX][MASQUE_MAX], S_chemin** L_chemin, coord case_ini, coord fin);
        void enfiler_element_chemin(S_chemin** L_deb, S_chemin* element);
        void supprimer_chemin(S_chemin* element);

#endif // MASQUE_H_INCLUDED
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : masque.cpp
    Dernière Modification : 16/05/2010

    Description :
        Contient toutes les fonctions qui ont rapport au masque.
        Un masque est un tableau centré sur un pion qui renseigne
        sur les cases qui lui sont accessibles et le nombre de coups
        qu'il lui faut pour y parvenir.
*/
#include <stdio.h>
#include <stdlib.h>

#include "masque.h"

void masque_prgm(S_masque masque[MASQUE_MAX][MASQUE_MAX], S_pion* pion_jeu, S_case plateau[TAILLE_PLAT][TAILLE_PLAT])
{
   int i,j;
   coord case_jeu = pion_jeu->pos;
   coord org_mask = origine_masque(case_jeu);
   int longueur = pion_jeu->vitesse;
   int joueur_actuel = pion_jeu->type_pion;

   for(j=0;j<MASQUE_MAX;j++)       // initialisation du tableau

    {
        for(i=0;i<MASQUE_MAX;i++)
                {
                        if(org_mask.x+i<0 || org_mask.x+i>=TAILLE_PLAT || org_mask.y+j<0 || org_mask.y+j>=TAILLE_PLAT)
                        {//En-dehors plateau

                                masque[i][j].occupe = BLOK;
                                masque[i][j].poids=-1;
                        }
                        else if(plateau[org_mask.x+i][org_mask.y+j].type==CONTROLE && joueur_actuel == VIRUS)
                        {//Prise contrôle par virus

                                masque[i][j].occupe = TARG;
                                masque[i][j].poids=-1;
                        }
                        else if(plateau[org_mask.x+i][org_mask.y+j].type!=NORMALE)
                        {//Case occupée

                                masque[i][j].occupe = BLOK;
                                masque[i][j].poids=-1;
                        }
                        else if(plateau[org_mask.x+i][org_mask.y+j].pion!=NULL && plateau[org_mask.x+i][org_mask.y+j].pion-
>type_pion==joueur_actuel && plateau[org_mask.x+i][org_mask.y+j].pion != plateau[case_jeu.x][case_jeu.y].pion)
                        {//Case amie

                                masque[i][j].occupe = BLOK;
                                masque[i][j].poids=-1;
                        }
                        else if(plateau[org_mask.x+i][org_mask.y+j].pion!=NULL && plateau[org_mask.x+i][org_mask.y+j].pion-
>type_pion!=joueur_actuel)
                        {
                                switch(plateau[org_mask.x+i][org_mask.y+j].pion->type_pion)
                                {
                                        case GR :
                                                masque[i][j].occupe = NEUTRE;
                                                break;
                                        default :
                                                masque[i][j].occupe = OPPOSANT;
                                                break;
                                }
                                masque[i][j].poids=-1;
```

```
                        }
                        else
                        {//Case libre

                                masque[i][j].occupe=0;
                                masque[i][j].poids=-1;
                        }
                }
    }

        coord pos;
                pos.x = VITESSE_MAX;
                pos.y = VITESSE_MAX;

   parcours(masque,pos,0,longueur);
}

void parcours(S_masque masque[MASQUE_MAX][MASQUE_MAX], coord pos, int cpt, int longueur)
{
        /*
                Fonction récursive qui place le nombre de coups minimaux qu'il faut pour parvenir
                à une case sur toutes les cases du masque.
        */
        int i;
        coord next_pos;

    if(masque[pos.x][pos.y].occupe!=BLOK || cpt==0) //si la case n'est pas bloquee ou amie

        {
                masque[pos.x][pos.y].poids=cpt;
                cpt++;

                if(masque[pos.x][pos.y].occupe==0 && cpt<=longueur)
                        for(i=0; i<4; i++)
                        {
                                next_pos = get_pos(pos, i);
                                if(poids(masque, next_pos)>cpt || poids(masque, next_pos)==-1)
                                        parcours(masque, next_pos, cpt, longueur);
                        }

        }
}

coord get_pos(coord pos, int pos_rel)
{
        /*
                Renvoie les coordonées de la case
                        au-dessus
                        en-dessous
                        en haut
                        en abas
                pour la fonction de parcours
        */
        switch (pos_rel) {
                case 0:
                        pos.x--;
                        break;
                case 1:
                        pos.y--;
                        break;
                case 2:
                        pos.x++;
                        break;
                default:
                        pos.y++;
                        break;
        }

        return pos;
}
```

```
int poids(S_masque masque[MASQUE_MAX][MASQUE_MAX], coord pos)
{
	/*
		Retourne le poids d'une case d'un masque si cette case fait
		bien partie du masque
	*/
	if(pos.x>=0 && pos.x<MASQUE_MAX && pos.y>=0 && pos.y<MASQUE_MAX)
		return masque[pos.x][pos.y].poids;
	else
		return 0;

}

coord origine_masque(coord centre_masque)
{
	//Retourne l'origine du masque

	   centre_masque.x -= VITESSE_MAX;
	   centre_masque.y -= VITESSE_MAX;

	   return centre_masque;
}

coord masque2tab(coord convert, coord centre_masque)
{
	//Convertit les coordonnées de masque en coordonnées de tableau

	coord org_masque = origine_masque(centre_masque);

	convert.x += org_masque.x;
	convert.y += org_masque.y;

	return convert;
}

coord tab2masque(coord convert, coord centre_masque)
{
	//Convertit lezs coordonnées de tableau en coordonnées de masque

	coord org_masque = origine_masque(centre_masque);

	convert.x -= org_masque.x;
	convert.y -= org_masque.y;

	return convert;
}

void chemin(S_masque masque[MASQUE_MAX][MASQUE_MAX], S_chemin** L_chemin, coord case_ini, coord fin)
{
	//Remplit une liste chaînée contenant les coordonnées successives du point de départ au point d'arrivée

	S_chemin* act = (S_chemin*) malloc(sizeof(S_chemin));
		act->etape = fin;
		act->suiv = NULL;

	fin = tab2masque(fin, case_ini);
	if(masque[fin.x][fin.y].poids>0)
	{
		if(masque[fin.x-1][fin.y].poids>=0 && masque[fin.x-1][fin.y].occupe==0 && masque[fin.x-1][fin.y].poids <
masque[fin.x][fin.y].poids)
			fin.x--;
		else if(masque[fin.x+1][fin.y].poids>=0 && masque[fin.x+1][fin.y].occupe==0 && masque[fin.x+1][fin.y].poids <
masque[fin.x][fin.y].poids)
			fin.x++;
		else if(masque[fin.x][fin.y-1].poids>=0 && masque[fin.x][fin.y-1].occupe==0 && masque[fin.x][fin.y-1].poids <
masque[fin.x][fin.y].poids)
			fin.y--;
		else if(masque[fin.x][fin.y+1].poids>=0 && masque[fin.x][fin.y+1].occupe==0 && masque[fin.x][fin.y+1].poids <
masque[fin.x][fin.y].poids)
```

```
                        fin.y++;

                chemin(masque, L_chemin, case_ini, masque2tab(fin, case_ini));
        }

        enfiler_element_chemin(L_chemin, act);
}

/*FONCTIONS DE LISTE ADAPTEES AU TYPE S_CHEMIN, CF LISTES.CPP*/

void enfiler_element_chemin(S_chemin** L_deb, S_chemin* element)
{
        S_chemin* actuel;

        if(*L_deb == NULL)
                *L_deb = element;
        else
        {
                actuel = *L_deb;
                while(actuel->suiv != NULL)
                        actuel = actuel->suiv;
                actuel->suiv = element;
        }
}

void supprimer_chemin(S_chemin* element)
{
        if(element != NULL)
        {
                supprimer_chemin(element->suiv);
                free(element);
        }
}
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : affichage_sdl.h
    Dernière Modification : 16/05/2010

    Description :
        Header
*/
#ifndef AFFICHAGE_SDL
        #define AFFICHAGE_SDL

        #include "structures.h"
        #include "jeu.h"
        #include "outils_sdl.h"
        #include "masque.h"
        #include "tableau_jeu.h"


    #define NB_SURFACE 32
    #define NB_POS 4
    #define NB_CHAR 11
    #define RESOLUTION 32

    #define LARGEUR_FENETRE 620
    #define HAUTEUR_FENETRE 730

    #define POS_PLATEAU_X 9
    #define POS_PLATEAU_Y 124


    /*
        DEFINITION DES CONSTANTES UTILES AU PARCOURS DES TABLEAU DE SURFACE ET DE POSITIONS
    */
    #define ECRAN 0
    #define EN_TETE 1
    #define PLATEAU 2
    #define QUADRILLAGE_H 3
    #define QUADRILLAGE_V 4

    #define GB1_1 5
    #define GB1_2 6
    #define GB1_3 7
    #define GB2_1 8
    #define GB2_2 9
    #define GB2_3 10
    #define GB3_1 11
    #define GB3_2 12
    #define GB3_3 13
    #define VIR1 14
    #define VIR2 15
    #define VIR3 16
    #define E_VIRUS 17
    #define E_GR 18
    #define BLOQ 19
    #define CTRL 20
    #define GROUGE1 21
    #define GROUGE2 22

    #define JOUEUR_GB 23
    #define JOUEUR_VIR 24

    #define MASQUE 25
    #define FOND_ECRAN 26
    #define BUILDER 27

    #define TOTO 3
    #define TOTO1 28
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : affichage_sdl.cpp
    Dernière Modification : 16/05/2010

    Description :
        Contient toutes les fonctions et procédures nécessaire à l'affichage de la SDL de partie()
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_ttf.h>
#include <FMOD/fmod.h>

#include "affichage_sdl.h"



void demander_coord(coord* case_a_jouer, int choix, SDL_Surface* tableau_surface[], SDL_Rect tableau_position[], int *quitter)
/* PS : case_a_jouer, quitter
    Aquiert les coordonnées de la case de départ ou d'arrivée selon si choix vaut 0 ou 1, modifie quitter si l'utilisateur
clique sur la croix de "fermer"
*/
{
        int continuer = 1;
    SDL_Event event;


        switch(choix)
        {
                case 1: // Cas où l'on veut la case de départ
                        while(continuer)
                        {
                            SDL_WaitEvent(&event); // Attend qu'un évenement se produise

                            switch(event.type)
                {
                    case SDL_QUIT: // Si l'utilisateur clique sur "fermer"
                        *quitter=1;
                        exit(EXIT_SUCCESS);
                        break;

                    case SDL_MOUSEBUTTONDOWN : // Si l'utilisateur clique bas
                        if((event.button.x>=POS_PLATEAU_X+1 && event.button.x<=POS_PLATEAU_X+601) &&
(event.button.y>=POS_PLATEAU_Y && event.button.y<=POS_PLATEAU_Y+601))// Si l'utilisateur a cliquer dans le plateau de jeu
                        {
                           continuer=0;
                           case_a_jouer->x=sdl2cons(event.button.x,'h'); // Conversion des pixels en unité de tableau
                           case_a_jouer->y=sdl2cons(event.button.y,'v');
                        }
                        break;
                }
                        }
        break;

                default: // Cas où l'on veut la case d'arrivée (même principe que ci-dessus)
                        while(continuer)
                        {
                            SDL_WaitEvent(&event);
                            switch(event.type)
                            {
                                case SDL_QUIT:
                        exit(EXIT_SUCCESS);
                        break;
```

```
                    case SDL_MOUSEBUTTONDOWN:
                        if((event.button.x>=POS_PLATEAU_X+1 && event.button.x<=POS_PLATEAU_X+601) &&
(event.button.y>=POS_PLATEAU_Y && event.button.y<=POS_PLATEAU_Y+601))
                        {
                            continuer=0;
                            case_a_jouer->x=sdl2cons(event.button.x, 'h');
                            case_a_jouer->y=sdl2cons(event.button.y, 'v');
                        }
                        break;
                            }
                        }
        break;
        }

}

void afficher_joueur(const int joueur, SDL_Surface* tableau_surface[], SDL_Rect tableau_position[])
/* PS : /
    Affiche le joueur actuel dans l'en-tête (équipe + image) ou l'en-tête de l'éditeur.
*/
{
    tableau_position[0].x=0; // Définition de la position de l'en-tête
    tableau_position[0].y=0;

    switch(joueur)
        {
                case GLOBULE:
            SDL_BlitSurface(tableau_surface[JOUEUR_GB], NULL, tableau_surface[EN_TETE], &tableau_position[0]); // Collage de
l'image joueur_gb sur l'en-tête
                        break;
                case VIRUS:
                        SDL_BlitSurface(tableau_surface[JOUEUR_VIR], NULL, tableau_surface[EN_TETE], &tableau_position[0]); //
Collage de l'image joueur_vir sur l'en-tête
                        break;
        default :
            SDL_BlitSurface(tableau_surface[BUILDER], NULL, tableau_surface[EN_TETE], &tableau_position[0]); // Collage de
l'image en_tete_builder sur l'en-tête
            break;

        }

        SDL_BlitSurface(tableau_surface[EN_TETE], NULL, tableau_surface[ECRAN], &tableau_position[0]);  // Collage de l'en-tête
sur l'écran
}


void afficher_plateau(S_case plateau[TAILLE_PLAT][TAILLE_PLAT], SDL_Surface* tableau_surface[], SDL_Rect tableau_position[],
coord coord_actuel, int choix)
/* PS : /
    Affiche la grille de jeu et les pions qui y figurent, choix permet d'afficher ou non le pion que l'on est en train de jouer.
Celà permet de ne pas avoir l'affichage
    de la position initial du pion lors de son déplacement graphique avec la fonction de chemin.
*/
{
        int pion,i,j; // Nécessaire au parcours du tableau
        coord coordonnees; // Nécessaire à l'utilisation de certaines fonctions

        tableau_position[0].x=0;  // Définition de la position de collage
        tableau_position[0].y=0;
        SDL_BlitSurface(tableau_surface[FOND_ECRAN], NULL, tableau_surface[PLATEAU], &tableau_position[0]); // Collage de
l'image de fond d'écran sur le plateau

    /* AFFICHAGE DES PIONS DU PLATEAU DE JEU */
        for(j=0; j<TAILLE_PLAT; j++)
        {
           for(i=0; i<TAILLE_PLAT; i++)
           {
               if(choix == 0 || (choix==1 && coord_actuel.x!=i || coord_actuel.y!=j))// Si l'on veut afficher la totalité du
plateau, ou si les coordonnes de la case actuel ne sont pas celles de la position initiale du pion
```

```
                {
                 coordonnees.x=i;
                 coordonnees.y=j;

                 tableau_position[0].x=cons2sdl(i);
                 tableau_position[0].y=cons2sdl(j);

                 pion=-1;

                 pion=contenu_case(coordonnees,plateau);
                 if(pion!=-1) // S'il y a quelque chose dans la case
                     SDL_BlitSurface(tableau_surface[pion], NULL, tableau_surface[PLATEAU], &tableau_position[0]);
                }
            }
        }

        SDL_BlitSurface(tableau_surface[PLATEAU], NULL, tableau_surface[ECRAN], &tableau_position[PLATEAU]); // Collage du
plateau sur l'écran
}

void afficher_masque(S_masque masque[MASQUE_MAX][MASQUE_MAX],SDL_Surface* tableau_surface[], SDL_Rect tableau_position[], coord
case_actuelle)
/* PS : /
    Affiche le masque, c'est à dire les déplacements autorisé en surbrillance
*/
{
    int i,j; // Nécessaire au parcours du tableau
    for(j=0;j<MASQUE_MAX;j++)
    {
        for(i=0;i<MASQUE_MAX;i++)
            {
                            if(masque[i][j].poids>0) // Si la case est atteignable (exclu la case initiale du déplacement du
pion)
                            {
                                tableau_position[0].x=cons2sdl(case_actuelle.x-VITESSE_MAX+i);// On doit convertir les
coordonées du tableau en coordonnées de pixels
                                tableau_position[0].y=cons2sdl(case_actuelle.y-VITESSE_MAX+j);
                                SDL_BlitSurface(tableau_surface[MASQUE], NULL, tableau_surface[PLATEAU],
&tableau_position[0]); // Collage de la surbrillance sur le plateau
                            }
                    }
        }

        SDL_BlitSurface(tableau_surface[PLATEAU], NULL, tableau_surface[ECRAN], &tableau_position[PLATEAU]); // Collage du
plateau sur l'écran
        SDL_Flip(tableau_surface[ECRAN]); // Mise à jour de l'écran
}


int selection_mode_jeu(SDL_Surface* tableau_surface[])
/* PS : mode_de_jeu (Humain VS ia ou Humain VS Humain)
    Affiche un écran permettant de sélectioner le mode de jeu désiré.
    Cette fonction est isolée des autres et ne sert qu'une seule fois. Elle a donc un tableau de surface propre.
*/
{
    int i, fond=0, ia=1, human=2, continuer=2;

    SDL_Surface* tableau_s[3];              // Création du tableau contenant les surfaces
    SDL_Rect pos_fond, pos_ia, pos_human;   // Création des position
    SDL_Event event;                        // Création d'un évenement
    FSOUND_SAMPLE *robot_voice = NULL;      // Création d'un son

    /* INITIALISATION DES IMAGES ET SON */
    TTF_Font *police = NULL;                // Définition de la police
    police = TTF_OpenFont("polices/police_menu.ttf", 28);
    SDL_Color blanc = {255,255,255};

    tableau_s[fond] = SDL_LoadBMP("pic/fonds/fond_mode_jeu.bmp");
    tableau_s[ia] = TTF_RenderText_Blended(police, "Humain VS Ordinateur",blanc);
    tableau_s[human] = TTF_RenderText_Blended(police, "Humain VS Humain",blanc);
```

```
    robot_voice = FSOUND_Sample_Load(FSOUND_FREE, "music/ready.mp3", 0, 0, 0);


    /* COLLAGE DES IMAGES ET TEXTES AU BON EMPLACEMENT */
    pos_fond.x=0;
    pos_fond.y=0;
    SDL_BlitSurface(tableau_s[fond], NULL, tableau_surface[ECRAN], &pos_fond);

    pos_ia.x=(LARGEUR_FENETRE-tableau_s[ia]->w)/2;
    pos_ia.y=100;
    SDL_BlitSurface(tableau_s[ia], NULL, tableau_surface[ECRAN], &pos_ia);


    pos_human.x=(LARGEUR_FENETRE-tableau_s[human]->w)/2;
    pos_human.y=200;
    SDL_BlitSurface(tableau_s[human], NULL, tableau_surface[ECRAN], &pos_human);

    SDL_Flip(tableau_surface[ECRAN]);


    while (continuer==2)
    {
        SDL_WaitEvent(&event);
        switch(event.type)
        {
            case SDL_QUIT:
                continuer =-1;
                break;
            case SDL_MOUSEBUTTONDOWN:
                if((event.button.x>=pos_ia.x && event.button.x<=pos_ia.x+tableau_s[ia]->w) && (event.button.y>=pos_ia.y &&
event.button.y<=pos_ia.y+tableau_s[ia]->h)) // Si l'utilisateur clique sur "Humain VS IA"
                    {
                        continuer=IA;
                        FSOUND_PlaySound(FSOUND_FREE, robot_voice); // Jouer le son
                        SDL_Delay(500); // Attendre le temps que le son se finisse
                    }

                else if((event.button.x>=pos_human.x && event.button.x<=pos_human.x+tableau_s[human]->w) &&
(event.button.y>=pos_human.y && event.button.y<=pos_human.y+tableau_s[human]->h))// Si l'utilisateur clique sur "Humain VS IA"
                    continuer=VS;
                break;
        }
    }

    /* LIBERATION DES SURFACES ET SON */
    for(i=0;i<3;i++)
        SDL_FreeSurface(tableau_s[i]);
    FSOUND_Sample_Free(robot_voice);

    SDL_FillRect(tableau_surface[ECRAN], NULL, SDL_MapRGB(tableau_surface[ECRAN]->format, 100, 100, 100)); // On efface l'écran.


    return continuer; // retourne IA ou VS
}

void afficher_chemin(S_masque masque[MASQUE_MAX][MASQUE_MAX], S_case plateau[TAILLE_PLAT][TAILLE_PLAT], coord case_ini, coord
case_fin, SDL_Surface* tableau_surface[], SDL_Rect tableau_position[])
/* PS : /
    Affiche le chemin parcouru par un pion lorsqu'il se déplace de case_ini à case_fin
*/
{
    S_chemin* L_chemin=NULL;
    S_chemin* parcours=NULL; // Sert au parcours de la liste
    SDL_Rect pos, pos_fond;
    SDL_Surface* pion,* fond;
    int i;

    pion = tableau_surface[contenu_case(case_ini, plateau)]; // pion devient la surface du pion que l'on veut déplacer
    afficher_plateau(plateau, tableau_surface, tableau_position, case_ini, 1);
```

```
    fond = tableau_surface[PLATEAU]; // Stockage de l'image du plateau précèdement affiché pour ne pas executer
afficher_plateau() (qui est très couteux) plein de fois

    pos_fond.x=POS_PLATEAU_X;
    pos_fond.y=POS_PLATEAU_Y;

    chemin(masque, &L_chemin, case_ini, case_fin); // Création de la liste de coordonnées du chemin

    parcours=L_chemin; // Initialisation de 'parcours' au premier élèment de la liste

    while(parcours->suiv!=NULL) // Tant qu'il reste un élèment après l'actuel
    {
        pos.x=POS_PLATEAU_X + cons2sdl(parcours->etape.x); // Définition de la position de l'image à coller au début de la case
actuelle de la liste
        pos.y=POS_PLATEAU_Y + cons2sdl(parcours->etape.y);

        /* Pour un déplacement plus fluide, on avance pixel par pixel tant que l'on est dans la case actuelle définie par ci-
dessus */
        for(i=0;i<20;i++) // Tant que l'on est dans les coordonnées de l'élèment actuel de la liste
        {
            if(parcours->suiv->etape.x==parcours->etape.x+1)
               pos.x++;
            else if(parcours->suiv->etape.x==parcours->etape.x-1)
                pos.x--;
            if(parcours->suiv->etape.y==parcours->etape.y+1)
                pos.y++;
            else if(parcours->suiv->etape.y==parcours->etape.y-1)
                pos.y--;
            SDL_BlitSurface(fond, NULL, tableau_surface[ECRAN], &pos_fond);  // Collage du plateau
            SDL_BlitSurface(pion, NULL, tableau_surface[ECRAN], &pos); // Collage de l'image du pion
            SDL_Flip(tableau_surface[ECRAN]); // Màj de l'écran
            SDL_Delay(5); // Attend ~15ms (granularité du temps dans le CPU) pour que l'affichage ne se fasse pas trop vite
        }


    parcours=parcours->suiv; // Passage à l'élèment suivant de la liste
    }
}

int contenu_case(coord coordonnees, S_case plateau[TAILLE_PLAT][TAILLE_PLAT])
/* PS : retour (qui prend la valeur de la constante correspondant au contenu de la case).
    Permet de savoir quel pion se trouve dans une case du tableau de jeu.
*/
{
    int retour, i, j;
    i=coordonnees.x;
    j=coordonnees.y;

    switch(plateau[i][j].type)
    {

        case NORMALE:

            if(plateau[i][j].pion!=NULL) // S'il y a un pion dans cette case
            {
                if((plateau[i][j].pion)->type_pion == GLOBULE && coord_eq(case_caract(plateau,coordonnees),coordonnees))// Si
c'est un globule blanc et s'il s'agit de sa case caractéristique
                {
                    switch((plateau[i][j].pion)->vitesse) // Différentes couleurs selon la vitesse du globule blanc
                    {
                        case 0: // Si le globule blanc est paralysé
                                switch((plateau[i][j].pion)->niveau) // Affichage du pion selon son niveau
                                {
                                    case 1:
                                            retour =  GB1_1; break;
                                    case 2:
                                            retour =  GB2_1; break;
                                    case 3:
                                            retour =  GB3_1; break;
                                }
```

```
                    break;

                case VITESSE_INI_GB: // Si le globule blanc a sa vitesse maximum
                        switch((plateau[i][j].pion)->niveau) // Affichage du pion selon son niveau
                        {
                            case 1:
                                    retour =  GB1_3; break;
                            case 2:
                                    retour =  GB2_3; break;
                            case 3:
                                    retour =  GB3_3; break;
                        }
                    break;
                default: // Si sa vitesse est comprise ente les deux
                        switch((plateau[i][j].pion)->niveau) // Affichage du pion selon son niveau
                        {
                            case 1:
                                    retour =  GB1_2; break;
                            case 2:
                                    retour =  GB2_2; break;
                            case 3:
                                    retour =  GB3_2; break;
                        }
                    break;
            }
        }
        else if((plateau[i][j].pion)->type_pion == VIRUS) // Si le pion est un virus
            {
                switch((plateau[i][j].pion)->niveau) // Affichage du pion selon sa facultée
                {
                    case 1 :
                        retour =  VIR1; break;
                    case 2 :
                        retour =  VIR2; break;
                    case 3 :
                        retour =  VIR3; break;
                }
            }

        else if((plateau[i][j].pion)->type_pion == GR) // Sinon, c'est forcement un globule rouge
            {
                srand(time(0));
                retour = rand()%2 == 0 ? GROUGE1 : GROUGE2;
            }

        else // Si la case n'est pas la case caractéristique du globule blanc
            retour=-1;

        break;
    }
    else // S'il n'y a rien dans la case
        retour = -1;
    break;

case BLOQUEE: // Si c'est une case bloquée
    retour =  BLOQ;
    break;
case CONTROLE: // Si c'est un point de contrôle
    retour =  CTRL;
    break;
case ENTREE_VIRUS: //  Si c'est un point d'entrée de virus
    retour =  E_VIRUS;
    break;
case ENTREE_GR: // Si c'est un point d'entrée de globule rouge
    retour =  E_GR;
    break;
default :
    retour=-1;
    break;
}
```

```
    return retour;
}

void afficher_vainqueur(S_pion* tab_listes[DIM_TAB_LISTES], SDL_Surface* tableau_surface[], int coups)
/* PS : /
    Affiche le vainqueur à la fin de la partie
*/
{   SDL_Surface* fond;
    SDL_Rect pos;
    SDL_Event event;
    int continuer=1;

    pos.x=0;
    pos.y=0;


    if(tab_listes[GLOBULE]==NULL || cpt_controle(0)>=COUPS_CONTROLE) // Si les virus gagnent
        {fond = SDL_LoadBMP("pic/fonds/virus_win.bmp"); // Chargement de l'image du vainqueur
        fprintf(stderr,"\nvictoire virus OK");}

    else if(tab_listes[VIRUS]==NULL || coups>=COUPS_VICTOIRE_GB)
        {fond = SDL_LoadBMP("pic/fonds/gb_win.bmp");
        fprintf(stderr,"\nvictoire gb OK");}
    else
        fprintf(stderr, "Erreur : Aucun gagnant");

    SDL_BlitSurface(fond, NULL, tableau_surface[ECRAN], &pos);
    SDL_Flip(tableau_surface[ECRAN]);

    /* ATTEND QUE L'UTILISATEUR CHOISISSE DE PASSER */

    while(continuer)
    {
        SDL_WaitEvent(&event);
        switch(event.type)
        {
            case SDL_KEYDOWN :
                if(event.key.keysym.sym==SDLK_ESCAPE)
                    continuer=0;
            break;
        }
    }

    SDL_FreeSurface(fond); // Libération de la surface
}




void afficher_toto(SDL_Surface* tableau_surface[], SDL_Rect tableau_position[])
{




    switch(cpt_controle(0))
    {
        case 0:
        SDL_BlitSurface(tableau_surface[TOTO1], NULL, tableau_surface[ECRAN], &tableau_position[TOTO]);
        break;

        case 1:
        SDL_BlitSurface(tableau_surface[TOTO2], NULL, tableau_surface[ECRAN], &tableau_position[TOTO]);
        break;

        case 2:
        SDL_BlitSurface(tableau_surface[TOTO3], NULL, tableau_surface[ECRAN], &tableau_position[TOTO]);
        break;
```

```
        case 3:
        SDL_BlitSurface(tableau_surface[TOTO4], NULL, tableau_surface[ECRAN], &tableau_position[TOTO]);
        break;

    }
}
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : outils_sdl.h
    Dernière Modification : 16/05/2010

    Description :
        Header
*/

#ifndef OUTILS_SDL_H_INCLUDED
#define OUTILS_SDL_H_INCLUDED

#include "structures.h"
#include "affichage_sdl.h"

int cons2sdl (int x);
int sdl2cons (int x, char choix);
void initialiser_SDL(SDL_Surface* tableau_surface[],SDL_Rect tableau_position[]);
void liberer_SDL(SDL_Surface* tableau_surface[]);

#endif // OUTILS_SDL_H_INCLUDED
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : outils_sdl.cpp
    Dernière Modification : 16/05/2010

    Description :
        Contient les fonctions utiles de la sdl (conversions, initialisation et libération)
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_ttf.h>


#include "outils_sdl.h"




int cons2sdl (int x)
/* PS : La valeur en pixel
    Converti la valeur d'une coordonnée du plateau de jeu en pixels
*/
{
    return 20*x+1;
}

int sdl2cons (int x, char choix)
/* PS : La valeur en coordonnée du plateau de jeu
    Converti une valeur en pixel. Il y a deux cas possible, si la fenetre est paramétrée differement. On laisse la
fonctionnalité en cas d'évolutivité.
*/
{
    if(choix=='h')
        return (int) floor((x-POS_PLATEAU_X)/20);
    else
        {
            return (int)  floor((x-POS_PLATEAU_Y)/20);
        }
}

void initialiser_SDL(SDL_Surface* tableau_surface[],SDL_Rect tableau_position[])
/* PS : /
    Initialise et définie les surfaces et positions dont la sdl a besoin.
*/
{
    int i;

/* INITIALISATION DES SURFACES */
    for(i=0;i<NB_SURFACE;i++)
        tableau_surface[i]=NULL;


/* DEFINITION DES SURFACES */
    tableau_surface[ECRAN] = SDL_SetVideoMode(LARGEUR_FENETRE, HAUTEUR_FENETRE, RESOLUTION, SDL_HWSURFACE);// creation fenetre
    SDL_FillRect(tableau_surface[ECRAN], NULL, SDL_MapRGB(tableau_surface[ECRAN]->format, 100, 100, 100));

    tableau_surface[EN_TETE] = SDL_CreateRGBSurface(SDL_HWSURFACE, 620, 120, RESOLUTION, 0, 0, 0, 0);
        SDL_FillRect(tableau_surface[EN_TETE], NULL, SDL_MapRGB(tableau_surface[ECRAN]->format, 70, 33, 220));
        tableau_position[EN_TETE].x=0;
        tableau_position[EN_TETE].y=0;

    tableau_surface[PLATEAU] = SDL_CreateRGBSurface(SDL_HWSURFACE, 602, 602, RESOLUTION, 0, 0, 0, 0);
        SDL_FillRect(tableau_surface[PLATEAU], NULL, SDL_MapRGB(tableau_surface[ECRAN]->format, 255, 255, 255));
```

```
    tableau_position[PLATEAU].x=POS_PLATEAU_X;
    tableau_position[PLATEAU].y=POS_PLATEAU_Y;

tableau_surface[QUADRILLAGE_H] = SDL_CreateRGBSurface(SDL_HWSURFACE, 602, 2, RESOLUTION, 0, 0, 0, 0);
    SDL_FillRect(tableau_surface[QUADRILLAGE_H], NULL, SDL_MapRGB(tableau_surface[ECRAN]->format, 0, 0, 0));
    SDL_SetAlpha(tableau_surface[QUADRILLAGE_H], SDL_SRCALPHA, 200);

tableau_surface[QUADRILLAGE_V] = SDL_CreateRGBSurface(SDL_HWSURFACE, 2, 602, RESOLUTION, 0, 0, 0, 0);
    SDL_FillRect(tableau_surface[QUADRILLAGE_V], NULL, SDL_MapRGB(tableau_surface[ECRAN]->format, 0, 0, 0));
    SDL_SetAlpha(tableau_surface[QUADRILLAGE_V], SDL_SRCALPHA, 200);


tableau_surface[FOND_ECRAN] = SDL_LoadBMP('pic/fonds/fond_ecran_jeu.bmp');

/* COLLAGE DU QUADRILLAGE DU PLATEAU DE JEU */
tableau_position[0].y=0;
for(i=0;i<=TAILLE_PLAT+4;i++)
{
    tableau_position[0].x=i*20;
    SDL_BlitSurface(tableau_surface[QUADRILLAGE_V], NULL, tableau_surface[FOND_ECRAN], &tableau_position[0]);
}

tableau_position[0].x=0;
for(i=0;i<=TAILLE_PLAT+4;i++)
{
    tableau_position[0].y=i*20;
    SDL_BlitSurface(tableau_surface[QUADRILLAGE_H], NULL, tableau_surface[FOND_ECRAN], &tableau_position[0]);
}

/* CHARGEMENT DES IMAGES */
tableau_surface[GB1_1] = IMG_Load('pic/pions/gb1_1.png');
tableau_surface[GB1_2] = IMG_Load('pic/pions/gb1_2.png');
tableau_surface[GB1_3] = IMG_Load('pic/pions/gb1_3.png');

tableau_surface[GB2_1] = IMG_Load('pic/pions/gb2_1.png');
tableau_surface[GB2_2] = IMG_Load('pic/pions/gb2_2.png');
tableau_surface[GB2_3] = IMG_Load('pic/pions/gb2_3.png');

tableau_surface[GB3_1] = IMG_Load('pic/pions/gb3_1.png');
tableau_surface[GB3_2] = IMG_Load('pic/pions/gb3_2.png');
tableau_surface[GB3_3] = IMG_Load('pic/pions/gb3_3.png');

tableau_surface[VIR1] = IMG_Load('pic/pions/virus1.png');
tableau_surface[VIR2] = IMG_Load('pic/pions/virus2.png');
tableau_surface[VIR3] = IMG_Load('pic/pions/virus3.png');
tableau_surface[E_VIRUS] = SDL_LoadBMP('pic/pt_clefs/entree_virus.bmp');

tableau_surface[GROUGE1] = IMG_Load('pic/pions/gr1.png');
tableau_surface[GROUGE2] = IMG_Load('pic/pions/gr2.png');

tableau_surface[E_GR] = SDL_LoadBMP('pic/pt_clefs/entree_gr.bmp');

tableau_surface[CTRL] = SDL_LoadBMP('pic/pt_clefs/pt_controle.bmp');

tableau_surface[BLOQ] = SDL_LoadBMP('pic/pt_clefs/bloquee.bmp');

tableau_surface[JOUEUR_GB] = SDL_LoadBMP('pic/en_tete/joueur_gb.bmp');
tableau_surface[JOUEUR_VIR] = SDL_LoadBMP('pic/en_tete/joueur_vir.bmp');

tableau_surface[MASQUE] = SDL_LoadBMP('pic/pt_clefs/masque.bmp');
SDL_SetAlpha(tableau_surface[MASQUE], SDL_SRCALPHA, 170);

tableau_surface[BUILDER] = SDL_LoadBMP('pic/en_tete/en_tete_builder.bmp');

tableau_surface[TOTO1] = IMG_Load('pic/toto/toto1.png');
tableau_surface[TOTO2] = IMG_Load('pic/toto/toto2.png');
tableau_surface[TOTO3] = IMG_Load('pic/toto/toto3.png');
tableau_surface[TOTO4] = IMG_Load('pic/toto/toto4.png');
tableau_position[TOTO].x = 10;
tableau_position[TOTO].y = 10;
```

```
}

void liberer_SDL(SDL_Surface* tableau_surface[])
/* PS : /
    Libère l'éspace mémoire dont la sdl a eu besoin
*/
{
    int i;

    for(i=0;i<NB_SURFACE;i++)
    {
        SDL_FreeSurface(tableau_surface[i]);
    }

}
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : extraction_fichier.h
    Dernière Modification : 16/05/2010

    Description :
        Header
*/
#ifndef EXTRACTION_FICHIER
        #define EXTRACTION_FICHIER

        #include "structures.h"
        #include "listes.h"

        #define ADDR_LEN 300
        #define NB_TYPES_A_EXTRAIRE 6

        void generer_plateau(S_case plateau_jeu[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], coord
tab_points_clefs[DIM_TAB_POINTS_CLEFS]);
        void ligne2pion(int ligne, int* pion, int* type);

#endif
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : extraction_fichier.cpp
    Dernière Modification : 16/05/2010

    Description :
        Contient la fonction generer_plateau, qui extrait
    les informations formatées contenues dans map.txt.
    Ces informations donnent les positions des différents
    éléments d'un plateau de jeu donné.
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "extraction_fichier.h"

void generer_plateau(S_case plateau_jeu[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], coord
tab_points_clefs[DIM_TAB_POINTS_CLEFS])
{
    /* DECLARATION DES VARIABLES */

    //tableau qui va stocker les adresses des pions
        coord element_a_placer;

    //La variable de fichier
        FILE* map = NULL;

    char c;     //utile au parcours du fichier
    int i, j, extract, pion, type;

        //On remplit le plateau avec les valeurs par défaut
        for(i=0; i<TAILLE_PLAT; i++)
        {
                for(j=0; j<TAILLE_PLAT; j++)
                {
                        plateau_jeu[i][j].pion = NULL;
                        plateau_jeu[i][j].type = NORMALE;
                }
        }


    /* OUVERTURE DU FICHIER */

    //Création du nom du fichier (en absolu)
    char addr_map[ADDR_LEN]; //Initialisation variable

        if(IS_MAC)
    {
        /*
            Cette partie était utile pour compiler sous mac, vu les
            différences de gestion des fichiers dans ce système
        */
        strcat(addr_map, "/Users/whitecoyote/Documents/Etudes/EPF/EPF-1A/Algo/Immuno-Wars/Projet_extraction_map/");   //Nom de
fichier
        strcat(addr_map, "map.txt");
        map = fopen(addr_map, "r");
    }
    else
        map = fopen("map/map.txt", "r");



    if(map != NULL)
    {
        for(extract=0; extract<NB_TYPES_A_EXTRAIRE; extract++)
        {
```

```
            //On se place après le % (séparateur) suivant
            do c = fgetc(map); while(c != '#' && c != EOF);
            fgetc(map);

            //En fonction de la ligne dans le fichier, on trouve le type d'élément à placer
            ligne2pion(extract, &pion, &type);

            i=0;
            do
            {
                c = fgetc(map);

                if(c == '(')
                {
                                    //On recupere les coordonnees de l'element a placer
                                    fscanf(map, "%d;%d)", &(element_a_placer.x), &(element_a_placer.y));

                                    //On les ré-indexe correctement
                                    element_a_placer.x--;
                                    element_a_placer.y--;

                                    //On remplit le tableau et les listes
                                    if(pion != CASE)
                                    {
                                            //Création du pion dans la liste et allocation.
                                            plateau_jeu[element_a_placer.x][element_a_placer.y].pion = nouveau_pion(pion,
tab_listes);
                                            plateau_jeu[element_a_placer.x][element_a_placer.y].pion->pos =
element_a_placer;
                                    }
                                    if(type != NORMALE)
                                    {
                                            plateau_jeu[element_a_placer.x][element_a_placer.y].type = type;
                                            tab_points_clefs[type] = element_a_placer;
                                    }

                                    i++;
                }
            }while(c != '\n' && c != EOF);
        }

        fclose(map);
    }

}

void ligne2pion(int ligne, int* pion, int* type)
{
    //En fonction de la ligne, on choisit quel item il faut placer sur le plateau et dans les listes.
    switch(ligne)
    {
        case 0:
            *pion = GLOBULE;
            *type = NORMALE;
            break;
        case 1:
            *pion = VIRUS;
            *type = NORMALE;
            break;
        case 2:
            *pion = CASE;
            *type = BLOQUEE;
            break;
        case 3:
            *pion = CASE;
            *type = CONTROLE;
            break;
        case 4:
            *pion = CASE;
            *type = ENTREE_VIRUS;
```

```
            break;
        case 5:
            *pion = CASE;
            *type = ENTREE_GR;
            break;
        default:
            *pion = CASE;
            *type = NORMALE;
            break;
    }
}
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : ia.h
    Dernière Modification : 16/05/2010

    Description :
        Header
*/

#ifndef FILE_IA
        #define FILE_IA

        #define COUPS_GB 3
        #define COUPS_VIRUS = (COUPS_GB)

        #define PROFONDEUR_MINMAX 3


        #include "structures.h"
        #include "masque.h"
        #include "listes.h"
        #include "tableau_jeu.h"

        struct S_deplacement
        {
                coord ini;
                coord fin;
        }; typedef struct S_deplacement S_deplacement;

        struct L_coups
        {
                coord ini;
                coord fin;
                struct L_coups* suiv;
        }; typedef struct L_coups L_coups;

        void creer_liste_filtree(S_pion** L_deb_a_jouer, S_case plateau_jeu[TAILLE_PLAT][TAILLE_PLAT], S_pion*
tab_listes[DIM_TAB_LISTES], int joueur);
        int is_movable(S_case plat[TAILLE_PLAT][TAILLE_PLAT], S_pion* pion);
        void meilleur_coup(S_case plateau_jeu[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], coord
tab_points_clefs[DIM_TAB_POINTS_CLEFS], coord* depart, coord* arrivee);
        void smart_move(S_pion* pion, S_case plateau_jeu[TAILLE_PLAT][TAILLE_PLAT], int* note_max, coord* coup_tmp, coord targ);

#endif
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : ia.cpp
    Dernière Modification : 16/05/2010

    Description :
        Contient les fonctions et procédures nécéssaires au bon fonctionement de l'IA
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "ia.h"

void meilleur_coup(S_case plateau_jeu[TAILLE_PLAT][TAILLE_PLAT], S_pion* tab_listes[DIM_TAB_LISTES], coord
tab_points_clefs[DIM_TAB_POINTS_CLEFS], coord* depart, coord* arrivee)
/* PS : coordonnées de départ et d'arrivée
    Fournit le meilleur virus a déplacé et les coordonnees de départ et d'arrivée de son meilleurs parcours
*/
{       /*
        DECLARATIONS
    */
    int eval = 0, eval_tmp = 0;
        S_pion* liste_pions = NULL;
        S_pion* pion;
        S_pion* pion_retenu;
        coord destination_tmp;

        //deplacement_pions(coup);

        creer_liste_filtree(&liste_pions, plateau_jeu, tab_listes, VIRUS);

        /*
        INITIALISATION
    */
        depart->x = -1;
        depart->y = -1;
        arrivee->x = -1;
        arrivee->y = -1;


        pion = liste_pions;
        while(pion!=NULL)
        {
                smart_move(pion, plateau_jeu, &eval_tmp, &destination_tmp, tab_points_clefs[CONTROLE]);

                if(eval_tmp > eval){
                        eval = eval_tmp;
                        pion_retenu = pion;
                        *depart = pion_retenu->pos;
                        *arrivee = destination_tmp;
                }
                pion = pion->suiv;
        }

        supprimer_liste(liste_pions);

}

void creer_liste_filtree(S_pion** L_deb_a_jouer, S_case plateau_jeu[TAILLE_PLAT][TAILLE_PLAT], S_pion*
tab_listes[DIM_TAB_LISTES], int joueur)
/* PS : Le premier pointeur si la liste est vide
    Créé une liste filtré permetantde ne traiter que les meilleurs coups
*/
{
        /*
        DECLARATIONS
```

```
    */
        S_masque grand_masque[TAILLE_PLAT][TAILLE_PLAT];
        S_masque petit_masque[MASQUE_MAX][MASQUE_MAX];
        S_pion* act;
        S_pion* next;
        coord org_petit_masque;
        int i, j, is_attack = 0, niveau_max=0;

        int opposant = joueur == GLOBULE ? VIRUS : GLOBULE;
        S_pion* pion = tab_listes[opposant];


        /*****************************CREATION D'UN MASQUE PONDERE GRANDEUR NATURE************************/

        //Initialisation

        for(i=0; i<TAILLE_PLAT; i++)
                for(j=0; j<TAILLE_PLAT; j++)
                {
                        grand_masque[i][j].occupe=0;
                        grand_masque[i][j].poids=-1;
                }

        /*
        REMPLISSAGE
    */
        while(pion != NULL)
        {
                masque_prgm(petit_masque, pion, plateau_jeu);
                org_petit_masque = origine_masque(pion->pos);

                for(i=0; i<MASQUE_MAX; i++)
                        for(j=0; j<MASQUE_MAX; j++)
                        {
                                if(org_petit_masque.x+i>=0 && org_petit_masque.x+i<TAILLE_PLAT && org_petit_masque.y+j >=0 &&
org_petit_masque.y+j< TAILLE_PLAT)
                                {
                                        grand_masque[org_petit_masque.x+i][org_petit_masque.y+j].occupe = petit_masque[i]
[j].occupe;
                                        if(petit_masque[i][j].occupe == OPPOSANT && petit_masque[i][j].poids != -1)
                                                is_attack = 1;

                                        if(grand_masque[org_petit_masque.x+i][org_petit_masque.y+j].poids < petit_masque[i]
[j].poids)
                                                grand_masque[org_petit_masque.x+i][org_petit_masque.y+j].poids = petit_masque[i]
[j].poids;
                                }
                        }

                pion = pion->suiv;
        }


        /*****************************CREATION D'UNE LISTE DES VIRUS FILTREE PAR NIVEAU************************/

        /*
                Ici on filtre la liste :

                Parcourir la liste
            Si un pion n'est pas bougeable
                Supprimer de la liste
                        Si un pion est dans le champ d'un opposant ET que le pion n'est pas dans le champ d'un opposant
                                Supprimer de la liste
                        Sinon
                                MàJ le + haut niveau

                Parcourir la liste
                        Si ce n'est pas un pion parmi les meilleurs
                                Supprimer de la liste
        */
```

```
        dupliquer_liste(L_deb_a_jouer, tab_listes[joueur]);

        act = *L_deb_a_jouer;
        next = *L_deb_a_jouer;

        while(act!=NULL)
        {
        next = act->suiv;

                if(!is_movable(plateau_jeu, act) || (is_attack==1 && grand_masque[act->pos.x][act->pos.y].poids == -1))
                        supprimer_element(L_deb_a_jouer, act);
                else
                        niveau_max = act->niveau > niveau_max ? act->niveau : niveau_max;

                act = next;
        }

        act = *L_deb_a_jouer;
        while(act != NULL)
        {
        next = act->suiv;
                if(act->niveau < niveau_max)
                        supprimer_element(L_deb_a_jouer, act);
                act = next;
        }

}

int is_movable(S_case plat[TAILLE_PLAT][TAILLE_PLAT], S_pion* pion)
/*  PS : 1 si le pion est bougeable , 0 sinon
    Vérifie que le virus n'est pas bloqué
*/
{
    coord h, b, g, d;

    h.x = (pion->pos).x;
        h = pion->pos;  h.y--;
        b = pion->pos;  b.y++;
        g = pion->pos;  g.x--;
        d = pion->pos;  d.x++;

    if(case_est_libre(plat, h) || case_est_libre(plat, b) || case_est_libre(plat, g) || case_est_libre(plat, d))
        return 1;
    else
        return 0;
}

void smart_move(S_pion* pion, S_case plateau_jeu[TAILLE_PLAT][TAILLE_PLAT], int* note_max, coord* coup_tmp, coord targ)
{
        S_masque masque_pion[MASQUE_MAX][MASQUE_MAX];
        int i, j;
        float dist=-1, dist_tmp;
        coord act;

        *note_max = 1;

        masque_prgm(masque_pion, pion, plateau_jeu);

        for(j=0; j<MASQUE_MAX; j++)
                for(i=0; i<MASQUE_MAX; i++)
                {
                        if(masque_pion[i][j].poids != -1 && masque_pion[i][j].occupe > *note_max)
                        {
                                coup_tmp->x = i; coup_tmp->y = j;
                                *coup_tmp = masque2tab(*coup_tmp, pion->pos);
                                *note_max = masque_pion[i][j].occupe;
                        }
                        else if(masque_pion[i][j].poids != -1 && *note_max ==1)
                        {
```

```
                                act.x = i; act.y = j;
                                act = masque2tab(act, pion->pos);
                                dist_tmp = sqrt(pow(act.x-targ.x, 2) + pow(act.y-targ.y, 2));
                                if(dist < 0 || dist_tmp < dist)
                                {
                                        *coup_tmp=act;
                                        dist = dist_tmp;
                                }
                        }
                }
}
```

```c
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : editeur.h
    Dernière Modification : 16/05/2010

    Description :
        Header
*/

#ifndef EDITEUR_H_INCLUDED
#define EDITEUR_H_INCLUDED

#include "structures.h"
#include "affichage_sdl.h"
#include "outils_sdl.h"

void editeur_prg();
void initialiser_plateau(S_case plateau[TAILLE_PLAT][TAILLE_PLAT]);
void new_coord(S_chemin** L_deb, int x, int y);
void suppr_elt(S_chemin* element);
void fprintCoord(FILE* fichier, S_chemin* L_coord);
void sauvegarde(S_case plateau[TAILLE_PLAT][TAILLE_PLAT]);

#endif // EDITEUR_H_INCLUDED
```

```
/*
    Projet second semestre EPF 1A, Promo 2014
    Nom du projet : IMMUNO WARS
    Auteurs : Simon CAMPANO & Nicolas MANIE

    Nom du fichier : editeur.cpp
    Dernière Modification : 16/05/2010

    Description :
        Contient toutes les fonctions et procédures nécessaire à l'editeur et l'éditeur lui même
*/

#include <stdlib.h>
#include <stdio.h>
#include <SDL/SDL.h>

#include"editeur.h"

void editeur_prg()
/* PS :/
    Editeur
*/
{   /* DECLARATIONS */
    S_case tableau[TAILLE_PLAT][TAILLE_PLAT];

    SDL_Surface* tableau_surface[NB_SURFACE], *temp;
        SDL_Rect tableau_position[NB_POS], coordonnees;
        coord a; // Doit être passé en argument pour certaines fonctions

        int nb_entree_gr=0, nb_entree_virus=0, nb_pt_controle=0;
    int continuer=1, continuer1=1, choix, quit=0;
    SDL_Event event;


        /* INITIALISATIONS */
        initialiser_plateau(tableau);
        initialiser_SDL(tableau_surface, tableau_position);


    /* AFFICHAGE */
    afficher_joueur(BUILDER, tableau_surface, tableau_position);
    afficher_plateau(tableau, tableau_surface, tableau_position, a, 0);

    SDL_Flip(tableau_surface[ECRAN]);




    while(!quit)
    {
        SDL_ShowCursor(SDL_ENABLE); // Affiche le curseur

        afficher_joueur(BUILDER, tableau_surface, tableau_position);
        afficher_plateau(tableau, tableau_surface, tableau_position, a, 0);
        temp=tableau_surface[PLATEAU]; // On creer une copie du plateau pour ne pas avoir a reparcourir le tableau pour
l'afficher

        /* REINITIALISATION DES VARIABLES */
        choix=0;
        continuer=1;
        continuer1=1;

        while (continuer)
        {
            SDL_WaitEvent(&event);
            switch(event.type)
            {
                case SDL_QUIT: // Si le joueur ferme, la map n'est pas sauvegardée
                    quit=1;
                    continuer=0;
```

```
                    break;
                case SDL_KEYDOWN: // S'il appui sur une touche
                    switch(event.key.keysym.sym)
                    {
                        /*
                            Selon la touche sur laquelle il appui, on selectione tel ou tel pion
                        */
                        case SDLK_1:
                            choix=GB1_3; break;
                        case SDLK_2:
                            choix=VIR1; break;
                        case SDLK_3:
                            choix=BLOQ; break;
                        case SDLK_4:
                            if(!nb_entree_gr)
                                choix=E_GR;
                            nb_entree_gr++;
                            break;
                        case SDLK_5:
                            if(!nb_entree_virus)
                                choix=E_VIRUS;
                            nb_entree_virus++;
                            break;
                        case SDLK_6:
                            if(!nb_pt_controle)
                                choix=CTRL;
                            nb_pt_controle++;
                            break;
                        case SDLK_ESCAPE:
                            continuer=0;
                            quit=1; break;
                        default : break;
                    }
                    if(choix!=0)
                        continuer=0; // Pour sortir de la boucle
                    break;
            }
        }

        SDL_ShowCursor(SDL_DISABLE); // On enlève le curseur

        if(!quit)
        {

            while(continuer1)
            {
                /*
                    On affiche le pion séléctioné qui suit la souris, puis se fixe sur l'écran
                */

                SDL_WaitEvent(&event);
                switch(event.type)
                {
                    case SDL_MOUSEMOTION:
                        if(event.motion.y>=POS_PLATEAU_Y && event.motion.y<=POS_PLATEAU_Y+tableau_surface[PLATEAU]->h &&
event.motion.x>=POS_PLATEAU_X && event.motion.x<=POS_PLATEAU_X+tableau_surface[PLATEAU]->w)
                        {
                         coordonnees.x = event.motion.x - tableau_surface[choix]->w/2;
                         coordonnees.y = event.motion.y - tableau_surface[choix]->h/2;

                         SDL_BlitSurface(temp, NULL, tableau_surface[ECRAN], &tableau_position[PLATEAU]);
                         SDL_BlitSurface(tableau_surface[choix], NULL, tableau_surface[ECRAN], &coordonnees);
                         SDL_Flip(tableau_surface[ECRAN]);
                        }
                    break;

                    case SDL_MOUSEBUTTONUP :
                        if(event.button.y>=tableau_surface[EN_TETE]->h && event.button.y<=POS_PLATEAU_Y
+tableau_surface[PLATEAU]->h && event.button.x>=POS_PLATEAU_X && event.button.x<=POS_PLATEAU_X+tableau_surface[EN_TETE]->w)
                        {
```

```
                        continuer1=0;
                        coordonnees.x = event.button.x - tableau_surface[choix]->w/2;
                        coordonnees.y = event.button.y - tableau_surface[choix]->h/2;

                        SDL_BlitSurface(temp, NULL, tableau_surface[ECRAN], &tableau_position[PLATEAU]);
                        SDL_BlitSurface(tableau_surface[choix], NULL, tableau_surface[ECRAN], &coordonnees);
                        SDL_Flip(tableau_surface[ECRAN]);
                        }
                        break;
                }
            }
                // Conversion des coordonnees pour les rentrer dans le tableau
                coordonnees.x=sdl2cons(coordonnees.x + tableau_surface[choix]->w/2, 'h');
                coordonnees.y=sdl2cons(coordonnees.y + tableau_surface[choix]->h/2, 'v');

                switch(choix)
                {
                    case GB1_3:
                        /*
                            On doit redéfinir le niveau et la vitesse pour le globule blanc et le virus pour que la fonction
d'affichage fonctionne
                        */
                        tableau[coordonnees.x][coordonnees.y].type=NORMALE;
                        tableau[coordonnees.x][coordonnees.y].pion = (S_pion*) malloc(sizeof (S_pion));
                        tableau[coordonnees.x][coordonnees.y].pion->type_pion=GLOBULE;
                        tableau[coordonnees.x][coordonnees.y].pion->vitesse=VITESSE_INI_GB;
                        tableau[coordonnees.x][coordonnees.y].pion->niveau=1;
                        tableau[coordonnees.x][coordonnees.y].pion->pos.x=coordonnees.x;
                        tableau[coordonnees.x][coordonnees.y].pion->pos.y=coordonnees.y;
                        break;
                    case VIR1:
                        tableau[coordonnees.x][coordonnees.y].type=NORMALE;
                        tableau[coordonnees.x][coordonnees.y].pion = (S_pion*) malloc(sizeof (S_pion));
                        tableau[coordonnees.x][coordonnees.y].pion->type_pion=VIRUS;
                        tableau[coordonnees.x][coordonnees.y].pion->vitesse=VITESSE_INI_VIRUS;
                        tableau[coordonnees.x][coordonnees.y].pion->niveau=1;
                        break;
                    case BLOQ:
                        tableau[coordonnees.x][coordonnees.y].type=BLOQUEE;
                        break;
                    case E_GR:
                        tableau[coordonnees.x][coordonnees.y].type=ENTREE_GR;
                        break;
                    case E_VIRUS:
                        tableau[coordonnees.x][coordonnees.y].type=ENTREE_VIRUS;
                        break;
                    case CTRL:
                        tableau[coordonnees.x][coordonnees.y].type=CONTROLE;
                        break;
                }
        }
    }

        sauvegarde(tableau); // Quand le joueur a décidé de quitter , on sauvegarde le plateau



}

void initialiser_plateau(S_case plateau[TAILLE_PLAT][TAILLE_PLAT])
/* PS : /
    Initialise le plateau de jeu
*/
{
    int i, j;

        for(i=0; i<TAILLE_PLAT; i++)
        {
            for(j=0; j<TAILLE_PLAT; j++)
            {
```

```
                plateau[i][j].pion = NULL;
                plateau[i][j].type = NORMALE;
            }
        }
}

void sauvegarde(S_case plateau[TAILLE_PLAT][TAILLE_PLAT])
/* PS :/
    Sauvegarde la plateau de jeu de l'éditeur
*/
{
    /*
        DECLARATIONS
    */
        int i, j;
        S_pion* pion;
    S_chemin *L_gb = NULL, *L_vir = NULL, *L_blok = NULL, *L_evir = NULL, *L_egr = NULL, *L_ctrl = NULL;


        /*
        Pour chaque case, on distingue chaque pion
    */
        for(j=0; j<TAILLE_PLAT; j++)
                for(i=0; i<TAILLE_PLAT; i++)
                {
                        switch(plateau[i][j].type)
                        {
                                case BLOQUEE : new_coord(&L_blok, i, j); break;
                                case CONTROLE : new_coord(&L_ctrl, i, j); break;
                                case ENTREE_VIRUS : new_coord(&L_evir, i, j); break;
                                case ENTREE_GR : new_coord(&L_egr, i, j); break;
                                default :
                    pion = plateau[i][j].pion;
                                        if(pion != NULL)
                                        {
                                                switch(pion->type_pion)
                                                {
                                                        case GLOBULE : new_coord(&L_gb, i, j); break;
                                                        case VIRUS : new_coord(&L_vir, i, j); break;
                                                }
                                        }
                                        break;
                        }
                }

        /*
        Ouverture du fichier texte
    */
        FILE* fichier = NULL;
    fichier = fopen("map/map.txt", "w+");

        /*
        Impression des caractères repères et des listes de coordonnées
    */
        if(fichier!=NULL)
        {
                fprintf(fichier, "FICHIER DE SAUVEGARDE\n#");
                fprintf(fichier, "Globules Blancs : ");
                fprintCoord(fichier, L_gb);
                fprintf(fichier, "Virus : ");
                fprintCoord(fichier, L_vir);
                fprintf(fichier, "Cases Bloquees : ");
                fprintCoord(fichier, L_blok);
                fprintf(fichier, "Points de controle : ");
                fprintCoord(fichier, L_ctrl);
                fprintf(fichier, "Entree Virus : ");
                fprintCoord(fichier, L_evir);
                fprintf(fichier, "Entree GR: ");
                fprintCoord(fichier, L_egr);
```

```
        }

        fclose(fichier);

    /*
        LIBERATION MEMOIRE
    */
        suppr_elt(L_gb);
        suppr_elt(L_vir);
        suppr_elt(L_blok);
        suppr_elt(L_evir);
        suppr_elt(L_egr);
        suppr_elt(L_ctrl);
}

void fprintCoord(FILE* fichier, S_chemin* L_coord)
/* PS : /
    Imprime la liste des coord d'une liste fixé dans le .txt
*/
{
        while(L_coord != NULL)
        {
                fprintf(fichier, "(%d;%d) ", L_coord->etape.x + 1, L_coord->etape.y + 1);
                L_coord = L_coord->suiv;
        }
        fprintf(fichier, "\n#\n");
}

void suppr_elt(S_chemin* element)
/* PS : /
    Supprime les elements
*/
{
        if(element != NULL)
        {
                suppr_elt(element->suiv);
                free(element);
        }
}

void new_coord(S_chemin** L_deb, int x, int y)
/* PS : L_deb
    Créé une nouvelle VD et la chaine avec les précèdentes
*/
{
        S_chemin* actuel;
        int i=0;

        S_chemin* element = (S_chemin*) malloc(sizeof(S_chemin));
                element->etape.x = x;
                element->etape.y = y;
                element->suiv = NULL;

        if(*L_deb == NULL)
                *L_deb = element;
        else
        {
                actuel = *L_deb;
                while(actuel->suiv != NULL)
                {
                    fprintf(stderr, "#%d\t%d\t%d", i, actuel, actuel->suiv);
                        actuel = actuel->suiv;
                }
                actuel->suiv = element;
        }

}
```