

`bash_commando`

0.2

Generated by Doxygen 1.9.8



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

fichier . . . . .	??
ligne . . . . .	??
ligne_commande	
Une representation d'une ligne de commande une ligne de commande doit contenir evidemment la commande et les parametres ; options qui peut être facultatif ou pas . . . . .	??
Mission . . . . .	??



# Chapter 2

## File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

include/affichage.h . . . . .	??
include/arborescence.h . . . . .	??
include/gestion_fichier.h	
Regroupe tous les fonctions qui manipule les fichiers . . . . .	??
include/menu.h . . . . .	??
include/mission.h . . . . .	??
include/prompt.h	
Regroupement des fonctions dans prompt.c . . . . .	??
include/sécuré.h . . . . .	??
src/affichage.c . . . . .	??
src/arborescence.c	
Fonction pour manipuler les arborescences . . . . .	??
src/gestion_fichier.c	
Regroupe tous les fonctions qui manipule les fichiers . . . . .	??
src/main.c	
: Fonction principale du jeu Bash Commando . . . . .	??
src/menu.c . . . . .	??
src/mission.c	
: Fonction pour gérer les missions du jeu . . . . .	??
src/prompt.c	
Ceci fait reference à la simulation de prompt . . . . .	??
src/sécuré.c	
Contenant 04 fonction pour qui lit en toute sécurité les reponses insereés sur "stdin" par l'utilisateur . . . . .	??



# Chapter 3

## Data Structure Documentation

### 3.1 fichier Struct Reference

```
#include <arborescence.h>
```

Collaboration diagram for fichier:



#### Data Fields

- char `nom` [20]
- int `estDossier`
- int `estProtege`
- int `nombre_ouverture`
- char `perm` [10]
- char \*\* `contenu`
- struct `fichier` \* `parent`
- struct `fichier` \* `premierfils`
- struct `fichier` \* `frereSuivant`

#### 3.1.1 Detailed Description

Definition at line 3 of file arborescence.h.

### 3.1.2 Field Documentation

#### 3.1.2.1 contenu

```
char** contenu
```

Definition at line 10 of file [arborescence.h](#).

#### 3.1.2.2 estDossier

```
int estDossier
```

Definition at line 6 of file [arborescence.h](#).

#### 3.1.2.3 estProtege

```
int estProtege
```

Definition at line 7 of file [arborescence.h](#).

#### 3.1.2.4 frereSuivant

```
struct fichier* frereSuivant
```

Definition at line 13 of file [arborescence.h](#).

#### 3.1.2.5 nom

```
char nom[20]
```

Definition at line 5 of file [arborescence.h](#).

#### 3.1.2.6 nombre\_ouverture

```
int nombre_ouverture
```

Definition at line 8 of file [arborescence.h](#).

#### 3.1.2.7 parent

```
struct fichier* parent
```

Definition at line 11 of file [arborescence.h](#).

### 3.1.2.8 perm

```
char perm[10]
```

Definition at line [9](#) of file [arborescence.h](#).

### 3.1.2.9 premierfils

```
struct fichier* premierfils
```

Definition at line [12](#) of file [arborescence.h](#).

The documentation for this struct was generated from the following file:

- [include/arborescence.h](#)

## 3.2 ligne Struct Reference

```
#include <prompt.h>
```

### Data Fields

- `char * commande`
- `char ** option`
- `char ** argument`

### 3.2.1 Detailed Description

Definition at line [40](#) of file [prompt.h](#).

### 3.2.2 Field Documentation

#### 3.2.2.1 argument

```
char** argument
```

la tableau des arguments dont le dernier doit être null

Definition at line [44](#) of file [prompt.h](#).

#### 3.2.2.2 commande

```
char* commande
```

la commande principale , c'est à dire le premier mot de la ligne de commande

Definition at line [42](#) of file [prompt.h](#).

### 3.2.2.3 option

```
char** option
```

la tableau des options dont le dernier doit être null

Definition at line 43 of file [prompt.h](#).

The documentation for this struct was generated from the following file:

- [include/prompt.h](#)

## 3.3 ligne\_commande Struct Reference

une representation d'une ligne de commande une ligne de commande doit contenir evidemment la commande et les parametres ; options qui peut être facultatif ou pas

```
#include <prompt.h>
```

### 3.3.1 Detailed Description

une representation d'une ligne de commande une ligne de commande doit contenir evidemment la commande et les parametres ; options qui peut être facultatif ou pas

The documentation for this struct was generated from the following file:

- [include/prompt.h](#)

## 3.4 Mission Struct Reference

```
#include <mission.h>
```

### Data Fields

- int [debloqué](#)
- int [terminé](#)
- char [titre](#) [50]
- int [point](#)
- char [indice](#) [100]

### 3.4.1 Detailed Description

Definition at line 10 of file [mission.h](#).

## 3.4.2 Field Documentation

### 3.4.2.1 debloqué

```
int debloqué
```

Definition at line 12 of file [mission.h](#).

### 3.4.2.2 indice

```
char indice[100]
```

Definition at line 16 of file [mission.h](#).

### 3.4.2.3 point

```
int point
```

Definition at line 15 of file [mission.h](#).

### 3.4.2.4 terminé

```
int terminé
```

Definition at line 13 of file [mission.h](#).

### 3.4.2.5 titre

```
char titre[50]
```

Definition at line 14 of file [mission.h](#).

The documentation for this struct was generated from the following file:

- [include/mission.h](#)

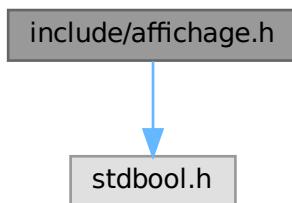


# Chapter 4

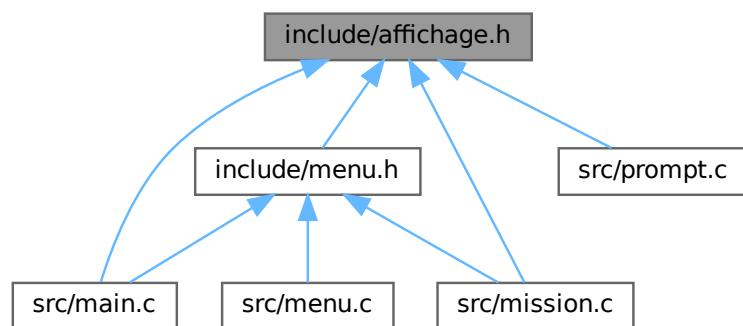
## File Documentation

### 4.1 include/affichage.h File Reference

```
#include <stdbool.h>
Include dependency graph for affichage.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- void `screen_clear ()`  
*Fonction pour nettoyer l'écran.*
- void `afficher_tableau_avec_titre_position_exacte (const char *titre, const char *nom_fichier)`
- void `afficher_tableau_fichier (const char *titre, const char *nom_fichier)`
- void `affiche_fichier_tableau curse_position_exacte (const char *titre, const char *nom_fichier)`
- bool `felicitation ()`  
*Affiche un message de félicitation.*
- bool `echec ()`  
*Affiche un message d'échec.*
- void `affiche_fichier (char *nom_fichier)`

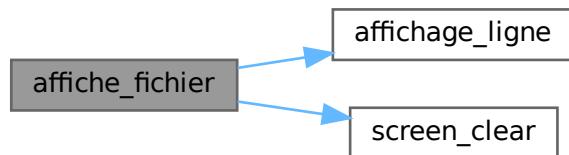
### 4.1.1 Function Documentation

#### 4.1.1.1 `affiche_fichier()`

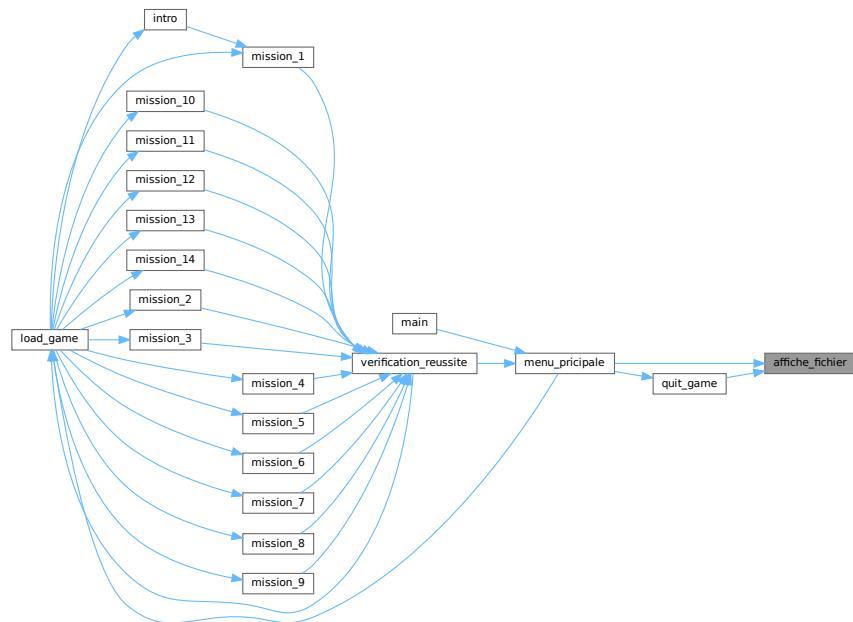
```
void affiche_fichier (
    char * nom_fichier )
```

Definition at line 45 of file `affichage.c`.

Here is the call graph for this function:



Here is the caller graph for this function:

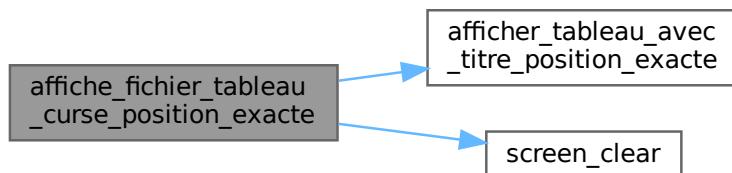


#### 4.1.1.2 `affiche_fichier_tableau curse_position_exacte()`

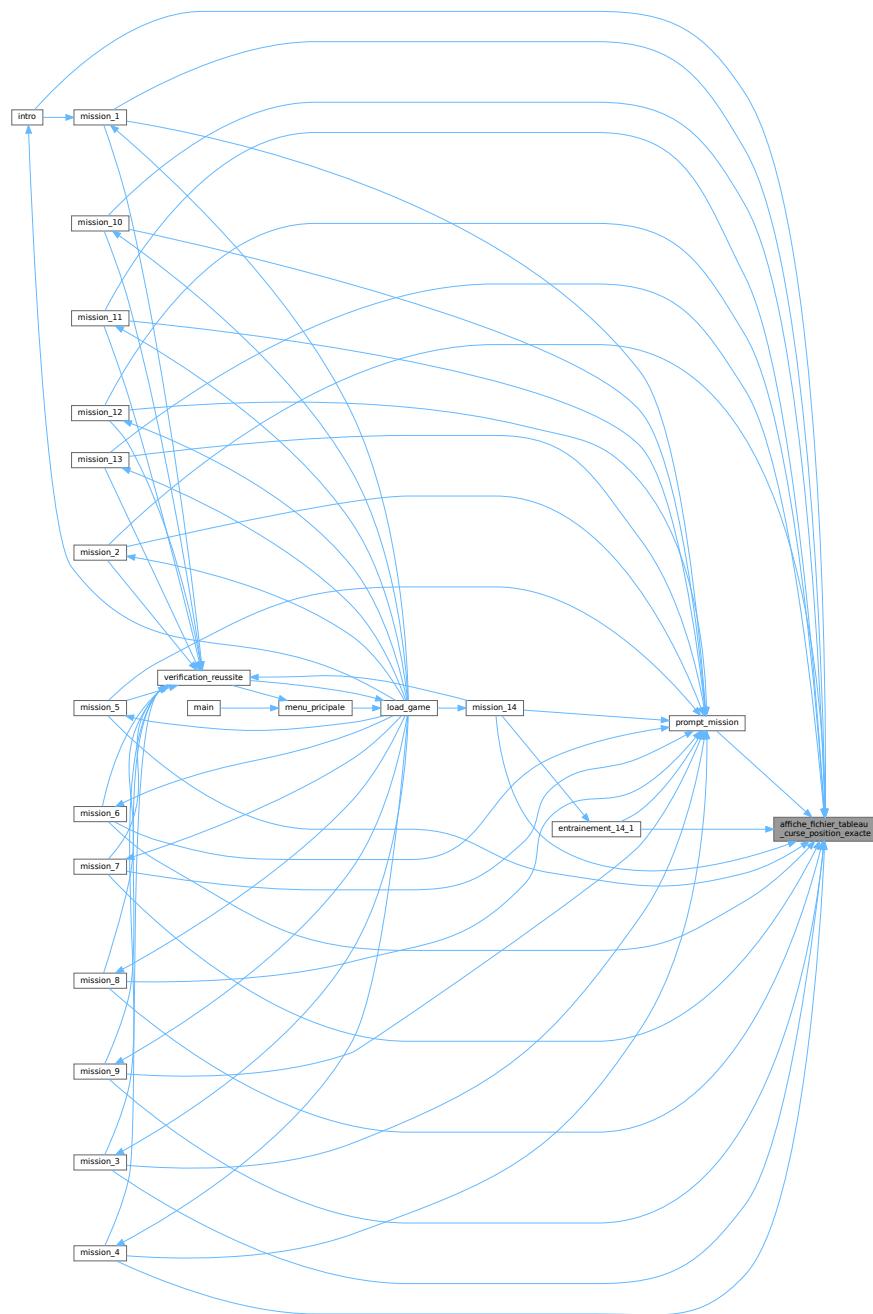
```
void affiche_fichier_tableau curse_position_exacte (
    const char * titre,
    const char * nom_fichier )
```

Definition at line 292 of file [affichage.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

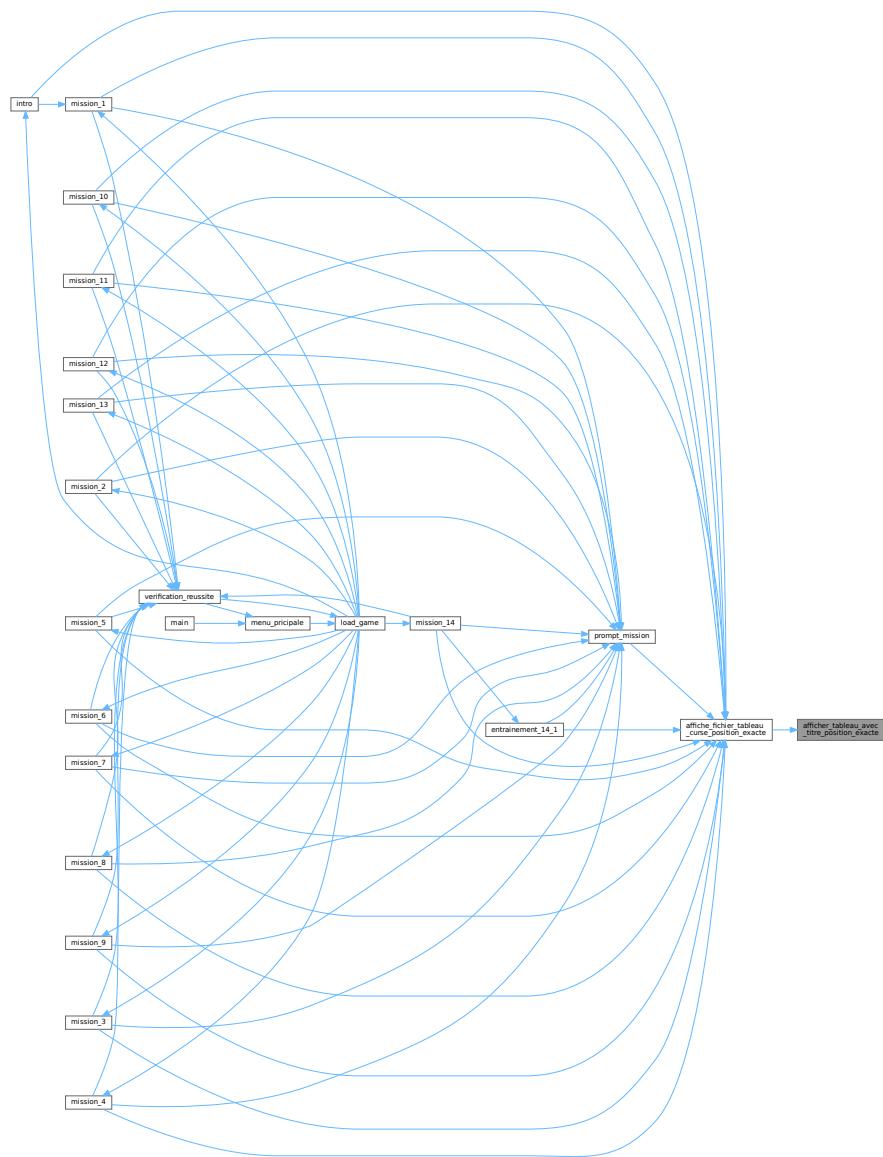


#### 4.1.1.3 `afficher_tableau_avec_titre_position_exacte()`

```
void afficher_tableau_avec_titre_position_exacte (
    const char * titre,
    const char * nom_fichier )
```

Definition at line 190 of file [affichage.c](#).

Here is the caller graph for this function:



#### 4.1.1.4 afficher\_tableau\_fichier()

```
void afficher_tableau_fichier (
    const char * titre,
    const char * nom_fichier )
```

#### 4.1.1.5 echec()

```
echec ( )
```

Affiche un message d'échec.

Affiche une message d'échec et récupère l'avis du joueur s'il souhaite recommencé ou pas.

Affiche une message félicitation et récupère l'avis du joueur s'il souhaite passé au mission suivant ou pas.

**Returns**

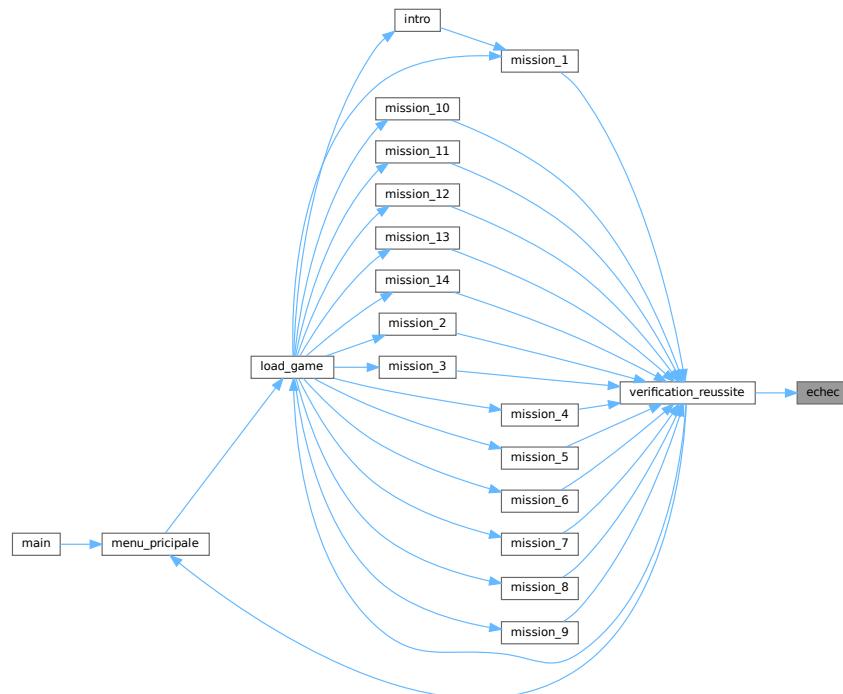
Retourne un bool

Definition at line 125 of file [affichage.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.1.6 felicitation()

```
felicitation( )
```

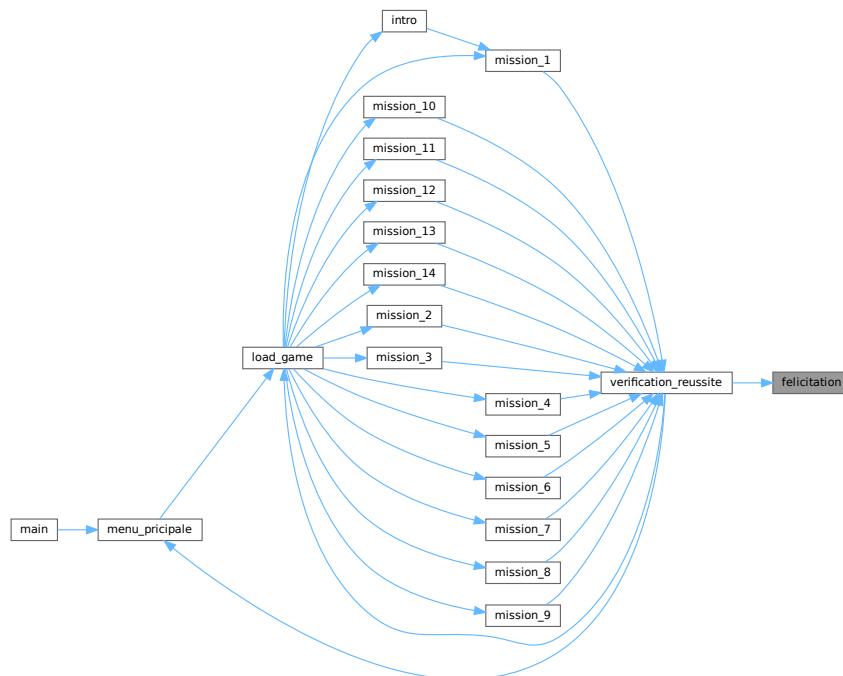
Affiche un message de félicitation.

Definition at line 57 of file [affichage.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



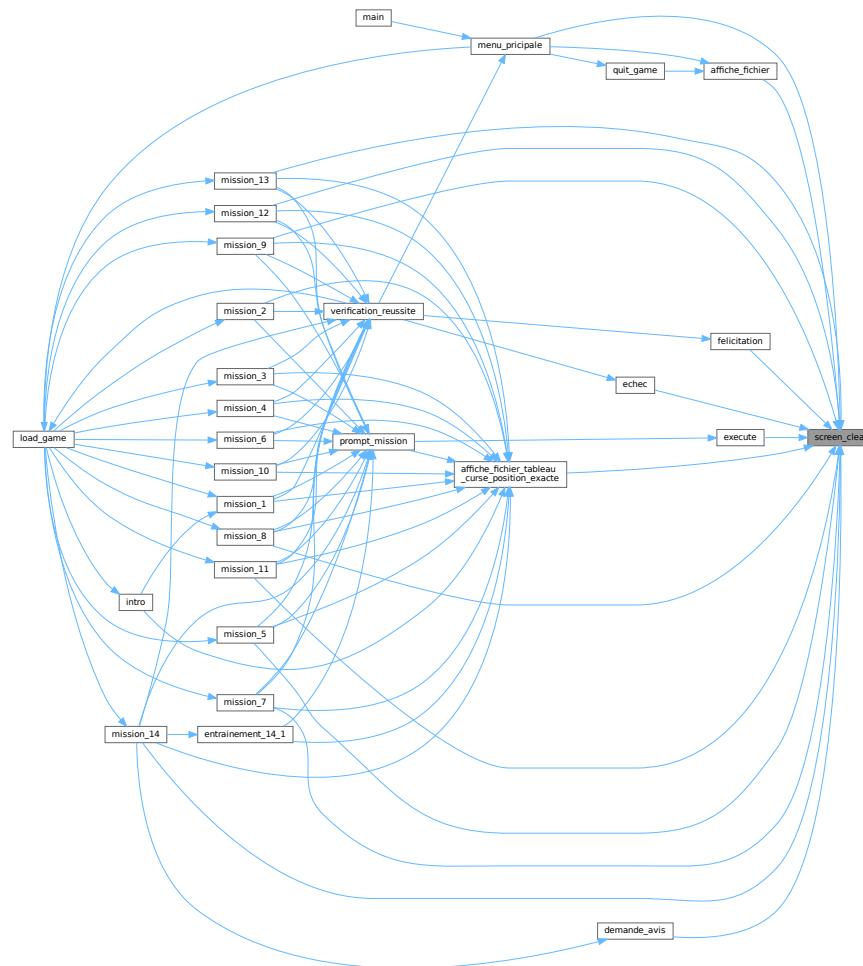
#### 4.1.1.7 screen\_clear()

```
screen_clear( )
```

Fonction pour nettoyer l'écran.

Definition at line 14 of file [affichage.c](#).

Here is the caller graph for this function:



## 4.2 affichage.h

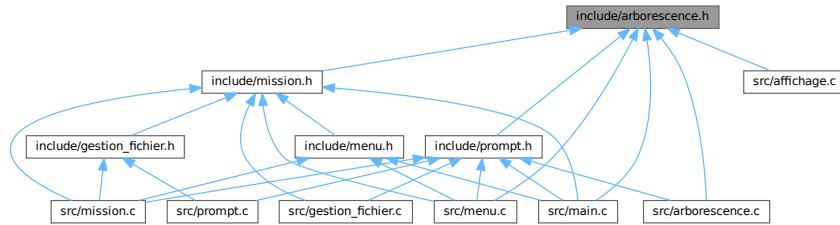
[Go to the documentation of this file.](#)

```

00001 #ifndef __AFFICHE__
00002 #define __AFFICHE__
00003
00004     #include <stdbool.h> // Pour le type booléen
00005
00010 void screen_clear();
00017 void afficher_tableau_avec_titre_position_exacte(const char *titre, const char *nom_fichier);
00024 void afficher_tableau_fichier(const char *titre, const char *nom_fichier);
00031 void affiche_fichier_tableau curse_position_exacte(const char *titre , const char *nom_fichier);
00032
00037 bool felicitation();
00042 bool echec();
00043 void affiche_fichier(char *nom_fichier) ;
00044
00045 #endif
  
```

## 4.3 include/arborescence.h File Reference

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct `fichier`

### Typedefs

- typedef struct `fichier` `fichier`

### Functions

- `fichier * creer (fichier *parent, char *nom, int estDossier, int statut)`  
*Crée un nouveau fichier ou dossier.*
- `void ajouter_enfant (fichier *parent, char *nom, int nombre, int statut)`  
*Ajoute un enfant (fichier ou dossier) à un dossier parent.*
- `void my_mkdir (fichier *racine, fichier *courant, char *nom, int statut)`  
*Crée un nouveau dossier dans le répertoire courant.*
- `void afficher (fichier *home, int prfnd)`  
*Affiche la structure de l'arborescence des fichiers et dossiers.*
- `void my_ls (fichier *racine, fichier *courant, char *nom)`  
*Liste les fichiers et dossiers dans le répertoire courant.*
- `void my_touch (fichier *racine, fichier *courant, char *nom, int statut)`  
*Crée un nouveau fichier dans le répertoire courant.*
- `void my_rmdir (fichier *racine, fichier *courant, char *nom)`  
*Supprime un dossier dans le répertoire courant.*
- `fichier * my_cd (fichier *racine, fichier *courant, char *nom)`  
*Change le répertoire courant.*
- `void initialiser (fichier *racine)`  
*Initialise l'arborescence des fichiers et dossiers avec une structure de base.*
- `void my_cp (fichier *racine, fichier *parent, char *nom, char *destination)`  
*Copie un fichier ou dossier vers une destination.*
- `fichier * chercher (fichier *courant, char *nom)`  
*Cherche un fichier ou dossier par son nom dans le répertoire courant.*
- `void my_mv (fichier *racine, fichier *parent, char *nom, char *destination)`  
*Déplace un fichier ou dossier vers une destination.*

- void `my_rm (fichier *racine, fichier *courant, char *nom)`  
*Supprime un fichier dans le répertoire courant.*
- void `my_mkdir_p (fichier *racine, fichier *parent, char *nom)`  
*Crée un chemin de dossiers, créant les dossiers intermédiaires si nécessaire.*
- void `my_pwd (fichier *rep_actuel)`  
*Affiche le chemin complet du répertoire courant.*
- void `my_echo (char **ligne_c)`  
*Affiche une ligne de texte.*
- void `my_ls_a (fichier *racine, fichier *courant, char *nom)`  
*Liste tous les fichiers et dossiers, y compris les fichiers cachés, dans le répertoire courant.*
- void `deplacer (fichier *dest, fichier *source)`  
*Déplace un fichier ou dossier d'une source vers une destination.*
- void `my_chmod (fichier *racine, fichier *courant, char *option, char *nom)`  
*Modifie les permissions d'un fichier ou dossier.*
- void `my_ls_l (fichier *racine, fichier *courant, char *nom)`  
*Liste les fichiers et dossiers avec des détails dans le répertoire courant.*
- void `afficher_nbr_ouverture (fichier *home, int prfnd)`  
*Affiche le nombre d'ouvertures pour chaque fichier dans l'arborescence.*
- int `recherche_max (fichier *home, int max)`  
*Recherche le nombre maximum d'ouvertures parmi les fichiers dans l'arborescence.*
- void `favori (fichier *home, int max)`  
*Affiche les fichiers ayant le nombre maximum d'ouvertures.*
- void `affichage (fichier *home)`  
*Affiche les fichiers les plus ouverts dans l'arborescence.*
- void `my_ls_la (fichier *racine, fichier *courant, char *nom)`  
*Liste tous les fichiers et dossiers avec des détails, y compris les fichiers cachés, dans le répertoire courant.*

### 4.3.1 Typedef Documentation

#### 4.3.1.1 fichier

```
typedef struct fichier fichier
```

Definition at line 15 of file [arborescence.h](#).

### 4.3.2 Function Documentation

#### 4.3.2.1 affichage()

```
void affichage (
    fichier * home )
```

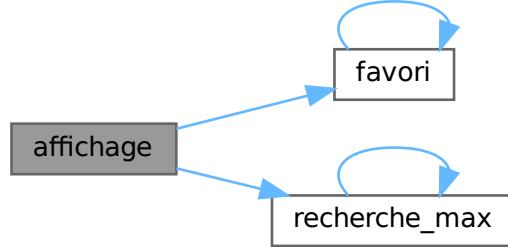
Affiche les fichiers les plus ouverts dans l'arborescence.

#### Parameters

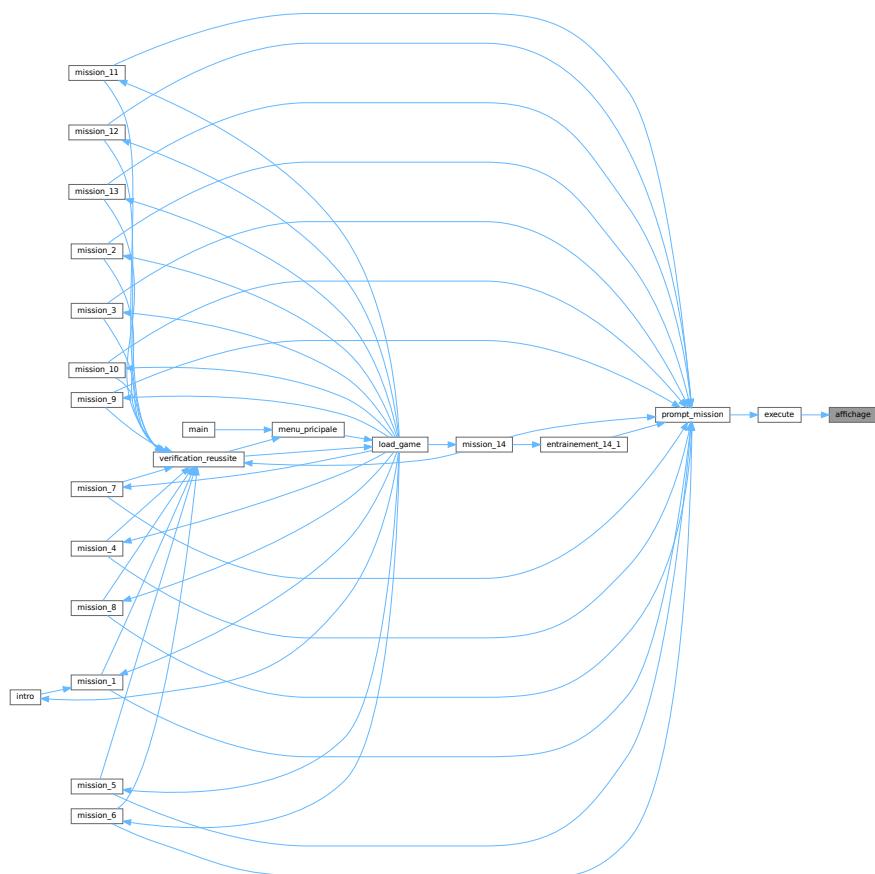
<code>home</code>	Pointeur vers le répertoire racine.
-------------------	-------------------------------------

Definition at line 1530 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.2 afficher()

```
void afficher (
```

```
fichier * home,
int prfnd )
```

Affiche la structure de l'arborescence des fichiers et dossiers.

#### Parameters

<i>home</i>	Pointeur vers le répertoire racine.
<i>prfnd</i>	Profondeur actuelle dans l'arborescence.

Definition at line 170 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.3 afficher\_nbr\_ouverture()

```
void afficher_nbr_ouverture (
    fichier * home,
    int prfnd )
```

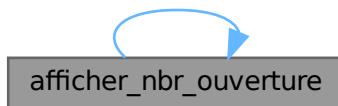
Affiche le nombre d'ouvertures pour chaque fichier dans l'arborescence.

#### Parameters

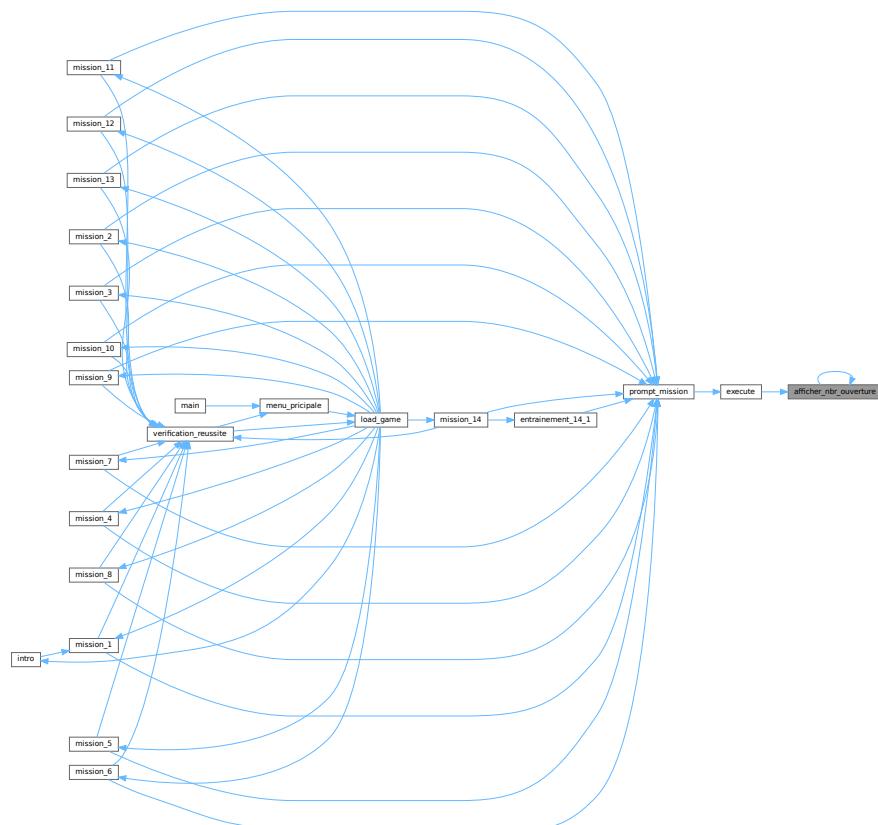
<i>home</i>	Pointeur vers le répertoire racine.
<i>prfnd</i>	Profondeur actuelle dans l'arborescence.

Definition at line 1486 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.4 ajouter\_enfant()

```
void ajouter_enfant (
    fichier * parent,
    char * nom,
    int nombre,
    int statut )
```

Ajoute un enfant (fichier ou dossier) à un dossier parent.

### Parameters

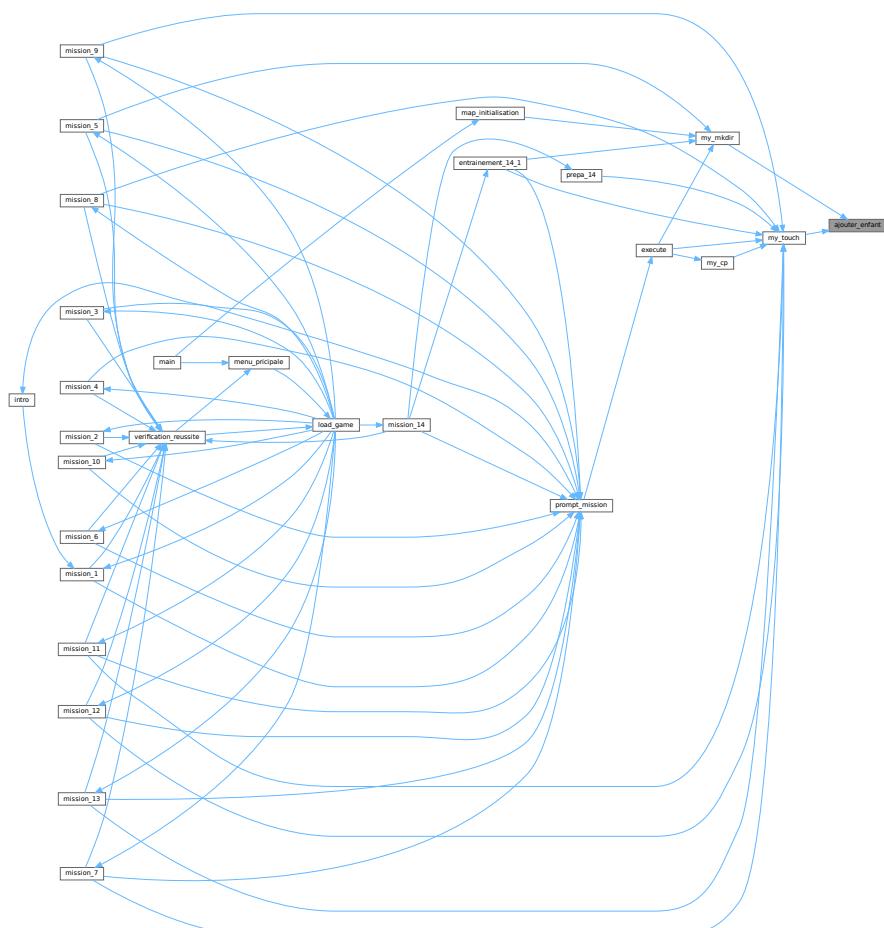
<i>parent</i>	Pointeur vers le dossier parent.
<i>nom</i>	Nom de l'enfant à ajouter.
<i>nombre</i>	Indicateur si c'est un dossier (1) ou un fichier (0).
<i>statut</i>	Statut de protection de l'enfant.

Definition at line 40 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.3.2.5 chercher()

```
fichier * chercher (
```

Cherche un fichier ou dossier par son nom dans le répertoire courant.

## Parameters

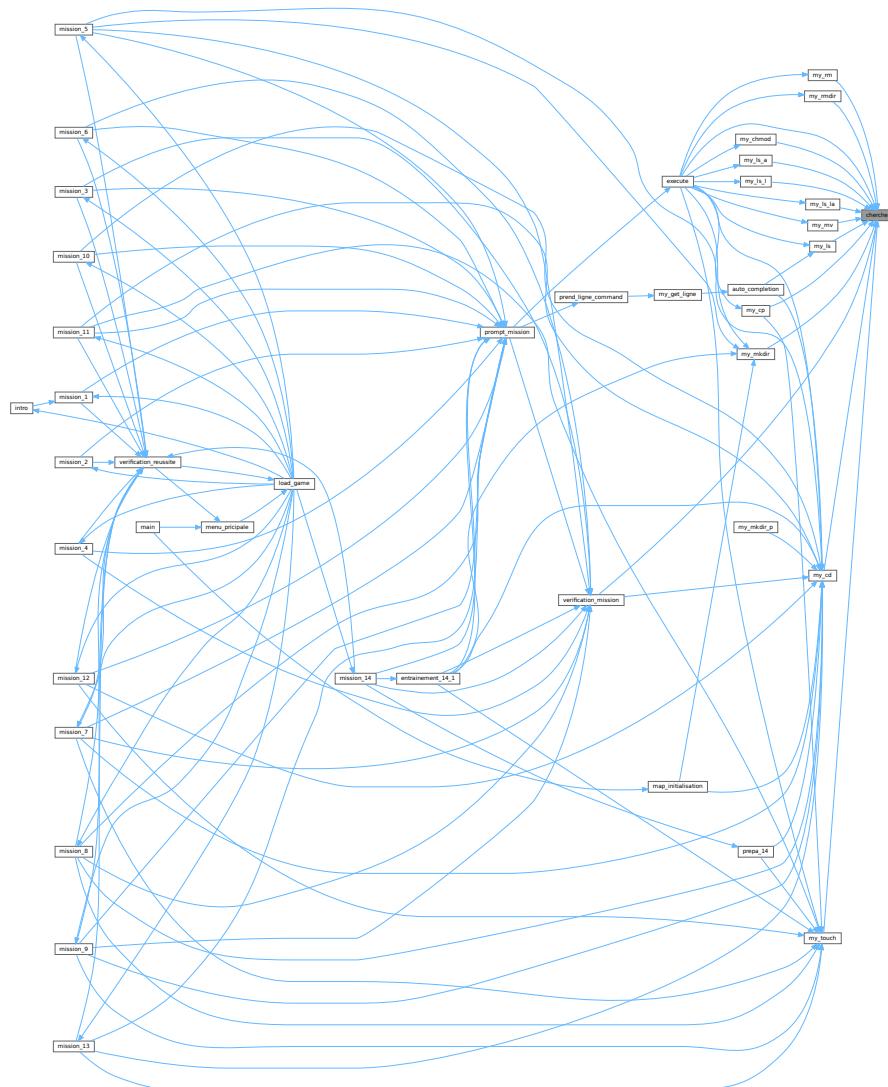
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du fichier ou dossier à chercher.

## Returns

Pointeur vers le fichier ou dossier trouvé, ou NULL s'il n'existe pas.

Definition at line 984 of file [arborescence.c](#).

Here is the caller graph for this function:



#### 4.3.2.6 creer()

```
fichier * creer (
    fichier * parent,
    char * nom,
    int estDossier,
    int statut )
```

Crée un nouveau fichier ou dossier.

##### Parameters

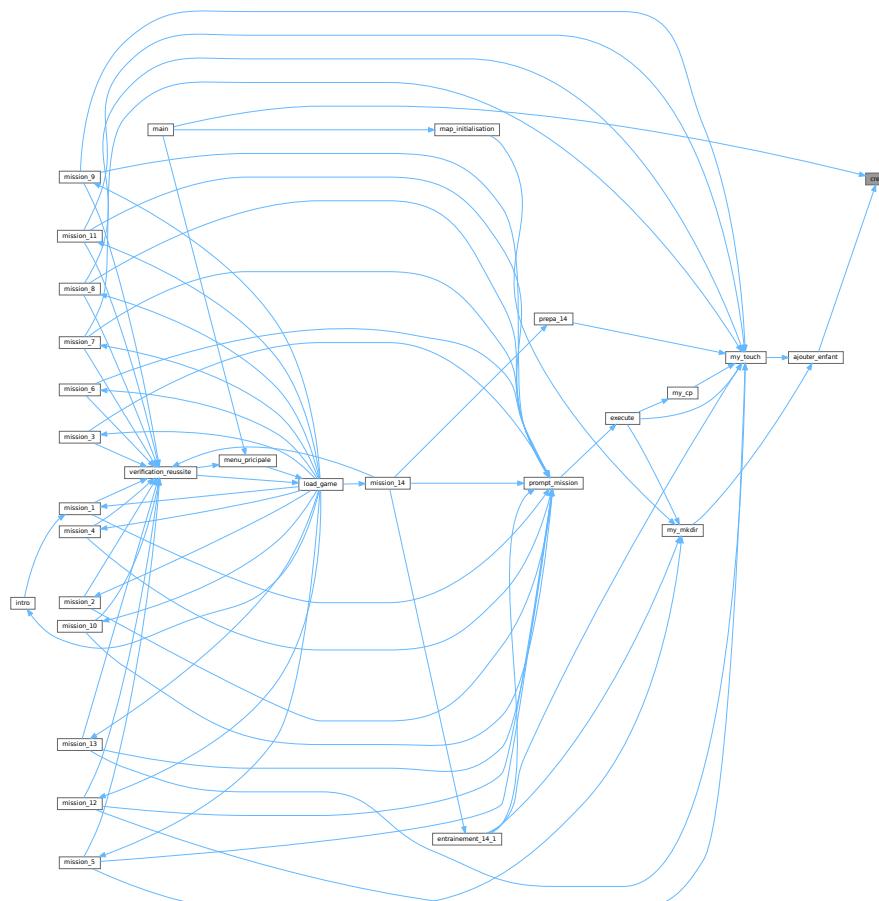
<i>parent</i>	Pointeur vers le dossier parent.
<i>nom</i>	Nom du fichier ou dossier à créer.
<i>estDossier</i>	Indicateur si c'est un dossier (1) ou un fichier (0).
<i>statut</i>	Statut de protection du fichier ou dossier.

##### Returns

Pointeur vers le nouveau fichier ou dossier créé.

Definition at line 19 of file [arborescence.c](#).

Here is the caller graph for this function:



### 4.3.2.7 `deplacer()`

```
void deplacer (
    fichier * dest,
    fichier * source )
```

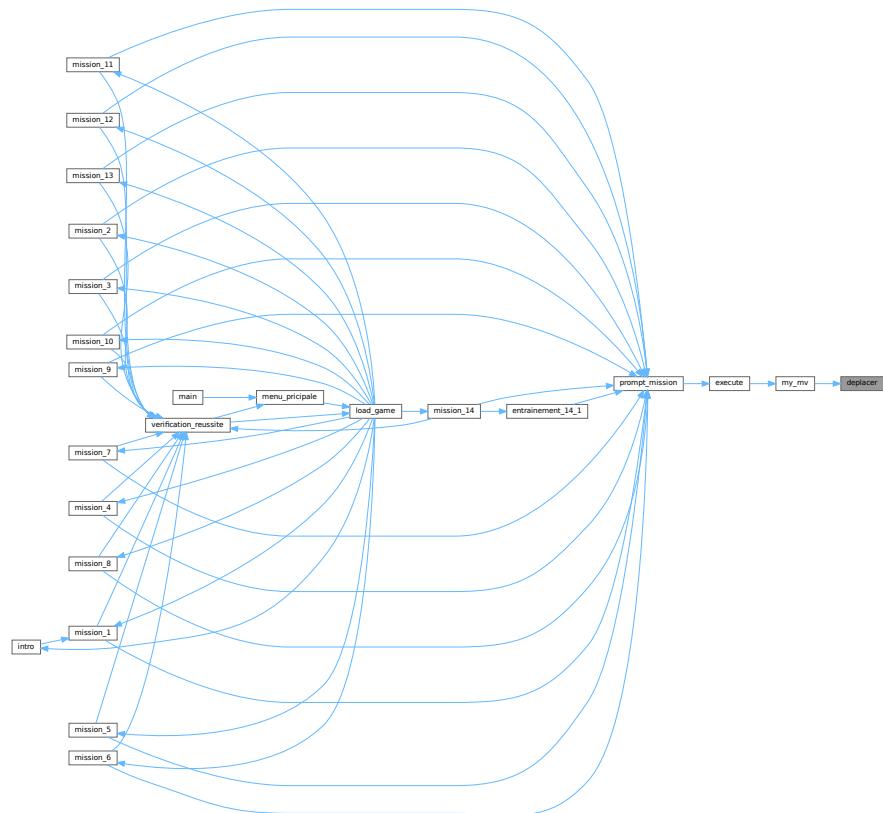
Déplace un fichier ou dossier d'une source vers une destination.

#### Parameters

<code>dest</code>	Pointeur vers le dossier de destination.
<code>source</code>	Pointeur vers le fichier ou dossier source à déplacer.

Definition at line 930 of file [arborescence.c](#).

Here is the caller graph for this function:



### 4.3.2.8 `favori()`

```
void favori (
    fichier * home,
    int max )
```

Affiche les fichiers ayant le nombre maximum d'ouvertures.

**Parameters**

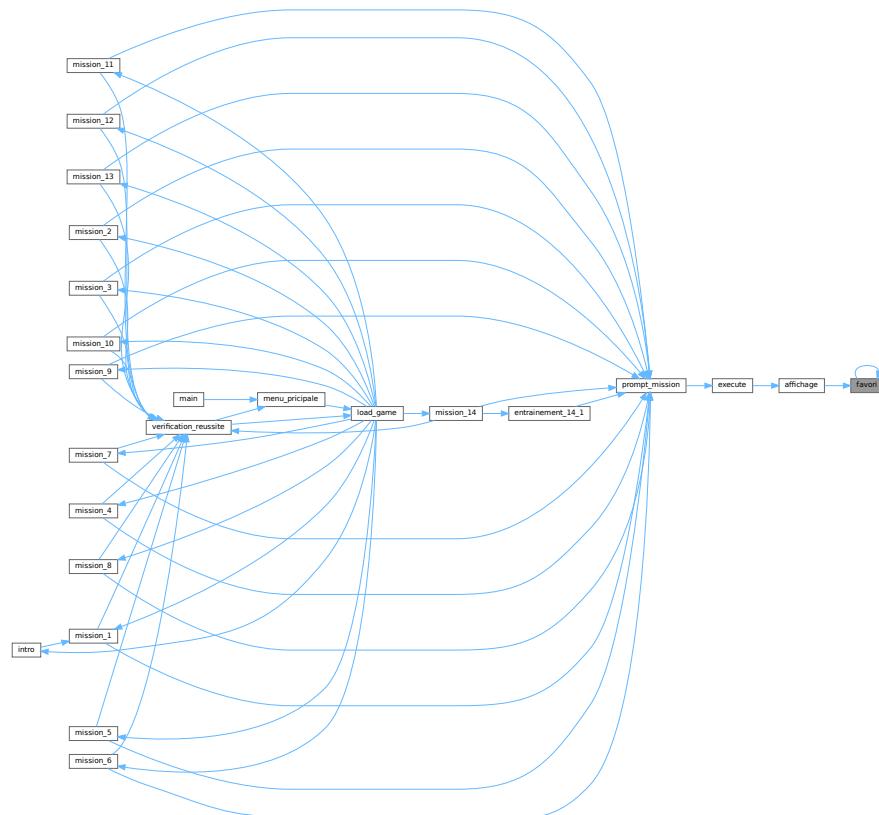
<i>home</i>	Pointeur vers le répertoire racine.
<i>max</i>	Nombre maximum d'ouvertures.

Definition at line 1516 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.9 initialiser()

```
void initialiser (
    fichier * racine )
```

Initialise l'arborescence des fichiers et dossiers avec une structure de base.

**Parameters**

<i>racine</i>	Pointeur vers le répertoire racine.
---------------	-------------------------------------

**4.3.2.10 my\_cd()**

```
fichier * my_cd (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Change le répertoire courant.

**Parameters**

<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du répertoire cible.

**Returns**

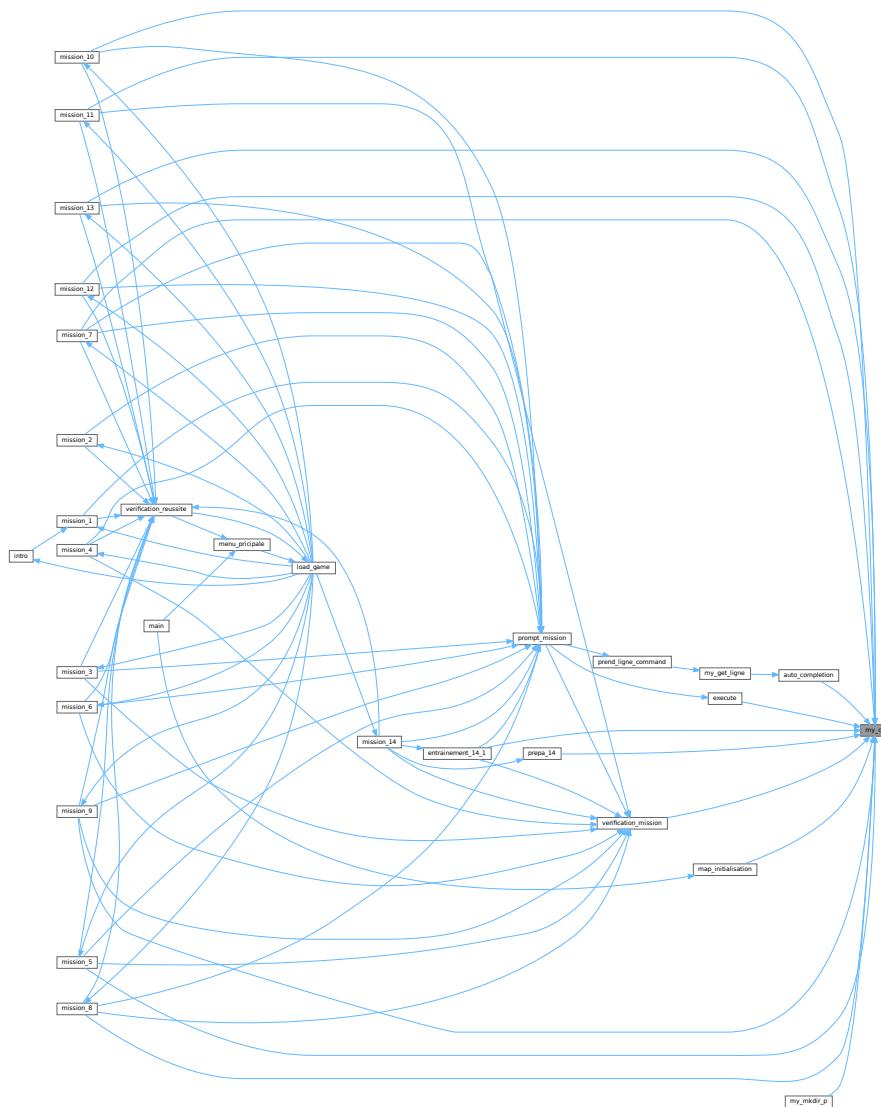
Pointeur vers le nouveau répertoire courant.

Definition at line 522 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.11 `my_chmod()`

```
void my_chmod (
    fichier * racine,
    fichier * courant,
    char * option,
    char * nom )
```

Modifie les permissions d'un fichier ou dossier.

##### Parameters

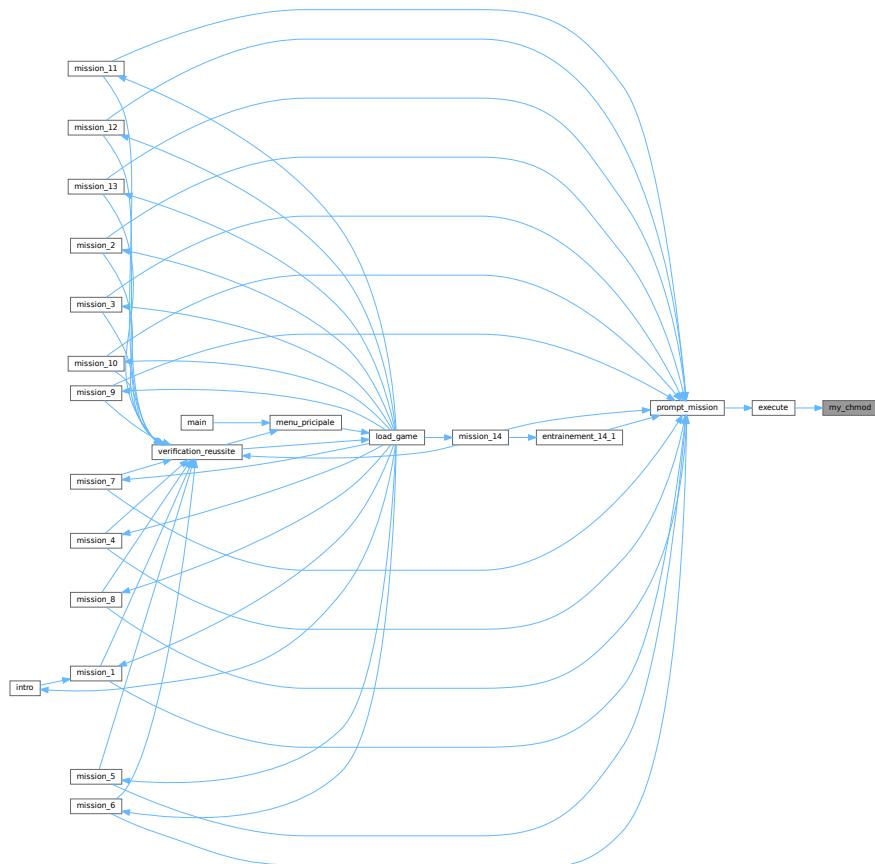
<code>racine</code>	Pointeur vers le répertoire racine.
<code>courant</code>	Pointeur vers le répertoire courant.
<code>option</code>	Chaîne de caractères représentant les options de permission.
<code>nom</code>	Nom du fichier ou dossier dont les permissions doivent être modifiées.

Definition at line 1259 of file arborescence.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.12 my\_cp()

```
void my_cp (
    fichier * racine,
    fichier * parent,
    char * nom,
    char * destination )
```

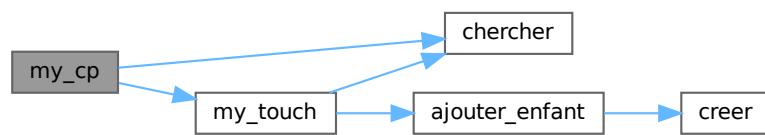
Copie un fichier ou dossier vers une destination.

## Parameters

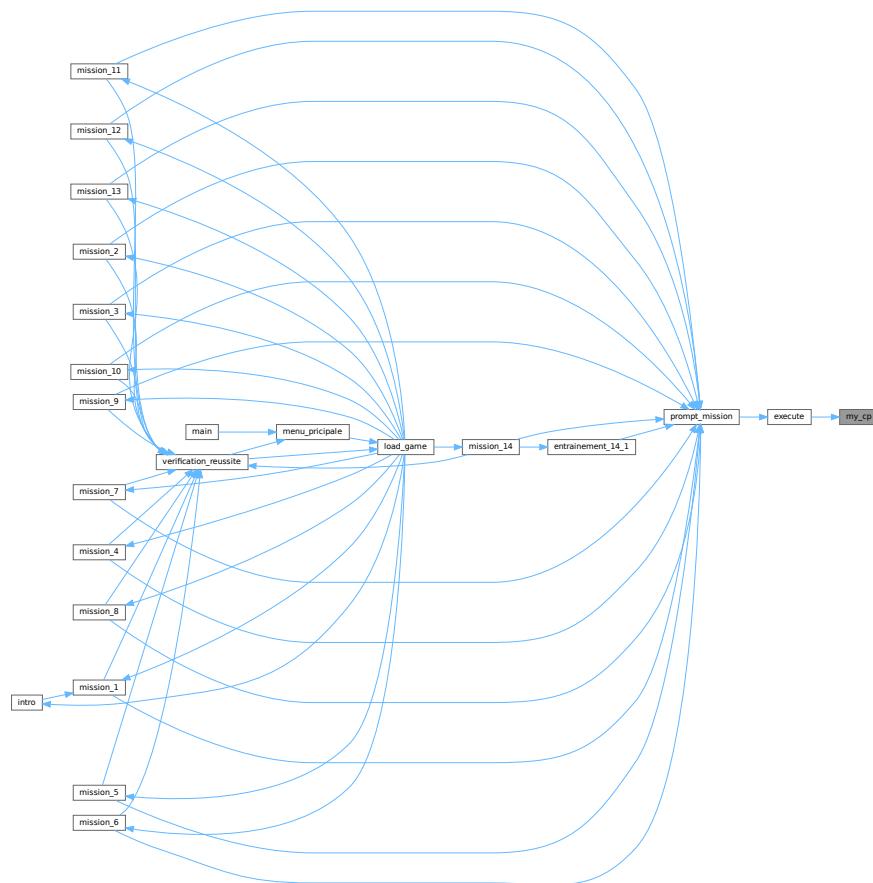
<i>racine</i>	Pointeur vers le répertoire racine.
<i>parent</i>	Pointeur vers le répertoire parent du fichier ou dossier à copier.
<i>nom</i>	Nom du fichier ou dossier à copier.
<i>destination</i>	Chemin de destination pour la copie.

Definition at line 611 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.3.2.13 my\_echo()

```
void my_echo (
    char ** ligne_c )
```

Affiche une ligne de texte.

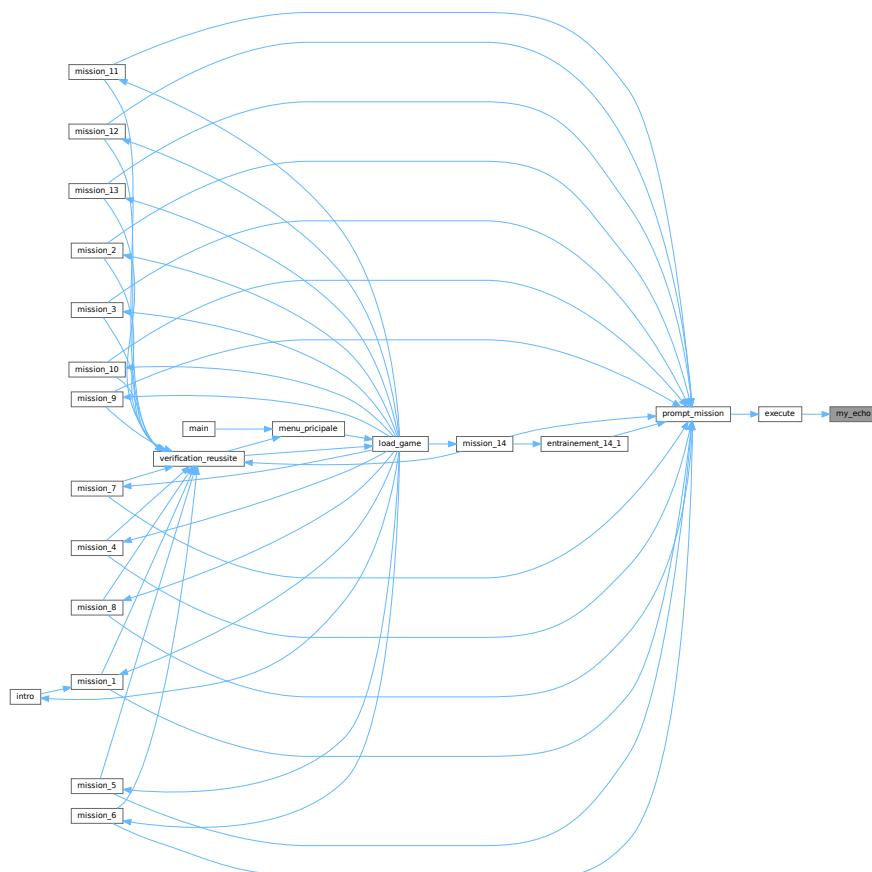
c'est comme la commande 'echo' mais en developpement

#### Parameters

<i>ligne_c</i>	Tableau de chaînes de caractères à afficher.
----------------	----------------------------------------------

Definition at line 1245 of file [arborescence.c](#).

Here is the caller graph for this function:



### 4.3.2.14 my\_ls()

```
void my_ls (
    fichier * racine,
```

```
fichier * courant,  
char * nom )
```

Liste les fichiers et dossiers dans le répertoire courant.

### Parameters

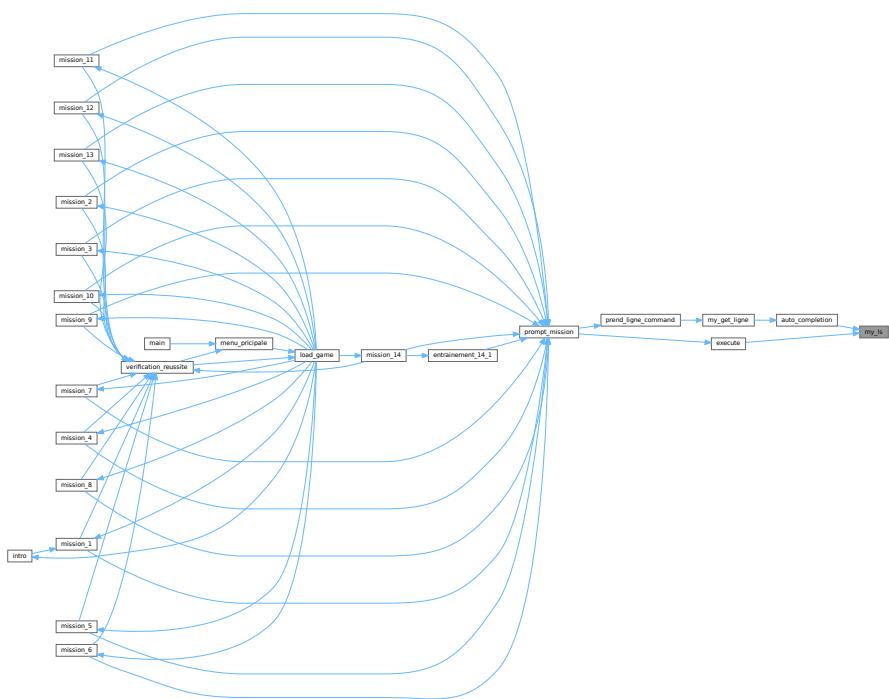
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du répertoire à lister.

Definition at line 192 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.3.2.15 my\_ls\_a()

```

void my_ls_a (
    fichier * racine,
    fichier * courant,
    char * nom )
  
```

Liste tous les fichiers et dossiers, y compris les fichiers cachés, dans le répertoire courant.

**Parameters**

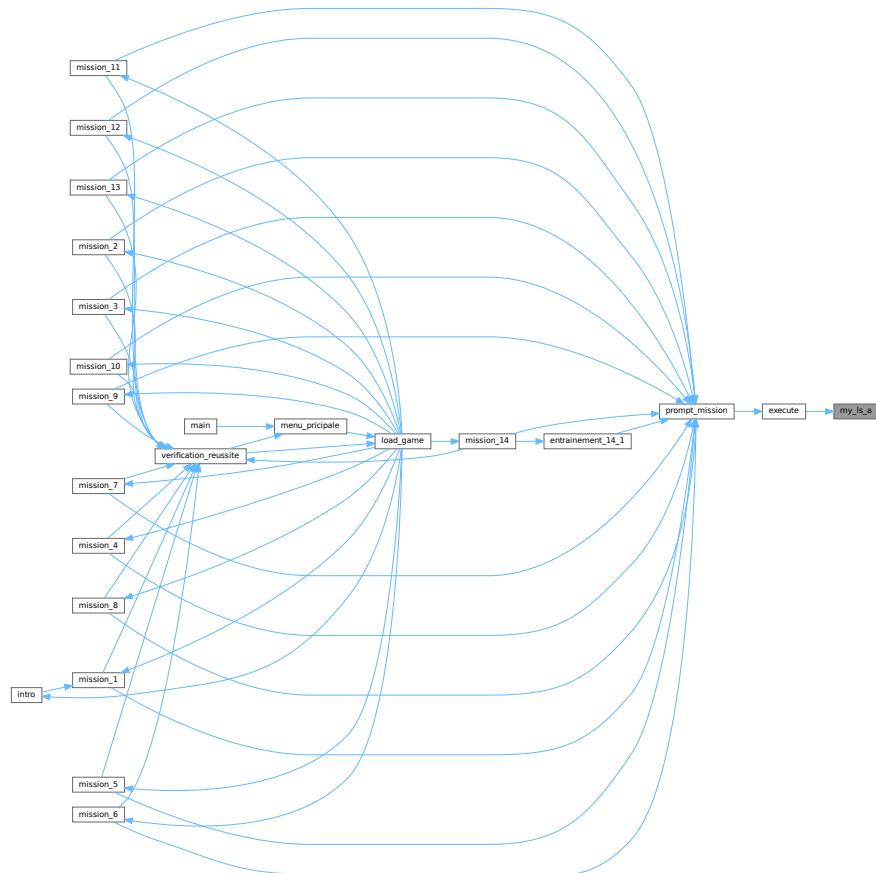
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du répertoire à lister.

Definition at line 296 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.16 my\_ls\_l()

```
void my_ls_l (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Liste les fichiers et dossiers avec des détails dans le répertoire courant.

##### Parameters

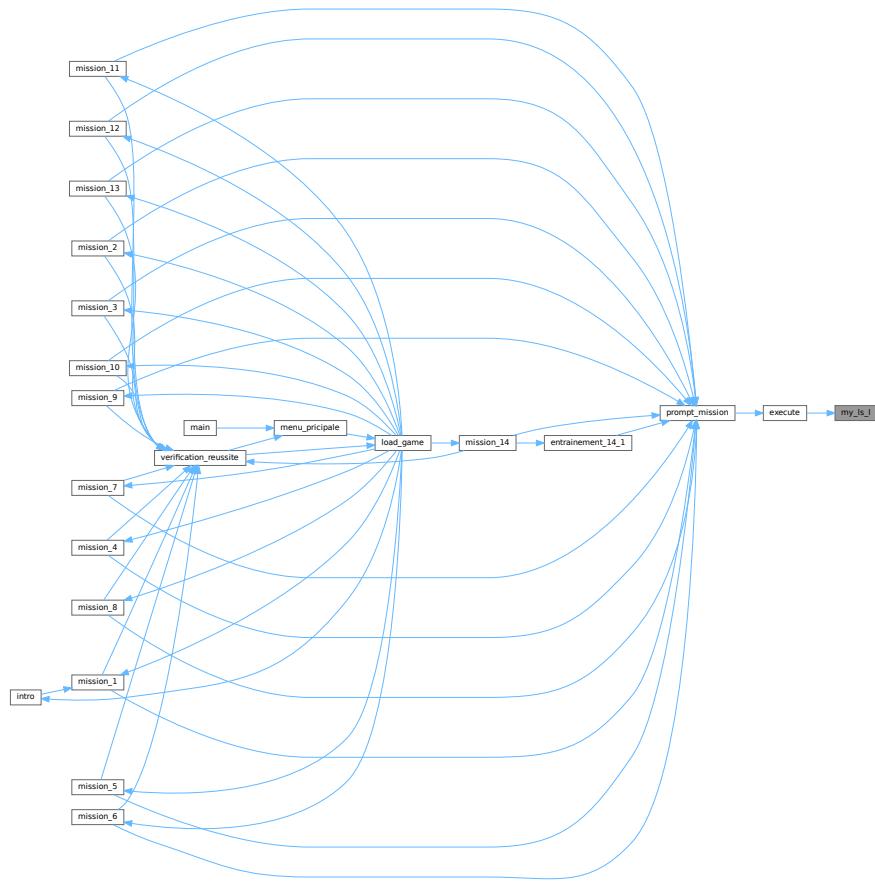
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du répertoire à lister.

Definition at line 1387 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.17 `my_ls_la()`

```
void my_ls_la (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Liste tous les fichiers et dossiers avec des détails, y compris les fichiers cachés, dans le répertoire courant.

##### Parameters

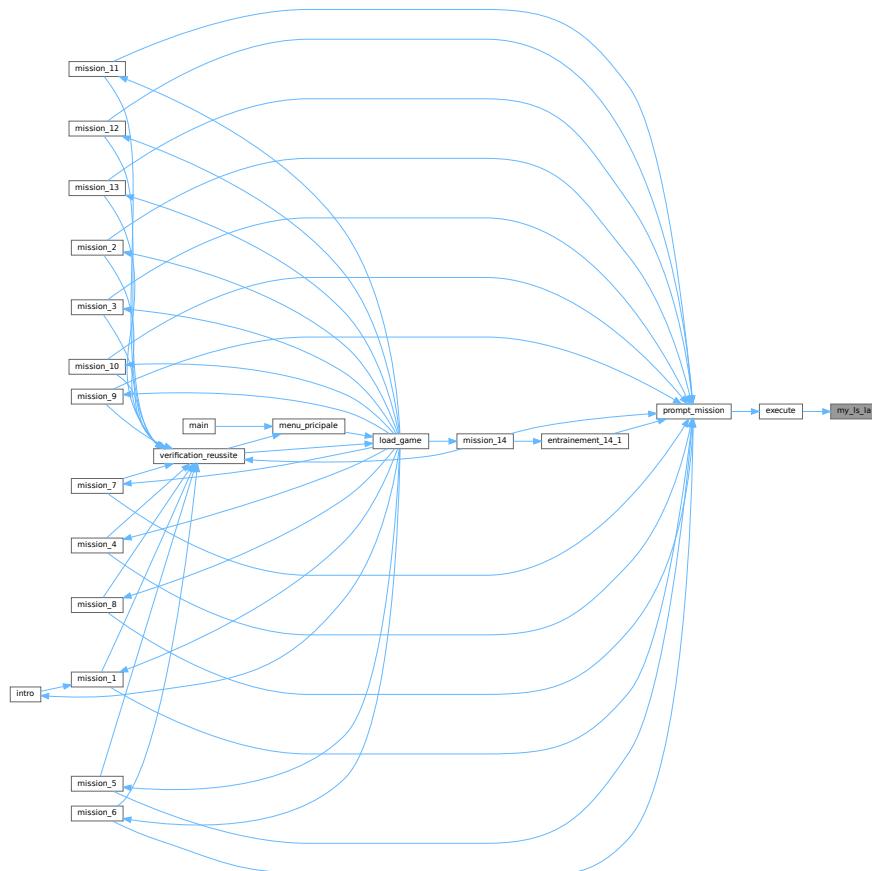
<code>racine</code>	Pointeur vers le répertoire racine.
<code>courant</code>	Pointeur vers le répertoire courant.
<code>nom</code>	Nom du répertoire à lister.

Definition at line 1537 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.18 my\_mkdir()

```

void my_mkdir (
    fichier * racine,
    fichier * courant,
    char * nom,
    int statut )
  
```

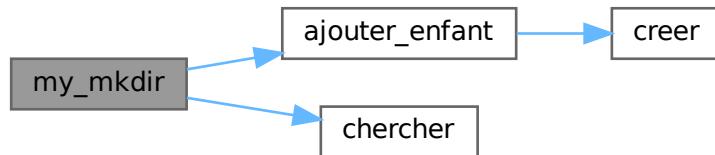
Crée un nouveau dossier dans le répertoire courant.

**Parameters**

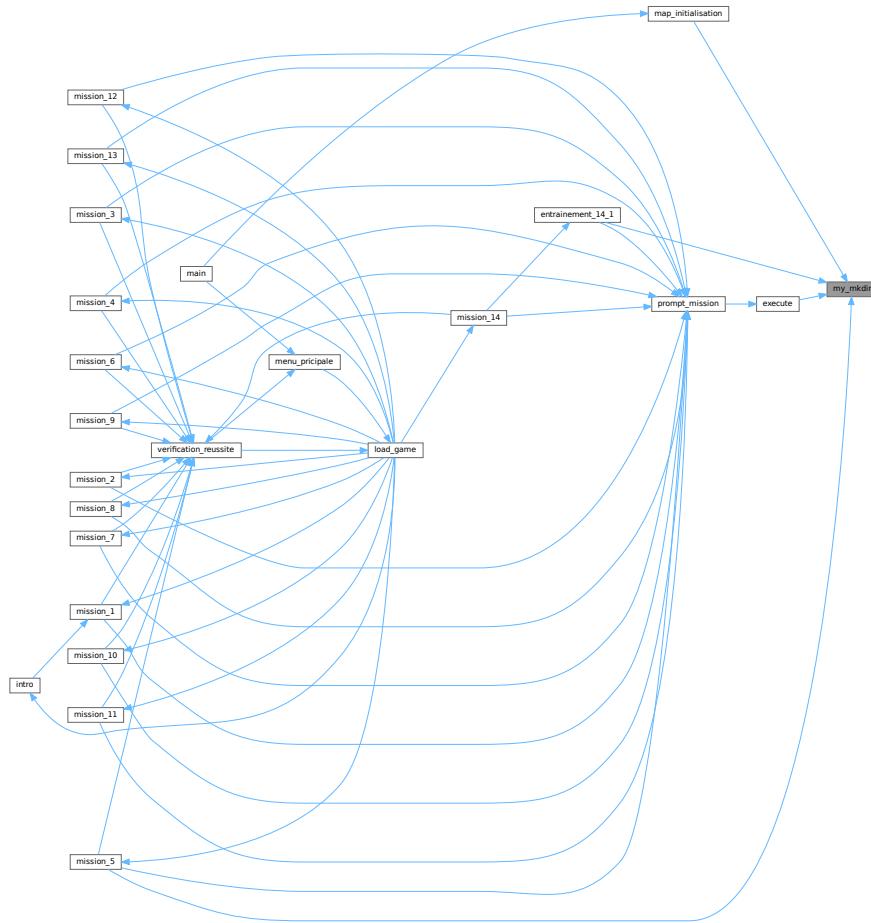
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du dossier à créer.
<i>statut</i>	Statut de protection du dossier.

Definition at line 1143 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.19 `my_mkdir_p()`

```
void my_mkdir_p (
    fichier * racine,
    fichier * parent,
    char * nom )
```

Crée un chemin de dossiers, créant les dossiers intermédiaires si nécessaire.

##### Parameters

<code>racine</code>	Pointeur vers le répertoire racine.
<code>parent</code>	Pointeur vers le répertoire parent.
<code>nom</code>	Chemin du dossier à créer.

Definition at line 1121 of file [arborescence.c](#).

Here is the call graph for this function:



#### 4.3.2.20 my\_mv()

```
void my_mv (
    fichier * racine,
    fichier * parent,
    char * nom,
    char * destination )
```

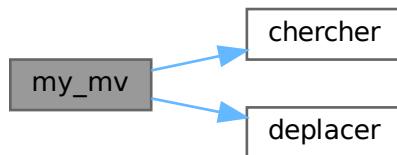
Déplace un fichier ou dossier vers une destination.

##### Parameters

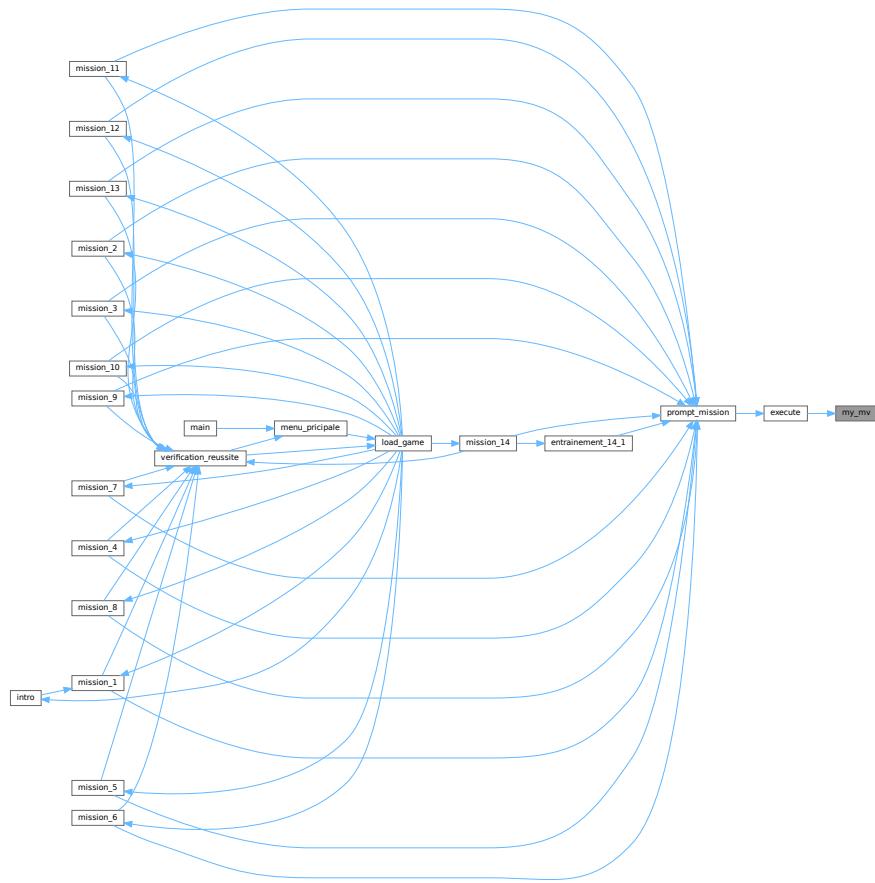
<i>racine</i>	Pointeur vers le répertoire racine.
<i>parent</i>	Pointeur vers le répertoire parent du fichier ou dossier à déplacer.
<i>nom</i>	Nom du fichier ou dossier à déplacer.
<i>destination</i>	Chemin de destination pour le déplacement.

Definition at line 751 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.21 my\_pwd()

```
void my_pwd ( fichier * rep_actuel )
```

Affiche le chemin complet du répertoire courant.

## Parameters

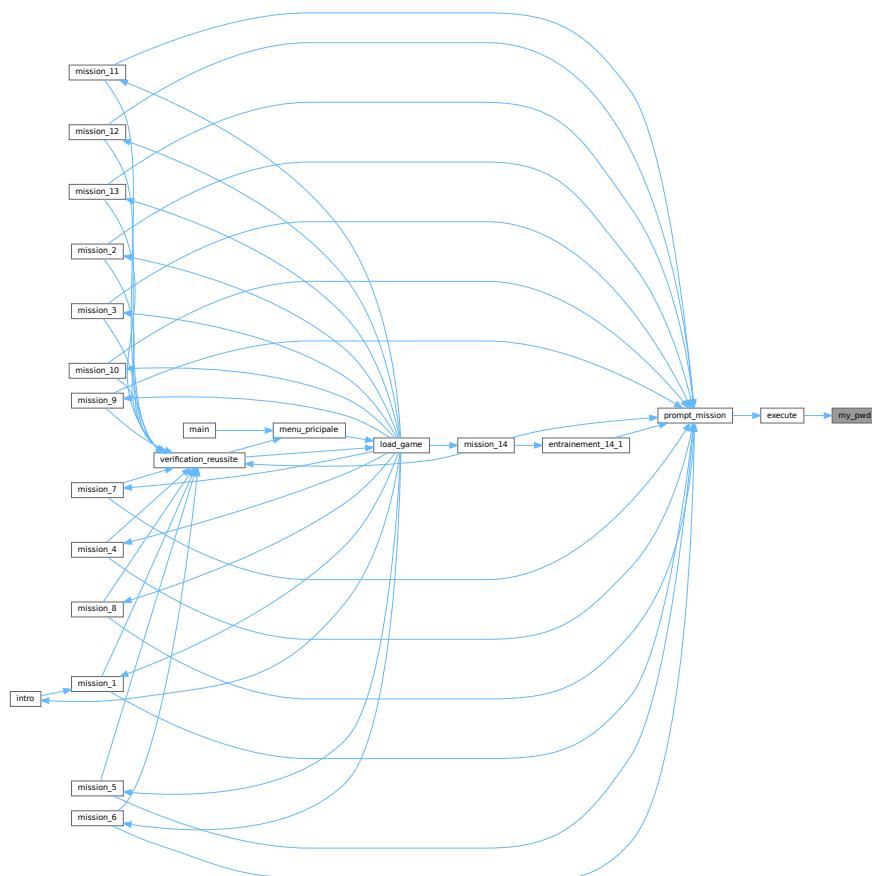
*rep\_actuel* Pointeur vers le répertoire courant.

Definition at line 1231 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.22 my\_rm()

```
void my_rm (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Supprime un fichier dans le répertoire courant.

**Parameters**

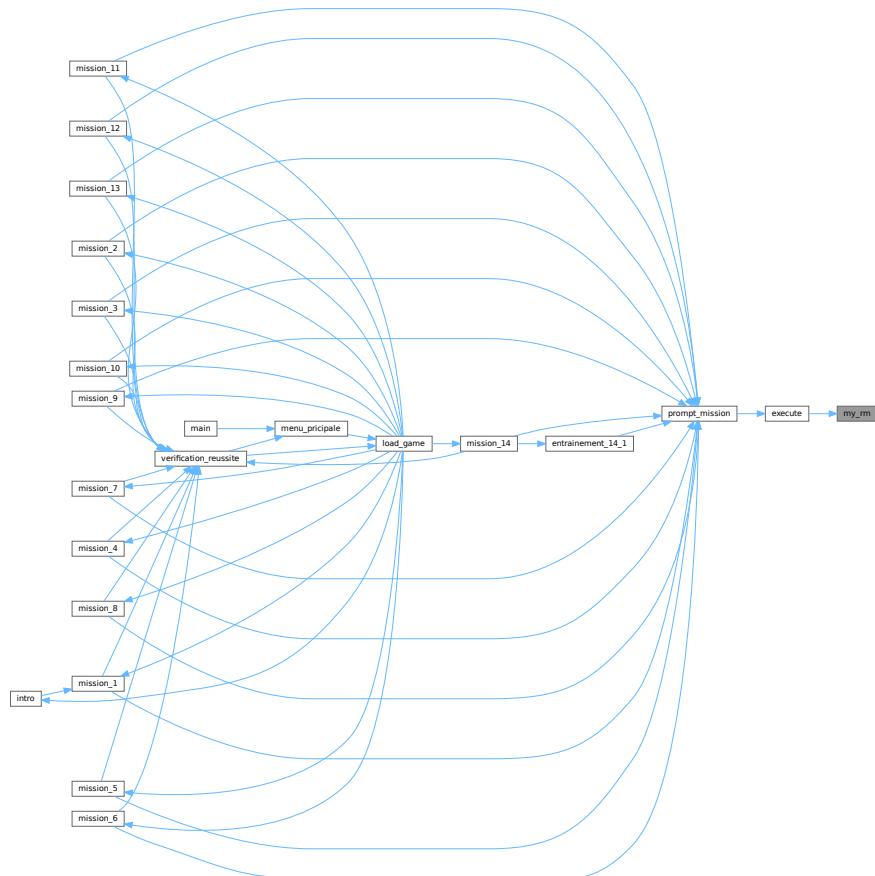
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du fichier à supprimer.

Definition at line 1006 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.3.2.23 my\_rmdir()

```
void my_rmdir (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Supprime un dossier dans le répertoire courant.

#### Parameters

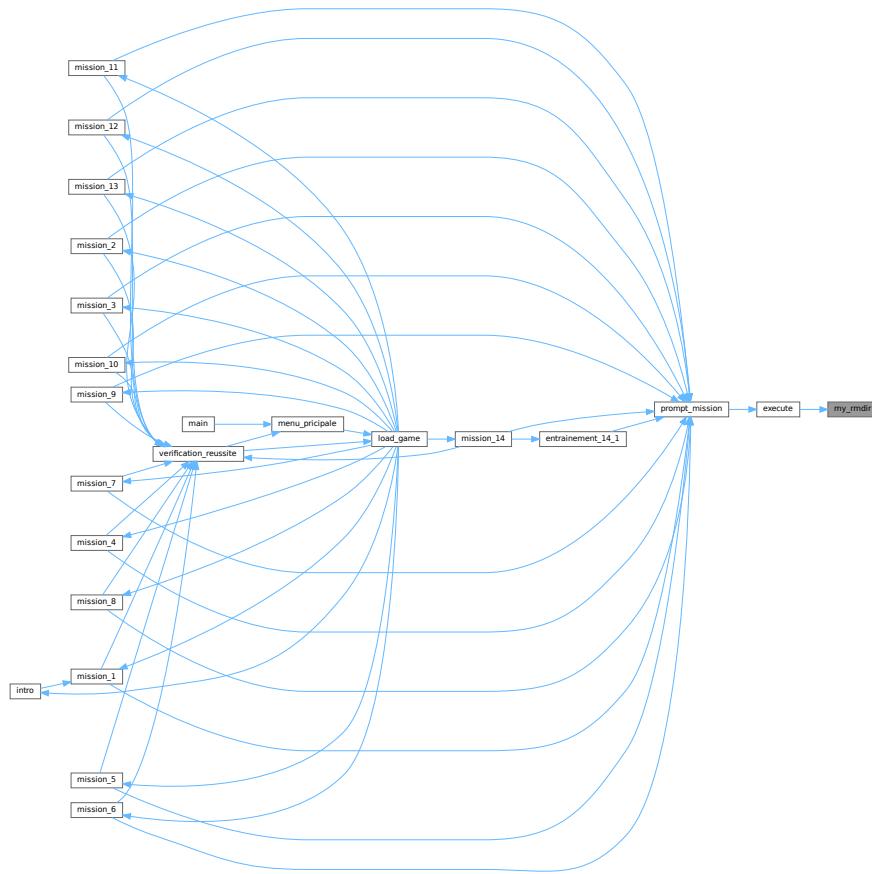
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du dossier à supprimer.

Definition at line 394 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.24 my\_touch()

```

void my_touch (
    fichier * racine,
    fichier * courant,
    char * nom,
    int statut )
  
```

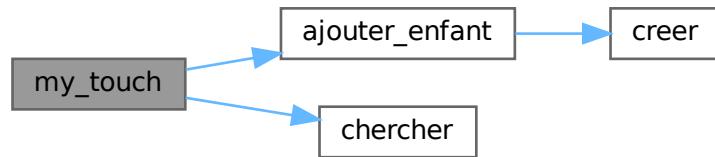
Crée un nouveau fichier dans le répertoire courant.

##### Parameters

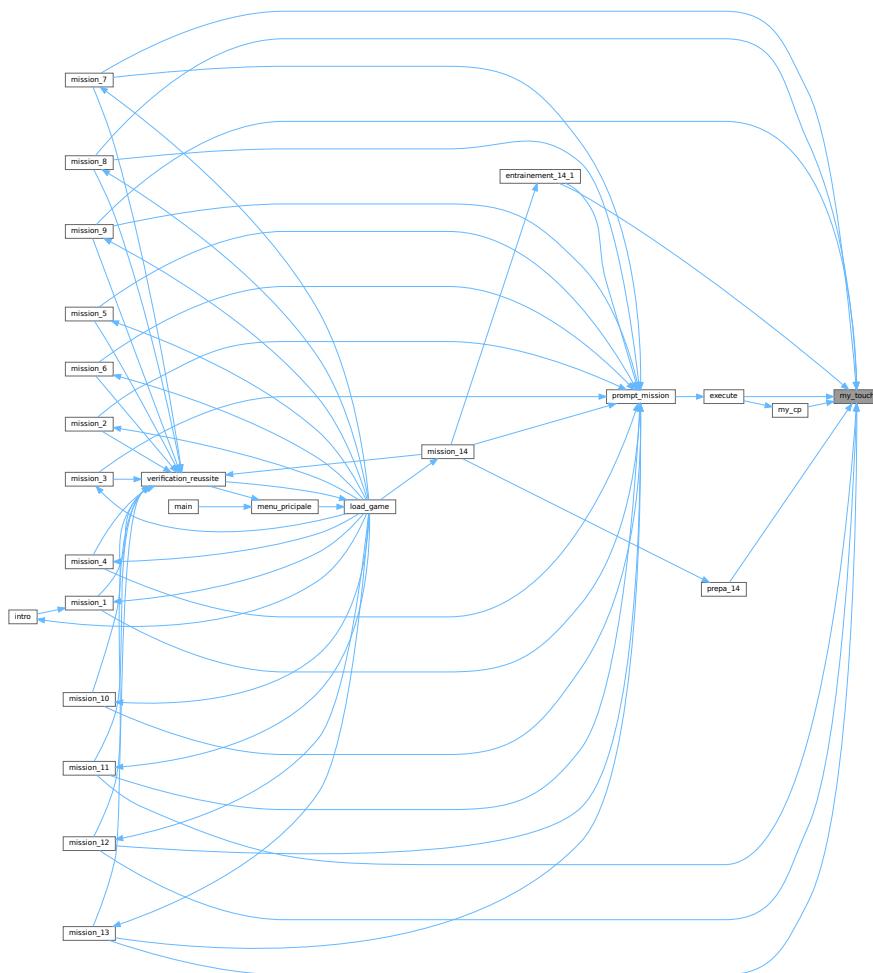
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du fichier à créer.
<i>statut</i>	Statut de protection du fichier.

Definition at line 71 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.2.25 recherche\_max()

```
int recherche_max (
    fichier * home,
    int max )
```

Recherche le nombre maximum d'ouvertures parmi les fichiers dans l'arborescence.

#### Parameters

<i>home</i>	Pointeur vers le répertoire racine.
<i>max</i>	Nombre maximum d'ouvertures trouvé jusqu'à présent.

#### Returns

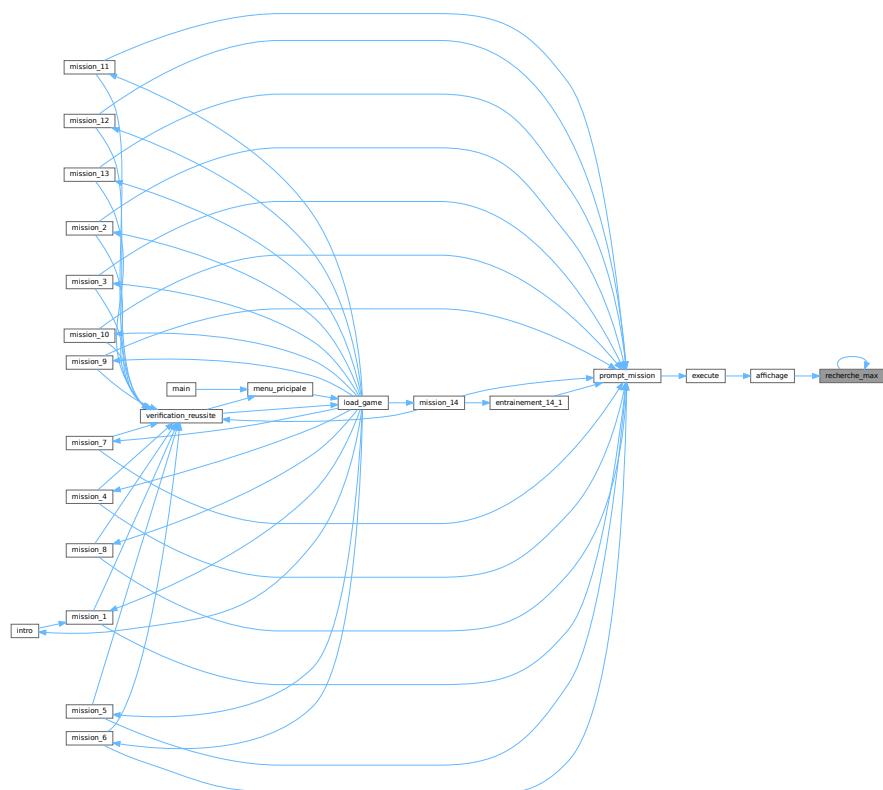
Nombre maximum d'ouvertures.

Definition at line 1501 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.4 arborescence.h

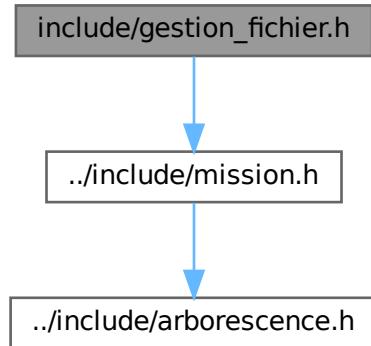
[Go to the documentation of this file.](#)

```
00001 #ifndef declaration
00002 #define declaration
00003 struct fichier
00004 {
00005     char nom[20] ;
00006     int estDossier ;
00007     int estProtege ;
00008     int nombre_ouverture;
00009     char perm[10] ;
00010     char **contenu ;
00011     struct fichier* parent;
00012     struct fichier* premierfils ;
00013     struct fichier* frereSuivant ;
00014 } ;
00015 typedef struct fichier fichier ;
00025 fichier* creer(fichier *parent, char *nom, int estDossier, int statut) ;
00034 void ajouter_enfant(fichier *parent, char *nom, int nombre, int statut) ;
00043 void my_mkdir(fichier *racine, fichier *courant, char *nom,int statut) ;
00050 void afficher(fichier* home, int prfnd) ;
00058 void my_ls(fichier* racine, fichier *courant,char *nom) ;
00067 void my_touch(fichier *racine, fichier *courant, char *nom, int statut) ;
00075 void my_rmdir(fichier *racine,fichier *courant, char *nom) ;
00084 fichier* my_cd(fichier *racine,fichier *courant, char *nom) ;
00090 void initialiser(fichier *racine) ;
00099 void my_cp(fichier *racine, fichier *parent, char *nom, char *destination) ;
00107 fichier* chercher(fichier *courant, char *nom) ;
00116 void my_mv(fichier *racine, fichier *parent, char *nom, char *destination) ;
00124 void my_rm(fichier *racine, fichier *courant, char *nom) ;
00132 void my_mkdir_p(fichier *racine, fichier *parent, char *nom) ;
00138 void my_pwd(fichier *rep_actuel);
00144 void my_echo(char **ligne_c);
00152 void my_ls_a(fichier* racine, fichier *courant,char *nom) ;
00159 void replacer(fichier* dest, fichier* source) ;
00165 fichier* chercher(fichier *courant, char *nom) ;
00174 void my_chmod(fichier* racine, fichier* courant, char* option, char* nom) ;
00182 void my_ls_l(fichier* racine, fichier *courant,char *nom) ;
00189 void afficher_nbr_ouverture(fichier* home, int prfnd) ;
00197 int recherche_max(fichier* home, int max) ;
00204 void favori(fichier* home, int max) ;
00210 void affichage(fichier* home) ;
00218 void my_ls_la(fichier* racine, fichier *courant,char *nom) ;
00219
00220 #endif
```

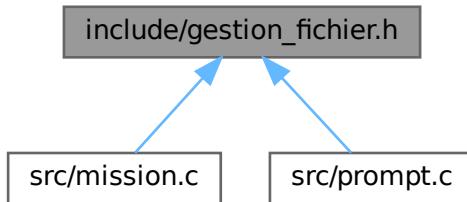
## 4.5 include/gestion\_fichier.h File Reference

regroupe tous les fonctions qui manipule les fichiers

```
#include "../include/mission.h"
Include dependency graph for gestion_fichier.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- void [head\\_fichier](#) (char \*nom\_fichier, int max)
 

*Il affiche des premiers lignes d'un fichier entrer en parametre*
- int [affiche\\_ligne](#) (char \*chemin, int num\_ligne, char \*emplacement)
 

*Lire une ligne precise dans un fichier depart de la fin si negative.*
- int [my\\_man](#) (char \*nom\_commande)
 

*Il affiche une page d'aide pour une commande entrer en argument.*
- void [qui\\_est\\_meilleur](#) (int n\_miss)
 

*cette fonction affche qui a le meilleur temps concernant une mission entrer en argument*
- void [voir\\_indice](#) (Mission \*mission)
 

*Elle affiche une petite aide apropos d'une mission en decrementant le score du joueur actuelle.*
- int [lire\\_histoire](#) ()
 

*cette fonction lit le fichier historique des commandes*
- void [tail\\_fichier](#) (char \*nom\_fichier, int min)

- void [cat\\_fichier](#) (char \*nom\_fichier, bool num)  
*afficher le fichier entrée en paramètre*
- int [my\\_history](#) (bool effacer)  
*afficher ou effacer les historiques de ligne de commande*
- void [insert\\_fin](#) (char \*nom\_fichier, char \*chaine\_caractere)

### 4.5.1 Detailed Description

regroupe tous les fonctions qui manipule les fichiers

#### Author

Valisoa

#### Date

29 Août 2025

toutes les manipulations de fichier comme les enregistrements et les affichages sont regroupés dans celle-ci ; ces fonctions sont définies dans gestion\_fichier.c

Definition in file [gestion\\_fichier.h](#).

### 4.5.2 Function Documentation

#### 4.5.2.1 affiche\_ligne()

```
int affiche_ligne (
    char * chemin,
    int num_ligne,
    char * emplacement )
```

Lire une ligne précise dans un fichier départ de la fin si négative.

#### Parameters

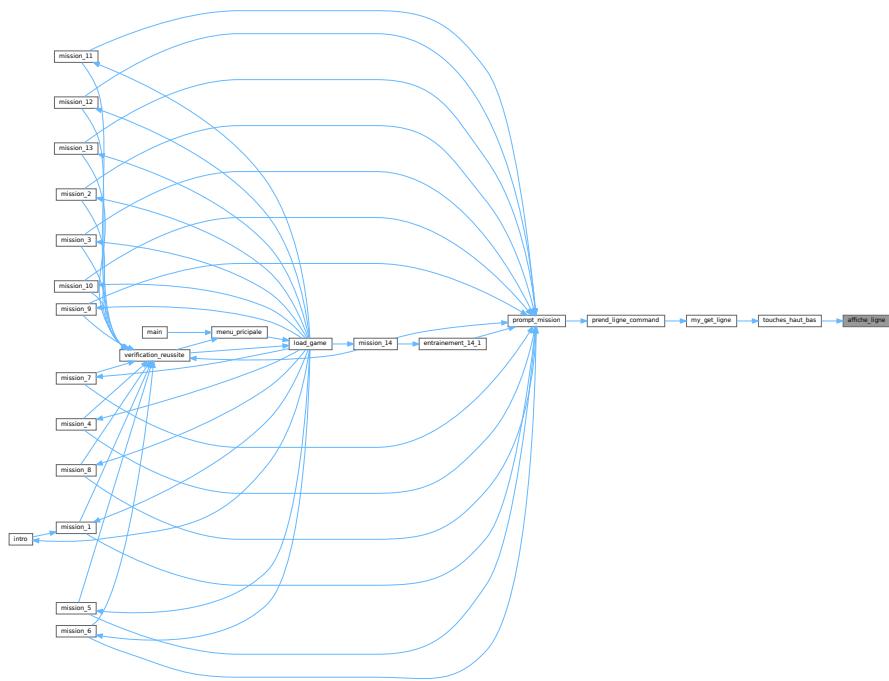
<i>chemin</i>	le chemin vers la fichier
<i>num_ligne</i>	le numéro de ligne à afficher
<i>emplacement</i>	la variable qui va contenir la ligne voulue

#### Returns

renvoie une valeur négative si le fichier n'est pas existant

Definition at line 51 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:



#### 4.5.2.2 cat\_fichier()

```
void cat_fichier (
    char * nom_fichier,
    bool num )
```

afficher le fichier entrée en parametre

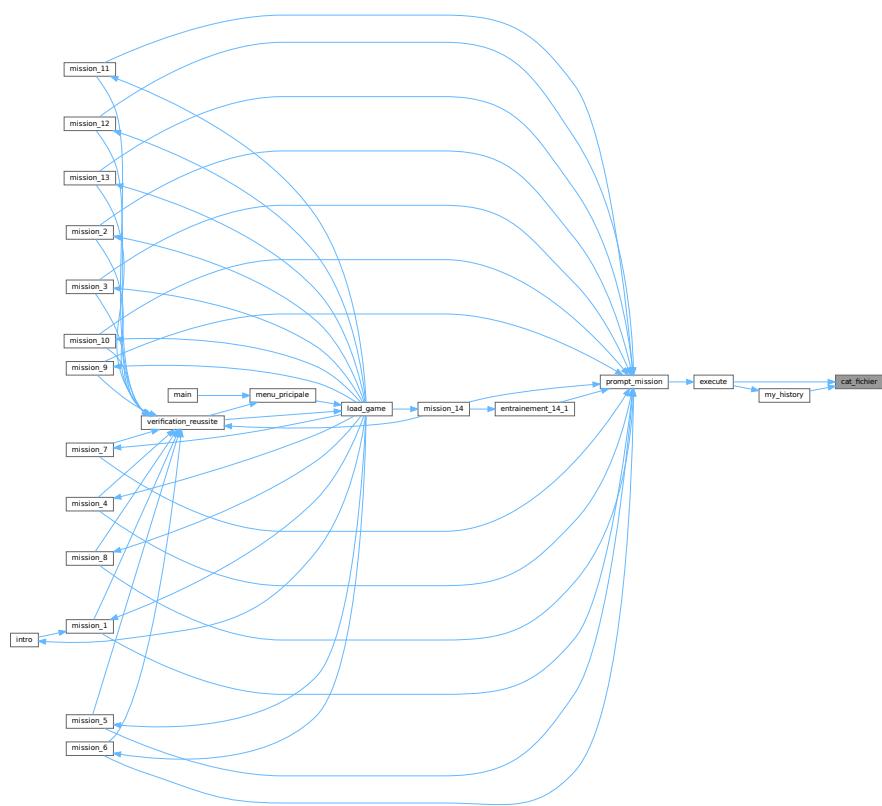
afficher le contenu d'un fichier

##### Parameters

<i>nom_fichier</i>	nom de la fichier à afficher
<i>nom_fichier</i>	le nom de fichier à afficher
<i>num</i>	affiche le numero de ligne si true est la valeur de celle ci

Definition at line 20 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:



#### 4.5.2.3 head\_fichier()

```
void head_fichier (
    char * nom_fichier,
    int max )
```

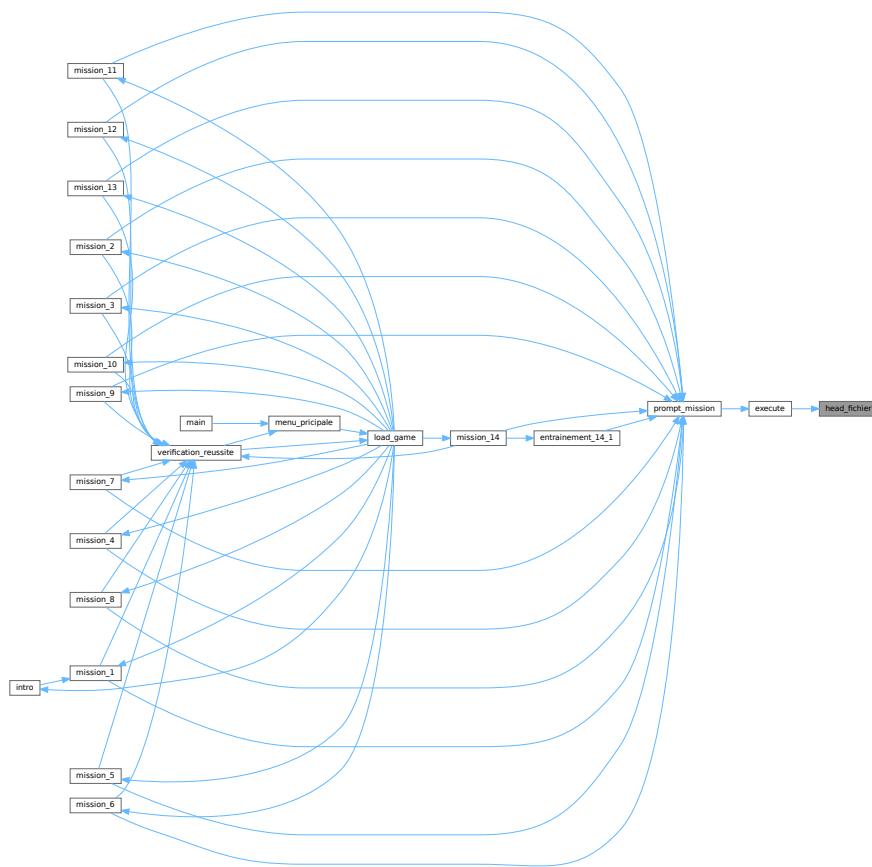
il affiche des premiers lignes d'une fichier entrer en parametre

##### Parameters

<i>nom_fichier</i>	le nom de la fichier
<i>max</i>	le nombre de ligne à afficher

Definition at line 149 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:



#### 4.5.2.4 insert\_fin()

```
void insert_fin (
    char * nom_fichier,
    char * chaine_caractere )
```

Definition at line 134 of file [gestion\\_fichier.c](#).

#### 4.5.2.5 lire\_histoire()

```
int lire_histoire ( )
cette fonction lit le fichier historique des commandes
```

##### Returns

une valeur negatif si le fichier histoire est vide ou inexistant

Definition at line 229 of file [gestion\\_fichier.c](#).

#### 4.5.2.6 my\_history()

```
int my_history (
    bool effacer )
```

afficher ou effacer les historiques de ligne de commande

**Parameters**

<code>effacer</code>	effacer le contenu historique si true
----------------------	---------------------------------------

**Returns**

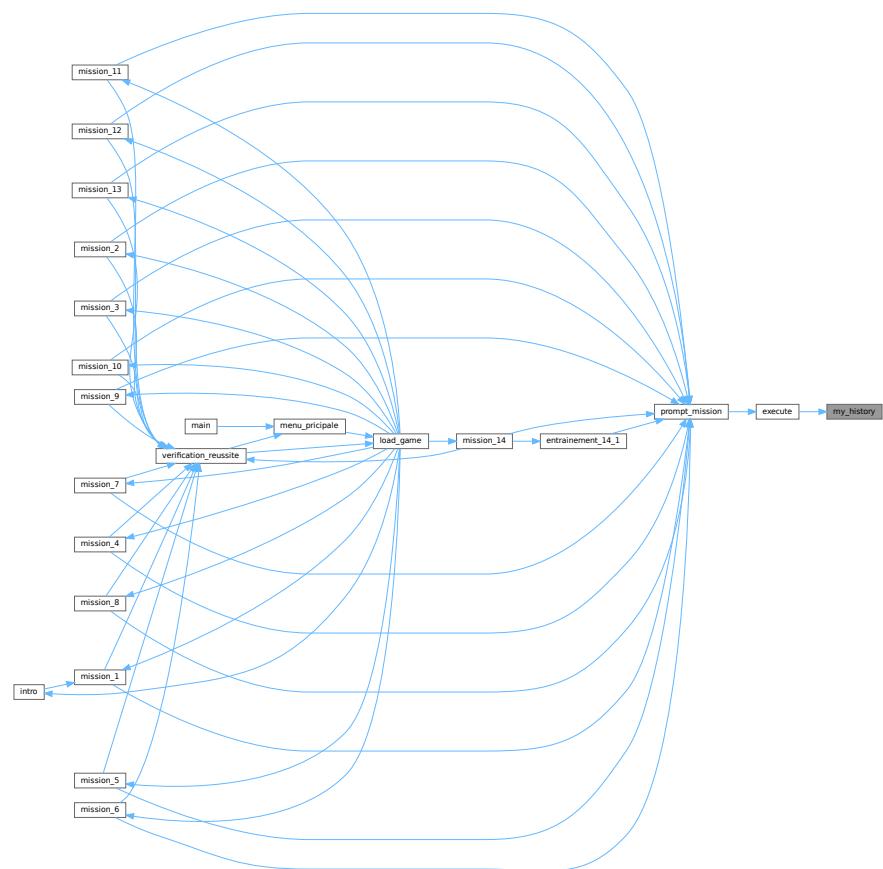
une valeur negative si la lecture a echouée

Definition at line 378 of file [gestion\\_fichier.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.5.2.7 my\_man()

```
int my_man (
    char * nom_commande )
```

Il affiche une page d'aide pour une commande entrer en argument.

##### Parameters

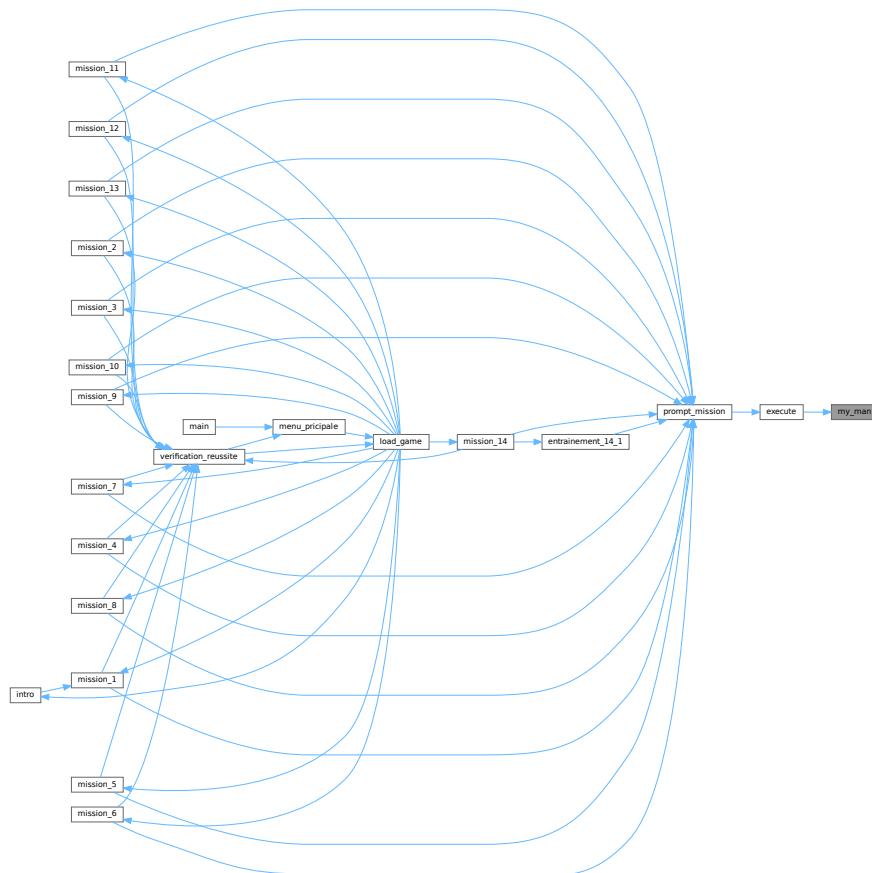
<i>nom_commande</i>	le nom de la commande à afficher l'aide
---------------------	-----------------------------------------

##### Returns

une valeur negative en cas d'erreur

Definition at line 353 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:



#### 4.5.2.8 qui\_est\_meilleur()

```
void qui_est_meilleur (
    int n_miss )
```

cette fonction affche qui a le meilleur temps concernant une mission entrer en argument

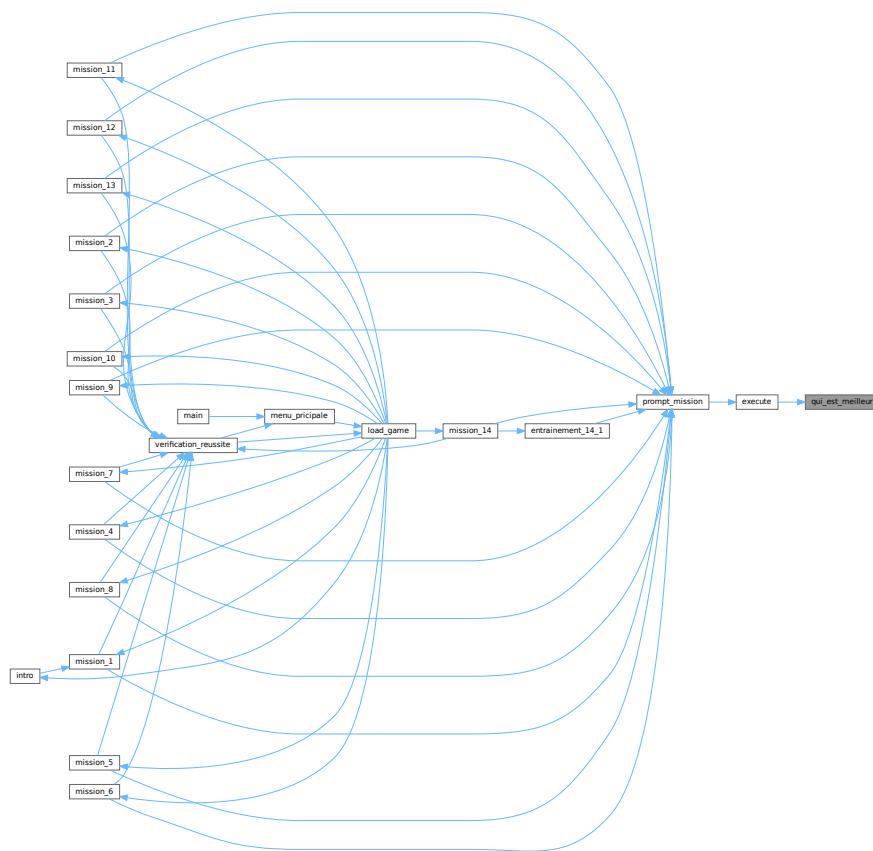
## Parameters

<i>n_miss</i>	le numero de la mission
---------------	-------------------------

traitement de donnée obtenus d'après la lecture du fichier de score

Definition at line 295 of file [gestion\\_fichier.c](#).

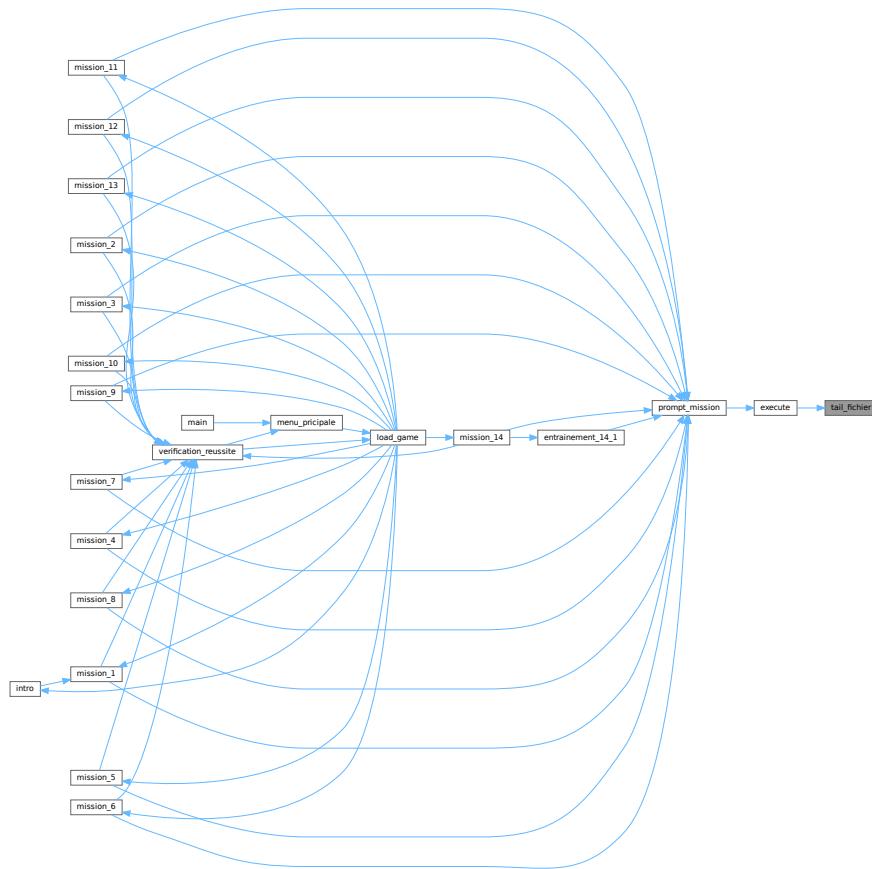
Here is the caller graph for this function:

**4.5.2.9 tail\_fichier()**

```
void tail_fichier (
    char * nom_fichier,
    int min )
```

Definition at line 173 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:



#### 4.5.2.10 voir\_indice()

```
void voir_indice (
    Mission * mission )
```

Elle affiche une petite aide apropos d'une mission en decrementant le score du joueur actuelle.

##### Parameters

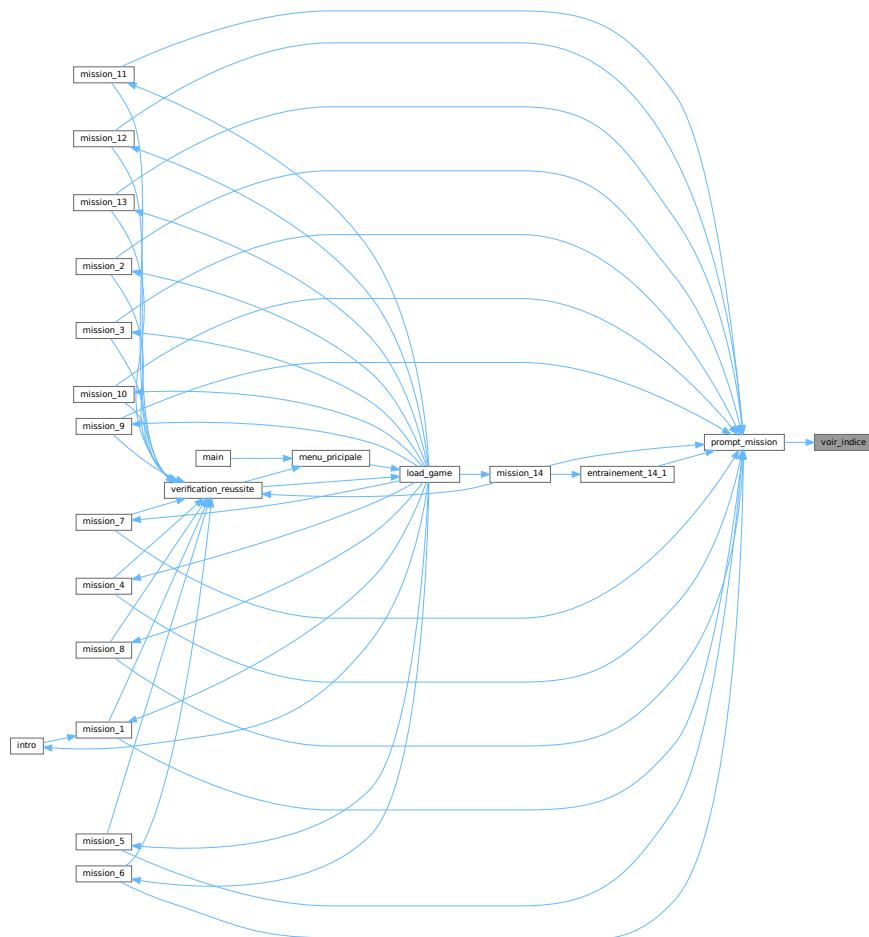
<i>mission</i>	l'adresse du mission que l'on veut voir l'indice
----------------	--------------------------------------------------

Definition at line 255 of file [gestion\\_fichier.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.6 gestion\_fichier.h

[Go to the documentation of this file.](#)

```
00001
00010 #ifndef GEST_FIC
00011 #define GEST_FIC
00012     #include "../include/mission.h"
00013
00020 void head_fichier(char *nom_fichier , int max);
00029 int affiche_ligne(char *chemin , int num_ligne , char *emplacement);
```

```

00036     int my_man(char *nom_commande);
00042     void qui_est_meilleur(int n_miss);
00048     void voir_indice(Mission *mission);
00049
00055     int lire_histoire();
00061     void tail_fichier(char *nom_fichier , int min);
00068     void cat_fichier(char *nom_fichier , bool num);
00075     int my_history(bool effacer);
00076
00082     void insert_fin(char *nom_fichier, char *chaine_caractere);
00083
00084 #endif

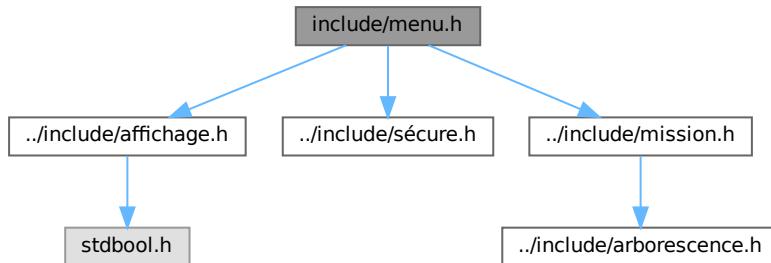
```

## 4.7 include/menu.h File Reference

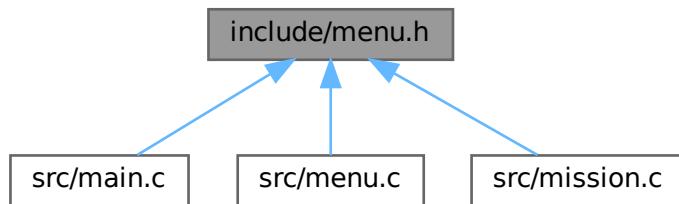
```

#include "../include/affichage.h"
#include "../include/sécuré.h"
#include "../include/mission.h"
Include dependency graph for menu.h:

```



This graph shows which files directly or indirectly include this file:



### Functions

- void `menu_principale (Mission *mission, fichier *racine)`
- void `load_game (Mission *mission, int rang_du_dernier_mission, fichier *racine)`
- void `ptit_prompt (char *nom)`

- int `charger_partie` (`Mission` \*mission)
- void `quit_game` (`Mission` \*mission)
- void `logging` ()  
*Fonction pour gérer la connexion de l'utilisateur.*
- int `dossierExiste` (const char \*chemin)

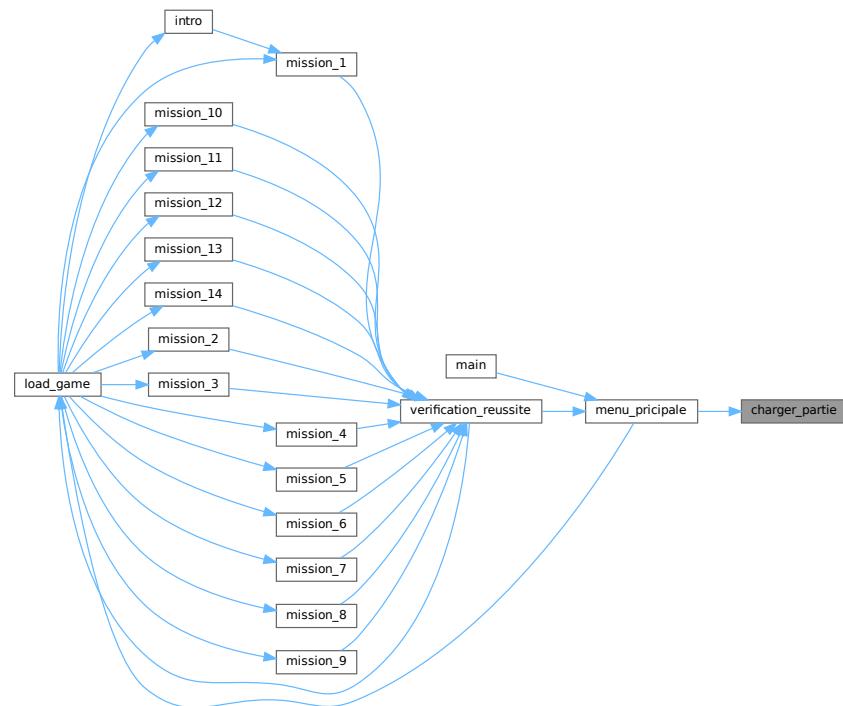
## 4.7.1 Function Documentation

### 4.7.1.1 charger\_partie()

```
int charger_partie (
    Mission * mission )
```

Definition at line 112 of file `menu.c`.

Here is the caller graph for this function:



### 4.7.1.2 dossierExiste()

```
int dossierExiste (
    const char * chemin )
```

Definition at line 18 of file `menu.c`.

Here is the caller graph for this function:

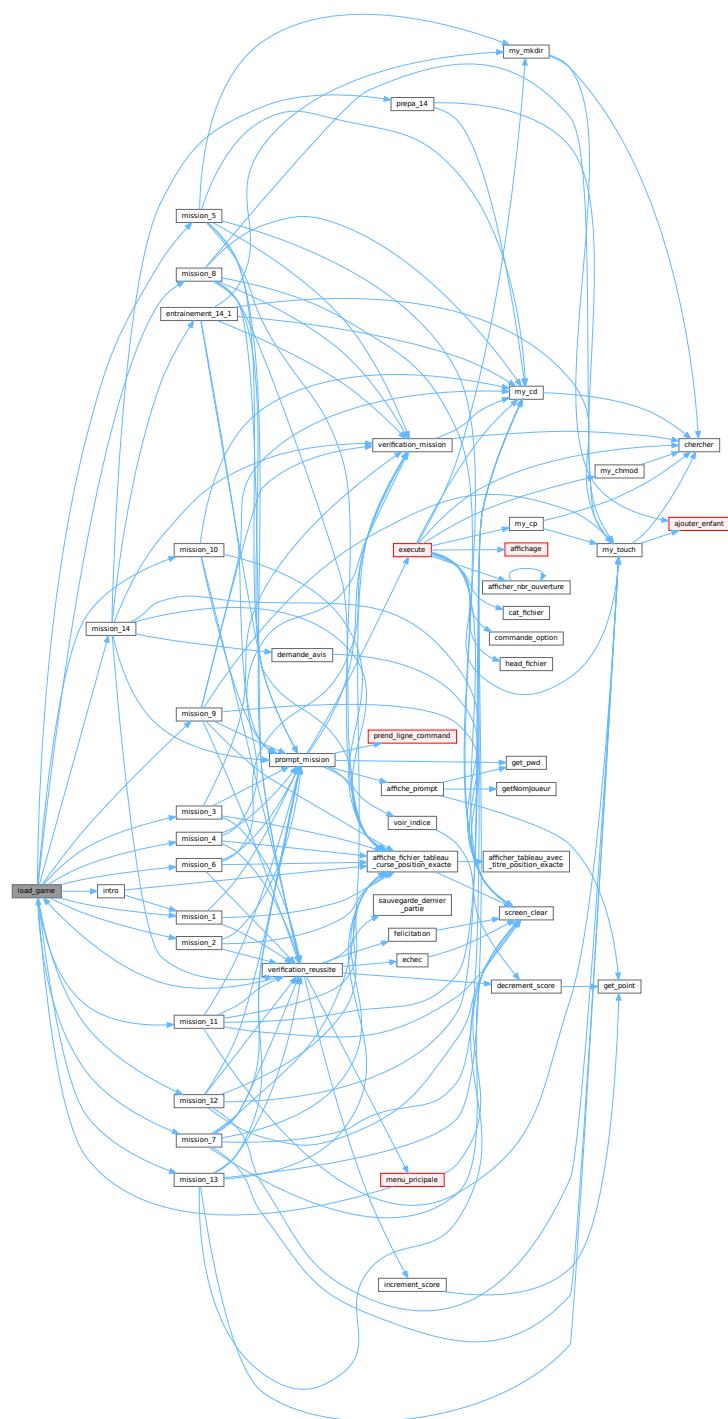


#### 4.7.1.3 `load_game()`

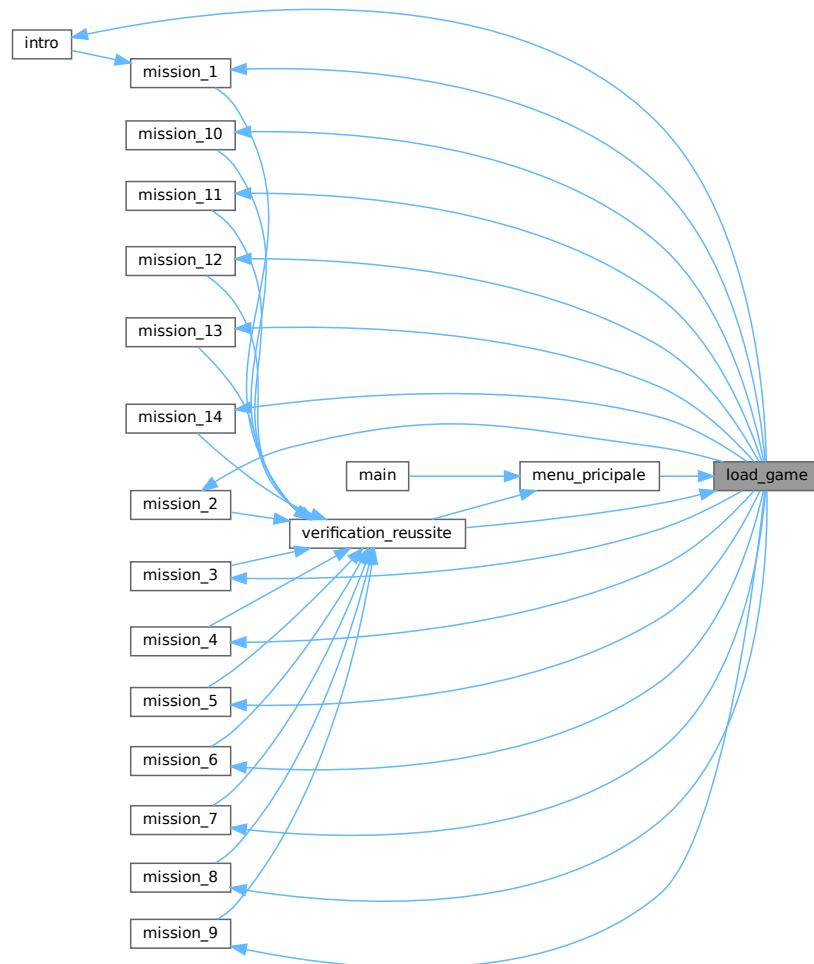
```
void load_game (
    Mission * mission,
    int rang_du_dernier_mission,
    fichier * racine )
```

Definition at line [95](#) of file [menu.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.7.1.4 loging()

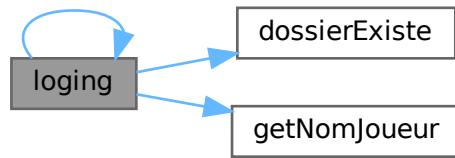
```
loging ( )
```

Fonction pour gérer la connexion de l'utilisateur.

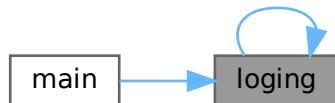
: Fonction qui récupère le loging du joueur et accorde le sauvegarde qui lui est attribué s'il a déjà jouer si non crée un nouveau dossier de sauvegarde pour l'user

Definition at line 36 of file [menu.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

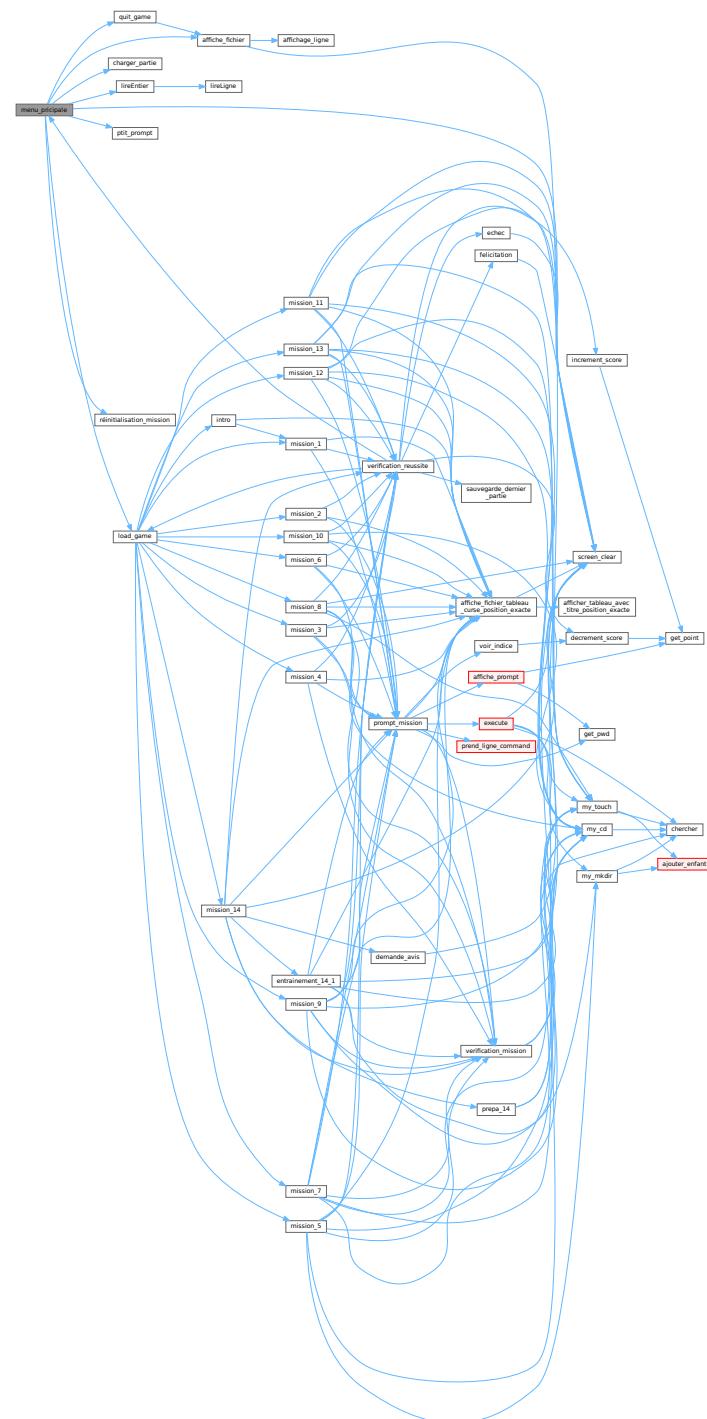


#### 4.7.1.5 menu\_principale()

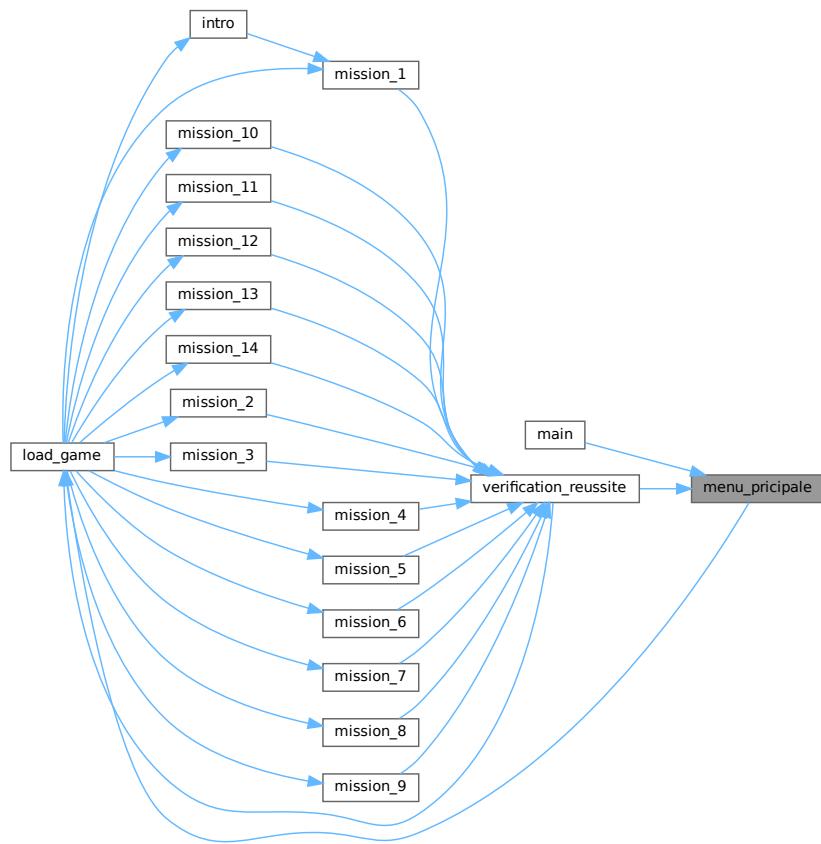
```
void menu_principale (
    Mission * mission,
    fichier * racine )
```

Definition at line [261](#) of file [menu.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

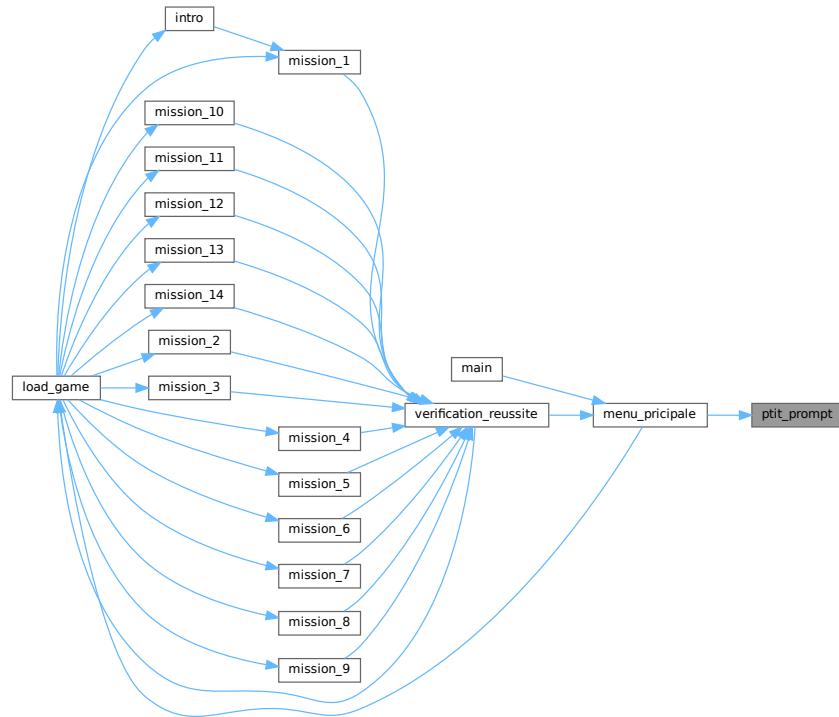


#### 4.7.1.6 ptit\_prompt()

```
void ptit_prompt (
    char * nom )
```

Definition at line 240 of file [menu.c](#).

Here is the caller graph for this function:

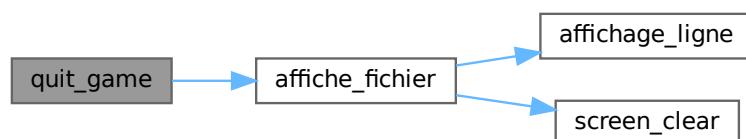


#### 4.7.1.7 `quit_game()`

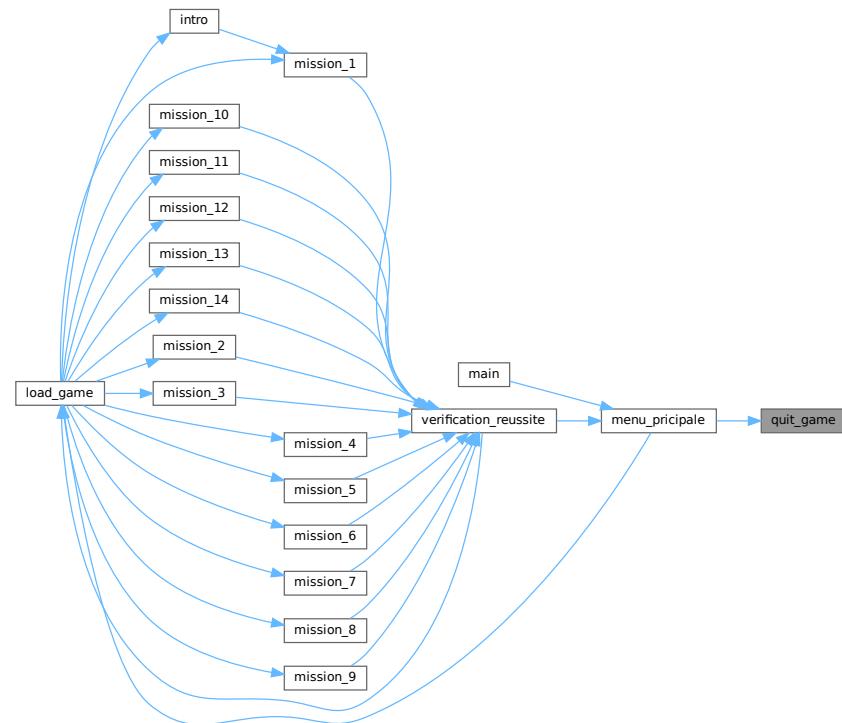
```
void quit_game (
    Mission * mission )
```

Definition at line 229 of file [menu.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



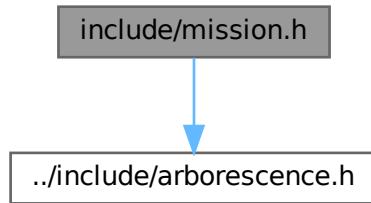
## 4.8 menu.h

[Go to the documentation of this file.](#)

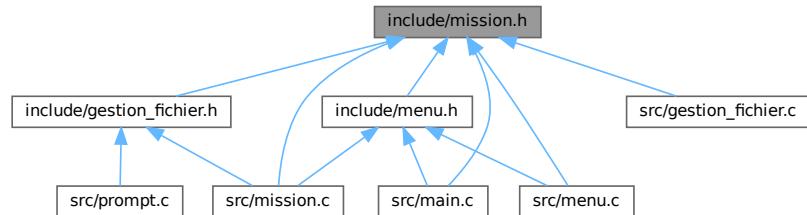
```
00001
00007 #ifndef __MENU__
00008 #define __MENU__
00009
00010 #include "../include/affichage.h" // Pour l'affichage
00011 #include "../include/securite.h" // Pour la sécurité
00012 #include "../include/mission.h" // Pour la gestion des missions
00013
00020 void menu_principale(Mission *mission , fichier *racine );
00028 void load_game(Mission *mission , int rang_du_dernier_mission , fichier *racine);
00034 void ptit_prompt(char *nom);
00040 int charger_partie(Mission *mission);
00046 void quit_game(Mission *mission);
00051 void logging();
00057 int dossierExiste(const char *chemin);
00058
00059 #endif
```

## 4.9 include/mission.h File Reference

```
#include "../include/arborescence.h"
Include dependency graph for mission.h:
```



This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [Mission](#)

### Macros

- #define [NOMBRE\\_MISSION](#) 14

### Typedefs

- typedef struct [Mission](#) [Mission](#)

## Functions

- void [intro](#) (Mission \*mission, fichier \*racine)
- void [réinitialisation\\_mission](#) (Mission \*mission)
- void [sauvegarde\\_dernier\\_partie](#) (Mission \*mission, int rang\_mission\_actuel)
- void [load\\_struct\\_mission](#) (Mission \*mission)
- void [mission\\_1](#) (Mission \*mission, fichier \*racine)
- void [mission\\_2](#) (Mission \*mission, fichier \*racine)
- void [mission\\_3](#) (Mission \*mission, fichier \*racine)
- void [mission\\_4](#) (Mission \*mission, fichier \*racine)
- void [mission\\_5](#) (Mission \*mission, fichier \*racine)
- void [mission\\_6](#) (Mission \*mission, fichier \*racine)
- void [mission\\_8](#) (Mission \*mission, fichier \*racine)
- void [mission\\_7](#) (Mission \*mission, fichier \*racine)
- void [mission\\_9](#) (Mission \*mission, fichier \*racine)
- void [mission\\_10](#) (Mission \*mission, fichier \*racine)
- void [mission\\_11](#) (Mission \*mission, fichier \*racine)
- void [mission\\_12](#) (Mission \*mission, fichier \*racine)
- void [mission\\_13](#) (Mission \*mission, fichier \*racine)
- void [verification\\_reussite](#) (Mission \*mission, fichier \*racine, bool reussite, int num\_mission)
- void [mission\\_14](#) (Mission \*mission, fichier \*racine)
- bool [verification\\_mission](#) (Mission \*mission, int rang\_mission, fichier \*racine, char \*courant)
- bool [prompt\\_mission](#) (fichier \*racine, char \*commande, char \*titre, char \*chemin, char \*rep\_courant, Mission \*miss\_actuelle, int rang\_mission, int autoverifi)
- void [map\\_initialisation](#) (Mission \*mission, fichier \*racine)

### 4.9.1 Macro Definition Documentation

#### 4.9.1.1 NOMBRE\_MISSION

```
#define NOMBRE_MISSION 14
```

Definition at line [20](#) of file [mission.h](#).

### 4.9.2 Typedef Documentation

#### 4.9.2.1 Mission

```
typedef struct Mission Mission
```

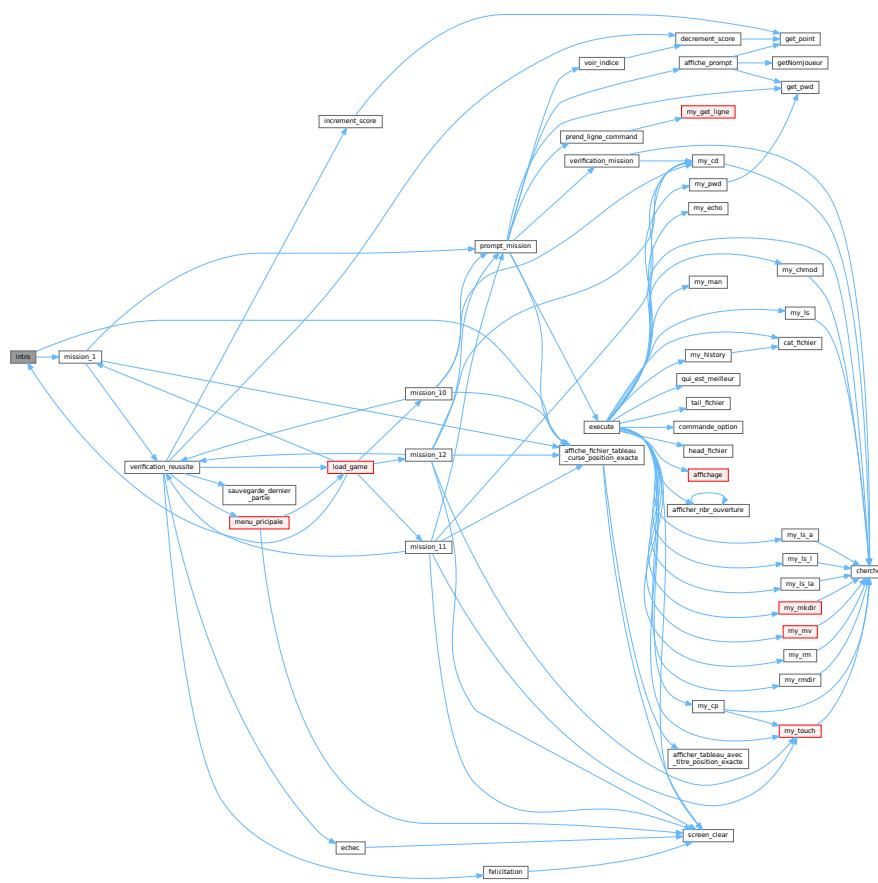
## 4.9.3 Function Documentation

### 4.9.3.1 intro()

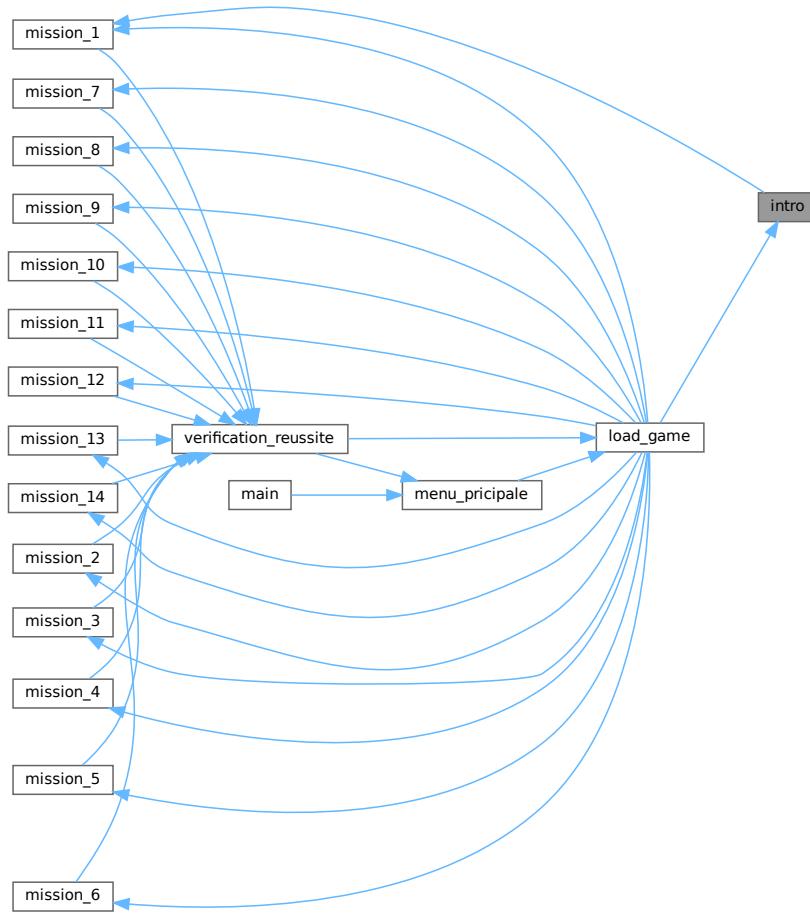
```
void intro (
    Mission * mission,
    fichier * racine )
```

Definition at line 289 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.9.3.2 load\_struct\_mission()

```
void load_struct_mission (
    Mission * mission )
```

Definition at line 80 of file [mission.c](#).

Here is the caller graph for this function:

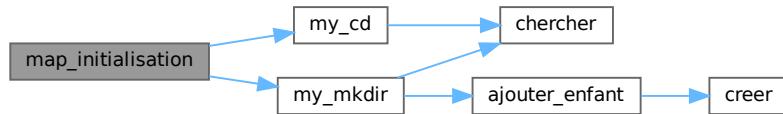


#### 4.9.3.3 map\_initialisation()

```
void map_initialisation (
    Mission * mission,
    fichier * racine )
```

Definition at line 32 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

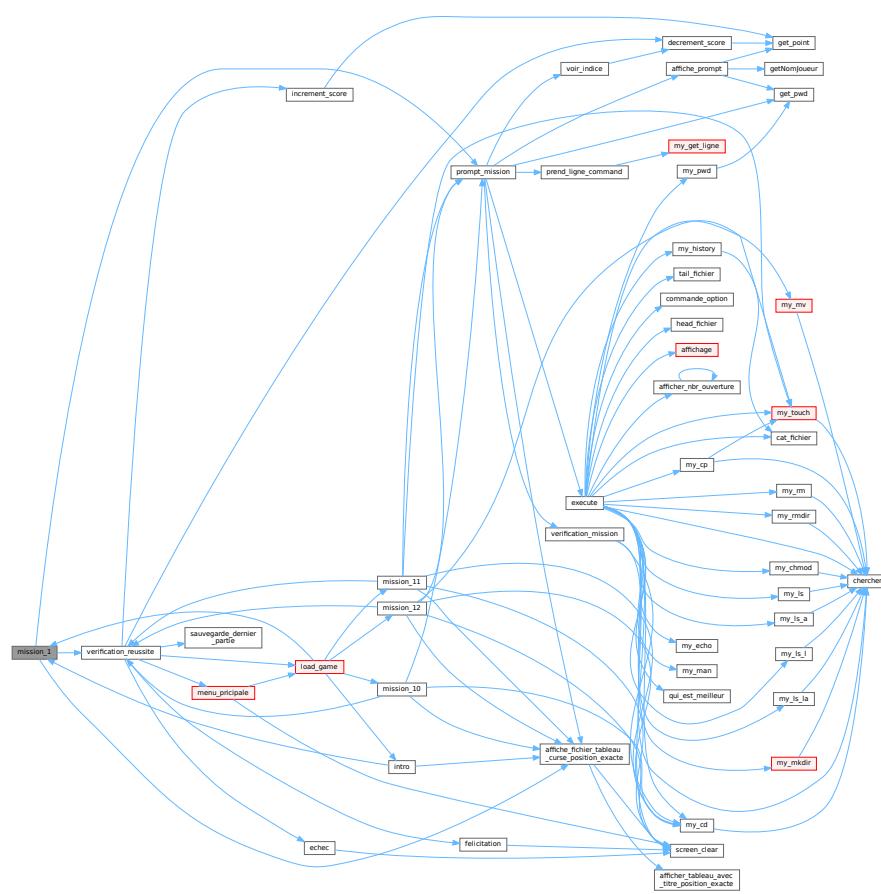


#### 4.9.3.4 mission\_1()

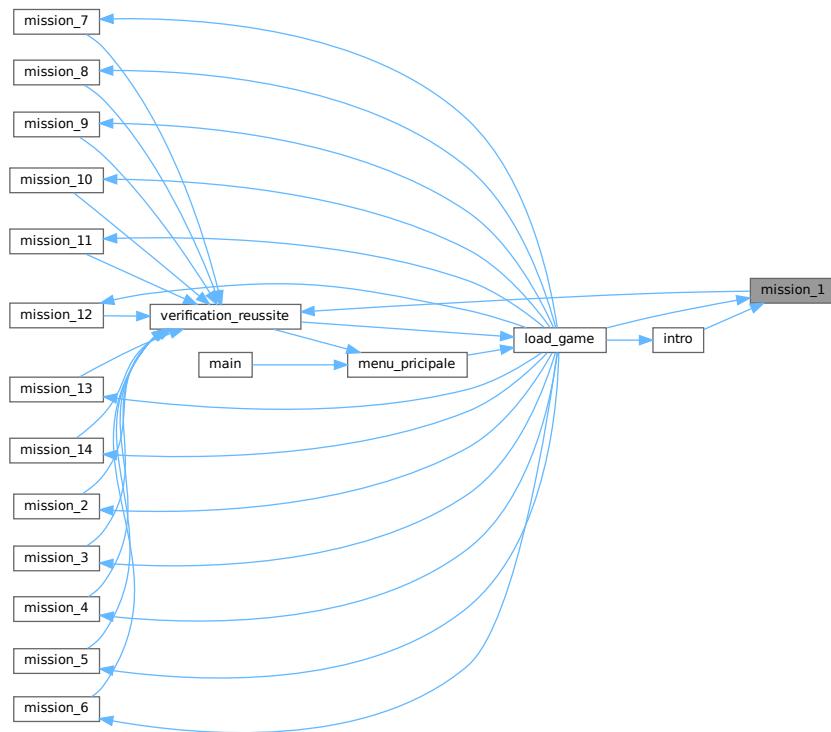
```
void mission_1 (
    Mission * mission,
    fichier * racine )
```

Definition at line 433 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

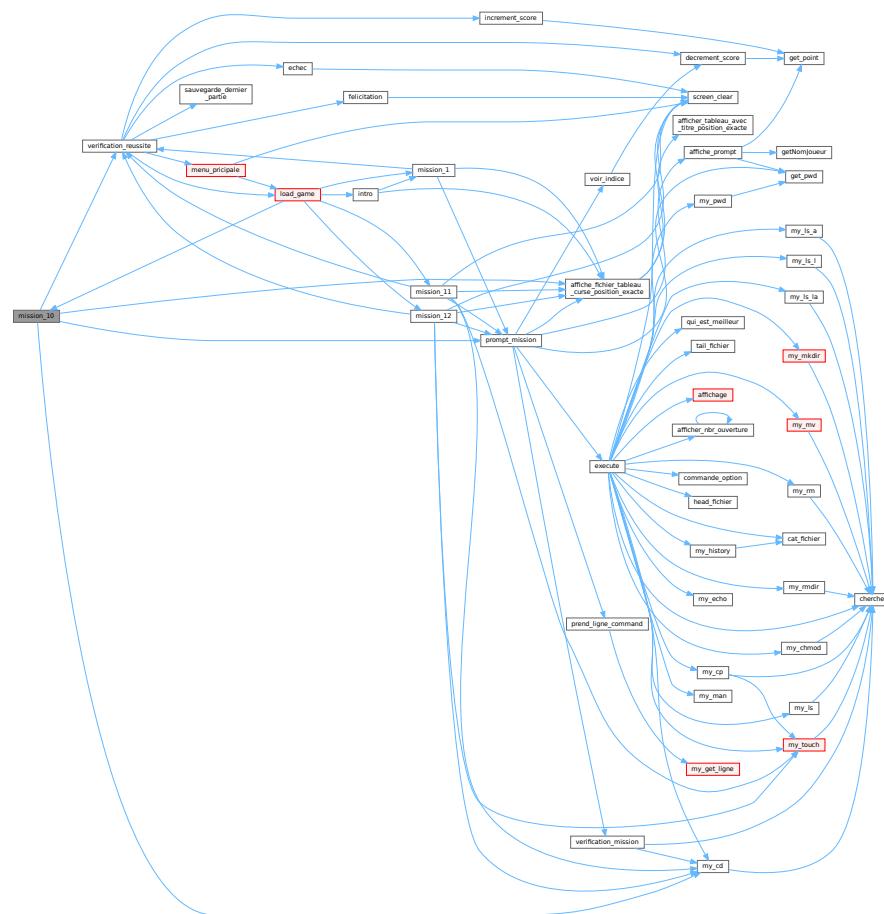


#### 4.9.3.5 mission\_10()

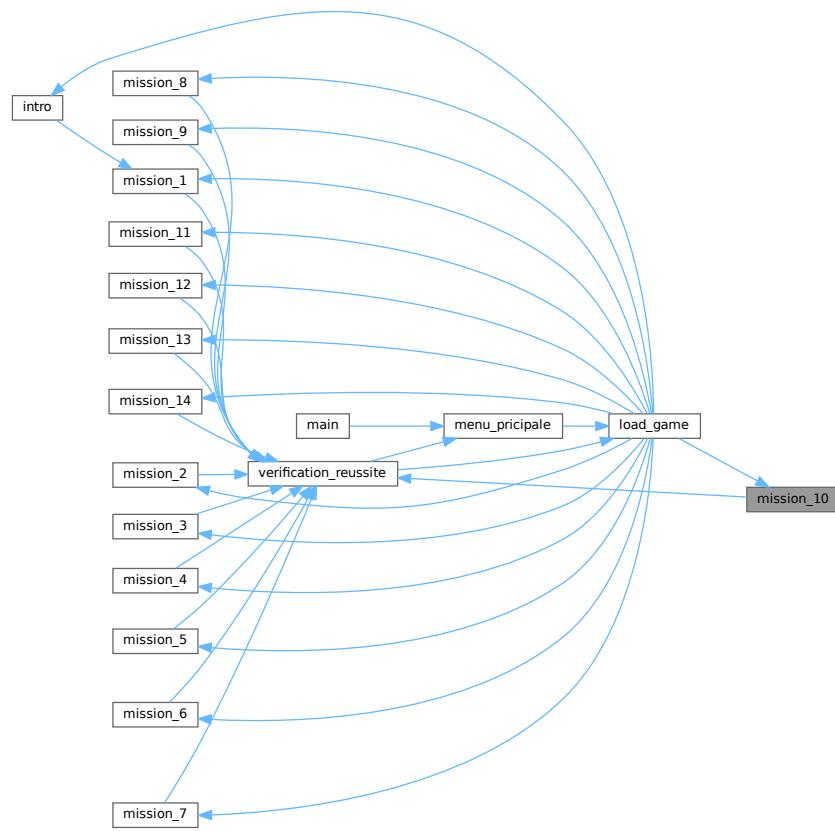
```
void mission_10 (
    Mission * mission,
    fichier * racine )
```

Definition at line [996](#) of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

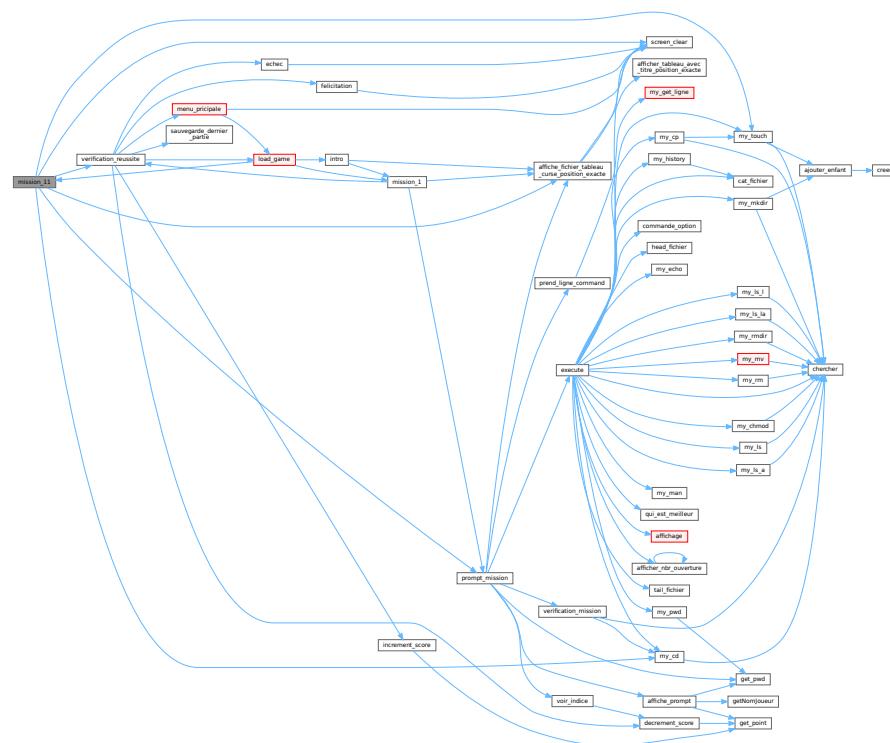


#### 4.9.3.6 mission\_11()

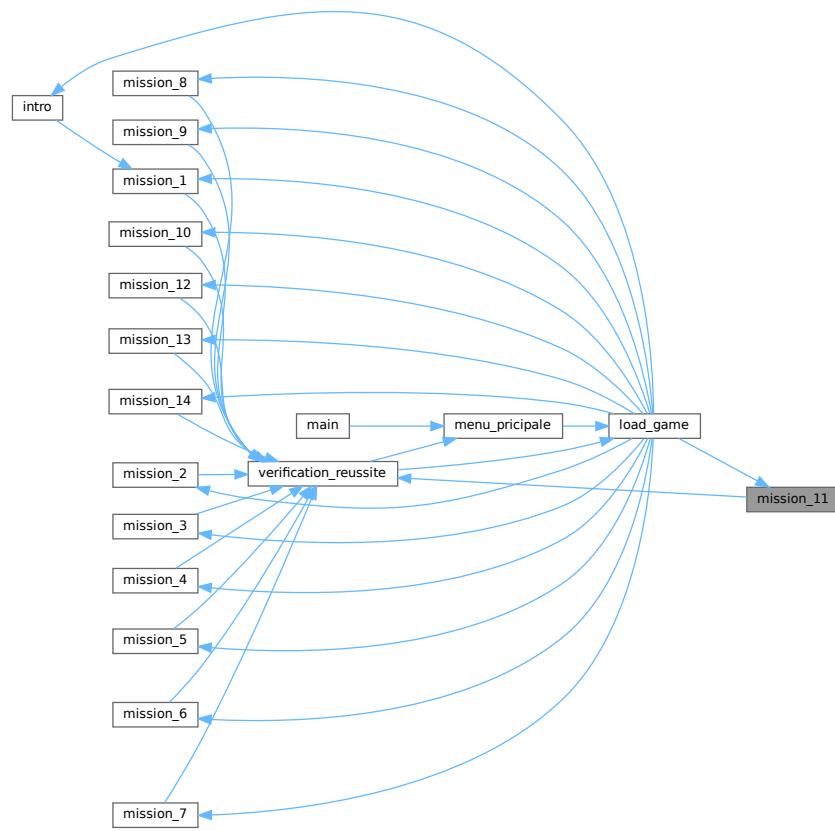
```
void mission_11 (
    Mission * mission,
    fichier * racine )
```

Definition at line 1033 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

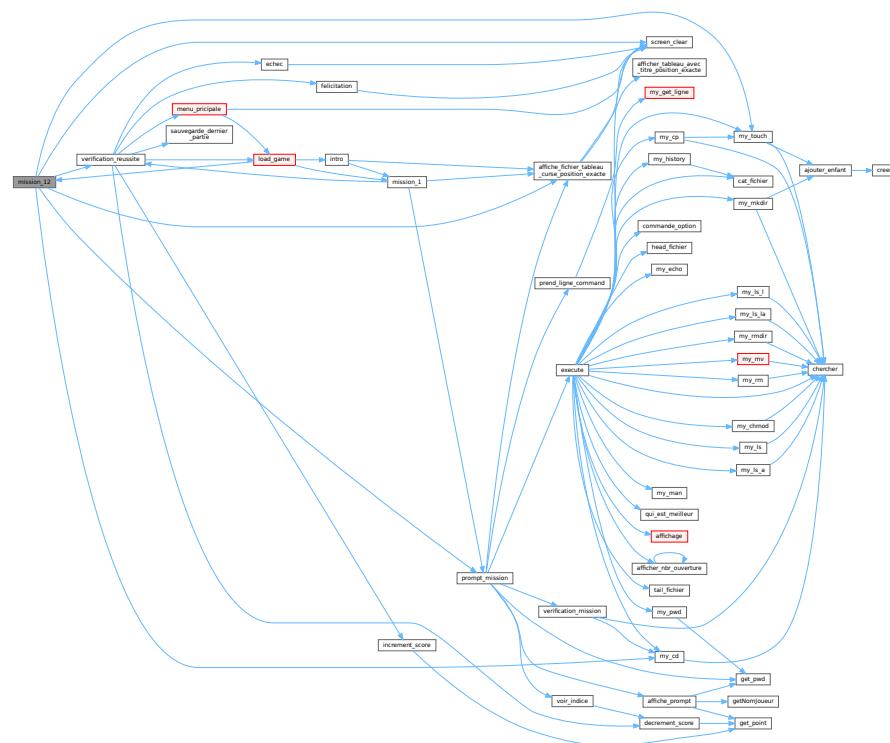


#### 4.9.3.7 mission\_12()

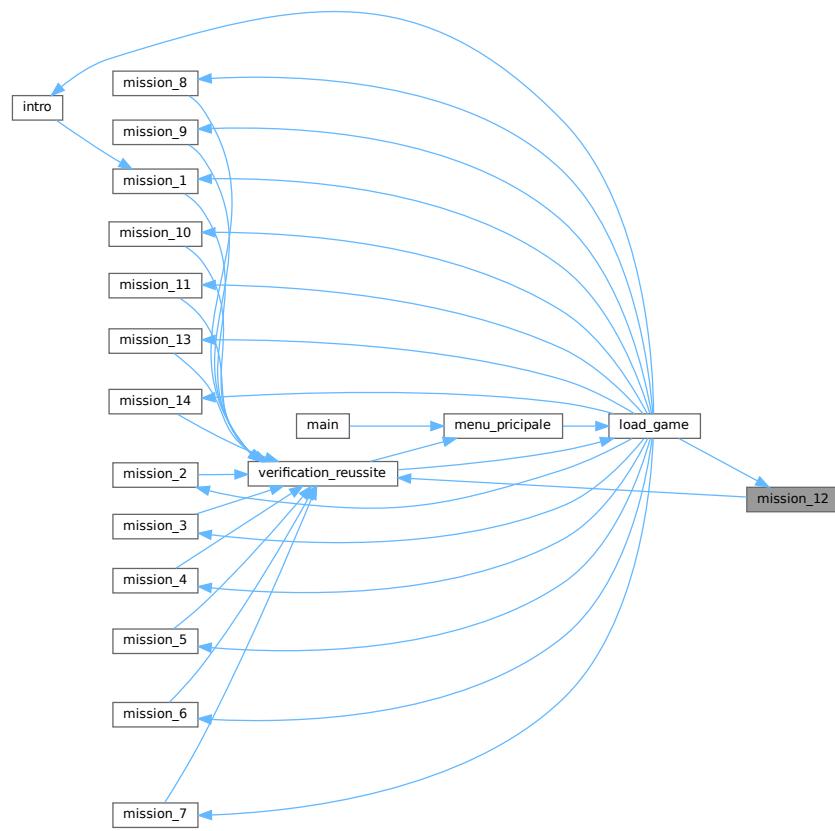
```
void mission_12 (
    Mission * mission,
    fichier * racine )
```

Definition at line 1076 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

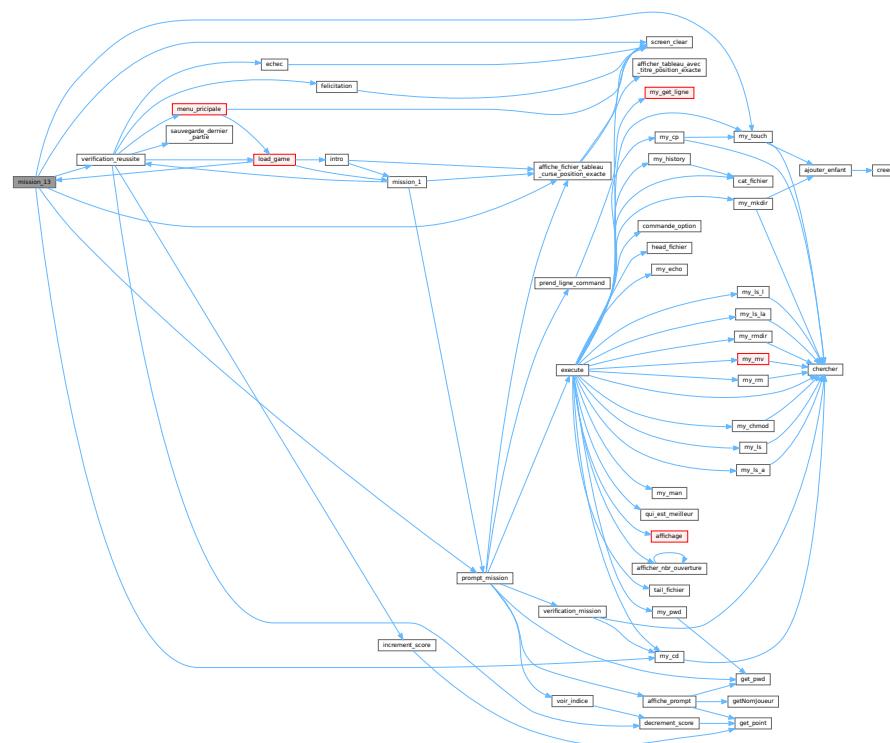


#### 4.9.3.8 mission\_13()

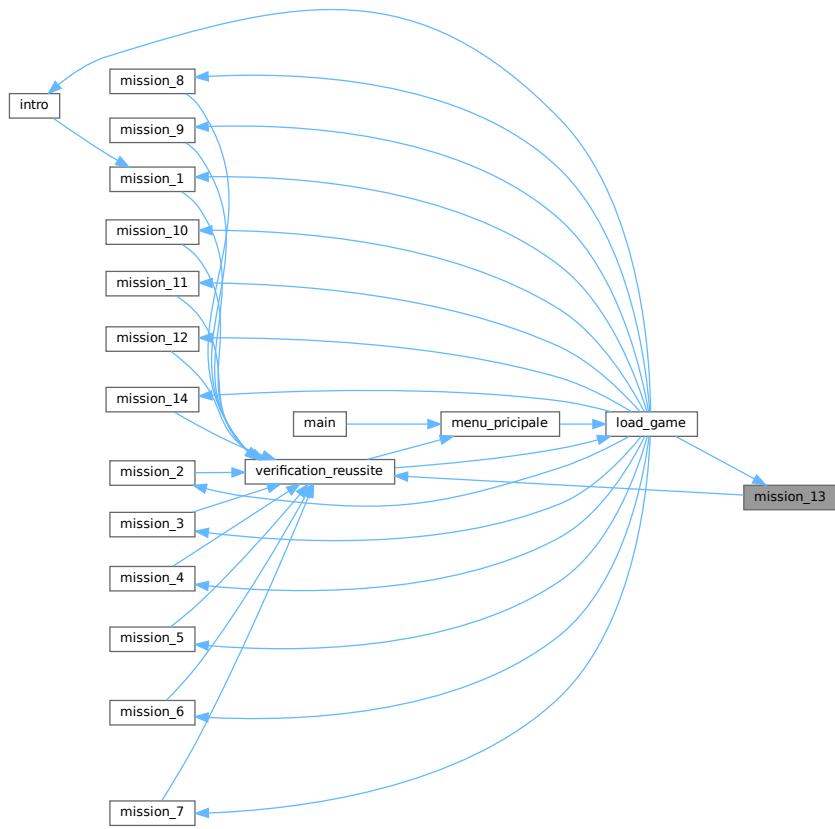
```
void mission_13 (
    Mission * mission,
    fichier * racine )
```

Definition at line 1119 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

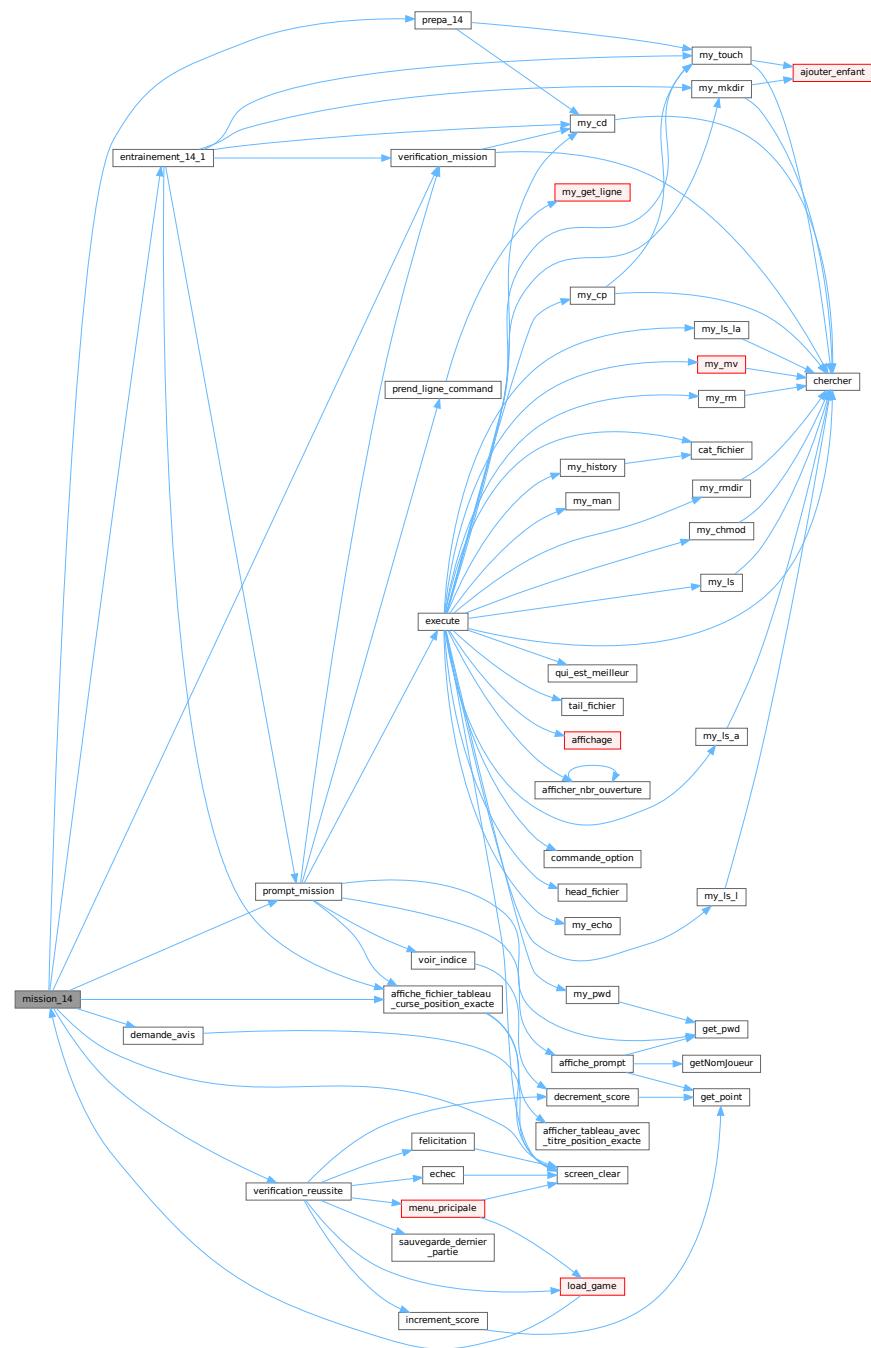


#### 4.9.3.9 mission\_14()

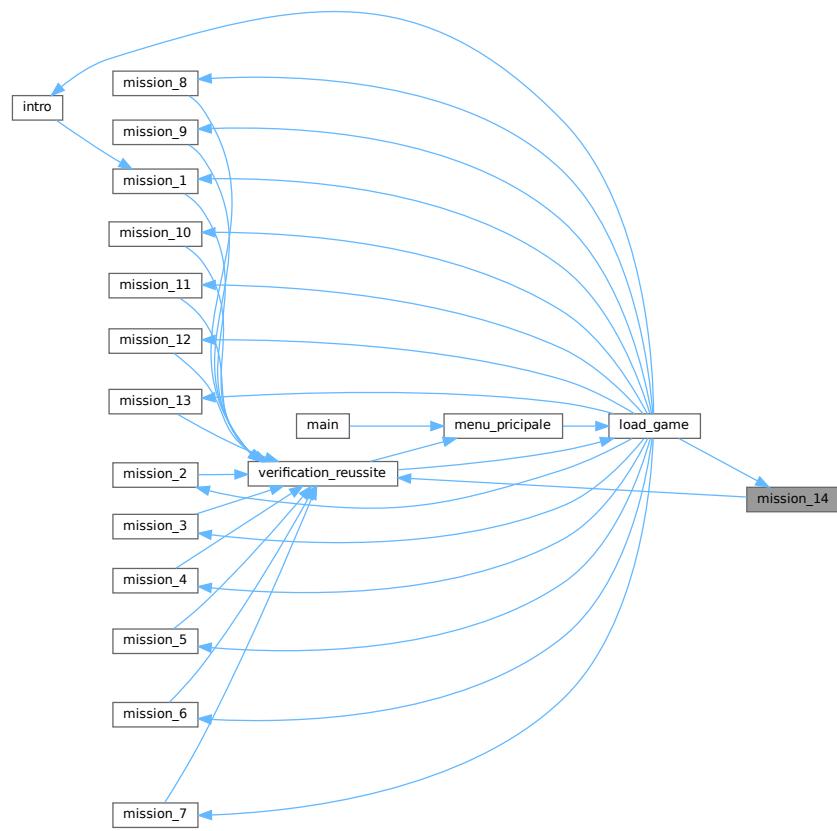
```
void mission_14 (
    Mission * mission,
    fichier * racine )
```

Definition at line 1252 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

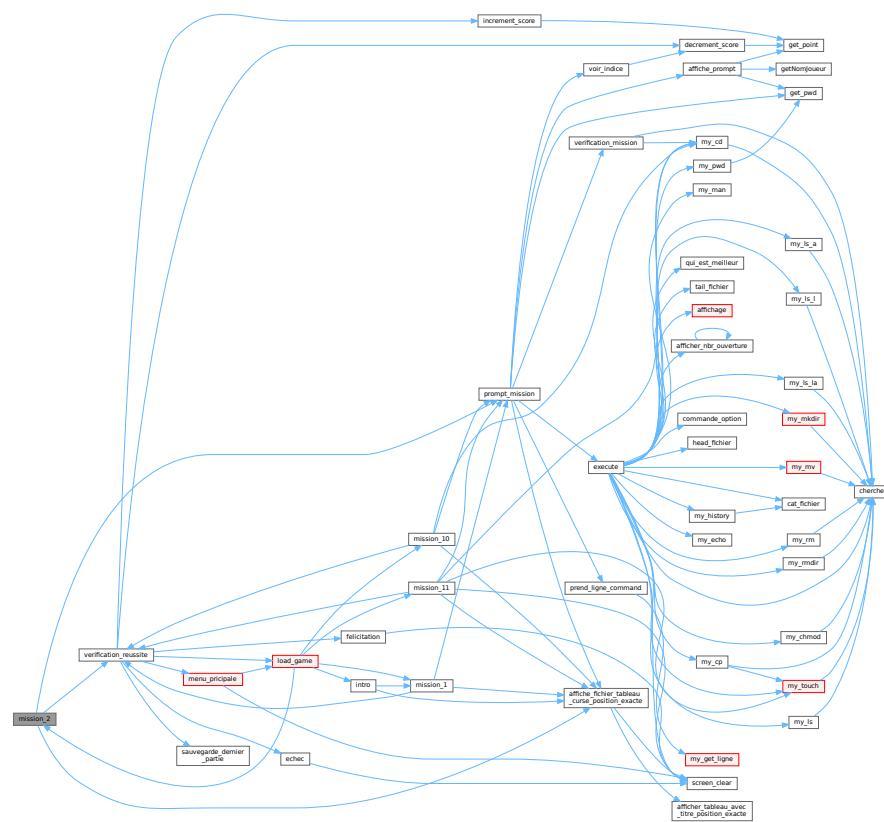


#### 4.9.3.10 mission\_2()

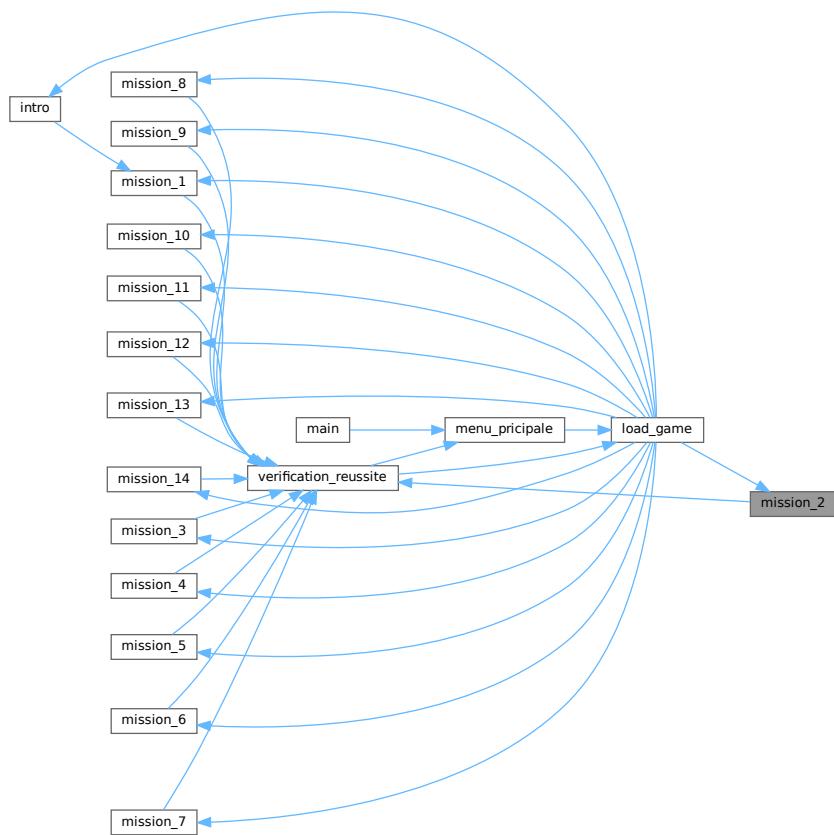
```
void mission_2 (
    Mission * mission,
    fichier * racine )
```

Definition at line [462](#) of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

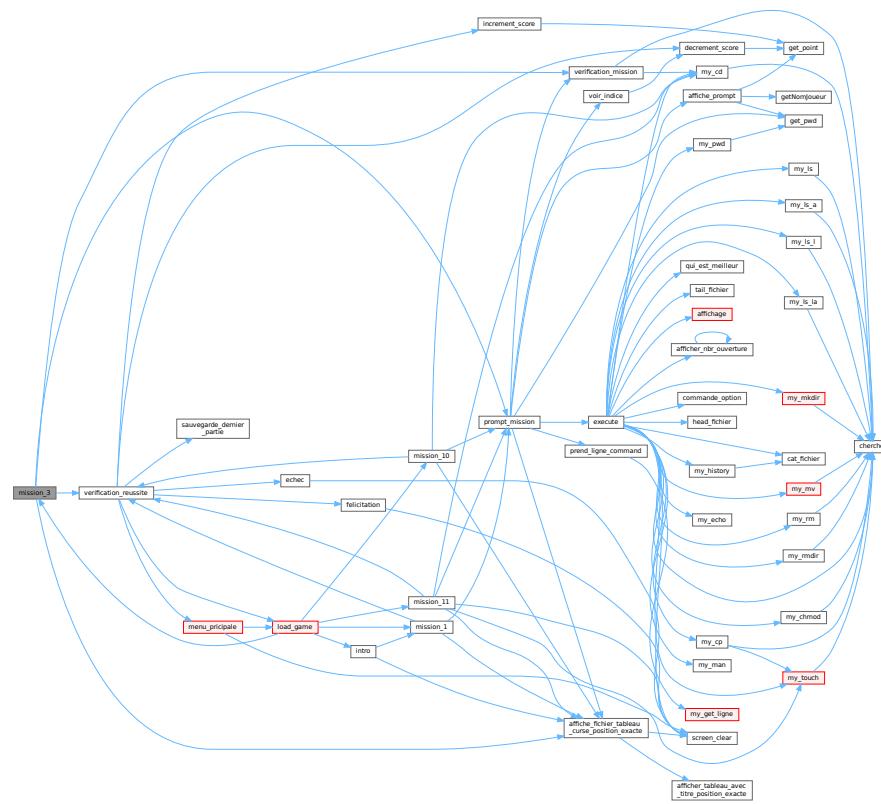


#### 4.9.3.11 mission\_3()

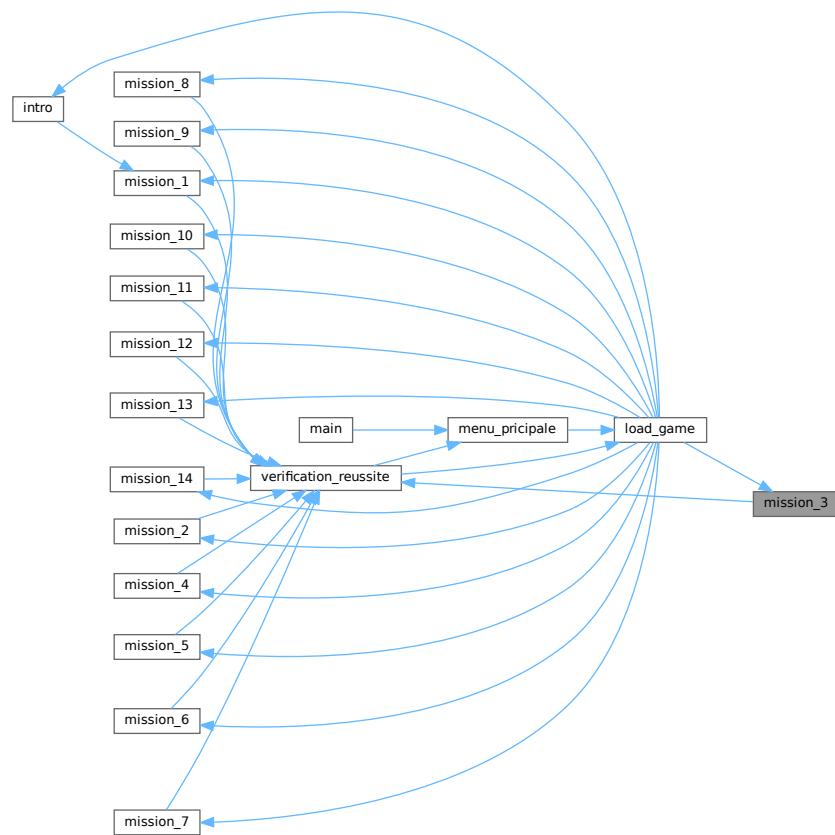
```
void mission_3 (
    Mission * mission,
    fichier * racine )
```

Definition at line 705 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

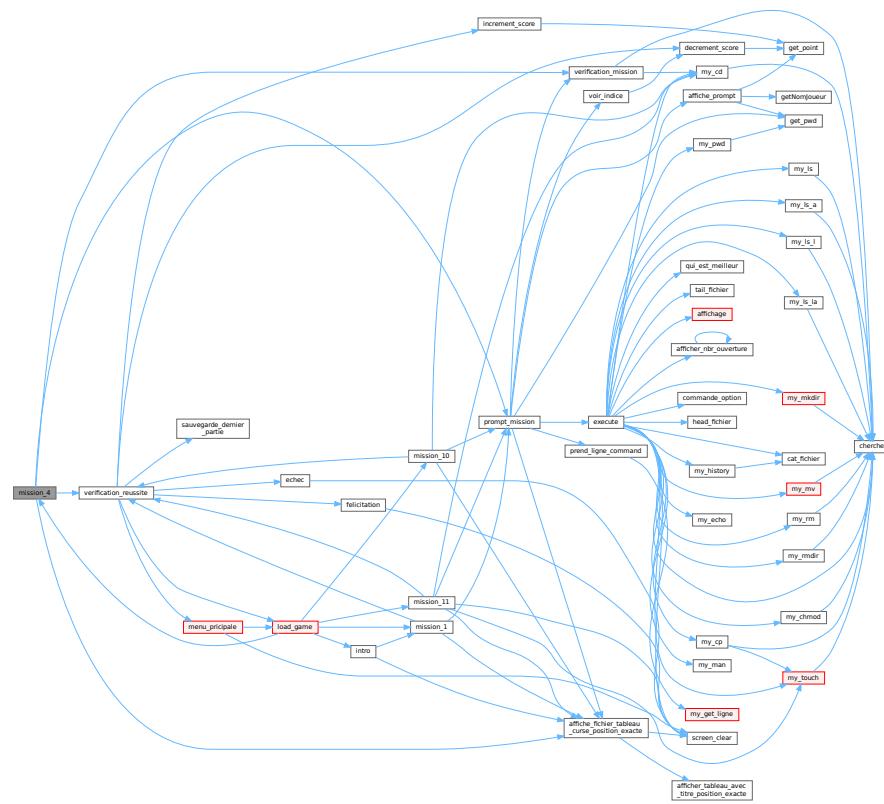


#### 4.9.3.12 mission\_4()

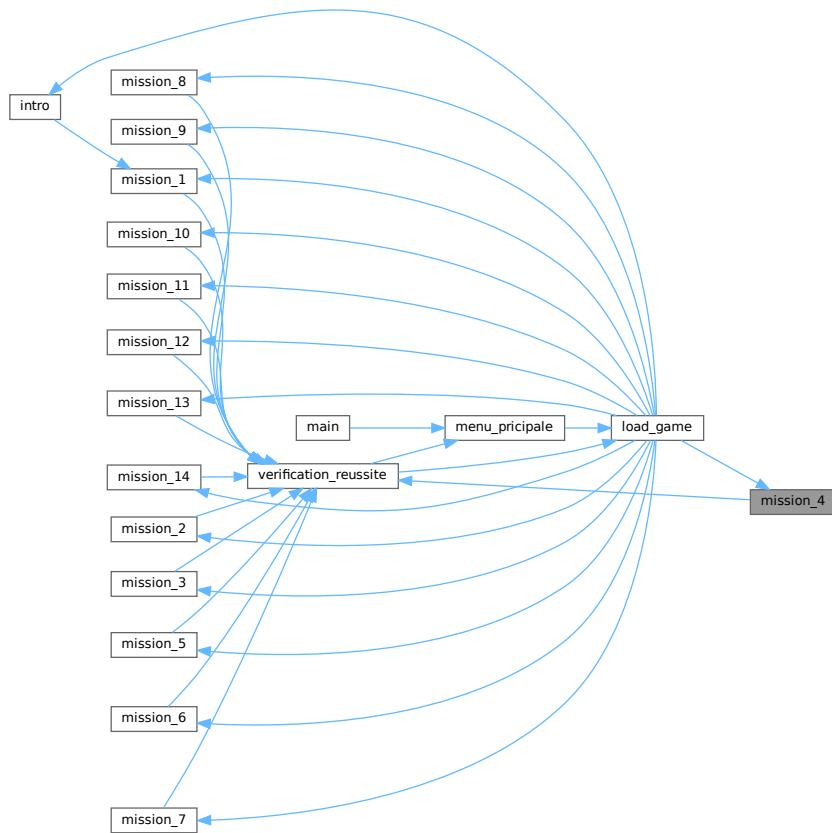
```
void mission_4 (
    Mission * mission,
    fichier * racine )
```

Definition at line 730 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

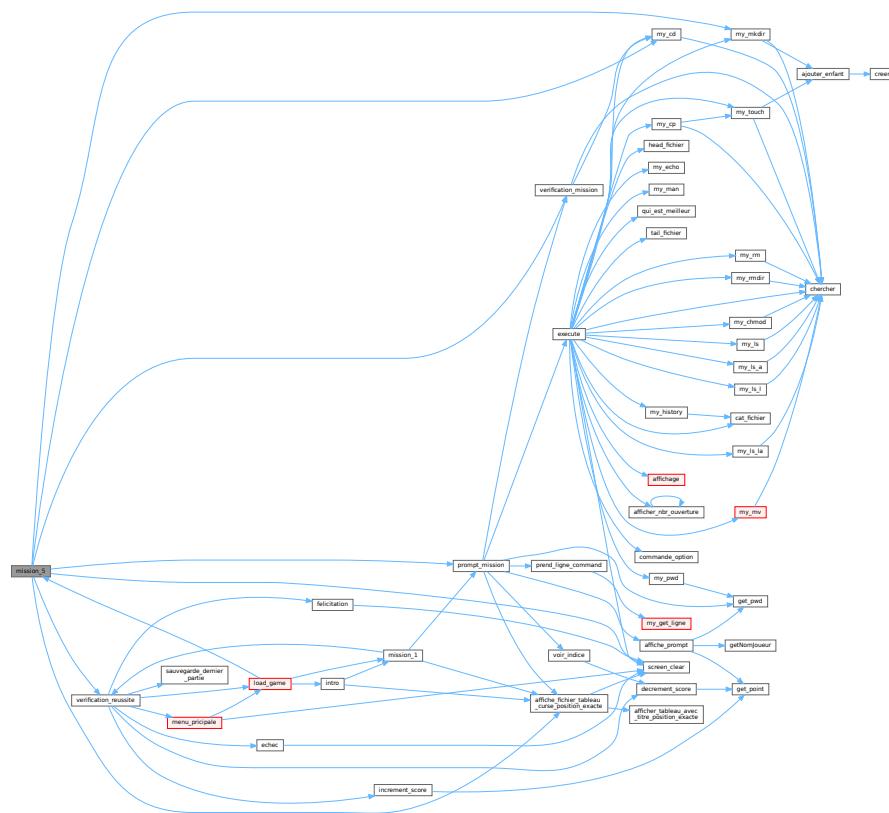


#### 4.9.3.13 mission\_5()

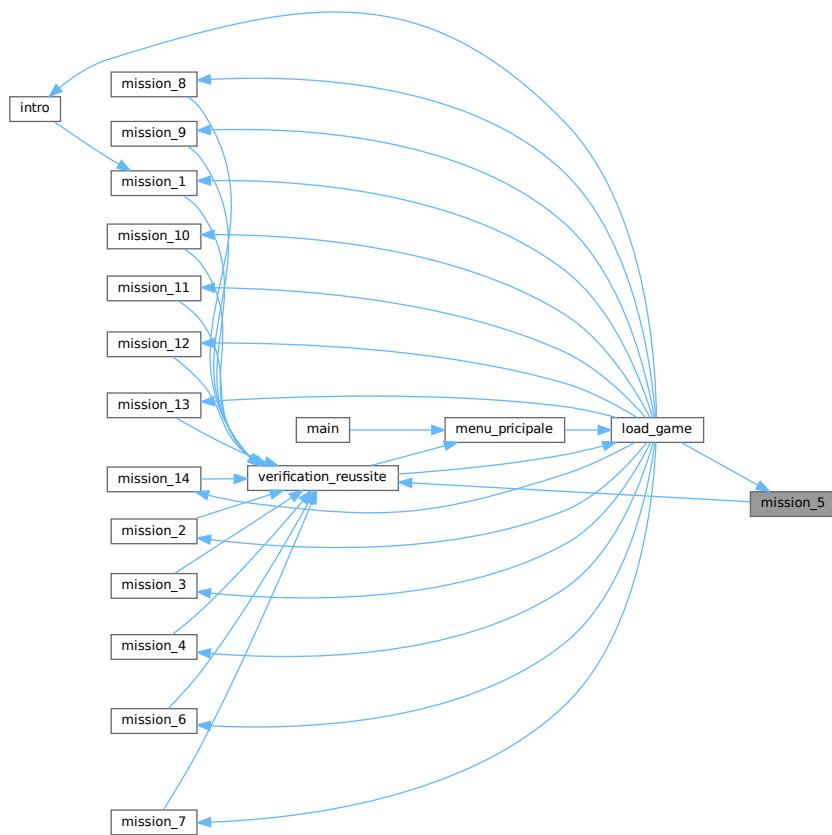
```
void mission_5 (
    Mission * mission,
    fichier * racine )
```

Definition at line 760 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

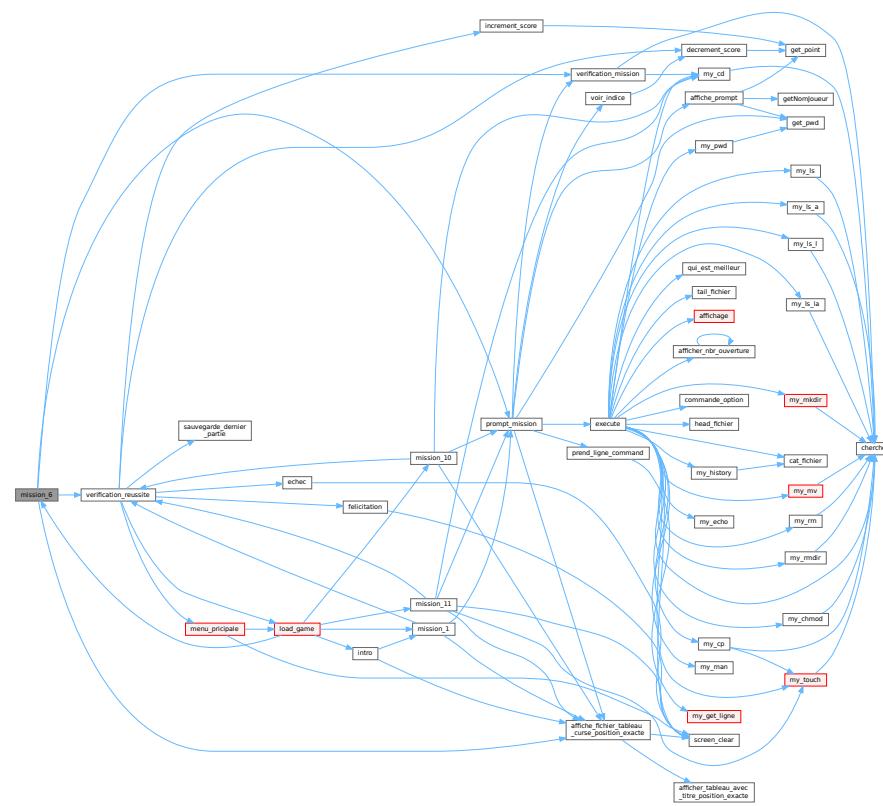


#### 4.9.3.14 mission\_6()

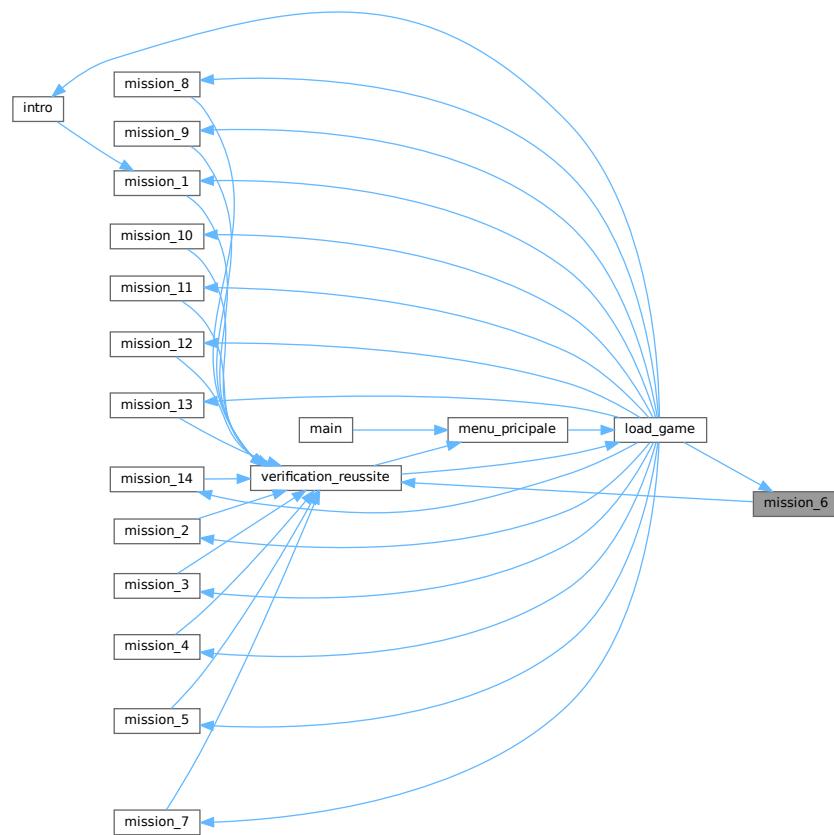
```
void mission_6 (
    Mission * mission,
    fichier * racine )
```

Definition at line 867 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

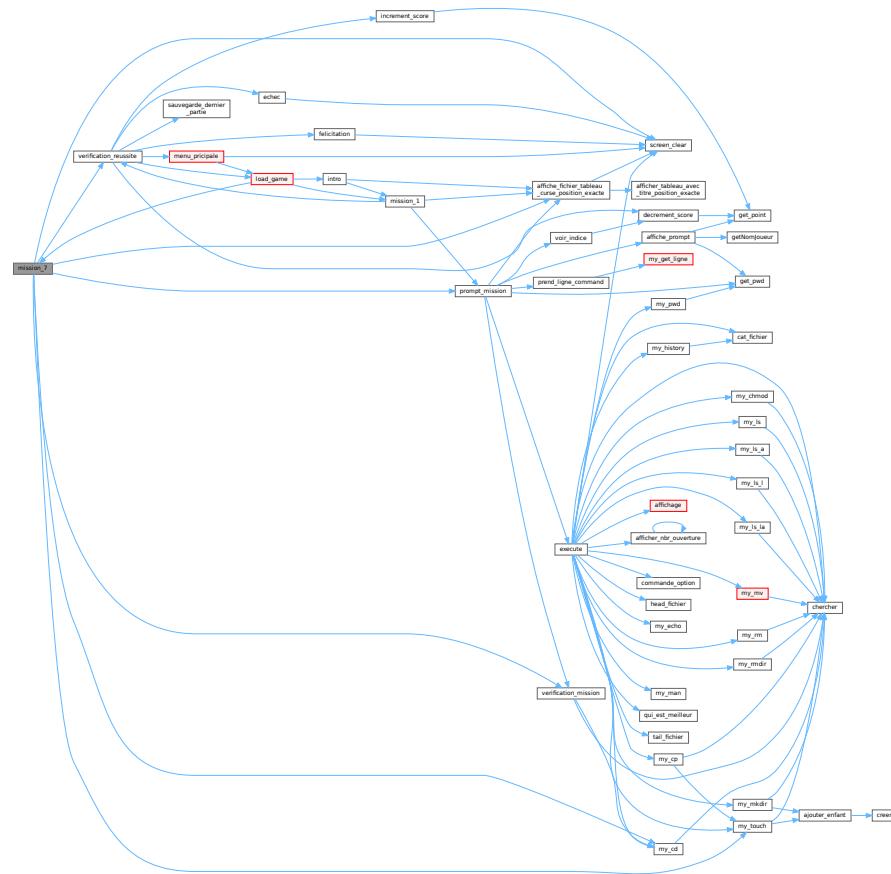


#### 4.9.3.15 mission\_7()

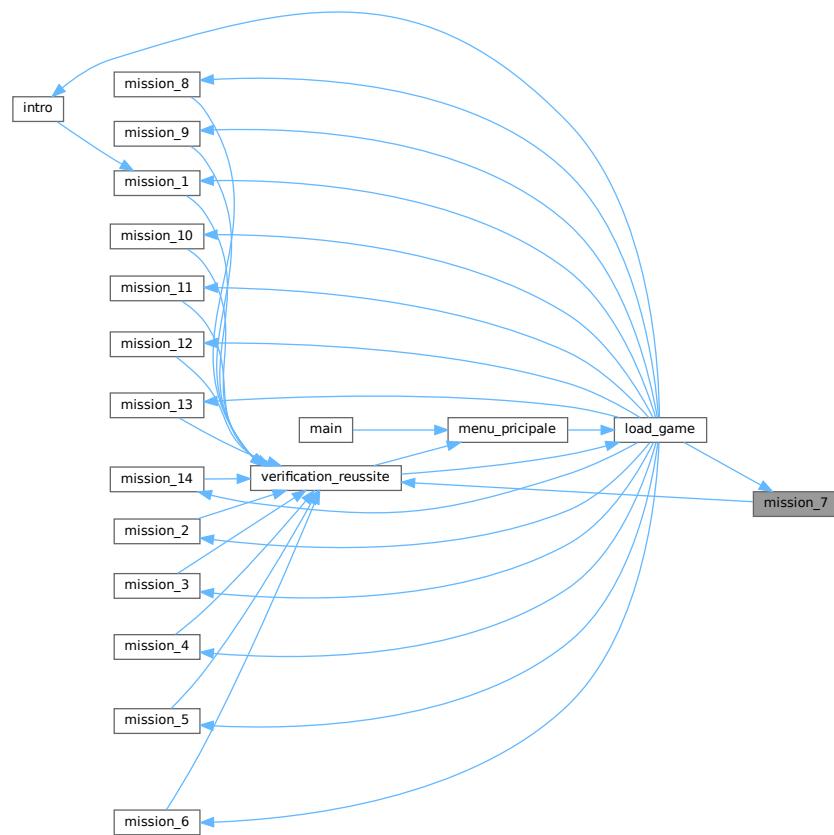
```
void mission_7 (
    Mission * mission,
    fichier * racine )
```

Definition at line 891 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

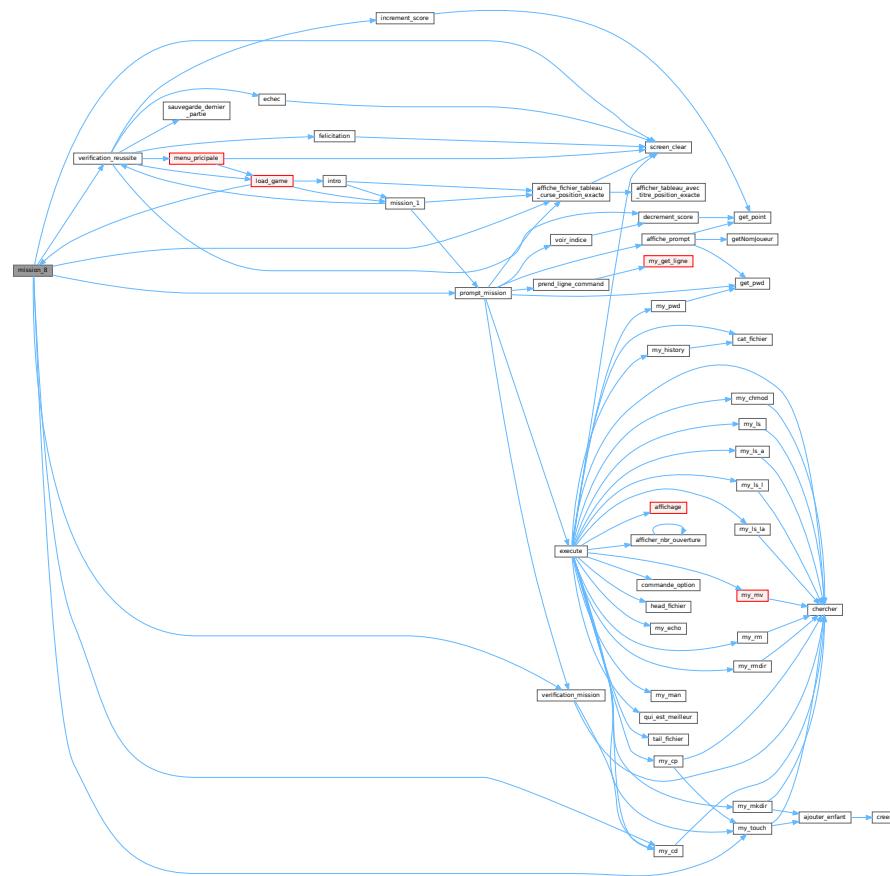


#### 4.9.3.16 mission\_8()

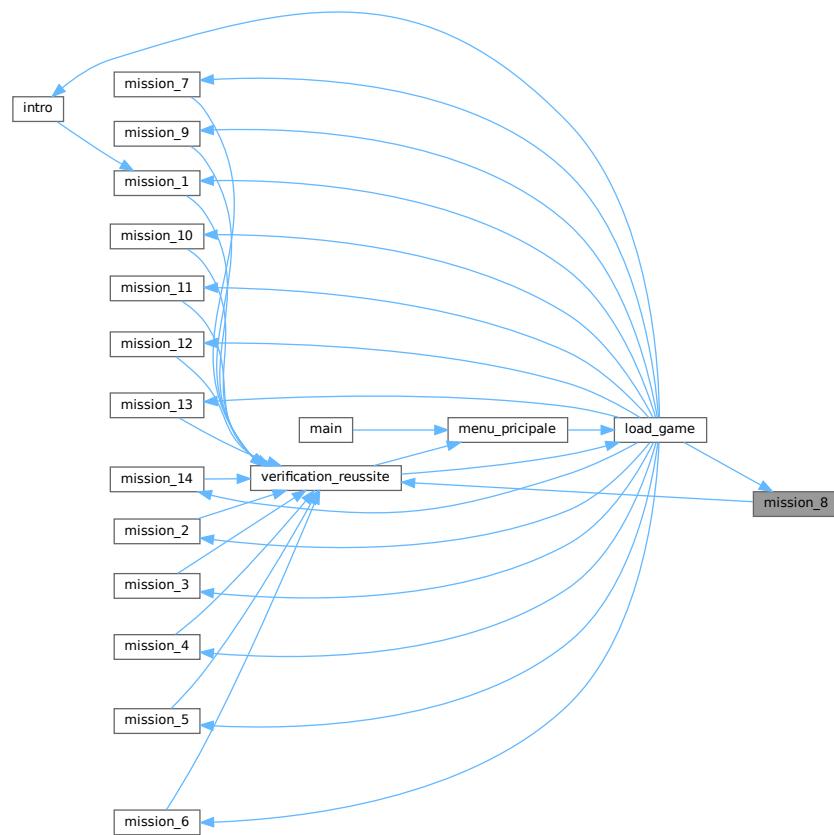
```
void mission_8 (
    Mission * mission,
    fichier * racine )
```

Definition at line 926 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

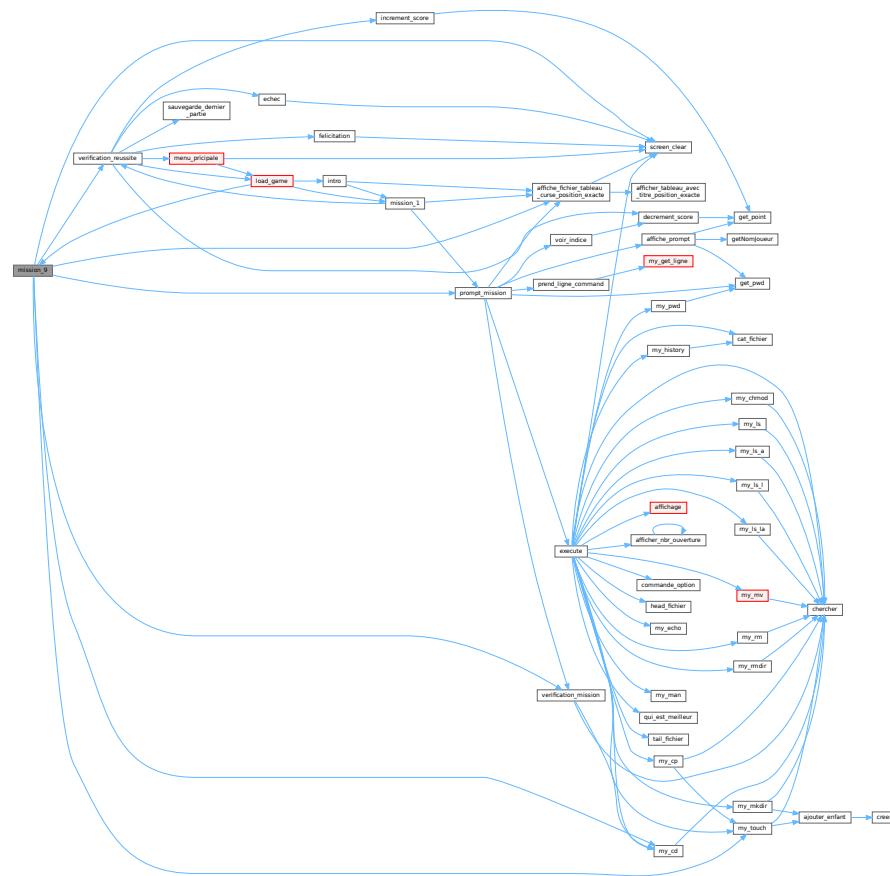


#### 4.9.3.17 `mission_9()`

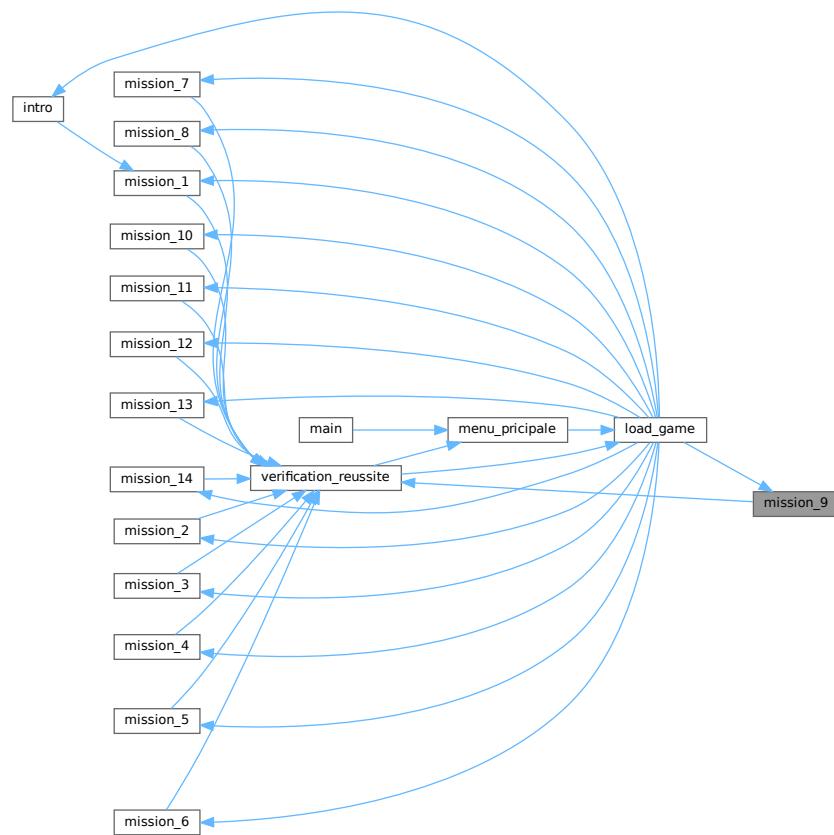
```
void mission_9 (
    Mission * mission,
    fichier * racine )
```

Definition at line 963 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

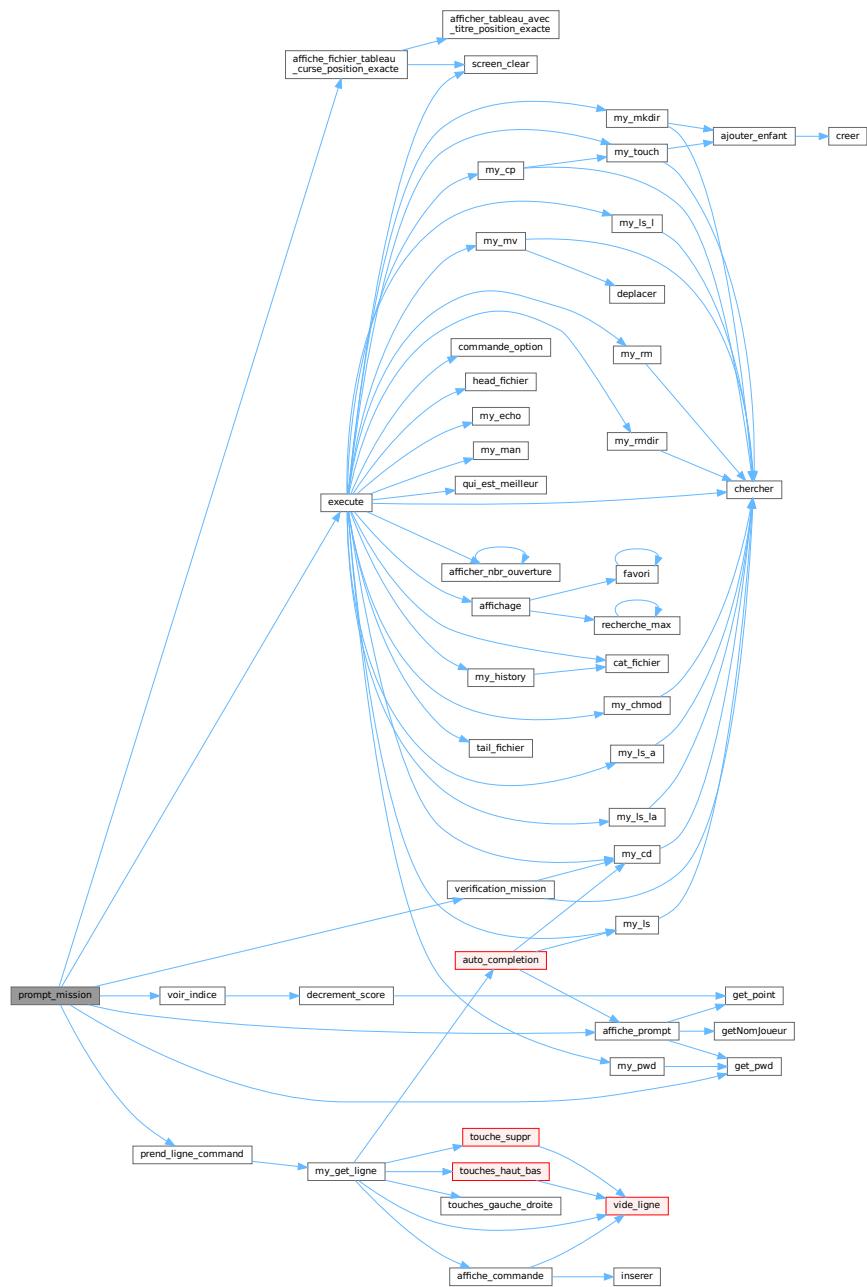


#### 4.9.3.18 `prompt_mission()`

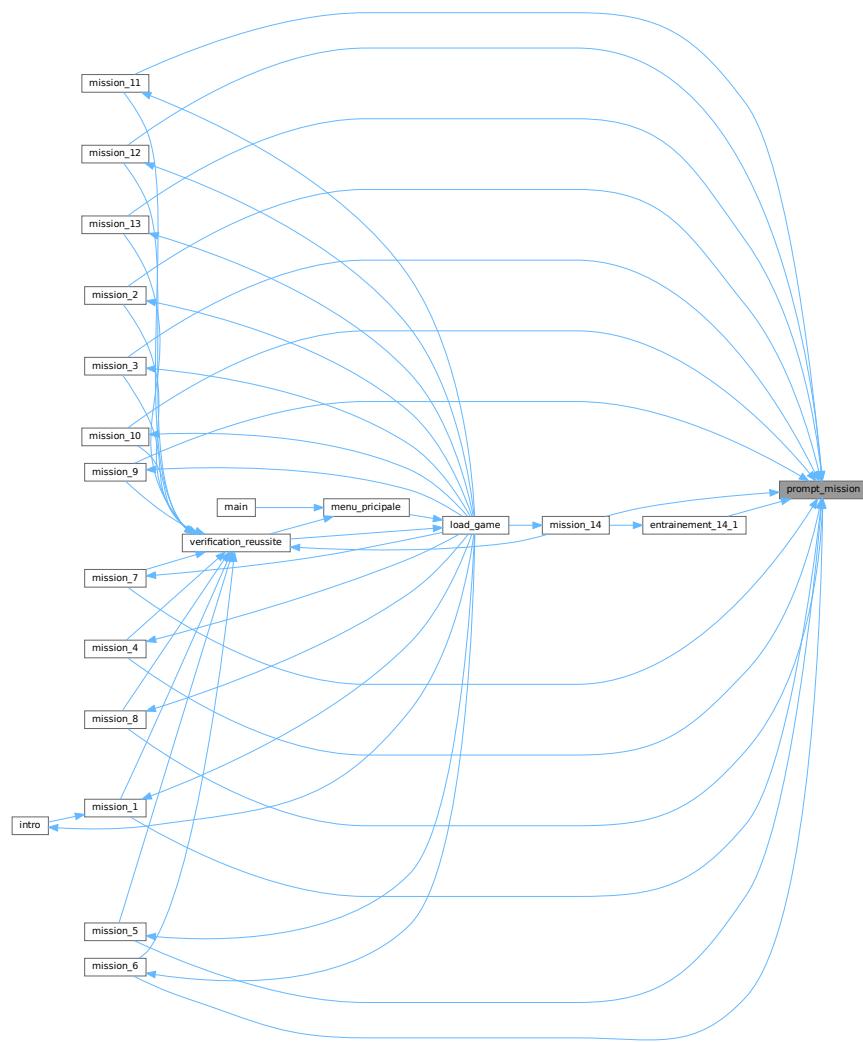
```
bool prompt_mission (
    fichier * racine,
    char * commande,
    char * titre,
    char * chemin,
    char * rep_courant,
    Mission * miss_actuelle,
    int rang_mission,
    int autoverifi )
```

Definition at line 309 of file [mission.c](#).

Here is the call graph for this function:



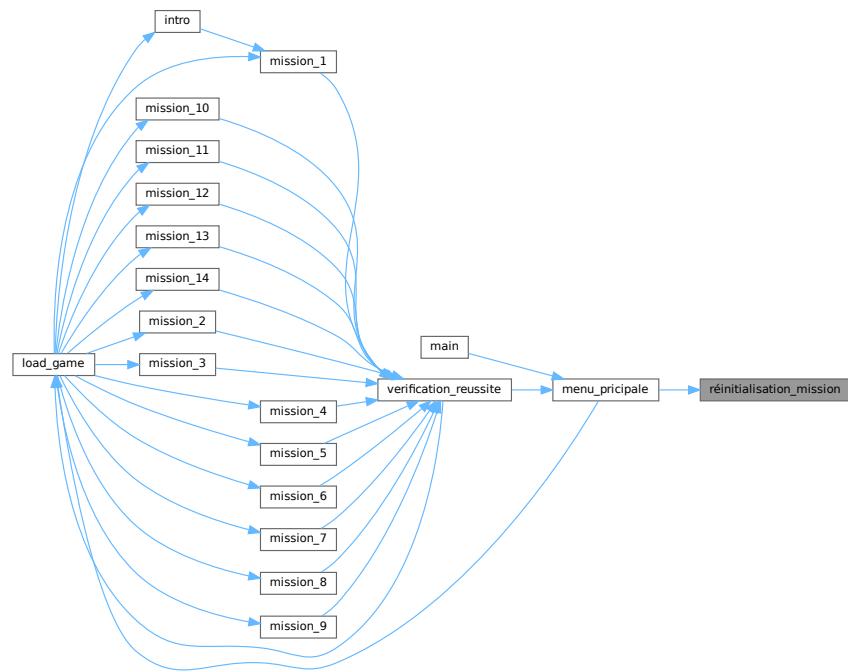
Here is the caller graph for this function:



#### 4.9.3.19 `réinitialisation_mission()`

```
void réinitialisation_mission (
    Mission * mission )
```

Here is the caller graph for this function:

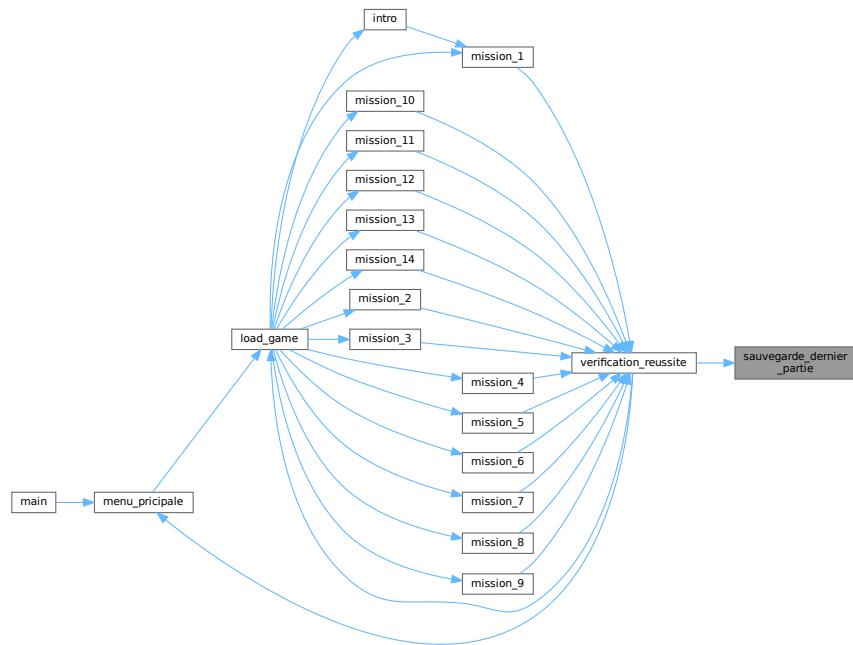


#### 4.9.3.20 `sauvegarde_dernier_partie()`

```
void sauvegarde_dernier_partie (
    Mission * mission,
    int rang_mission_actuel )
```

Definition at line 111 of file [mission.c](#).

Here is the caller graph for this function:



#### 4.9.3.21 verification\_mission()

```

bool verification_mission (
    Mission * mission,
    int rang_mission,
    fichier * racine,
    char * courant )

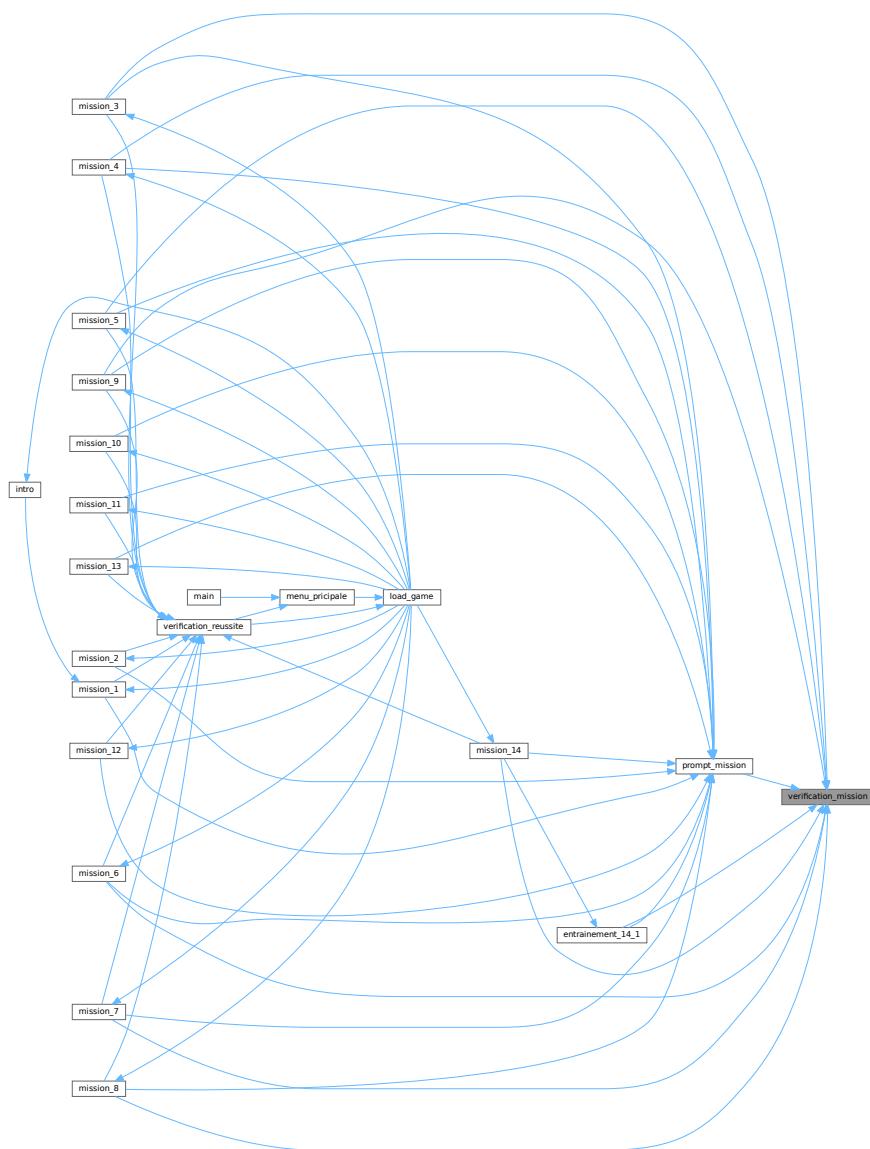
```

Definition at line 548 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

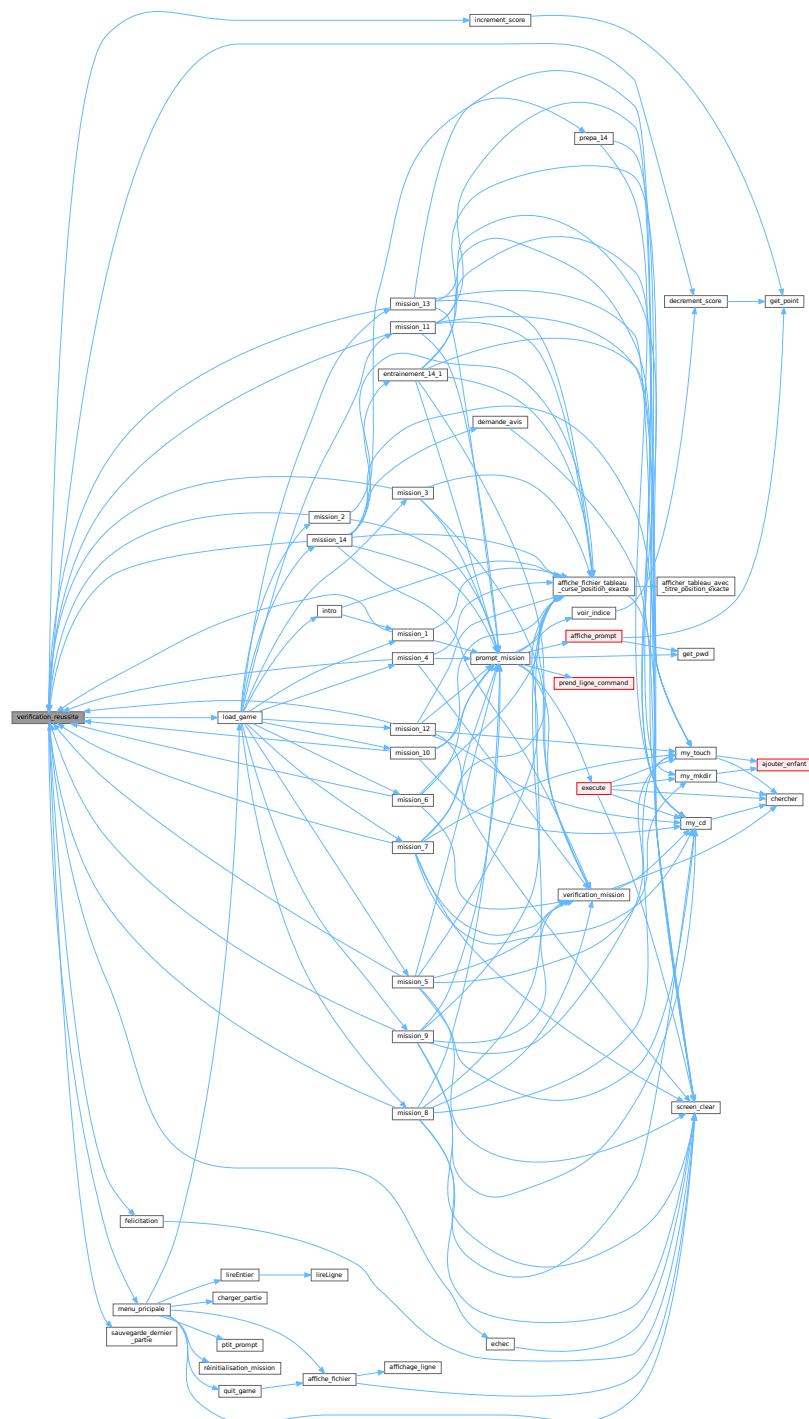


#### 4.9.3.22 `verification_reussite()`

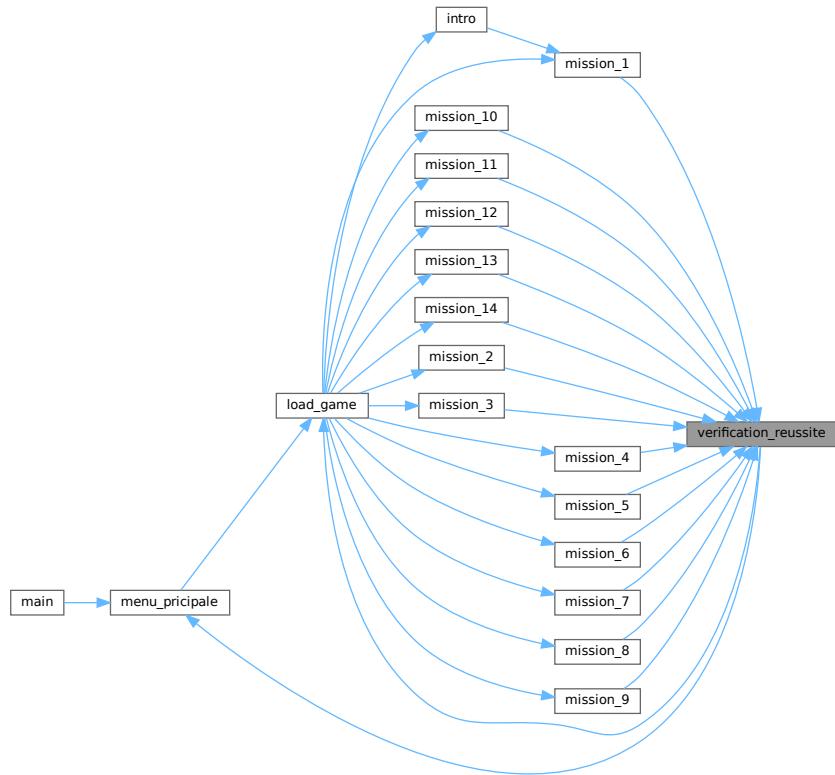
```
void verification_reussite (
    Mission * mission,
    fichier * racine,
    bool reussite,
    int num_mission )
```

Definition at line 246 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.10 mission.h

[Go to the documentation of this file.](#)

```

00001
00007 #ifndef __MISSION__
00008 #define __MISSION__
00009
00010     typedef struct Mission // Structure pour gérer les missions
00011     {
00012         int débloqué ; // 1 si la mission est débloqué , 0 sinon
00013         int terminé ; // 1 si la mission est terminée , 0 sinon
00014         char titre[50] ; // Titre de la mission
00015         int point ; // Points attribués pour la mission
00016         char indice[100]; // Indices pour la mission
00017     }Mission;
00018
00019
00020 #define NOMBRE_MISSION 14 // Nombre de missions dans le jeu
00021 #include "../include/arborescence.h" // Pour la struct fichier
00022
00029 void intro(Mission *mission , fichier *racine);
00035 void réinitialisation_mission( Mission *mission );
00042 void sauvegarde_dernier_partie( Mission *mission , int rang_mission_actuel );
00048 void load_struct_mission(Mission *mission );
00055 void mission_1( Mission *mission , fichier *racine );
00062 void mission_2( Mission *mission , fichier *racine );
00069 void mission_3( Mission *mission , fichier *racine );
00076 void mission_4(Mission *mission , fichier *racine );
00083 void mission_5(Mission *mission , fichier *racine );
00090 void mission_6(Mission *mission , fichier *racine ) ; // cree un fichier
00096 void mission_8(Mission *mission , fichier *racine ) ; // Deplacer avec cp
00103 void mission_7(Mission *mission , fichier *racine ) ;
00110 void mission_9(Mission *mission , fichier *racine ) ;
00117 void mission_10(Mission *mission , fichier *racine ) ;
00124 void mission_11(Mission *mission , fichier *racine ) ;
  
```

```

00131     void mission_12(Mission *mission , fichier *racine ) ;
00138     void mission_13(Mission *mission , fichier *racine ) ;
00139
00148     void verification_reussite(Mission *mission , fichier *racine , bool reuissite , int num_mission
00149 );
00156     void mission_14(Mission *mission , fichier *racine) ; // Chmod
00157
00166     bool verification_mission(Mission *mission , int rang_mission , fichier *racine , char* courant);
00179     bool prompt_mission(fichier *racine , char *commande , char *titre , char *chemin , char
00180     *rep_courant , Mission *miss_actuelle , int rang_mission , int autoverifi);
00186     void map_initialisation(Mission *mission , fichier *racine);
00187
00188
00189 #endif

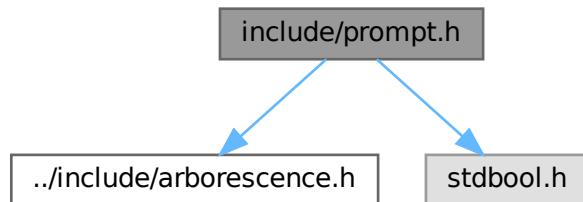
```

## 4.11 include/prompt.h File Reference

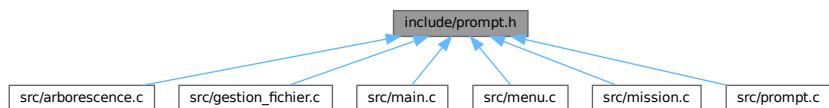
regroupement des fonctions dans [prompt.c](#)

```
#include "../include/arborescence.h"
#include <stdbool.h>
```

Include dependency graph for prompt.h:



This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [ligne](#)

### Macros

- #define [SORTIR](#) "exit"  
*c'est la commande pour sortir du petit terminal*
- #define [LIGNE](#) 1000
- #define [MOT](#) 100  
*c'est la taille max defini pour les mots .*

## Typedefs

- `typedef struct ligne ligne_commande`

## Functions

- `void affiche_prompt (fichier *repertoire_actuel)`  
*il affiche le ligne de prompt où l'on insere du commande*
- `int prend_ligne_command (char **emplacement, int taille, fichier *repertoire_actuel)`
- `int getNomJoueur (char *emplacement)`  
*trouver le nom du joueur actuelle et puis l'insérer dans emplacement*
- `fichier * execute (char **commande, fichier *repertoire_actue, int nbMot)`  
*execution du commande entrée en argument*
- `void get_pwd (fichier *adresse, char *emplacement)`  
*trouver le chemin du repertoire actuelle et puis l'insérer dans emplacement*
- `int get_point ()`  
*c'est pour savoir le point que le joueur actuel possède*
- `void increment_score (int actuPTS)`  
*c'est pour augmenter le nombre de score du joueur actuel*
- `int decrement_score (int penalite)`  
*c'est une fonction qui fait decremener le nombre de score d'une joueur en jeu*
- `int my_get_ligne (char *mobile, int taille_Max, fichier *repertoire_actuel)`
- `int rapport_mission (int num_mission, bool reussie)`  
*celui ci écrit le joueur , temps de jeu , reussie ou pas et numero de mission dans le fichier d'enregistrement*
- `void vide_ligne (int droite, char *chaine)`  
*c'est une simple fonction qui permet de vider la ligne de commande actuelle*
- `void efface_cara (char *chaine, int *position)`  
*On lui donne une chaine de caractere et une position ; Il supprime le caractere qui se trouve à cette position.*
- `int taille_de (char *ligne)`  
*elle ne compte pas les caracteres comme strlen ; elle donne la taille en affichage*
- `int prend_mot_dans (int position, char *chaine, char *emplacement)`
- `int auto_completion (char *chaine, fichier *repertoire_actuel, int *nb_Tab, int *pos_droite)`  
*c'est la fonction principale de l'auto-completion sur ligne de commande*
- `void touche_suppr (int *position, int *pos_droite, char *chaine)`  
*c'est la fonction associer à la touche suppr*
- `void touches_gauche_droite (int *indice, int *pos_droite, char *chaine, int gauche)`  
*c'est la fonction associer à la touche gauche et droite*
- `void touches_haut_bas (int *indice, int *position, int *droite, char *chaine, char *tmp, int haut)`  
*la fonctionne associer à la touche haut et bas*
- `void inserer_mot_dans (int position, char *chaine, char *insertion)`  
*c'est pour inserer une mot dans une position precise dans une chaine ; separateur = espace*
- `int trouver_avant (char *chaine, char *recherche)`  
*Elle permet de trouver si la chaine recherche est le debut de chaine ; les espaces sont ignorés.*
- `void affiche_commande (int *indice, char caractere, char *chaine, int droite)`  
*Elle permet d'inserer une caractere dans une chaine et d'affcher la chaine.*
- `int compter_mot (char *chaine)`  
*fonction qui compte le nombre de mot dans une chaine de caractere*
- `int commande_option (ligne_commande *ligne, char **liste, int nb_mot)`  
*cette fonction represente une ligne de commande dans une structure ligne\_commande*
- `void inserer (char caractere, int position, char *chaine)`  
*Elle permet d'inserer une caractere dans une chaine à une position preciser en argument.*

### 4.11.1 Detailed Description

regroupement des fonctions dans [prompt.c](#)

Author

Valisoa

Definition in file [prompt.h](#).

### 4.11.2 Macro Definition Documentation

#### 4.11.2.1 LIGNE

```
#define LIGNE 1000
```

Definition at line [29](#) of file [prompt.h](#).

#### 4.11.2.2 MOT

```
#define MOT 100
```

c'est la taille max defini pour les mots .

Definition at line [34](#) of file [prompt.h](#).

#### 4.11.2.3 SORTIR

```
#define SORTIR "exit"
```

c'est la commande pour sortir du petit terminale

Definition at line [13](#) of file [prompt.h](#).

### 4.11.3 Typedef Documentation

#### 4.11.3.1 ligne\_commande

```
typedef struct ligne ligne_commande
```

Definition at line [46](#) of file [prompt.h](#).

### 4.11.4 Function Documentation

#### 4.11.4.1 affiche\_commande()

```
void affiche_commande (
    int * indice,
    char caractere,
    char * chaine,
    int droite )
```

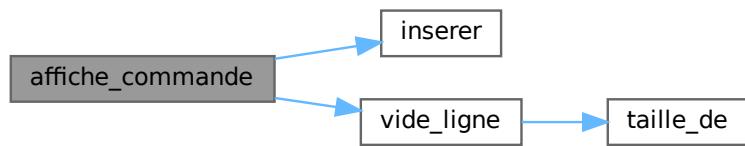
Elle permet d'insérer un caractère dans une chaîne et d'afficher la chaîne.

### Parameters

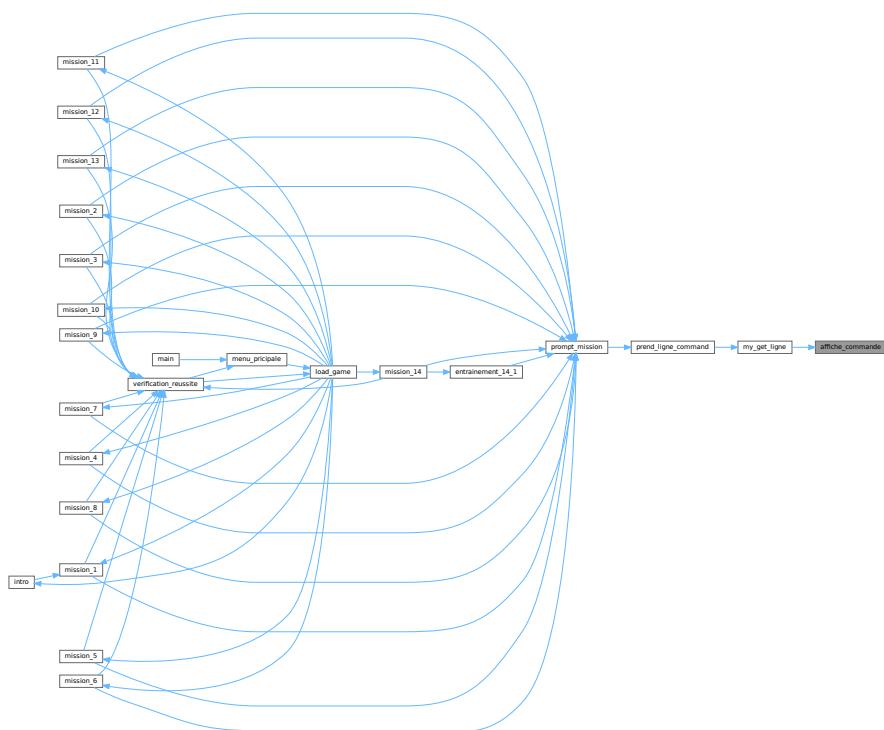
<i>indice</i>	la position d'insertion dans la chaine
<i>caractere</i>	le caractere à insérer dans la chaine
<i>chaine</i>	la chaine où l'on va insérer une caractere
<i>droite</i>	la position actuelle en se référant de la position la plus à droite

Definition at line 213 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.4.2 affiche\_prompt()

```
void affiche_prompt (
    fichier * repertoire_actuel )
```

il affiche le ligne de prompt où l'on insere du commande

il affiche le ligne de prompt où l'on insere du commande

<ncurses.h>

"../include/gestion\_fichier.h"

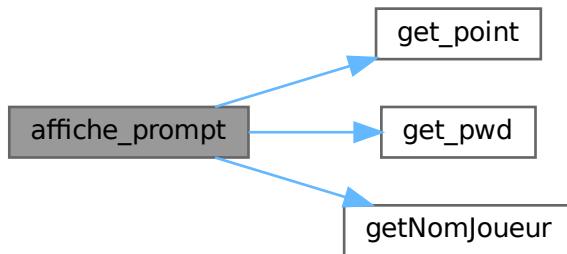
pour la gestion des fichiers ( adaptation des mes besoins )

"../include/prompt.h"

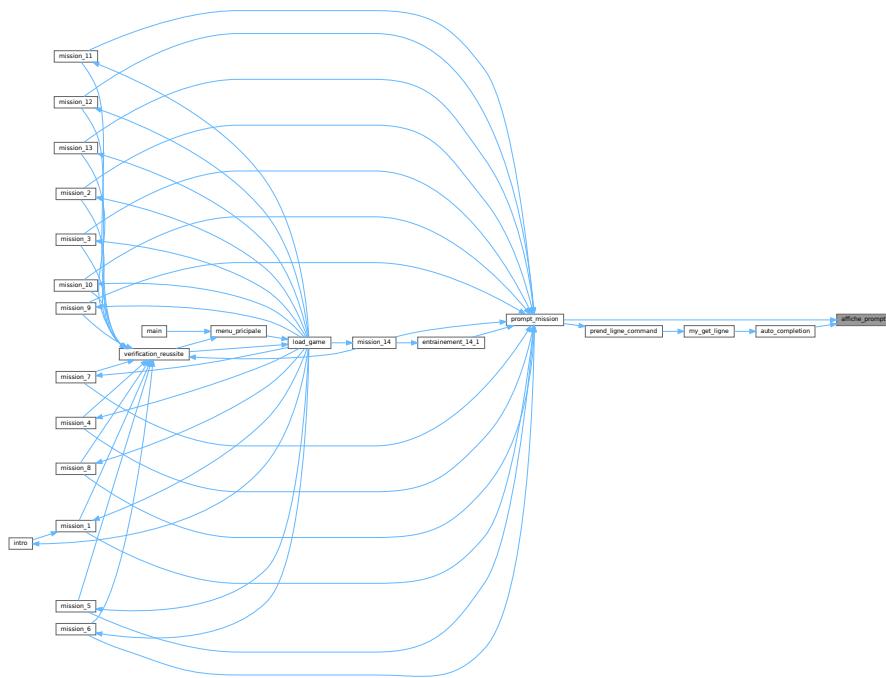
inclusion de tous les fonctions de ce fichier [ certains appelle les autres ]

Definition at line 35 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.4.3 auto\_completion()

```
int auto_completion (
    char * chaine,
    fichier * repertoire_actuel,
    int * nb_Tab,
    int * pos_droite )
```

c'est la fonction principale de l'auto-completion sur ligne de commande

##### Parameters

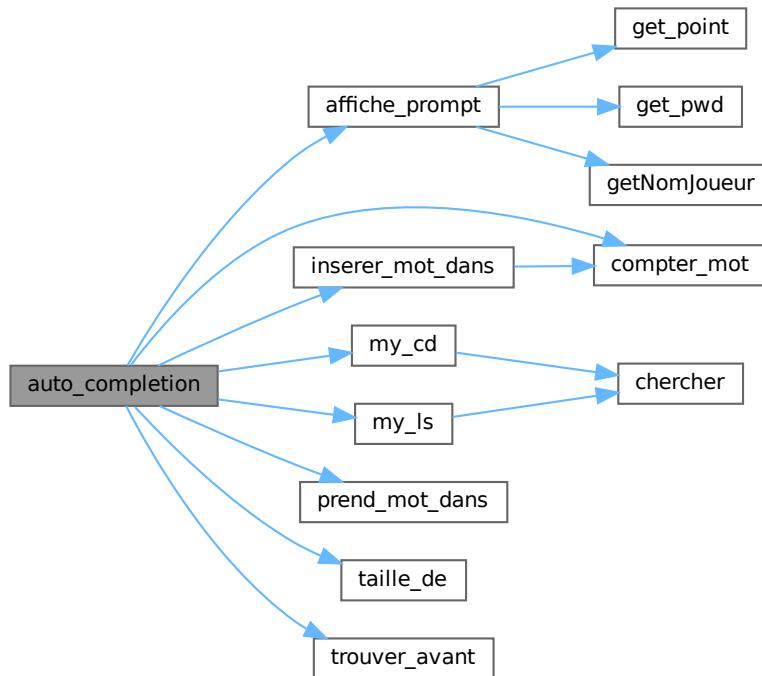
<i>chaine</i>	la chaine à completer
<i>repertoire_actuel</i>	l'adresse du repertoire actuelle
<i>nb_Tab</i>	nombre de tabulation déjà effectuer
<i>pos_droite</i>	la position actuelle par rapport à la position la plus à droite

##### Returns

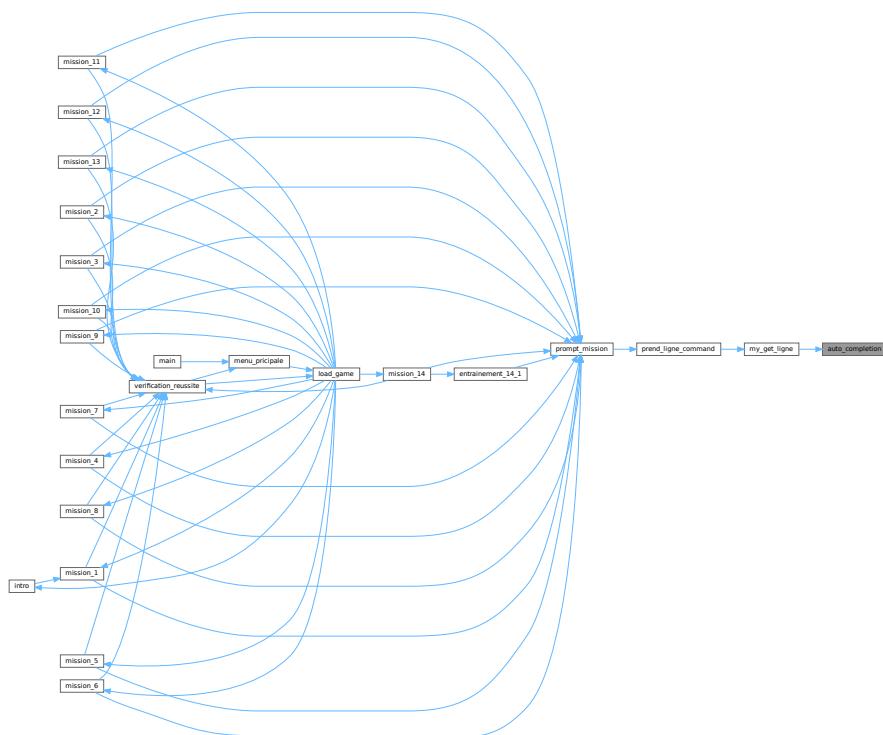
la taille de la chaine en affichage ; donc réussi si différent de zero

Definition at line [408](#) of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.4.4 commande\_option()

```
int commande_option (
    ligne_commande * ligne,
    char ** liste,
    int nb_mot )
```

cette fonction represente une ligne de commande dans une structure `ligne_commande`

##### Parameters

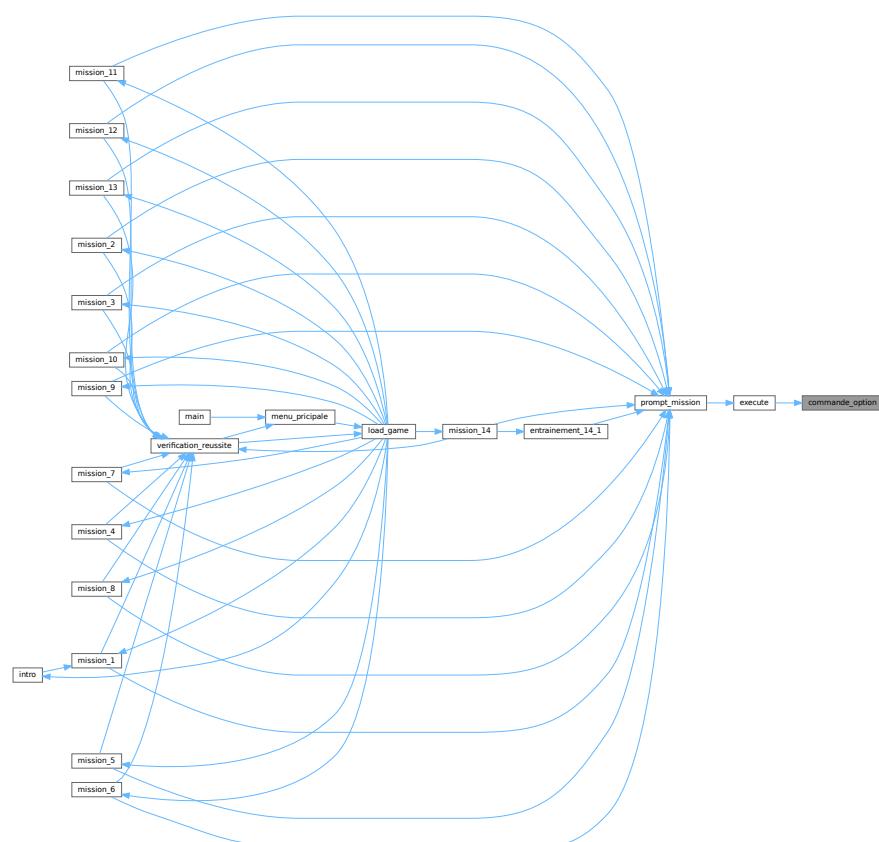
<code>ligne</code>	l'adresse du structure où l'on stockera la representation de la ligne de commande
<code>liste</code>	la ligne actuelle deja separer mot par mot ( dans une tableau mot )
<code>nb_mot</code>	le nombre de mot dans liste

##### Returns

0 si tous s'est bien passer

Definition at line 1305 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.11.4.5 compter\_mot()

```
int compter_mot (
    char * chaine )
```

fonction qui compte le nombre de mot dans une chaine de caractere

##### Parameters

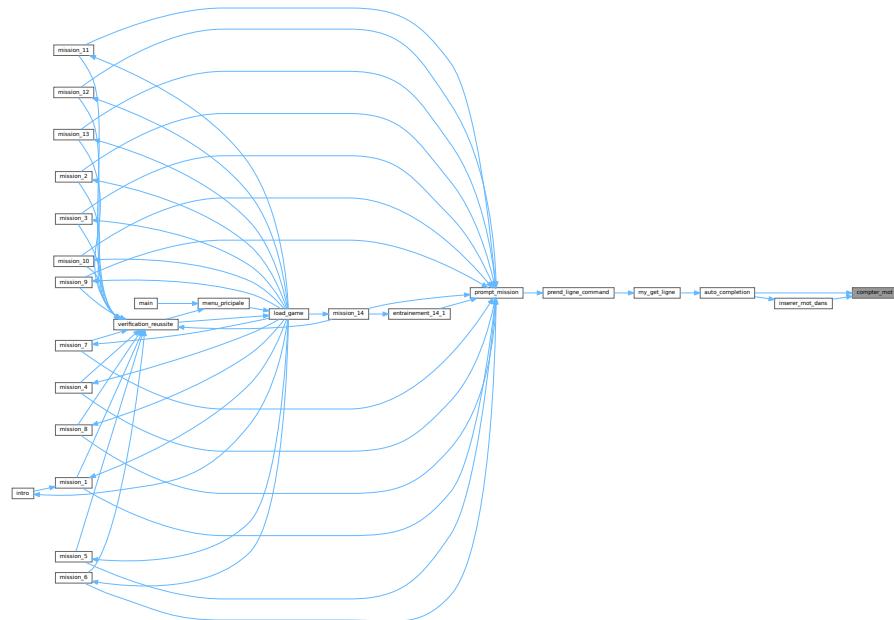
<i>chaine</i>	la chaine dont on veut compter le nombre de mot
---------------	-------------------------------------------------

##### Returns

le nombre de mot dans chaine

Definition at line 908 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.11.4.6 decrement\_score()

```
int decrement_score (
    int penalite )
```

c'est une fonction qui fait decremener le nombre de score d'une joueur en jeu

##### Parameters

<i>penalite</i>	c'est le nombre de point à enlever dans le nombre de score
-----------------	------------------------------------------------------------

**Returns**

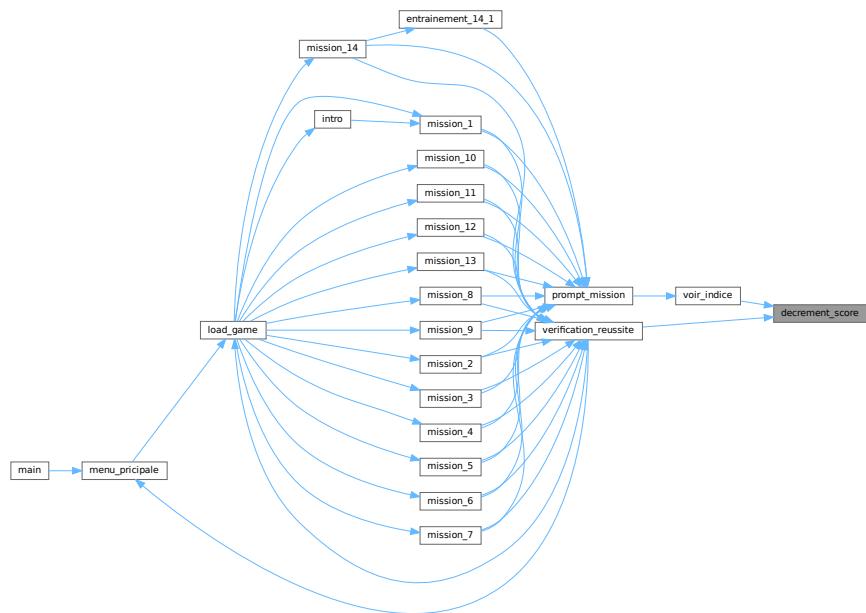
renvoie une entier negatif si le score est insuffisant ou n'existe pas encore

Definition at line 1214 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**4.11.4.7 efface\_cara()**

```
void efface_cara (
    char * chaine,
    int * position )
```

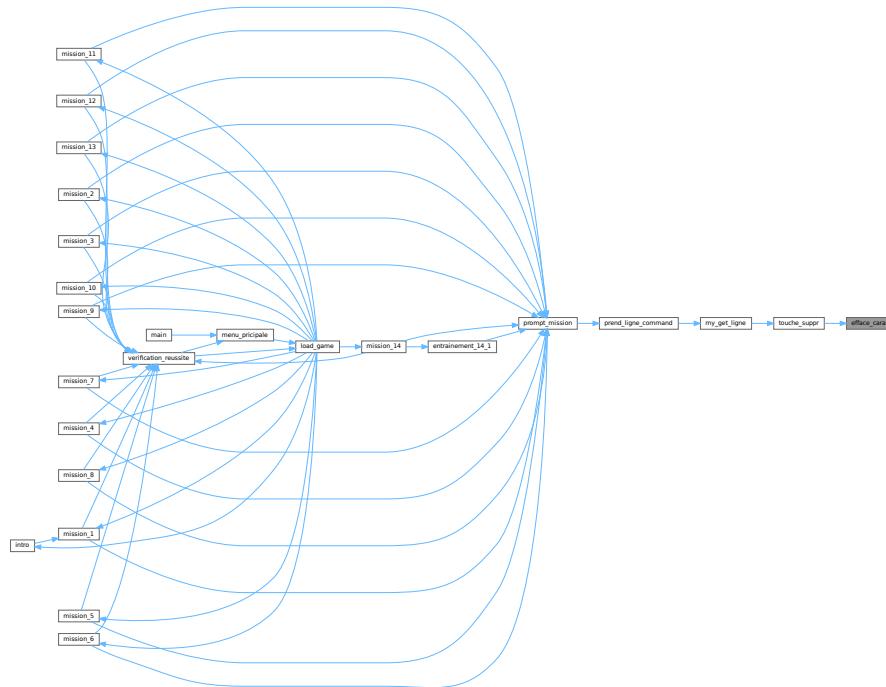
On lui donne une chaine de caractere et une position ; Il supprime le caractere qui se trouve à cette position.

**Parameters**

<i>chaine</i>	la chaine de caractere dont on veut supprimer un caractere
<i>position</i>	la position du caractere dans la chaine

Definition at line 52 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.11.4.8 execute()

```
fichier * execute (
    char ** commande,
    fichier * repertoire_actuel,
    int nbMot )
```

execution du commande entrée en argument

##### Parameters

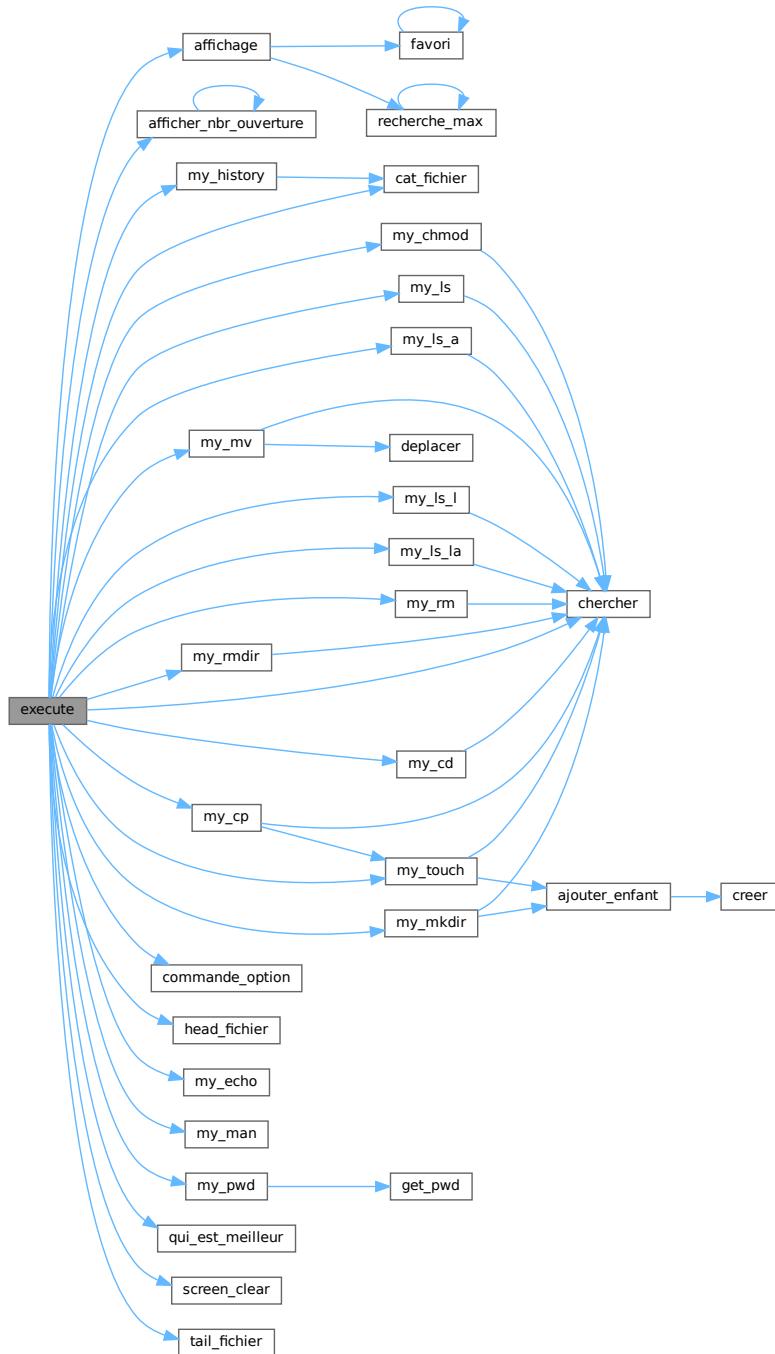
<i>commande</i>	La ligne entrer par l'utilisateur
<i>repertoire_actuel</i>	l'adresse du repertoire où l'on travaille actuellement
<i>nbMot</i>	le nombre de mot prise dans commande qui est de type char**

##### Returns

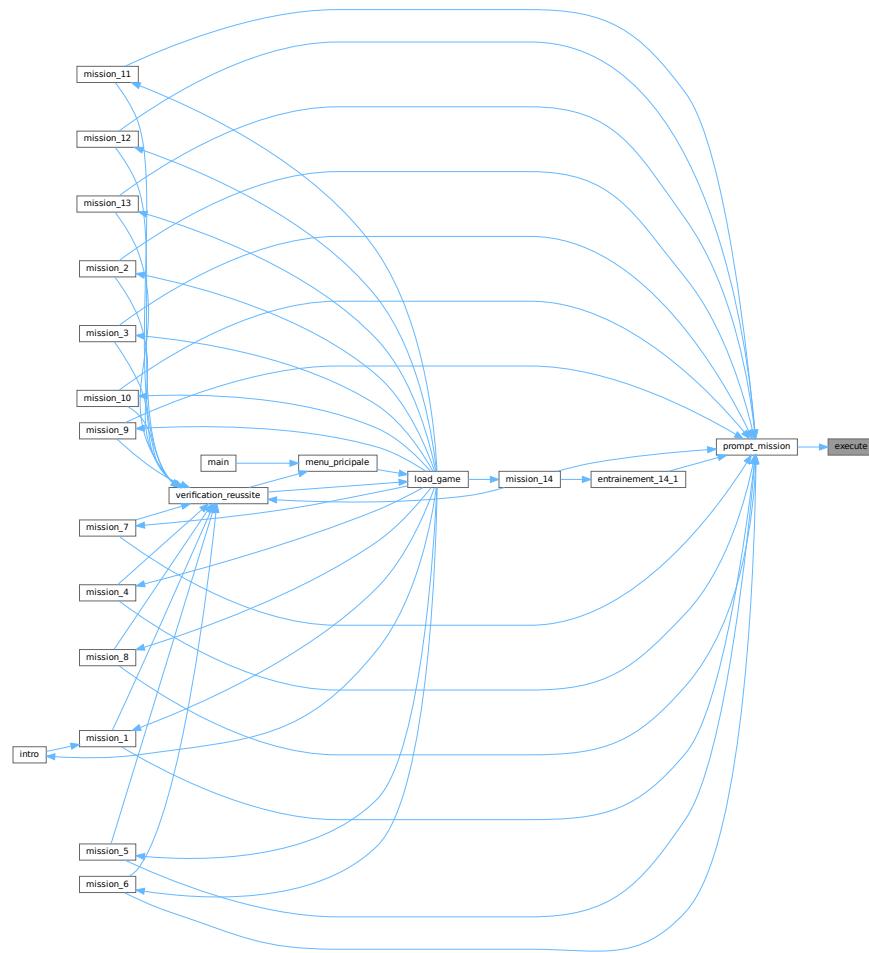
l'adresse du repertoire de travaille qui a peut etre changé durant le prompt

Definition at line 1349 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.4.9 get\_point()

```
int get_point ( )
```

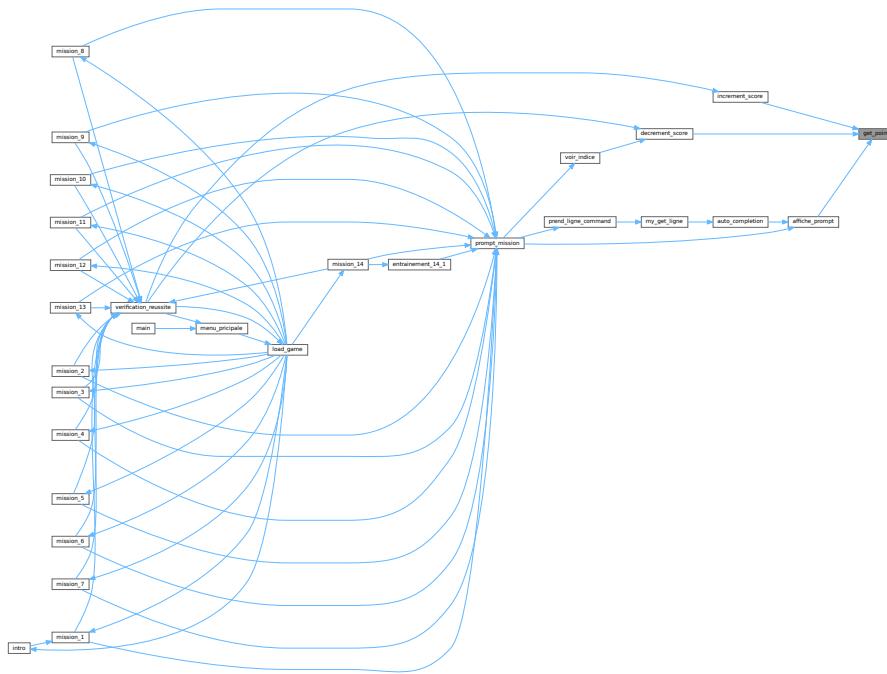
c'est pour savoir le point que le joueur actuel possède

## Returns

il return le nombre de point de l'utilisateur actuel

Definition at line 1121 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.11.4.10 get\_pwd()

```
void get_pwd (fichier * adresse, char * emplacement )
```

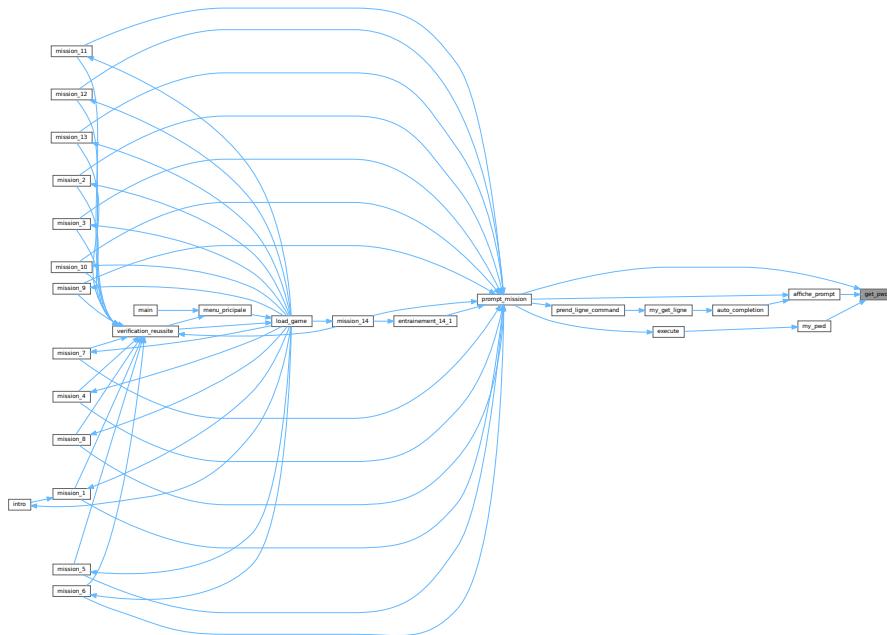
trouver le chemin du repertoire actuelle et puis l'insérer dans emplacement

## Parameters

<i>adresse</i>	l'emplacement actuelle dans l'arborescence
<i>emplacement</i>	la variable ou stocker le resultat

Definition at line 1260 of file [prompt.c](#).

Here is the caller graph for this function:



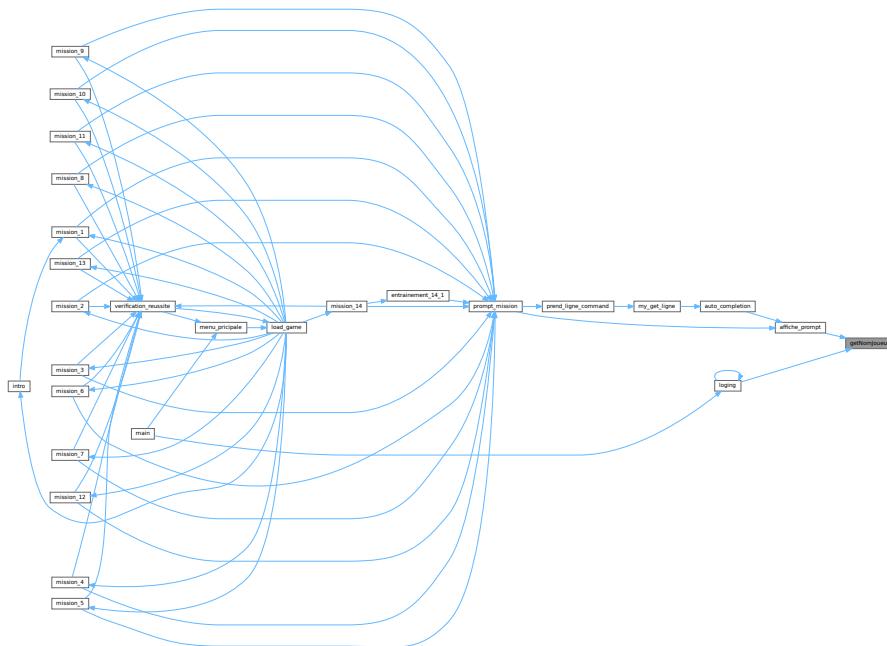
#### 4.11.4.11 getNomJoueur()

```
int getNomJoueur (
    char * emplacement )
```

trouver le nom du joueur actuelle et puis l'inserer dans emplacement

Definition at line 1104 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.11.4.12 increment\_score()

```
void increment_score (
    int actuPTS )
```

c'est pour augmenter le nombre de score du joueur actuel

##### Parameters

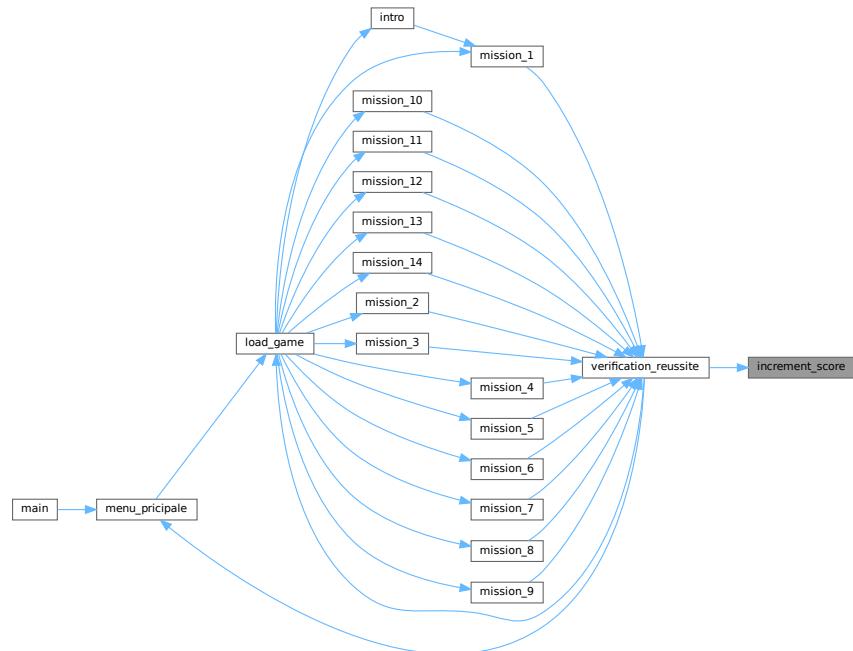
actuPTS	le nombre de point à ajouter au score actuel
---------	----------------------------------------------

Definition at line 1165 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.4.13 inserer()

```
void inserer (
    char caractere,
    int position,
    char * chaine )
```

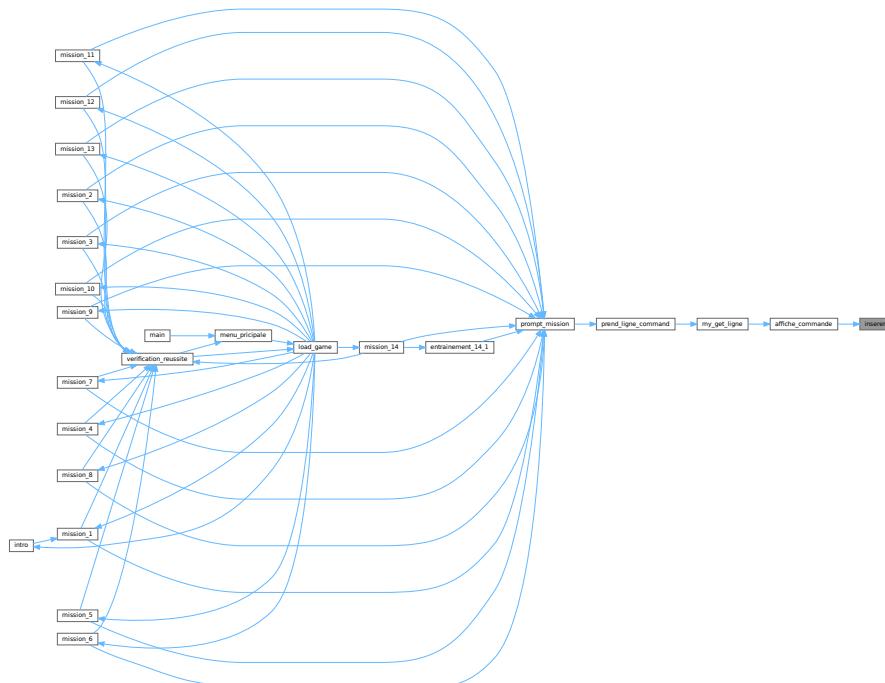
Elle permet d'inserer une caractere dans une chaine à une position preciser en argument.

##### Parameters

<i>caractere</i>	le caractere à inserer dans une chaine
<i>position</i>	la position de l'insertion
<i>chaine</i>	la chaine où l'on va inserer caractere

Definition at line 1006 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.11.4.14 inserer\_mot\_dans()

```
void inserer_mot_dans (
    int position,
    char * chaine,
    char * insertion )
```

c'est pour inserer une mot dans une position precise dans une chaine ; separateur = espace

### Parameters

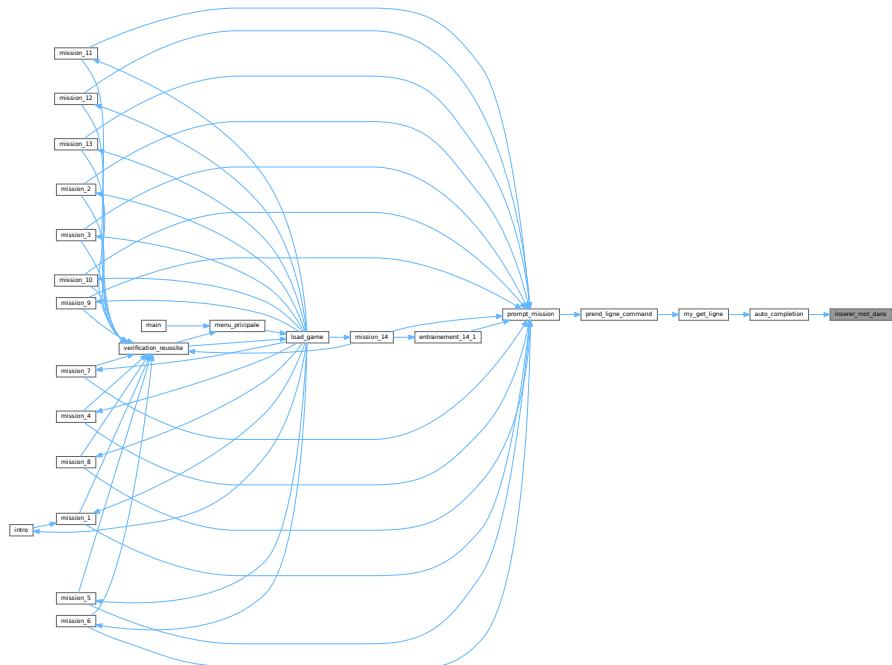
<i>position</i>	la position de l'insertion dans la chaîne
<i>chaine</i>	la chaîne d'insertion
<i>insertion</i>	la chaîne à insérer

Definition at line 818 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



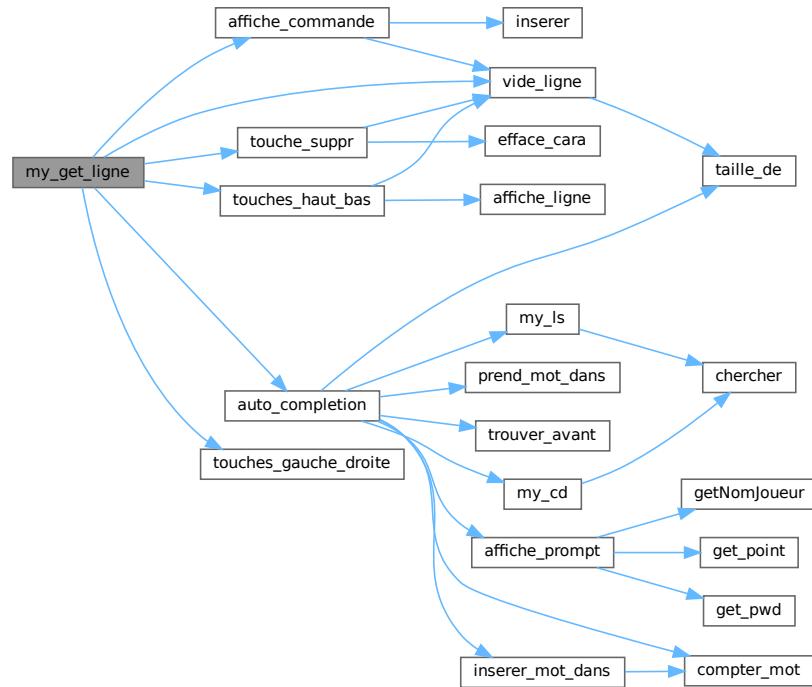
#### 4.11.4.15 my\_get\_ligne()

```

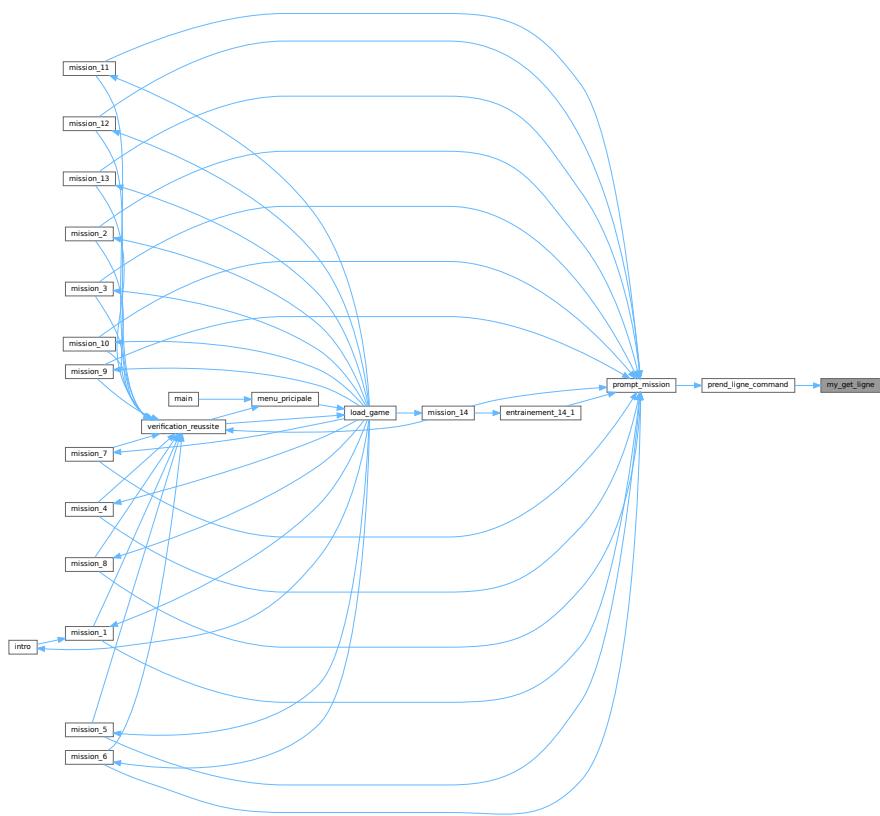
int my_get_ligne (
    char * mobile,
    int taille_Max,
    fichier * repertoire_actuel )
  
```

Definition at line 78 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

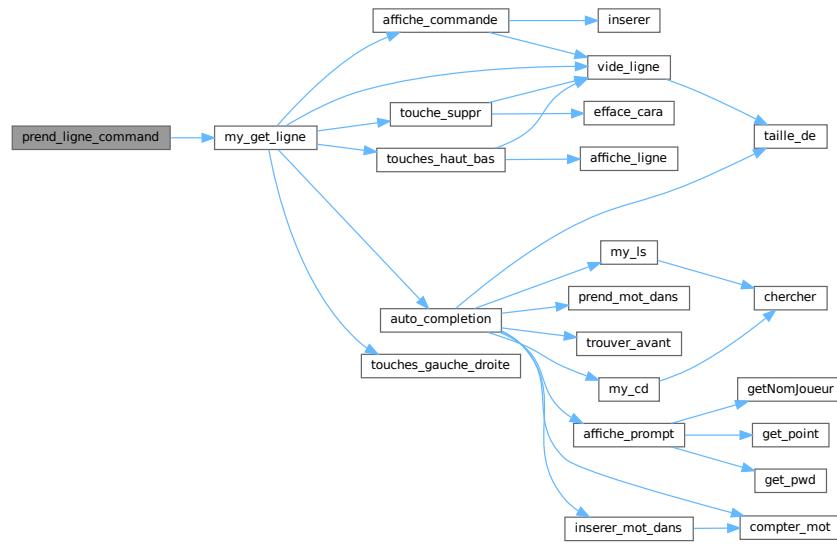


#### 4.11.4.16 `prend_ligne_command()`

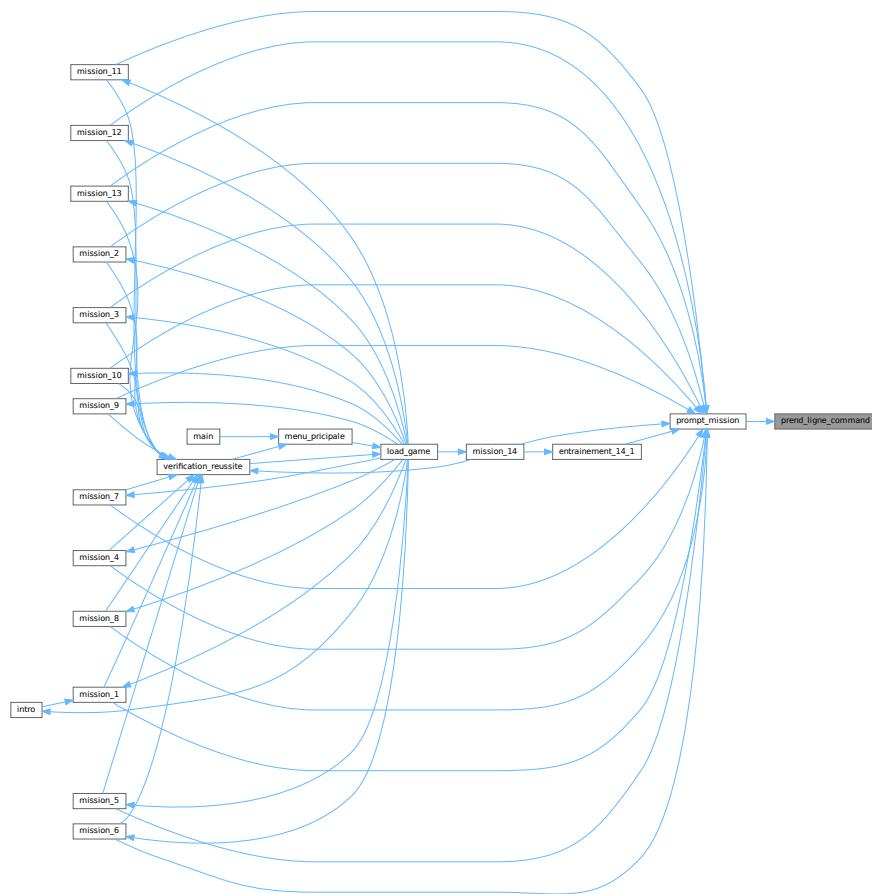
```
int prend_ligne_command (
    char ** emplacement,
    int taille,
    fichier * repertoire_actuel )
```

Definition at line 1027 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.4.17 prend\_mot\_dans()

```
int prend_mot_dans (
    int position,
    char * chaine,
    char * emplacement )
```

##### Parameters

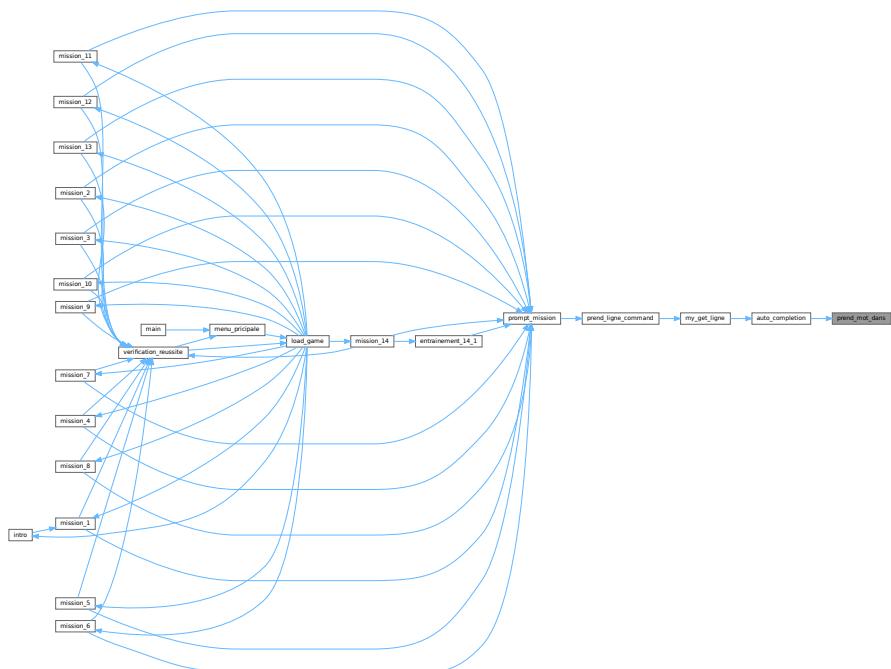
<i>position</i>	la position du mot à prendre
<i>chaine</i>	la chaine de caractere où extraire le mot
<i>emplacement</i>	l'adresse de l'emplacement de la chaine extractée

##### Returns

renvoie 0 si tous s'est bien passé

Definition at line 940 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.11.4.18 rapport\_mission()

```
int rapport_mission (
    int num_mission,
    bool reussie )
```

celui ci écrit le joueur , temps de jeu , réussie ou pas et numéro de mission dans le fichier d'enregistrement

**Parameters**

<i>num_mission</i>	c'est la numero de la mission dont on va faire le rapport
<i>reussie</i>	possede une valeur true si la mission est reussie

**Returns**

1 si l'ecriture s'est bien passe ou -1 si non

Definition at line 1989 of file [prompt.c](#).

**4.11.4.19 taille\_de()**

```
int taille_de (
    char * ligne )
```

elle ne compte pas les caracteres comme strlen ; elle donne la taille en affichage

**Parameters**

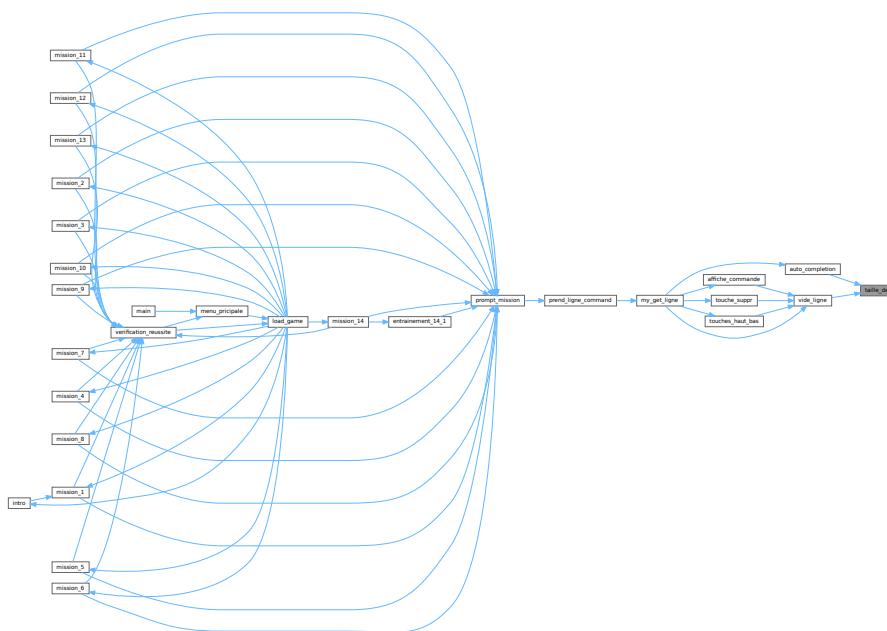
<i>ligne</i>	la chaine de caractere à compter
--------------	----------------------------------

**Returns**

le nombre de caractere de la chaine entrer en argument

Definition at line 384 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.11.4.20 touche\_suppr()

```
void touche_suppr (
    int * position,
    int * pos_droite,
    char * chaine )
```

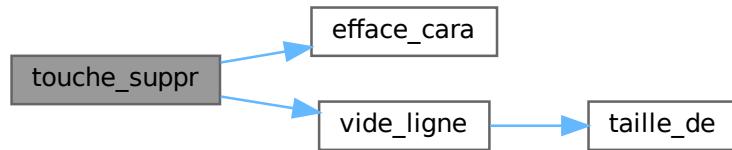
c'est la fonction associer à la touche suppr

##### Parameters

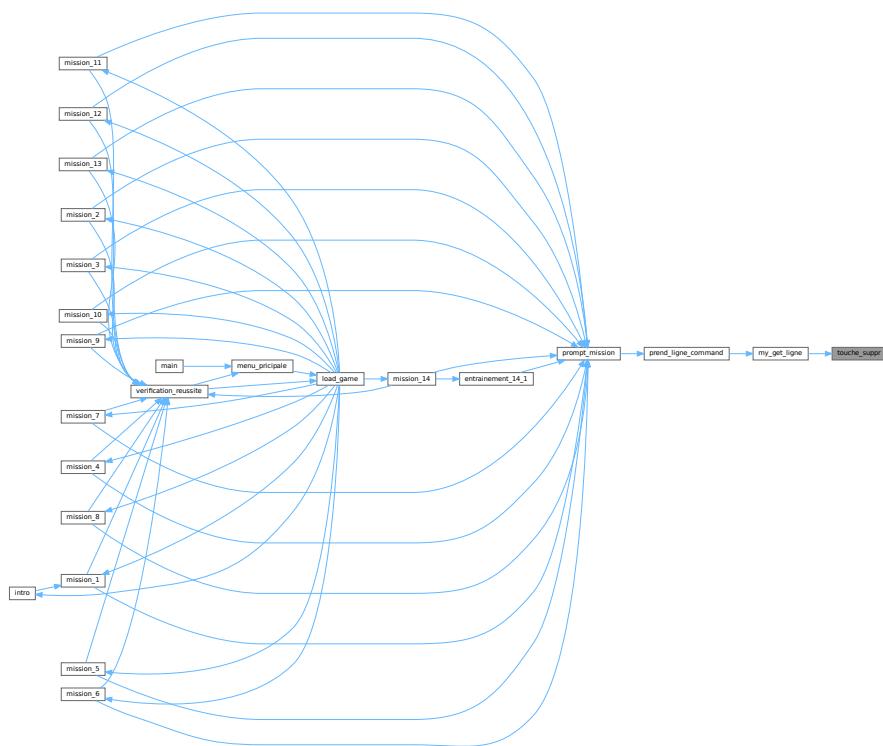
<i>position</i>	la position dans le chaine de caractere
<i>pos_droite</i>	la position de curseur en comptant depuis la droite
<i>chaine</i>	le ligne où l'on veut supprimer une caractere

Definition at line 348 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.4.21 touches\_gauche\_droite()

```
void touches_gauche_droite (
    int * indice,
    int * pos_droite,
    char * chaine,
    int gauche )
```

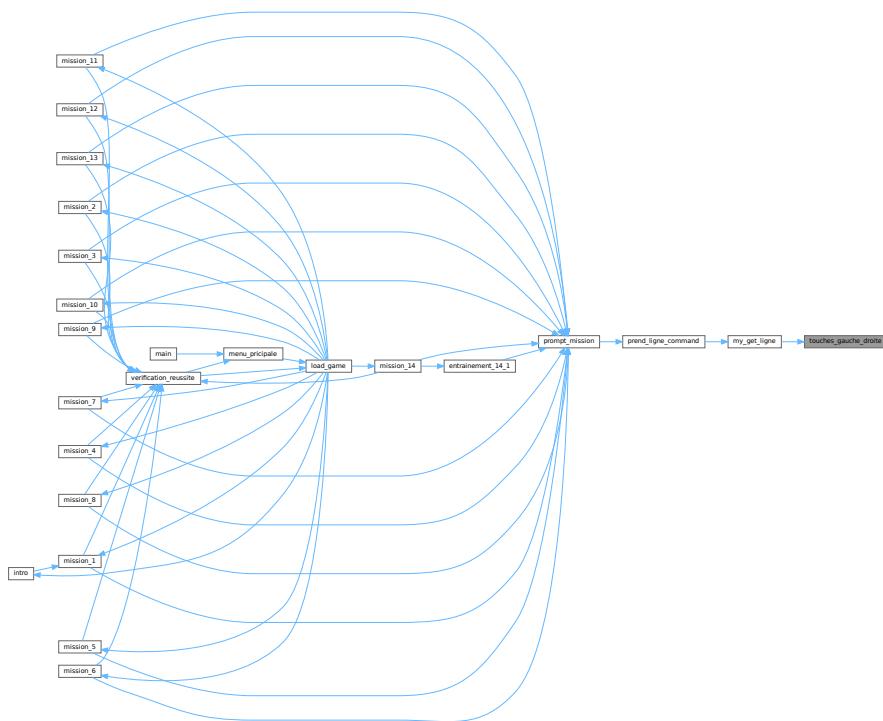
c'est la fonction associer à la touche gauche et droite

##### Parameters

<i>indice</i>	la position dans la ligne actuelle
<i>pos_droite</i>	la position par rapport à la position la plus à droite
<i>chaine</i>	la ligne actuelle
<i>gauche</i>	valeur si 0 si touche droite et 1 si touche gauche

Definition at line 231 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.11.4.22 touches\_haut\_bas()

```
void touches_haut_bas (
    int * indice,
    int * position,
    int * droite,
    char * chaine,
    char * tmp,
    int haut )
```

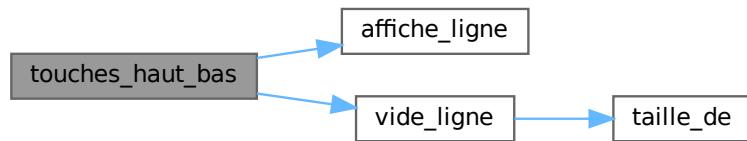
la fonctionne associer à la touche haut et bas

##### Parameters

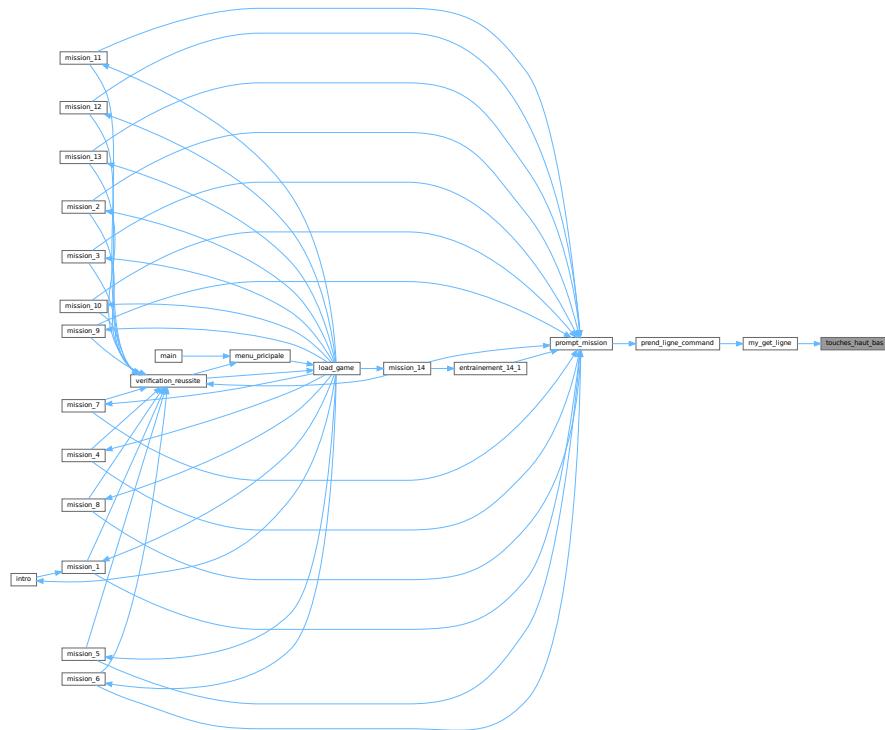
<i>indice</i>	position dans la chaine
<i>position</i>	position actuelle dans les visions des historiques des commandes
<i>droite</i>	position par rapport à la plus droite
<i>chaine</i>	la chaine de caractere contenant la ligne
<i>tmp</i>	la chaine ou l'on stocke une ligne temporairement ; ici la ligne la plus bas est enregistrer
<i>haut</i>	1 si touche haut et 0 si touche bas

Definition at line 276 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.4.23 trouver\_avant()

```
int trouver_avant (
    char * chaine,
    char * recherche )
```

Elle permet de trouver si la chaîne recherche est le début de chaîne ; les espaces sont ignorés.

##### Parameters

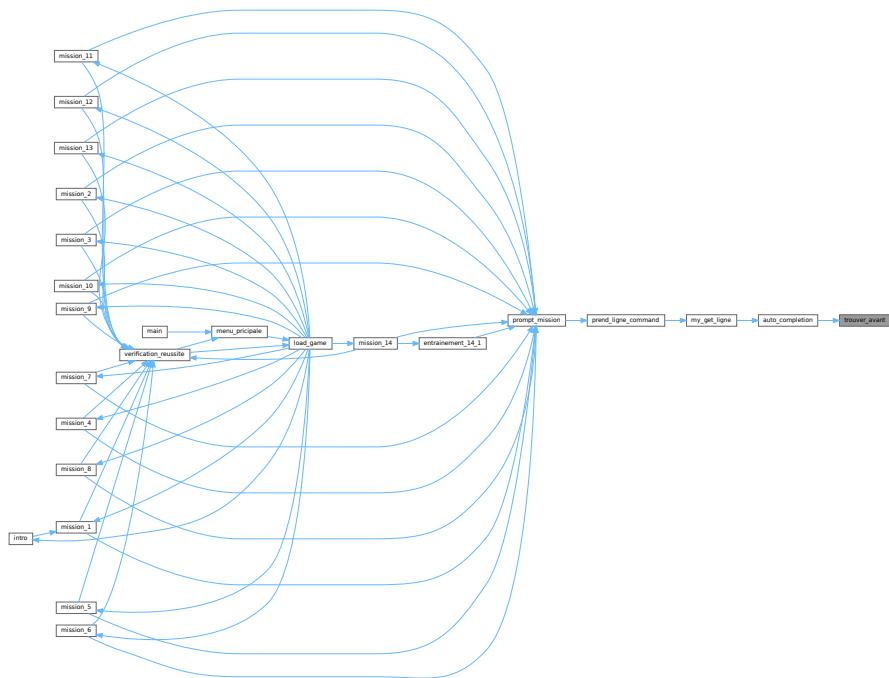
<i>chaine</i>	la chaîne de recherche
<i>recherche</i>	la chaîne à rechercher

## Returns

Elle renvoie la taille de chaine semblable

Definition at line 889 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.11.4.24 vide\_ligne()

```
void vide_ligne ( int droite, char * chaine )
```

c'est une simple fonction qui permet de vider la ligne de commande actuelle

## Parameters

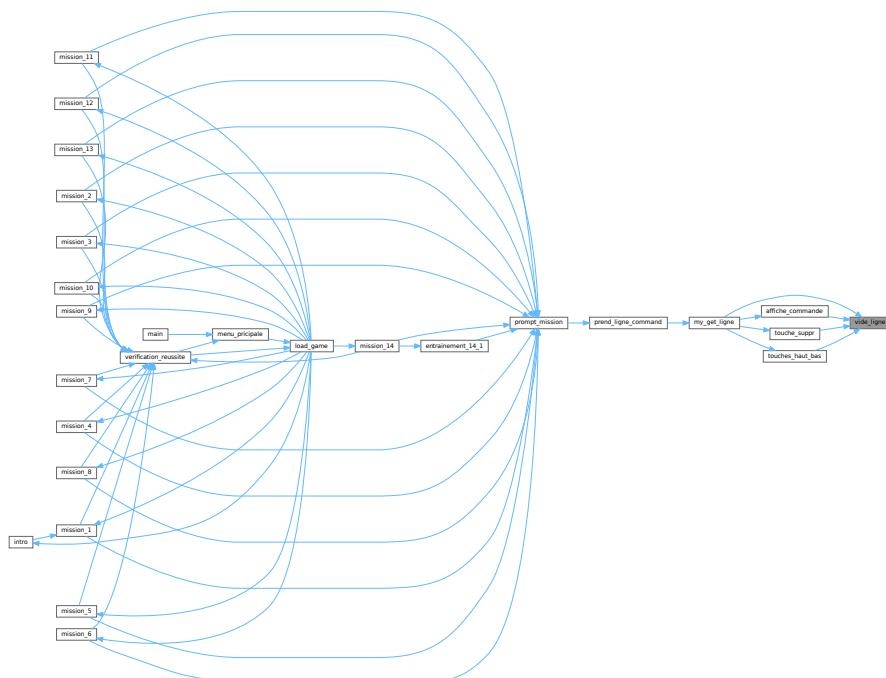
<i>droite</i>	la position par rapport à la plus droite
<i>chaine</i>	la chaîne de caractère de la ligne à vider

Definition at line 368 of file `prompt.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.12 prompt.h

[Go to the documentation of this file.](#)

```

00001
00006 #ifndef DECL
00007 #define DECL
00008
00013     #define SORTIR "exit"
00018     #include "../include/arborescence.h"
00023     #include <stdbool.h>
00024
00029     #define LIGNE 1000
00034     #define MOT 100
00040     struct ligne
00041     {
00042         char *commande;
00043         char **option;
00044         char **argument;
00045     };
00046     typedef struct ligne ligne_commande;
00051     void affiche_prompt(fichier *repertoire_actuel);
00052
  
```

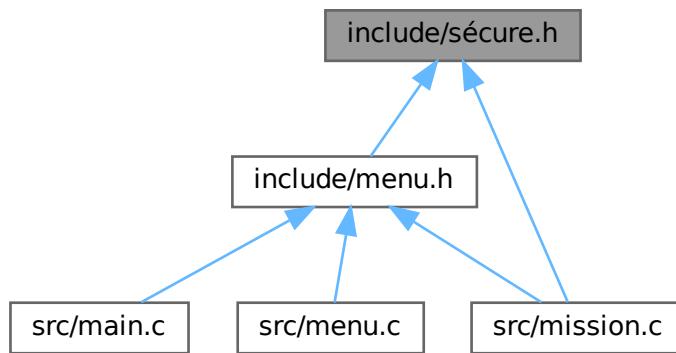
```

00058     int prend_ligne_command(char **emplacement, int taille , fichier *repertoire_actuel);
00059
00064     int getNomJoueur(char *emplacement);
00065
00074     fichier *execute(char **commande , fichier *repertoire_actue , int nbMot);
00075
00082     void get_pwd(fichier *adresse, char *emplacement);
00088     int get_point();
00094     void increment_score(int actuPTS);
00095
00102     int decrement_score(int penalite);
00112     int my_get_ligne(char *mobile , int taille_Max , fichier *repertoire_actuel);
00113
00121     int rapport_mission(int num_mission , bool reussie);
00128     void vide_ligne(int droite , char *chaine);
00135     void efface_cara(char *chaine , int *position);
00142     int taille_de(char *ligne);
00150     int prend_mot_dans(int position , char *chaine , char *emplacement);
00160     int auto_completion(char *chaine , fichier *repertoire_actuel , int *nb_Tab , int *pos_droite);
00168     void touche_suppr(int *position , int *pos_droite , char *chaine);
00177     void touches_gauche_droite(int *indice , int *pos_droite , char *chaine , int gauche);
00188     void touches_haut_bas(int *indice , int *position , int *droite , char *chaine , char *tmp , int haut);
00196     void insérer_mot_dans(int position , char *chaine , char *insertion);
00204     int trouver_avant(char *chaine , char *recherche);
00213     void affiche_commande(int *indice , char caractere ,char *chaine ,int droite);
00220     int compter_mot(char *chaine);
00229     int commande_option(ligne_commande *ligne , char **liste , int nb_mot);
00237     void insérer(char caractere , int position , char *chaine);
00238
00239 #endif

```

## 4.13 include/sécurité.h File Reference

This graph shows which files directly or indirectly include this file:



### Functions

- void lireLigne (char \*line, int taille)
- int lireEntier ()  
*Lire un entier.*
- float lireFloat ()  
*Lit un float.*
- double lireDouble ()  
*Lit un double.*

### 4.13.1 Function Documentation

#### 4.13.1.1 lireDouble()

```
lireDouble ( )
```

Lit un double.

##### Returns

Return un variable de type double

Definition at line 69 of file [sécur.c](#).

Here is the call graph for this function:



#### 4.13.1.2 lireEntier()

```
lireEntier ( )
```

Lire un entier.

##### Returns

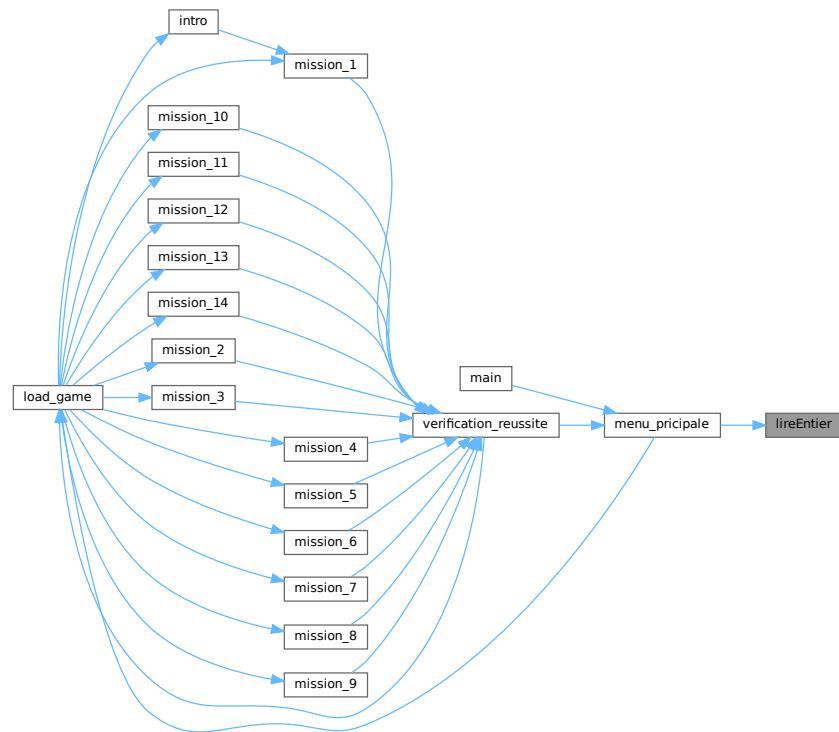
Return l'entier entré

Definition at line 40 of file [sécur.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.1.3 lireFloat()

```
lireFloat ( )
```

Lit un float.

##### Returns

Return le float entré

Definition at line 55 of file [sécur.c](#).

Here is the call graph for this function:

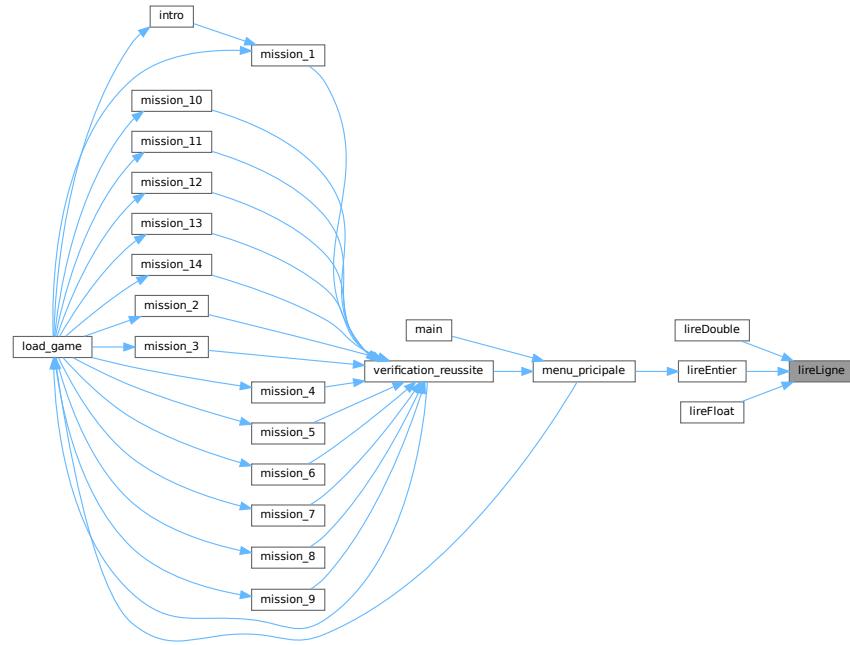


#### 4.13.1.4 lireLigne()

```
void lireLigne (
    char * line,
    int taille )
```

Definition at line 20 of file [sécure.c](#).

Here is the caller graph for this function:



## 4.14 sécurité.h

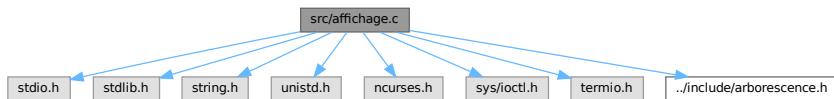
[Go to the documentation of this file.](#)

```
00001 #ifndef LECTURE_SECURISÉ
00002 #define LECTURE_SECURISÉ
00003
00004 void lireLigne( char *line , int taille);
00005 int lireEntier();
00006 float lireFloat();
00007 double lireDouble();
00008
00009 #endif
```

## 4.15 src/affichage.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ncurses.h>
#include <sys/ioctl.h>
```

```
#include <termio.h>
#include "../include/arborescence.h"
Include dependency graph for affichage.c:
```



## Macros

- `#define ESPACE 7`

## Functions

- `void screen_clear ()`  
*Fonction pour nettoyer l'écran.*
- `void affichage_ligne (char *nom_fichier)`
- `void affiche_fichier (char *nom_fichier)`
- `bool felicitation ()`  
*Affiche un message de félicitation.*
- `bool echec ()`  
*Affiche un message d'échec.*
- `void afficher_tableau_avec_titre_position_exacte (const char *titre, const char *nom_fichier)`
- `void affiche_fichier_tableau curse_position_exacte (const char *titre, const char *nom_fichier)`

### 4.15.1 Macro Definition Documentation

#### 4.15.1.1 ESPACE

```
#define ESPACE 7
```

Definition at line 11 of file [affichage.c](#).

### 4.15.2 Function Documentation

#### 4.15.2.1 affichage\_ligne()

```
void affichage_ligne (
    char * nom_fichier )
```

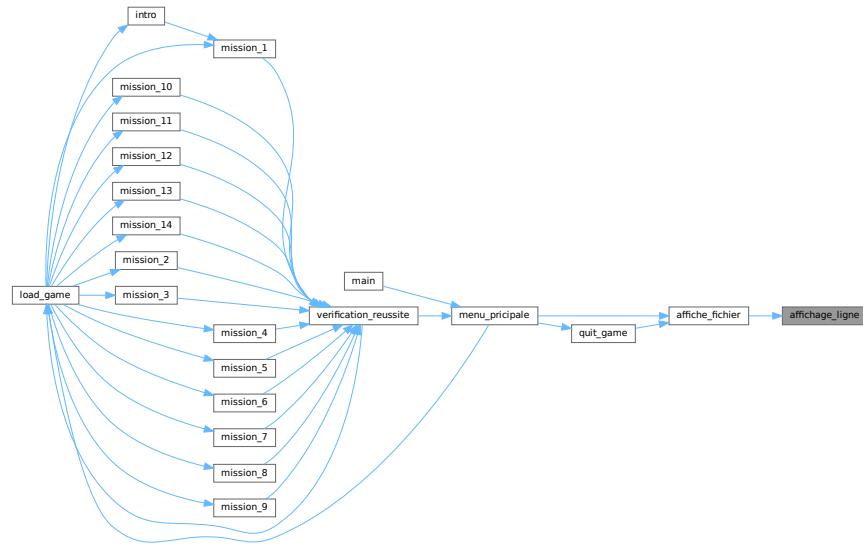
Affiche les lignes d'une phrase successivement

#### Parameters

a	: Le nom du fichier à afficher
---	--------------------------------

Definition at line 23 of file [affichage.c](#).

Here is the caller graph for this function:

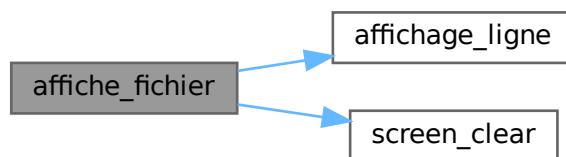


#### 4.15.2.2 `affiche_fichier()`

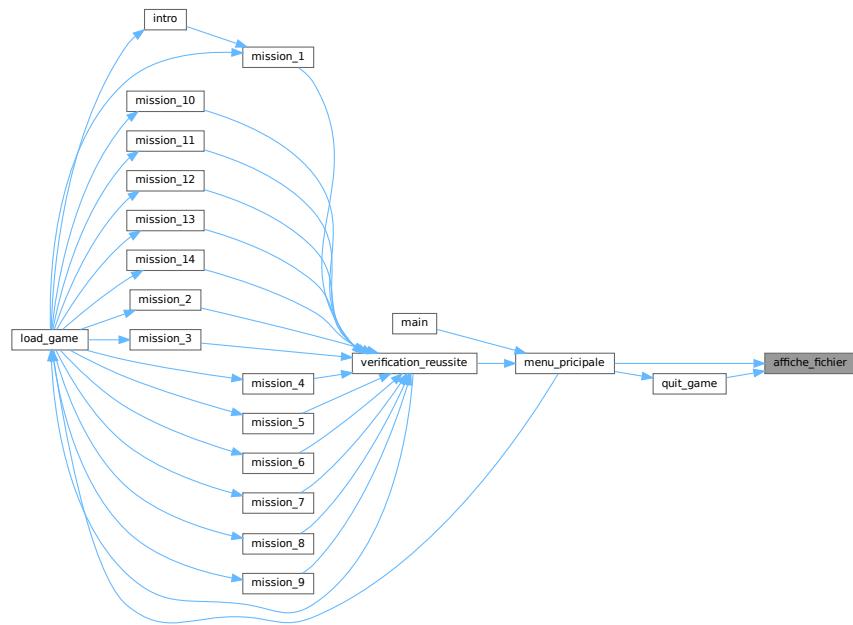
```
void affiche_fichier (
    char * nom_fichier )
```

Definition at line 45 of file [affichage.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

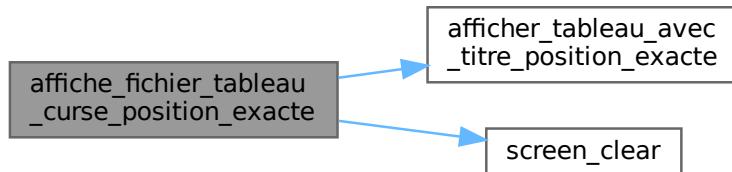


#### 4.15.2.3 affiche\_fichier\_tableau curse\_position\_exacte()

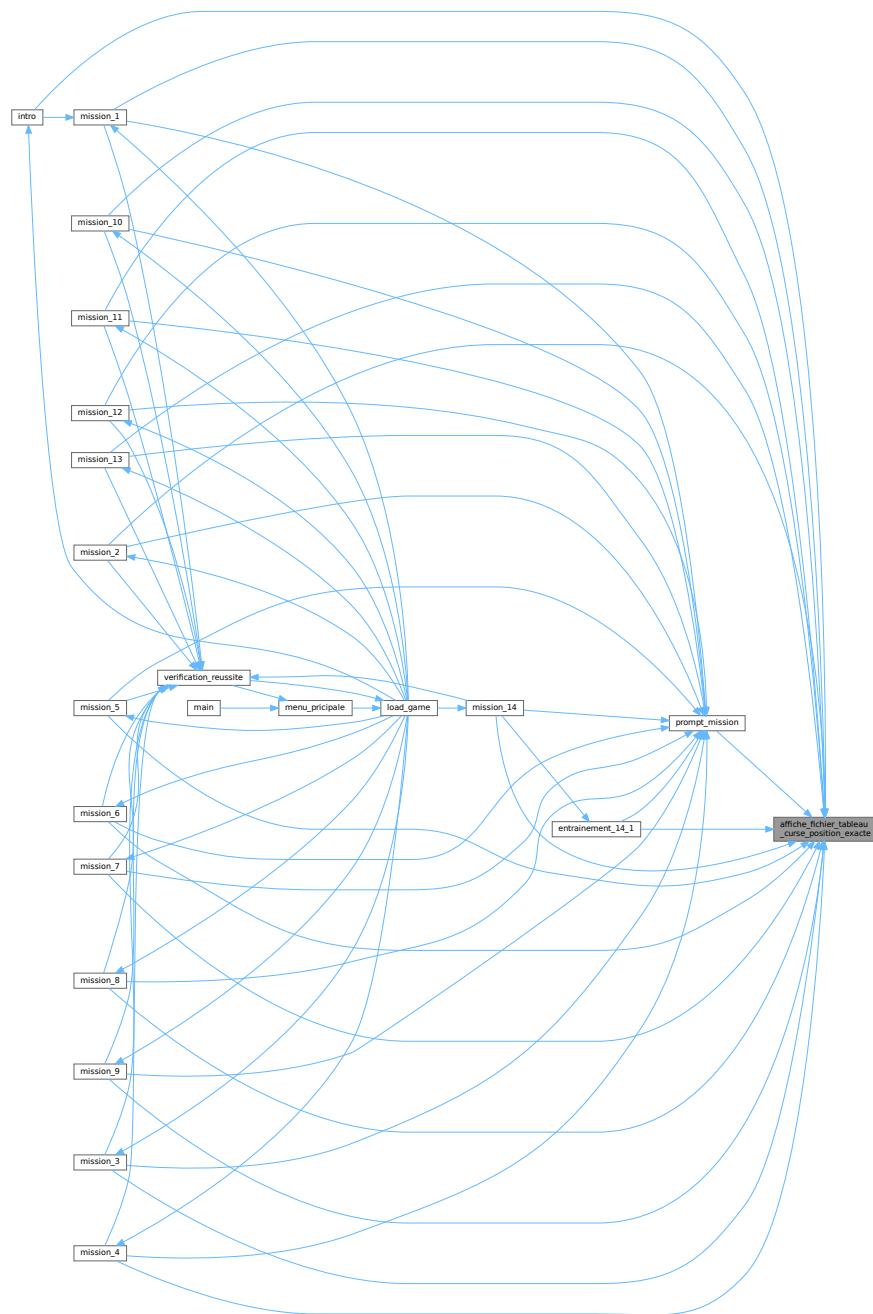
```
void affiche_fichier_tableau curse_position_exacte (
    const char * titre,
    const char * nom_fichier )
```

Definition at line 292 of file [affichage.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

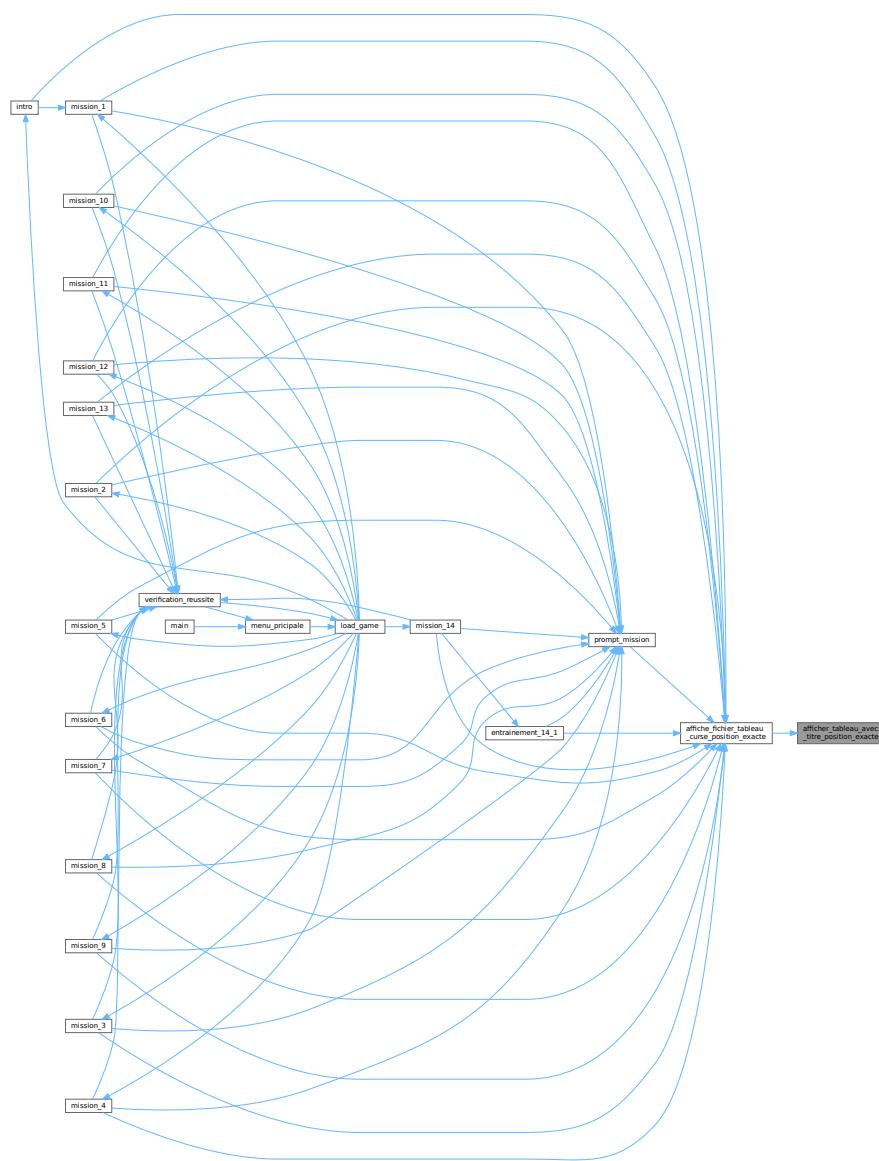


#### 4.15.2.4 afficher\_tableau\_avec\_titre\_position\_exacte()

```
void afficher_tableau_avec_titre_position_exacte (
    const char * titre,
    const char * nom_fichier )
```

Definition at line 190 of file [affichage.c](#).

Here is the caller graph for this function:



#### 4.15.2.5 echec()

```
bool echec ( )
```

Affiche un message d'échec.

Affiche une message d'échec et récupère l'avis du joueur s'il souhaite recommencé ou pas.

Affiche une message de félicitation et récupère l'avis du joueur s'il souhaite passé au mission suivant ou pas.

**Returns**

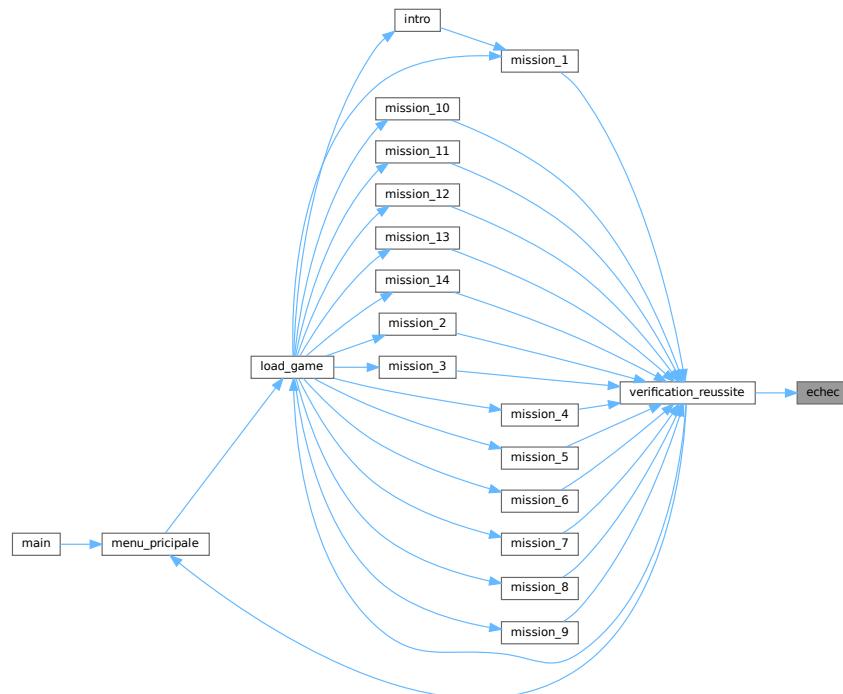
Retourne un bool

Definition at line 125 of file [affichage.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



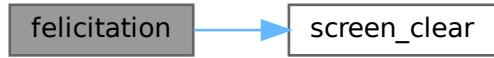
#### 4.15.2.6 felicitation()

```
bool felicitation ( )
```

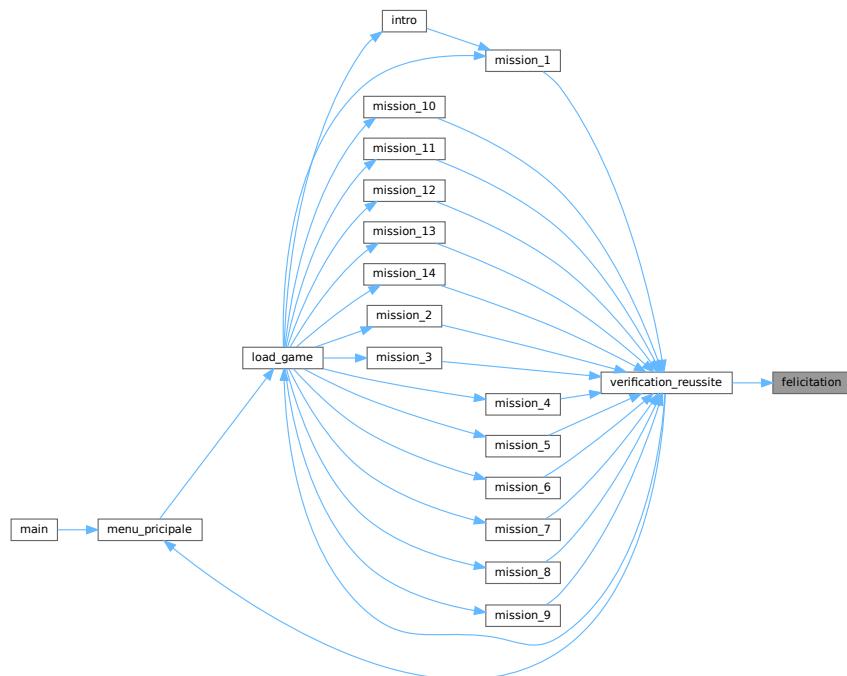
Affiche un message de félicitation.

Definition at line 57 of file [affichage.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



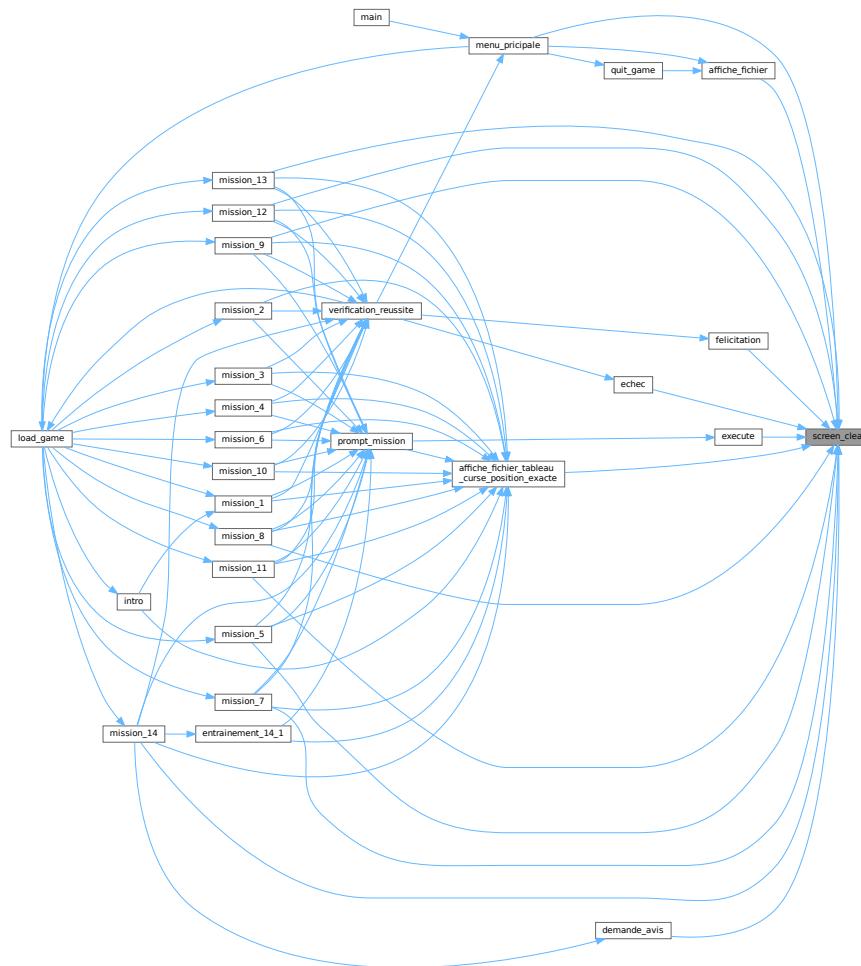
#### 4.15.2.7 screen\_clear()

```
void screen_clear ( )
```

Fonction pour nettoyer l'écran.

Definition at line 14 of file [affichage.c](#).

Here is the caller graph for this function:



## 4.16 affichage.c

[Go to the documentation of this file.](#)

```

00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <string.h>
00004 #include <unistd.h>
00005 #include <ncurses.h>
00006 #include <sys/ioctl.h>
00007 #include <termio.h>
00008 #include "../include/arborescence.h"
00009
00010
00011 #define ESPACE 7 // Espace du texte dans la fonction echec() et felicitation()
00012
00013 // Clear the terminal screen
00014 void screen_clear()
00015 {
00016     system("clear");
00017 }
00018
00019 void affichage_ligne(char *nom_fichier)
00020 {
00021     FILE *fichier = NULL ;
00022     char texte[1000]="";
00023     fichier = fopen( nom_fichier , "r" );
00024     if(fichier == NULL)
  
```

```

00030      {
00031          printf("!!! Erreur d'ouverture fichier !!!\n");
00032      }
00033
00034      while( fgets(texte , sizeof(texte) , fichier) )
00035      {
00036          printf("%s",texte);
00037          // fflush(stdout);
00038          usleep(10000) ;
00039      }
00040
00041      fclose(fichier);
00042  }
00043
00044 // Affiche le fichier
00045 void affiche_fichier(char *nom_fichier )
00046 {
00047
00048     screen_clear();
00049     affichage_ligne(nom_fichier );
00050 }
00051
00052 bool felicitation()
00053 {
00054     screen_clear();
00055
00056     struct winsize terminal ; // Structure contenant la taille du terminal
00057     int touche , choix ;
00058     int ligne = 1 ;
00059
00060     ioctl( STDOUT_FILENO , TIOCGWINSZ , &terminal ); // Récupre la taille du terminal
00061
00062     initscr(); // Début ncurses
00063
00064     curs_set(0); // Désactiver le curseur
00065     cbreak();
00066     noecho(); // N'affiche pas les touche tapé par le joueur
00067     keypad(stdscr , TRUE ); // Activer les touche de direction du clavier
00068
00069     start_color();
00070     init_pair(1 , COLOR_MAGENTA , -1 ); // ID = 1 texte rouge sur fond noir
00071     init_pair(2 , COLOR_GREEN , -1 ); // texte en vert
00072
00073     mvprintw( terminal.ws_row/2 + 2 , (terminal.ws_col-strlen("MISSION TERMINÉE ! BRAVO SOLDAT !"))/2
00074     , "==>");
00075
00076     while(1)
00077     {
00078         attron(COLOR_PAIR(1) | A_BOLD );
00079         mvprintw( terminal.ws_row/2 , (terminal.ws_col-strlen("MISSION TERMINEE ! BRAVO
00080             SOLDAT !"))/2 , "MISSION TERMINEE ! BRAVO SOLDAT !");
00081         attroff(COLOR_PAIR(1) | A_BOLD );
00082         attron( COLOR_PAIR(2) | A_BOLD );
00083         mvprintw( terminal.ws_row/2 + 2 , (terminal.ws_col)/2 - ESPACE , "CONTINUER");
00084         mvprintw( terminal.ws_row/2 + 3 , (terminal.ws_col)/2 - ESPACE , "MENU PRINCIPALE");
00085         attroff( COLOR_PAIR(2) | A_BOLD );
00086
00087         refresh();
00088
00089         touche = getch(); // Récupère une touche
00090
00091         if(touche == KEY_UP && ligne > 1 )
00092         {
00093             clear();
00094             mvprintw( terminal.ws_row/2 + 2 , (terminal.ws_col-strlen("MISSION TERMINÉE ! BRAVO
00095                 SOLDAT !"))/2 , "==>"); // Mission terminée
00096             ligne--;
00097         }
00098         else if(touche == KEY_DOWN && ligne < 2)
00099         {
00100             clear();
00101             mvprintw( terminal.ws_row/2 + 3 , (terminal.ws_col-strlen("MISSION TERMINÉE ! BRAVO
00102                 SOLDAT !"))/2 , "==>"); // Menu principale
00103             ligne++;
00104         }
00105         else if( touche == '\n' && ligne == 1 )
00106         {
00107             clear();
00108             endwin();
00109             return true;
00110         }
00111         else if( touche == '\n' && ligne == 2 )
00112         {
00113             clear();
00114             endwin();
00115             return false;
00116         }
00117     }

```

```

00118         }
00119     }
00120     bool echec()
00121 {
00122     screen_clear();
00123
00124     struct winsize terminal ; // Structure contenant la taille du terminal
00125     int touche , choix ;
00126     int ligne = 1 ;
00127
00128     ioctl( STDOUT_FILENO , TIOCGWINSZ , &terminal );
00129
00130     initscr(); // Début ncurses
00131
00132     curs_set(0);
00133     cbreak();
00134     noecho();
00135     keypad(stdscr , TRUE );
00136
00137     start_color();
00138     init_pair(1 , COLOR_MAGENTA , -1 ); // ID = 1 texte rouge sur fond noir
00139     init_pair(2 , COLOR_GREEN , -1 ); // texte en vert
00140
00141     mvprintw( terminal.ws_row/2 + 2 , (terminal.ws_col-strlen("MISSION ECHOUÉE ! RETENZE VOTRE CHANCE
SOLDAT !"))/2 , "==>" );
00142
00143     while(1)
00144     {
00145         attron(COLOR_PAIR(1) | A_BOLD );
00146         mvprintw( terminal.ws_row/2 , (terminal.ws_col-strlen("MISSION ECHOUÉE ! RETENZE
VOTRE CHANCE SOLDAT !"))/2 , "MISSION ECHOUÉE ! RETENZE VOTRE CHANCE SOLDAT !" );
00147        attroff(COLOR_PAIR(1) | A_BOLD );
00148         attron( COLOR_PAIR(2) | A_BOLD );
00149         mvprintw( terminal.ws_row/2 + 2 , (terminal.ws_col)/2 - ESPACE , "RECOMMENCER");
00150         mvprintw( terminal.ws_row/2 + 3 , (terminal.ws_col)/2 - ESPACE , "MENU PRINCIPALE");
00151        attroff( COLOR_PAIR(2) | A_BOLD );
00152
00153         refresh();
00154
00155         touche = getch();
00156
00157         if(touche == KEY_UP && ligne > 1 )
00158         {
00159             clear();
00160             mvprintw( terminal.ws_row/2 + 2 , (terminal.ws_col-strlen("MISSION ECHOUÉE ! RETENZE
VOTRE CHANCE SOLDAT !"))/2 , "==>" );
00161             ligne--;
00162         }
00163         else if(touche == KEY_DOWN && ligne < 2)
00164         {
00165             clear();
00166             mvprintw( terminal.ws_row/2 + 3 , (terminal.ws_col-strlen("MISSION ECHOUÉE ! RETENZE
VOTRE CHANCE SOLDAT !"))/2 , "==>" );
00167             ligne++;
00168         }
00169         else if( touche == '\n' && ligne == 1 )
00170         {
00171             clear();
00172             endwin();
00173             return true;
00174         }
00175         else if( touche == '\n' && ligne == 2 )
00176         {
00177             clear();
00178             endwin();
00179             return false;
00180         }
00181     }
00182
00183 }
00184
00185 void afficher_tableau_avec_titre_position_exacte(const char *titre, const char *nom_fichier)
00186 {
00187     FILE *fichier = fopen(nom_fichier, "r");
00188     char lignes[300][300];
00189     int nb = 0;
00190     int max_len = 0;
00191     int hauteur , largeur , starty , startx , x_centre , y_centre , longueur_ligne ;
00192
00193     if (!fichier)
00194     {
00195         endwin();
00196         printf("Erreur : impossible d'ouvrir %s\n", nom_fichier);
00197         return;
00198     }
00199
00200     while (fgets(lignes[nb], sizeof(lignes[nb]), fichier) && nb < 300)
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01
```

```

00206     {
00207         longueur_ligne = strlen(lignes[nb]);
00208
00209         if (lignes[nb][longueur_ligne-1] == '\n')
00210             lignes[nb][longueur_ligne-1] = '\0';
00211
00212         longueur_ligne = strlen(lignes[nb]);
00213         if (longueur_ligne > max_len)
00214         {
00215             max_len = longueur_ligne;
00216         }
00217         nb++;
00218     }
00219     fclose(fichier);
00220
00221     hauteur = nb + 4;
00222     largeur = max_len + 4;
00223
00224     if(hauteur > LINES)
00225     {
00226         hauteur = LINES - 2 ;
00227     }
00228     if(largeur > COLS)
00229     {
00230         largeur = COLS - 2 ;
00231     }
00232
00233     y_centre = (LINES - hauteur) / 2;
00234     x_centre = (COLS - largeur) / 2;
00235
00236
00237 // Tableau ascii
00238 WINDOW *win = newwin(hauteur, largeur, y_centre, x_centre );
00239 wborder(win, '|', '|', '-', '-', '+', '+', '+', '+');
00240
00241 // Titre (seul élément centré)
00242 wattroff(win, COLOR_PAIR(5) | A_BOLD);
00243 mvwprintw(win, 1, (largeur - strlen(titre)) / 2, "%s", titre);
00244 wattroff(win, COLOR_PAIR(5) | A_BOLD);
00245
00246 // Texte NON CENTRÉ (respect exact des positions)
00247 for (int i = 0; i < nb; i++)
00248 {
00249     // Si c'est la dernière ligne
00250     if ( i == nb-1 )
00251     {
00252         // Dernière ligne clignotante + rouge
00253         wattroff(win, COLOR_PAIR(5) | A_BOLD | A_BLINK);
00254         mvwprintw(win, i+2 , 2 , "%s", lignes[i]);
00255         wattroff(win, COLOR_PAIR(5) | A_BOLD | A_BLINK);
00256     }
00257     else
00258     {
00259         // Colore les majuscules en vert
00260         if ( lignes[i][0] >= 'A' && lignes[i][0] <= 'Z' )
00261         {
00262             wattroff(win, COLOR_PAIR(4) | A_BOLD );
00263             mvwprintw(win, i+2 , 2 , "%s", lignes[i]);
00264             wattroff(win, COLOR_PAIR(4) | A_BOLD );
00265         }
00266         else
00267         {
00268             mvwprintw(win, i+2 , 2 , "%s", lignes[i]);
00269         }
00270     }
00271 }
00272
00273 // Rafraîchissement et pause
00274 refresh();
00275 wrefresh(win);
00276
00277 while( getch() != '\n' )
00278 {
00279     getch();
00280 }
00281
00282
00283 delwin(win);
00284 }
00285
00292 void affiche_fichier_tableau curse_position_exacte(const char *titre , const char *nom_fichier)
00293 {
00294     screen_clear();
00295
00296     initscr();           // initialise ncurses
00297     clear();              // ne pas afficher les touches tapées
00298

```

```

00299     cbreak();           // entrée sans mise en tampon
00300     curs_set(0);        // cache le curseur
00301
00302     start_color();      // active les couleurs
00303     use_default_colors(); // garde fond terminal
00304     init_pair(1, COLOR_YELLOW, -1); // titre
00305     init_pair(4, COLOR_CYAN, -1); // texte
00306     init_pair(5, COLOR_RED, -1); // message qui clignote
00307
00308     afficher_tableau_avec_titre_position_exacte(titre, nom_fichier);
00309
00310     endwin(); // restaure le terminal
00311     clear();
00312 }

```

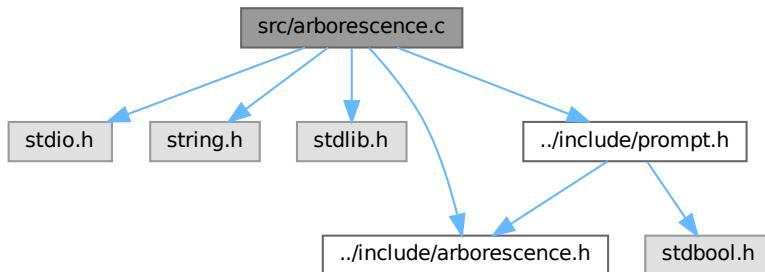
## 4.17 src/arborescence.c File Reference

fonction pour manipuler les arborescences

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "../include/arborescence.h"
#include "../include/prompt.h"
Include dependency graph for arborescence.c:

```



### Functions

- **fichier \* creer (fichier \*parent, char \*nom, int estDossier, int statut)**  
*Crée un nouveau fichier ou dossier.*
- **void ajouter\_enfant (fichier \*parent, char \*nom, int nombre, int statut)**  
*Ajoute un enfant (fichier ou dossier) à un dossier parent.*
- **void my\_touch (fichier \*racine, fichier \*courant, char \*nom, int statut)**  
*Crée un nouveau fichier dans le répertoire courant.*
- **void afficher (fichier \*home, int prfnd)**  
*Affiche la structure de l'arborescence des fichiers et dossiers.*
- **void my\_ls (fichier \*racine, fichier \*courant, char \*nom)**  
*Liste les fichiers et dossiers dans le répertoire courant.*
- **void my\_ls\_a (fichier \*racine, fichier \*courant, char \*nom)**  
*Liste tous les fichiers et dossiers, y compris les fichiers cachés, dans le répertoire courant.*
- **void my\_rmdir (fichier \*racine, fichier \*courant, char \*nom)**

- **fichier \* my\_cd (fichier \*racine, fichier \*courant, char \*nom)**  
*Supprime un dossier dans le répertoire courant.*
- **void my\_cp (fichier \*racine, fichier \*rep\_actuel, char \*nom, char \*destination)**  
*Change le répertoire courant.*
- **void my\_mv (fichier \*racine, fichier \*rep\_actuel, char \*nom, char \*destination)**  
*Copie un fichier ou dossier vers une destination.*
- **void my\_mv (fichier \*racine, fichier \*rep\_actuel, char \*nom, char \*destination)**  
*Déplace un fichier ou dossier vers une destination.*
- **void deplacer (fichier \*dest, fichier \*source)**  
*Déplace un fichier ou dossier d'une source vers une destination.*
- **fichier \* chercher (fichier \*courant, char \*nom)**  
*Cherche un fichier ou dossier par son nom dans le répertoire courant.*
- **void my\_rm (fichier \*racine, fichier \*courant, char \*nom)**  
*Supprime un fichier dans le répertoire courant.*
- **void my\_mkdir\_p (fichier \*racine, fichier \*parent, char \*nom)**  
*Crée un chemin de dossiers, créant les dossiers intermédiaires si nécessaire.*
- **void my\_mkdir (fichier \*racine, fichier \*courant, char \*nom, int statut)**  
*Crée un nouveau dossier dans le répertoire courant.*
- **void my\_pwd (fichier \*rep\_actuel)**  
*Affiche le chemin complet du répertoire courant.*
- **void my\_echo (char \*\*ligne\_c)**  
*Affiche une ligne de texte.*
- **void my\_chmod (fichier \*racine, fichier \*courant, char \*option, char \*nom)**  
*Modifie les permissions d'un fichier ou dossier.*
- **void my\_ls\_l (fichier \*racine, fichier \*courant, char \*nom)**  
*Liste les fichiers et dossiers avec des détails dans le répertoire courant.*
- **void afficher\_nbr\_ouverture (fichier \*home, int prfnd)**  
*Affiche le nombre d'ouvertures pour chaque fichier dans l'arborescence.*
- **int recherche\_max (fichier \*home, int max)**  
*Recherche le nombre maximum d'ouvertures parmi les fichiers dans l'arborescence.*
- **void favori (fichier \*home, int max)**  
*Affiche les fichiers ayant le nombre maximum d'ouvertures.*
- **void affichage (fichier \*home)**  
*Affiche les fichiers les plus ouverts dans l'arborescence.*
- **void my\_ls\_la (fichier \*racine, fichier \*courant, char \*nom)**  
*Liste tous les fichiers et dossiers avec des détails, y compris les fichiers cachés, dans le répertoire courant.*

### 4.17.1 Detailed Description

fonction pour manipuler les arborescences

Definition in file [arborescence.c](#).

### 4.17.2 Function Documentation

#### 4.17.2.1 affichage()

```
void affichage (
    fichier * home )
```

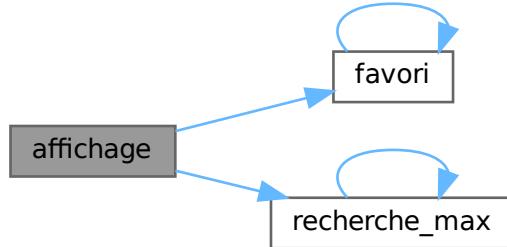
Affiche les fichiers les plus ouverts dans l'arborescence.

**Parameters**

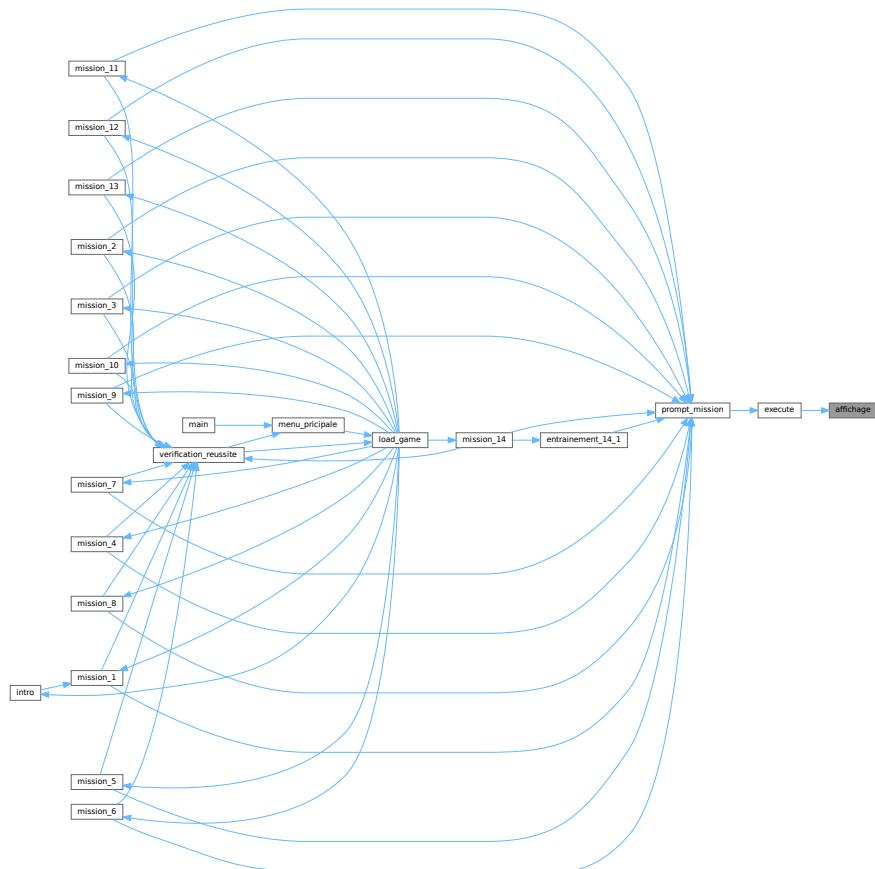
<code>home</code>	Pointeur vers le répertoire racine.
-------------------	-------------------------------------

Definition at line 1530 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.17.2.2 afficher()

```
void afficher (
    fichier * home,
    int prfnd )
```

Affiche la structure de l'arborescence des fichiers et dossiers.

#### Parameters

<i>home</i>	Pointeur vers le répertoire racine.
<i>prfnd</i>	Profondeur actuelle dans l'arborescence.

Definition at line 170 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.17.2.3 afficher\_nbr\_ouverture()

```
void afficher_nbr_ouverture (
    fichier * home,
    int prfnd )
```

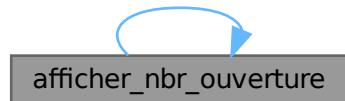
Affiche le nombre d'ouvertures pour chaque fichier dans l'arborescence.

**Parameters**

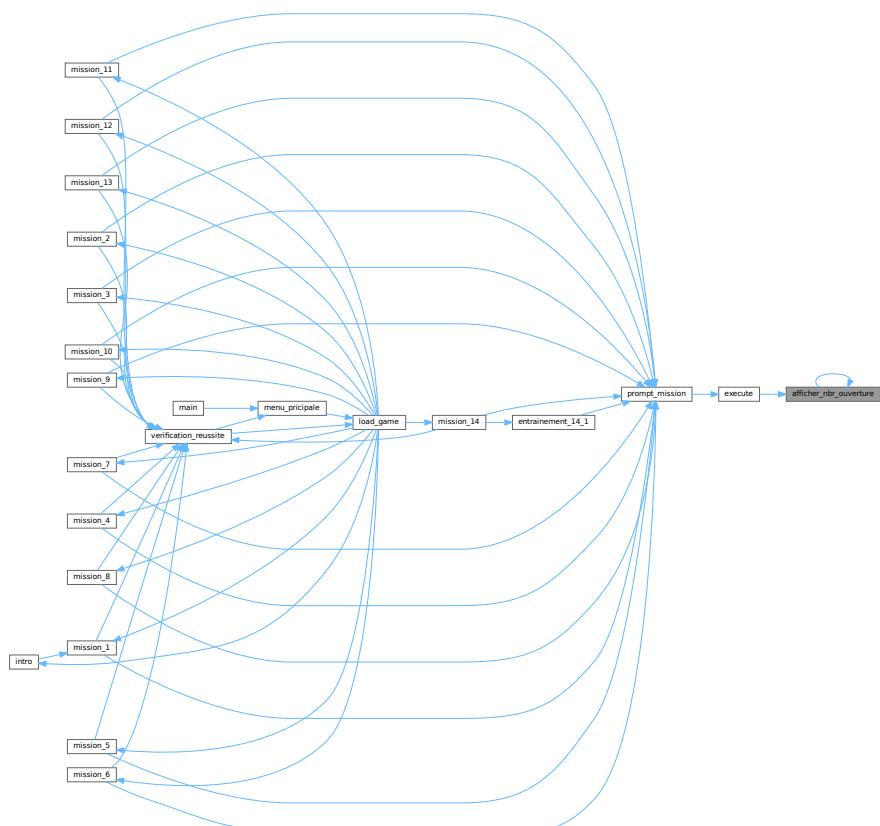
<i>home</i>	Pointeur vers le répertoire racine.
<i>prfnd</i>	Profondeur actuelle dans l'arborescence.

Definition at line 1486 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.4 ajouter\_enfant()

```
void ajouter_enfant (
    fichier * parent,
    char * nom,
    int nombre,
    int statut )
```

Ajoute un enfant (fichier ou dossier) à un dossier parent.

##### Parameters

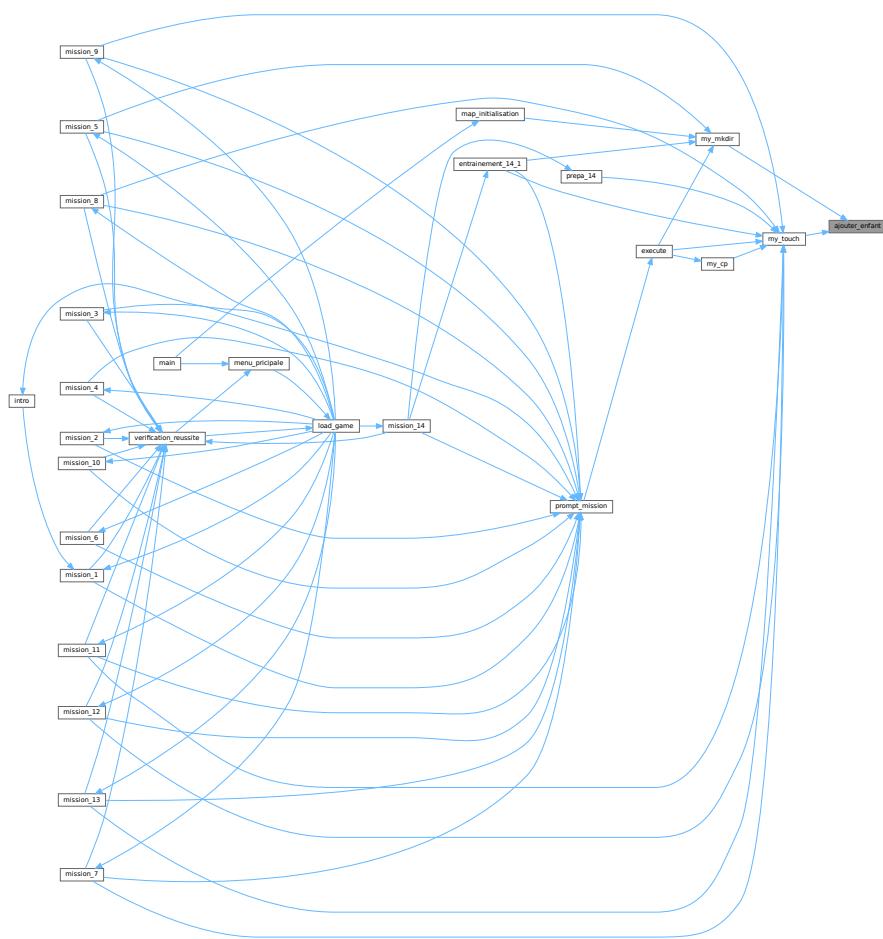
<i>parent</i>	Pointeur vers le dossier parent.
<i>nom</i>	Nom de l'enfant à ajouter.
<i>nombre</i>	Indicateur si c'est un dossier (1) ou un fichier (0).
<i>statut</i>	Statut de protection de l'enfant.

Definition at line 40 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.5 chercher()

```
fichier * chercher (
    fichier * courant,
    char * nom )
```

Cherche un fichier ou dossier par son nom dans le répertoire courant.

##### Parameters

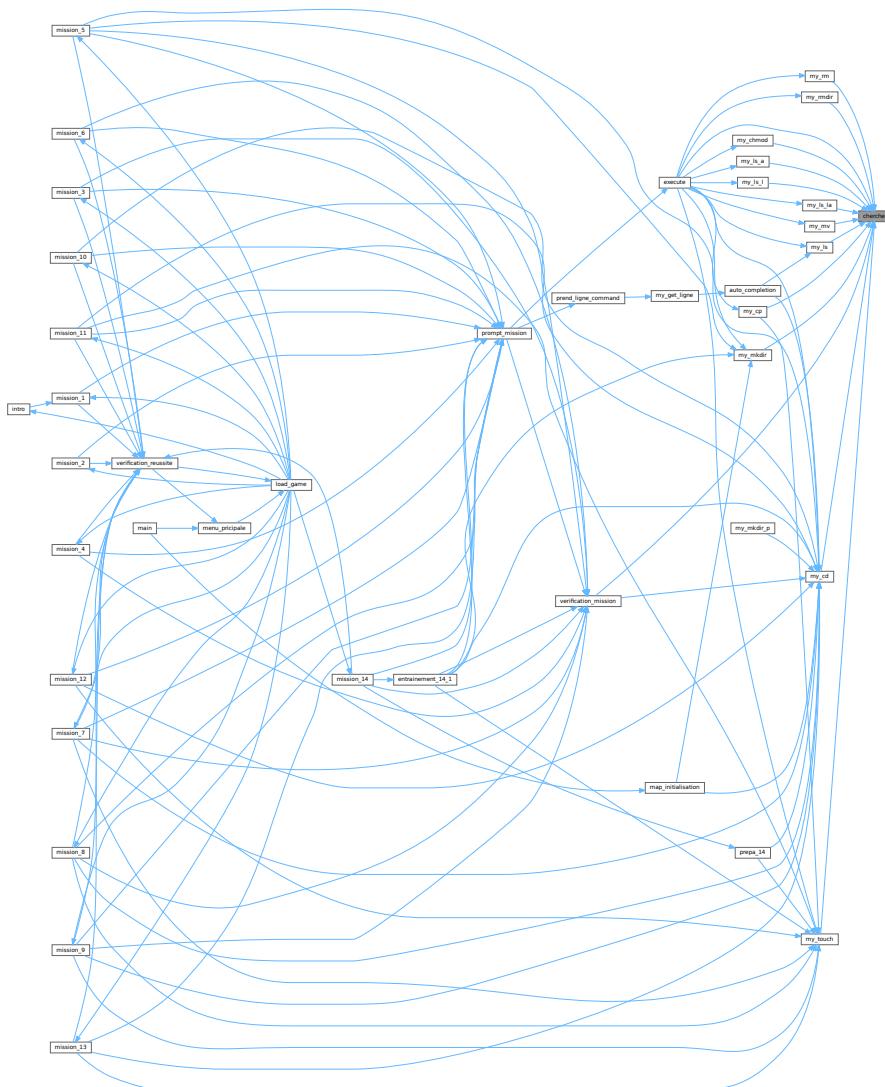
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du fichier ou dossier à chercher.

##### Returns

Pointeur vers le fichier ou dossier trouvé, ou NULL s'il n'existe pas.

Definition at line 984 of file [arborescence.c](#).

Here is the caller graph for this function:



#### 4.17.2.6 creer()

```
fichier * creer (
    fichier * parent,
    char * nom,
    int estDossier,
    int statut )
```

Crée un nouveau fichier ou dossier.

## Parameters

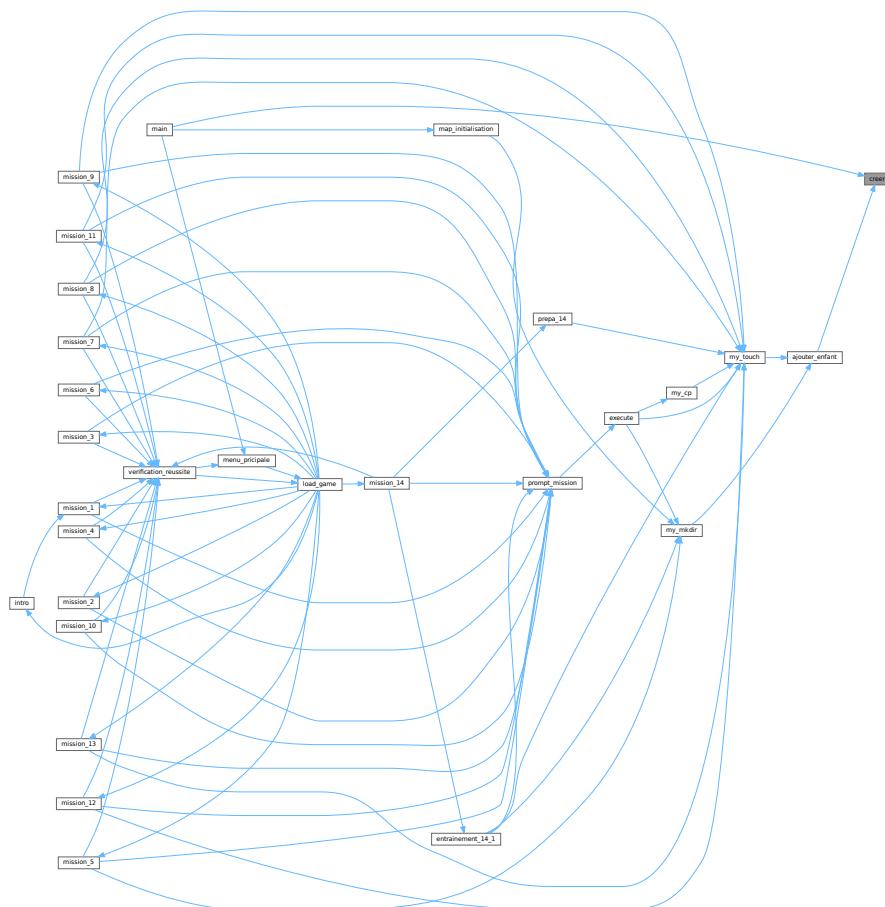
<i>parent</i>	Pointeur vers le dossier parent.
<i>nom</i>	Nom du fichier ou dossier à créer.
<i>estDossier</i>	Indicateur si c'est un dossier (1) ou un fichier (0).
<i>statut</i>	Statut de protection du fichier ou dossier.

**Returns**

Pointeur vers le nouveau fichier ou dossier créé.

Definition at line 19 of file [arborescence.c](#).

Here is the caller graph for this function:

**4.17.2.7 deplacer()**

```
void deplacer (
    fichier * dest,
    fichier * source )
```

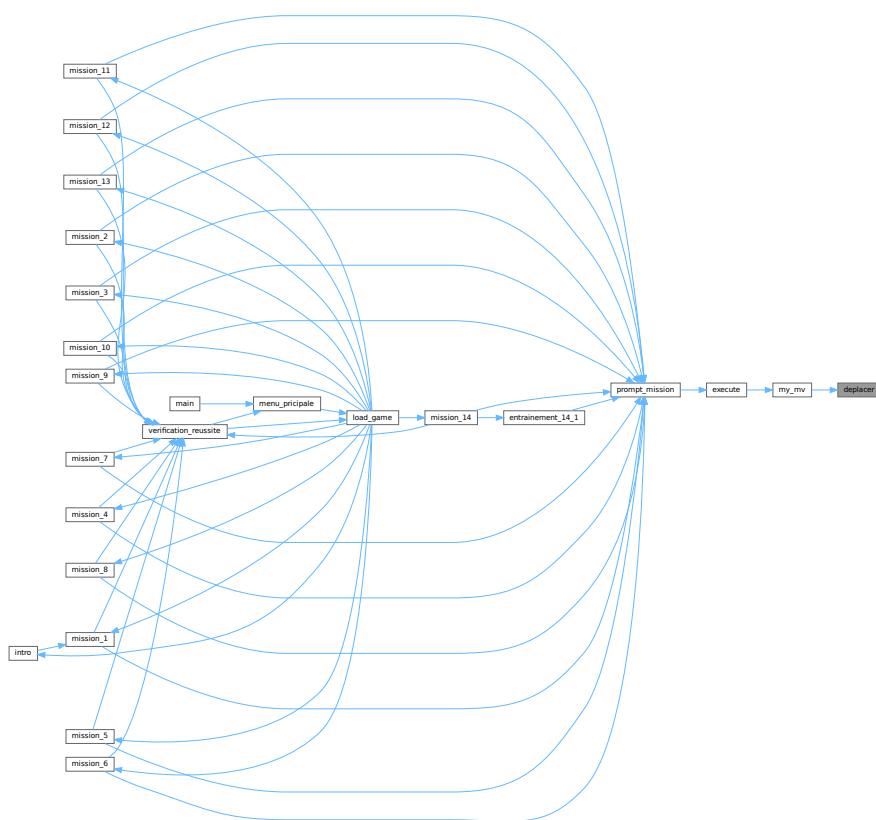
Déplace un fichier ou dossier d'une source vers une destination.

**Parameters**

<i>dest</i>	Pointeur vers le dossier de destination.
<i>source</i>	Pointeur vers le fichier ou dossier source à déplacer.

Definition at line 930 of file [arborescence.c](#).

Here is the caller graph for this function:



#### 4.17.2.8 favori()

```
void favori (
    fichier * home,
    int max )
```

Affiche les fichiers ayant le nombre maximum d'ouvertures.

##### Parameters

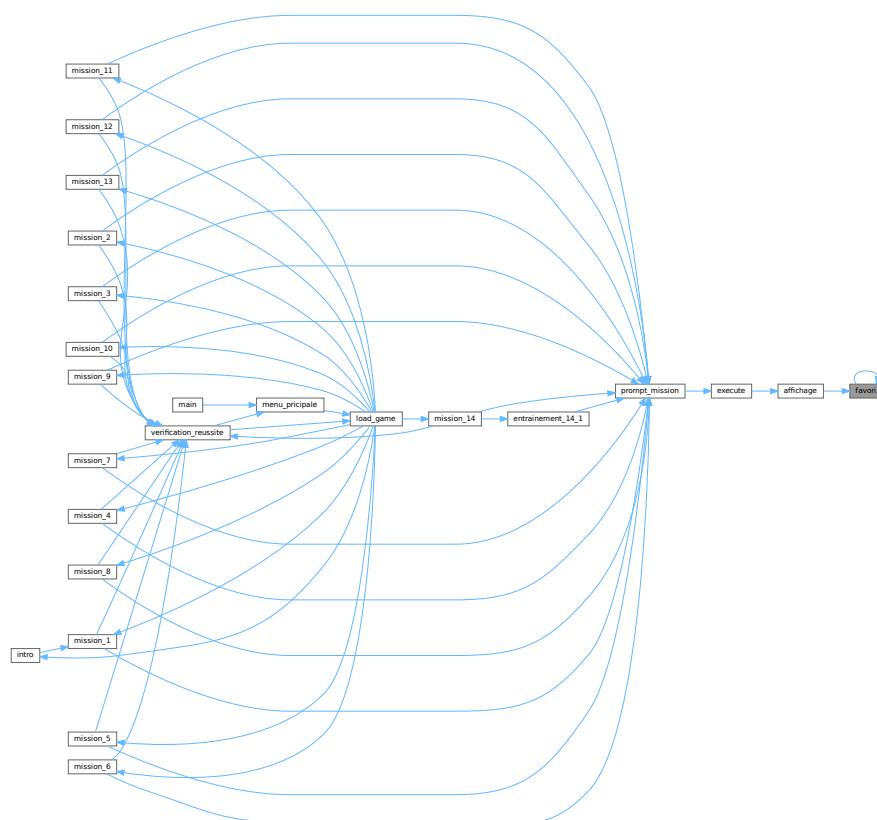
<i>home</i>	Pointeur vers le répertoire racine.
<i>max</i>	Nombre maximum d'ouvertures.

Definition at line 1516 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.9 my\_cd()

```
fichier * my_cd (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Change le répertoire courant.

**Parameters**

<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du répertoire cible.

**Returns**

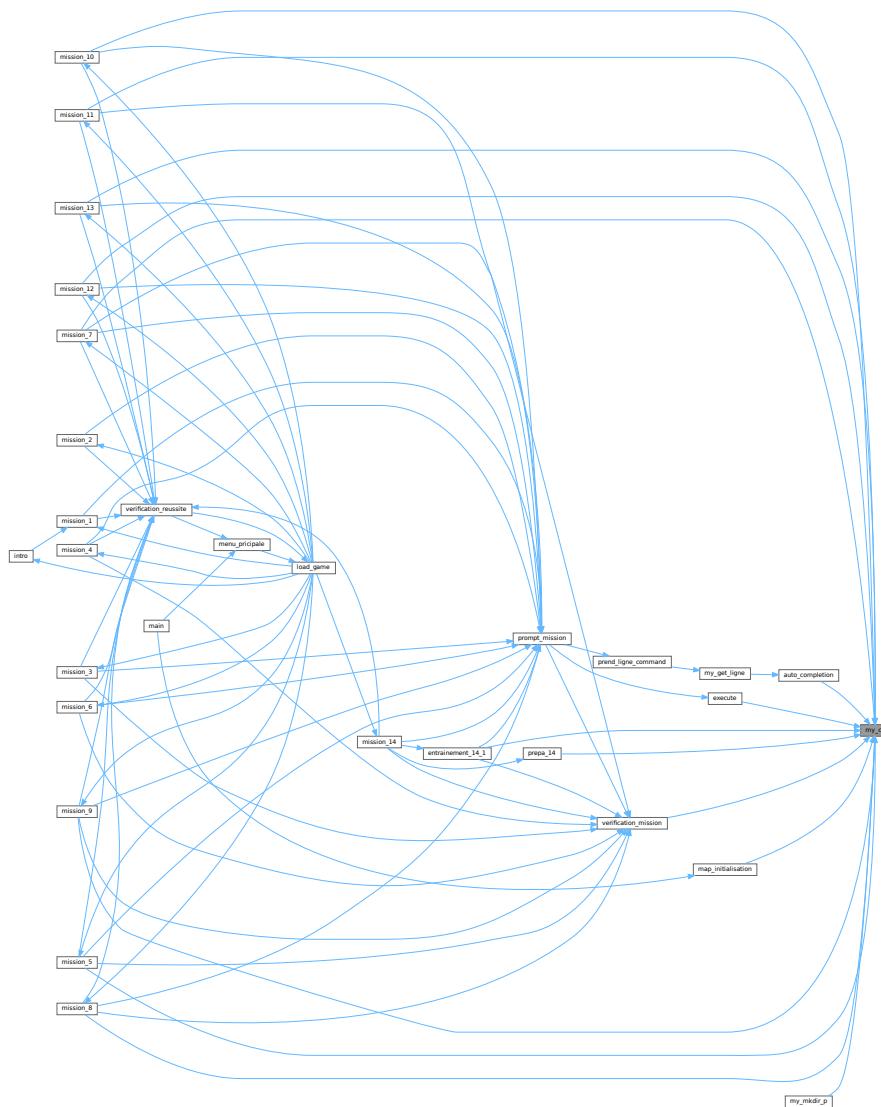
Pointeur vers le nouveau répertoire courant.

Definition at line [522](#) of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.10 my\_chmod()

```
void my_chmod (
    fichier * racine,
    fichier * courant,
    char * option,
    char * nom )
```

Modifie les permissions d'un fichier ou dossier.

##### Parameters

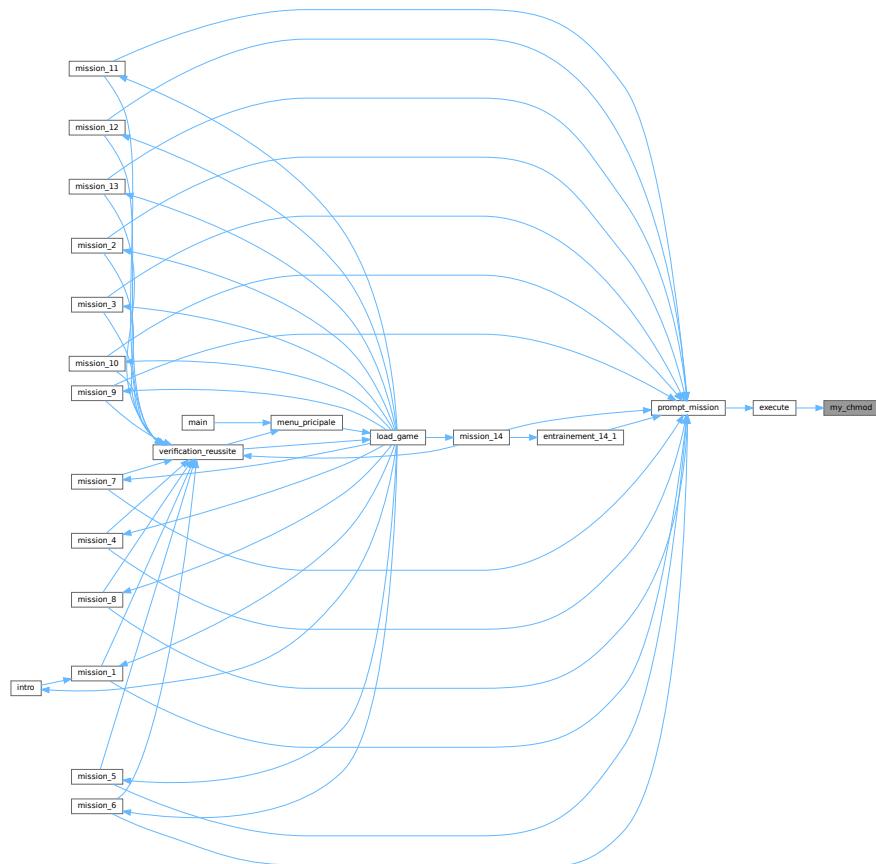
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>option</i>	Chaîne de caractères représentant les options de permission.
<i>nom</i>	Nom du fichier ou dossier dont les permissions doivent être modifiées.

Definition at line 1259 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.11 `my_cp()`

```
void my_cp (
    fichier * racine,
    fichier * rep_actuel,
    char * nom,
    char * destination )
```

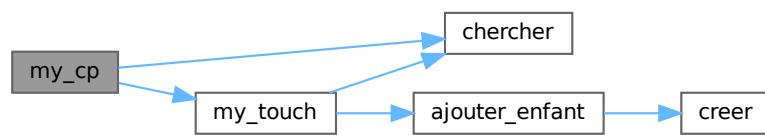
Copie un fichier ou dossier vers une destination.

### Parameters

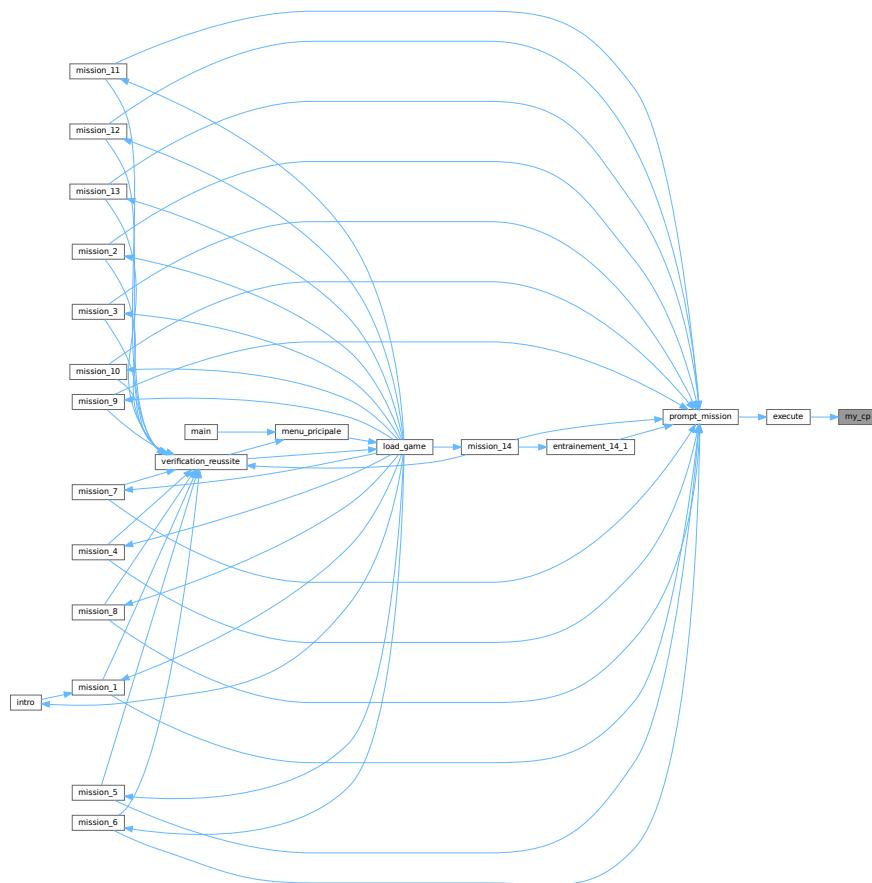
<i>racine</i>	Pointeur vers le répertoire racine.
<i>parent</i>	Pointeur vers le répertoire parent du fichier ou dossier à copier.
<i>nom</i>	Nom du fichier ou dossier à copier.
<i>destination</i>	Chemin de destination pour la copie.

Definition at line 611 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.12 my\_echo()

```
void my_echo (
    char ** ligne_c )
```

Affiche une ligne de texte.

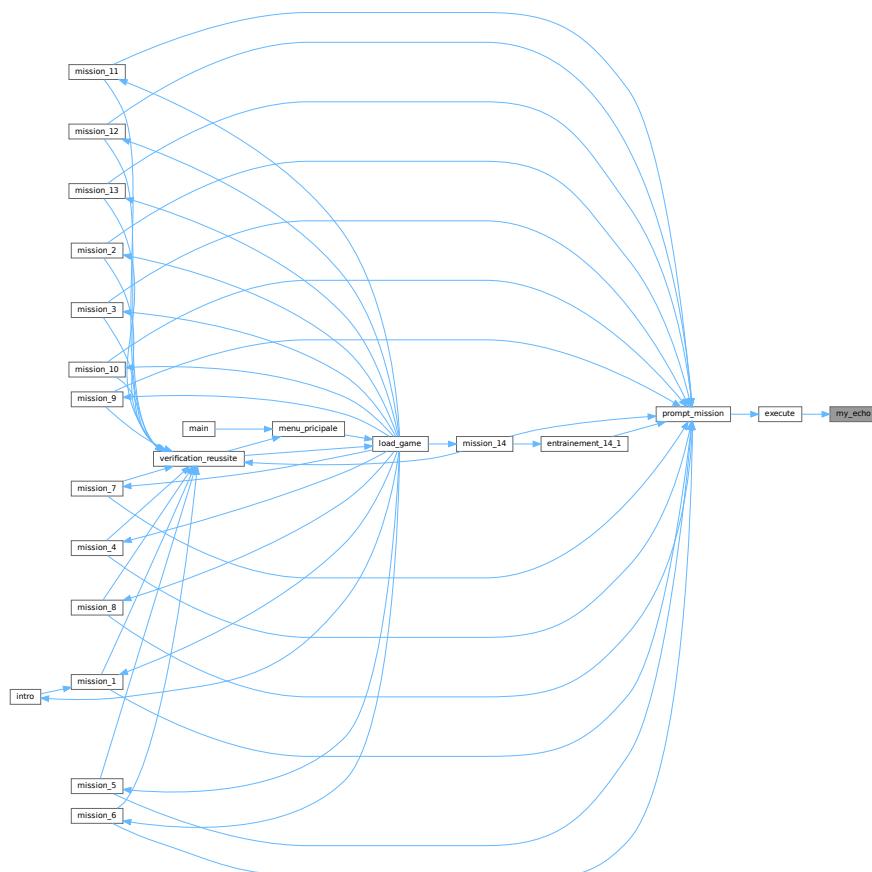
c'est comme la commande 'echo' mais en developpement

##### Parameters

<i>ligne_c</i>	Tableau de chaînes de caractères à afficher.
----------------	----------------------------------------------

Definition at line 1245 of file [arborescence.c](#).

Here is the caller graph for this function:



#### 4.17.2.13 my\_ls()

```
void my_ls (
    fichier * racine,
```

```
fichier * courant,  
char * nom )
```

Liste les fichiers et dossiers dans le répertoire courant.

## Parameters

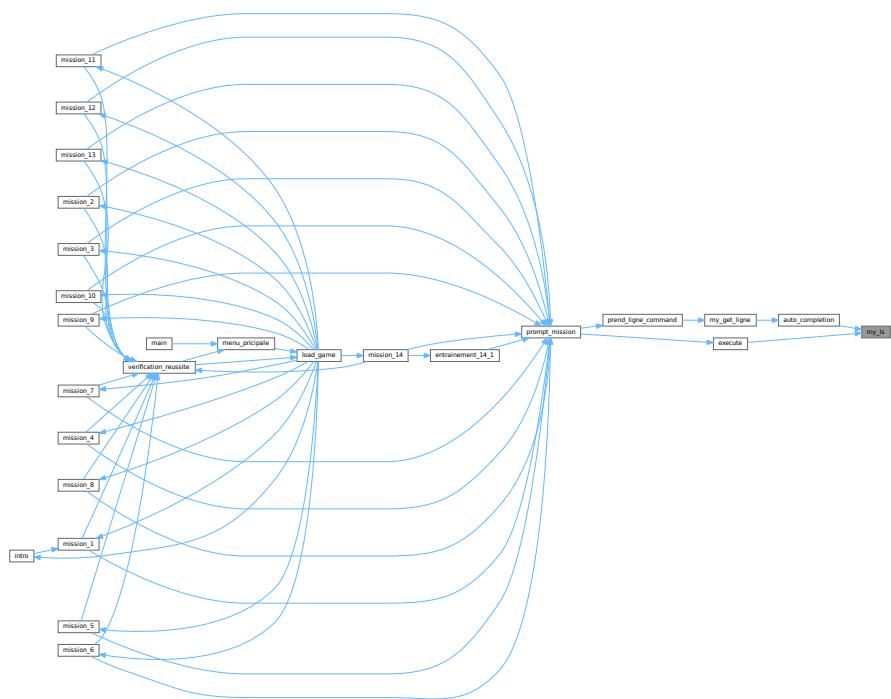
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du répertoire à lister.

Definition at line 192 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.14 my\_ls\_a()

```
void my_ls_a (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Liste tous les fichiers et dossiers, y compris les fichiers cachés, dans le répertoire courant.

### Parameters

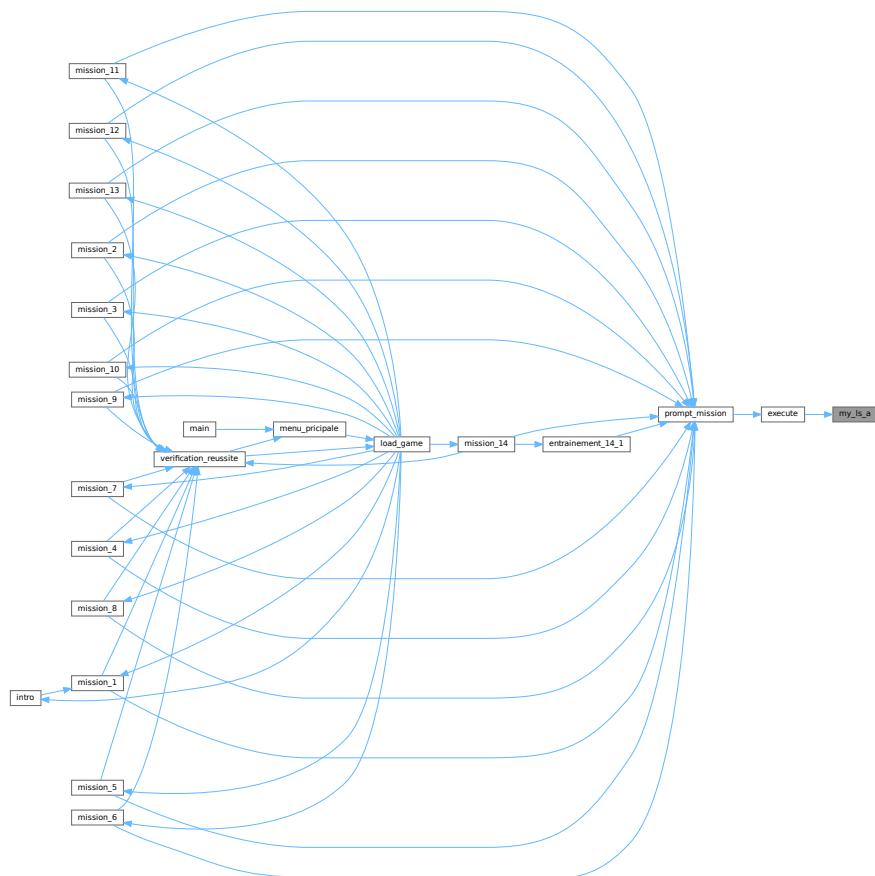
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du répertoire à lister.

Definition at line 296 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.15 my\_ls\_l()

```
void my_ls_l (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Liste les fichiers et dossiers avec des détails dans le répertoire courant.

##### Parameters

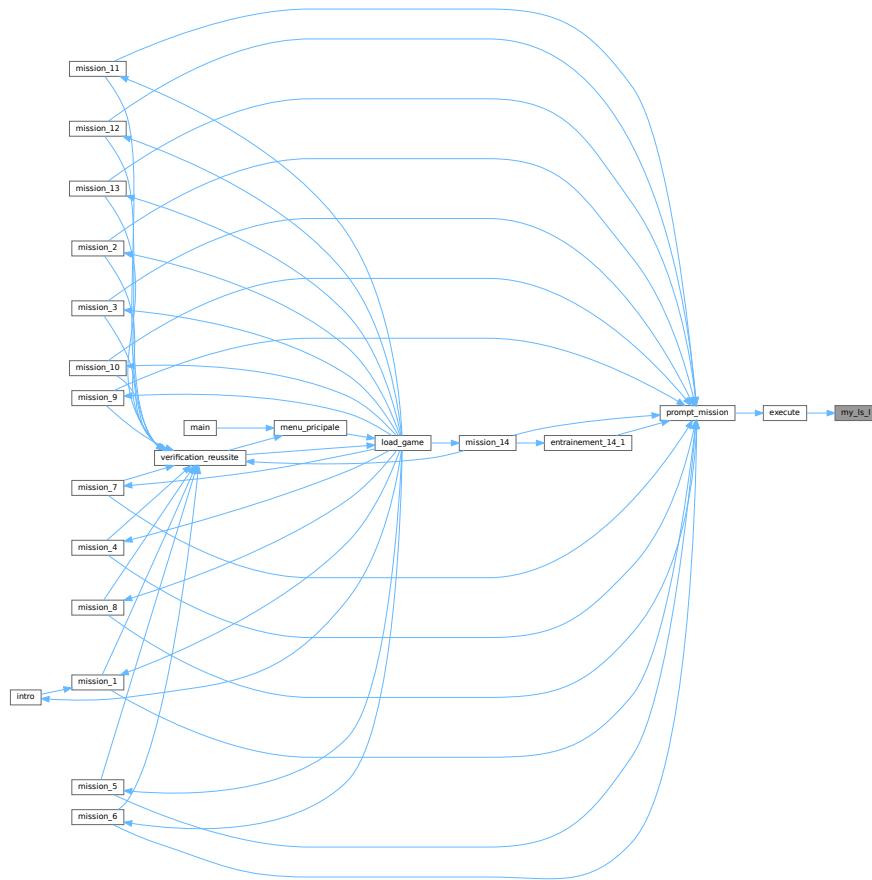
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du répertoire à lister.

Definition at line 1387 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.16 my\_ls\_la()

```
void my_ls_la (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Liste tous les fichiers et dossiers avec des détails, y compris les fichiers cachés, dans le répertoire courant.

##### Parameters

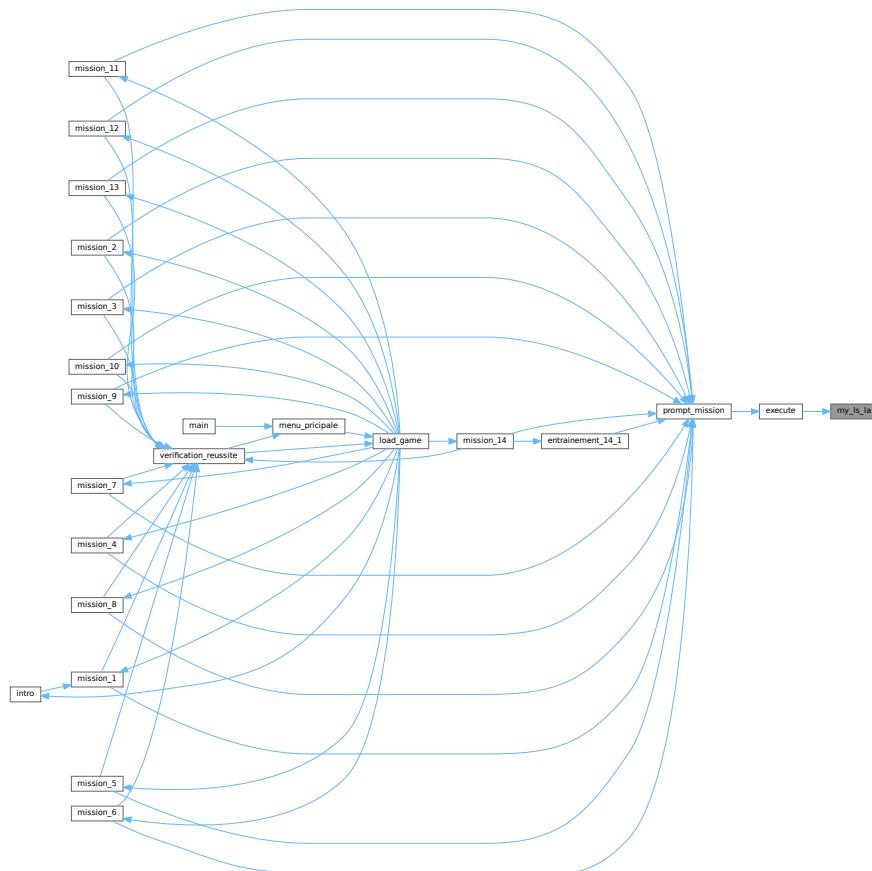
<code>racine</code>	Pointeur vers le répertoire racine.
<code>courant</code>	Pointeur vers le répertoire courant.
<code>nom</code>	Nom du répertoire à lister.

Definition at line 1537 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.17 my\_mkdir()

```
void my_mkdir (
    fichier * racine,
    fichier * courant,
    char * nom,
    int statut )
```

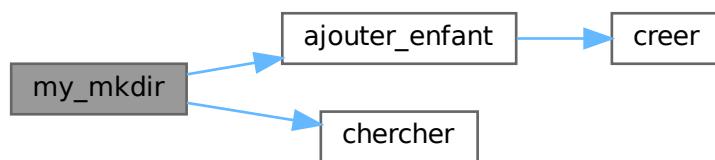
Crée un nouveau dossier dans le répertoire courant.

**Parameters**

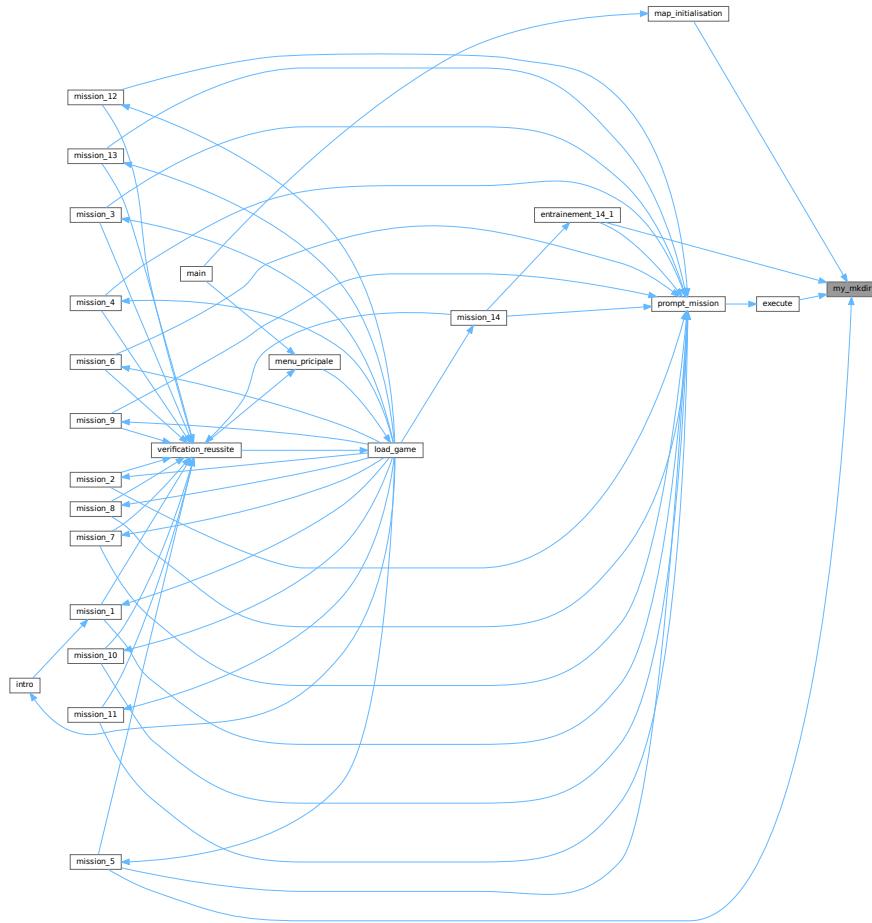
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du dossier à créer.
<i>statut</i>	Statut de protection du dossier.

Definition at line 1143 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.18 my\_mkdir\_p()

```
void my_mkdir_p (
    fichier * racine,
    fichier * parent,
    char * nom )
```

Crée un chemin de dossiers, créant les dossiers intermédiaires si nécessaire.

##### Parameters

<i>racine</i>	Pointeur vers le répertoire racine.
<i>parent</i>	Pointeur vers le répertoire parent.
<i>nom</i>	Chemin du dossier à créer.

Definition at line 1121 of file [arborescence.c](#).

Here is the call graph for this function:



#### 4.17.2.19 my\_mv()

```
void my_mv (
    fichier * racine,
    fichier * rep_actuel,
    char * nom,
    char * destination )
```

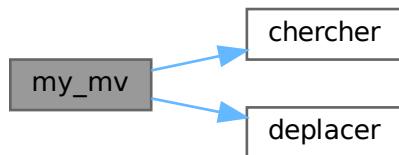
Déplace un fichier ou dossier vers une destination.

##### Parameters

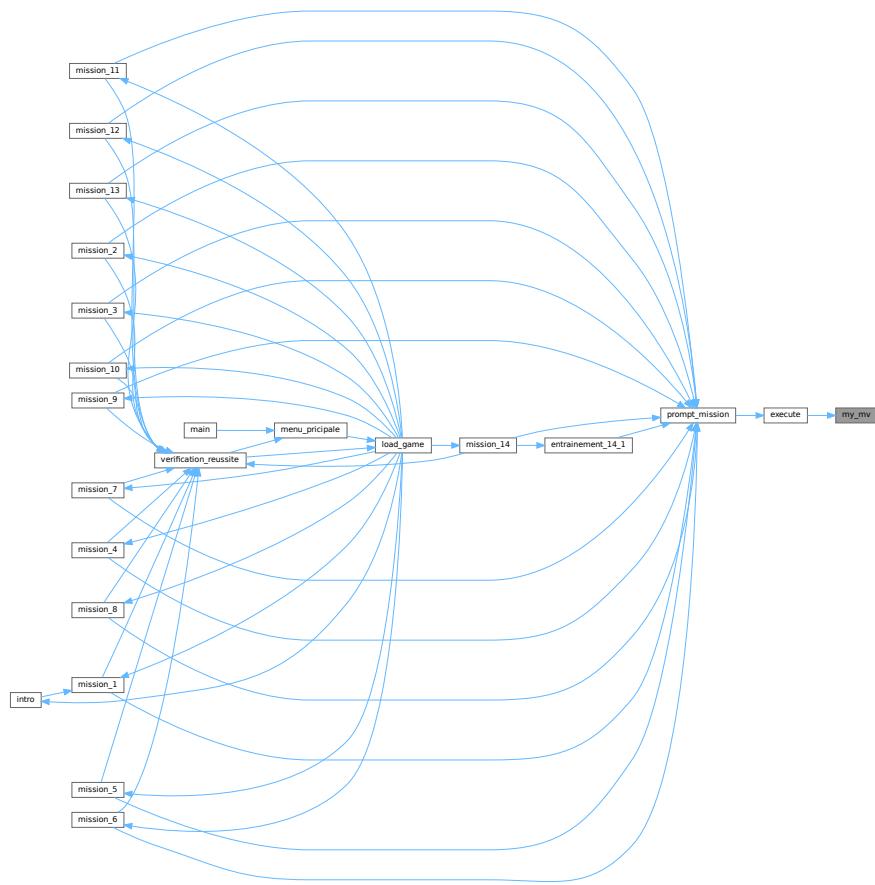
<i>racine</i>	Pointeur vers le répertoire racine.
<i>parent</i>	Pointeur vers le répertoire parent du fichier ou dossier à déplacer.
<i>nom</i>	Nom du fichier ou dossier à déplacer.
<i>destination</i>	Chemin de destination pour le déplacement.

Definition at line 751 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.20 `my_pwd()`

```
void my_pwd (
    fichier * rep_actuel )
```

Affiche le chemin complet du répertoire courant.

##### Parameters

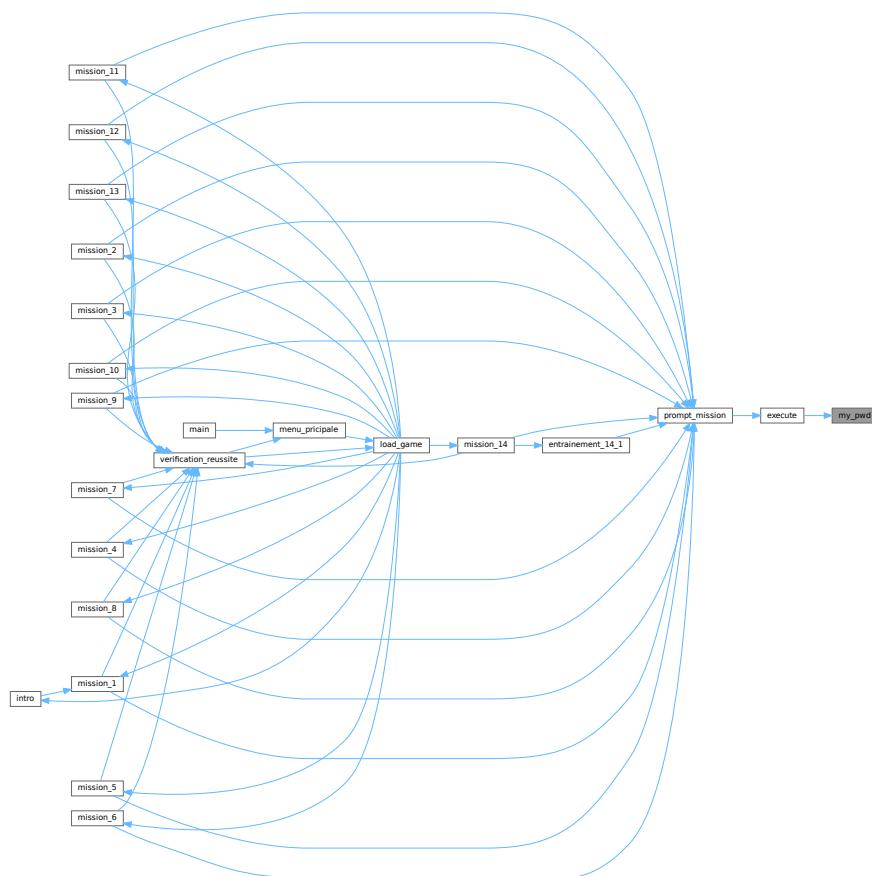
<code>rep_actuel</code>	Pointeur vers le répertoire courant.
-------------------------	--------------------------------------

Definition at line 1231 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.21 my\_rm()

```
void my_rm (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Supprime un fichier dans le répertoire courant.

**Parameters**

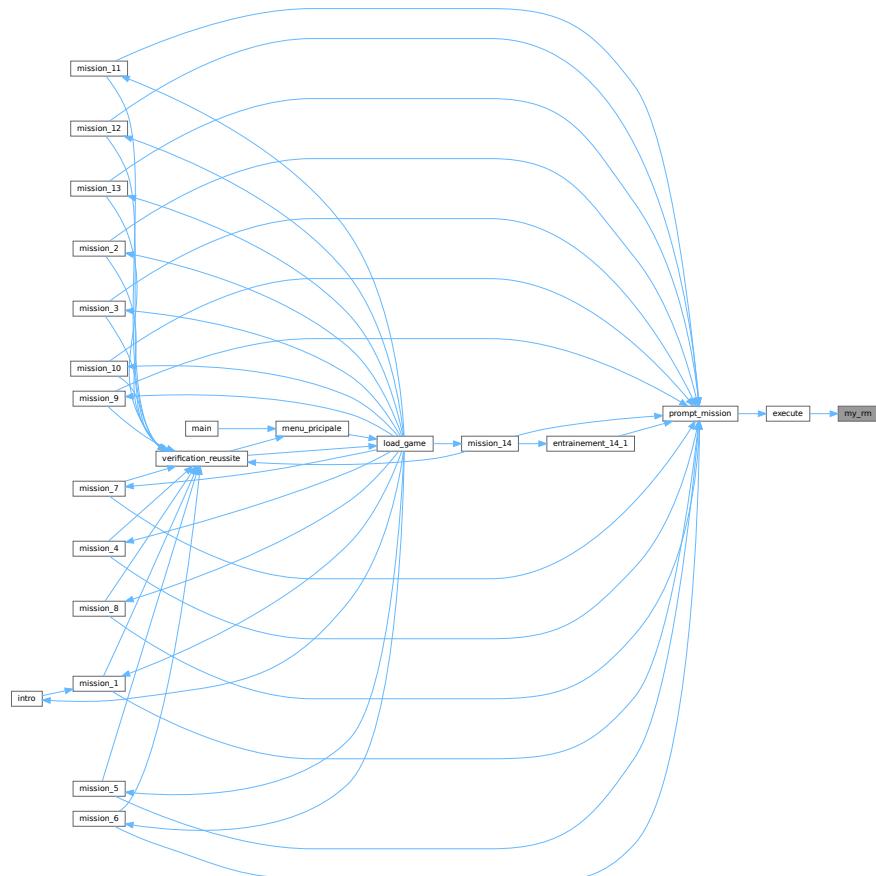
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du fichier à supprimer.

Definition at line 1006 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.22 my\_rmdir()

```
void my_rmdir (
    fichier * racine,
    fichier * courant,
    char * nom )
```

Supprime un dossier dans le répertoire courant.

##### Parameters

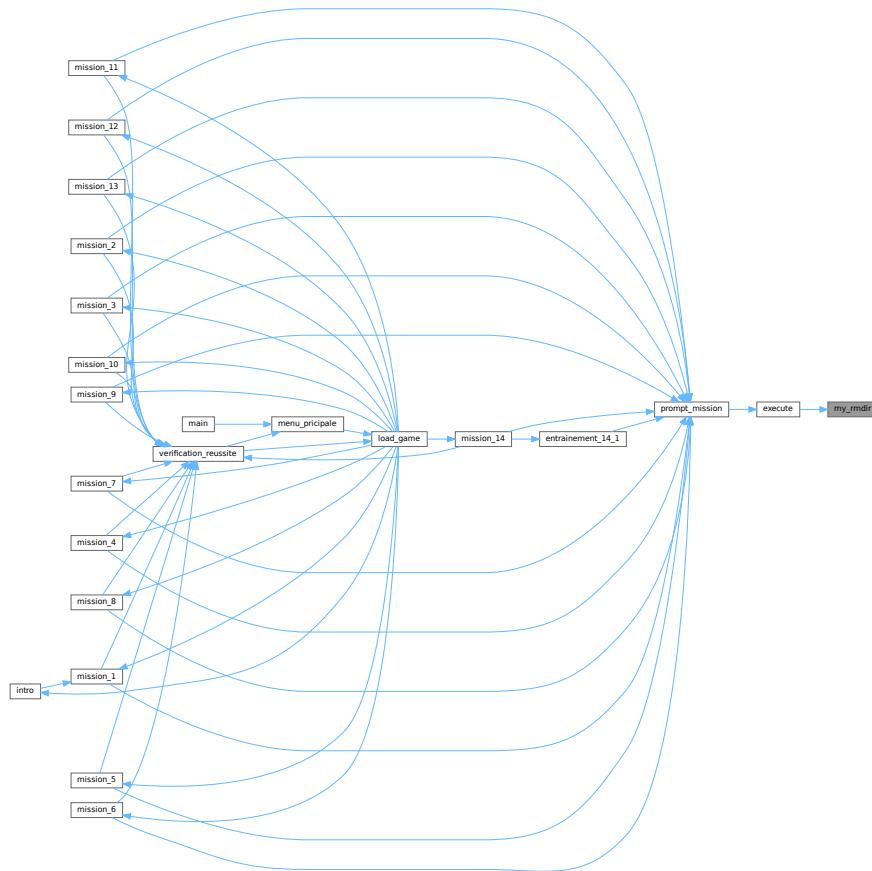
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du dossier à supprimer.

Definition at line 394 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.23 my\_touch()

```
void my_touch (
    fichier * racine,
    fichier * courant,
    char * nom,
    int statut )
```

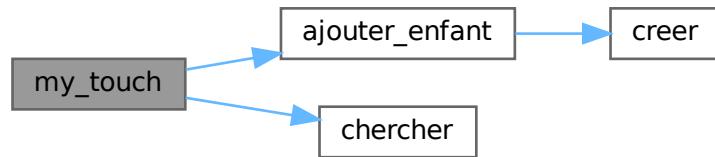
Crée un nouveau fichier dans le répertoire courant.

##### Parameters

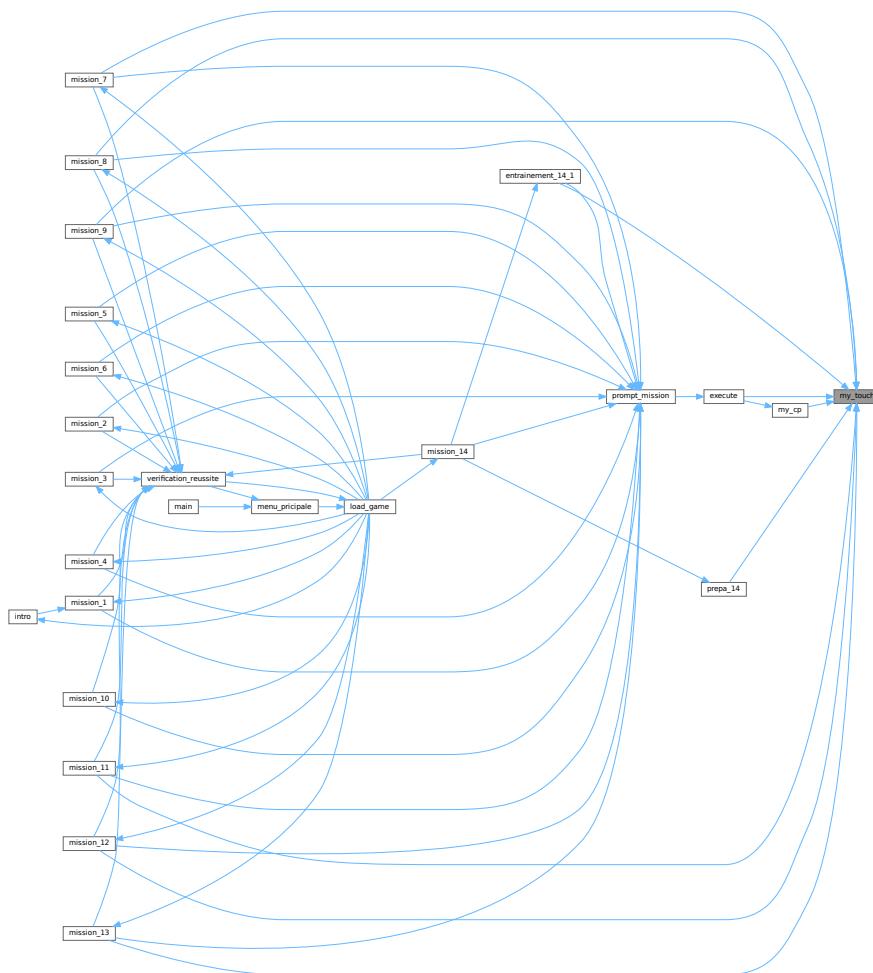
<i>racine</i>	Pointeur vers le répertoire racine.
<i>courant</i>	Pointeur vers le répertoire courant.
<i>nom</i>	Nom du fichier à créer.
<i>statut</i>	Statut de protection du fichier.

Definition at line 71 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.2.24 recherche\_max()

```
int recherche_max (
    fichier * home,
    int max )
```

Recherche le nombre maximum d'ouvertures parmi les fichiers dans l'arborescence.

#### Parameters

<i>home</i>	Pointeur vers le répertoire racine.
<i>max</i>	Nombre maximum d'ouvertures trouvé jusqu'à présent.

#### Returns

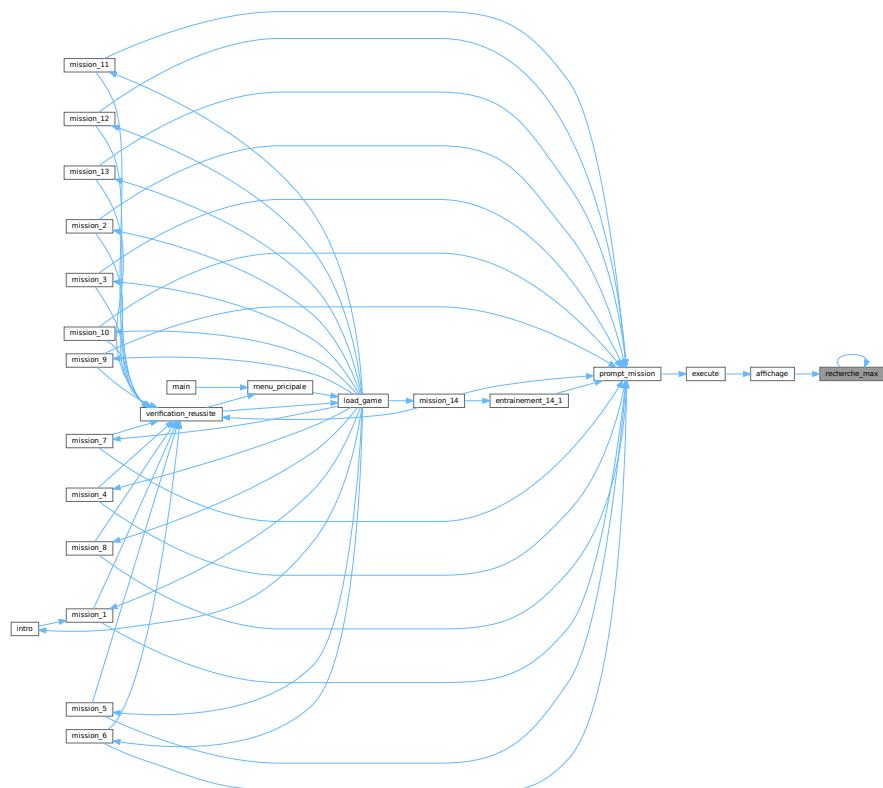
Nombre maximum d'ouvertures.

Definition at line 1501 of file [arborescence.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.18 arborescence.c

[Go to the documentation of this file.](#)

```

00001
00005 #include<stdio.h>
00006 #include<string.h>
00007 #include<stdlib.h>
00008 #include "../include/arborescence.h"
00009 #include "../include/prompt.h"
00010
00011
00019 fichier* creer(fichier *parent, char *nom, int estDossier, int statut)
00020 {
00021     fichier* nouveau = (fichier*)malloc(sizeof(fichier)) ;
00022     strcpy(nouveau->nom, nom) ;
00023     nouveau->estDossier = estDossier ;
00024     nouveau->parent = parent ;
00025     nouveau->estProtege = statut ;
00026     strcpy(nouveau->perm, "rwx") ;
00027     nouveau->premierfils = NULL ;
00028     nouveau->frereSuivant = NULL ;
00029     nouveau->nombre_ouverture = 0 ;
00030     return nouveau ;
00031 }
00032
00033
00040 void ajouter_enfant(fichier *parent, char *nom, int nombre, int statut)
00041 {
00042     if(!parent->estDossier)      // On ne peut rien y ajouter si il n'est pas un dossier
00043     {
00044         printf("Erreur: %s n'est pas un dossier.\n", parent->nom) ;
00045         return ;
00046     }
00047
00048     fichier *enfant = creer(parent, nom, nombre,statut) ;
00049     if(parent->premierfils == NULL)    //Si il n'y pas déjà un premier fils dans le dossier
00050     {
00051         parent->premierfils = enfant ;
00052     }
00053     else
00054     {
00055         fichier *temp = parent->premierfils ;
00056         while(temp->frereSuivant != NULL)      //Parcourir dans les frères suivants si il y a déjà des
00057             contenus dans le dossier
00058         {
00059             temp = temp->frereSuivant ;
00060         }
00061         temp->frereSuivant = enfant ;
00062     }
00063
00064
00071 void my_touch(fichier *racine, fichier *courant, char *nom,int statut)
00072 {
00073     char *dir , *dernier ;
00074     char *mot ;
00075     mot = malloc(strlen(nom) *sizeof(char)) ;
00076     int compt = 0 ;
00077     fichier* temoin ;
00078     fichier *parent ;
00079     strcpy(mot, nom) ;
00080     dir = strtok(mot, "/") ;    //Vérifier si le nom de dossier est un répertoire absolu
(exemple: ../../.../....) ou un seul mot
00081     while(dir != NULL)      //Parcourir le répertoire absolu
00082     {
00083         compt++ ;
00084         dernier = dir ;
00085         dir = strtok(NULL, "/") ;
00086     }
00087     if (compt == 1)    //Si on ne voit pas un répertoire absolu mais un seul mot
00088     {
00089         temoin = chercher(courant, dernier) ;    //Vérifier si existe déjà
00090         if(temoin == NULL)
00091         {
00092             if(courant->perm[1] != 'w' || courant->perm[2] != 'x')
00093             {
00094                 printf("Permission non accordée\n") ;
00095                 return ;
00096             }
00097             ajouter_enfant(courant, dernier, 0, statut) ;
00098         }
00099         else
00100         {
00101             printf("Erreur: le fichier existe déjà\n") ;
00102         }

```

```

00103         return ;
00104     }
00105     if (nom[0] == '/')      //Si le répertoire absolu se commence par un "/" , alors on va parcourir
00106     depuis la racine
00107     {
00108         parent = racine;
00109     }  

00110     else    //Sinon parcourir depuis le répertoire courant
00111     {
00112         parent = courant ;
00113     }  

00114     if (compt > 1)      //Si le nom de dossier est un répertoire absolu (exemple:/.../.../....)
00115     {
00116         dir = strtok(nom, "/") ;
00117         if(strcmp(dir,"..") == 0)
00118         {
00119             strcpy(dir,courant->parent->nom) ;
00120         }  

00121         if(strcmp(dir,".") == 0)
00122         {
00123             strcpy(dir,courant->nom) ;
00124         }  

00125         while(dir != NULL)      //Parcourir la chaîne
00126         {
00127             if (strcmp(dir, dernier) == 0)
00128             {
00129                 break ;
00130             }  

00131             if(dir != dernier)
00132             {
00133                 parent = chercher(parent, dir) ;    //Chercher les dossiers un par un si ils existent
00134             }  

00135         }  

00136         if(parent == NULL)      //Si on ne trouve pas, on envoie un message d'erreur: destination
00137         introuvable
00138         {
00139             printf("Erreur: %s : aucun dossier de ce nom\n", dir) ;
00140             return ;
00141         }
00142         dir = strtok(NULL, "/") ;
00143     }
00144 }
00145
00146 temoin = chercher(parent, dernier) ;    //Vérifier si existe déjà
00147 if(temoin == NULL)
00148 {
00149     if(parent->perm[1] != 'w' || parent->perm[2] != 'x')
00150     {
00151         printf("Permission non accordée\n") ;
00152         return ;
00153     }
00154     ajouter_enfant(parent, dernier, 0,statut) ;
00155 }
00156 else
00157 {
00158     printf("Erreur: le fichier existe déjà\n") ;
00159 }
00160
00161 }
00162
00163
00170 void afficher(fichier* home, int prfnd)
00171 {
00172     if(home == NULL)
00173     {
00174         return ;
00175     }
00176     for(int i = 0; i<prfnd; i++)
00177     {
00178         printf(" ") ;
00179     }
00180     printf("%s%s\n", home->nom, home->estDossier ? "/" : "") ;
00181     afficher(home->premierfils, prfnd + 1) ;    //On appelle récursivement la fonction lui-même
00182     afficher(home->frereSuivant, prfnd) ;
00183 }
00184
00185
00186
00192 void my_ls(fichier* racine, fichier *courant,char *nom)
00193 {
00194     fichier *repertoire ;
00195     if(nom[0]=='\0')
00196     {
00197         repertoire = courant ;
00198     }

```

```

00199     else if(strcmp(nom, "/") == 0)
00200     {
00201         repertoire = racine ;
00202     }
00203     else
00204     {
00205         if (strcmp(nom, "..") == 0)           //Pour remonter dans son parent
00206         {
00207             if (courant->parent)
00208             {
00209                 repertoire = courant->parent ; //dans son parent
00210             }
00211             else
00212             {
00213                 repertoire = courant; // reste à la racine
00214             }
00215         }
00216
00217         else if (strcmp(nom, ".") == 0)      //Pour ne pas faire un déplacement
00218         {
00219             repertoire = courant ;
00220         }
00221         else
00222         {
00223             char *dir ;
00224             fichier *tmp , *dernier;
00225             if (nom[0] == '/')           //Si le nom se commence par un "/", donc parcourir depuis la
racine
00226             {
00227                 tmp = racine;
00228             }
00229             else    //Sinon, parcourir depuis le répertoire courant
00230             {
00231                 tmp = courant ;
00232             }
00233             dir = strtok(nom,"/") ;      //Si le destination se fait dans un répertoire absolu
00234             if(strcmp(dir,".") == 0)
00235             {
00236                 dir = strtok(NULL,"/") ;
00237             }
00238             if(strcmp(dir,"..") == 0)
00239             {
00240                 if(courant->parent != NULL)
00241                 {
00242                     tmp = courant->parent ;
00243                 }
00244                 else
00245                 {
00246                     tmp = courant ;
00247                 }
00248                 dir = strtok(NULL,"/") ;
00249             }
00250
00251             while(dir != NULL)        //Parcourir la chaîne
00252             {
00253                 tmp = chercher(tmp,dir) ; //Vérifier un par un si les dossiers existent
00254                 if (tmp == NULL)
00255                 {
00256                     printf("Erreur %s : aucun dossier de ce nom\n", dir) ;
00257                     return ;
00258                 }
00259                 repertoire = tmp ;
00260                 dir = strtok(NULL, "/") ;
00261             }
00262         }
00263     }
00264     if(repertoire->perm[0] != 'r')
00265     {
00266         printf("Permission non accordée\n") ;
00267         return ;
00268     }
00269     fichier *temp = repertoire->premierfils ;
00270     while(temp)
00271     {
00272         if (temp->nom[0] != '.')      //On n'affiche pas un fichier si son nom se commence par un
point
00273         {
00274             if (temp->estDossier)
00275             {
00276                 printf("\033[1;36m%s\033[0m ",temp->nom);
00277             }
00278             else
00279             {
00280                 printf("\033[1;37m%s\033[0m ",temp->nom);
00281             }
00282         }
00283     }

```

```

00284         temp = temp->frereSuivant ;
00285     }
00286     printf("\n") ;
00287 }
00288
00289
00290
00291 void my_ls_a(fichier* racine, fichier *courant,char *nom)
00292 {
00293     fichier *repertoire ;
00294     if(nom[0]=='\0')
00295     {
00296         repertoire = courant ;
00297     }
00298     else if(strcmp(nom, "/") == 0)
00299     {
00300         repertoire = racine ;
00301     }
00302     else
00303     {
00304         if (strcmp(nom, "..") == 0)      //Pour remonter dans son parent
00305         {
00306             if (courant->parent)
00307             {
00308                 repertoire = courant->parent ; //dans son parent
00309             }
00310             else
00311             {
00312                 repertoire = courant; // reste à la racine
00313             }
00314         }
00315     }
00316
00317     else if (strcmp(nom, ".") == 0)    //Pour ne pas faire un déplacement
00318     {
00319         repertoire = courant ;
00320     }
00321     else
00322     {
00323         char *dir ;
00324         fichier *tmp , *dernier;
00325         if (nom[0] == '/')           //Si le nom se commence par un "/", donc parcourir depuis la
00326         racine
00327         {
00328             tmp = racine;
00329         }
00330         else    //Sinon, parcourir depuis le répertoire courant
00331         {
00332             tmp = courant ;
00333         }
00334         dir = strtok(nom,"/") ;    //Si le destination se fait dans un répertoire absolu
00335         if(strcmp(dir,".") == 0)
00336         {
00337             dir = strtok(NULL,"/") ;
00338         }
00339         if(strcmp(dir,"..") == 0)
00340         {
00341             if(courant->parent != NULL)
00342             {
00343                 tmp = courant->parent ;
00344             }
00345             else
00346             {
00347                 tmp = courant ;
00348             }
00349             dir = strtok(NULL,"/") ;
00350         }
00351
00352         while(dir != NULL)        //Parcourir la chaîne
00353         {
00354             tmp = chercher(tmp,dir) ; //Vérifier un par un si les dossiers existent
00355             if (tmp == NULL)
00356             {
00357                 printf("Erreur %s : aucun dossier de ce nom\n", dir) ;
00358                 return ;
00359             }
00360             repertoire = tmp ;
00361             dir = strtok(NULL, "/") ;
00362         }
00363     }
00364     if(repertoire->perm[0] != 'r')
00365     {
00366         printf("Permission non accordée\n") ;
00367         return ;
00368     }
00369     fichier *temp = repertoire->premierfils ;
00370     while(temp)
00371
00372
00373
00374

```

```

00375     {
00376         if (temp->estDossier)
00377         {
00378             printf("\033[1;36m%s\033[0m ",temp->nom);
00379         }
00380         else
00381         {
00382             printf("\033[1;37m%s\033[0m ",temp->nom);
00383         }
00384         temp = temp->frereSuivant ;
00385     }
00386     printf("\n") ;
00387 }
00394 void my_rmdir(fichier *racine, fichier *courant, char *nom)
00395 {
00396     char *dir , *dernier ;
00397     char *mot ;
00398     mot = malloc(strlen(nom) *sizeof(char)) ;
00399     int compt = 0 ;
00400     fichier* temoin ;
00401     fichier *parent ;
00402     char *suppr ;
00403     strcpy(mot, nom) ;
00404     if(strcmp(nom,"/") == 0)
00405     {
00406         printf("Erreur: impossible de supprimer racine\n") ;
00407         return ;
00408     }
00409
00410
00411     dir = strtok(mot, "/") ;      //Vérifier si le nom de dossier est un répertoire absolu
(exemple: ../../.../) ou un seul mot
00412     while(dir != NULL)    //Parcourir le répertoire absolu
00413     {
00414         compt++ ;
00415         dernier = dir ;
00416         dir = strtok(NULL, "/") ;
00417     }
00418     if (compt == 1)    //Si on ne voit pas un répertoire absolu mais un seul mot
00419     {
00420         parent = courant ;
00421         suppr = dernier ;
00422     }
00423     else
00424     {
00425         if (nom[0] == '/')      //Si le répertoire absolu se commence par un "/" , alors on va
parcourir depuis la racine
00426         {
00427             parent = racine;
00428         }
00429         else    //Sinon parcourir depuis le répertoire courant
00430         {
00431             parent = courant ;
00432         }
00433         dir = strtok(nom, "/") ;
00434         if(strcmp(dir,"..") == 0)
00435         {
00436             //strcpy(dir,courant->parent->nom) ;
00437             dir = strtok(NULL, "/") ;
00438         }
00439         if(strcmp(dir,".") == 0)
00440         {
00441             //strcpy(dir,courant->nom) ;
00442             dir = strtok(NULL, "/") ;
00443         }
00444         while(dir != NULL)    //Parcourir la chaîne
00445         {
00446             if (strcmp(dir, dernier) == 0)
00447             {
00448                 suppr = dernier ;
00449                 break ;
00450             }
00451
00452             if(dir != dernier)
00453             {
00454                 parent = chercher(parent, dir) ;    //Chercher les dossiers un par un si ils
existent
00455             }
00456
00457             if(parent == NULL)    //Si on ne trouve pas, on envoie un message d'erreur:
destination introuvable
00458             {
00459                 printf("Erreur: %s : aucun dossier de ce nom\n", dir) ;
00460                 return ;
00461             }
00462
00463 }
```

```

00464         dir = strtok(NULL, "/");
00465     }
00466
00467 }
00468 if (parent->perm[1] != 'w' || parent->perm[2] != 'x')
00469 {
00470     printf("rm : %s: Permission non accordée\n", nom) ;
00471     return ;
00472 }
00473 fichier *temp = parent->premierfils ;
00474 fichier *prec = NULL ;
00475
00476 while(temp != NULL)      //Parcourir les contenus du répertoire
00477 {
00478     if(strcmp(temp->nom, suppr) == 0)
00479     {
00480         if(!temp->estDossier) //Si il n'est pas un dossier
00481         {
00482             printf("Erreur: %s n'est pas un dossier\n", temp->nom) ;
00483             return ;
00484         }
00485         if(temp->premierfils != NULL) //Si il n'est pas vide
00486         {
00487             printf("Erreur: %s n'est pas vide\n", temp->nom) ;
00488             return ;
00489         }
00490         if(temp->estProtege)
00491         {
00492             printf("Erreur: ce dossier est protégé\n") ;
00493             return ;
00494         }
00495         if (prec != NULL) //Pour ne pas détruire l'arborescence
00496         {
00497             prec->frereSuivant = temp->frereSuivant;
00498         }
00499         else
00500         {
00501             courant->premierfils = temp->frereSuivant;
00502         }
00503
00504         free(temp) ;
00505         return ;
00506     }
00507     prec = temp ;
00508     temp = temp->frereSuivant ;
00509 }
00510 }
00511 printf("Erreur: %s n'existe pas\n", suppr) ;
00512 }

00522 fichier* my_cd(fichier *racine,fichier *courant, char *nom)
00523 {
00524     if(nom == NULL || strcmp(nom,"/") == 0)
00525     {
00526         return(racine) ;
00527     }
00528     if (strcmp(nom, "..") == 0)      //Pour remonter dans son parent
00529     {
00530         if (courant->parent)
00531         {
00532             courant->parent->nombre_ouverture++ ;
00533             return courant->parent ; //dans son parent
00534         }
00535         courant->nombre_ouverture++ ;
00536         return courant; // reste à la racine
00537     }
00538     if (strcmp(nom, ".") == 0)    //Pour ne pas faire un déplacement
00539     {
00540         courant->nombre_ouverture++ ;
00541         return(courant) ;
00542     }
00543     char *dir ;
00544     fichier *tmp , *dernier;
00545     if (nom[0] == '/') //Si le nom se commence par un "/", donc parcourir depuis la racine
00546     {
00547         tmp = racine;
00548     }
00549     else //Sinon, parcourir depuis le répertoire courant
00550     {
00551         tmp = courant ;
00552     }
00553     dir = strtok(nom,"/") ;    //Si le destination se fait dans un répertoire absolu
00554     if(strcmp(dir,".") == 0)
00555     {
00556         dir = strtok(NULL,"/") ;
00557     }

```

```

00558     if(strcmp(dir,"..") == 0)
00559     {
00560         if(courant->parent != NULL)
00561         {
00562             tmp = courant->parent ;
00563         }
00564         else
00565         {
00566             tmp = courant ;
00567         }
00568         dir = strtok(NULL,"/") ;
00569     }
00570
00571     while(dir != NULL)      //Parcourir la chaine
00572     {
00573         tmp = chercher(tmp,dir) ;    //Vérifier un par un si les dossiers existent
00574         if (tmp == NULL)
00575         {
00576             printf("Erreur %s : aucun dossier de ce nom\n", dir) ;
00577             return (courant) ;
00578         }
00579         dernier = tmp ;
00580         dir = strtok(NULL, "/") ;
00581     }
00582     if (dernier->estDossier == 1)    //On fait le déplacement si il est vraiment dossier
00583     {
00584         if(dernier->perm[2] != 'x')
00585         {
00586             printf("cd : %s: Permission non accordée\n", dernier->nom) ;
00587             return (courant) ;
00588         }
00589         dernier->nombre_ouverture++ ;
00590         return dernier;
00591     }
00592     else if (dernier->estDossier == 0)    //Si il n'est pas dossier, on envoie message d'erreur alors
00593     on ne fait rien de déplacement
00594     {
00595         printf("Erreur: %s n'est pas un dossier\n", dernier->nom) ;
00596         return(courant) ;
00597     }
00598     printf("my_cd: dossier '%s' introuvable\n", nom);
00599     return courant;
00600 }

00611 void my_cp(fichier *racine, fichier *rep_actuel, char *nom, char *destination)
00612 {
00613     if(strcmp(nom, "/") == 0)
00614     {
00615         printf("Erreur: impossible de copier racine\n") ;
00616         return ;
00617     }
00618     fichier *source ;
00619     char *mot1 ;
00620     fichier *temp, *templ, *courant1 ;
00621     if (nom[0] == '/')    //Si le nom source se commence par un "/", parcourir depuis la racine
00622     {
00623         courant1 = racine;
00624     }
00625     else    //Sinon, parcourir depuis le répertoire actuel
00626     {
00627         courant1 = rep_actuel ;
00628     }
00629     mot1 = strtok(nom, "/") ;
00630     if(strcmp(mot1,"..") == 0)
00631     {
00632         mot1 = strtok(NULL,"/") ;
00633     }
00634     if(strcmp(mot1,"..") == 0)
00635     {
00636         if(rep_actuel->parent != NULL)
00637         {
00638             courant1 = rep_actuel->parent ;
00639         }
00640         else
00641         {
00642             courant1 = rep_actuel ;
00643         }
00644         mot1 = strtok(NULL,"/") ;
00645     }
00646     while (mot1 != NULL)      //parcourir la source
00647     {
00648         source = chercher(courant1, mot1) ;    //Vérifier un par un si source existe
00649         if (source == NULL)        //Si source introuvable
00650         {
00651             printf("Erreur: %s (source) introuvable\n", mot1) ;

```

```

00652         return ;
00653     }
00654     courant1 = source ;
00655     mot1 = strtok(NULL, "/") ;
00656 }
00657 if (source->estDossier)      //Si le source est un dossier, on envoie un message d'erreur
00658 {
00659     printf("cp: -r non spécifié ; omission du répertoire %s\n", source->nom) ;
00660     return ;
00661 }
00662
00663
00664 //pour la destination
00665 fichier *courant, *dest ;
00666 char *mot ;
00667 if(strcmp(destination, ".") == 0)
00668 {
00669     dest = rep_actuel ;
00670 }
00671 else if(strcmp(destination, "..") == 0)
00672 {
00673     if(rep_actuel->parent == NULL)
00674     {
00675         printf("Erreur: %s n'a pas de dossier parent\n", rep_actuel->nom) ;
00676         return ;
00677     }
00678     dest = rep_actuel->parent ;
00679 }
00680 else if(strcmp(destination,"/") == 0)
00681 {
00682     dest = racine ;
00683 }
00684 else
00685 {
00686     if (destination[0] == '/')      //Si le destination se commence par un "/", parcourir depuis la
racine
00687     {
00688         courant = racine;
00689     }
00690     else
00691     {
00692         courant = rep_actuel ; //Sinon, parcourir depuis rep_actuel
00693     }
00694     mot = strtok(destination, "/") ;
00695     if(strcmp(mot,".") == 0)
00696     {
00697         mot = strtok(NULL,"/") ;
00698     }
00699     if(strcmp(mot,"..") == 0)
00700     {
00701         if(rep_actuel->parent != NULL)
00702         {
00703             courant = rep_actuel->parent ;
00704         }
00705         else
00706         {
00707             courant = rep_actuel ;
00708         }
00709         mot = strtok(NULL,"/") ;
00710     }
00711     while (mot != NULL)
00712     {
00713         dest = chercher(courant, mot) ;      //Vérifier un par un si les dossiers existent
00714         if (dest == NULL)          //Si destination introuvable
00715         {
00716             printf("Erreur: %s (destination) introuvable\n", mot) ;
00717             return ;
00718         }
00719         courant = dest ;
00720         mot = strtok(NULL, "/") ;
00721     }
00722     if (!dest->estDossier)      //Si destination n'est pas un dossier, on ne peut rien y ajouter
00723     {
00724         printf("Erreur: %s (destination) n'est pas un dossier", dest->nom) ;
00725         return ;
00726     }
00727 }
00728 if(chercher(dest, source->nom) != NULL)
00729 {
00730     printf("Erreur: ce nom de fichier existe déjà\n") ;
00731     return ;
00732 }
00733 if(dest->perm[1] != 'w' || dest->perm[2] != 'x')
00734 {
00735     printf("Permission non accordée\n") ;
00736     return ;
00737 }

```

```

00738     my_touch(racine,dest, source->nom, source->estProtege) ;
00739 }
00740
00741
00742
00751 void my_mv(fichier *racine, fichier *rep_actuel, char *nom, char *destination)
00752 {
00753     if(strcmp(nom, "/") == 0)
00754     {
00755         printf("Erreur: impossible de déplacer racine\n");
00756         return ;
00757     }
00758     fichier *source ;
00759     char *mot1 ;
00760     fichier *temp, *temp1, *courant1 ;
00761     if (nom[0] == '/') //Si le nom source se commence par un "/", parcourir depuis la racine
00762     {
00763         courant1 = racine ;
00764     }
00765     else //Sinon, parcourir depuis le répertoire actuel
00766     {
00767         courant1 = rep_actuel ;
00768     }
00769     mot1 = strtok(nom, "/");
00770     if(strcmp(mot1,"..") == 0)
00771     {
00772         mot1 = strtok(NULL,"/");
00773     }
00774     if(strcmp(mot1,".") == 0)
00775     {
00776         if(rep_actuel->parent != NULL)
00777         {
00778             courant1 = rep_actuel->parent ;
00779         }
00780         else
00781         {
00782             courant1 = rep_actuel ;
00783         }
00784         mot1 = strtok(NULL,"/");
00785     }
00786
00787     while (mot1 != NULL) //parcourir la source
00788     {
00789         source = chercher(courtant1, mot1); //Vérifier un par un si source existe
00790         if (source == NULL) //Si source introuvable
00791         {
00792             printf("Erreur: %s (source) introuvable\n", mot1);
00793             return ;
00794         }
00795         courtant1 = source ;
00796         mot1 = strtok(NULL, "/");
00797     }
00798
00799 //pour la destination
00800 fichier *courtant, *dest ;
00801 char *mot ;
00802 char copie_dest[20] ;
00803 char *dernier ;
00804 if(strcmp(destination, ".") == 0)
00805 {
00806     if(rep_actuel == source->parent)
00807     {
00808         printf("Erreur: impossible de déplacer %s dans le même répertoire\n", source->nom);
00809         return ;
00810     }
00811     if(rep_actuel->perm[1] != 'w' || rep_actuel->perm[2] != 'x')
00812     {
00813         printf("Permission non accordée\n");
00814         return ;
00815     }
00816     déplacer(rep_actuel, source) ;
00817 }
00818
00819 else if(strcmp(destination, "..") == 0)
00820 {
00821     if(rep_actuel->parent == NULL)
00822     {
00823         printf("Erreur: %s n'a pas de dossier parent\n", rep_actuel->nom);
00824         return ;
00825     }
00826     if(rep_actuel->parent->perm[1] != 'w' || rep_actuel->parent->perm[2] != 'x')
00827     {
00828         printf("Permission non accordée\n");
00829         return ;
00830     }
00831     déplacer(rep_actuel->parent, source) ;

```

```

00833     }
00834     else if(strcmp(destination, "/") == 0)
00835     {
00836         déplacer(racine, source) ;
00837     }
00838     else
00839     {
00840         char* test ;
00841         strcpy(copie_dest, destination) ;
00842         test = strtok(copie_dest,"/") ;
00843         while(test != NULL)
00844         {
00845             dernier = test ;
00846             test = strtok(NULL, "/") ;
00847         }
00848         if (destination[0] == '/')      //Si le destination se commence par un "/", parcourir depuis la
racine
00849         {
00850             courant = racine;
00851         }
00852         if(strchr(destination,'/')==NULL)
00853         {
00854             dest = chercher(rep_actuel,destination) ;
00855             if(dest != NULL)
00856             {
00857                 if(dest->perm[1] != 'w' || dest->perm[2] != 'x')
00858                 {
00859                     printf("Permission non accordée\n") ;
00860                     return ;
00861                 }
00862                 déplacer(dest, source) ;
00863                 return ;
00864             }
00865             strcpy(source->nom, destination) ;      //renommer
00866             return;
00867         }
00868     else
00869     {
00870         courant = rep_actuel ; //Sinon, parcourir depuis rep_actuel
00871     }
00872
00873     mot = strtok(destination, "/") ;
00874
00875     while (mot != NULL)
00876     {
00877         dest = chercher(courant, mot) ;      //Vérifier un par un si les dossiers existent
00878         if(strcmp(mot,dernier) == 0 && dest != NULL)
00879         {
00880             if (!dest->estDossier)      //Si destination n'est pas un dossier, on ne peut rien y
ajouter
00881             {
00882                 printf("Erreur: %s (destination) n'est pas un dossier", dest->nom) ;
00883                 return ;
00884             }
00885             if(dest->perm[1] != 'w' || dest->perm[2] != 'x')
00886             {
00887                 printf("Permission non accordée\n") ;
00888                 return ;
00889             }
00890             déplacer(dest, source) ;
00891             return ;
00892         }
00893         if(strcmp(mot,dernier) == 0 && dest == NULL)
00894         {
00895             if (!courant->estDossier)      //Si destination n'est pas un dossier, on ne peut rien y
ajouter
00896             {
00897                 printf("Erreur: %s (destination) n'est pas un dossier\n", courant->nom) ;
00898                 return ;
00899             }
00900             if(chercher(courant, dernier) != NULL)
00901             {
00902                 printf("Erreur: ce nom de fichier existe déjà\n") ;
00903                 return;
00904             }
00905             if(courant->perm[1] != 'w' || courant->perm[2] != 'x')
00906             {
00907                 printf("Permission non accordée\n") ;
00908                 return ;
00909             }
00910             strcpy(source->nom, dernier) ;      //renommer
00911             déplacer(courant, source) ;
00912             return ;
00913         }
00914         if (dest == NULL)      //Si destination introuvable
00915         {
00916             printf("Erreur: %s (destination) introuvable\n", mot) ;

```

```

00917             return ;
00918         }
00919
00920         courant = dest ;
00921         mot = strtok(NULL, "/");
00922     }
00923
00924 }
00925 }
00926
00930 void deplacer(fichier* dest, fichier* source)
00931 {
    //effacer la vraie source
00932     fichier *copie_source = source ;
00933     fichier *temporaire = source->parent->premierfils ;
00934     fichier *prec = NULL ;
00935     while(temporaire != NULL) //Parcourir les contenus du dossier afin de trouver le fichier à
        supprimer
00936     {
00937         if(strcmp(temporaire->nom, source->nom) == 0)
00938         {
00939             if (prec != NULL)
00940             {
00941                 prec->frereSuivant = temporaire->frereSuivant ;
00942             }
00943             else
00944             {
00945                 source->parent->premierfils = temporaire->frereSuivant ;
00946             }
00947             break ;
00948         }
00949         prec = temporaire ;
00950         temporaire = temporaire->frereSuivant ;
00951     }
00952
00953     copie_source->parent = dest ;
00954     if(dest->premierfils == NULL) //Si il n'y pas déjà un premier fils dans le dossier
00955     {
00956         dest->premierfils = copie_source ;
00957         copie_source->frereSuivant = NULL ;
00958     }
00959     else
00960     {
00961         fichier *templ = dest->premierfils ;
00962         while(templ->frereSuivant != NULL) //Parcourir dans les frères suivants si il y a déjà des
            contenus dans le dossier
00963         {
00964             templ = templ->frereSuivant ;
00965         }
00966         templ->frereSuivant = copie_source ;
00967         copie_source->frereSuivant = NULL ;
00968     }
00969 }
00970
00971 }
00972
00973 }
00974
00975
00976
00977
00984 fichier* chercher(fichier *courant, char *nom)
00985 {
00986     fichier *tmp ;
00987     tmp = courant->premierfils ;
00988     while(tmp != NULL) //Parcourir dans les fils du répertoire
00989     {
00990         if(strcmp(tmp->nom,nom) == 0)
00991         {
00992             return (tmp) ;
00993         }
00994         tmp = tmp->frereSuivant ;
00995     }
00996     return (NULL) ;
00997 }
00998
00999
01006 void my_rm(fichier *racine, fichier *courant, char *nom)
01007 {
01008     if(strcmp(nom,"/") == 0)
01009     {
01010         printf("Impossible de supprimer racine\n") ;
01011         return ;
01012     }
01013     char *dir , *dernier ;
01014     char *mot ;
01015     mot = malloc(strlen(nom) *sizeof(char)) ;
01016     int compt = 0 ;

```

```

01017     fichier* temoin ;
01018     fichier *parent ;
01019     char *suppr ;
01020     strcpy(mot, nom) ;
01021     dir = strtok(mot, "/") ;    //Vérifier si le nom de dossier est un répertoire absolu
01022     (exemple: ../../.../) ou un seul mot
01023     while(dir != NULL)      //Parcourir le répertoire absolu
01024     {
01025         compt++ ;
01026         dernier = dir ;
01027         dir = strtok(NULL, "/") ;
01028     if (compt == 1)    //Si on ne voit pas un répertoire absolu mais un seul mot
01029     {
01030         parent = courant ;
01031         suppr = dernier ;
01032     }
01033 else
01034 {
01035     if (nom[0] == '/')      //Si le répertoire absolu se commence par un "/" , alors on va
01036     parcourir depuis la racine
01037     {
01038         parent = racine;
01039     else    //Sinon parcourir depuis le répertoire courant
01040     {
01041         parent = courant ;
01042     }
01043     dir = strtok(nom, "/") ;
01044     if(strcmp(dir,"..") == 0)
01045     {
01046         //strcpy(dir,courant->parent->nom) ;
01047         dir = strtok(NULL, "/") ;
01048     }
01049     if(strcmp(dir,".") == 0)
01050     {
01051         //strcpy(dir,courant->nom) ;
01052         dir = strtok(NULL, "/") ;
01053     }
01054     while(dir != NULL)      //Parcourir la chaîne
01055     {
01056         if (strcmp(dir, dernier) == 0)
01057         {
01058             suppr = dernier ;
01059             break ;
01060         }
01061         if(dir != dernier)
01062         {
01063             parent = chercher(parent, dir) ;    //Chercher les dossiers un par un si ils existent
01064         }
01065     if(parent == NULL)      //Si on ne trouve pas, on envoie un message d'erreur: destination
01066     introuvable
01067     {
01068         printf("Erreur: %s : aucun dossier de ce nom\n", dir) ;
01069         return ;
01070     }
01071     dir = strtok(NULL, "/") ;
01072 }
01073 }
01074 }
01075 }
01076 }
01077 if (courant->perm[1] != 'w' || courant->perm[2] != 'x')
01078 {
01079     printf("rm : %s: Permission non accordée\n", nom) ;
01080     return ;
01081 }
01082 fichier *temp = parent->premierfils ;
01083 fichier *prec = NULL ;
01084 while(temp != NULL)      //Parcourir les contenus du dossier afin de trouver le fichier à supprimer
01085 {
01086     if(strcmp(temp->nom, suppr) == 0)
01087     {
01088         if(temp->estDossier)      //On envoie "erreur" si il est dossier
01089         {
01090             printf("Erreur: %s est un dossier\n", temp->nom) ;
01091             return ;
01092         }
01093         if (prec != NULL)
01094         {
01095             prec->frereSuivant = temp->frereSuivant;
01096         }
01097         else
01098         {
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02260
02261
02262
02263
02264
02265
02266
02267
02268
02269
02270
02271
02272
02273
02274
02275
02276
02277
02278
02279
02280
02281
02282
02283
02284
02285
02286
02287
02288
02289
02290
02291
02292
02293
02294
02295
02296
02297
02298
02299
02300
02301
02302
02303
02304
02305
02306
02307
02308
02309
02310
02311
02312
02313
02314
02315
02316
02317
02318
02319
02320
02321
02322
02323
02324
02325
02326
02327
02328
02329
02330
02331
02332
02333
02334
02335
02336
02337
02338
02339
02340
02341
02342
02343
02344
02345
02346
02347
02348
02349
02350
02351
02352
02353
02354
02355
02356
02357
02358
02359
02360
02361
02362
02363
02364
02365
02366
02367
02368
02369
02370
02371
02372
02373
02374
02375
02376
02377
02378
02379
02380
02381
02382
02383
02384
02385
02386
02387
02388
02389
02390
02391
02392
02393
02394
02395
02396
02397
02398
02399
02400
02401
02402
02403
02404
02405
02406
02407
02408
02409
02410
02411
02412
02413
02414
02415
02416
02417
02418
02419
02420
02421
02422
02423
02424
02425
02426
02427
02428
02429
02430
02431
02432
02433
02434
02435
02436
02437
02438
02439
02440
02441
02442
02443
02444
02445
02446
02447
02448
02449
02450
02451
02452
02453
02454
02455
02456
02457
02458
02459
02460
02461
02462
02463
02464
02465
02466
02467
02468
02469
02470
02471
02472
02473
02474
02475
02476
02477
02478
02479
02480
02481
02482
02483
02484
02485
02486
02487
02488
02489
02490
02491
02492
02493
02494
02495
02496
02497
02498
02499
02500
02501
02502
02503
02504
02505
02506
02507
02508
02509
02510
02511
02512
02513
02514
02515
02516
02517
02518
02519
02520
02521
02522
02523
02524
02525
02526
02527
02528
02529
02530
02531
02532
02533
02534
02535
02536
02537
02538
02539
02540
02541
02542
02543
02544
02545
02546
02547
02548
02549
02550
02551
02552
02553
02554
02555
02556
02557
02558
02559
02560
02561
02562
02563
02564
02565
02566
02567
02568
02569
02570
02571
02572
02573
02574
02575
02576
02577
02578
02579
02580
02581
02582
02583
02584
02585
02586
02587
02588
02589
02590
02591
02592
02593
02594
02595
02596
02597
02598
02599
02600
02601
02602
02603
02604
02605
02606
02607
02608
02609
02610
02611
02612
02613
02614
02615
02616
02617
02618
02619
02620
02621
02622
02623
02624
02625
02626
02627
02628
02629
02630
02631
02632
02633
02634
02635
02636
02637
02638
02639
02640
02641
02642
02643
02644
02645
02646
02647
02648
02649
02650
02651
02652
02653
02654
02655
02656
02657
02658
02659
02660
02661
02662
02663
02664
02665
02666
02667
02668
02669
02670
02671
02672
02673
02674
02675
02676
02677
02678
02679
02680
02681
02682
02683
02684
02685
02686
02687
02688
02689
02690
02691
02692
02693
02694
02695
02696
02697
02698
02699
02700
02701
02702
02703
02704
02705
02706
02707
02708
02709
02710
02711
02712
02713
02714
02715
02716
02717
02718
02719
02720
02721
02722
02723
02724
02725
02726
02727
02728
02729
02730
02731
02732
02733
02734
02735
02736
02737
02738
02739
02740
02741
02742
02743
02744
02745
02746
02747
02748
02749
02750
02751
02752
02753
02754
02755
02756
02757
02758
02759
02760
02761
02762
02763
02764
02765
02766
02767
02768
```

```

01101         courant->premierfils = temp->frereSuivant;
01102     }
01103     free(temp) ;
01104     return ;
01105   }
01106   prec = temp ;
01107   temp = temp->frereSuivant ;
01108 }
01109 printf("Erreur: %s n'existe pas\n", suppr) ;
01110 }
01111 }
01112 }
01113
01114 void my_mkdir_p(fichier *racine, fichier *parent, char *nom)
01115 {
01116     char *dir ;
01117     fichier *courant ;
01118     dir = strtok(nom, "/") ;
01119     while(dir != NULL)
01120     {
01121         //my_mkdir(racine, parent, dir, statut) ;
01122         courant = my_cd(racine, parent, dir) ;
01123         parent = courant ;
01124         dir = strtok(NULL, "/") ;
01125     }
01126 }
01127 }
01128
01129
01130
01131
01132
01133 }
01134
01135
01136
01137 void my_mkdir(fichier *racine, fichier *courant, char *nom,int statut)
01138 {
01139     if(strcmp(nom,"/") == 0)
01140     {
01141         printf("Erreur: impossible de créer racine\n") ;
01142         return ;
01143     }
01144     char *dir , *dernier ;
01145     char *mot ;
01146     mot = malloc(strlen(nom) *sizeof(char)) ;
01147     fichier *temoin ;
01148     int compt = 0 ;
01149     fichier *parent ;
01150     strcpy(mot, nom) ;
01151     dir = strtok(mot, "/") ;    //Vérifier si le nom de dossier est un répertoire absolu
(exemple:/.../.../....) ou un seul mot
01152     while(dir != NULL)      //Parcourir le répertoire absolu
01153     {
01154         compt++ ;
01155         dernier = dir ;
01156         dir = strtok(NULL, "/") ;
01157     }
01158     if (compt == 1)    //Si on ne voit pas un répertoire absolu mais un seul mot
01159     {
01160         temoin = chercher(courant, dernier) ;    //Vérifier si existe déjà
01161         if(temoin == NULL)
01162         {
01163             if(courant->perm[1] != 'w' || courant->perm[2] != 'x')
01164             {
01165                 printf("Permission non accordée\n") ;
01166                 return ;
01167             }
01168             ajouter_enfant(courant, dernier, 1, statut) ;
01169         }
01170         else
01171         {
01172             printf("Erreur: le fichier existe déjà\n") ;
01173         }
01174     }
01175     return ;
01176 }
01177 }
01178
01179
01180
01181 }
01182     if (nom[0] == '/')      //Si le répertoire absolu se commence par un "/" , alors on va parcourir
depuis la racine
01183     {
01184         parent = racine;
01185     }
01186     else      //Sinon parcourir depuis le répertoire courant
01187     {
01188         parent = courant ;
01189     }
01190     if (compt > 1)      //Si le nom de dossier est un répertoire absolu (exemple:/.../.../....)
01191     {
01192         dir = strtok(nom, "/") ;
01193
01194         while(dir != NULL)      //Parcourir la chaîne
01195         {
01196             if (strcmp(dir, dernier) == 0)
01197             {
01198                 break ;
01199             }

```

```

01200
01201         if(dir != dernier)
01202         {
01203             parent = chercher(parent, dir) ; //Chercher les dossiers un par un si ils existent
01204         }
01205
01206         if(parent == NULL) //Si on ne trouve pas, on envoie un message d'erreur: destination
01207             introuvable
01208         {
01209             printf("Erreur: %s : aucun dossier de ce nom\n", dir) ;
01210             return ;
01211         }
01212         dir = strtok(NULL, "/");
01213     }
01214 }
01215 témoin = chercher(parent, dernier) ; //Vérifier si existe déjà
01216 if(témoin == NULL)
01217 {
01218     if(parent->perm[1] != 'w' || parent->perm[2] != 'x')
01219     {
01220         printf("Permission non accordée\n");
01221         return ;
01222     }
01223     ajouter_enfant(parent, dernier, 1, statut) ;
01224 }
01225 else
01226 {
01227     printf("Erreur: le dossier existe déjà\n") ;
01228 }
01229 }
01230
01231 void my_pwd(fichier *rep_actuel)
01232 {
01233     char *chemin;
01234
01235     chemin = calloc(32 , sizeof(char));
01236     get_pwd( rep_actuel , chemin );
01237     printf("%s\n", chemin);
01238
01239     free(chemin);
01240 }
01241 void my_echo(char **ligne_c)
01242 {
01243     int i;
01244
01245     i = 1;
01246     while(ligne_c[i][0] != '\0')
01247     {
01248         printf("%s " , ligne_c[i]);
01249         i++;
01250     }
01251     printf("\n");
01252 }
01253
01254
01255 void my_chmod(fichier* racine, fichier* courant, char* option, char* nom)
01256 {
01257     if(strcmp(nom,"/") == 0)
01258     {
01259         printf("Erreur: impossible de modifier les permissions de racine\n") ;
01260         return ;
01261     }
01262     char *dir ;
01263     fichier *tmp , *dernier;
01264     if (nom[0] == '/') //Si le nom se commence par un "/", donc parcourir depuis la racine
01265     {
01266         tmp = racine;
01267     }
01268     else //Sinon, parcourir depuis le répertoire courant
01269     {
01270         tmp = courant ;
01271     }
01272     dir = strtok(nom,"/") ; //Si le destination se fait dans un répertoire absolu
01273     if(strcmp(dir,".") == 0)
01274     {
01275         dir = strtok(NULL,"/");
01276     }
01277     if(strcmp(dir,"..") == 0)
01278     {
01279         if(courant->parent != NULL)
01280         {
01281             tmp = courant->parent ;
01282         }
01283         else
01284         {
01285             tmp = courant ;
01286         }
01287     }
01288 }
```

```

01290         }
01291         dir = strtok(NULL,"/");
01292     }
01293
01294     while(dir != NULL) //Parcourir la chaine
01295     {
01296         tmp = chercher(tmp,dir); //Vérifier un par un si les dossiers existent
01297         if (tmp == NULL)
01298         {
01299             printf("Erreur %s : aucun dossier de ce nom\n", dir);
01300             return;
01301         }
01302         dernier = tmp;
01303         dir = strtok(NULL, "/");
01304     }
01305
01306     if(option[0] == '+')
01307     {
01308         //On ajoute
01309         if(strchr(option,'r'))
01310         {
01311             dernier->perm[0] = 'r';
01312         }
01313         if(strchr(option,'w'))
01314         {
01315             dernier->perm[1] = 'w';
01316         }
01317         if(strchr(option,'x'))
01318         {
01319             dernier->perm[2] = 'x';
01320         }
01321         if(strchr(option,'r') == NULL && strchr(option,'w') == NULL && strchr(option,'x') == NULL)
01322         {
01323             printf("Erreur: option %s incorrecte\n", option);
01324             return;
01325         }
01326     }
01327     else if(option[0] == '-')
01328     {
01329         if(strchr(option,'r'))
01330         {
01331             dernier->perm[0] = '-';
01332         }
01333         if(strchr(option,'w'))
01334         {
01335             dernier->perm[1] = '-';
01336         }
01337         if(strchr(option,'x'))
01338         {
01339             dernier->perm[2] = '-';
01340         }
01341         if(strchr(option,'r') == NULL && strchr(option,'w') == NULL && strchr(option,'x') == NULL)
01342         {
01343             printf("Erreur: option %s incorrecte\n", option);
01344             return;
01345         }
01346     }
01347     else if(option[0] == '0')
01348     {
01349         strcpy(dernier->perm, "---");
01350     }
01351     else if(option[0] == '1')
01352     {
01353         strcpy(dernier->perm, "--x");
01354     }
01355     else if(option[0] == '2')
01356     {
01357         strcpy(dernier->perm, "-w-");
01358     }
01359     else if(option[0] == '3')
01360     {
01361         strcpy(dernier->perm, "-wx");
01362     }
01363     else if(option[0] == '4')
01364     {
01365         strcpy(dernier->perm, "r--");
01366     }
01367     else if(option[0] == '5')
01368     {
01369         strcpy(dernier->perm, "r-x");
01370     }
01371     else if(option[0] == '6')
01372     {
01373         strcpy(dernier->perm, "rw-");
01374     }
01375     else if(option[0] == '7')
01376     {

```

```

01377         strcpy(dernier->perm, "rwx") ;
01378     }
01379     else
01380     {
01381         printf("Erreur: option %c incorrecte\n", option[0]) ;
01382         return ;
01383     }
01384 }
01385
01386
01387 void my_ls_1(fichier* racine, fichier *courant,char *nom)
01388 {
01389
01390     fichier *repertoire ;
01391     if(nom[0]=='\0')
01392     {
01393         repertoire = courant ;
01394     }
01395     else if (strcmp(nom, "/") == 0)
01396     {
01397         repertoire = racine ;
01398     }
01399     else
01400     {
01401         if (strcmp(nom, "..") == 0)           //Pour remonter dans son parent
01402         {
01403             if (courant->parent)
01404             {
01405                 repertoire = courant->parent ; //dans son parent
01406             }
01407             else
01408             {
01409                 repertoire = courant; // reste à la racine
01410             }
01411         }
01412
01413         else if (strcmp(nom, ".") == 0)    //Pour ne pas faire un déplacement
01414         {
01415             repertoire = courant ;
01416         }
01417         else
01418         {
01419             char *dir ;
01420             fichier *tmp , *dernier;
01421             if (nom[0] == '/')           //Si le nom se commence par un "/", donc parcourir depuis la
racine
01422             {
01423                 tmp = racine;
01424             }
01425             else    //Sinon, parcourir depuis le répertoire courant
01426             {
01427                 tmp = courant ;
01428             }
01429             dir = strtok(nom,"/") ;      //Si le destination se fait dans un répertoire absolu
01430             if(strcmp(dir,".") == 0)
01431             {
01432                 dir = strtok(NULL,"/") ;
01433             }
01434             if(strcmp(dir,"..") == 0)
01435             {
01436                 if(courant->parent != NULL)
01437                 {
01438                     tmp = courant->parent ;
01439                 }
01440                 else
01441                 {
01442                     tmp = courant ;
01443                 }
01444             dir = strtok(NULL,"/") ;
01445         }
01446
01447             while(dir != NULL)        //Parcourir la chaîne
01448             {
01449                 tmp = chercher(tmp,dir) ; //Vérifier un par un si les dossiers existent
01450                 if (tmp == NULL)
01451                 {
01452                     printf("Erreur %s : aucun dossier de ce nom\n", dir) ;
01453                     return ;
01454                 }
01455                 repertoire = tmp ;
01456                 dir = strtok(NULL, "/" ) ;
01457             }
01458         }
01459     }
01460     if(repertoire->perm[0] != 'r' || repertoire->perm[2] != 'x')
01461     {
01462         printf("Permission non accordée\n") ;

```

```

01463         return ;
01464     }
01465     fichier *temp = repertoire->premierfils ;
01466     while(temp)
01467     {
01468         if (temp->nom[0] != '.')      //On n'affiche pas un fichier si son nom se commence par un
01469         point
01470         {
01471             if (temp->estDossier)
01472             {
01473                 printf("%s\t\033[1;36m%s\033[0m\n",temp->perm,temp->nom);
01474             }
01475             else
01476             {
01477                 printf("%s\t\033[1;37m%s\033[0m\n", temp->perm, temp->nom);
01478             }
01479         }
01480         temp = temp->frereSuivant ;
01481     }
01482     printf("\n") ;
01483 }
01484
01485
01486 void afficher_nbr_ouverture(fichier* home, int prfnd)
01487 {
01488     if(home == NULL)
01489     {
01490         return ;
01491     }
01492     for(int i = 0; i<prfnd; i++)
01493     {
01494         printf("---");
01495     }
01496     printf("%d : %s\n", home->nombre_ouverture, home->nom ) ;
01497     afficher_nbr_ouverture(home->premierfils, prfnd + 1) ; //On appelle recursivement la fonction
01498     lui-même
01499     afficher_nbr_ouverture(home->frereSuivant, prfnd) ;
01500 }
01501 int recherche_max(fichier* home, int max)
01502 {
01503     if(home == NULL)
01504     {
01505         return max;
01506     }
01507     if(strcmp(home->nom , "AK-07") != 0 && home->nombre_ouverture > max)
01508     {
01509         max = home->nombre_ouverture ;
01510     }
01511     max = recherche_max(home->premierfils, max) ; //On appelle recursivement la fonction lui-même
01512     max = recherche_max(home->frereSuivant, max) ;
01513     return max ;
01514 }
01515
01516 void favori(fichier* home, int max)
01517 {
01518     if(home == NULL)
01519     {
01520         return ;
01521     }
01522     if(strcmp(home->nom, "AK-07") != 0 && home->nombre_ouverture == max)
01523     {
01524         printf("%d : %s\n", home->nombre_ouverture, home->nom ) ;
01525     }
01526     favori(home->premierfils, max) ; //On appelle recursivement la fonction lui-même
01527     favori(home->frereSuivant, max) ;
01528 }
01529
01530 void affichage(fichier* home)
01531 {
01532     int max;
01533     max = recherche_max(home,0) ;
01534     favori(home, max) ;
01535 }
01536
01537 void my_ls_la(fichier* racine, fichier *courant,char *nom)
01538 {
01539     fichier *repertoire ;
01540     if(nom[0]=='\0')
01541     {
01542         repertoire = courant ;
01543     }
01544     else if(strcmp(nom,"/") == 0)
01545     {
01546         repertoire = racine ;
01547     }

```

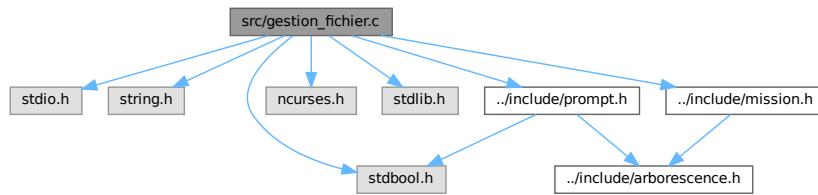
```

01548     else
01549     {
01550         if (strcmp(nom, "..") == 0)      //Pour remonter dans son parent
01551     {
01552         if (courant->parent)
01553     {
01554         repertoire = courant->parent ; //dans son parent
01555     }
01556     else
01557     {
01558         repertoire = courant; // reste à la racine
01559     }
01560 }
01561
01562     else if (strcmp(nom, ".") == 0)    //Pour ne pas faire un déplacement
01563     {
01564         repertoire = courant ;
01565     }
01566     else
01567     {
01568         char *dir ;
01569         fichier *tmp , *dernier;
01570         if (nom[0] == '/')           //Si le nom se commence par un "/", donc parcourir depuis la
racine
01571     {
01572         tmp = racine;
01573     }
01574     else    //Sinon, parcourir depuis le répertoire courant
01575     {
01576         tmp = courant ;
01577     }
01578     dir = strtok(nom,"/") ;        //Si la destination se fait dans un répertoire absolu
01579     if(strcmp(dir,".") == 0)
01580     {
01581         dir = strtok(NULL,"/") ;
01582     }
01583     if(strcmp(dir,"..") == 0)
01584     {
01585         if(courant->parent != NULL)
01586     {
01587         tmp = courant->parent ;
01588     }
01589     else
01590     {
01591         tmp = courant ;
01592     }
01593     dir = strtok(NULL,"/") ;
01594 }
01595
01596     while(dir != NULL)          //Parcourir la chaîne
01597     {
01598         tmp = chercher(tmp,dir) ; //Vérifier un par un si les dossiers existent
01599         if (tmp == NULL)
01600     {
01601         printf("Erreur %s : aucun dossier de ce nom\n", dir) ;
01602         return ;
01603     }
01604         repertoire = tmp ;
01605         dir = strtok(NULL, "/" ) ;
01606     }
01607 }
01608 }
01609 if(repertoire->perm[0] != 'r' || repertoire->perm[2] != 'x')
01610 {
01611     printf("Permission non accordée\n") ;
01612     return ;
01613 }
01614 fichier *temp = repertoire->premierfils ;
01615 while(temp)
01616 {
01617     if (temp->estDossier)
01618     {
01619         printf("%s\t\033[1;36m%s\033[0m \n",temp->perm,temp->nom);
01620     }
01621     else
01622     {
01623         printf("%s\t\033[1;37m%s\033[0m \n", temp->perm, temp->nom);
01624     }
01625
01626
01627     temp = temp->frereSuivant ;
01628 }
01629 printf("\n") ;
01630 }
01631 }
```

## 4.19 src/gestion\_fichier.c File Reference

regroupe tous les fonctions qui manipule les fichiers

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <ncurses.h>
#include <stdlib.h>
#include "../include/mission.h"
#include "../include/prompt.h"
Include dependency graph for gestion_fichier.c:
```



### Functions

- void [cat\\_fichier](#) (char \*nom\_fichier, bool num)  
*afficher le fichier entrée en parametre*
- int [affiche\\_ligne](#) (char \*chemin, int num\_ligne, char \*emplacement)  
*Lire une ligne precise dans un fichier depart de la fin si negative.*
- void [insert\\_fin](#) (char \*nom\_fichier, char \*chaine\_caractere)
- void [head\\_fichier](#) (char \*nom\_fichier, int max)  
*il affiche des premiers lignes d'une fichier entrer en parametre*
- void [tail\\_fichier](#) (char \*nom\_fichier, int min)
- int [lire\\_histoire](#) ()  
*cette fonction lit le fichier historique des commandes*
- void [voir\\_indice](#) (Mission \*mission)  
*Elle affiche une petite aide apropos d'une mission en decrementant le score du joueur actuelle.*
- void [qui\\_est\\_meilleur](#) (int n\_miss)  
*cette fonction affche qui a le meilleur temps concernant une mission entrer en argument*
- int [my\\_man](#) (char \*nom\_commande)  
*Il affiche une page d'aide pour une commande entrer en argument.*
- int [my\\_history](#) (bool effacer)  
*afficher ou effacer les historiques de ligne de commande*

### 4.19.1 Detailed Description

regroupe tous les fonctions qui manipule les fichiers

**Author**

Valisoa

**Date**

29 Août 2025

**Version**

1.0.0

toutes les manipulations de fichier comme les enregistrements et les affichages sont regroupés dans celle-ci ; ces fonctions sont cités dans gestion\_fichier.h

Definition in file [gestion\\_fichier.c](#).

## 4.19.2 Function Documentation

### 4.19.2.1 affiche\_ligne()

```
int affiche_ligne (
    char * chemin,
    int num_ligne,
    char * emplacement )
```

Lire une ligne précise dans un fichier départ de la fin si négative.

**Parameters**

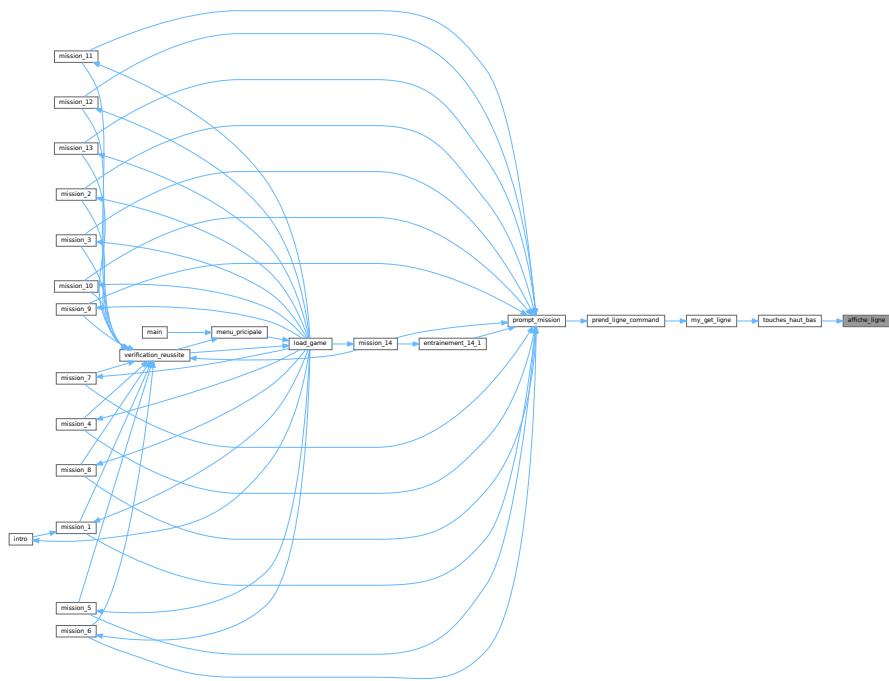
<i>chemin</i>	le chemin vers le fichier
<i>num_ligne</i>	le numéro de ligne à afficher
<i>emplacement</i>	la variable qui va contenir la ligne voulue

**Returns**

renvoie une valeur négative si le fichier n'est pas existant

Definition at line 51 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:



#### 4.19.2.2 cat\_fichier()

```
void cat_fichier (
    char * nom_fichier,
    bool num )
```

afficher le fichier entrée en parametre

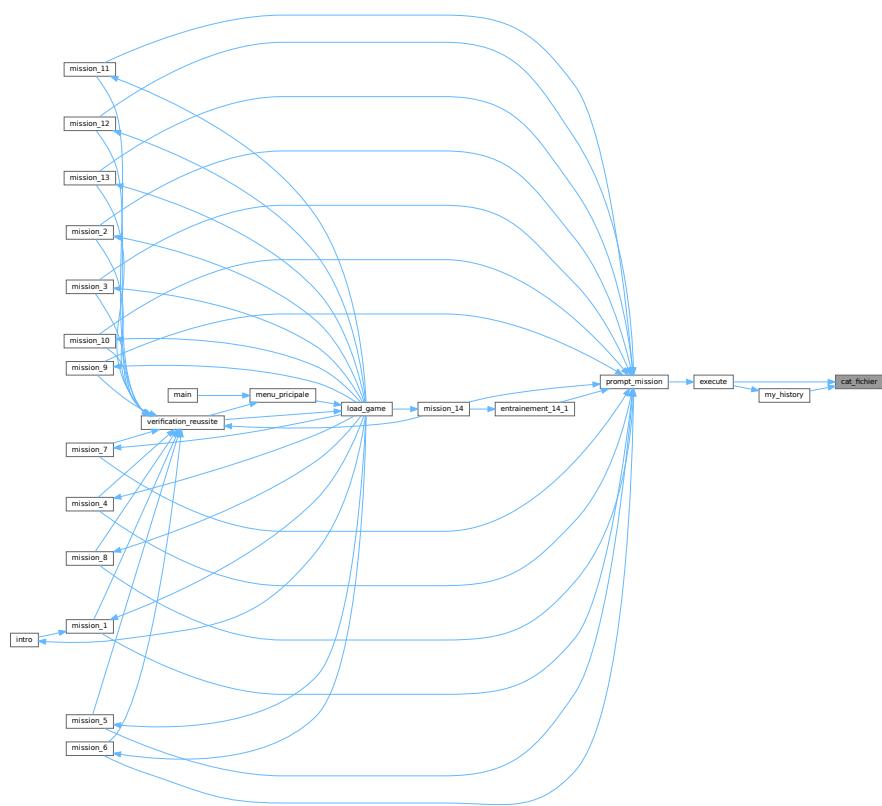
afficher le contenu d'un fichier

##### Parameters

<i>nom_fichier</i>	nom de la fichier à afficher
<i>nom_fichier</i>	le nom de fichier à afficher
<i>num</i>	affiche le numero de ligne si true est la valeur de celle ci

Definition at line 20 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:



#### 4.19.2.3 head\_fichier()

```
void head_fichier (
    char * nom_fichier,
    int max )
```

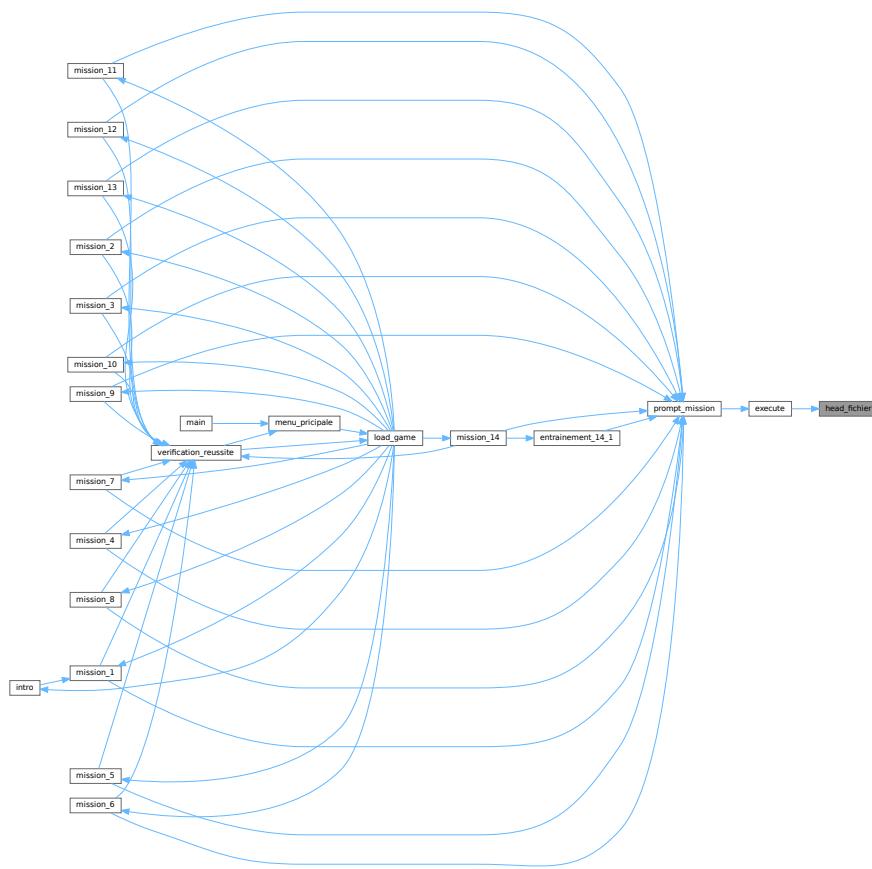
il affiche des premiers lignes d'une fichier entrer en parametre

##### Parameters

<i>nom_fichier</i>	le nom de la fichier
<i>max</i>	le nombre de ligne à afficher

Definition at line 149 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:



#### 4.19.2.4 insert\_fin()

```
void insert_fin (
    char * nom_fichier,
    char * chaine_caractere )
```

Definition at line 134 of file [gestion\\_fichier.c](#).

#### 4.19.2.5 lire\_histoire()

```
int lire_histoire ( )
cette fonction lit le fichier historique des commandes
```

##### Returns

une valeur negatif si le fichier histoire est vide ou inexistant

Definition at line 229 of file [gestion\\_fichier.c](#).

#### 4.19.2.6 my\_history()

```
int my_history (
    bool effacer )
```

afficher ou effacer les historiques de ligne de commande

**Parameters**

<code>effacer</code>	effacer le contenu historique si true
----------------------	---------------------------------------

**Returns**

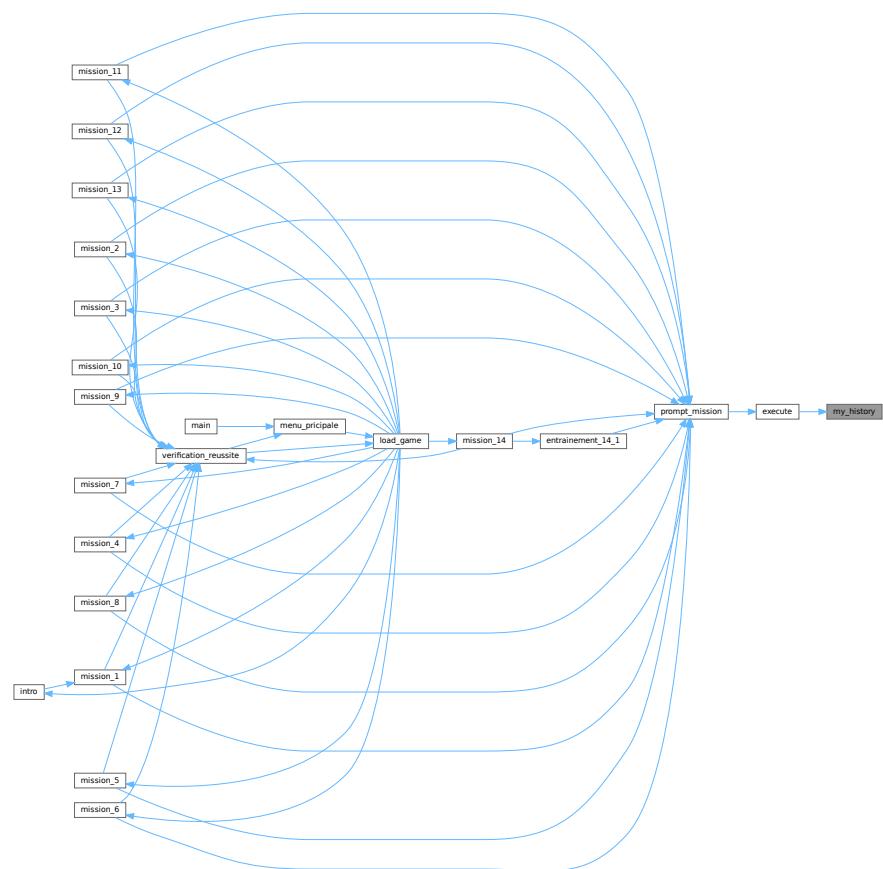
une valeur negative si la lecture a echouée

Definition at line 378 of file [gestion\\_fichier.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.19.2.7 my\_man()

```
int my_man (
    char * nom_commande )
```

Il affiche une page d'aide pour une commande entrer en argument.

##### Parameters

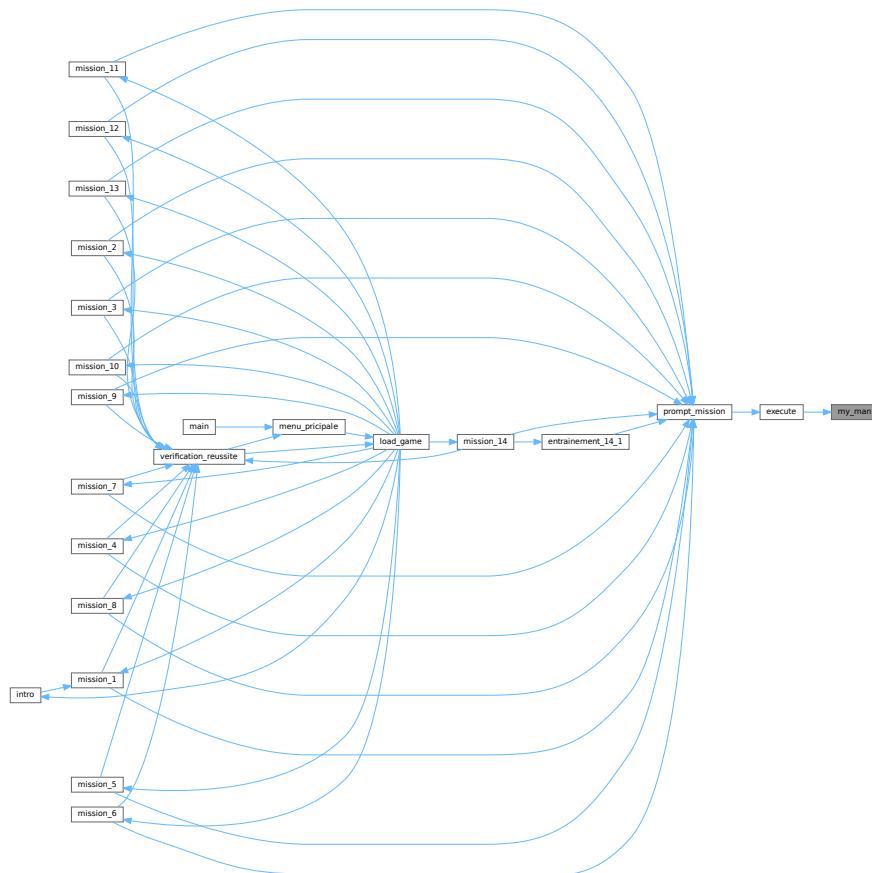
<i>nom_commande</i>	le nom de la commande à afficher l'aide
---------------------	-----------------------------------------

##### Returns

une valeur negative en cas d'erreur

Definition at line 353 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:



#### 4.19.2.8 qui\_est\_meilleur()

```
void qui_est_meilleur (
    int n_miss )
```

cette fonction affche qui a le meilleur temps concernant une mission entrer en argument

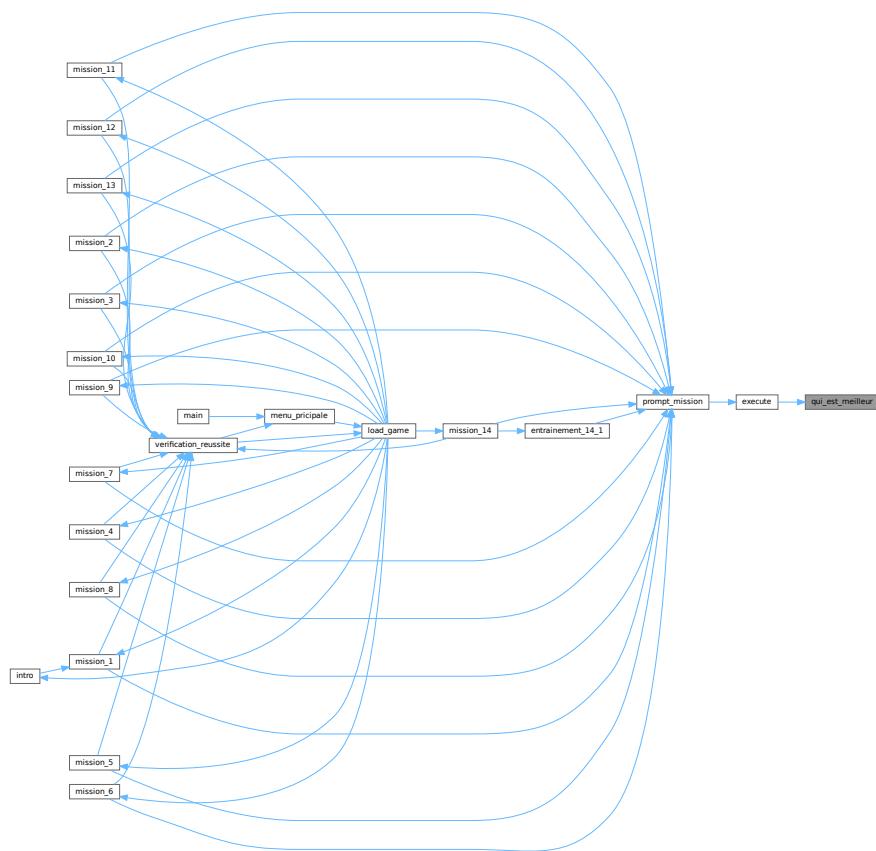
## Parameters

<i>n_miss</i>	le numero de la mission
---------------	-------------------------

traitement de donnée obtenus d'après la lecture du fichier de score

Definition at line 295 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:

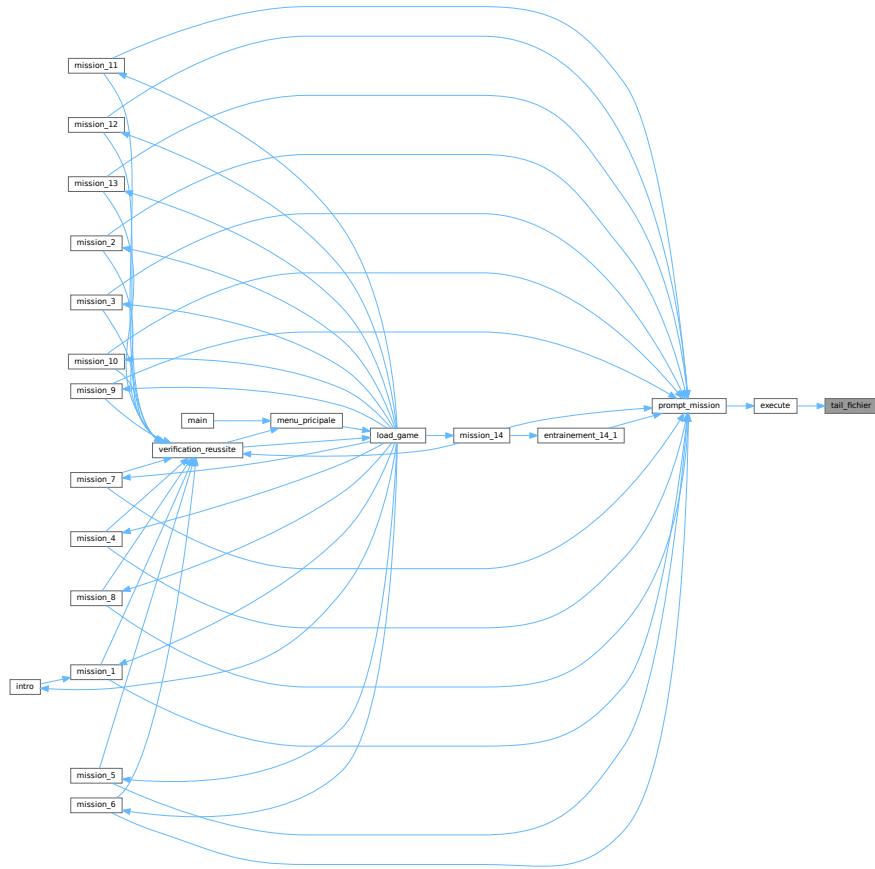


#### 4.19.2.9 tail\_fichier()

```
void tail_fichier (
    char * nom_fichier,
    int min )
```

Definition at line 173 of file [gestion\\_fichier.c](#).

Here is the caller graph for this function:



#### 4.19.2.10 voir\_indice()

```
void voir_indice ( Mission * mission )
```

Elle affiche une petite aide apropos d'une mission en decrementant le score du joueur actuelle.

## Parameters

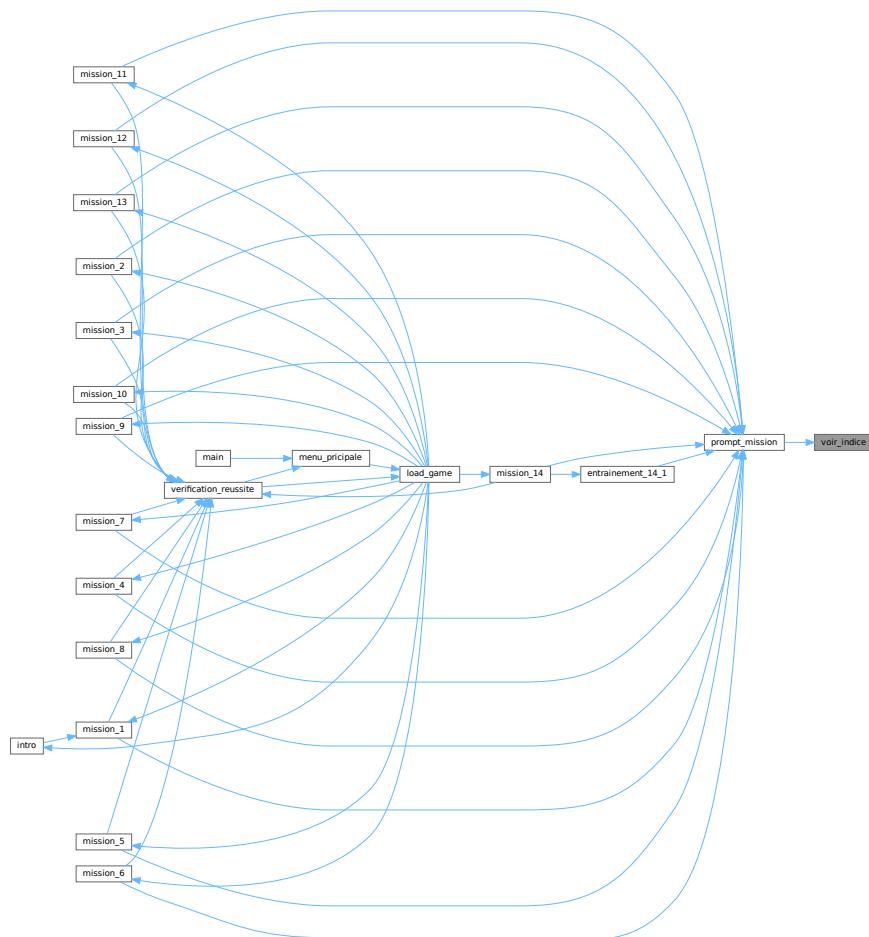
*mission* l'adresse du mission que l'on veut voir l'indice

Definition at line 255 of file [gestion\\_fichier.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.20 gestion\_fichier.c

[Go to the documentation of this file.](#)

```
00001
00012 #include <stdio.h>
00013 #include <string.h>
00014 #include <stdbool.h>
00015 #include <ncurses.h>
00016 #include <stdlib.h>
00017 #include "../include/mission.h"
```

```

00018 #include "../include/prompt.h"
00019
00020 void cat_fichier(char *nom_fichier , bool num)
00021 {
00022     FILE *ptr_Fic;
00023     char *ligne;
00024     int num_ligne;
00025
00026     num_ligne = 0;
00027     ptr_Fic = NULL;
00028     ptr_Fic = fopen(nom_fichier, "r");
00029
00030     ligne = calloc(LIGNE, sizeof(char));
00031     if(ptr_Fic == NULL)
00032     {
00033         return;
00034     }
00035     while( !(feof(ptr_Fic)) )
00036     {
00037         fgets(ligne, LIGNE, ptr_Fic);
00038         if(num_ligne != 0)
00039         {
00040             if(num)
00041             {
00042                 printf("%d\t", num_ligne);
00043             }
00044         }
00045         printf("%s", ligne);
00046         num_ligne++;
00047     }
00048     free(ligne);
00049     printf("\n");
00050 }
00051 int affiche_ligne(char *chemin , int num_ligne , char *emplacement)
00052 {
00053     FILE *ptr_Fic;
00054     int nb_ligne , position , i;
00055     char caractere;
00056     char *ligne;
00057     bool limite;
00058
00059     ptr_Fic = fopen(chemin , "r");
00060     if(ptr_Fic == NULL)
00061     {
00062         return(-1);
00063     }
00064
00065 // Compter le nombre de ligne pour eviter les boucles infinies sur mon lecture debutant vers la
00066 fin
00067     nb_ligne = 0;
00068     ligne = calloc(LIGNE , sizeof(char));
00069     while( !(feof(ptr_Fic)) )
00070     {
00071         fgets(ligne , LIGNE , ptr_Fic);
00072         nb_ligne++;
00073     }
00074     rewind(ptr_Fic);
00075     limite = false;
00076     // Ouverture a la ligne desire
00077     position = 0;
00078     if(num_ligne <= 0)
00079     {
00080         num_ligne *= (-1);
00081         if( num_ligne >= nb_ligne )
00082         {
00083             num_ligne = 1;
00084             limite = true;
00085         }
00086         else if(num_ligne == 0)
00087         {
00088             num_ligne = nb_ligne;
00089         }
00090         else
00091         {
00092             num_ligne = nb_ligne - num_ligne;
00093         }
00094     while( !(feof(ptr_Fic)) && (position != num_ligne) )
00095     {
00096         fgets(ligne , LIGNE , ptr_Fic);
00097         position++;
00098     }
00099     i = 0;
00100     while((ligne[i] != '\n') && ((ligne[i] != '\0')) )
00101     {
00102         printf("%c" , ligne[i]);
00103         emplacement[i] = ligne[i];
00104         i++;
00105     }
00106 }

```

```
00104         }
00105         emplacement[i] = '\0';
00106     }
00107     else
00108     {
00109         while( !(feof(ptr_Fic)) && (position != num_ligne) )
00110     {
00111         fgets(ligne , LIGNE , ptr_Fic);
00112         position++;
00113     }
00114     i = 0;
00115     while((ligne[i] != '\n') && ((ligne[i] != '\0')) )
00116     {
00117         printf("%c" , ligne[i]);
00118         emplacement[i] = ligne[i];
00119         i++;
00120     }
00121     emplacement[i] = '\0';
00122 }
00123 fclose(ptr_Fic);
00124 free(ligne);
00125
00126 if( limite )
00127 {
00128     return (2);
00129 }
00130
00131 return (1);
00132 }
00133
00134 void insert_fin(char *nom_fichier, char *chaine_caractere)
00135 {
00136     FILE *ptr_Fic;
00137
00138     ptr_Fic = NULL;
00139     ptr_Fic = fopen(nom_fichier, "a");
00140     if(ptr_Fic == NULL)
00141     {
00142         exit(-1);
00143     }
00144     fprintf(ptr_Fic, "%s", chaine_caractere);
00145
00146     fclose(ptr_Fic);
00147 }
00148
00149 void head_fichier(char *nom_fichier , int max)
00150 {
00151     FILE *ptr_Fic;
00152     char *ligne;
00153     int num_ligne;
00154
00155     num_ligne = 0;
00156     ptr_Fic = NULL;
00157     ptr_Fic = fopen(nom_fichier, "r");
00158
00159     ligne = calloc(LIGNE, sizeof(char));
00160     if(ptr_Fic == NULL)
00161     {
00162         printf("%s n'existe pas ; si non vous n'avez pas le droit de voir ce qu'il y a dedans\n",
00163             nom_fichier);
00164         return;
00165     }
00166     while( !(feof(ptr_Fic)) && (num_ligne <= max) )
00167     {
00168         fgets(ligne, LIGNE, ptr_Fic);
00169         num_ligne++;
00170         printf("%s", ligne);
00171     }
00172     free(ligne);
00173 }
00174
00175 void tail_fichier(char *nom_fichier , int min)
00176 {
00177     FILE *ptr_Fic;
00178     char *ligne;
00179     long num_ligne , nb_ligne , pos_actuel;
00180
00181     ptr_Fic = NULL;
00182     ptr_Fic = fopen(nom_fichier, "r");
00183
00184     ligne = calloc(LIGNE, sizeof(char));
00185     if(ptr_Fic == NULL)
00186     {
00187         printf("%s n'existe pas ; si non vous n'avez pas le droit de voir ce qu'il y a dedans\n",
00188             nom_fichier);
00189         return;
00190     }
```

```

00189     nb_ligne = 0;
00190     pos_actuel = 0;
00191     while( !(feof(ptr_Fic)) )
00192     {
00193         caractere = fgetc(ptr_Fic);
00194         pos_actuel++;
00195         if(caractere == '\n')
00196         {
00197             nb_ligne++;
00198         }
00199     }
00200     fseek(ptr_Fic , -1 , SEEK_END);
00201     // Deplacer vers la ligne N° min
00202     num_ligne = 0;
00203     while(num_ligne != min)
00204     {
00205         if(pos_actuel == 0)
00206         {
00207             break;
00208         }
00209         caractere = fgetc(ptr_Fic);
00210         if(caractere == '\n')
00211         {
00212             num_ligne++;
00213         }
00214         pos_actuel--;
00215         fseek(ptr_Fic , -2 , SEEK_CUR);
00216     }
00217     if(pos_actuel != 0)
00218     {
00219         fgets(ligne, LIGNE, ptr_Fic);
00220     }
00221     while( !(feof(ptr_Fic)) )
00222     {
00223         fgets(ligne, LIGNE, ptr_Fic);
00224         printf("%s", ligne);
00225     }
00226     printf("\n");
00227     free(ligne);
00228 }
00229 int lire_histoire()
00230 {
00231     FILE *hist;
00232     char ligne[100];
00233     hist = fopen("../data/histoire.txt" , "r");
00234
00235     if(hist == NULL)
00236     {
00237         return (-1);
00238     }
00239     if(feof(hist))
00240     {
00241         return (-2);
00242     }
00243     else
00244     {
00245         while( feof(hist) == 0 )
00246         {
00247             fgets(ligne , 100 , hist);
00248             printf("%s" , ligne);
00249         }
00250     }
00251
00252     fclose(hist);
00253     return (1);
00254 }
00255 void voir_indice(Mission *mission)
00256 {
00257     int valideite , retour , i;
00258     char rep;
00259     char reponse[7];
00260
00261     strcat(reponse , "YyNnOo");
00262     printf("Voir de l'indice decremente votre score actuel !\n");
00263     valideite = 0;
00264     do
00265     {
00266         printf("Veuillez valider l'action ? [Y/N]");
00267         retour = scanf("%c" , &rep);
00268         // vider les inpuretés pour eviter les erreurs du prochain scanf ou autre entree
00269         while(getchar() != '\n' );
00270         for(i=0 ; i<7 ; i++)
00271         {
00272             if(rep == reponse[i])
00273             {
00274                 valideite = 1;
00275             }

```

```

00276         }
00277     }
00278     while( ( valide != 1 ) || ( retour != 1 ) );
00279
00280     if(rep == 'Y' || rep == 'y' || rep == 'O' || rep == 'o')
00281     {
00282         valide = decrement_score(1);
00283
00284         if(validite < 0)
00285         {
00286             printf("Votre score est insuffisante !\n");
00287         }
00288         else
00289         {
00290             printf("%s\n" , mission->indice);
00291         }
00292     }
00293 }
00294
00295 void qui_est_meilleur(int n_miss)
00296 {
00297     FILE *record;
00298     char *ligne , *nom , *ndj;
00299     int reussite , num_mission , i;
00300     char tmp[10];
00301     double temps , min_trouve;
00302
00303     record = fopen("../save/record.txt" , "r");
00304     min_trouve = 1000;
00305     if(record == NULL)
00306     {
00307         fprintf(stderr , "Record inexistant ! \n");
00308     }
00309     else
00310     {
00311         ligne = calloc(LIGNE , sizeof(char));
00312         nom = calloc(MOT , sizeof(char));
00313         ndj = calloc(MOT , sizeof(char));
00314         while( !(feof(record)) )
00315         {
00316             fgets(ligne , LIGNE , record);
00317             sscanf(ligne , "%d\t%d\t%s\t%s\n" , &num_mission , &reussite , tmp , nom );
00318             i = 0;
00319             while(tmp[i] != ',' && tmp[i] != '\0')
00320             {
00321                 i++;
00322             }
00323             if(tmp[i] != '\0')
00324             {
00325                 tmp[i] = '.';
00326             }
00327             temps = atof(tmp);
00328             if(num_mission == n_miss)
00329             {
00330                 if(temps < min_trouve)
00331                 {
00332                     min_trouve = temps;
00333                     strcpy(ndj , nom);
00334                 }
00335             }
00336         }
00337
00338         if((ndj[0] != '\0') && (min_trouve != 1000))
00339         {
00340             printf("La meilleur pour la mission %d est %s \n" , n_miss , ndj);
00341         }
00342         else
00343         {
00344             printf("Score pas encore batu pour la mission n°%d .\n" , n_miss);
00345         }
00346     }
00347     fclose(record);
00348     free(ligne);
00349     free(ndj);
00350     free(nom);
00351 }
00352 }
00353 int my_man(char *nom_commande)
00354 {
00355     FILE *la_page;
00356     char *chaine_char;
00357
00358     chaine_char = calloc(LIGNE , sizeof(char));
00359
00360     sprintf(chaine_char , "../data/page_man/aide_%s.txt" , nom_commande);
00361     la_page = fopen(chaine_char , "r");
00362     if(la_page == NULL)
00363     {

```

```

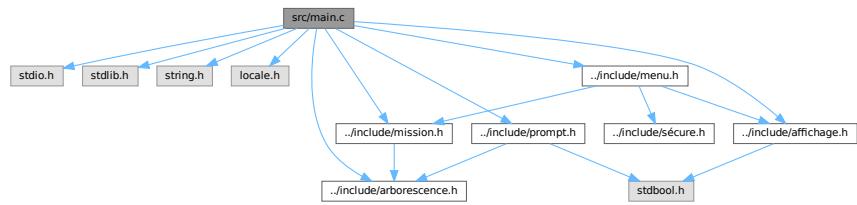
00364     printf("Aucune page d'aide pour %s\n" , nom_commande);
00365     return (-1);
00366 }
00367
00368 while( !feof(la_page) )
00369 {
00370     fgets(chaine_char , LIGNE , la_page);
00371     printf("%s" , chaine_char);
00372 }
00373
00374 free(chaine_char);
00375 fclose(la_page);
00376 return (1);
00377 }
00378 int my_history(bool effacer)
00379 {
00380     FILE *histoire;
00381     char *nom , *emplacement;
00382     int taille ;
00383
00384 // Vision du fichier d'historique de commande
00385 nom = getenv("USER");
00386 taille = 23;
00387 taille += strlen(nom);
00388 emplacement = calloc(taille , sizeof(char));
00389 sprintf(emplacement , "../save/%s/histoire.txt" , nom);
00390 histoire = fopen(emplacement , "r");
00391 if(histoire == NULL)
00392 {
00393     printf("Histoire : vide !\n");
00394     free(emplacement);
00395     return (-1);
00396 }
00397 else
00398 {
00399     fclose(histoire);
00400 }
00401
00402
00403 // vider le fichier de histoire
00404 if(effacer)
00405 {
00406     remove(emplacement);
00407 }
00408 else
00409 {
00410     cat_fichier(emplacement , true);
00411 }
00412
00413 free(emplacement);
00414
00415 return (1);
00416 }
```

## 4.21 src/main.c File Reference

: Fonction principale du jeu Bash Commando

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include "../include/arborescence.h"
#include "../include/affichage.h"
#include "../include/mission.h"
#include "../include/menu.h"
#include "../include/prompt.h"
```

Include dependency graph for main.c:



## Functions

- int [main \(\)](#)

### 4.21.1 Detailed Description

: Fonction principale du jeu Bash Commando

#### Author

: RAKOTOARIVONY Zo Mamitiana Olivier

#### Date

: 03 déc 2025

Definition in file [main.c](#).

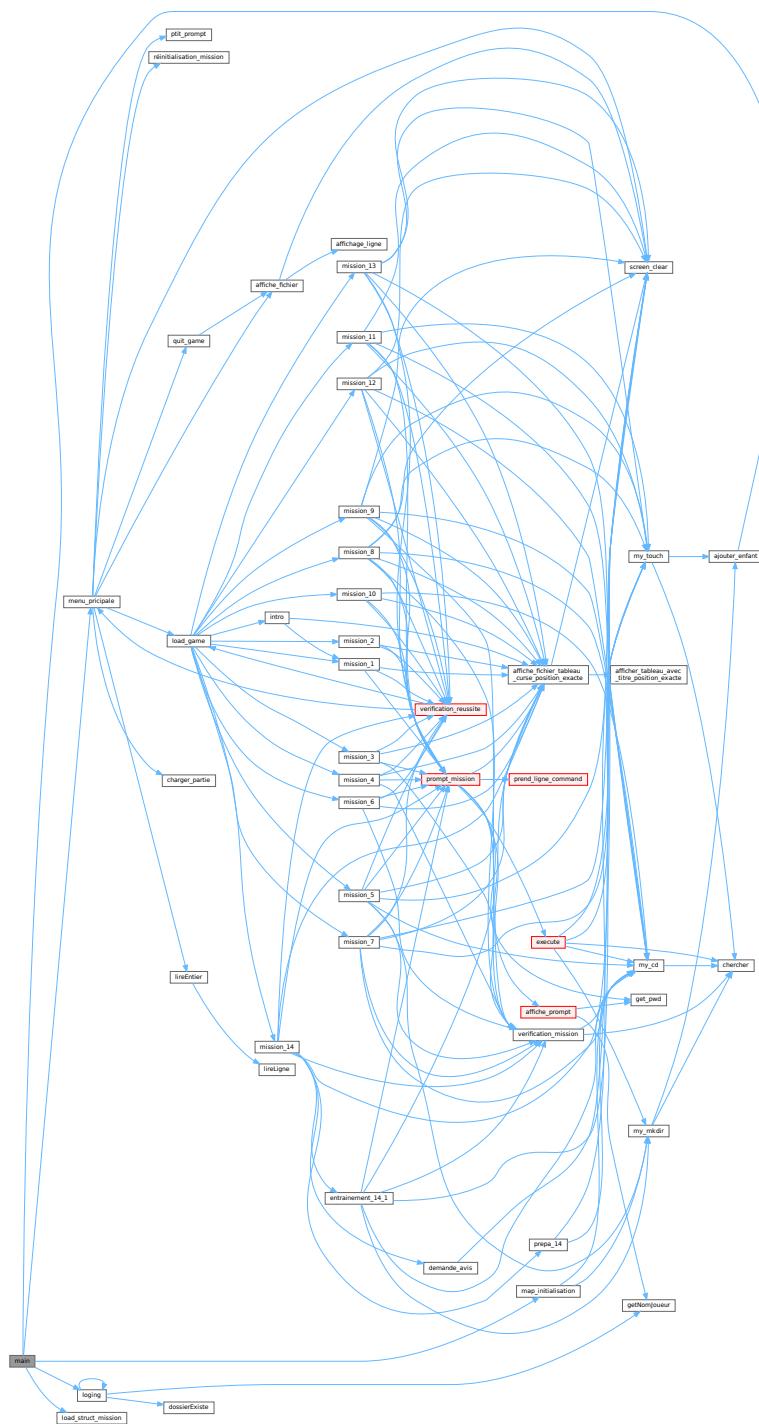
### 4.21.2 Function Documentation

#### 4.21.2.1 [main\(\)](#)

```
int main ( )
```

Definition at line 13 of file [main.c](#).

Here is the call graph for this function:



## 4.22 main.c

[Go to the documentation of this file.](#)

```

00001
00002 #include <stdio.h>
00003 #include <stdlib.h>
00004 #include <string.h>
```

```

00005 #include <locale.h> // Pour les accents (setlocale)
00006 #include "../include/arborescence.h"
00007 #include "../include/affichage.h"
00008 #include "../include/mission.h"
00009 #include "../include/menu.h"
00010 #include "../include/prompt.h"
00011
00012
00013 int main()
00014 {
00015
00016     Mission *mission = NULL ;
00017     fichier *racine = NULL ;
00018
00019     setlocale(LC_ALL, "");
00020     racine = creer(NULL , "AK-07" , 1 , 1 ); // création du racine
00021     mission = malloc( NOMBRE_MISSION * sizeof(Mission) ); // Allocation du structure mission
00022
00023     loging();
00024     load_struct_mission(mission) ; // Chargement du structure mission
00025     map_initialisation( mission , racine ) ; // Construction du map du jeu
00026
00027     menu_principale(mission , racine );
00028
00029     free(mission); // liberation du mémoire alloué au struct mission
00030
00031     return 0 ;
00032
00033 }

```

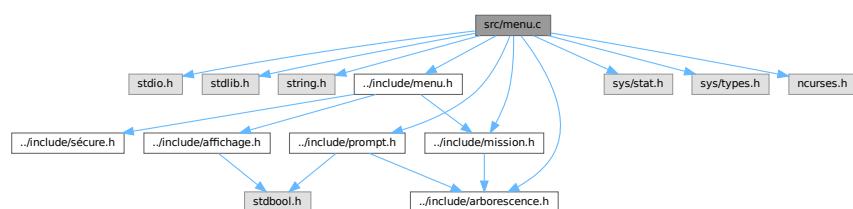
## 4.23 src/menu.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../include/menu.h"
#include "../include/arborescence.h"
#include "../include/prompt.h"
#include "../include/mission.h"
#include <sys/stat.h>
#include <sys/types.h>
#include <ncurses.h>

```

Include dependency graph for menu.c:



### Macros

- #define LARGEUR 70

### Functions

- int dossierExiste (const char \*chemin)
- void loging ()

*Fonction pour gérer la connexion de l'utilisateur.*

- void `load_game` (`Mission` \*mission, int rang\_du\_dernier\_mission, `fichier` \*racine)
- int `charger_partie` (`Mission` \*mission)
- void `quit_game` (`Mission` \*mission)
- void `prompt` (char \*nom)
- void `menu_principale` (`Mission` \*mission, `fichier` \*racine)

## 4.23.1 Macro Definition Documentation

### 4.23.1.1 LARGEUR

```
#define LARGEUR 70
```

Definition at line 12 of file `menu.c`.

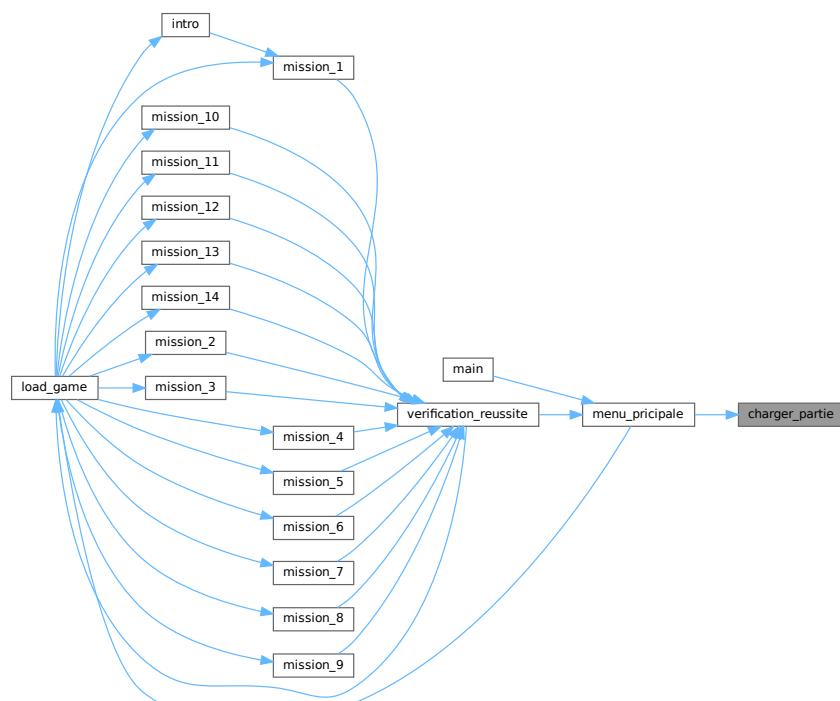
## 4.23.2 Function Documentation

### 4.23.2.1 charger\_partie()

```
int charger_partie (
    Mission * mission )
```

Definition at line 112 of file `menu.c`.

Here is the caller graph for this function:



#### 4.23.2.2 dossierExiste()

```
int dossierExiste (
    const char * chemin )
```

Definition at line 18 of file [menu.c](#).

Here is the caller graph for this function:

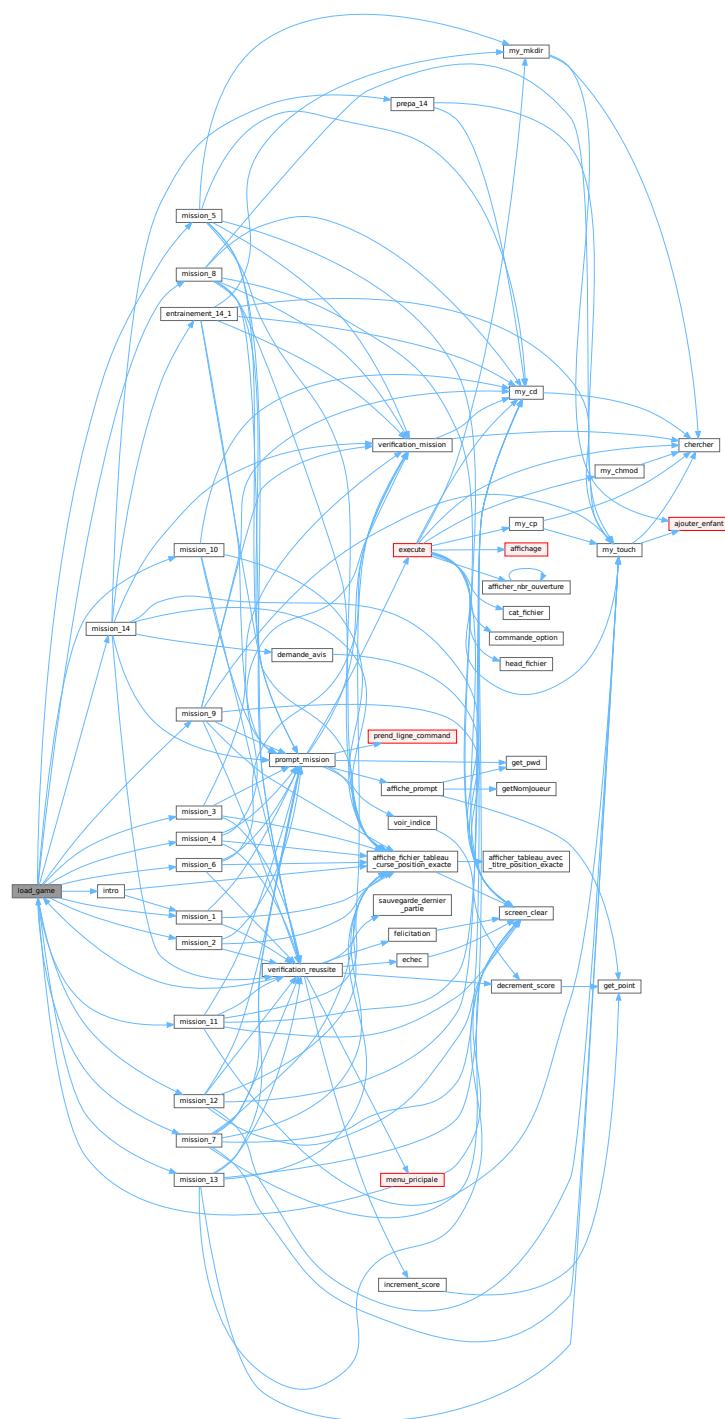


#### 4.23.2.3 load\_game()

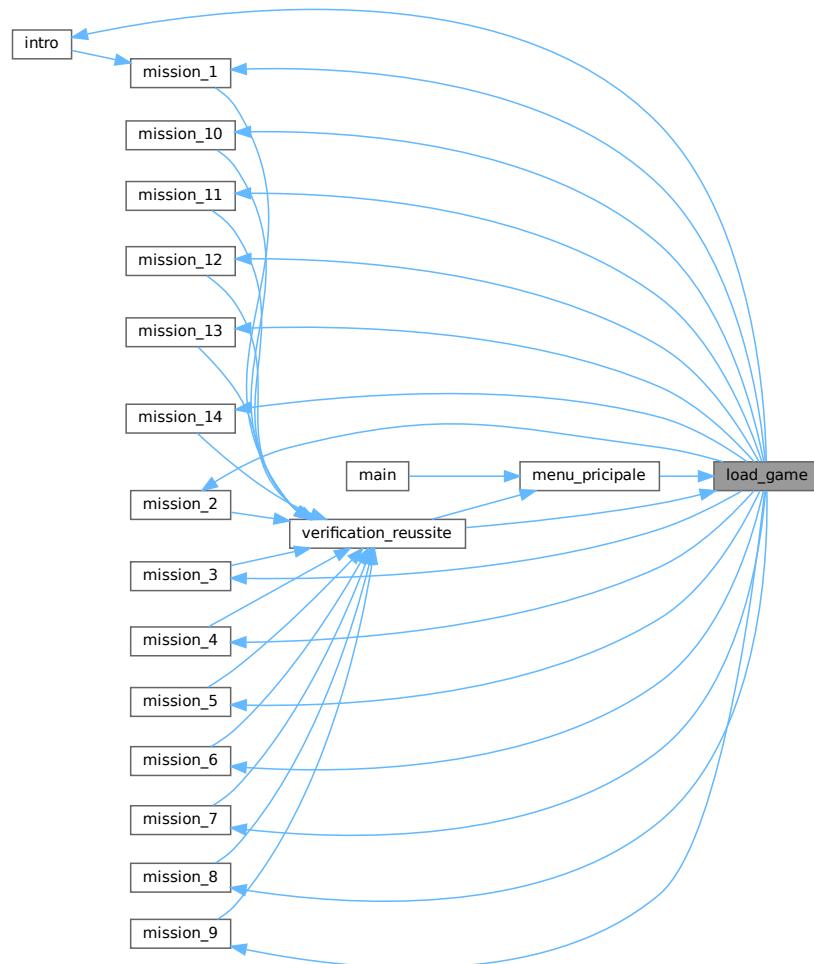
```
void load_game (
    Mission * mission,
    int rang_du_dernier_mission,
    fichier * racine )
```

Definition at line 95 of file [menu.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.23.2.4 logging()

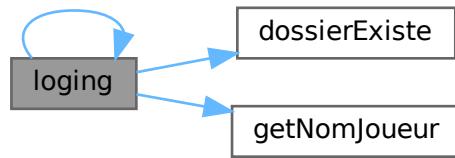
```
void logging ( )
```

Fonction pour gérer la connexion de l'utilisateur.

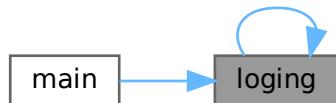
: Fonction qui récupère le logging du joueur et accorde le sauvegarde qui lui est attribué s'il a déjà jouer si non crée un nouveau dossier de sauvegarde pour l'user

Definition at line 36 of file [menu.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

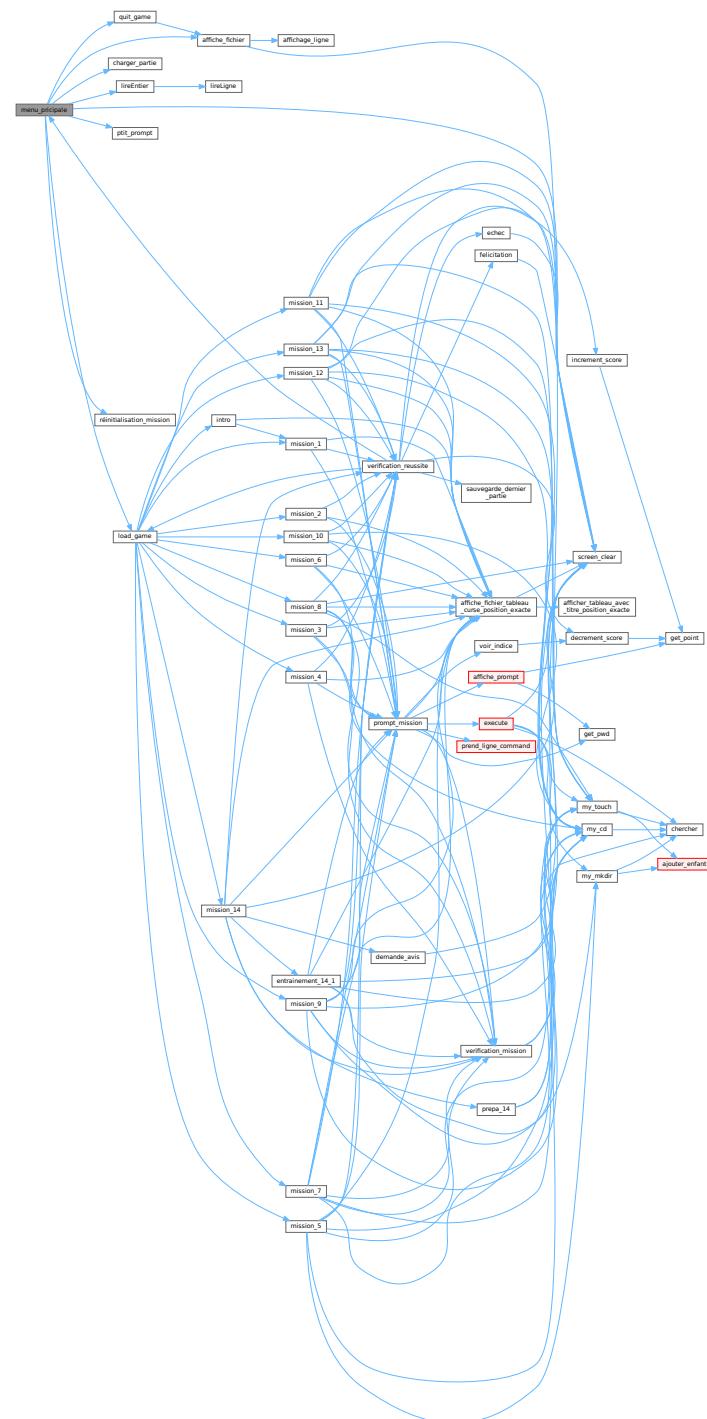


#### 4.23.2.5 menu\_principale()

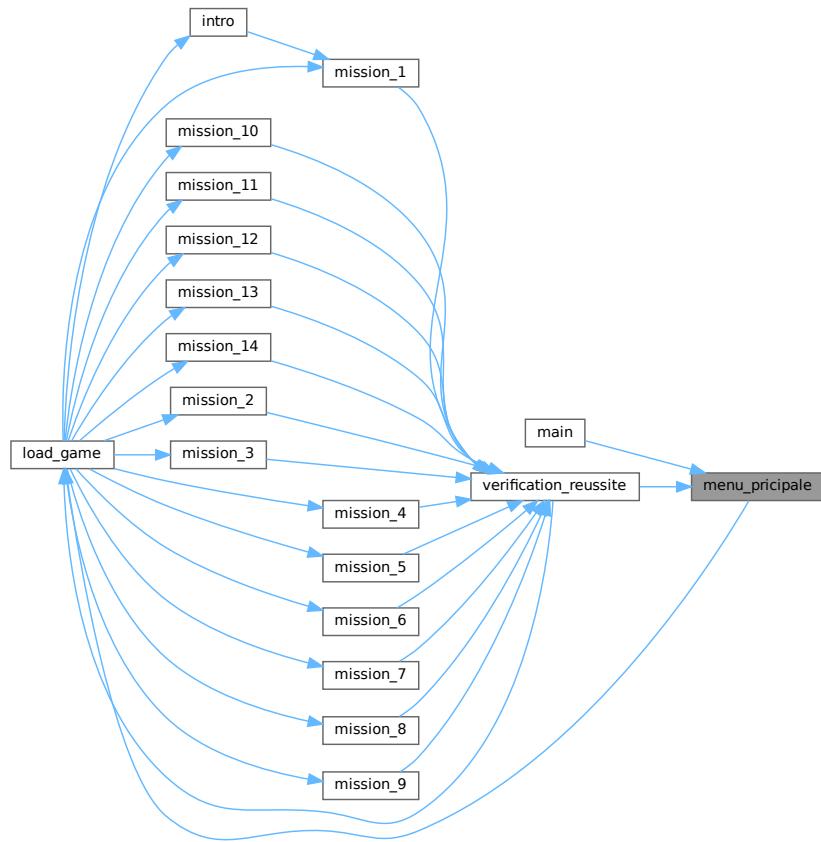
```
void menu_principale (
    Mission * mission,
    fichier * racine )
```

Definition at line [261](#) of file [menu.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

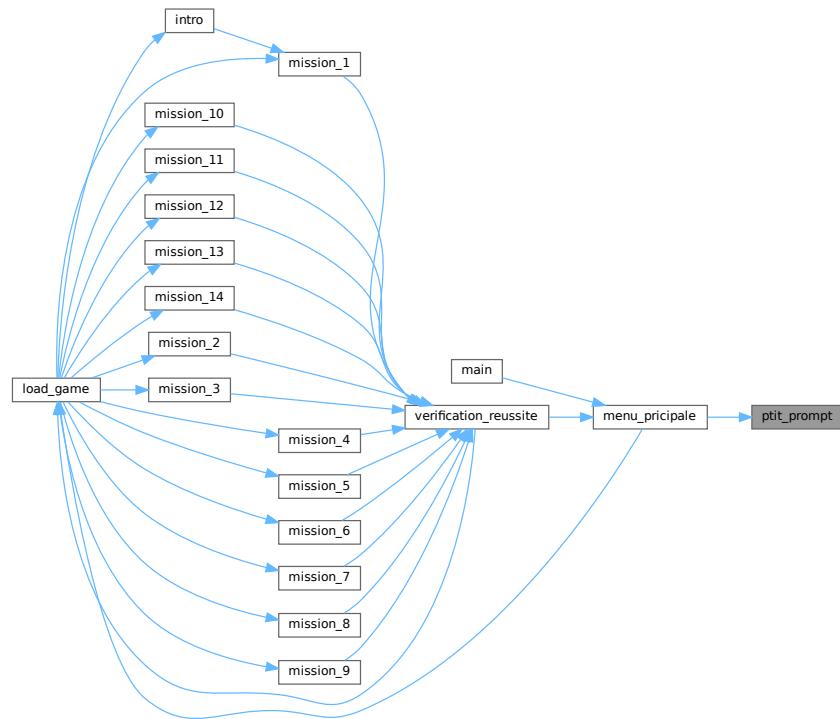


#### 4.23.2.6 ptit\_prompt()

```
void ptit_prompt (
    char * nom )
```

Definition at line 240 of file [menu.c](#).

Here is the caller graph for this function:

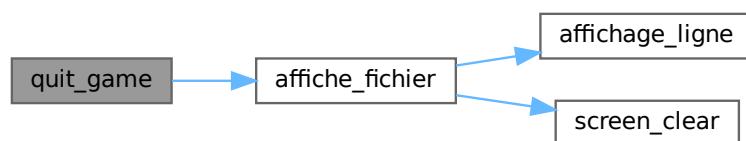


#### 4.23.2.7 quit\_game()

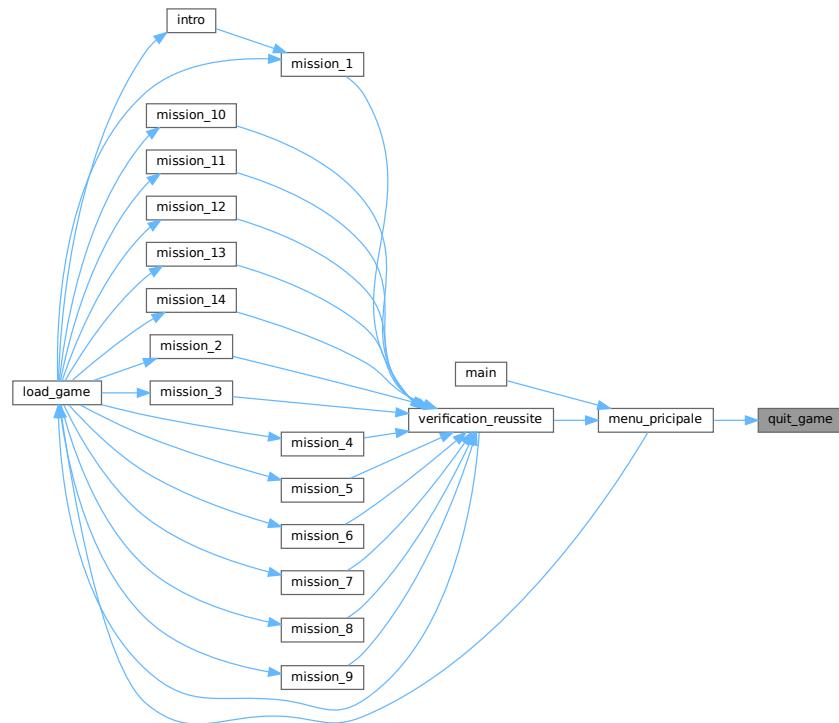
```
void quit_game (
    Mission * mission )
```

Definition at line 229 of file [menu.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.24 menu.c

[Go to the documentation of this file.](#)

```

00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <string.h>
00004 #include "../include/menu.h"
00005 #include "../include/arborescence.h"
00006 #include "../include/prompt.h"
00007 #include "../include/mission.h"
00008 #include <sys/stat.h>
00009 #include <sys/types.h>
00010 #include <ncurses.h>
00011
00012 #define LARGEUR 70 // Largeur du tableau curses
00018 int dossierExiste(const char *chemin)
00019 {
00020     struct stat info ;
00021
00022     if (stat(chemin , &info) != 0 ) // Si le dossier n'existe pas
00023     {
00024         return 0;
00025     }
00026
00027     // S_ISDIR vérifie que c'est bien un dossier
00028     return S_ISDIR(info.st_mode) ;
00029 }
00030
00036 void loging()
00037 {
00038     char *logging = NULL ;
00039     FILE *fichier_temporaire = NULL ;
00040     char *chemin = calloc(50 , sizeof(char));
00041
00042     fichier_temporaire = fopen("../save/temp.txt" , "w+");
00043
00044     if ( !fichier_temporaire )
00045     {

```

```

00046     printf("Erreur : Fichier temporaire introuvable\n");
00047     exit(0);
00048 }
00049
00050 loging = calloc(50 , sizeof(char));
00051 // Erreur de recuperation du loging
00052 if( getNomJoueur(loging) == -1)
00053 {
00054     printf("Erreur : Impossible de récupérer le loging !!!\n");
00055     exit(0);
00056 }
00057
00058 fprintf(fichier_temporaire , "../save/%s", loging);
00059 rewind(fichier_temporaire);
00060 fscanf( fichier_temporaire , "%s" , chemin );
00061
00062 // Verification si le joueuere possede deja un sauvegarde si no cr閑
00063 if ( dossierExiste(chemin) == 0 )
00064 {
00065     if ( mkdir(chemin , 0755 ) != 0 )
00066     {
00067         printf("Erreur : Impossible de cr閑 le dossier.\n");
00068     }
00069 }
00070
00071 // ecriture du chemin de level.txt dans temp.txt
00072 rewind(fichier_temporaire);
00073 fprintf(fichier_temporaire , "%s/level.txt\n", chemin );
00074
00075 // Ecriture du chemin de mission.bin dans temp.txt
00076 fprintf(fichier_temporaire , "%s/mission.bin\n", chemin );
00077
00078 // Ecriture du chemin de point
00079 fprintf(fichier_temporaire , "%s/pTSci", chemin );
00080
00081 free(loging);
00082 fclose(fichier_temporaire);
00083
00084 }
00085
00086
00087 //-----
00088
00089 void load_game(Mission *mission , int rang_du_dernier_mission , fichier *racine)
00090 {
00091     // Cr閏ation du tableau de pointeurs pointant sur les fonctions mission
00092     void (*fonctions_mission[])(Mission * , fichier *) = { intro , mission_1 , mission_2 , mission_3
00093 , mission_4 , mission_5 , mission_6 , mission_7 , mission_8 , mission_9 , mission_10
00094 , mission_11 , mission_12 , mission_13 , mission_14 };
00095
00096     // Appel du mission du d閞nier mission ;
00097     fonctions_mission[rang_du_dernier_mission]( mission , racine );
00098 }
00099
00100
00101 //-----
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112 int charger_partie(Mission *mission)
00113 {
00114     int i = 0 , nb_ligne , j = 0 , nbMot = 0 , MAX = 0 ,
00115     touch = 0 , hauteur = 0 , largeur = 0 , posix = 0 , posy = 0 , choix = 0;
00116     char TAB[NOMBRE_MISSION][50] ;
00117
00118     FILE *fichier=fopen( "../data/selection_partie.txt" , "r" );
00119
00120     while( i < NOMBRE_MISSION && mission[i].debloqu  == 1 && fgets(TAB[i] , 50 , fichier) )
00121     {
00122         nbMot = strlen(TAB[i]);
00123
00124         if ( TAB[i][nbMot-1]== '\n' )
00125         {
00126             TAB[i][nbMot-1] = '\0';
00127         }
00128
00129         if ( nbMot > MAX )
00130         {
00131             MAX = nbMot ;
00132         }
00133
00134         i++ ;
00135
00136         nb_ligne = i ;
00137     }
00138
00139     fclose(fichier);
00140
00141 // Pour ajouter un peu de marge au tableau

```

```

00142     hauteur = i + 6 ;
00143     largeur = MAX ;
00144 // Pour trouver le centre du terminal
00145     posix = (COLS - LARGEUR) / 2 ;
00146     posy = (LINES - i) / 2 ;
00147
00148 // Cr ation du tableau win
00149     WINDOW *win = newwin(hauteur , LARGEUR , posy , posix ) ;
00150
00151 // Config curses
00152     keypad(win , true );
00153     curs_set(0);
00154     noecho();
00155
00156 while(1)
00157 {
00158     // Effacer la fen tre (pas l' cran principal)
00159     werase(win);
00160     wborder(win, '|', '|', '-', '-', '+', '+', '+', '+');
00161
00162     // Affichage Selection partie
00163     wattron(win, COLOR_RED | A_BOLD);
00164     mvwprintw(win, 1, (LARGEUR - strlen("SELECTION PARTIE")) / 2, "SELECTION PARTIE");
00165     wattroff(win, COLOR_RED | A_BOLD);
00166
00167     // Affichage des missions d bloqu es
00168     for (j = 0 ; j < nb_ligne ; j++)
00169     {
00170         if(j == choix)
00171         {
00172             wattron(win, COLOR_GREEN | A_BOLD);
00173             mvwprintw(win, j + 3, 2 , "=> %s", TAB[j]);
00174             wattroff(win, COLOR_GREEN | A_BOLD);
00175         }
00176         else
00177         {
00178             mvwprintw(win, j + 3, 6, "%s", TAB[j]);
00179         }
00180     }
00181
00182     // Instructions
00183     mvwprintw(win, hauteur - 2 , 2, "Utilisez les fl ches et appuyez sur ENTR E ou q ");
00184
00185     wrefresh(win); // Rafra chir la fen tre
00186
00187     touch = wgetch(win);
00188
00189     if (touch == KEY_UP && choix > 0)
00190     {
00191         choix-- ;
00192     }
00193     else if(touch == KEY_DOWN && choix < nb_ligne - 1)
00194     {
00195         choix++ ;
00196     }
00197     else if( touch == KEY_UP && choix == 0 )
00198     {
00199         choix = nb_ligne - 1 ;
00200     }
00201     else if( touch == KEY_DOWN && choix == nb_ligne - 1 )
00202     {
00203         choix = 0 ;
00204     }
00205     else if (touch == '\n' || touch == KEY_ENTER)
00206     {
00207         delwin(win);
00208         clear();
00209         refresh();
00210         return choix ; // Retourne le rang de la mission choisie
00211     }
00212     else if (touch == 27 || touch == 'q') // ESC ou 'q' pour quitter
00213     {
00214         delwin(win);
00215         clear();
00216         refresh();
00217         return -1; // Indiquer que l'utilisateur a annul 
00218     }
00219 }
00220
00221 }
00222
00223 //-----
00229 void quit_game( Mission *mission )
00230 {
00231     printf("\033[1;31m");
00232     affiche_fichier("../ascii/quit.txt");
00233     printf("\033[0m");

```

```
00234
00235     free(mission); // libération du mémoire alloué au struct mission
00236     exit (0) ;
00237 }
00238
00239
00240 void ptit_prompt(char *nom)
00241 {
00242
00243     char *user = getenv("USER");
00244
00245     if( strcmp( "USER" , nom ) == 0 )
00246     {
00247         printf("\n\t==> \033[1;33m%s\033[0m\033[1;34m@\033[0m\033[1;31mBash-Commando:\033[0m ", user);
00248     }
00249     else
00250     {
00251         printf("\n\t==> \033[1;33m%s\033[0m\033[1;34m@\033[0m\033[1;31mBash-Commando:\033[0m ", nom);
00252     }
00253 }
00254
00255
00256 void menu_principale(Mission *mission , fichier *racine )
00257 {
00258
00259     int choix = 0 ;
00260     int compteur = 0 ;
00261     FILE *save = NULL ;
00262     int rang_du_dernier_mission = 0 ;
00263     char chemin_level[30] = {"\0"};
00264     char chemin_mission[30] = {"\0"};
00265
00266
00267     while(1)
00268     {
00269         do
00270         {
00271             if (compteur == 0)
00272             {
00273                 affiche_fichier("../ascii/menu.txt" );
00274             }
00275             else
00276             {
00277                 printf("\t%d : Commande introuvable\n", choix);
00278             }
00279
00280             ptit_prompt("USER");
00281             choix=lireEntier();
00282             compteur ++ ;
00283
00284         }while( choix <= 0 || choix > 4 );
00285
00286         compteur = 0 ; // Réinitialisation du compteur a zero
00287         switch (choix)
00288         {
00289             case 1 :
00290                 // Regarde la dernière mission jouer par l'utilisateur
00291                 save = fopen( "../save/temp.txt" , "r" );
00292                 fscanf( save , "%a" , chemin_level );
00293                 fscanf( save , "%s" , chemin_mission );
00294                 fclose(save);
00295                 save = fopen( chemin_level , "r" ) ;
00296
00297                 if( save == NULL )
00298                 {
00299                     // Si le fichier n'existe pas , on commence une nouvelle partie
00300                     load_game( mission , 0 , racine);
00301                 }
00302                 else
00303                 {
00304                     fscanf( save , "%d" , &rang_du_dernier_mission ) ;
00305                     fclose( save ) ;
00306                 }
00307                 load_game( mission , rang_du_dernier_mission , racine);
00308
00309             break;
00310             case 2 :
00311                 screen_clear();
00312                 endwin();
00313                 initscr();
00314                 clear();
00315                 refresh();
00316                 choix = charger_partie(mission);
00317                 if (choix != -1 )
00318                 {
00319                     load_game(mission , choix+1 , racine );
00320                 }
00321                 endwin();
00322             break;
00323             case 3 :
00324                 réinitialisation_mission(mission);
```

```

00327     break;
00328     case 4 :
00329         quit_game( mission );
00330         return ;
00331     break;
00332     default:
00333         printf("Option invalide !!!\n\n");
00334     }
00335 }
00336 }
```

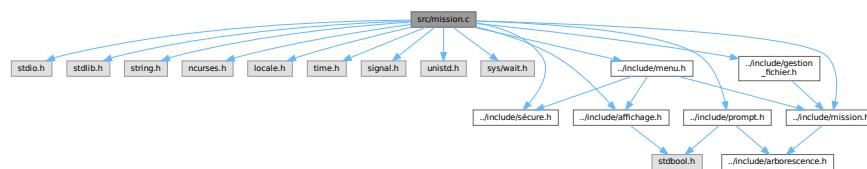
## 4.25 src/mission.c File Reference

: Fonction pour gérer les missions du jeu

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ncurses.h>
#include <locale.h>
#include <time.h>
#include <signal.h>
#include <unistd.h>
#include <sys/wait.h>
#include "../include/sécurité.h"
#include "../include/affichage.h"
#include "../include/gestion_fichier.h"
#include "../include/prompt.h"
#include "../include/mission.h"
#include "../include/menu.h"
```

Include dependency graph for mission.c:



### Functions

- void [map\\_initialisation](#) (Mission \*mission, fichier \*racine)
- void [load\\_struct\\_mission](#) (Mission \*mission)
- void [sauvegarde\\_dernier\\_partie](#) (Mission \*mission, int rang\_mission\_actuel)
- void [réinitialisation\\_mission](#) (Mission mission[ ])
- void [verification\\_reussite](#) (Mission \*mission, fichier \*racine, bool reussite, int num\_mission)
- void [intro](#) (Mission \*mission, fichier \*racine)
- bool [prompt\\_mission](#) (fichier \*racine, char \*commande, char \*titre, char \*chemin, char \*rep\_courant, Mission \*miss\_actuelle, int rang\_mission, int autoverifi)
- void [mission\\_1](#) (Mission \*mission, fichier \*racine)
- void [mission\\_2](#) (Mission \*mission, fichier \*racine)
- bool [verification\\_mission](#) (Mission \*mission, int rang\_mission, fichier \*racine, char \*courant)
- void [mission\\_3](#) (Mission \*mission, fichier \*racine)
- void [mission\\_4](#) (Mission \*mission, fichier \*racine)
- void [mission\\_5](#) (Mission \*mission, fichier \*racine)

- void [mission\\_6](#) ([Mission](#) \*mission, [fichier](#) \*racine)
- void [mission\\_7](#) ([Mission](#) \*mission, [fichier](#) \*racine)
- void [mission\\_8](#) ([Mission](#) \*mission, [fichier](#) \*racine)
- void [mission\\_9](#) ([Mission](#) \*mission, [fichier](#) \*racine)
- void [mission\\_10](#) ([Mission](#) \*mission, [fichier](#) \*racine)
- void [mission\\_11](#) ([Mission](#) \*mission, [fichier](#) \*racine)
- void [mission\\_12](#) ([Mission](#) \*mission, [fichier](#) \*racine)
- void [mission\\_13](#) ([Mission](#) \*mission, [fichier](#) \*racine)
- void [entrainement\\_14\\_1](#) ([Mission](#) \*mission, [fichier](#) \*racine)
- void [prepa\\_14](#) ([fichier](#) \*racine)
- int [demande\\_avis](#) ()
- void [mission\\_14](#) ([Mission](#) \*mission, [fichier](#) \*racine)

#### 4.25.1 Detailed Description

: Fonction pour gérer les missions du jeu

Author

: RAKOTOARIVONY Zo Mamitiana Olivier

Date

: 03 déc 2025

Definition in file [mission.c](#).

#### 4.25.2 Function Documentation

##### 4.25.2.1 demande\_avis()

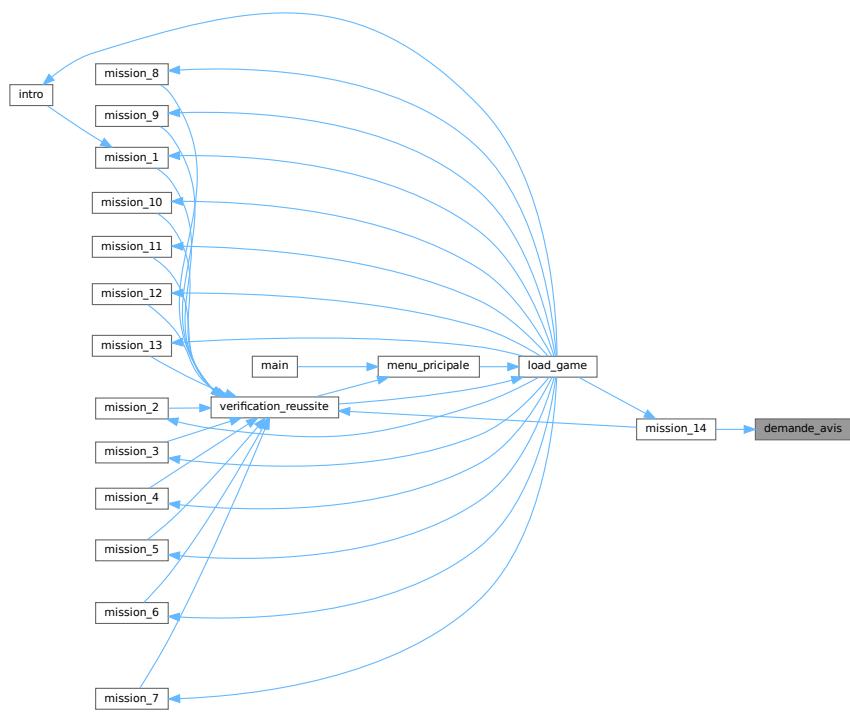
```
int demande_avis ( )
```

Definition at line [1212](#) of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

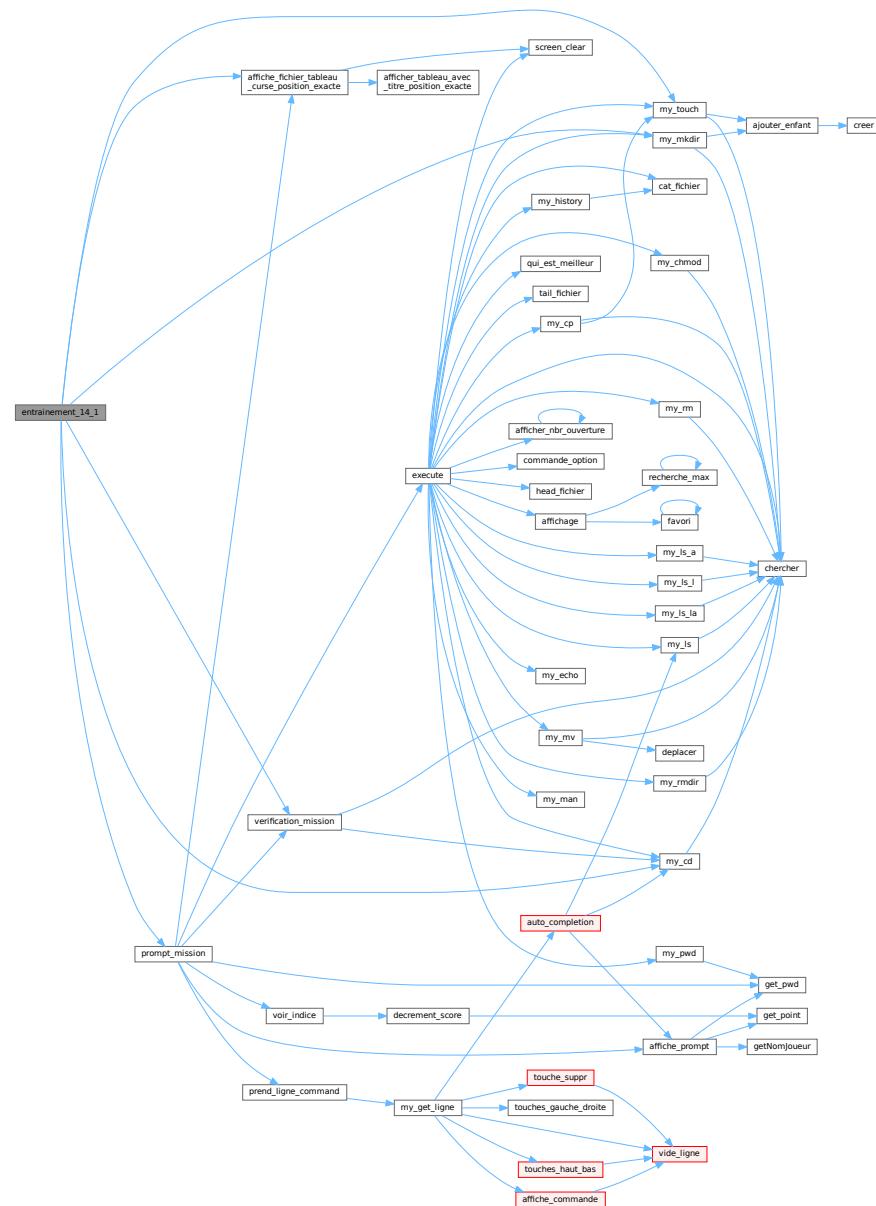


#### 4.25.2.2 entrainement\_14\_1()

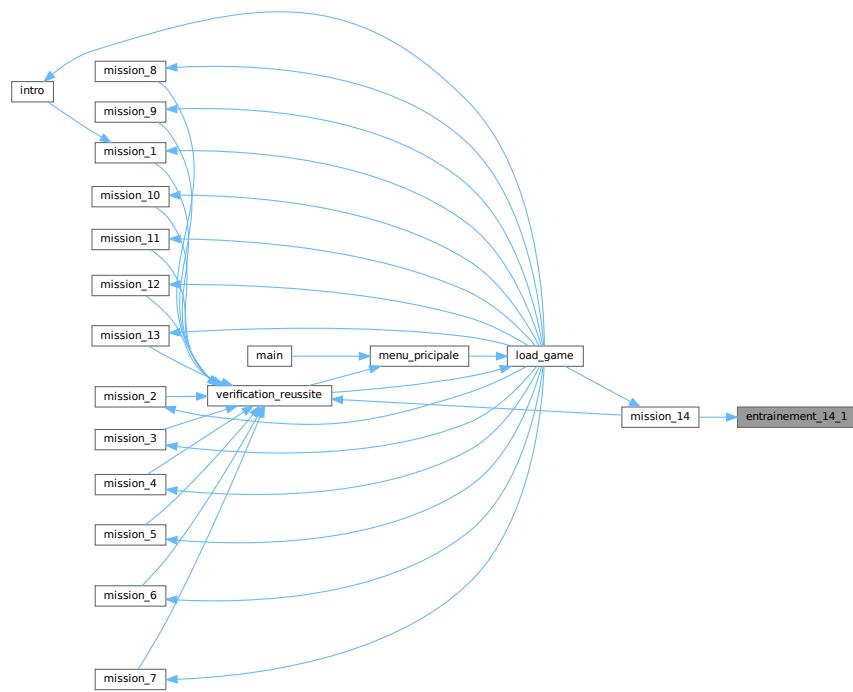
```
void entrainement_14_1 (
    Mission * mission,
    fichier * racine )
```

Definition at line 1157 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

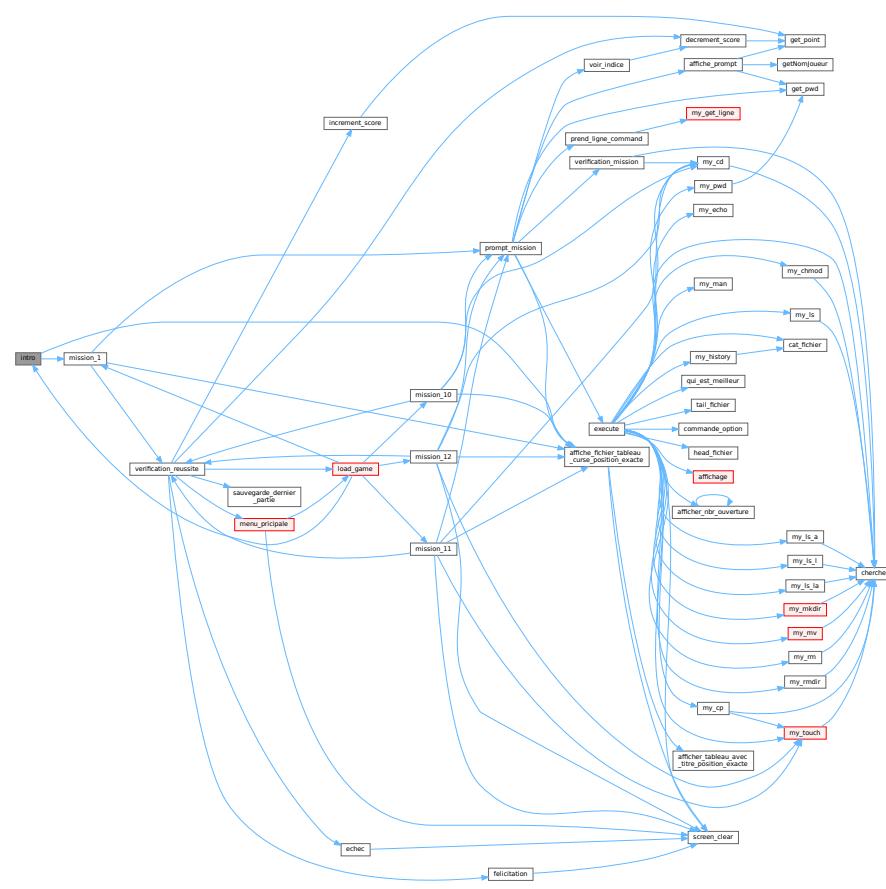


#### 4.25.2.3 `intro()`

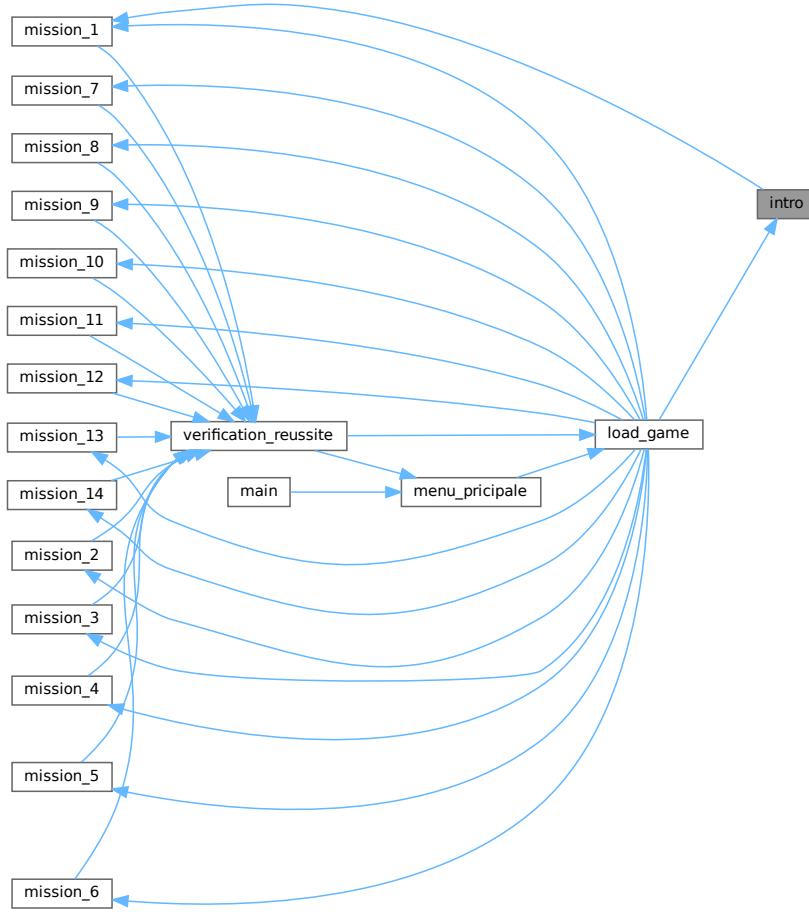
```
void intro (
    Mission * mission,
    fichier * racine )
```

Definition at line 289 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.25.2.4 load\_struct\_mission()

```
void load_struct_mission (
    Mission * mission )
```

Definition at line 80 of file [mission.c](#).

Here is the caller graph for this function:

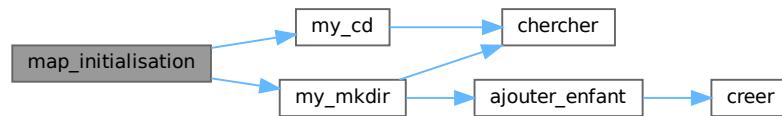


#### 4.25.2.5 map\_initialisation()

```
void map_initialisation (
    Mission * mission,
    fichier * racine )
```

Definition at line 32 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

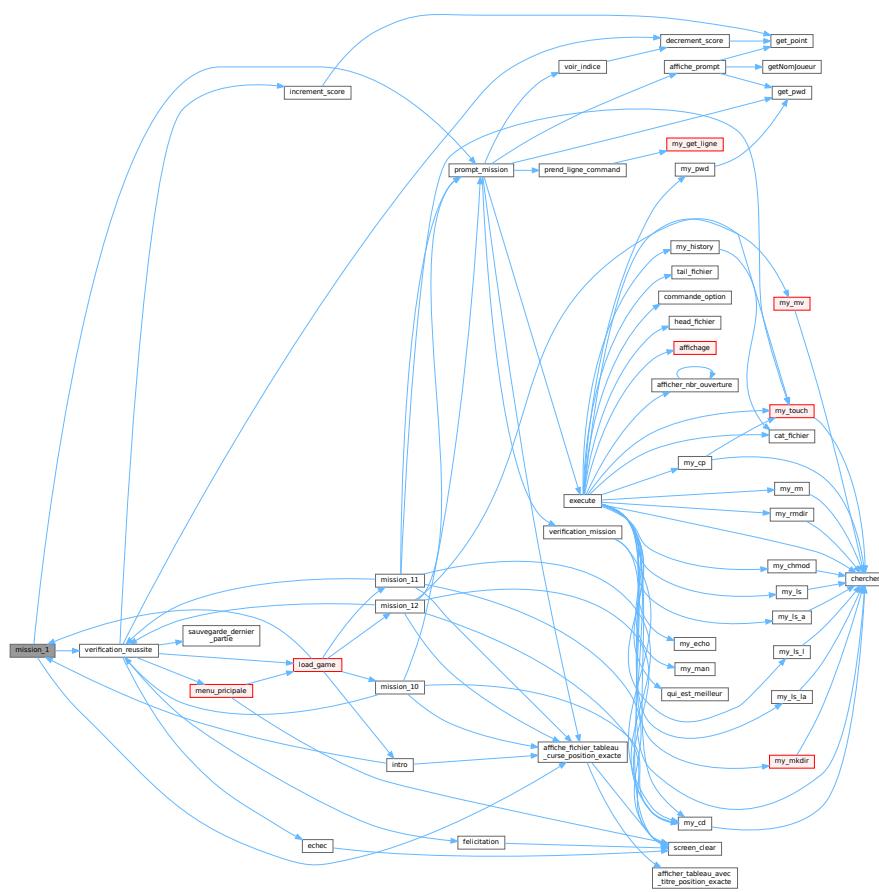


#### 4.25.2.6 mission\_1()

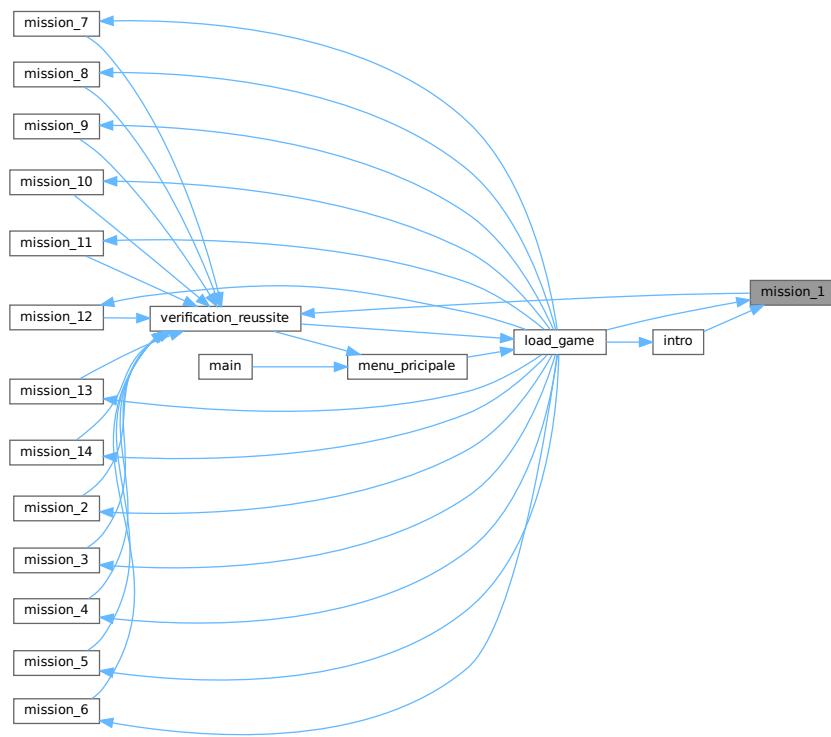
```
void mission_1 (
    Mission * mission,
    fichier * racine )
```

Definition at line 433 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

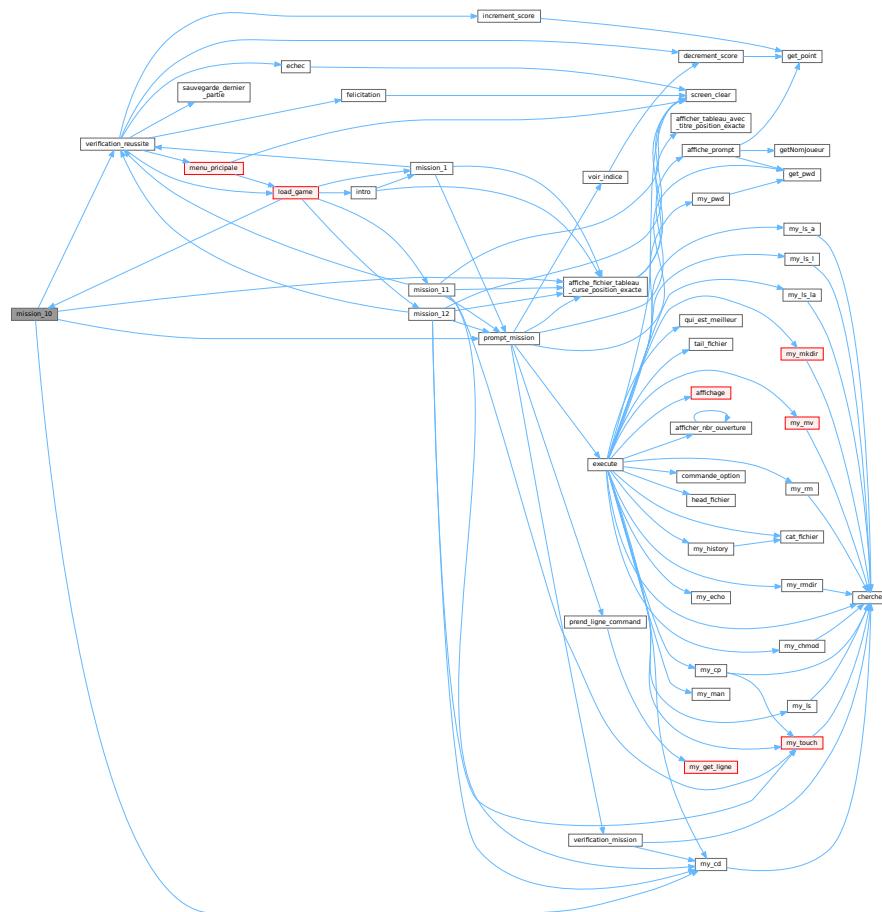


#### 4.25.2.7 mission\_10()

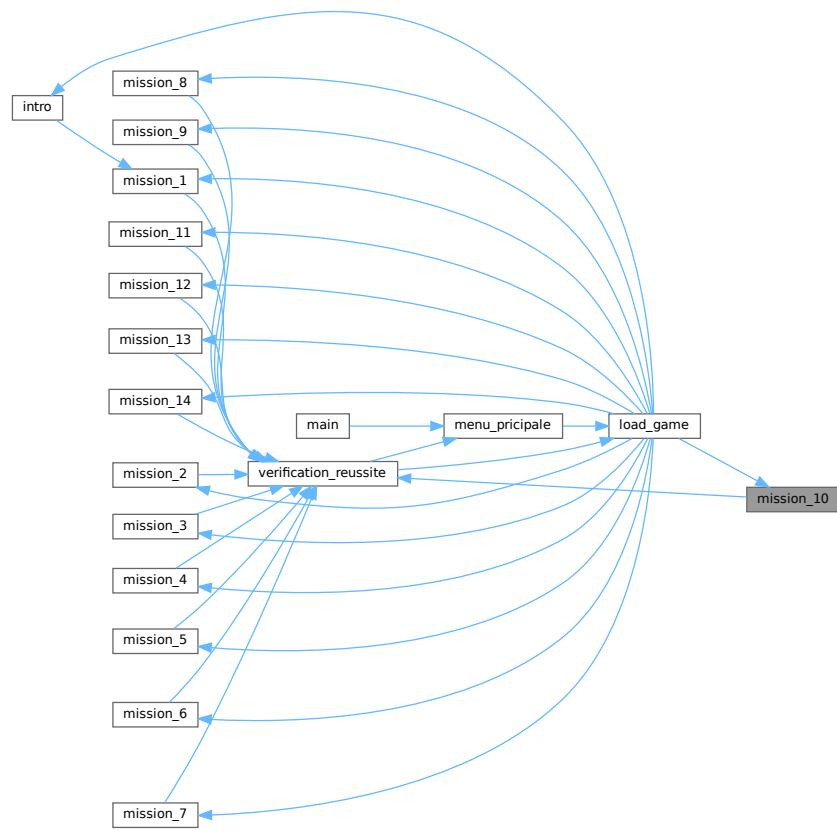
```
void mission_10 (
    Mission * mission,
    fichier * racine )
```

Definition at line 996 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

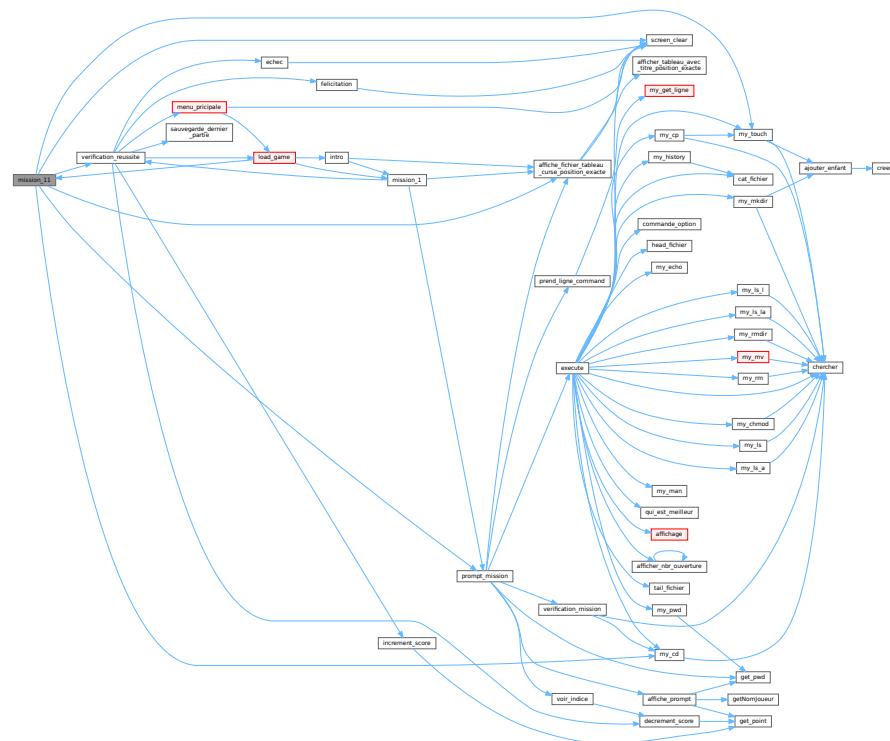


#### 4.25.2.8 mission\_11()

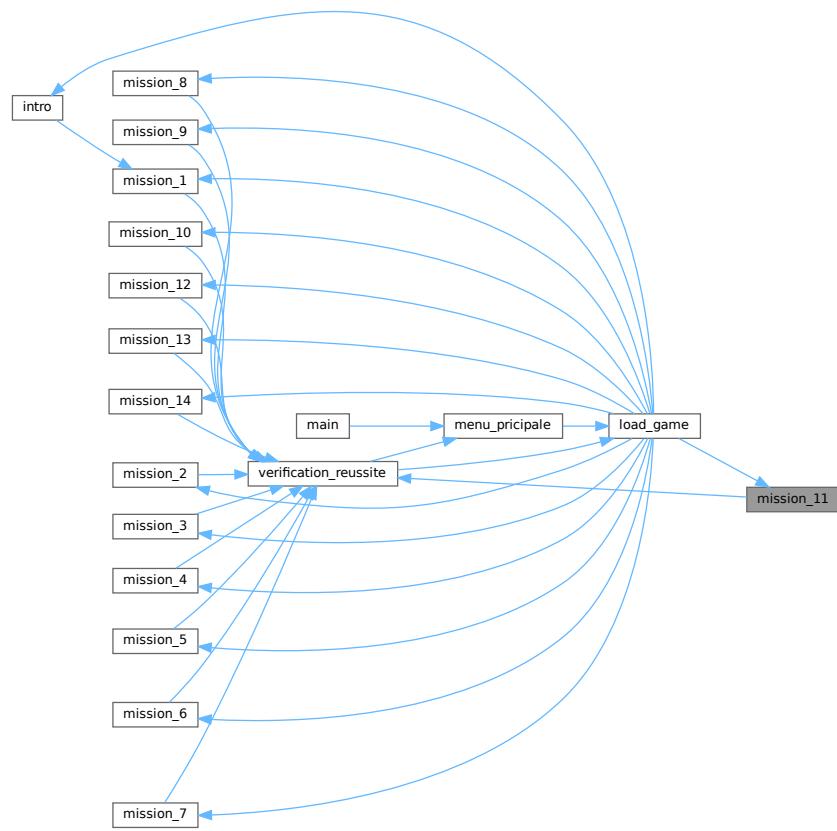
```
void mission_11 (
    Mission * mission,
    fichier * racine )
```

Definition at line 1033 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

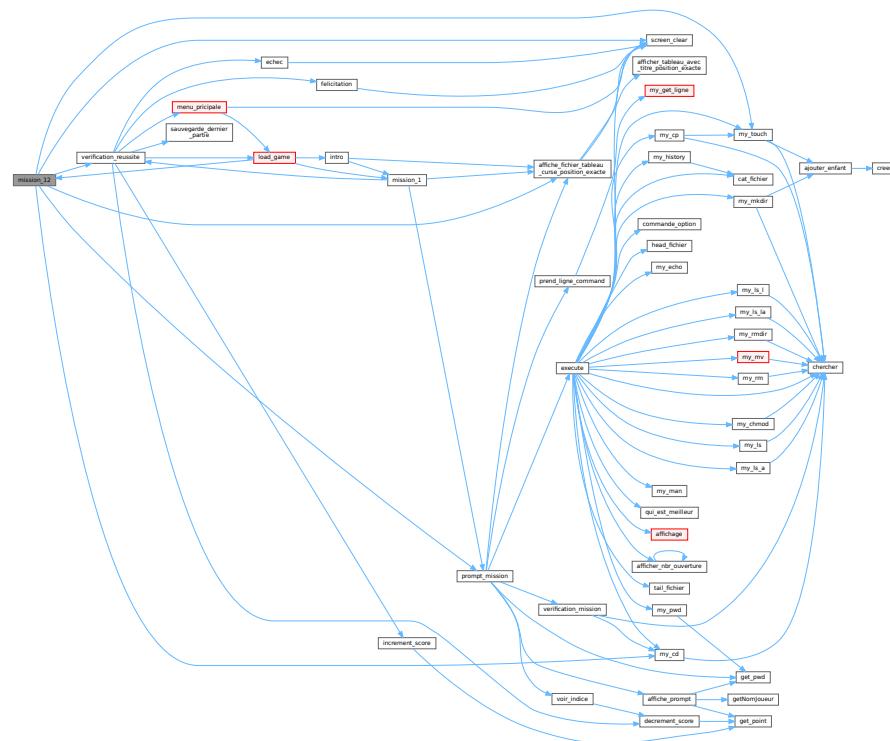


#### 4.25.2.9 mission\_12()

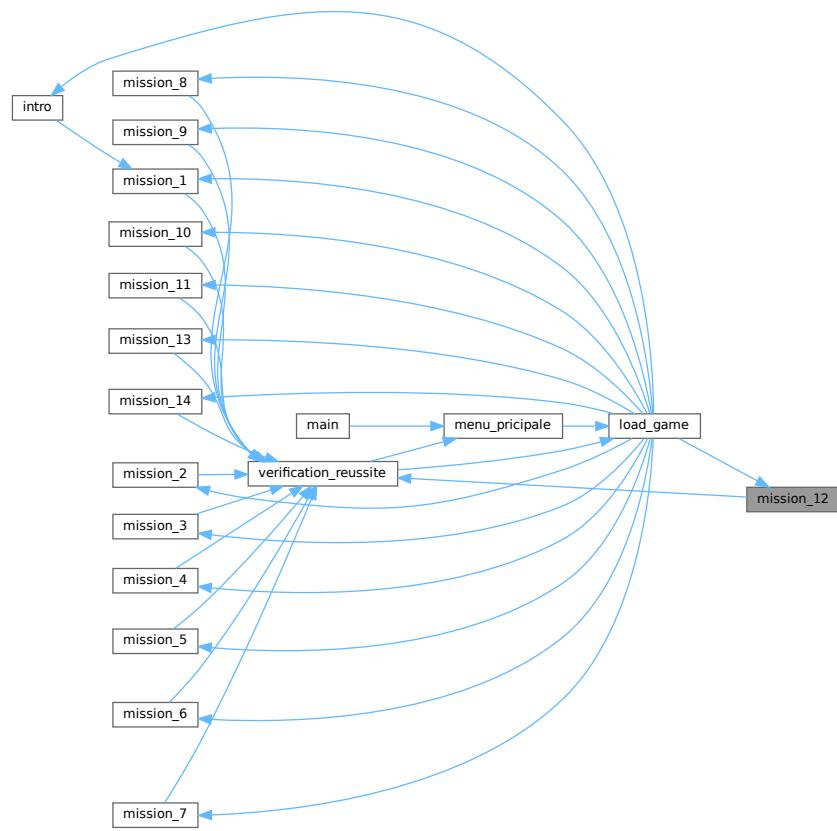
```
void mission_12 (
    Mission * mission,
    fichier * racine )
```

Definition at line 1076 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

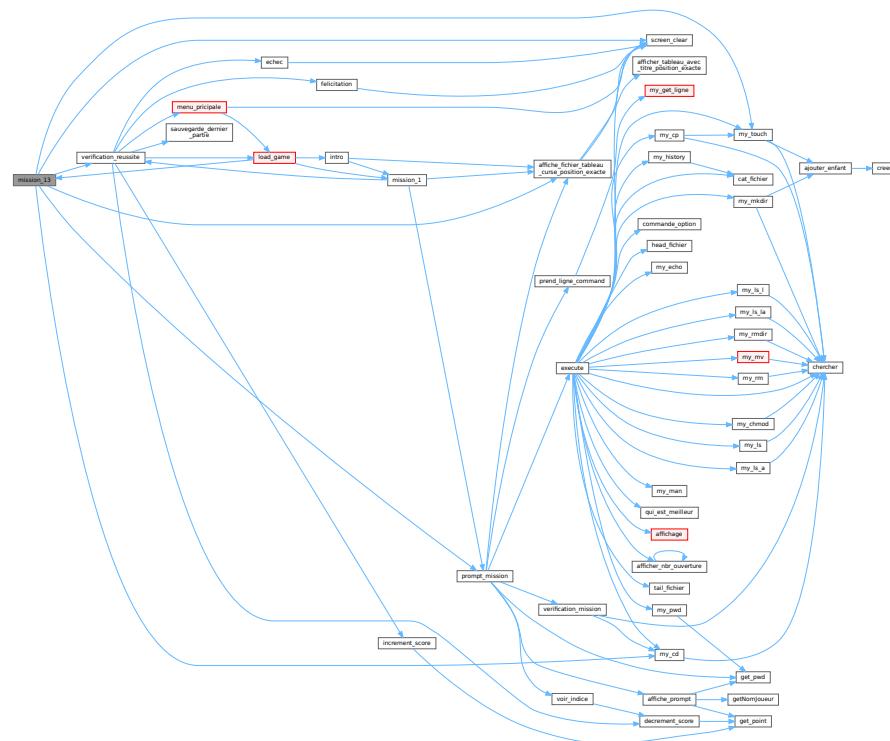


#### 4.25.2.10 mission\_13()

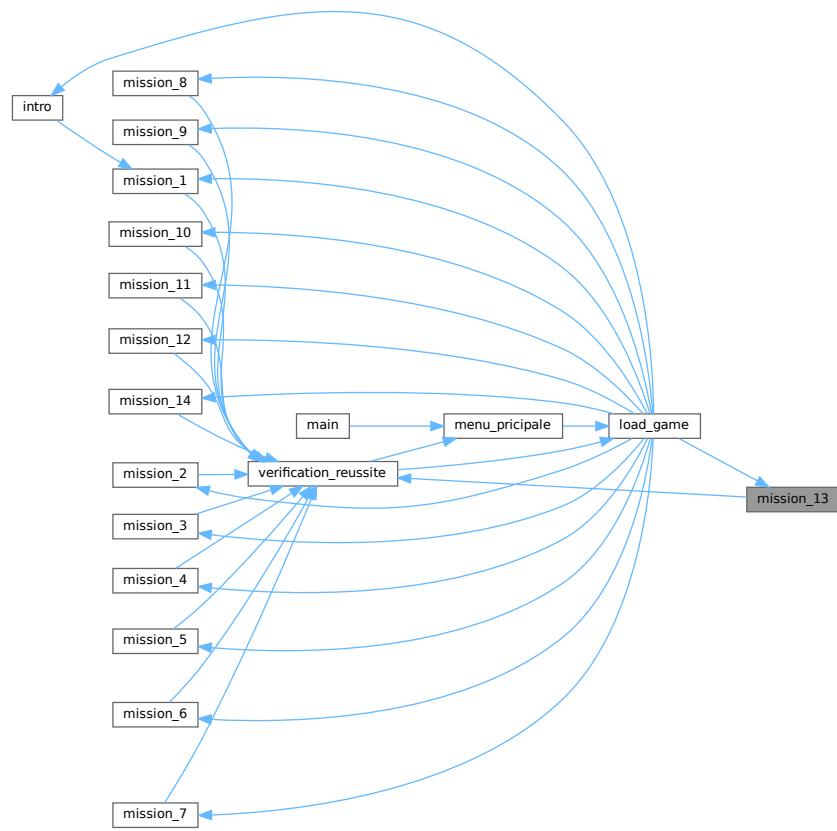
```
void mission_13 (
    Mission * mission,
    fichier * racine )
```

Definition at line 1119 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

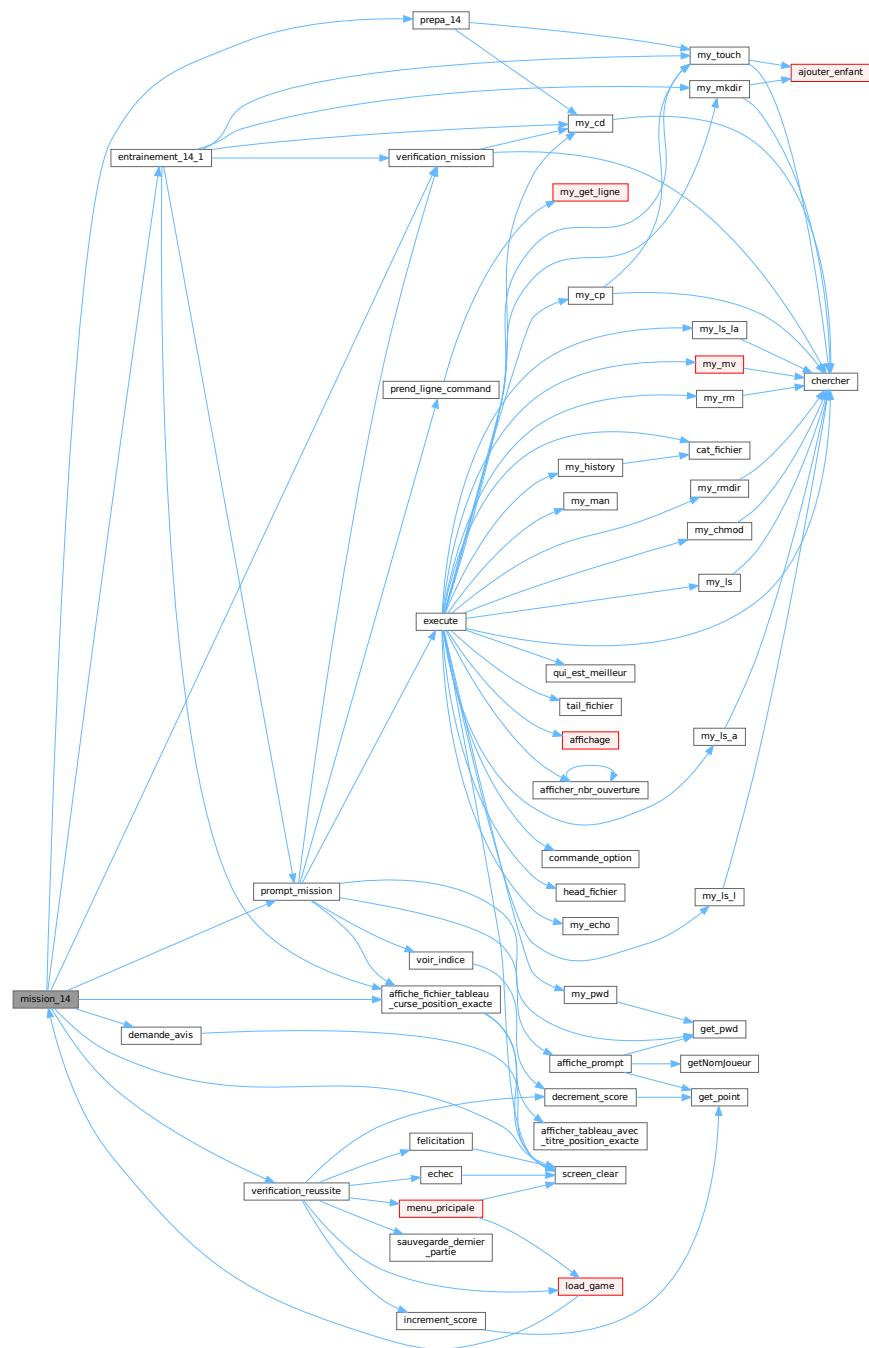


#### 4.25.2.11 mission\_14()

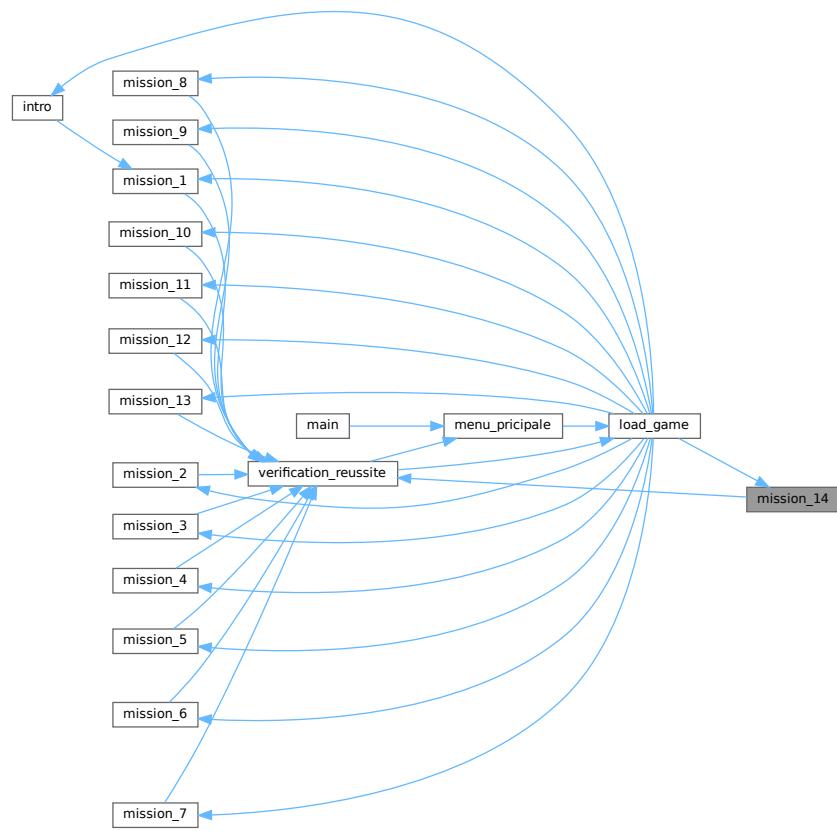
```
void mission_14 (
    Mission * mission,
    fichier * racine )
```

Definition at line 1252 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

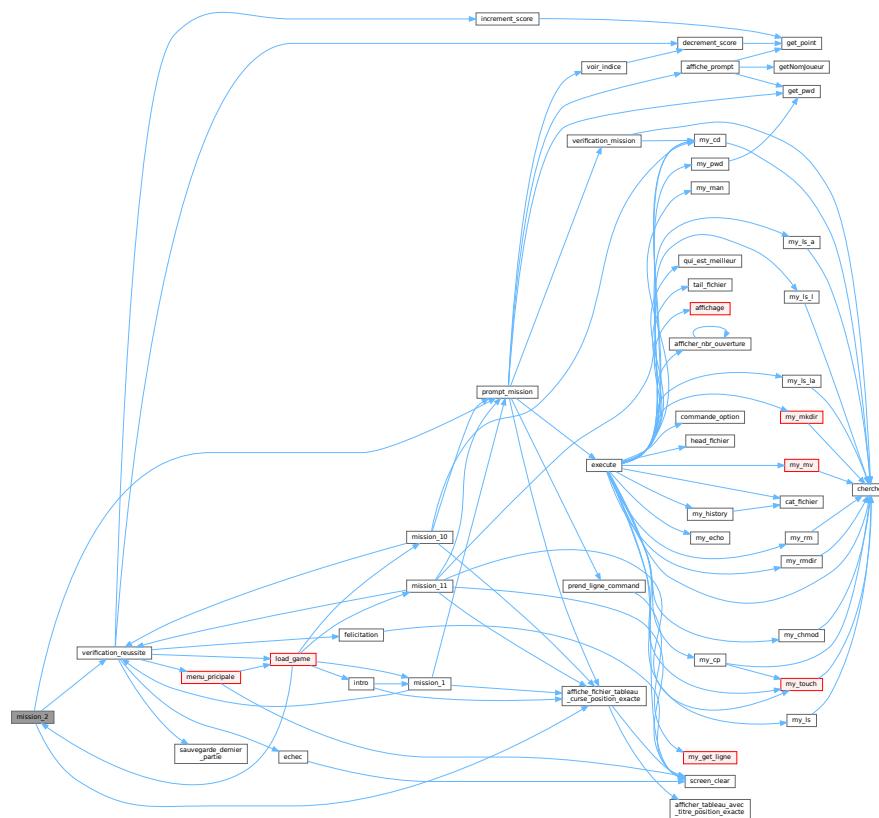


#### 4.25.2.12 mission\_2()

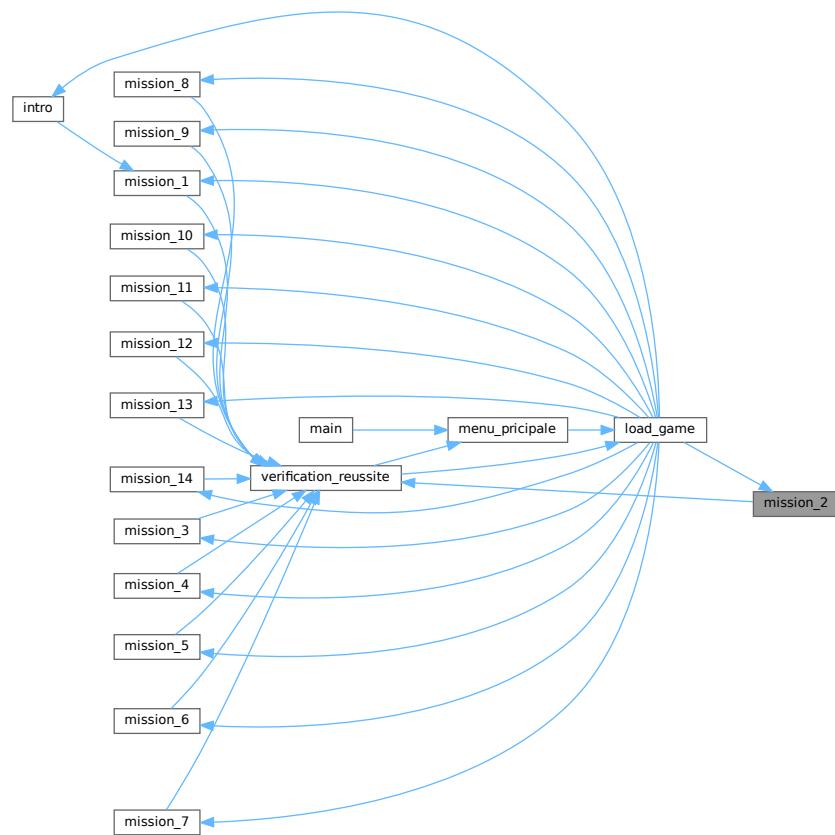
```
void mission_2 (
    Mission * mission,
    fichier * racine )
```

Definition at line 462 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

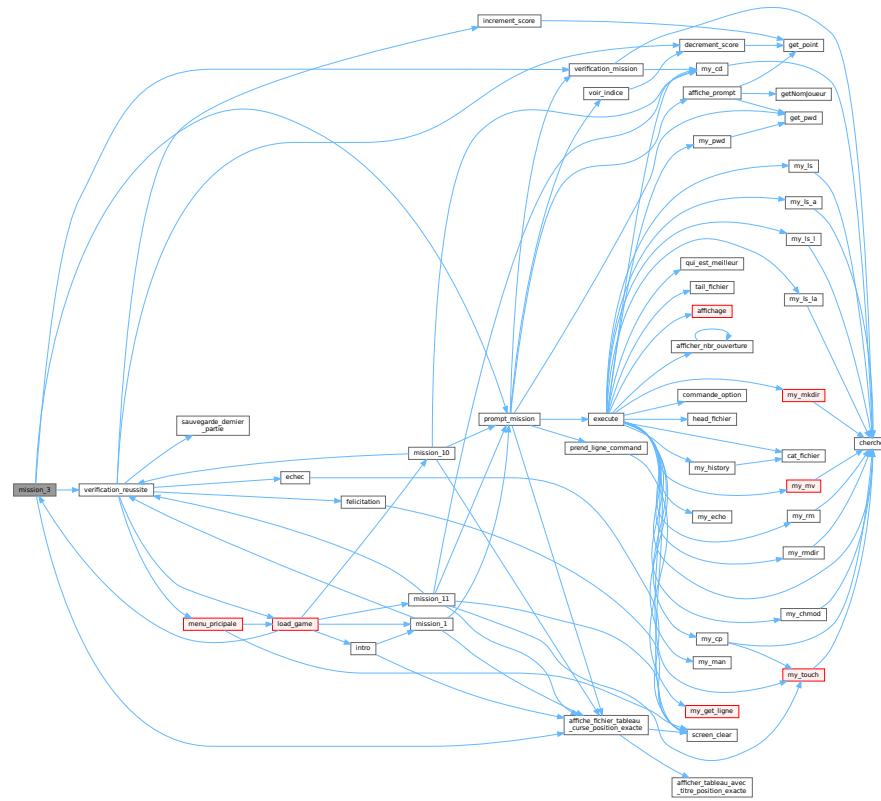


#### 4.25.2.13 mission\_3()

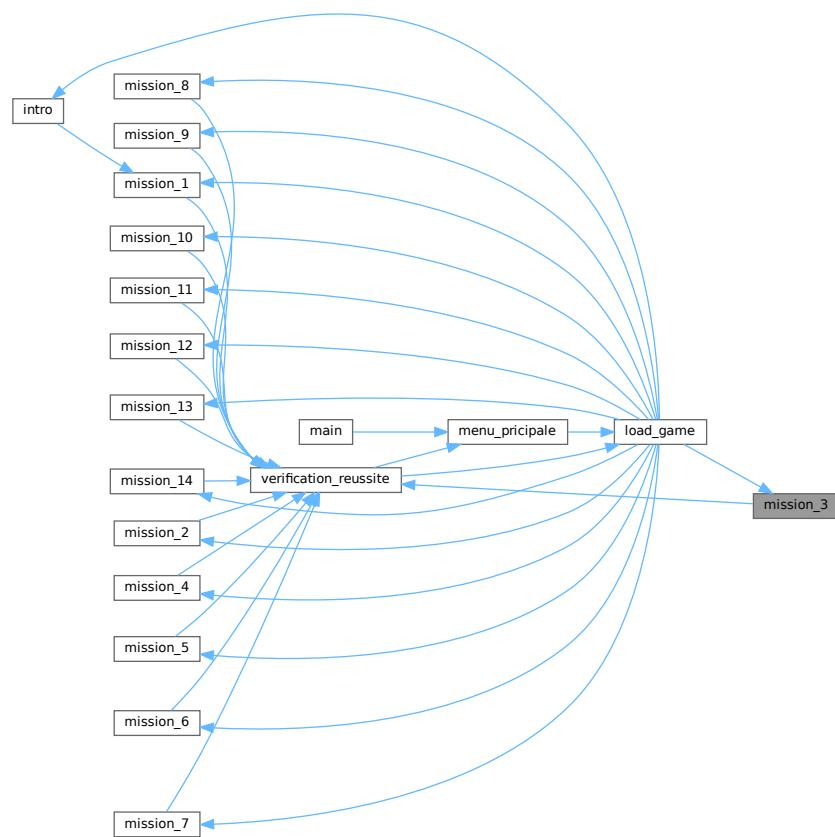
```
void mission_3 (
    Mission * mission,
    fichier * racine )
```

Definition at line 705 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

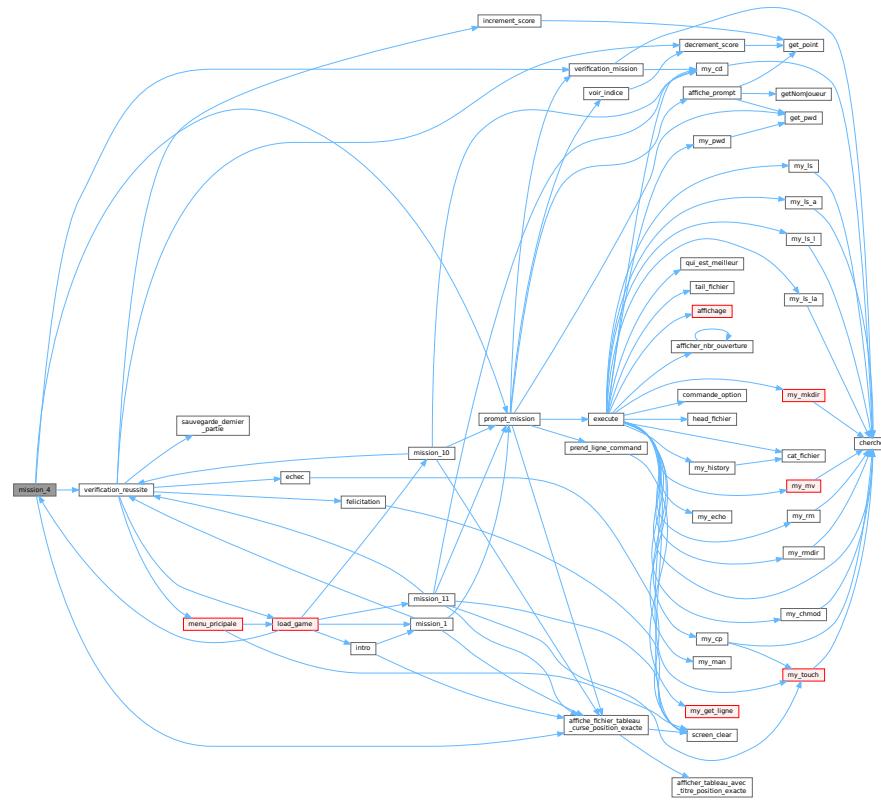


#### 4.25.2.14 mission\_4()

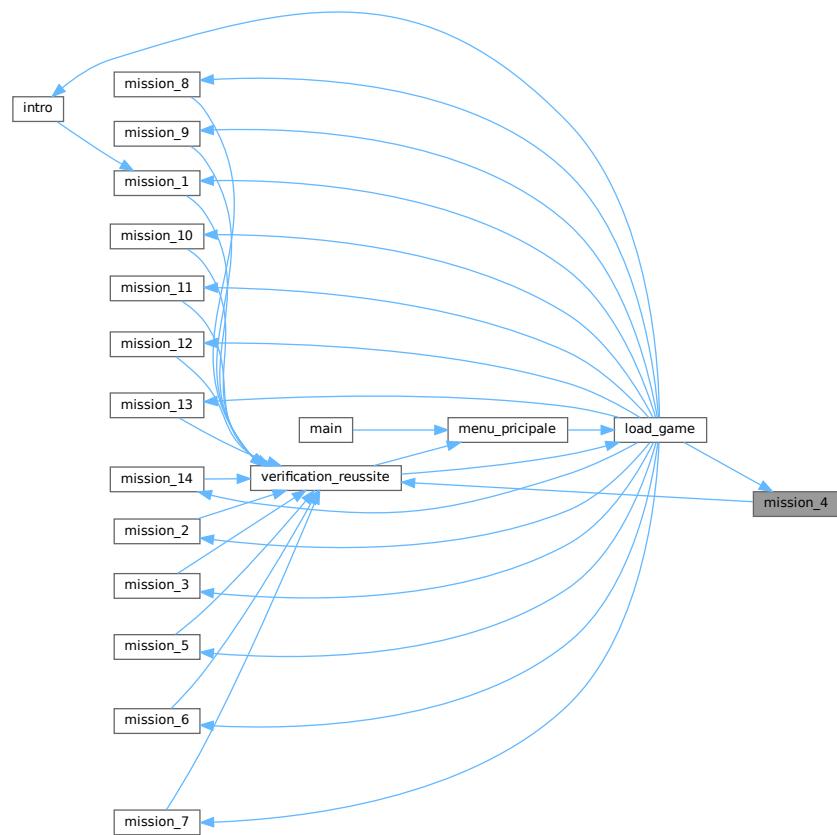
```
void mission_4 (
    Mission * mission,
    fichier * racine )
```

Definition at line 730 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

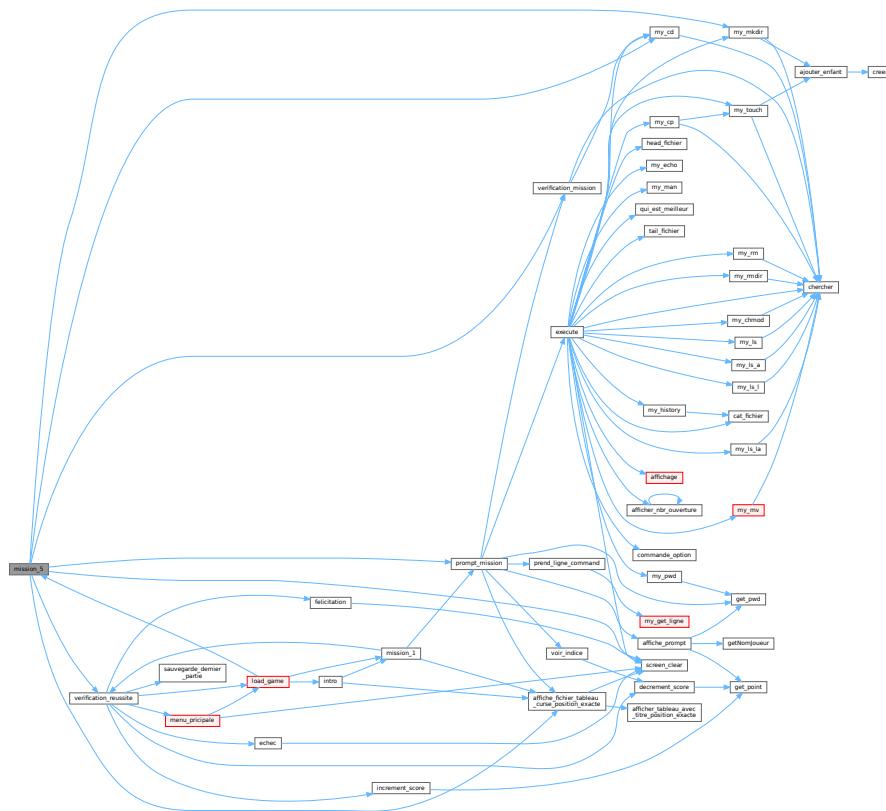


#### 4.25.2.15 mission\_5()

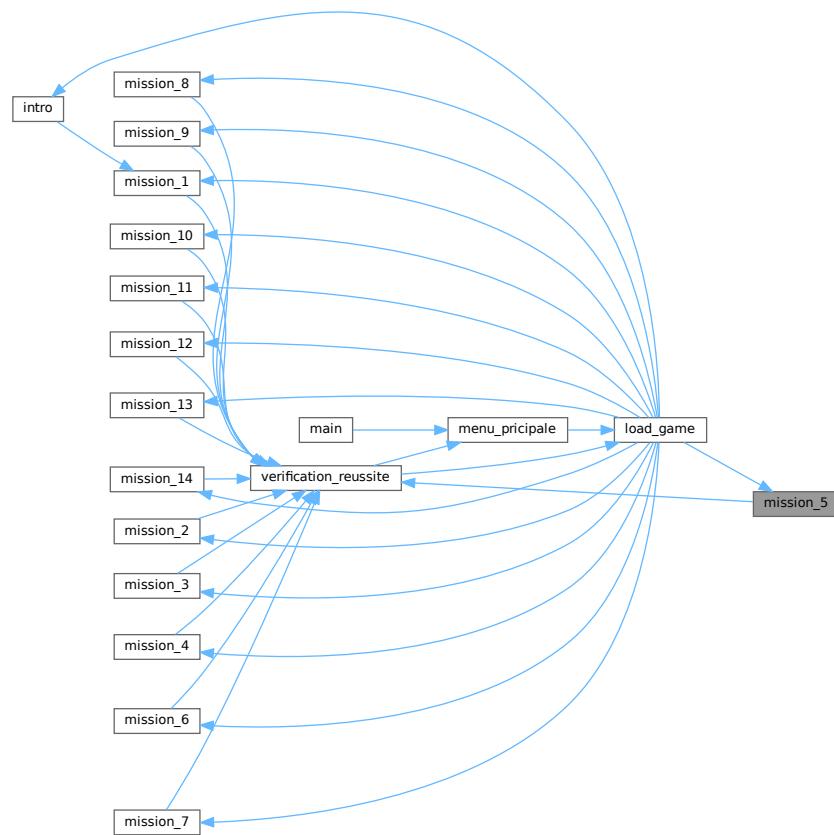
```
void mission_5 (
    Mission * mission,
    fichier * racine )
```

Definition at line 760 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

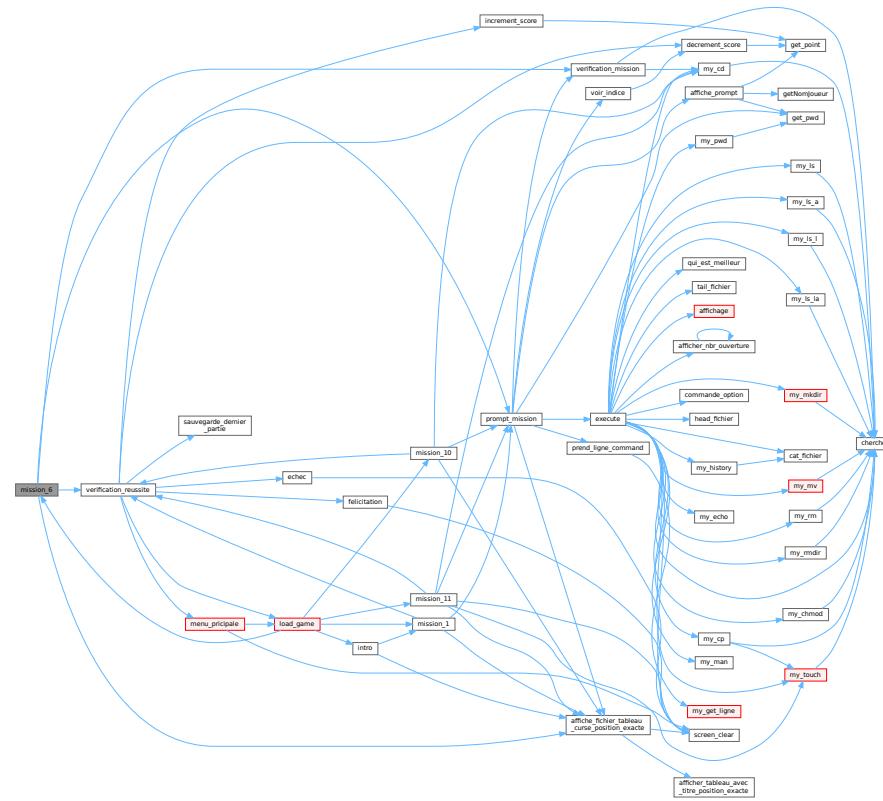


#### 4.25.2.16 mission\_6()

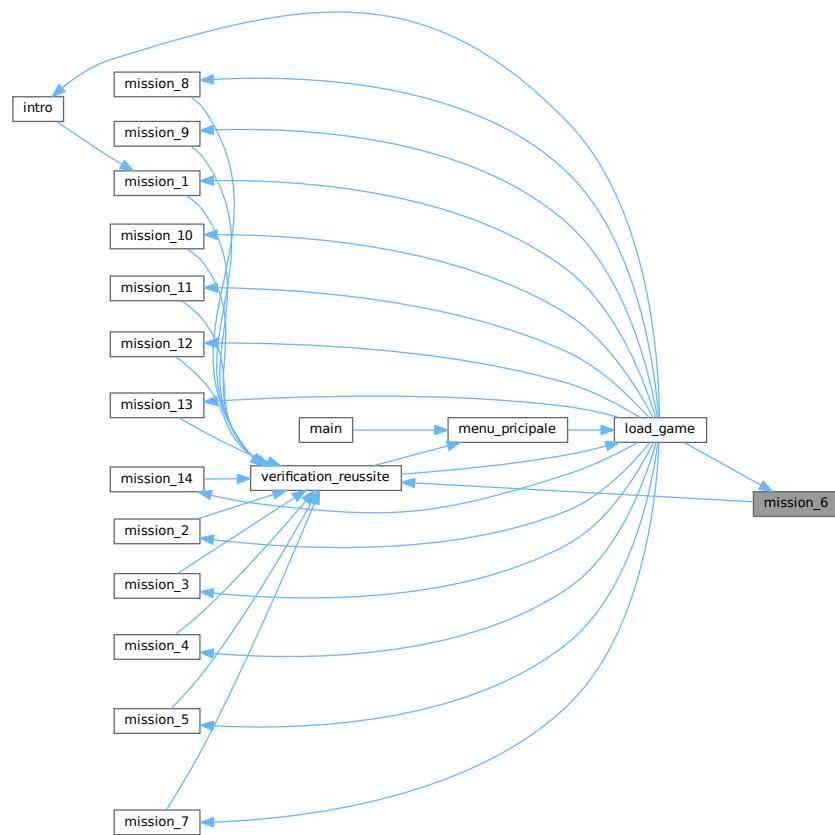
```
void mission_6 (
    Mission * mission,
    fichier * racine )
```

Definition at line 867 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

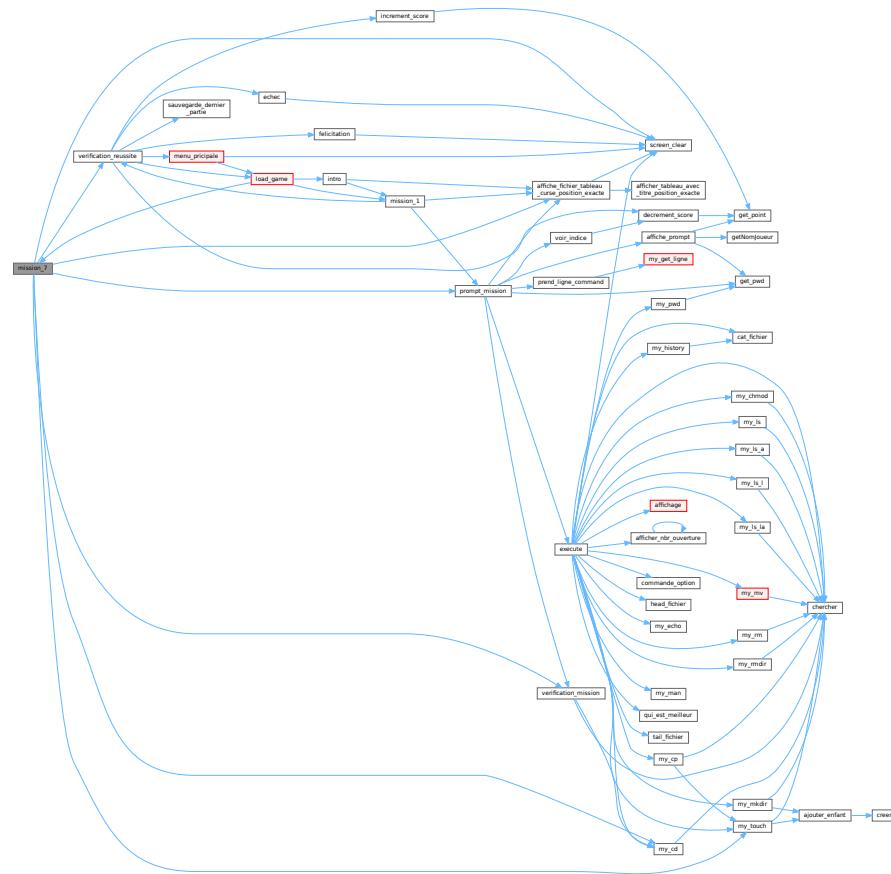


#### 4.25.2.17 mission\_7()

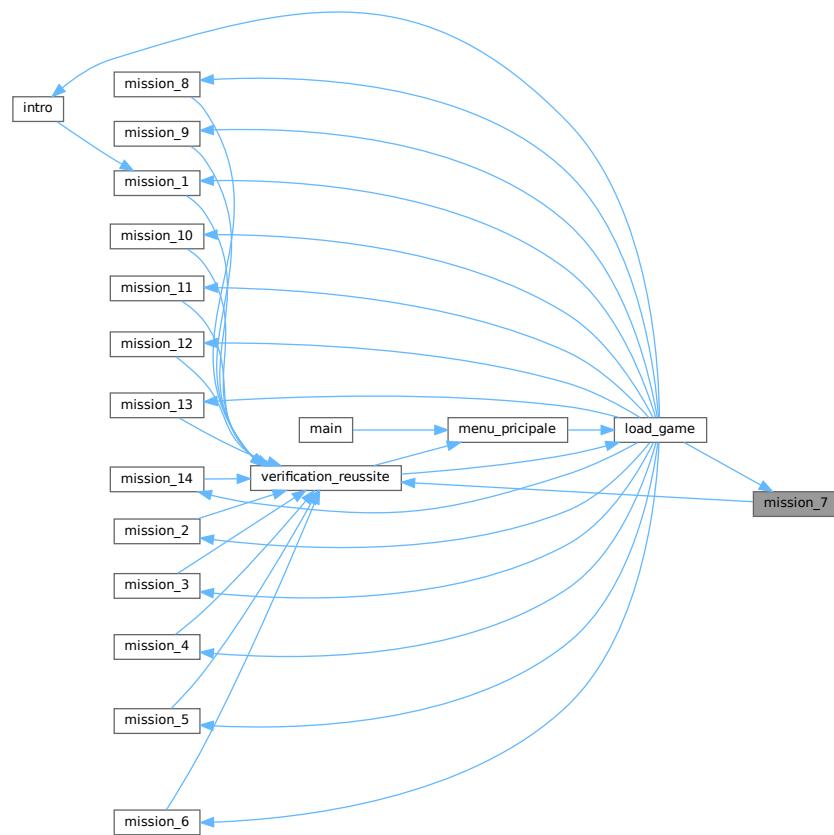
```
void mission_7 (
    Mission * mission,
    fichier * racine )
```

Definition at line 891 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

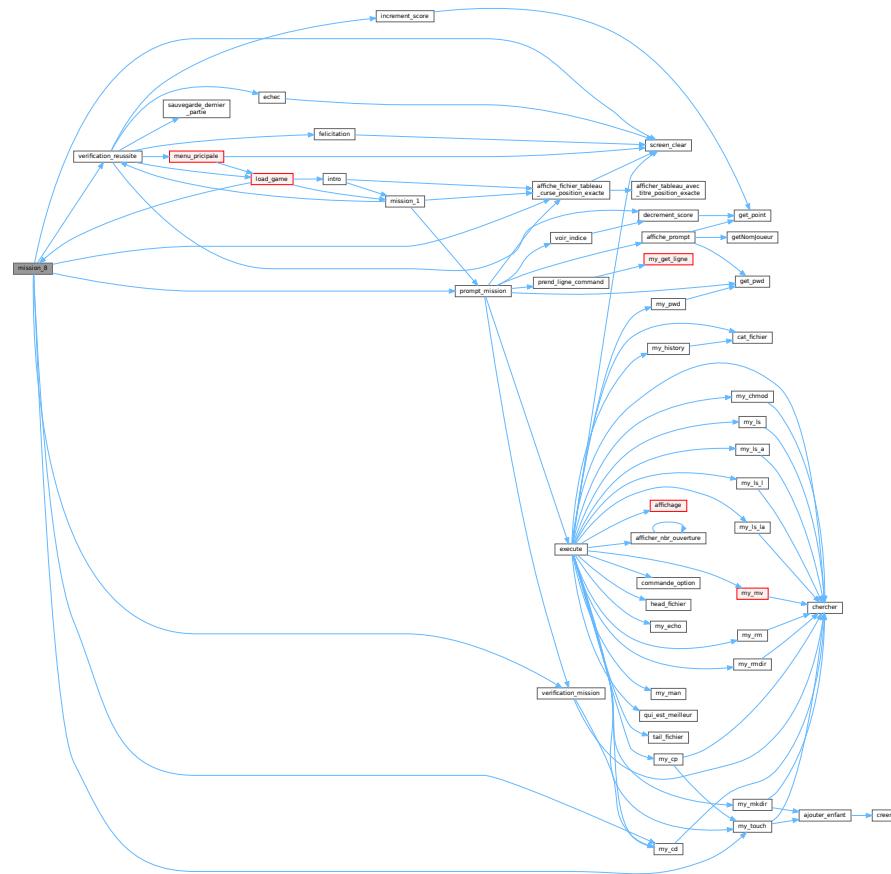


#### 4.25.2.18 mission\_8()

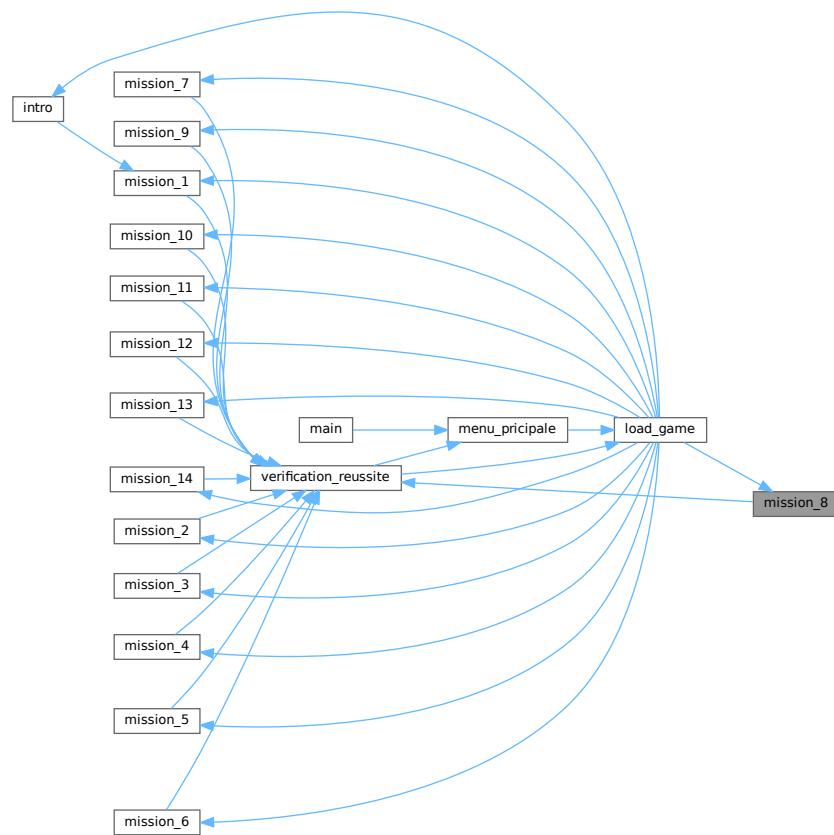
```
void mission_8 (
    Mission * mission,
    fichier * racine )
```

Definition at line 926 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

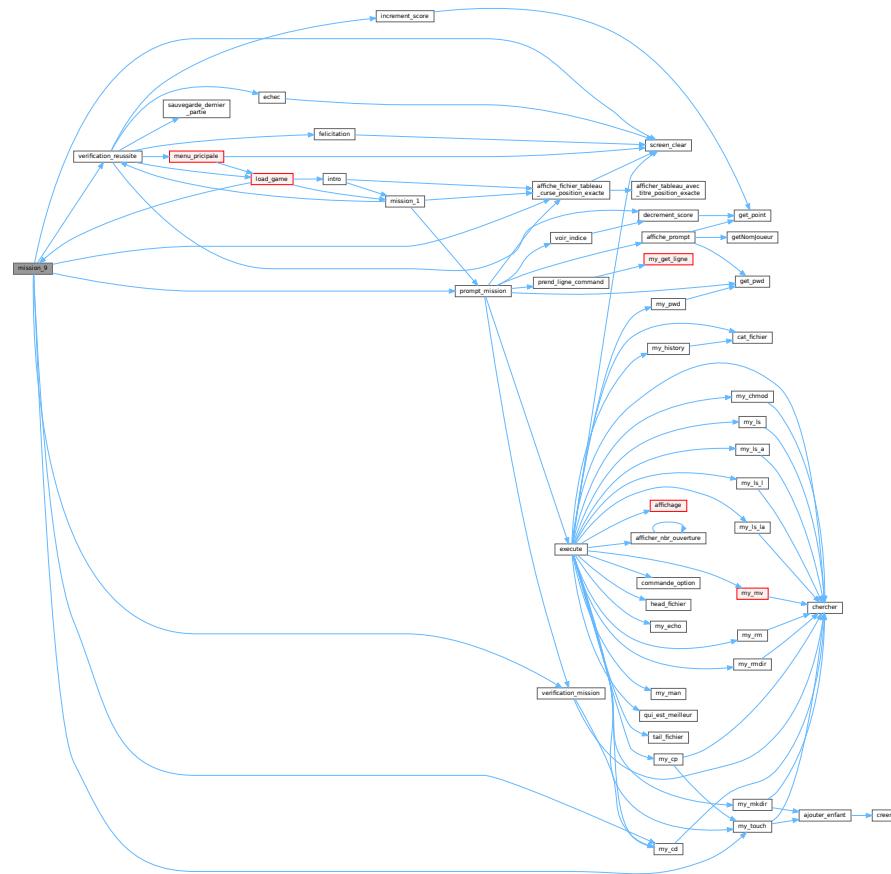


#### 4.25.2.19 mission\_9()

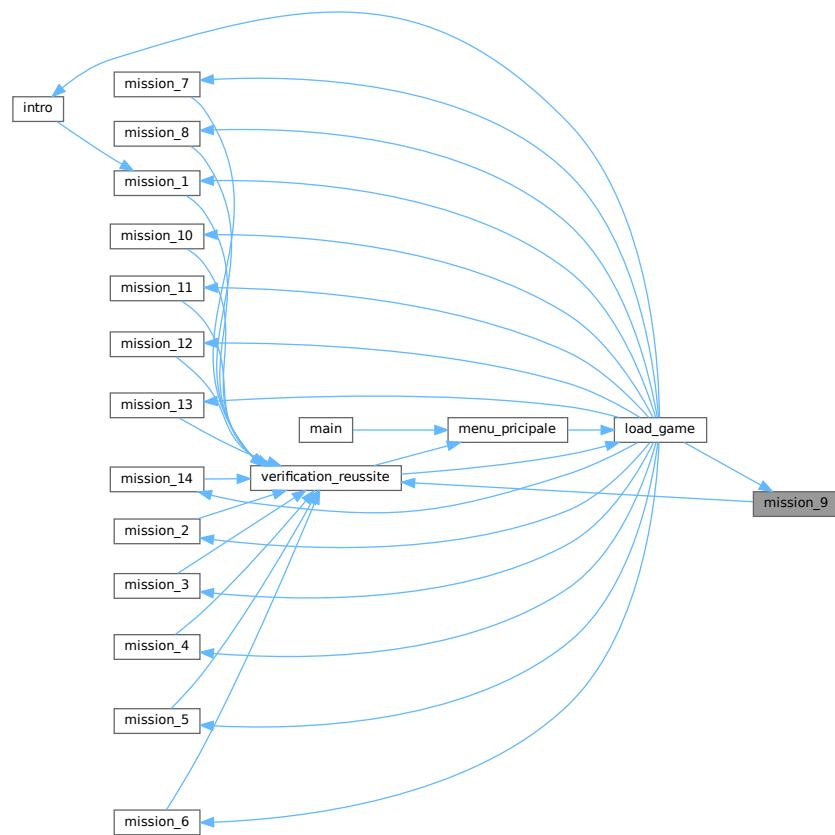
```
void mission_9 (
    Mission * mission,
    fichier * racine )
```

Definition at line 963 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

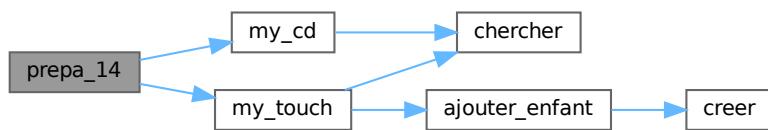


#### 4.25.2.20 prep\_a\_14()

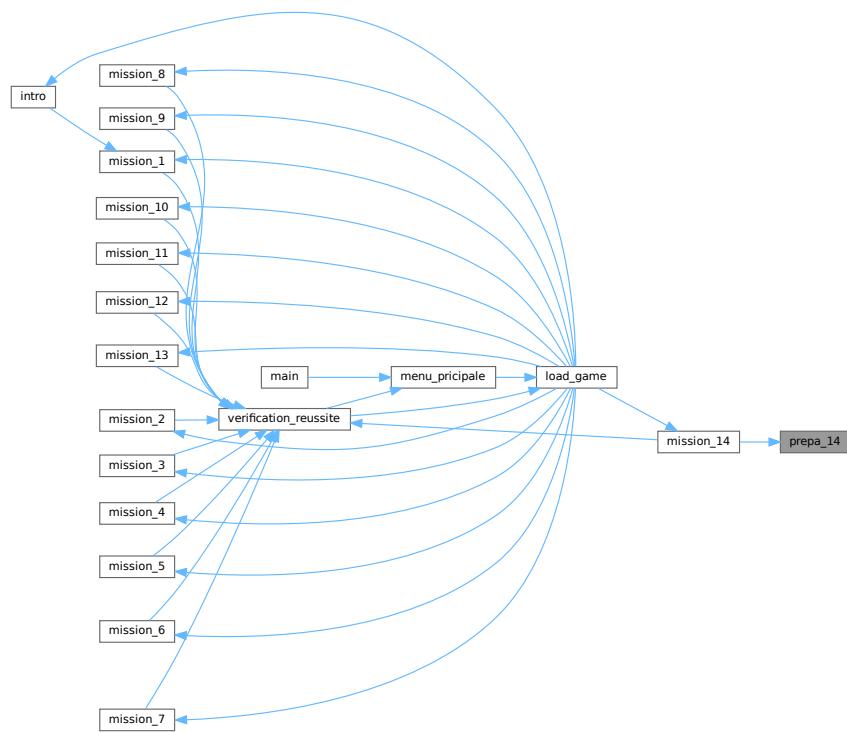
```
void prep_a_14 (
    fichier * racine )
```

Definition at line 1186 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

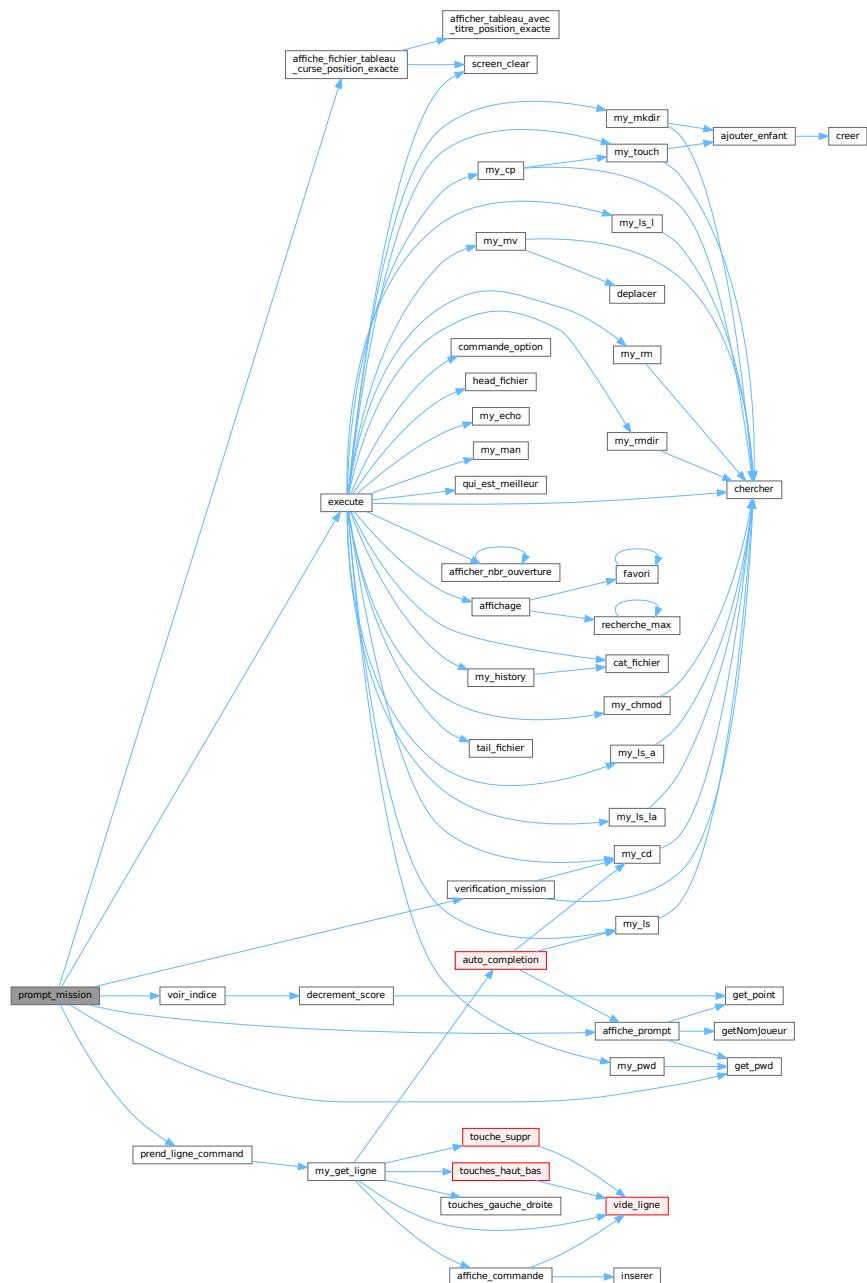


#### 4.25.2.21 `prompt_mission()`

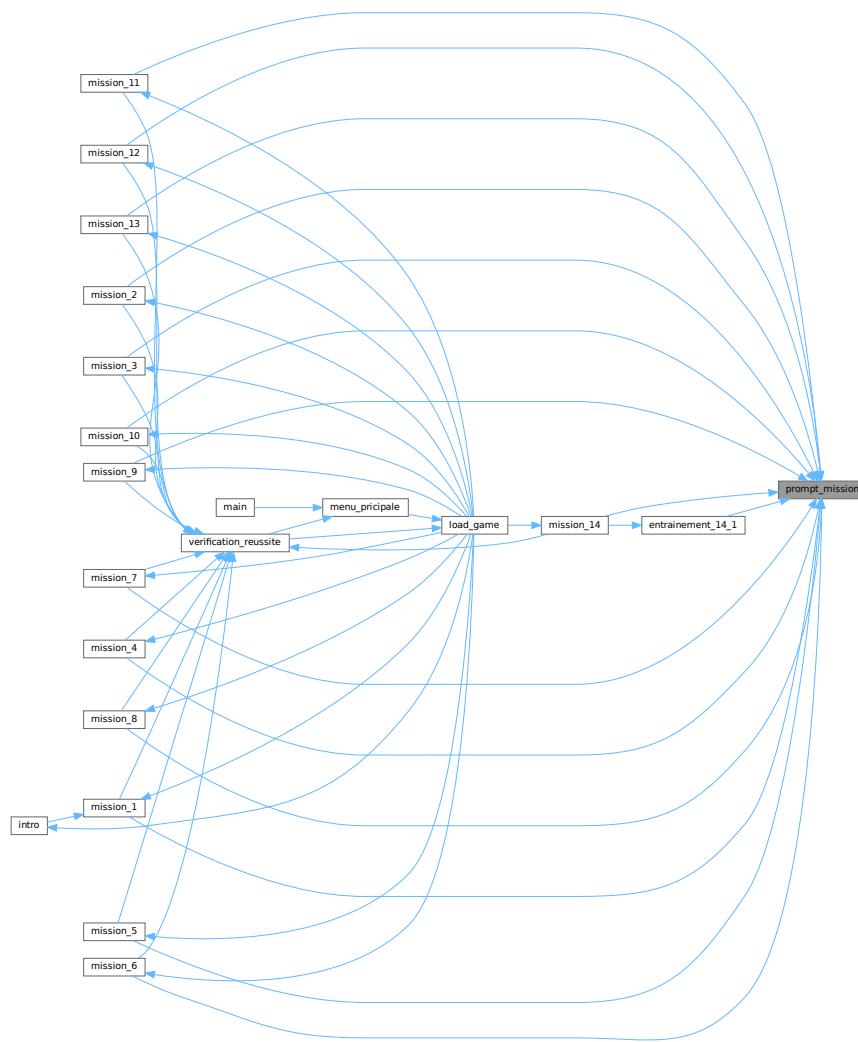
```
bool prompt_mission (
    fichier * racine,
    char * commande,
    char * titre,
    char * chemin,
    char * rep_courant,
    Mission * miss_actuelle,
    int rang_mission,
    int autoverifi )
```

Definition at line 309 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.25.2.22 réinitialisation\_mission()

```
void réinitialisation_mission (
    Mission mission[] )
```

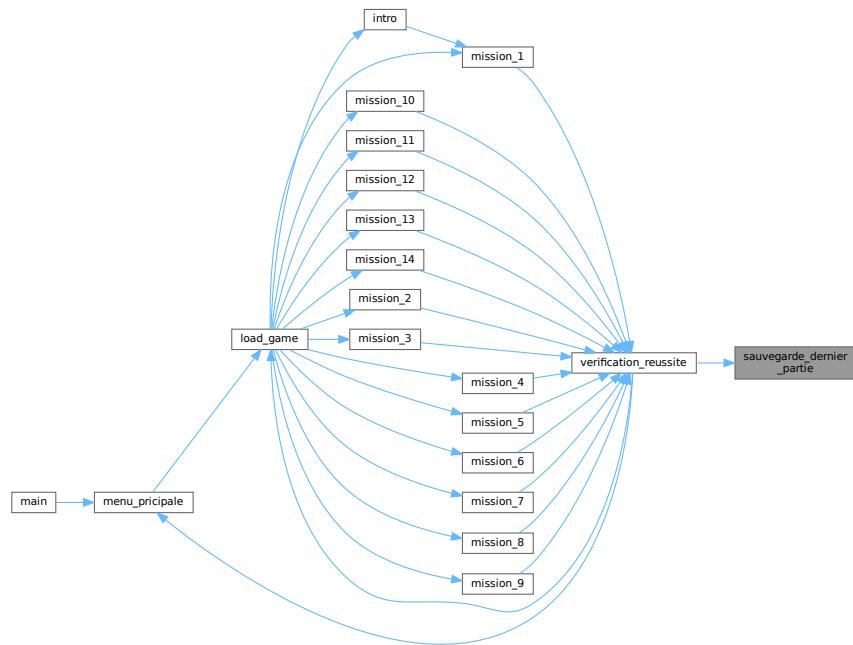
Definition at line 165 of file [mission.c](#).

#### 4.25.2.23 sauvegarde\_dernier\_partie()

```
void sauvegarde_dernier_partie (
    Mission * mission,
    int rang_mission_actuel )
```

Definition at line 111 of file [mission.c](#).

Here is the caller graph for this function:



#### 4.25.2.24 verification\_mission()

```

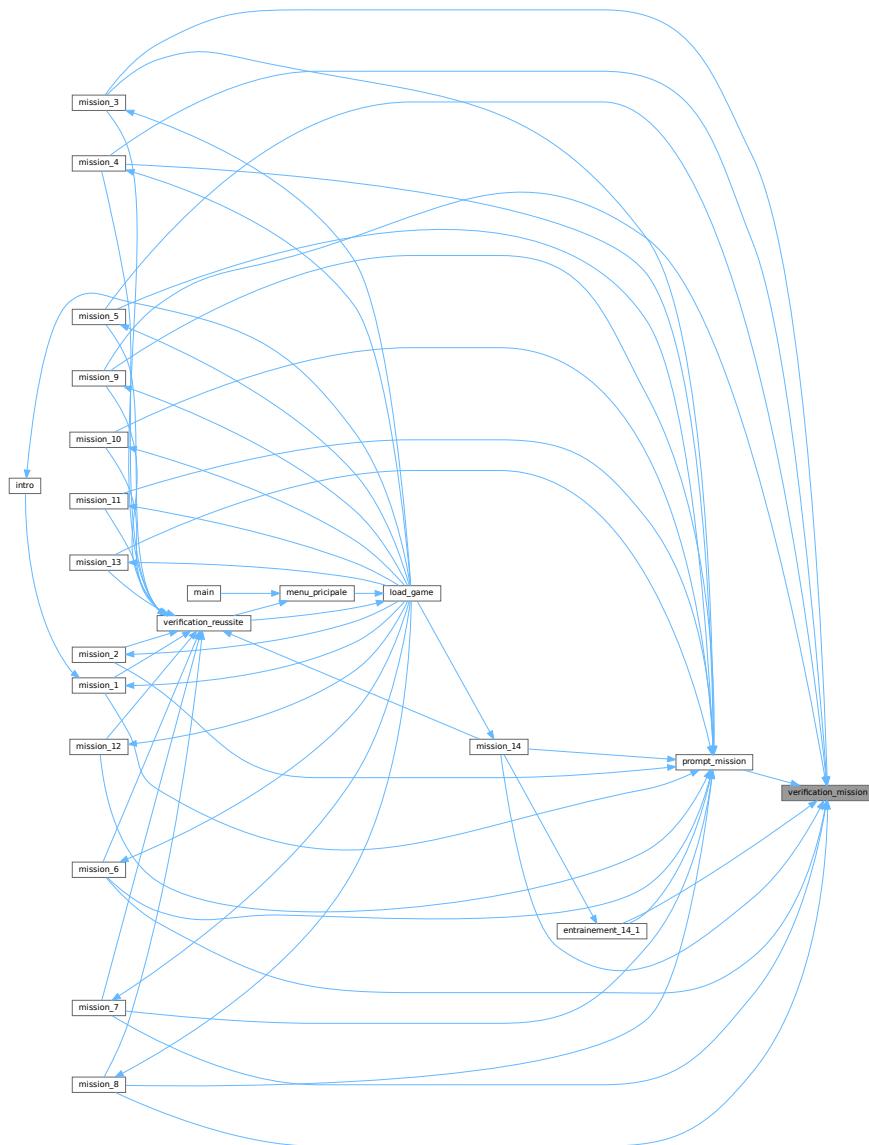
bool verification_mission (
    Mission * mission,
    int rang_mission,
    fichier * racine,
    char * courant )
  
```

Definition at line 548 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

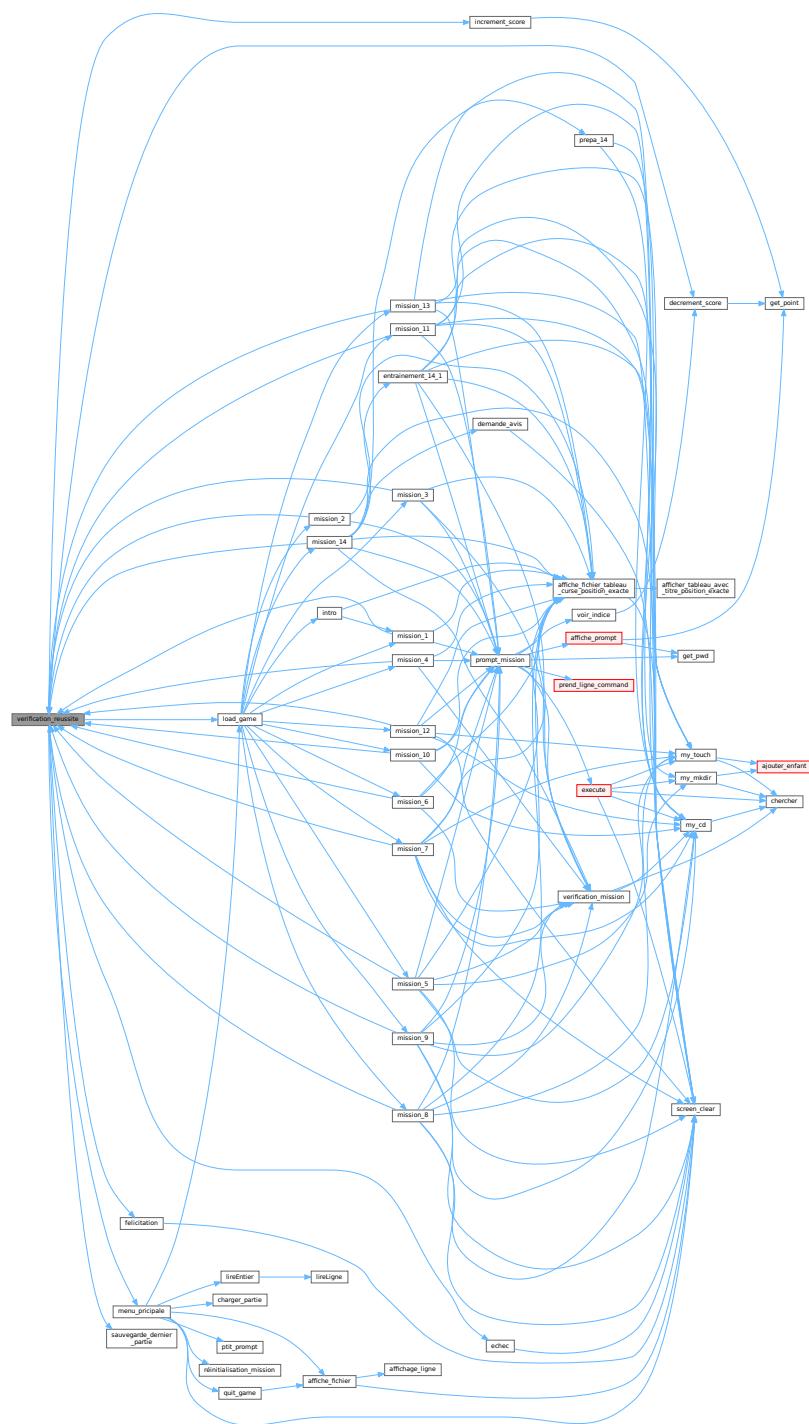


#### 4.25.2.25 `verification_reussite()`

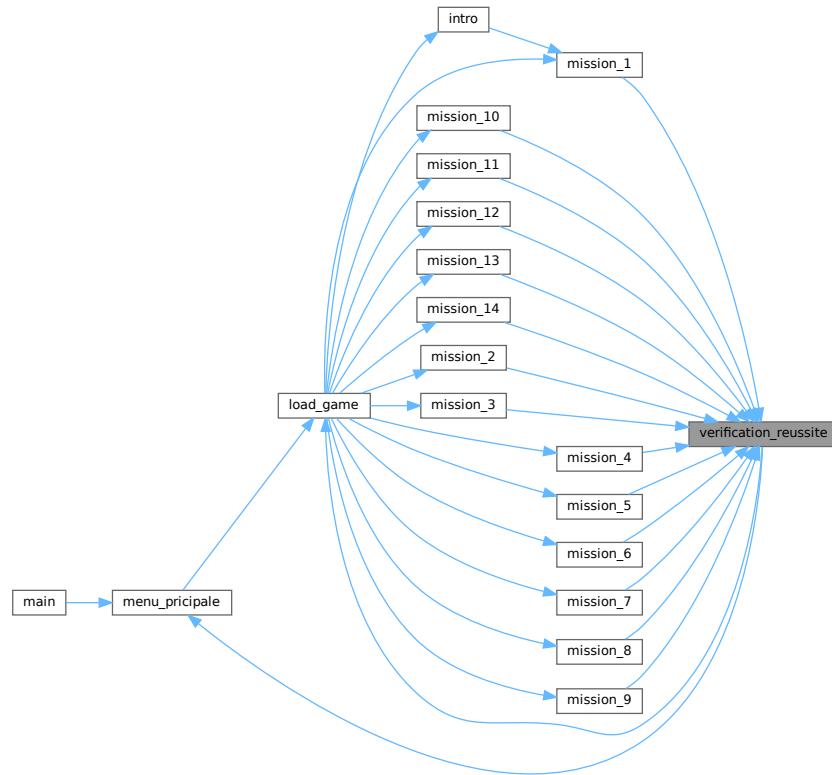
```
void verification_reussite (
    Mission * mission,
    fichier * racine,
    bool reussite,
    int num_mission )
```

Definition at line 246 of file [mission.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.26 mission.c

[Go to the documentation of this file.](#)

```

00001
00008 /***** *****/
00009 #include <stdio.h>
00010 #include <stdlib.h>
00011 #include <string.h>
00012 #include <ncurses.h>
00013 #include <locale.h>
00014 #include <time.h>
00015 #include <signal.h>
00016 #include <unistd.h>
00017 #include <sys/wait.h>
00018 #include "../include/sécurité.h"
00019 #include "../include/affichage.h"
00020 #include "../include/gestion_fichier.h"
00021 #include "../include/prompt.h"
00022 #include "../include/mission.h"
00023 #include "../include/menu.h"
00024
00025
00032 void map_initialisation(Mission *mission , fichier *racine)
00033 {
00034     fichier *courant = NULL ;
00035
00036     my_mkdir(racine , racine , "Poste_Commandement" , 1) ;
00037     courant=my_cd(racine,racine,"Poste_Commandement") ;
00038     my_mkdir(racine,courant,"cartes_stratégiques", 1);
00039     my_mkdir(racine,courant,"transmission", 1);
00040     my_mkdir(racine,courant,"journal_missions", 1);
00041     my_mkdir(racine , racine , "Quartier_Généraux" , 1) ;
00042     courant=my_cd(racine,racine,"Quartier_Généraux" ) ;
00043     my_mkdir(racine,courant,"bureau_du_général" , 1);
00044     my_mkdir(racine,courant,"salle_briefing" , 1);

```

```

00045     my_mkdir(racine,courant,"système_sécurité" , 1);
00046     my_mkdir(racine, racine , "Hangar" , 1);
00047     courant=my_cd(racine,racine,"Hangar");
00048     my_mkdir(racine,courant,"mechas" , 1);
00049     my_mkdir(racine,courant,"vehicules_blindes" , 1);
00050     my_mkdir(racine,courant,"carburant" , 1);
00051     my_mkdir(racine , racine , "Caserne" , 1);
00052     courant=my_cd(racine,racine,"Caserne") ;
00053     my_mkdir(racine,courant,"dortoirs" , 1);
00054     my_mkdir(racine,courant,"infirmerie" , 1);
00055     my_mkdir(racine,courant,"armurerie" , 1);
00056     my_mkdir(racine , racine , "Laboratoire" , 1);
00057     courant=my_cd(racine,racine,"Laboratoire") ;
00058     my_mkdir(racine,courant,"prototypes" , 1);
00059     my_mkdir(racine,courant,"biogenierie" , 1);
00060     my_mkdir(racine,courant,"archives_secretes" , 1);
00061     my_mkdir(racine , racine , "Zone_Entrainement" , 1);
00062     courant=my_cd(racine,racine,"Zone_Entrainement") ;
00063     my_mkdir(racine,courant,"tir" , 1);
00064     my_mkdir(racine,courant,"combats" , 1);
00065     my_mkdir(racine , racine , "Bunker_Souterrain" , 1);
00066     courant=my_cd(racine,racine,"Bunker_Souterrain") ;
00067     my_mkdir(racine,courant,"salle_serveurs" , 1);
00068     my_mkdir(racine,courant,"chambre_de_contôle" , 1);
00069     my_mkdir(racine,courant,"zone_niveau" , 1);
00070 }
00071
00072 //-----
00073
00080 void load_struct_mission(Mission *mission )
00081 {
00082     FILE *save = NULL , *tiavina = NULL ;
00083     char chemin_level[30] = {"\0"};
00084     char chemin_mission[30] = {"\0"};
00085
00086     save = fopen( "../save/temp.txt" , "r");
00087         fscanf( save , "%s" , chemin_level );
00088         fscanf( save , "%s" , chemin_mission );
00089     fclose(save);
00090
00091     save = fopen( chemin_mission , "rb" );
00092     if ( save == NULL )
00093     {
00094         return ;
00095     }
00096     else
00097     {
00098         fread( mission , sizeof(Mission) , NOMBRE_MISSION , save );
00099     }
00100    fclose( save );
00101
00102 }
00103 //-----
00104
00111 void sauvegarde_dernier_partie( Mission *mission , int rang_mission_actuel )
00112 {
00113     FILE *save = NULL ;
00114     char chemin_level[30] = {"\0"};
00115     char chemin_mission[30] = {"\0"};
00116
00117     // Sauvegarde du dernier partie terminer pour que le joueur puisse continuer la où il etait la
00118     // derniere fois
00119
00120     save = fopen( "../save/temp.txt" , "r");
00121     if( save == NULL ) // Si le fichier est introuvable
00122     {
00123         printf("Erreur : fichier temp.txt introuvable\n");
00124         exit(0) ;
00125     }
00126     fscanf( save , "%s" , chemin_level );
00127     fscanf( save , "%s" , chemin_mission );
00128     fclose(save);
00129
00130     save = fopen( chemin_level , "w" );
00131
00132     if( save == NULL ) // Si le fichier est introuvable
00133     {
00134         printf("Erreur : fichier level.txt introuvable\n");
00135     }
00136     else
00137     {
00138         fprintf(save , "%d", rang_mission_actuel) ;
00139         fclose(save) ;
00140     }

```

```

00141 // Sauvegarde du status mission dans le fichier pour la fonction charger_partie
00142 save = fopen( chemin_mission , "wb");
00144
00145 if(save == NULL) // Si le fichier est introuvable
00146 {
00147     printf("Erreur : fichier level.txt introuvable\n");
00148 }
00149 else
00150 {
00151     fwrite(mission , sizeof(Mission) , NOMBRE_MISSION , save );
00152     fclose(save);
00153 }
00154
00155 }
00156
00157
//-----
00158
00165 void réinitialisation_mission(Mission mission[])
00166 {
00167     int i = 0 ;
00168     FILE *save = NULL; // fichier de sauvegarde du dernière mission
00169     char chemin_level[30] = {"\0"} ;
00170     char chemin_mission[30] = {"\0"} ;
00171     char chemin_point[30] = {"\0"} ;
00172
00173     // Reinitialisation du structure mission
00174
00175     for( i = 0 ; i < NOMBRE_MISSION ; ++i )
00176     {
00177         if (i == 0)
00178         {
00179             mission[i].débloqué = 1 ;
00180             strcpy(mission[i].titre , "MISSION-1 : PWD") ;
00181         }
00182         else
00183         {
00184             mission[i].débloqué = 0 ;
00185         }
00186
00187         mission[i].terminé = 0 ;
00188     }
00189
00190     save = fopen( "../save/temp.txt" , "r");
00191     fscanf( save , "%s" , chemin_level );
00192     fscanf( save , "%s" , chemin_mission );
00193     fscanf( save , "%s" , chemin_point );
00194     fclose(save);
00195
00196     // Sauvegarde du status mission dans le fichier pour la fonction charger_partie
00197     save = fopen( chemin_mission , "wb");
00198     if(!save) // Si le fichier est introuvable
00199     {
00200         printf("Erreur : fichier mission.bin introuvable\n");
00201         exit(0);
00202     }
00203     else
00204     {
00205         fwrite(mission , sizeof(Mission) , NOMBRE_MISSION , save );
00206     }
00207     fclose(save);
00208
00209     // Reinitialiser le rang du mission à 0
00210     save = fopen( chemin_level , "w" );
00211     if(!save)
00212     {
00213         printf("Erreur : fichier level.txt introuvable\n");
00214         exit(0);
00215     }
00216     else
00217     {
00218         fprintf(save, "%d", 0);
00219     }
00220     fclose(save);
00221
00222     // Reinitialisation du point à 0
00223     save = fopen( chemin_point , "w" );
00224     if(!save)
00225     {
00226         printf("Erreur : fichier level.txt introuvable\n");
00227     }
00228     else
00229     {
00230         fprintf(save, "%d", 0);
00231     }
00232     fclose(save);

```

```

00233
00234 }
00235
00236 //-----
00237
00246 void verification_reussite(Mission *mission , fichier *racine , bool reussite , int num_mission )
00247 {
00248     // Ecriture de la rapport apropos du mission
00249     // rapport_mission(num_mission , reussite) ;
00250     if(reussite) // mission reussie ecrit félicitation
00251     {
00252         // Changement statut mission
00253         mission[num_mission - 1 ].terminé = 1 ; // Mission actuel terminé
00254         mission[num_mission ].debloqué = 1 ; // Mission suivant débloqué
00255         sauvegarde_dernier_partie( mission , num_mission + 1); // Sauvegarde du dernier rang
00256         increment_score(mission[num_mission].point);
00257
00258         if(felicitation()) // Si le joueur souhaite continuer la partie
00259         {
00260             load_game(mission , num_mission + 1 , racine ) ;
00261         }
00263         else // Si le joueur veut retourner au menu principal
00264         {
00265             menu_principale( mission , racine ) ;
00266         }
00267     }
00268     else // Mission Failed
00269     {
00270         decrement_score(1);
00271         if (echec()) // Si le joueur veut recommencer le mission
00272         {
00273             load_game( mission , num_mission , racine );
00274         }
00275         else // S'il veut retourner au menu principale
00276         {
00277             menu_principale( mission , racine );
00278         }
00279     }
00280 }
00281
00282 //-----
00289 void intro(Mission *mission , fichier *racine)
00290 {
00291     setlocale(LC_ALL, "");
00292     affiche_fichier_tableau curse_position_exacte("BIEVENUE SUR BASH COMMANDO" , "../data/intro.txt");
00293
00294     mission_1(mission , racine);
00295
00296 }
00297
00298 //-----
00309 bool prompt_mission(fichier *racine , char *commande , char *titre , char *chemin , char *rep_courant
, Mission *miss_actuelle , int rang_mission , int autoverifi)
00310 {
00311     bool sortir = false , reussite = false ;
00312     int nbMot , i ;
00313     clock_t debut , fin;
00314     double temps_seconde;
00315     char **ligne ;
00316     char *temps_ecoule ;
00317     fichier *tmp ;
00318     char *ligne_complet = calloc( LIGNE , sizeof(char));
00319
00320     tmp = racine ;
00321     ligne = calloc( LIGNE , sizeof(char *));
00322     temps_ecoule = calloc( LIGNE , sizeof(char));
00323
00324     for(nbMot=0 ; nbMot<LIGNE ; nbMot++)// atao anaty tableau 2D ilay ligne amzay mora ny ahafantarana
ny argument ex : ls -l --> ligne[0]="ls" ; ligne[1]=="-l"
00325     {
00326         ligne[nbMot] = calloc( MOT , sizeof(char));
00327     }
00328
00329     debut = clock();
00330     while( !(sortir) ) // faire une prompt tant que la commande sortir est false
00331     {
00332         affiche_prompt(tmp); // affiche une prompt avec rep_actuel : fichier *tmp
00333         for(nbMot=0 ; nbMot<LIGNE ; nbMot++)// atao anaty tableau 2D ilay ligne amzay mora ny
ahafantarana ny argument ex : ls -l --> ligne[0]=="ls" ; ligne[1]=="-l"
00334         {
00335             ligne[nbMot][0] = '\0';
00336         }
00337
00338         nbMot = prend_ligne_command(ligne , LIGNE, tmp);
00339         // Recuperation du repertoire courant
00340         get_pwd( tmp , rep_courant );

```

```

00341     ligne_complet[0]='\0' ;
00342     if(strcmp(ligne[0] , commande ) == 0) // la mission est reussi si l'utilisateur ecrit la
00343         commande voulue
00344     {
00345         reussite = true ;
00346     }
00347     else
00348     {
00349         for(i=0 ; i<nbMot ; i++)
00350         {
00351             strcat(ligne_complet , ligne[i]);
00352             if(i != (nbMot-1))
00353             {
00354                 strcat(ligne_complet , " ");
00355             }
00356         }
00357         if(strcmp(ligne_complet , commande) == 0)
00358         {
00359             reussite = true;
00360         }
00361     }
00362
00363     if( strcmp(ligne[0] , "mission" ) == 0 )
00364     {
00365         affiche_fichier_tableau curse_position_exacte( titre , chemin );
00366     }
00367     else if(strcmp(ligne[0] , "indice" ) == 0)
00368     {
00369         if(nbMot != 1)
00370         {
00371             printf("(EE : Pas d'argument autorisee !\n");
00372         }
00373         else
00374         {
00375             voir_indice(miss_actuelle) ;
00376         }
00377     }
00378     else if (strcmp(ligne[0] , "stat" ) == 0) // Si l'user veut checker le status de la mission
00379     {
00380         if(autoverifi == 1) // Si l'auto verification est activée
00381         {
00382             if(reussite == true )
00383             {
00384                 printf("Mission terminé soldat !\nVous pouvez taper sur exit !\n");
00385             }
00386             else
00387             {
00388                 printf("Mission non terminé soldat !\n");
00389             }
00390         }
00391         else
00392         {
00393             if( verification_mission(miss_actuelle , rang_mission , racine , rep_courant) == true )
00394             {
00395                 printf("Mission terminé soldat !\nVous pouvez taper sur exit !\n");
00396             }
00397             else
00398             {
00399                 printf("Mission non terminé soldat !\n");
00400             }
00401         }
00402     }
00403     else
00404     {
00405         tmp = execute(ligne , tmp , nbMot ); // return NULL si la ligne ecrit est "exit"
00406     }
00407
00408     if(tmp == NULL )
00409     {
00410         break;
00411     }
00412 }
00413 fin = clock();
00414 temps_seconde = (double) (fin - debut);
00415 temps_seconde /= CLOCKS_PER_SEC;
00416 sprintf(temps_ecoule , "TEMPS_DERNIER_PROMPT=%lf" , temps_seconde);
00417 putenv(temps_ecoule);
00418
00419 free(ligne_complet);
00420
00421 return (reussite);
00422 }
00423 }
00424 //-----
00425

```

```

00433 void mission_1( Mission *mission , fichier *racine ) // Afficher le répertoire courant
00434 {
00435
00436     bool reussite = false ;
00437     char *courant = calloc(MOT , sizeof(char)) ;
00438
00439     mission[1].point = 0 ;
00440
00441 // affiche_fichier_tableau curse("MISSION 1", "../data/mission_1.txt");
00442 // affiche_fichier_tableau curse_position_exacte( "MISSION 1" , "../data/pwd.txt" );
00443
00444     reussite = prompt_mission(racine , "pwd" , "MISSION 1" , "../data/pwd.txt" , courant , (mission+1)
00445 , 1 , 1 ) ;
00446     if(reussite == true )
00447     {
00448         affiche_fichier_tableau curse_position_exacte( "MISSION 1" , "../data/cours_1.txt" );
00449     }
00450
00451     verification_reussite( mission , racine , reussite , 1 );
00452 }
00453
00454 //-----
00455
00456 void mission_2(Mission *mission, fichier *racine)
00457 {
00458     bool reussite = false;
00459     char *courant = calloc(MOT, sizeof(char));
00460     mission[2].point = 1;
00461
00462     strcpy(mission[2].titre, "MISSION-2 : LS");
00463     affiche_fichier_tableau curse_position_exacte("MISSION 2", "../data/mission_2.txt");
00464
00465 // ----- FILS 1 : Chrono -----
00466     pid_t pid_chrono = fork();
00467     if(pid_chrono == 0)
00468     {
00469         int max_sec = 5;
00470         for(int i = 0; i < max_sec*10; ++i)
00471         {
00472             usleep(100000);
00473         }
00474
00475         exit(1); // code 1 = temps écoulé
00476     }
00477
00478 // ----- FILS 2 : Mission -----
00479     pid_t pid_mission = fork();
00480     if(pid_mission == 0)
00481     {
00482         // exécute prompt_mission dans un sous-fils
00483         reussite = prompt_mission(racine, "ls", "MISSION 2", "../data/mission_2.txt", courant,
00484 (mission+2), 2, 1) ;
00485         exit(reussite ? 0 : 2) ;
00486     }
00487
00488 // ----- PÈRE : supervise fils chrono et fils mission -----
00489     int status;
00490     pid_t fini;
00491
00492     while(1)
00493     {
00494         // vérifie si le fils mission a terminé
00495         fini = waitpid(pid_mission, &status, WNOHANG);
00496         if(fini == pid_mission)
00497         {
00498             reussite = (WIFEXITED(status) && WEXITSTATUS(status) == 0);
00499             // tuer le chrono si la mission a fini
00500             kill(pid_chrono, SIGKILL);
00501             break;
00502         }
00503
00504         // vérifie si le chrono a fini
00505         fini = waitpid(pid_chrono, &status, WNOHANG);
00506         if(fini == pid_chrono)
00507         {
00508             // temps écoulé + tuer le fils mission
00509             kill(pid_mission, SIGKILL);
00510             affiche_fichier_tableau curse_position_exacte("", "../data/vide");
00511             reussite = false ;
00512             break;
00513         }
00514
00515         usleep(5000);
00516     }
00517
00518 // nettoyage des fils

```

```

00524     waitpid(pid_chrono, NULL, 0);
00525     waitpid(pid_mission, NULL, 0);
00526
00527     // ----- étape suivante si la mission réussie -----
00528     if(reussite)
00529     {
00530         reussite = true;
00531         affiche_fichier_tableau curse_position_exacte("MISSION 2", "../data/mission_2_1.txt");
00532         reussite = prompt_mission(racine, "ls Quartier_Généraux/", "MISSION
00532         2", "../data/mission_2_1.txt", courant, (mission+2), 2, 1);
00533     }
00534
00535     verification_reussite(mission, racine, reussite, 2);
00536 }
00537
00538 // -----
00539
00548 bool verification_mission(Mission *mission , int rang_mission , fichier *racine , char* courant)
00549 {
00550     fichier *repertoire_courant = NULL ;
00551     fichier *repertoire_courant_1 = NULL ;
00552     char chemin[] = {"Caserne/amurerie"};
00553
00554     switch (rang_mission)
00555     {
00556     case 3 :
00557         // Verification si l'user est dans camp
00558
00559         if( strcmp(courant , "/AK-07/Zone_Entrainement/combats/") == 0 )
00560         {
00561             return true ;
00562         }
00563         else
00564         {
00565             return false ;
00566         }
00567
00568     break;
00569     case 4 :
00570         // Verification si l'user est dans camp
00571
00572         // On se deplace d'abord dans le repertoire "/AK-07/Caserne/dortoirs"
00573         repertoire_courant = my_cd( racine , racine , "Caserne");
00574         repertoire_courant = my_cd( racine , repertoire_courant , "dortoirs");
00575
00576         // On cherche dans le repertoire si l'user à bien crée "tente_1"
00577         if (chercher( repertoire_courant , "tente_1") != NULL )
00578         {
00579             return true ;
00580         }
00581         else
00582         {
00583             return false ;
00584         }
00585
00586     break;
00587     case 5 :
00588
00589         // On se deplace d'abord dans le repertoire "/AK-07/Caserne/dortoirs"
00590         repertoire_courant = my_cd( racine , racine , "Caserne");
00591         repertoire_courant = my_cd( racine , repertoire_courant , "dortoirs");
00592         // On cherche dans le repertoire si l'user à bien supprimer la tente_2 et tente_3
00593         if ( (chercher( repertoire_courant , "tente_2") == NULL ) && (chercher( repertoire_courant ,
00593         "tente_3") == NULL ) )
00594         {
00595             return true ;
00596         }
00597         else
00598         {
00599             return false ;
00600         }
00601
00602     break;
00603     case 6 :
00604
00605         // On cherche dans le repertoire "/AK-07/Caserne/armurerie" si l'user a bien fabriquer
00606         "pistolet"
00607         repertoire_courant = my_cd( racine , racine , "Caserne");
00608         repertoire_courant = my_cd( racine , repertoire_courant , "armurerie");
00609         if ( (chercher( repertoire_courant , "pistolet") != NULL ) ) // Si pistolet trouvé
00610         {
00611             return true ;
00612         }
00613         else
00614         {
00615             return false ;

```

```

00615     }
00616
00617     break;
00618     case 7 :
00619
00620         repertoire_courant = my_cd( racine , racine , "Zone_Entrainement");
00621         // On cherche dans le reperptoire si l'user à bien supprimer la tente_2 et tente_3
00622         if ( (chercher( repertoire_courant , "fichier_indesirable") == NULL ) )
00623         {
00624             return true ;
00625         }
00626         else
00627         {
00628             return false ;
00629         }
00630
00631     break;
00632     case 8 :
00633
00634     // Deplacement dans les repertoire a vérifier [Hangar et Zone_Entrainement]
00635     repertoire_courant = my_cd(racine , racine , "Hangar" ) ;
00636     repertoire_courant_1 = my_cd(racine , racine , "Zone_Entrainement" ) ;
00637
00638     // On cherche dans le reperptoire si l'user à bien déplacer la caisse dans le Hangar vers la
00639     Zone_Entrainement
00640     if( chercher( repertoire_courant , "caisse") != NULL && chercher( repertoire_courant_1 ,
00641     "caisse") != NULL )
00642     {
00643         return true ;
00644     }
00645     else
00646     {
00647         return false ;
00648     }
00649     break ;
00650     case 9 :
00651     // Deplacement dans les repertoire a vérifier [Hangar et Zone_Entrainement]
00652     repertoire_courant = my_cd(racine , racine , "Hangar" ) ;
00653     repertoire_courant_1 = my_cd(racine , racine , "Zone_Entrainement" ) ;
00654
00655     // On cherche dans le reperptoire si l'user à bien déplacer la caisse dans le Hangar vers la
00656     Zone_Entrainement
00657     if( chercher( repertoire_courant , "caisse") == NULL && chercher( repertoire_courant_1 ,
00658     "caisse") != NULL )
00659     {
00660         return true ;
00661     }
00662     else
00663     {
00664         return false ;
00665     }
00666     break ;
00667     case 141 :
00668
00669     // Verification si l'user est dans le Coffre Fort
00670     if( strcmp(courant , "/AK-07/Zone_Entrainement/Coffre_Fort/") == 0 )
00671     {
00672         return true ;
00673     }
00674     else
00675     {
00676         return false ;
00677     }
00678
00679     break;
00680     case 14 :
00681
00682     // Deplacement dans les repertoire a vérifier [ /AK-07/Laboratoire/archives_secretes/ &&
00683     /AK-07/Quartier_Généraux/bureau_du_général/ ]
00684     repertoire_courant = my_cd(racine , racine , "Laboratoire" );
00685     repertoire_courant = chercher(repertoire_courant , "archives_secretes");
00686     repertoire_courant_1 = my_cd(racine , racine , "Quartier_Généraux" );
00687     repertoire_courant_1 = my_cd(racine , repertoire_courant_1 , "bureau_du_général");
00688
00689     // On cherche dans le reperptoire si le repertoire est bien protégé, le joueur n'est plus dans
00690     le l'archives_secretes et le fichier demandé est dans le bureau du général
00691     if( strcmp( repertoire_courant->perm , "----" ) == 0 && chercher( repertoire_courant_1 ,
00692     "X4-77-DELTA" ) != NULL && strcmp(courant , "/AK-07/Laboratoire/archives_secretes/") != 0 )
00693     {
00694         return true ;
00695     }
00696     else
00697     {
00698         return false ;
00699     }
00700 default:

```

```

00695     break;
00696 }
00697 }
00698 //-----
00705 void mission_3(Mission *mission , fichier *racine ) // se déplacer dans un dossier
00706 {
00707     bool reussite = false ;
00708     char *courant = calloc(MOT , sizeof(char)) ;
00709
00710     strcpy(mission[3].titre , "MISSION-3 : CD");
00711     strcpy(mission[3].indice , "tu dois simplement entrer le dossier combats qui est dans la zone
d'entraînement");
00712     mission[3].point = 1;
00713
00714     affiche_fichier_tableau curse_position_exacte("MISSION 3", "../data/mission_3.txt");
00715     prompt_mission(racine , "ls" , "MISSION 3" , "../data/mission_3.txt" , courant , (mission+3) , 3 ,
0);
00716
00717     reussite = verification_mission( mission , 3 , racine , courant) ;
00718
00719     verification_reussite( mission , racine , reussite , 3 );
00720 }
00721
00722
00723 //-----
00730 void mission_4(Mission *mission , fichier *racine ) // création des dossiers
00731 {
00732     bool reussite = false ;
00733     fichier *repertoire_courant = NULL;
00734     char *courant = calloc(MOT , sizeof(char)) ;
00735
00736     strcpy(mission[4].titre , "MISSION-4 : mkdir");
00737     mission[4].point = 2;
00738
00739     affiche_fichier_tableau curse_position_exacte("MISSION 4", "../data/mission_4.txt");
00740     prompt_mission(racine , "mkdir" , "MISSION 4" , "../data/mission_4.txt" , courant , (mission+4) , 4
, 0 );
00741
00742     reussite = verification_mission( mission , 4 , racine , courant);
00743
00744     if (reussite == true)
00745     {
00746         affiche_fichier_tableau curse_position_exacte("MISSION ACCOMPLIE",
"../data/felicitation_4.txt");
00747     }
00748
00749     verification_reussite( mission , racine , reussite , 4 );
00750
00751 }
00752
00753
00754 //-----
00760 void mission_5(Mission *mission , fichier *racine ) // Supprimer des dossiers
00761 {
00762     bool reussite = false ;
00763     fichier *repertoire_courant = NULL;
00764     int max_sec = 60 ;
00765     char *courant = calloc(MOT , sizeof(char)) ;
00766     // pipe pour recevoir le resultat
00767     int fd[2];
00768     pipe(fd);
00769     int status;
00770     pid_t fini;
00771
00772     strcpy(mission[5].titre , "MISSION-5 : rmdir");
00773     mission[5].point = 2;
00774
00775     // On se deplace d'abord dans le repertoire "/AK-07/Caserne/dortoirs"
00776     repertoire_courant = my_cd( racine , racine , "Caserne");
00777     repertoire_courant = my_cd( racine , repertoire_courant , "dortoirs");
00778     // Creation du repertoire tente_2 et tente_3
00779     my_mkdir(racine , repertoire_courant , "tente_2" , 0);
00780     my_mkdir(racine , repertoire_courant , "tente_3" , 0);
00781
00782     screen_clear();
00783
00784     // Affichage de la mission
00785     affiche_fichier_tableau curse_position_exacte("MISSION 5" , "../data/mission_5.txt");
00786     affiche_fichier_tableau curse_position_exacte("MISSION 5 - TEMPS LIMITE : 60 secondes",
"../data/vide");
00787
00788
00789     // ----- FILS 1 : Chrono -----
00790     pid_t pid_chrono = fork();
00791     if(pid_chrono == 0)
00792     {

```

```

00793         for(int i = 0; i < max_sec*10; ++i)
00794         {
00795             usleep(100000);
00796         }
00797         exit(1); // code 1 = temps écoulé
00799     }
00800
00801 // ----- FILS 2 : Mission -----
00802 pid_t pid_mission = fork();
00803 if(pid_mission == 0)
00804 {
00805     close(fd[0]); // fermer lecture
00806
00807     // exécute prompt_mission dans un sous-fils
00808     prompt_mission(racine, "rmdir", "MISSION 5", "../data/mission_5.txt", courant,
00809     (mission+5), 5, 0);
00810     reussite = verification_mission( mission, 5, racine, courant );
00811
00812     // Envoie le résultat au père
00813     write( fd[1] , &reussite , sizeof(bool) );
00814     close(fd[1]); // fermer écriture
00815
00816     exit(0);
00817 }
00818 // ----- PÈRE : supervise fils chrono et fils mission -----
00819
00820
00821 while(1)
00822 {
00823     // vérifie si le fils mission a terminé
00824     fini = waitpid(pid_mission, &status, WNOHANG);
00825     if(fini == pid_mission)
00826     {
00827         // tuer le chrono si la mission a fini
00828         kill(pid_chrono, SIGKILL);
00829         reussite = true ;
00830         break;
00831     }
00832
00833     // vérifie si le chrono a fini
00834     fini = waitpid(pid_chrono, &status, WNOHANG);
00835
00836     if(fini == pid_chrono)
00837     {
00838         // temps écoulé → tuer le fils mission
00839         kill(pid_mission, SIGKILL);
00840         affiche_fichier_tableau curse_position_exacte("TEMPS ÉCOULÉ !", "../data/vide");
00841         reussite = false ;
00842         break;
00843     }
00844
00845     usleep(5000);
00846 }
00847
00848 close(fd[1]); // fermer écriture
00849 // Récupération du résultat du fils mission
00850 read( fd[0] , &reussite , sizeof(bool) );
00851 close(fd[0]); // fermer lecture
00852
00853 // nettoyage des fils
00854 waitpid(pid_chrono, NULL, 0);
00855 waitpid(pid_mission, NULL, 0);
00856
00857 verification_reussite( mission, racine, reussite, 5 );
00858 }
00859
00860
00861 //-----
00862
00863 void mission_6(Mission *mission, fichier *racine) // crée un fichier pistolet
00864 {
00865     bool reussite = false ;
00866     fichier *repertoire_courant = NULL ;
00867     char *courant = calloc(MOT, sizeof(char)) ;
00868
00869     strcpy(mission[6].titre, "MISSION-6 : touch") ;
00870     mission[6].point = 2;
00871     affiche_fichier_tableau curse_position_exacte("MISSION 6", "../data/mission_6.txt");
00872
00873     // lancement du prompt
00874     prompt_mission(racine, "rm", "MISSION 6", "../data/mission_6.txt", courant, (mission+6), 6
00875     , 0 );
00876
00877     reussite = verification_mission( mission, 6, racine, courant);
00878     verification_reussite( mission, racine, reussite, 6 );
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02260
02261
02262
02263
02264
02265
02266
02267
02268
02269
02270
02271
02272
02273
02274
02275
02276
02277
02278
02279
02280
02281
02282
02283
02284
02285
02286
02287
02288
02289
02290
02291
02292
02293
02294
02295
02296
02297
02298
02299
02300
02301
02302
02303
02304
02305
02306
02307
02308
02309
02310
02311
02312
02313
02314
02315
02316
02317
02318
02319
02320
02321
02322
02323
02324
02325
02326
02327
02328
02329
02330
02331
02332
02333
02334
02335
02336
02337
02338
02339
02340
02341
02342
02343
02344
02345
02346
02347
02348
02349
02350
02351
02352
02353
02354
02355
02356
02357
02358
02359
02360
02361
02362
02363
02364
02365
02366
02367
02368
02369
02370
02371
02372
02373
02374
02375
02376
02377
02378
02379
02380
02381
02382
02383
02384
02385
02386
02387
02388
02389
02390
02391
02392
02393
02394
02395
02396
02397
02398
02399
02400
02401
02402
02403
02404
02405
02406
02407
02408
02409
02410
02411
02412
02413
02414
02415
02416
02417
02418
02419
02420
02421
02422
02423
02424
02425
02426
02427
02428
02429
02430
02431
02432
02433
02434
02435
02436
02437
02438
02439
02440
02441
02442
02443
02444
02445
02446
02447
02448
02449
02450
02451
02452
02453
02454
02455
02456
02457
02458
02459
02460
02461
02462
02463
02464
02465
02466
02467
02468
02469
02470
02471
02472
02473
02474
02475
02476
02477
02478
02479
02480
02481
02482
02483
02484
02485
02486
02487
02488
02489
02490
02491
02492
02493
02494
02495
02496
02497
02498
02499
02500
02501
02502
02503
02504
02505
02506
02507
02508
02509
02510
02511
02512
02513
02514
02515
02516
02517
02518
02519
02520
02521
02522
02523
02524
02525
02526
02527
02528
02529
02530
02531
02532
02533
02
```

```

00883 }
00884 //-----
00891 void mission_7(Mission *mission , fichier *racine )
00892 {
00893     bool reussite = false ;
00894     fichier *repertoire_courant = NULL;
00895     char *courant;
00896     courant = calloc(MOT , sizeof(char)) ;
00897
00898 // Preparation de la terrain de jeu
00899 repertoire_courant = my_cd( racine , racine , "Zone_Entrainement");
00900 my_touch( racine , repertoire_courant , "fichier_indesirable" ,0);
00901
00902 strcpy(mission[7].titre , "MISSION-7 : rm") ;
00903 mission[7].point = 2;
00904 affiche_fichier_tableau curse_position_exacte("MISSION 7" , "../data/mission_7.txt");
00905
00906 screen_clear();
00907 // lancement du prompt
00908 prompt_mission(racine , "rm" , "MISSION 7" , "../data/mission_7.txt" , courant , (mission+7) , 7
, 0 );
00909
00910 reussite = verification_mission(mission , 7 , racine , courant) ;
00911 if( reussite == true )
00912 {
00913     affiche_fichier_tableau curse_position_exacte("MISSION ACCOMPLIE",
00914     "../data/felicitation_7.txt");
00915
00916 verification_reussite( mission , racine , reussite , 7 );
00917 free(courant);
00918 }
00919 //-----
00926 void mission_8(Mission *mission , fichier *racine ) // Copier avec cp
00927 {
00928     bool reussite = false ;
00929     fichier *repertoire_courant = NULL;
00930     char *courant = calloc(4096 , sizeof(char)) ;
00931
00932     char tente_1[] = "Caserne/dortoirs/tente_1";
00933     char tente_2[] = "Caserne/dortoirs/tente_2";
00934     char tente_3[] = "Caserne/dortoirs/tente_3";
00935
00936 strcpy(mission[8].titre , "MISSION-8 : cp") ;
00937 mission[8].point = 2;
00938
00939 affiche_fichier_tableau curse_position_exacte("MISSION 8" , "../data/mission_8.txt");
00940
00941 // Déplacement dans le repertoire du partie
00942     repertoire_courant = my_cd(racine , racine , "Hangar" ) ;
00943     my_touch(racine , repertoire_courant , "caisse" , 0 );
00944
00945 screen_clear();
00946
00947 // lancement du prompt
00948 prompt_mission(racine , "cp" , "MISSION 8" , "../data/mission_8.txt" , courant , (mission+8) , 8
, 0 );
00949
00950 reussite = verification_mission(mission , 8 , racine , courant);
00951 free(courant);
00952 verification_reussite( mission , racine , reussite , 8 );
00953 }
00954
00955 //-----
00956
00963 void mission_9(Mission *mission , fichier *racine ) // Déplacer avec mv
00964 {
00965     bool reussite = false ;
00966     fichier *repertoire_courant = NULL;
00967     char *courant;
00968     courant = calloc(MOT , sizeof(char)) ;
00969
00970 strcpy(mission[9].titre , "MISSION-9 : echo") ;
00971 mission[9].point = 2;
00972
00973 affiche_fichier_tableau curse_position_exacte("MISSION 9" , "../data/mission_9.txt");
00974
00975 // Déplacement dans le repertoire du partie
00976     repertoire_courant = my_cd(racine , racine , "Hangar" ) ;
00977     my_touch(racine , repertoire_courant , "caisse" , 0 );
00978
00979 screen_clear();
00980 // lancement du prompt
00981 prompt_mission(racine , "mv" , "MISSION 9" , "../data/mission_9.txt" , courant , (mission+9) , 9 ,

```

```
0 );
00982     reussite = verification_mission(mission , 9 , racine , courant );
00984     verification_reussite( mission , racine , reussite , 9 );
00985
00986     free(courant);
00987 }
00988
00989
//-----
00996 void mission_10(Mission *mission , fichier *racine )
00997 {
00998     bool reussite = false ;
00999     fichier *repertoire_courant = NULL;
01000     char *courant;
01001     courant = calloc(MOT , sizeof(char)) ;
01002
01003     strcpy(mission[10].titre , "MISSION-10 : echo") ;
01004     mission[10].point = 2;
01005
01006     affiche_fichier_tableau curse_position_exacte("MISSION 10", "../data/mission_10.txt");
01007
01008
// Deplacement dans la terrain de jeu
01009     repertoire_courant = my_cd( racine , racine , "Zone_Entrainement");
01010
// lancement du prompt
01012     reussite = prompt_mission(repertoire_courant , "echo" , "MISSION 10" , "../data/mission_10.txt" ,
01013     courant , (mission+10) , 10 , 1 );
01014
// On verifie s'il a au moins taper echo
01015     if ( reussite )
01016     {
01018         affiche_fichier_tableau curse_position_exacte("MISSION ACCOMPLIE",
01019         "../data/felicitation_10.txt");
01020
01021         verification_reussite( mission , racine , reussite , 10 );
01022
01023     free(courant);
01024 }
01025
01026 /*****
01033 void mission_11(Mission *mission , fichier *racine)
01034 {
01035     bool reussite ;
01036     fichier *repertoire_courant ;
01037     char *courant;
01038
01039     reussite = false;
01040     courant = calloc( MOT , sizeof(char)) ;
01041     repertoire_courant = NULL;
01042
01043     strcpy(mission[11].titre , "MISSION-11 : cat") ;
01044     mission[11].point = 3;
01045
01046     affiche_fichier_tableau curse_position_exacte("MISSION 11", "../data/mission_11.txt");
01047
01048
// Preparation : terrain de jeu
01049     repertoire_courant = my_cd( racine , racine , "Laboratoire");
01050     repertoire_courant = my_cd( racine , repertoire_courant , "archives_secretes");
01051     my_touch(racine , repertoire_courant , "convers.crpt" , 0);
01052
01054     screen_clear();
01055
// lancement du prompt
01056     reussite = prompt_mission(racine , "cat convers.crpt" , "MISSION 11" , "../data/mission_11.txt" ,
01057     courant , (mission+11) , 11 , 1 );
01058
// On verifie s'il a fait le mission correspondant
01059     if ( reussite )
01060     {
01061         affiche_fichier_tableau curse_position_exacte("MISSION ACCOMPLIE",
01062         "../data/felicitation_11.txt");
01063
01064         verification_reussite( mission , racine , reussite , 11 );
01065
01066     free(courant);
01067 }
01068
01069 /*****
01076 void mission_12(Mission *mission , fichier *racine)
01077 {
01078     bool reussite ;
01079     fichier *repertoire_courant ;
01080     char *courant;
```

```

01081    reussite = false ;
01082    courant = calloc(MOT , sizeof(char)) ;
01083    repertoire_courant = NULL ;
01084
01085    strcpy(mission[12].titre , "MISSION-12 : head") ;
01086    mission[12].point = 3;
01087
01088    affiche_fichier_tableau curse_position_exacte("MISSION 12", ".../data/mission_12.txt");
01089
01090
01091    // Preparation : terrain de jeu
01092    repertoire_courant = my_cd( racine , racine , "Laboratoire");
01093    repertoire_courant = my_cd( racine , repertoire_courant , "archives_secretes");
01094    my_touch(racine , repertoire_courant , "convers.crpt" , 0);
01095
01096    screen_clear();
01097    // lancement du prompt
01098    reussite = prompt_mission(racine , "head -n 5 convers.crpt" , "MISSION 12" ,
01099        ".../data/mission_12.txt" , courant , (mission+12) , 12 , 1 );
01100
01101    // On verifie s'il a fait la mission correspondant
01102    if ( reussite )
01103    {
01104        affiche_fichier_tableau curse_position_exacte("MISSION ACCOMPLIE",
01105            ".../data/felicitation_12.txt");
01106    }
01107
01108    verification_reussite( mission , racine , reussite , 12 );
01109
01110    free(courant);
01111
01112 /*****+
01113 void mission_13(Mission *mission , fichier *racine)
01114 {
01115     bool reussite ;
01116     fichier *repertoire_courant ;
01117     char *courant;
01118
01119     reussite = false;
01120     courant = calloc(MOT , sizeof(char)) ;
01121     repertoire_courant = NULL;
01122
01123     strcpy(mission[13].titre , "MISSION-13 : tail") ;
01124     mission[13].point = 3;
01125
01126     affiche_fichier_tableau curse_position_exacte("MISSION 13", ".../data/mission_13.txt");
01127
01128    // Preparation : terrain de jeu
01129    repertoire_courant = my_cd( racine , racine , "Laboratoire");
01130    repertoire_courant = my_cd( racine , repertoire_courant , "archives_secretes");
01131    my_touch(racine , repertoire_courant , "convers.crpt" , 0);
01132
01133    screen_clear();
01134
01135    // lancement du prompt
01136    reussite = prompt_mission(racine , "tail -n 5 convers.crpt" , "MISSION 13" ,
01137        ".../data/mission_13.txt" , courant , (mission+13) , 13 , 1 );
01138
01139    // On verifie s'il a fait la mission correspondant
01140    if ( reussite )
01141    {
01142        affiche_fichier_tableau curse_position_exacte("MISSION ACCOMPLIE",
01143            ".../data/felicitation_13.txt");
01144    }
01145
01146    verification_reussite( mission , racine , reussite , 13 );
01147
01148    free(courant);
01149
01150
01151 /*****+
01152 void entrainement_14_1(Mission *mission , fichier *racine)
01153 {
01154     fichier *repertoire_courant = NULL , *repertoire_courant1 = NULL ;
01155     char *courant = calloc(MOT , sizeof(char)) ;
01156     bool reussite = false ;
01157
01158    // Preparation du terrain d'entraînement
01159    repertoire_courant = my_cd( racine , racine , "Zone_Entrainement" ) ;
01160    my_mkdir(racine , repertoire_courant , "Coffre_Fort" , 1 ) ;
01161    repertoire_courant1 = my_cd( racine , repertoire_courant , "Coffre_Fort" ) ;
01162    my_touch(racine , repertoire_courant1 , "Gilet" , 0 ) ;
01163    my_touch(racine , repertoire_courant1 , "Munitions" , 0 ) ;
01164    my_touch(racine , repertoire_courant1 , "AK-47" , 0 ) ;

```

```

01170     strcpy(repertoire_courant->perm , "---") ;
01171
01172 // Lancement du jeu
01173 affiche_fichier_tableau curse_position_exacte("ENTRAÎNEMENT", "../data/test_14.txt");
01174 prompt_mission( repertoire_courant , "chmod" , "MISSION 14" , "../data/test_14.txt" , courant ,
01175 (mission+14) , 141 , 0 ) ;
01176
01177 // Vérification du statuts mission
01178 reussite = verification_mission(mission , 141 , racine , courant );
01179 if(reussite)
01180 {
01181     affiche_fichier_tableau curse_position_exacte("Entraînement réussi",
01182     "../data/felicitation_14_1.txt");
01183 }
01184 free(courant);
01185
01186 void prepa_14(fichier * racine)
01187 {
01188     fichier *repertoire_courant = NULL ;
01189
01190 // Préparation : terrain de jeu
01191 repertoire_courant = my_cd( racine , racine , "Laboratoire") ;
01192 repertoire_courant = my_cd( racine , repertoire_courant , "archives_secretes") ;
01193     my_touch(racine , repertoire_courant , "R+8" , 0) ;
01194     my_touch(racine , repertoire_courant , ".Satan_2" , 0) ;
01195     my_touch(racine , repertoire_courant , "enigma" , 0) ;
01196     my_touch(racine , repertoire_courant , "sigma" , 0) ;
01197     my_touch(racine , repertoire_courant , "Q-2" , 0) ;
01198     my_touch(racine , repertoire_courant , "Section_14" , 0) ;
01199     my_touch(racine , repertoire_courant , ".Manhattan" , 0) ;
01200     my_touch(racine , repertoire_courant , "Blame" , 0) ;
01201     my_touch(racine , repertoire_courant , ".Abdul" , 0) ;
01202     my_touch(racine , repertoire_courant , "Falcon" , 0) ;
01203     my_touch(racine , repertoire_courant , "Eclipse" , 0) ;
01204     my_touch(racine , repertoire_courant , "Shadow" , 0) ;
01205     my_touch(racine , repertoire_courant , ".Vortex" , 0) ;
01206     my_touch(racine , repertoire_courant , ".Ironclad" , 0) ;
01207     my_touch(racine , repertoire_courant , "Phantom" , 0) ;
01208     my_touch(racine , repertoire_courant , "Aurora" , 0) ;
01209     my_touch(racine , repertoire_courant , ".Titan" , 0) ;
01210     strcpy( repertoire_courant->perm , "---" ) ;
01211 }
01212 int demande_avis()
01213 {
01214     int choix ;
01215     char texte[]={ "Souhaitez-tu commencer par l'entraînement ou pas soldat ??[Y/N] :" } ;
01216
01217     screen_clear();
01218
01219     initscr();           // initialise ncurses
01220     clear();
01221     noecho();            // ne pas afficher les touches tapées
01222     cbreak();             // entrée sans mise en tampon
01223     curs_set(0);          // cache le curseur
01224
01225
01226     mvprintw( LINES /2 , (COLS -strlen(texte)) /2 , "%s" , texte ) ;
01227
01228     do
01229     {
01230         choix = getch();
01231     } while (choix != 'Y' && choix != 'y' && choix != 'N' && choix && 'n') ;
01232
01233     if (choix == 'Y' || choix == 'y')
01234     {
01235         endwin(); // restaure le terminal
01236         clear();
01237         return 1 ;
01238     }
01239     else if (choix == 'N' || choix == 'n')
01240     {
01241         endwin(); // restaure le terminal
01242         clear();
01243         return 0 ;
01244     }
01245 }
01252 void mission_14(Mission *mission , fichier *racine) // Chmod
01253 {
01254     bool reussite = false ;
01255     char *courant ;
01256     int max_sec = 90 ;
01257
01258     courant = calloc(MOT , sizeof(char)) ;
01259     strcpy(mission[14].titre , "MISSION-14 : chmod");

```

```

01261     mission[14].point = 3;
01262     // pipe pour recevoir le resultat
01263     int fd[2];
01264     pipe(fd);
01265
01266
01267     // Demande si le joueur veut recommencer au cours au passer juste au mission
01268     if( mission[13].terminé == 1 )
01269     {
01270         if(demande_avis() == 1 ) // Si l'user veux de tout recommencer
01271         {
01272             // Lancement du première entrainement
01273             affiche_fichier_tableau curse_position_exacte("Chmod", ".../data/cours_14.txt");
01274             entrainement_14_1(mission , racine ) ;
01275
01276             // Preparation : terrain de jeu
01277             prepa_14(racine);
01278             screen_clear() ;
01279
01280             // lancement du vraie mission 14 .
01281             affiche_fichier_tableau curse_position_exacte("Chmod", ".../data/mission_14.txt");
01282             affiche_fichier_tableau curse_position_exacte("MISSION 14 - TEMPS LIMITE : 90
01283             secondes", ".../data/vide");
01284
01285             // ----- FILS 1 : Chrono -----
01286             pid_t pid_chrono = fork();
01287             if(pid_chrono == 0)
01288             {
01289                 for(int i = 0; i < max_sec*10; ++i)
01290                 {
01291                     usleep(100000);
01292
01293                     exit(1); // code 1 = temps écoulé
01294                 }
01295
01296             // ----- FILS 2 : Mission -----
01297             pid_t pid_mission = fork();
01298             if(pid_mission == 0)
01299             {
01300                 // fermeture du pide lecture
01301                 close(fd[0]);
01302
01303                 // exécute prompt_mission dans un sous-fils
01304                 prompt_mission(racine , "chmod" , "MISSION 14" , ".../data/mission_14.txt" , courant
01305                 , (mission+14) , 14 , 0 );
01306                 réussite = verification_mission(mission , 14 , racine , courant ) ;
01307
01308                 // Envoie le resultat au père
01309                 write( fd[1] , &réussite , sizeof(bool) );
01310                 close( fd[1] ) ;
01311                 exit(0) ;
01312             }
01313
01314             // ----- PÈRE : supervise fils chrono et fils mission -----
01315             int status;
01316             pid_t fini;
01317
01318             while(1)
01319             {
01320                 // vérifie si le fils mission a terminé
01321                 fini = waitpid(pid_mission, &status, WNOHANG);
01322                 if(fini == pid_mission)
01323                 {
01324                     // tuer le chrono si la mission a fini
01325                     kill(pid_chrono, SIGKILL);
01326                     break;
01327                 }
01328
01329                 // vérifie si le chrono a fini
01330                 fini = waitpid(pid_chrono, &status, WNOHANG);
01331                 if(fini == pid_chrono)
01332                 {
01333                     // temps écoulé + tuer le fils mission
01334                     kill(pid_mission, SIGKILL);
01335                     affiche_fichier_tableau curse_position_exacte("TEMPS ÉCOULÉ !" , ".../data/vide");
01336                     réussite = false ;
01337                     break;
01338
01339                     usleep(5000);
01340                 }
01341
01342             // nettoyage des fils
01343             waitpid(pid_chrono, NULL, 0);
01344             waitpid(pid_mission, NULL, 0);
01345         }

```

```

01346     else
01347     {
01348         // Preparation : terrain de jeu
01349         prepa_14(racine);
01350         screen_clear() ;
01351
01352         // lancement du vraie mission 14 .
01353         affiche_fichier_tableau curse_position_exacte("Chmod", ".../data/mission_14.txt");
01354         prompt_mission(racine , "chmod" , "MISSION 14" , ".../data/mission_14.txt" , courant
01355         , (mission+14) , 14 , 0 );
01356     }
01357 }
01358 else // Si la mission n'etait pas encore terminé avant
01359 {
01360     // Lancement du première entrainement
01361     affiche_fichier_tableau curse_position_exacte("Chmod", ".../data/cours_14.txt");
01362     // entrainement_14_1(mission , racine ) ;
01363
01364     // Preparation : terrain de jeu
01365     prepa_14(racine);
01366     screen_clear() ;
01367
01368     // lancement du vraie mission 14 .
01369     affiche_fichier_tableau curse_position_exacte("MISSION 14" , ".../data/mission_14.txt");
01370     affiche_fichier_tableau curse_position_exacte("MISSION 14 - TEMPS LIMITE : 90 secondes" ,
01371     ".../data/vide");
01372
01373     pid_t pid_chrono = fork();
01374     if(pid_chrono == 0)
01375     {
01376         for(int i = 0; i < max_sec*10; ++i)
01377         {
01378             usleep(100000);
01379         }
01380
01381         exit(1); // code 1 = temps écoulé
01382     }
01383
01384 // ----- FILS 2 : Mission -----
01385 pid_t pid_mission = fork();
01386 if(pid_mission == 0)
01387 {
01388     // fermeture du pide lecture
01389     close(fd[0]);
01390
01391     // exécute prompt_mission dans un sous-fils
01392     prompt_mission(racine , "chmod" , "MISSION 14" , ".../data/mission_14.txt" , courant
01393     , (mission+14) , 14 , 0 );
01394     reussite = verification_mission(mission , 14 , racine , courant) ;
01395
01396     // Envoie le resultat au père
01397     write( fd[1] , &reussite , sizeof(bool) );
01398     close(fd[1]) ;
01399     exit(0) ;
01400 }
01401
01402 // ----- PÈRE : supervise fils chrono et fils mission -----
01403 int status;
01404 pid_t fini;
01405
01406 while(1)
01407 {
01408     // vérifie si le fils mission a terminé
01409     fini = waitpid(pid_mission, &status, WNOHANG);
01410     if(fini == pid_mission)
01411     {
01412         // tuer le chrono si la mission a fini
01413         kill(pid_chrono, SIGKILL);
01414         break;
01415     }
01416
01417     // vérifie si le chrono a fini
01418     fini = waitpid(pid_chrono, &status, WNOHANG);
01419     if(fini == pid_chrono)
01420     {
01421         // temps éoulé + tuer le fils mission
01422         kill(pid_mission, SIGKILL);
01423         affiche_fichier_tableau curse_position_exacte("TEMPS ÉCOULÉ !" , ".../data/vide");
01424         reussite = false ;
01425         break;
01426     }
01427
01428     usleep(5000);
01429 }

```

```

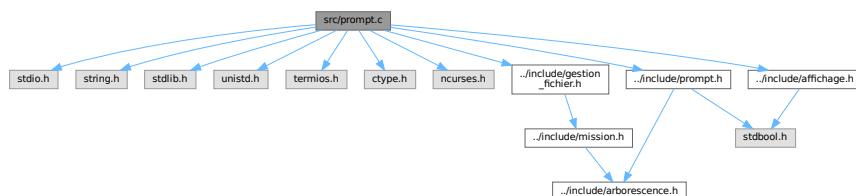
01430         // nettoyage des fils
01431         waitpid(pid_chrono, NULL, 0);
01432         waitpid(pid_mission, NULL, 0);
01433     }
01434     // Reception du resultat du fils mission
01435     close(fd[1]); // fermer écriture
01436     read( fd[0] , &reussite , sizeof(bool) );
01437     close(fd[0]);
01438
01439     // Vérification de la réussite de la mission
01440     verification_reussite( mission , racine , reussite , 14 );
01441
01442     free(courant);
01443 }
```

## 4.27 src/prompt.c File Reference

ceci fait reference à la simulation de prompt

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <termios.h>
#include <ctype.h>
#include <ncurses.h>
#include "../include/gestion_fichier.h"
#include "../include/prompt.h"
#include "../include/affichage.h"
```

Include dependency graph for prompt.c:



### Functions

- void **affiche\_prompt** (fichier \*repertoire\_actuel)
 

*Quelques fonctions de Olivier utilise cette bibliothèques.*
- void **efface\_cara** (char \*chaine, int \*position)
 

*On lui donne une chaine de caractere et une position ; Il supprime le caractere qui se trouve à cette position.*
- int **my\_get\_ligne** (char \*mobile, int taille\_Max, fichier \*repertoire\_actuel)
- void **affiche\_commande** (int \*indice, char caractere, char \*chaine, int droite)
 

*Elle permet d'insérer une caractere dans une chaine et d'afficher la chaine.*
- void **touches\_gauche\_droite** (int \*indice, int \*pos\_droite, char \*chaine, int gauche)
 

*c'est la fonction associer à la touche gauche et droite*
- void **touches\_haut\_bas** (int \*indice, int \*position, int \*droite, char \*chaine, char \*tmp, int haut)
 

*la fonctionne associer à la touche haut et bas*
- void **touche\_suppr** (int \*position, int \*pos\_droite, char \*chaine)
 

*c'est la fonction associer à la touche suppr*

- void [vide\\_ligne](#) (int droite, char \*chaine)
 

*c'est une simple fonction qui permet de vider la ligne de commande actuelle*
- int [taille\\_de](#) (char \*ligne)
 

*elle ne compte pas les caractères comme strlen ; elle donne la taille en affichage*
- int [auto\\_completion](#) (char \*chaine, [fichier](#) \*repertoire\_actuel, int \*nb\_Tab, int \*pos\_droite)
 

*c'est la fonction principale de l'auto-completion sur ligne de commande*
- void [inserer\\_mot\\_dans](#) (int position, char \*chaine, char \*insertion)
 

*c'est pour inserer une mot dans une position précise dans une chaine ; séparateur = espace*
- int [trouver\\_avant](#) (char \*chaine, char \*recherche)
 

*Elle permet de trouver si la chaine recherche est le début de chaine ; les espaces sont ignorés.*
- int [compter\\_mot](#) (char \*chaine)
 

*fonction qui compte le nombre de mot dans une chaine de caractère*
- int [prend\\_mot\\_dans](#) (int position, char \*chaine, char \*emplacement)
 

*Elle permet d'inserer une caractere dans une chaine à une position préciser en argument.*
- void [inserer](#) (char caractere, int position, char \*chaine)
 

*trouver le nom du joueur actuelle et puis l'inserer dans emplacement*
- int [get\\_point](#) ()
 

*c'est pour savoir le point que le joueur actuel possède*
- void [increment\\_score](#) (int actuPTS)
 

*c'est pour augmenter le nombre de score du joueur actuel*
- int [decrement\\_score](#) (int penalite)
 

*c'est une fonction qui fait decremener le nombre de score d'une joueur en jeu*
- void [get\\_pwd](#) ([fichier](#) \*adresse, char \*emplacement)
 

*trouver le chemin du repertoire actuelle et puis l'inserer dans emplacement*
- int [commande\\_option](#) ([ligne\\_commande](#) \*ligne, char \*\*liste, int nb\_mot)
 

*cette fonction represente une ligne de commande dans une structure ligne\_commande*
- [fichier](#) \* [execute](#) (char \*\*commande, [fichier](#) \*repertoire\_actuel, int nbMot)
 

*execution du commande entrée en argument*
- int [rapport\\_mission](#) (int num\_mission, bool réussie)
 

*celui ci écrit le joueur , temps de jeu , réussie ou pas et numero de mission dans le fichier d'enregistrement*

### 4.27.1 Detailed Description

ceci fait référence à la simulation de prompt

#### Author

Valisoa

Il est regroupe les fonctions utiles pour notre petite terminale ;

- affiche le repertoire courant ; le nom d'utilisateur ; la machine
- execute le commande decrit

Definition in file [prompt.c](#).

## 4.27.2 Function Documentation

### 4.27.2.1 affiche\_commande()

```
void affiche_commande (
    int * indice,
    char caractere,
    char * chaine,
    int droite )
```

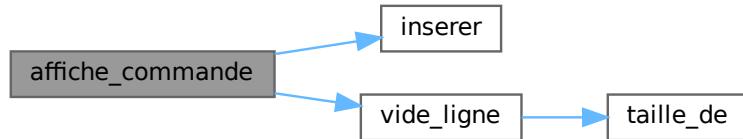
Elle permet d'insérer un caractère dans une chaîne et d'afficher la chaîne.

#### Parameters

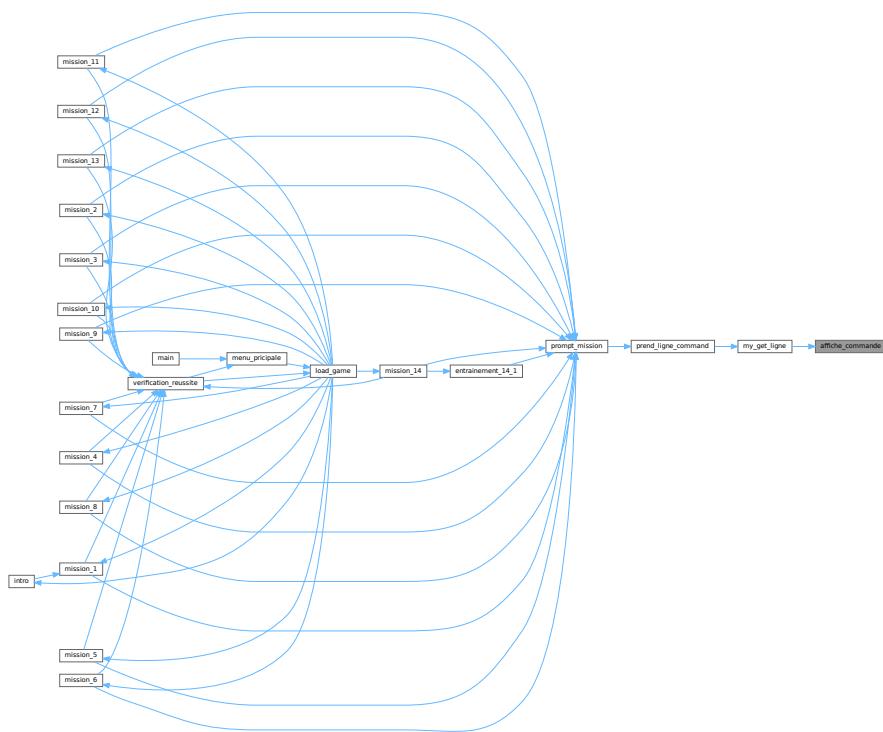
<i>indice</i>	la position d'insertion dans la chaîne
<i>caractere</i>	le caractère à insérer dans la chaîne
<i>chaine</i>	la chaîne où l'on va insérer un caractère
<i>droite</i>	la position actuelle en se référant de la position la plus à droite

Definition at line 213 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.27.2.2 affiche\_prompt()

```
void affiche_prompt (
    fichier * repertoire_actuel )
```

Quelques fonctions de Olivier utilise cette bibliotheques.

il affiche le ligne de prompt où l'on insere du commande

<ncurses.h>

"./include/gestion\_fichier.h"

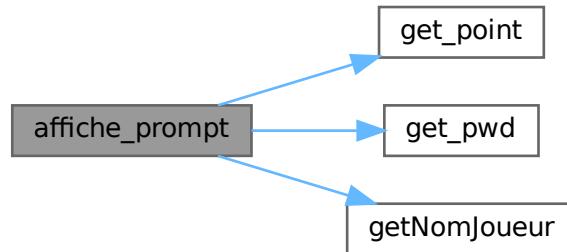
pour la gestion des fichiers ( adaptation des mes besoins )

"./include/prompt.h"

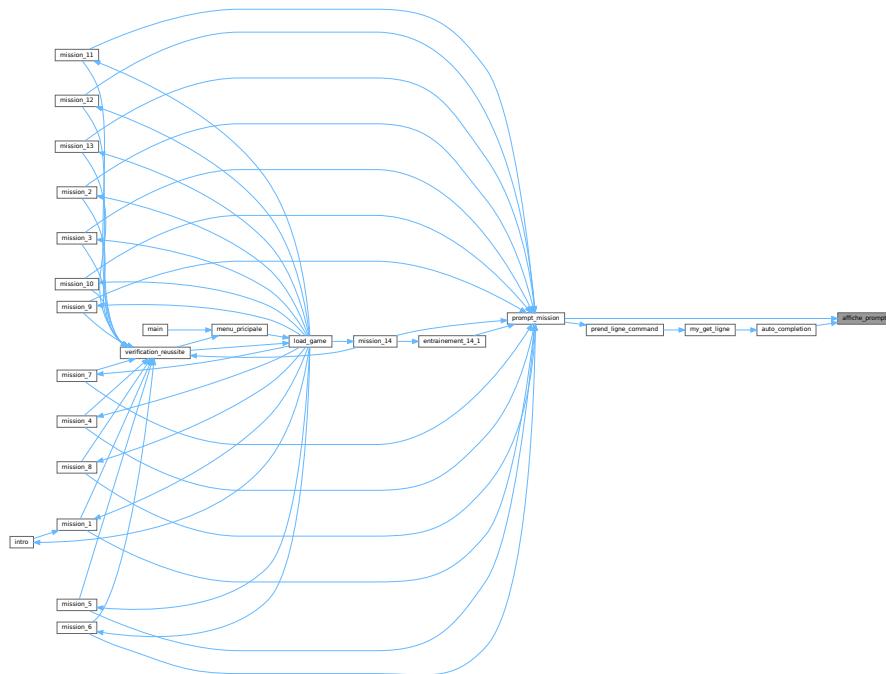
inclusion de tous les fonctions de ce fichier [ certains appelle les autres ]

Definition at line 35 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.27.2.3 auto\_completion()

```

int auto_completion (
    char * chaine,
    fichier * repertoire_actuel,
    int * nb_Tab,
    int * pos_droite )
  
```

c'est la fonction principale de l'auto-completion sur ligne de commande

### Parameters

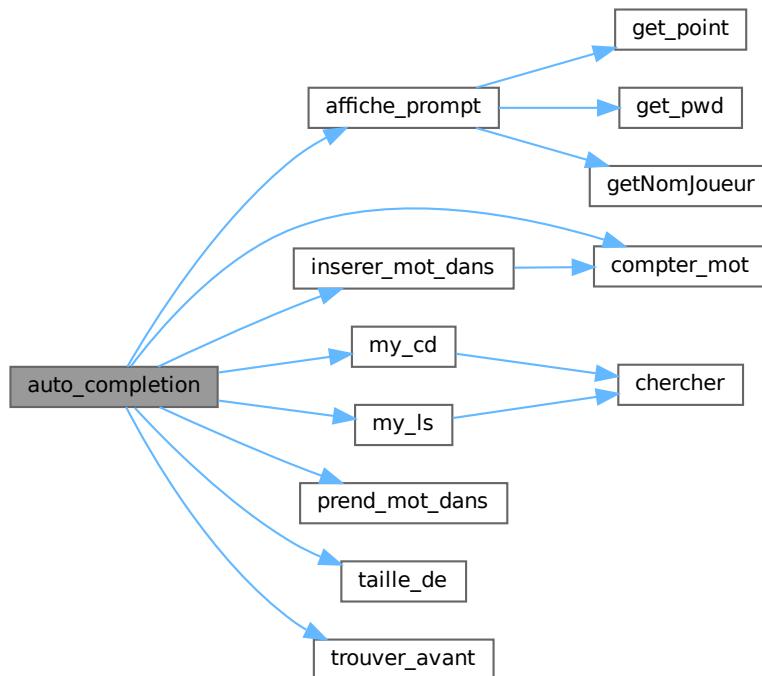
<i>chaine</i>	la chaine à completer
<i>repertoire_actuel</i>	l'adresse du repertoire actuelle
<i>nb_Tab</i>	nombre de tabulation déjà effectuer
<i>pos_droite</i>	la position actuelle par rapport à la position la plus à droite

### Returns

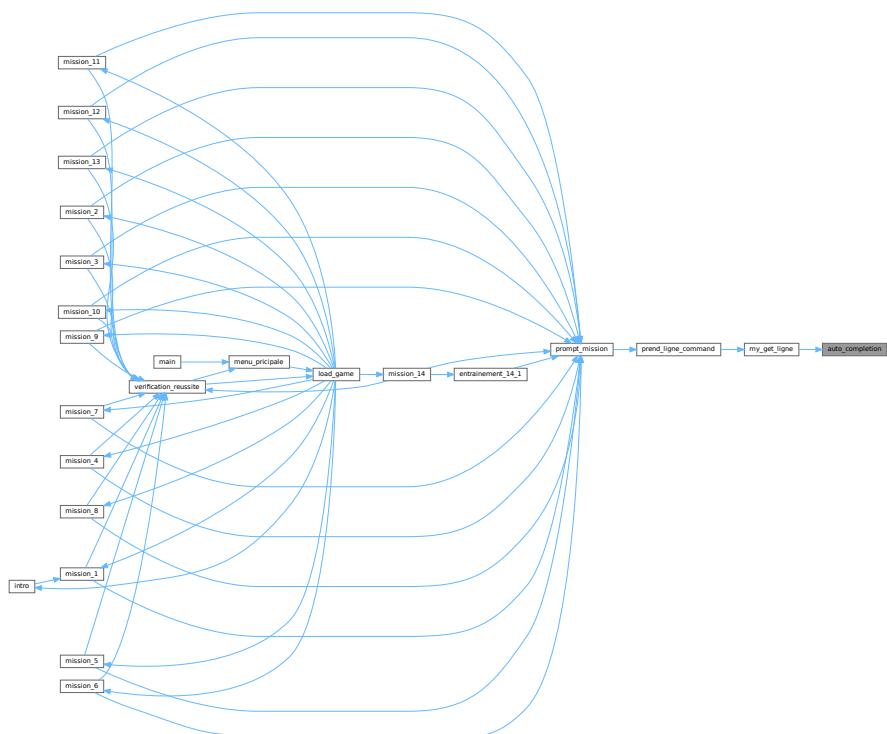
la taille de la chaine en affichage ; donc réussi si différent de zero

Definition at line 408 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.27.2.4 commande\_option()

```
int commande_option (
    ligne_commande * ligne,
    char ** liste,
    int nb_mot )
```

cette fonction represente une ligne de commande dans une structure `ligne_commande`

##### Parameters

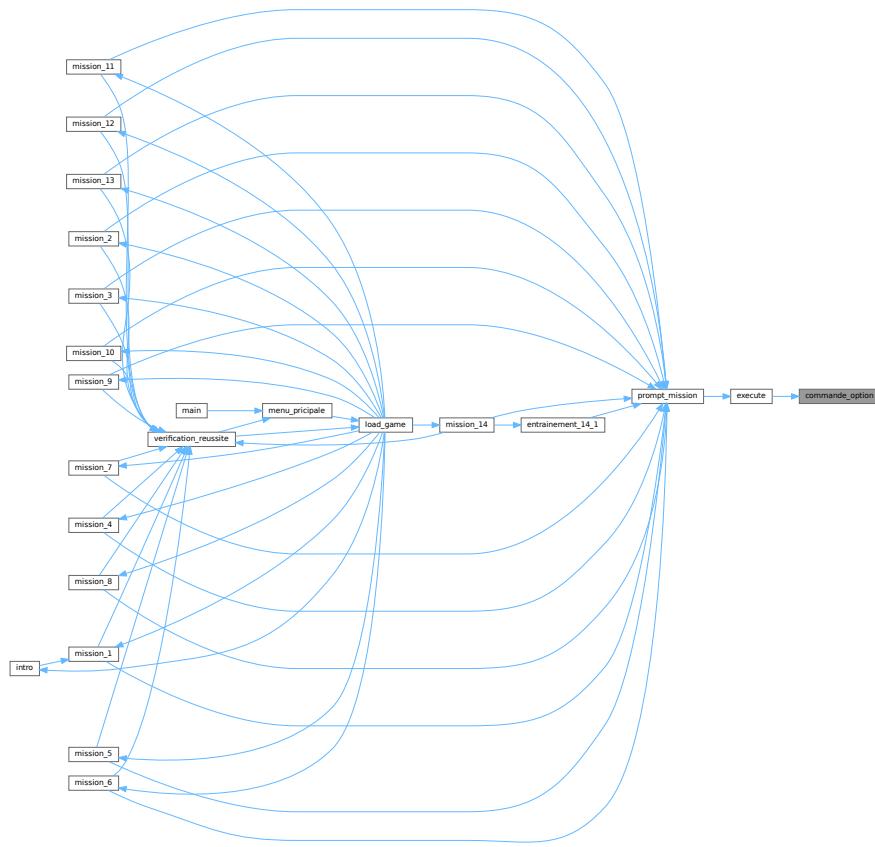
<code>ligne</code>	l'adresse du structure où l'on stockera la representation de la ligne de commande
<code>liste</code>	la ligne actuelle deja separer mot par mot ( dans une tableau mot )
<code>nb_mot</code>	le nombre de mot dans liste

##### Returns

0 si tous s'est bien passer

Definition at line 1305 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.27.2.5 compter\_mot()

```
int compter_mot (
    char * chaine )
```

fonction qui compte le nombre de mot dans une chaine de caractere

##### Parameters

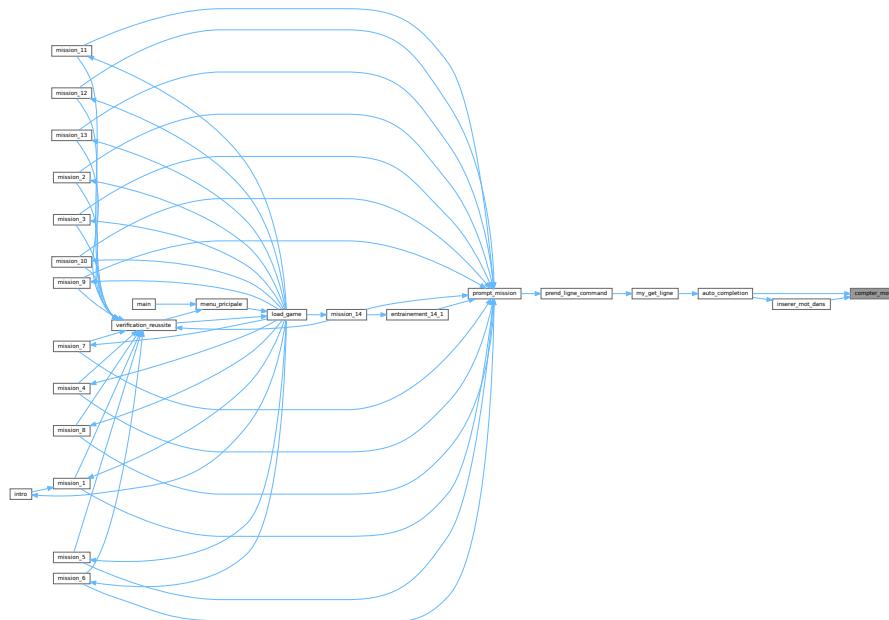
<i>chaine</i>	la chaine dont on veut compter le nombre de mot
---------------	-------------------------------------------------

##### Returns

le nombre de mot dans chaine

Definition at line 908 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.27.2.6 decrement\_score()

```
int decrement_score (
    int penalite )
```

c'est une fonction qui fait decremener le nombre de score d'une joueur en jeu

##### Parameters

<i>penalite</i>	c'est le nombre de point à enlever dans le nombre de score
-----------------	------------------------------------------------------------

##### Returns

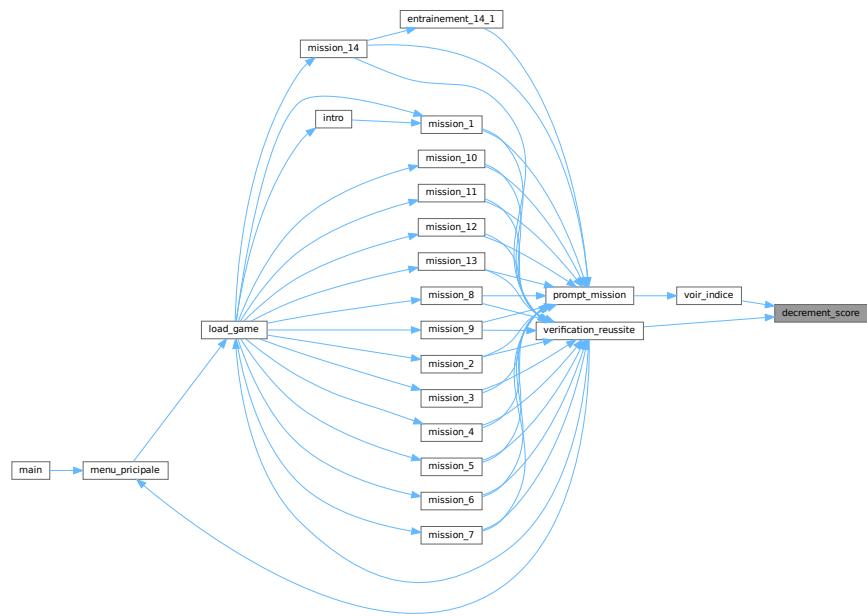
renvoie une entier negatif si le score est insuffisant ou n'existe pas encore

Definition at line 1214 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.27.2.7 efface\_cara()

```

void efface_cara (
    char * chaine,
    int * position )
  
```

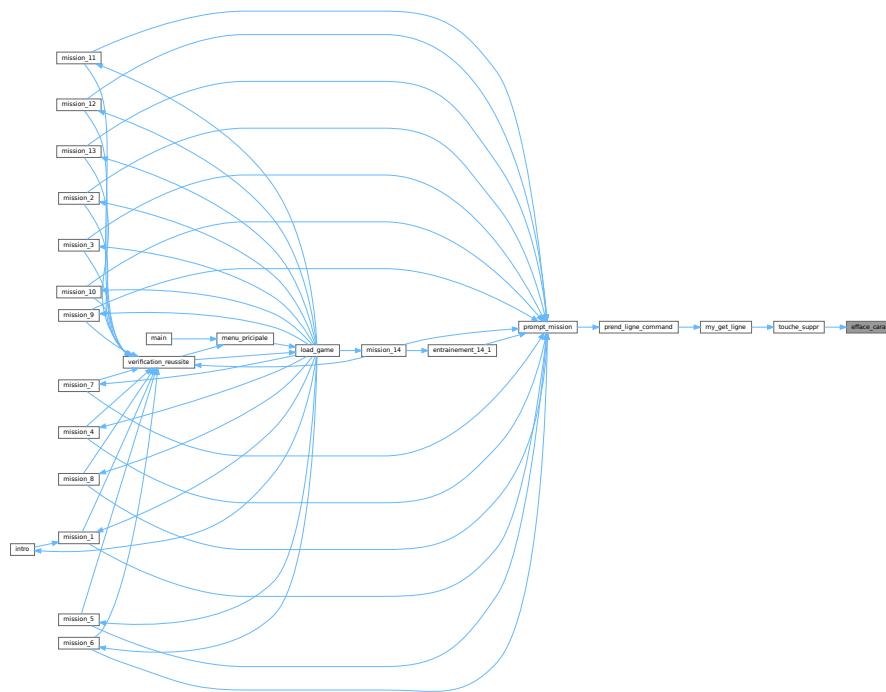
On lui donne une chaine de caractere et une position ; Il supprime le caractere qui se trouve à cette position.

##### Parameters

<i>chaine</i>	la chaine de caractere dont on veut supprimer un caractere
<i>position</i>	la position du caractere dans la chaine

Definition at line 52 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.27.2.8 execute()

```
fichier * execute (
    char ** commande,
    fichier * repertoire_actuel,
    int nbMot )
```

execution du commande entrée en argument

##### Parameters

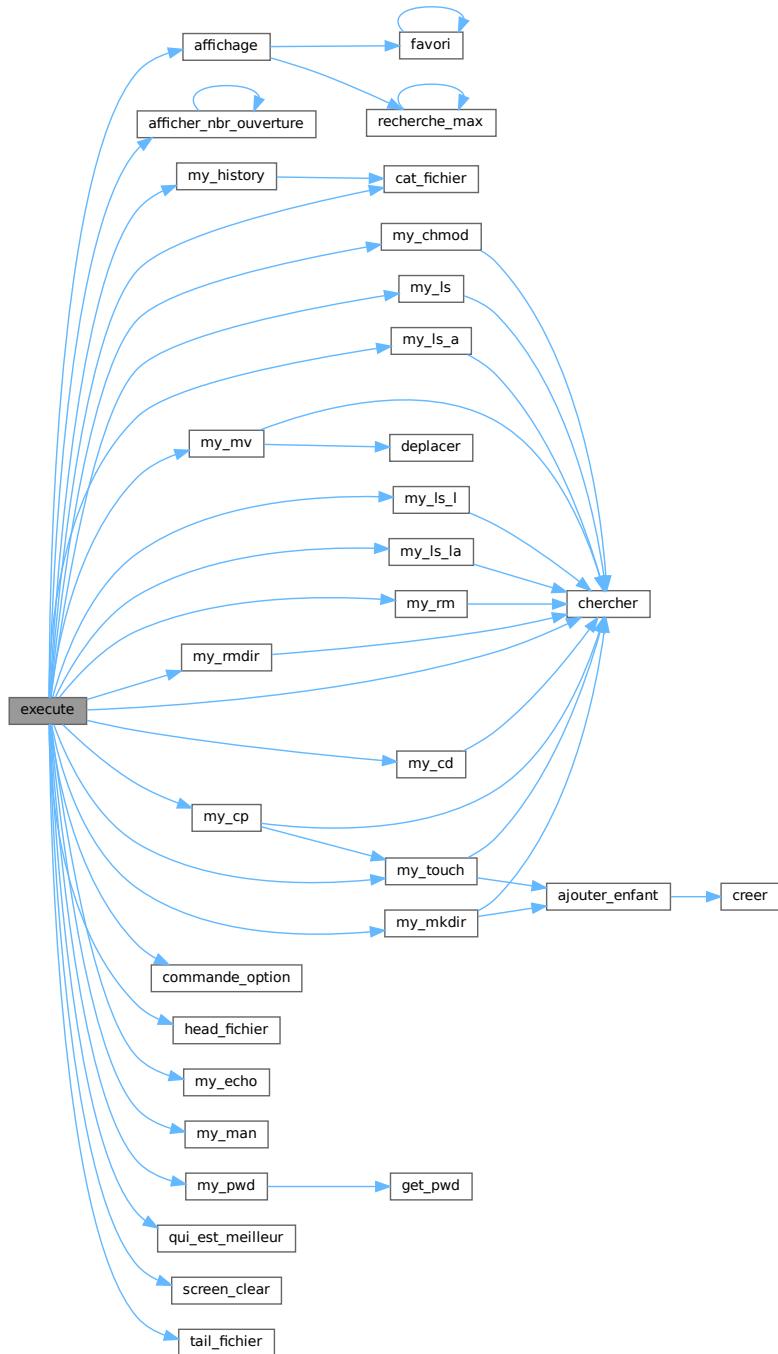
<i>commande</i>	La ligne entrer par l'utilisateur
<i>repertoire_actuel</i>	l'adresse du repertoire où l'on travaille actuellement
<i>nbMot</i>	le nombre de mot prise dans commande qui est de type char**

##### Returns

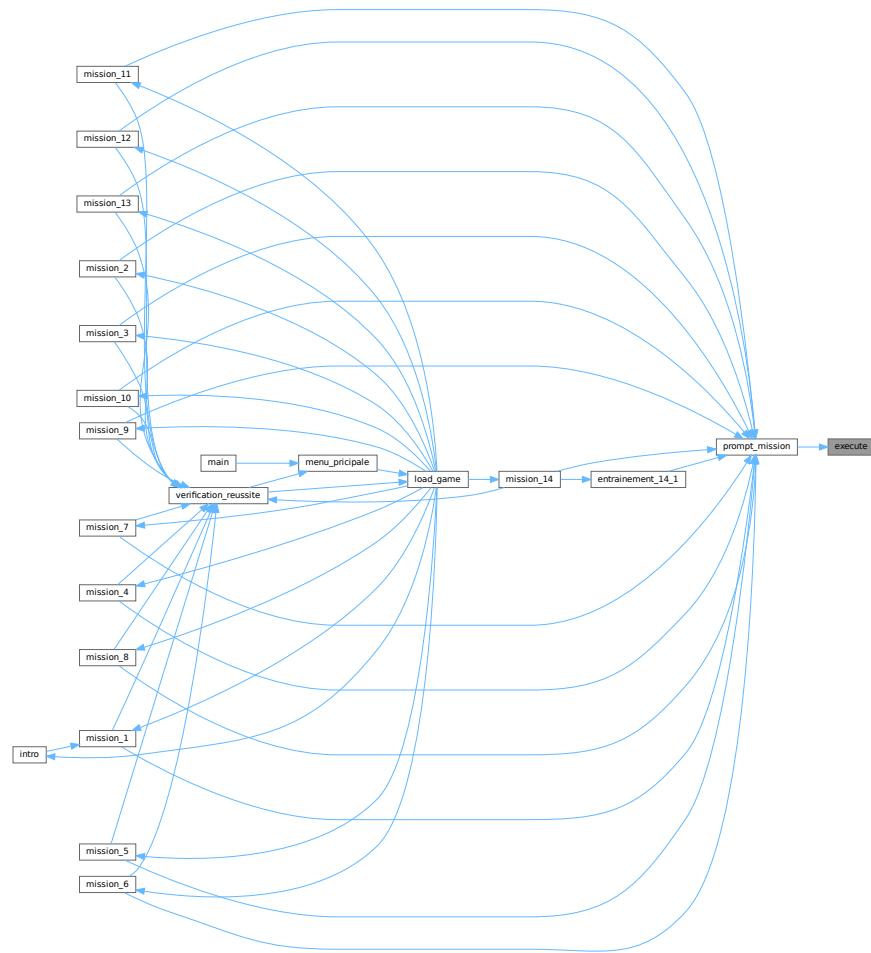
l'adresse du repertoire de travaille qui a peut etre changé durant le prompt

Definition at line 1349 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.27.2.9 get\_point()

```
int get_point ( )
```

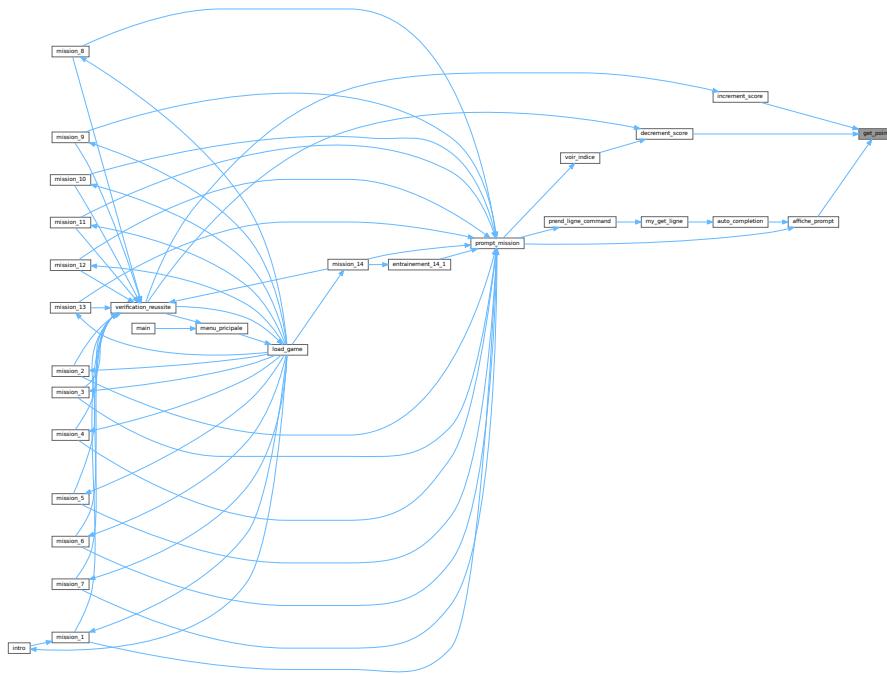
c'est pour savoir le point que le joueur actuel possède

## Returns

il return le nombre de point de l'utilisateur actuel

Definition at line 1121 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.27.2.10 get\_pwd()

```
void get_pwd (
    fichier * adresse,
    char * emplacement )
```

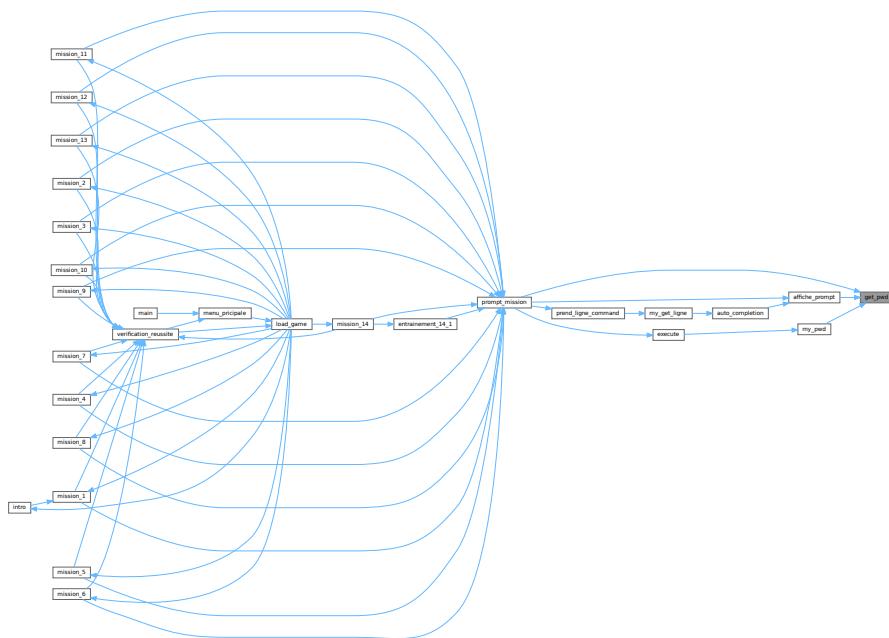
trouver le chemin du repertoire actuelle et puis l'insérer dans emplacement

##### Parameters

<i>adresse</i>	l'emplacement actuelle dans l'arborescence
<i>emplacement</i>	la variable ou stocker le resultat

Definition at line 1260 of file [prompt.c](#).

Here is the caller graph for this function:



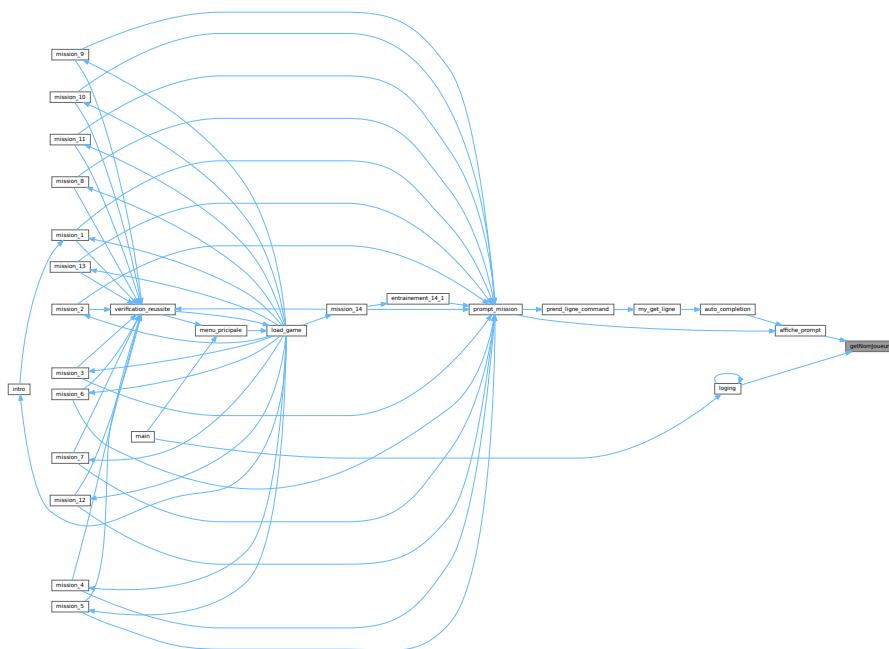
#### 4.27.2.11 getNomJoueur()

```
int getNomJoueur (
    char * emplacement )
```

trouver le nom du joueur actuelle et puis l'inserer dans emplacement

Definition at line 1104 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.27.2.12 increment\_score()

```
void increment_score (
    int actuPTS )
```

c'est pour augmenter le nombre de score du joueur actuel

##### Parameters

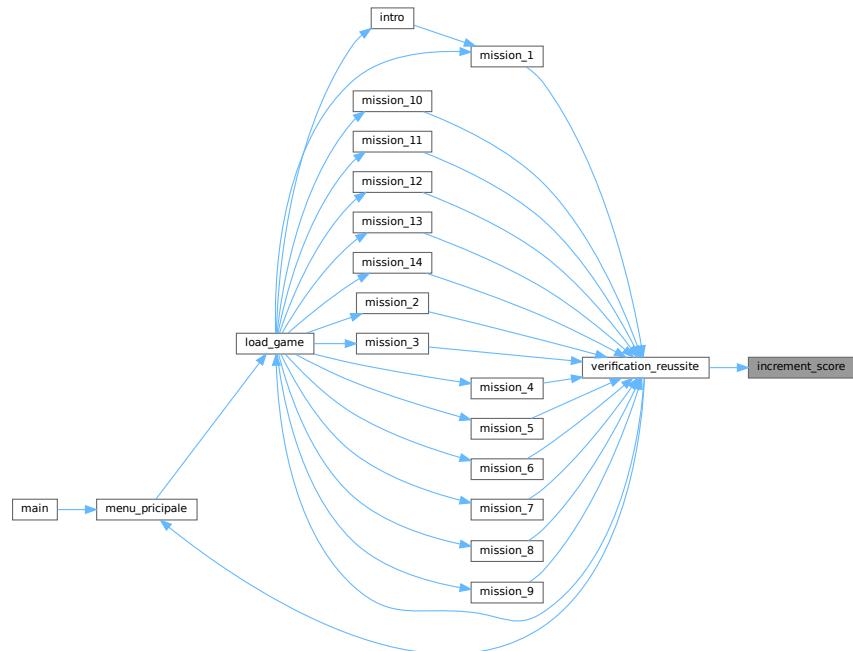
<i>actuPTS</i>	le nombre de point à ajouter au score actuel
----------------	----------------------------------------------

Definition at line 1165 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.27.2.13 inserer()

```
void inserer (
    char caractere,
    int position,
    char * chaine )
```

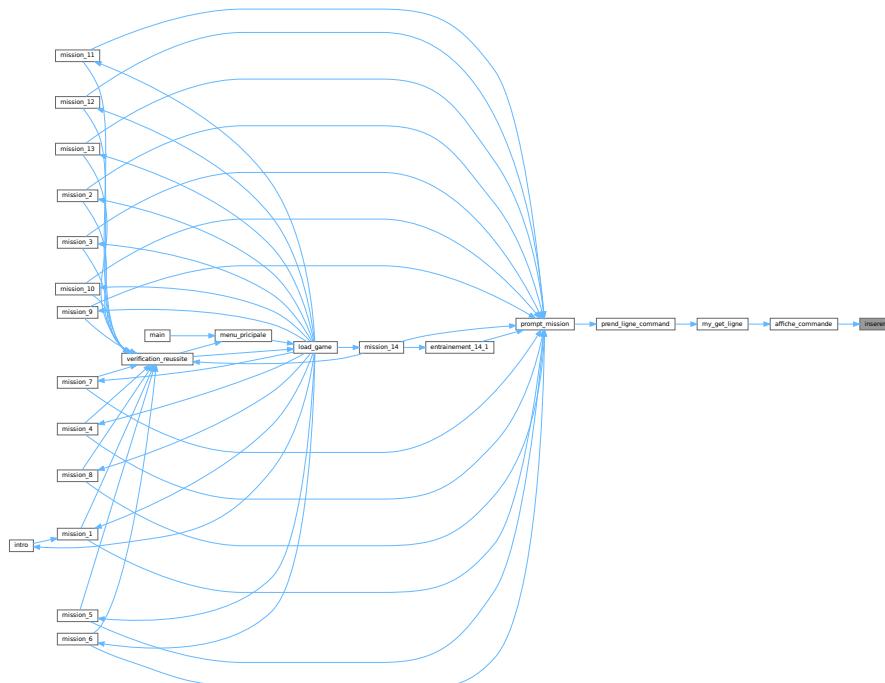
Elle permet d'inserer une caractere dans une chaine à une position preciser en argument.

##### Parameters

<i>caractere</i>	le caractere à inserer dans une chaine
<i>position</i>	la position de l'insertion
<i>chaine</i>	la chaine où l'on va inserer caractere

Definition at line 1006 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.27.2.14 inserer\_mot\_dans()

```
void inserer_mot_dans (
    int position,
    char * chaine,
    char * insertion )
```

c'est pour inserer une mot dans une position precise dans une chaine ; separateur = espace

### Parameters

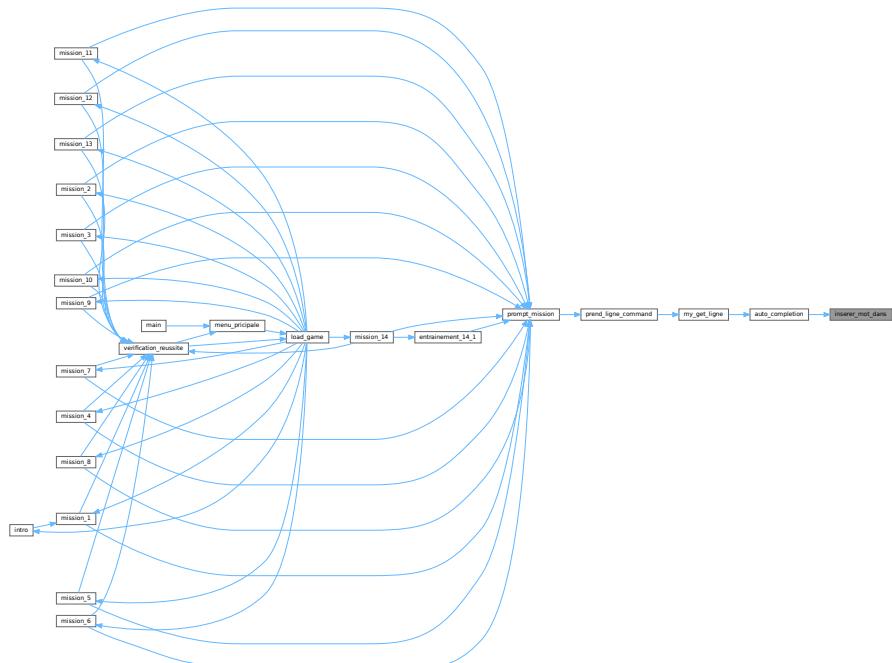
<i>position</i>	la position de l'insertion dans la chaîne
<i>chaine</i>	la chaîne d'insertion
<i>insertion</i>	la chaîne à insérer

Definition at line 818 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



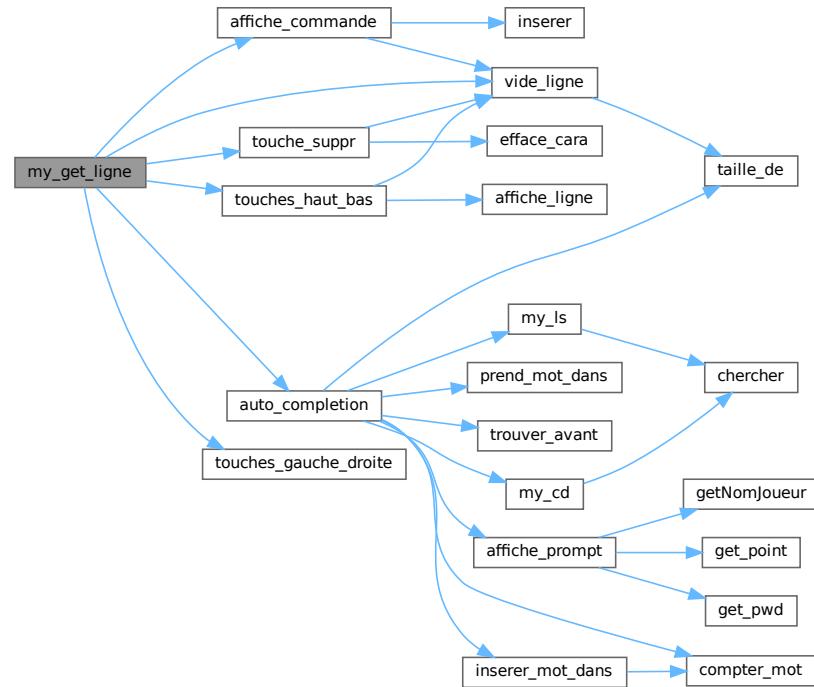
### 4.27.2.15 my\_get\_ligne()

```

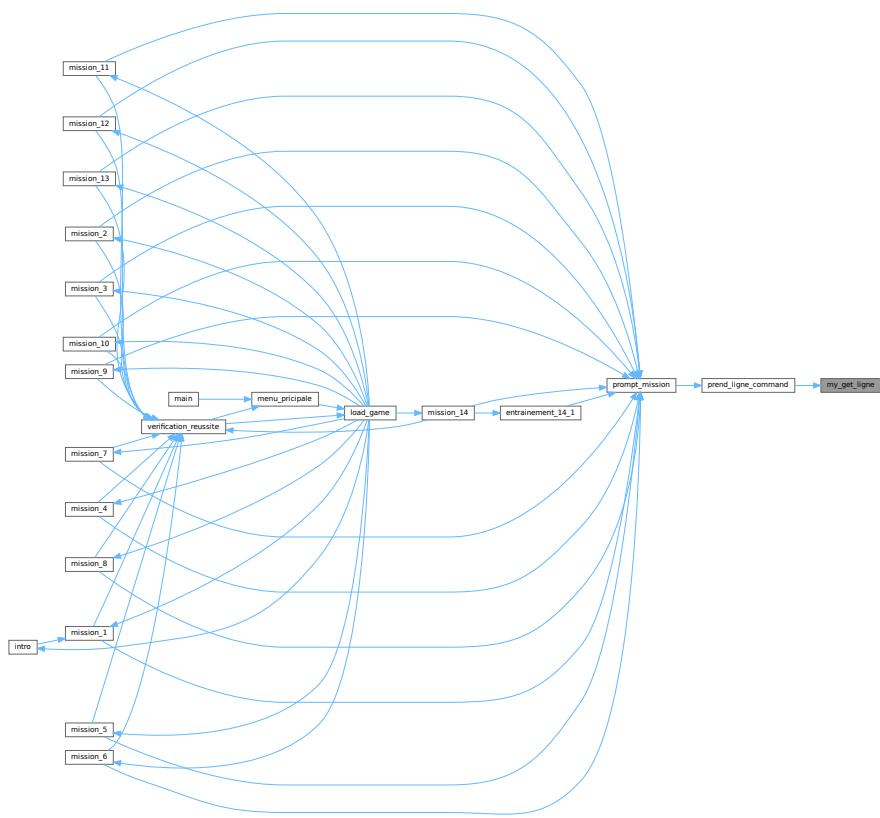
int my_get_ligne (
    char * mobile,
    int taille_Max,
    fichier * repertoire_actuel )
  
```

Definition at line 78 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

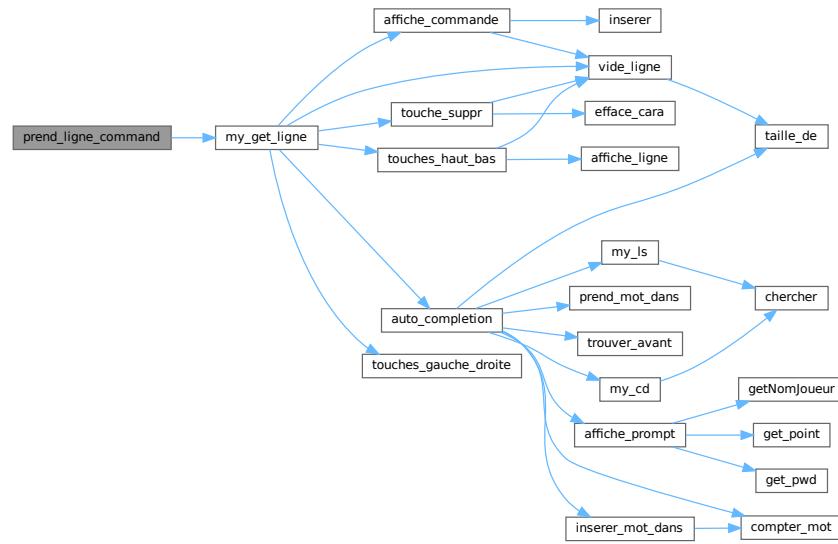


#### 4.27.2.16 `prend_ligne_command()`

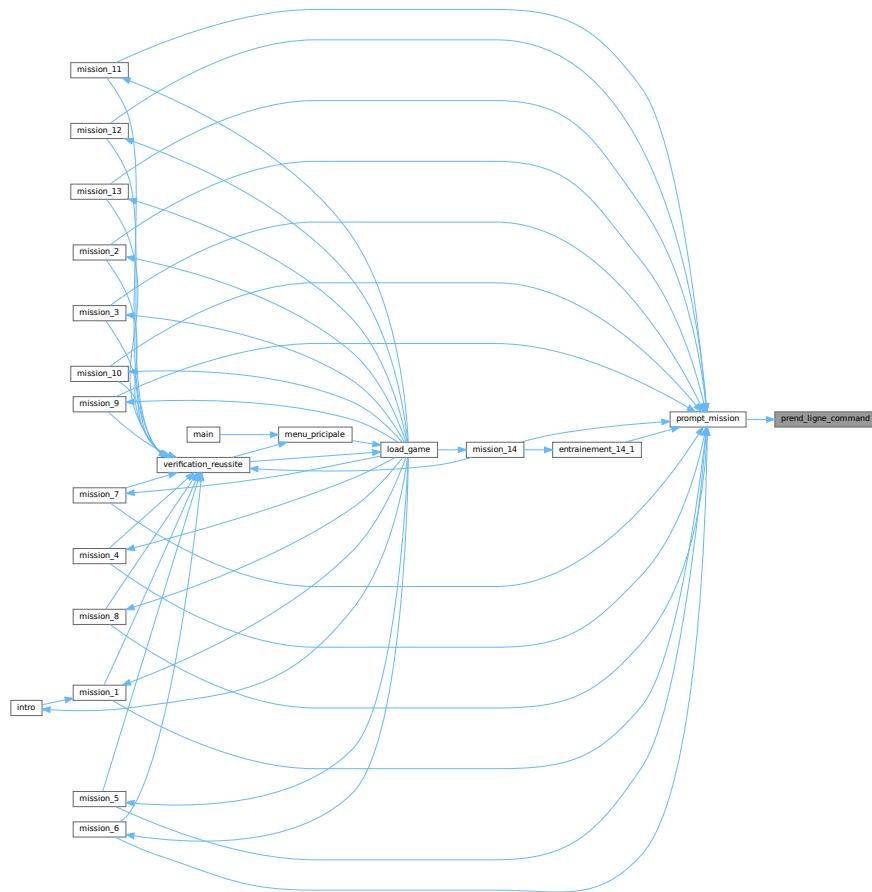
```
int prend_ligne_command (
    char ** emplacement,
    int taille,
    fichier * repertoire_actuel )
```

Definition at line 1027 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.27.2.17 prend\_mot\_dans()

```
int prend_mot_dans (
    int position,
    char * chaine,
    char * emplacement )
```

#### Parameters

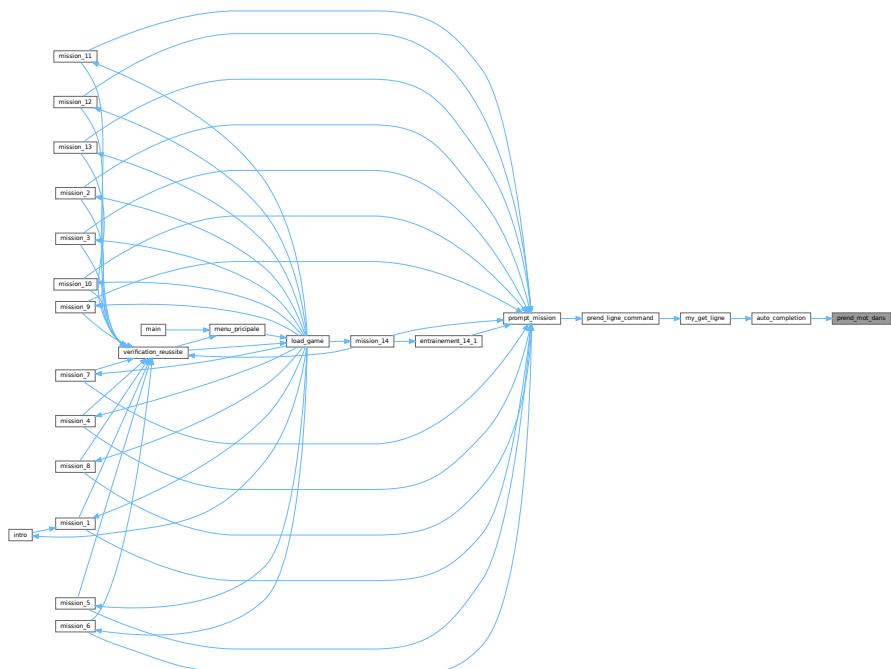
<i>position</i>	la position du mot à prendre
<i>chaine</i>	la chaine de caractere où extraire le mot
<i>emplacement</i>	l'adresse de l'emplacement de la chaine extractée

#### Returns

renvoie 0 si tous s'est bien passé

Definition at line 940 of file [prompt.c](#).

Here is the caller graph for this function:



### 4.27.2.18 rapport\_mission()

```
int rapport_mission (
    int num_mission,
    bool reussie )
```

celui ci écrit le joueur , temps de jeu , réussie ou pas et numéro de mission dans le fichier d'enregistrement

**Parameters**

<i>num_mission</i>	c'est la numero de la mission dont on va faire le rapport
<i>reussie</i>	possede une valeur true si la mission est reussie

**Returns**

1 si l'ecriture s'est bien passe ou -1 si non

Definition at line 1989 of file [prompt.c](#).

**4.27.2.19 taille\_de()**

```
int taille_de (
    char * ligne )
```

elle ne compte pas les caracteres comme strlen ; elle donne la taille en affichage

**Parameters**

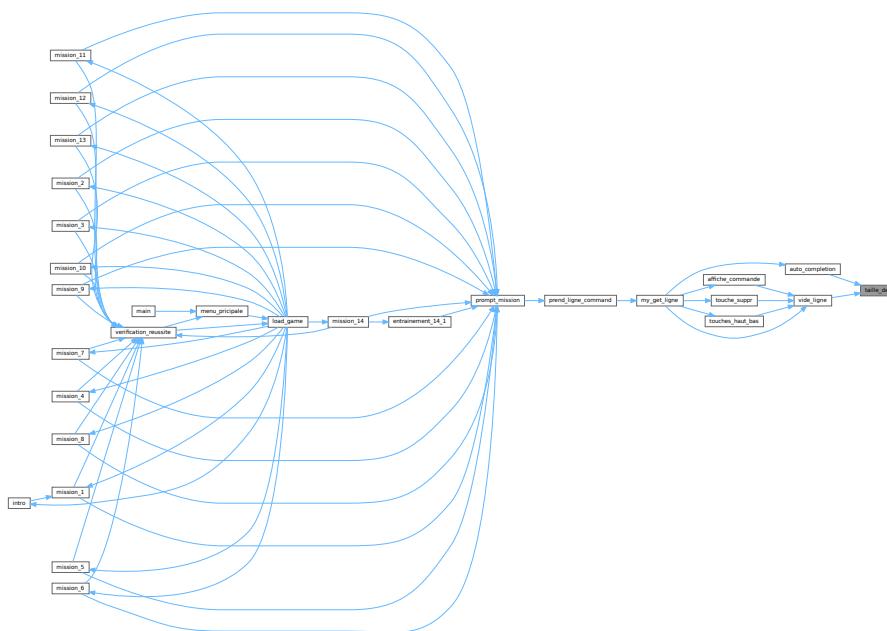
<i>ligne</i>	la chaine de caractere à compter
--------------	----------------------------------

**Returns**

le nombre de caractere de la chaine entrer en argument

Definition at line 384 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.27.2.20 touche\_suppr()

```
void touche_suppr (
    int * position,
    int * pos_droite,
    char * chaine )
```

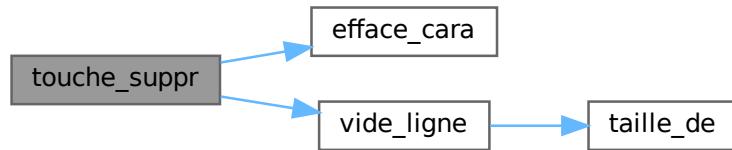
c'est la fonction associer à la touche suppr

##### Parameters

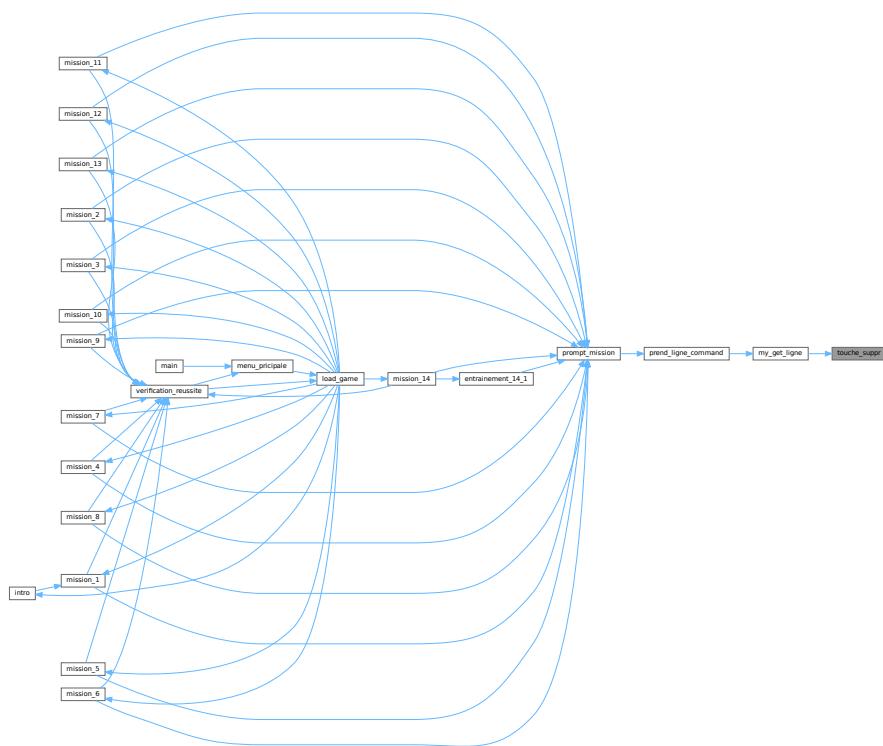
<i>position</i>	la position dans le chaine de caractere
<i>pos_droite</i>	la position de curseur en comptant depuis la droite
<i>chaine</i>	le ligne où l'on veut supprimer une caractere

Definition at line 348 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.27.2.21 touches\_gauche\_droite()

```
void touches_gauche_droite (
    int * indice,
    int * pos_droite,
    char * chaine,
    int gauche )
```

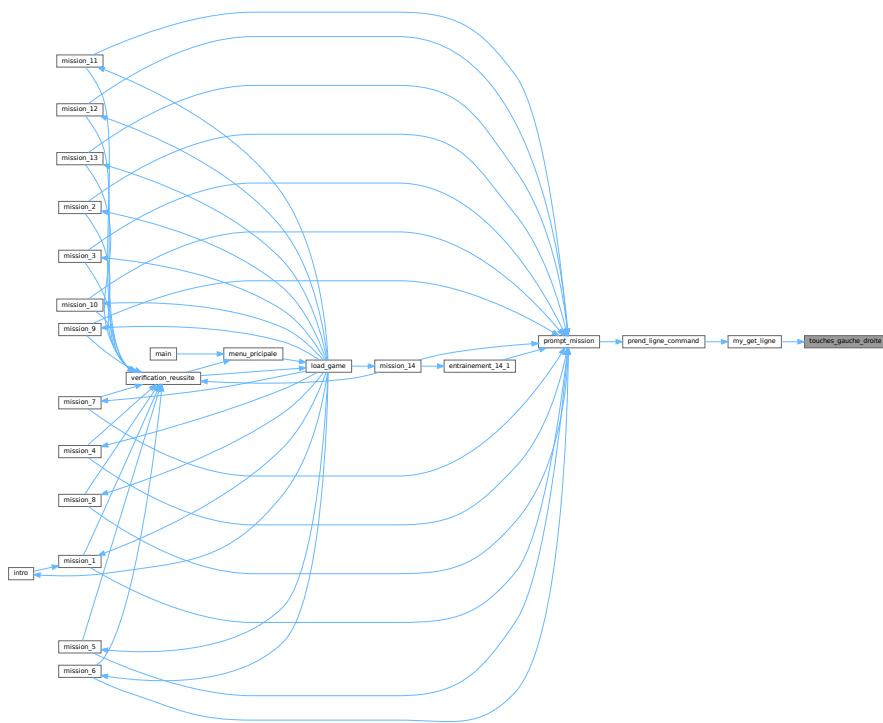
c'est la fonction associer à la touche gauche et droite

##### Parameters

<i>indice</i>	la position dans la ligne actuelle
<i>pos_droite</i>	la position par rapport à la position la plus à droite
<i>chaine</i>	la ligne actuelle
<i>gauche</i>	valeur si 0 si touche droite et 1 si touche gauche

Definition at line 231 of file [prompt.c](#).

Here is the caller graph for this function:



#### 4.27.2.22 touches\_haut\_bas()

```
void touches_haut_bas (
    int * indice,
    int * position,
    int * droite,
    char * chaine,
    char * tmp,
    int haut )
```

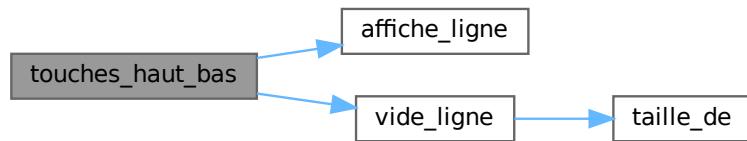
la fonctionne associer à la touche haut et bas

##### Parameters

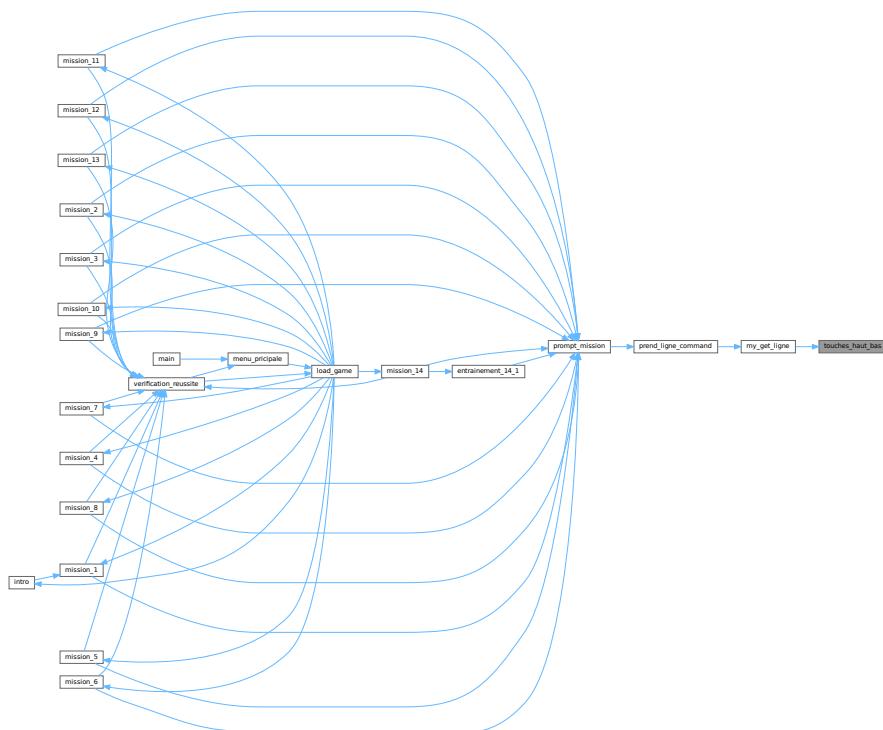
<i>indice</i>	position dans la chaine
<i>position</i>	position actuelle dans les visions des historiques des commandes
<i>droite</i>	position par rapport à la plus droite
<i>chaine</i>	la chaine de caractere contenant la ligne
<i>tmp</i>	la chaine ou l'on stocke une ligne temporairement ; ici la ligne la plus bas est enregistrer
<i>haut</i>	1 si touche haut et 0 si touche bas

Definition at line 276 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.27.2.23 trouver\_avant()

```
int trouver_avant (
    char * chaine,
    char * recherche )
```

Elle permet de trouver si la chaîne recherche est le début de chaîne ; les espaces sont ignorés.

##### Parameters

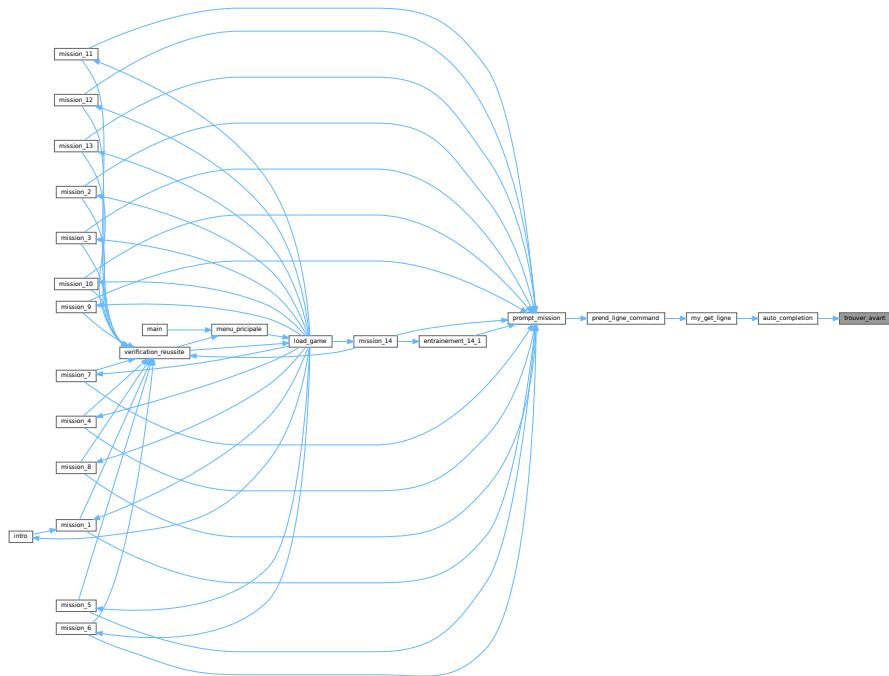
<i>chaine</i>	la chaîne de recherche
<i>recherche</i>	la chaîne à rechercher

**Returns**

Elle renvoie la taille de chaîne semblable

Definition at line 889 of file [prompt.c](#).

Here is the caller graph for this function:

**4.27.2.24 vide\_ligne()**

```
void vide_ligne (
    int droite,
    char * chaine )
```

c'est une simple fonction qui permet de vider la ligne de commande actuelle

**Parameters**

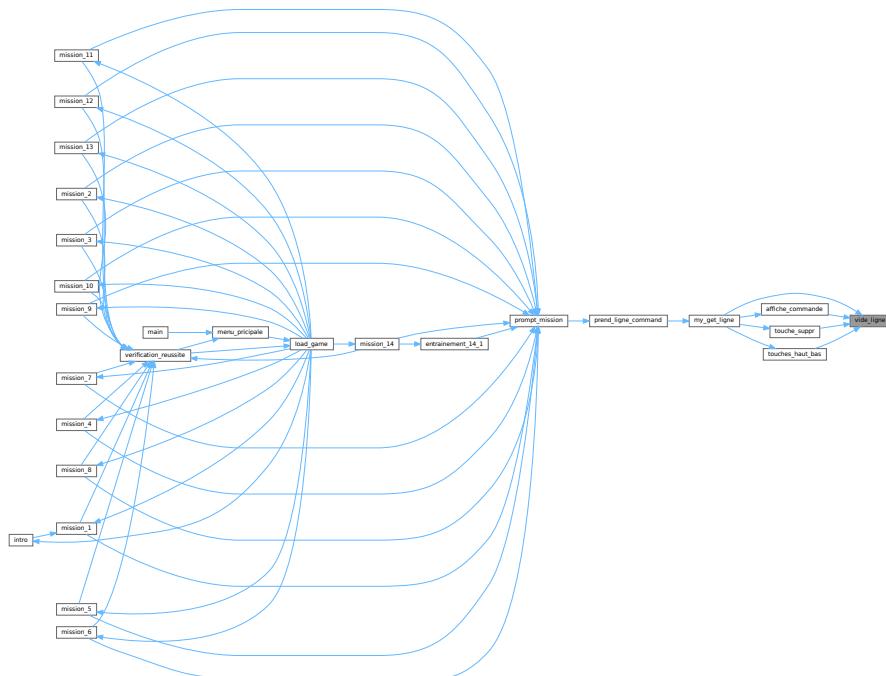
<i>droite</i>	la position par rapport à la plus droite
<i>chaine</i>	la chaîne de caractère de la ligne à vider

Definition at line 368 of file [prompt.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.28 prompt.c

[Go to the documentation of this file.](#)

```
00001
00011 #include <stdio.h>
00012 #include <string.h>
00013 #include <stdlib.h>
00014 #include <unistd.h>
00015 #include <termios.h>
00016 #include <ctype.h>
00021 #include <n curses.h>
00026 #include "../include/gestion_fichier.h"
00027
00032 #include "../include/prompt.h"
00033 #include "../include/affichage.h"
00034
00035 void affiche_prompt(fichier *repertoire_actuel)
00036 {
00037     int point_actu;
00038     char *nom_joueur, *chemin_repertoire_actuel;
00039
00040     nom_joueur = malloc(MOT * sizeof(char));
```

```
00041     chemin_reperatoire_actuel = calloc(MOT , sizeof(char));
00042
00043     point_actu = get_point();
00044     get_pwd( repertoire_actuel , chemin_reperatoire_actuel );
00045
00046     getNomJoueur(nom_joueur);
00047     printf("%d\033[1;33m%033[0m\033[1;34m@\033[0m\033[1;31mBash-Commando:\033[0m :%s $ ",
00048         point_actu , nom_joueur, chemin_reperatoire_actuel);
00049     free(nom_joueur);
00050     free(chemin_reperatoire_actuel);
00051 }
00052 void efface_cara(char *chaine, int *position)
00053 {
00054     int point_actu, longueur, i;
00055
00056     point_actu = *position;
00057
00058     if ((*position) > 0)
00059     {
00060         // Reculer pour trouver le début du caractère UTF-8
00061         (*position)--;
00062         while (((*position) > 0) && ((chaine[(*position)] & 0xC0) == 0x80))
00063     {
00064         (*position)--;
00065     }
00066
00067     longueur = point_actu - (*position);
00068
00069     // Décaler les caractères dans la chaîne
00070     point_actu = strlen(chaine) - longueur;
00071     for (i = (*position); i < point_actu; i++)
00072     {
00073         chaine[i] = chaine[i + longueur];
00074     }
00075     chaine[point_actu] = '\0';
00076 }
00077 }
00078 int my_get_ligne(char *mobile, int taille_Max, fichier *repertoire_actuel)
00079 {
00080     struct termios ancien, nouveau;
00081     char *chaine, *tmp ;
00082     char caractere;
00083     int i, position, j, test, droite, tabulation;
00084
00085     // Initialisation
00086     mobile[0] = '\0';
00087
00088     // Configuration du terminal en mode non canonique
00089     tcgetattr(STDIN_FILENO, &ancien);
00090     nouveau = ancien;
00091     nouveau.c_lflag &= ~(ICANON | ECHO);
00092     tcsetattr(STDIN_FILENO, TCSANOW, &nouveau);
00093
00094
00095     // Initialisation des variables
00096     i = 0;
00097     position = -1; // position dans l'historique (-1 = pas dans l'historique)
00098     tmp = calloc(LIGNE, sizeof(char));
00099     droite = 0; // décalage du curseur vers la droite
00100     tabulation = 0; // compteur pour les tabulations successives
00101
00102     while (true)
00103     {
00104         caractere = getchar();
00105
00106         // Touche Entrée
00107         if (caractere == '\n' || caractere == '\r')
00108     {
00109         // Réinitialiser le compteur de tabulation à la fin de la ligne
00110         tabulation = 0;
00111         break;
00112     }
00113
00114         // Vérifier la taille maximale
00115         if (i >= taille_Max - 1)
00116     {
00117         break;
00118     }
00119
00120         // Touche Backspace/Suppr
00121         if(caractere == 127 || caractere == 8)
00122     {
00123             touche_suppr(&i , &droite , mobile);
00124             continue;
00125     }
00126 }
```

```

00127      // Séquence escape (touches fléchées)
00128      if (caractere == 27)
00129      {
00130          // Lire le caractère suivant avec vérification
00131          caractere = getchar();
00132          if (caractere == '[')
00133          {
00134              caractere = getchar();
00135              switch (caractere)
00136              {
00137                  case 'A': // Flèche haut (historique précédent)
00138                  {
00139                      touches_haut_bas(&i , &position , &droite , mobile , tmp , 1);
00140                      // Réinitialiser le compteur de tabulation
00141                      tabulation = 0;
00142                      break;
00143                  }
00144                  case 'B': // Flèche bas (historique suivant)
00145                  {
00146                      touches_haut_bas(&i , &position , &droite , mobile , tmp , 0);
00147                      // Réinitialiser le compteur de tabulation
00148                      tabulation = 0;
00149                      break;
00150                  }
00151                  case 'C': // Flèche droite
00152                  {
00153                      touches_gauche_droite(&i , &droite , mobile , 0);
00154                      break;
00155                  }
00156                  case 'D': // Flèche gauche
00157                  {
00158                      touches_gauche_droite(&i , &droite , mobile , 1);
00159                      break;
00160                  }
00161                  default:
00162                  {
00163                      continue;
00164                  }
00165              }
00166          }
00167          else
00168          {
00169              // Si ce n'est pas une flèche, ignorer la séquence
00170              continue;
00171          }
00172      }
00173      else if (caractere == '\t') // Tabulation
00174      {
00175          vide_ligne(droite , mobile);
00176          auto_completion(mobile, repertoire_actuel, &tabulation , &droite);
00177          i = strlen(mobile);
00178          for(test=0 ; test<droite ; test++)
00179          {
00180              i--;
00181              while ( (i > 0) && ((mobile[i] & 0xC0) == 0x80) )
00182              {
00183                  i--;
00184              }
00185              printf("\033[1D");
00186          }
00187      }
00188      // Caractères imprimables
00189      else if (isprint(caractere))
00190      {
00191          // réinitialiser tabulation quand on tape autre chose qu'un espace
00192          if (caractere != ' ')
00193          {
00194              tabulation = 0;
00195          }
00196          // insertion de caractère et affichage du ligne
00197          affiche_commande(&i , caractere , mobile , droite);
00198      }
00199
00200      if (caractere == ' ')
00201      {
00202          tabulation = 0;
00203      }
00204  }
00205  printf("\n");
00206
00207  // Restaurer les paramètres du terminal
00208  tcsetattr(STDIN_FILENO, TCSANOW, &ancien);
00209
00210  free(tmp);
00211  return i;
00212 }
00213 void affiche_commande(int *indice , char caractere ,char *chaine ,int droite)

```

```

00214 {
00215     int i;
00216
00217     vide_ligne(droite , chaine);
00218     insérer(caractere, (*indice), chaine);
00219     printf("%s" , chaine);
00220     (*indice) = strlen(chaine);
00221     for(i=0 ; i<droite ; i++)
00222     {
00223         (*indice)--;
00224         while ( ((*indice) > 0) && ((chaine[(*indice)] & 0xC0) == 0x80) )
00225         {
00226             (*indice)--;
00227         }
00228         printf("\033[1D");
00229     }
00230 }
00231 void touches_gauche_droite(int *indice , int *pos_droite , char *chaine , int gauche)
00232 {
00233     switch(gauche)
00234     {
00235         case 0: // Flèche droite
00236         {
00237             if ((*indice) < strlen(chaine))
00238             {
00239                 if ((*pos_droite) > 0)
00240                 {
00241                     (*pos_droite)--;
00242                     printf("\033[1C");
00243                     (*indice)++;
00244                     while ( ( (*indice) < strlen(chaine) ) && ( (chaine[(*indice)] & 0xC0) == 0x80) )
00245                     {
00246                         (*indice)++;
00247                     }
00248                 }
00249             }
00250             else
00251             {
00252                 printf("\a"); // bip sonore
00253             }
00254             break;
00255         }
00256         case 1: // Flèche gauche
00257         {
00258             if ((*indice) > 0)
00259             {
00260                 (*pos_droite)++;
00261                 printf("\033[1D");
00262                 (*indice]--;
00263                 while ( ((*indice) > 0) && ((chaine[(*indice)] & 0xC0) == 0x80) )
00264                 {
00265                     (*indice)--;
00266                 }
00267             }
00268             else
00269             {
00270                 printf("\a");
00271             }
00272             break;
00273         }
00274     }
00275 }
00276 void touches_haut_bas(int *indice , int *position , int *droite , char *chaine , char *tmp , int haut)
00277 {
00278     int test;
00279     char *chemin , *nom;
00280
00281     // Préparation du chemin pour l'historique
00282     test = 23;
00283     nom = getenv("USER");
00284     if (nom != NULL)
00285     {
00286         test += strlen(nom);
00287         chemin = calloc(test , sizeof(char));
00288         sprintf(chemin, "../save/%s/histoire.txt", nom);
00289     }
00290
00291     switch(haut)
00292     {
00293         case 1 :
00294         {
00295             if ((*position) == -1)
00296             {
00297                 strcpy(tmp, chaine);
00298             }
00299             vide_ligne((*droite) , chaine);
00300             chaine[0] = '\0';

```

```

00301
00302     (*position)++;
00303     test = affiche_ligne(chemin, (-(*position)), chaine);
00304
00305     if (test != -1)
00306     {
00307         (*indice) = strlen(chaine);
00308         // Remettre à zéro le décalage du curseur quand on change de ligne
00309         (*droite) = 0;
00310         if (test == 2) // limite atteinte
00311         {
00312             (*position)--;
00313         }
00314     }
00315     break ;
00316 }
00317 case 0 :
00318 {
00319     if ((*position) == 0)
00320     {
00321         vide_ligne((*droite) , chaine);
00322         (*position) = -1;
00323         strcpy(chaine, tmp);
00324         printf("%s", chaine);
00325         (*indice) = strlen(chaine);
00326         (*droite) = 0;
00327     }
00328     else if ((*position) >= 1)
00329     {
00330         vide_ligne((*droite) , chaine);
00331         chaine[0] = '\0';
00332         (*position)--;
00333         test = affiche_ligne(chemin, (-(*position)), chaine);
00334         if (test != -1)
00335         {
00336             (*indice) = strlen(chaine);
00337             (*droite) = 0;
00338         }
00339     }
00340     break;
00341 }
00342 }
00343 if(nom != NULL)
00344 {
00345     free(chemin);
00346 }
00347 }
00348 void touche_suppr(int *position , int *pos_droite , char *chaine)
00349 {
00350     int i;
00351     if ((*position) > 0)
00352     {
00353         vide_ligne((*pos_droite), chaine);
00354         efface_cara(chaine , position);
00355         printf("%s" , chaine);
00356         (*position) = strlen(chaine);
00357         for(i=0 ; i<(*pos_droite) ; i++)
00358         {
00359             (*position)--;
00360             while ( ((*position) > 0) && ((chaine[(*position)] & 0xC0) == 0x80) )
00361             {
00362                 (*position)--;
00363             }
00364             printf("\033[1D");
00365         }
00366     }
00367 }
00368 void vide_ligne(int droite , char *chaine)
00369 {
00370     int position;
00371
00372     for(position=0 ; position<droite ; position++)
00373     {
00374         printf("\033[1C");
00375     }
00376     position = taille_de(chaine);
00377     while(position != 0)
00378     {
00379         printf("\b \b");
00380         position--;
00381         fflush(stdout);
00382     }
00383 }
00384 int taille_de(char *ligne)
00385 {
00386     int i , longueur;
00387

```

```

00388     longueur = 0;
00389     i = 0;
00390     if(ligne!= NULL)
00391     {
00392         while(ligne[i] != '\0')
00393         {
00394             if((ligne[i] & 0xC0) == 0xC0) // Lohana Accent sy ny tariny , utf-8 reny
00395             {
00396                 longueur++;
00397             }
00398             else if((ligne[i] & 0x80) == 0) // rehefa ASCII
00399             {
00400                 longueur++;
00401             }
00402             i++;
00403         }
00404     }
00405
00406     return (longueur);
00407 }
00408 int auto_completion(char *chaine, fichier *repertoire_actuel, int *nb_Tab , int *pos_droite)
00409 {
00410     int i, longueur , j , test , compteur_fichiers , nbMot , nb_commande , pos_cursur;
00411     fichier *fichier_pass , *racine , *tmp;
00412     char *brouillon , *tempo , *chemin , *reste;
00413
00414     // Liste des commandes disponibles
00415     char *liste_commandes[] = {
00416         "ls", "cd", "pwd", "cat", "echo", "mkdir", "rm", "cp", "mv",
00417         "rmdir", "tail", "chmod", "exit", "clear", "aide", "histoire",
00418         "liste_commande", "major", "mission", "head" , "touch", "favori",
00419         "demos" , "nombre_ouverture" , NULL
00420     };
00421     nb_commande = 23;
00422
00423     fichier_pass = repertoire_actuel;
00424     racine = repertoire_actuel;
00425     // Recherche de fichier *racine
00426     while( racine->parent != NULL)
00427     {
00428         racine = racine->parent;
00429     }
00430     i = strlen(chaine);
00431     for(j=0 ; j<(*pos_droite) ; j++)
00432     {
00433         i--;
00434         while ( (i > 0) && ((chaine[i] & 0xC0) == 0x80) )
00435         {
00436             i--;
00437         }
00438     }
00439     reste = calloc(LIGNE , sizeof(char));
00440     test = i;
00441     j = 0;
00442     do
00443     {
00444         reste[j] = chaine[i];
00445         i++;
00446         j++;
00447         if(i > strlen(chaine))
00448         {
00449             break;
00450         }
00451     }
00452     while(chaine[i] != '\0');
00453     chaine[test] = '\0';
00454     longueur = strlen(chaine);
00455     nbMot = compter_mot(chaine);
00456     i = 0;
00457     j = 0;
00458     if (longueur == 0 || nbMot == 0)
00459     {
00460         strcat(chaine, "aide");
00461     }
00462     else
00463     {
00464         // compter le nombre de mot
00465         pos_cursur = nbMot;
00466         brouillon = calloc(LIGNE , sizeof(char));
00467         strcpy(brouillon , "aide");
00468         // trouver le nombre de caractere de brouillon au debut de chaine
00469         i = trouver_avant(chaine , brouillon);
00470
00471         // appuyer tab avec aide aide suivie d'espacement
00472         if( ( nbMot == 1 ) && ( i == 4 ) )
00473         {
00474             strcat(chaine , " aide");

```

```

00475      }
00476      else if( nbMot == 2 && (i == 4) )
00477      {
00478          longueur = strlen(chaine);
00479          if(longueur > 0)
00480          {
00481              if(chaine[longueur-1] != ' ')
00482              {
00483                  prendre_mot_dans(2 , chaine , brouillon);
00484                  compteur_fichiers = 0;
00485                  tempo = calloc(MOT , sizeof(char));
00486                  for(i=0 ; i<nb_commande ; i++)
00487                  {
00488                      j = trouver_avant(liste_commandes[i] , brouillon);
00489                      if(j == strlen(brouillon))
00490                      {
00491                          compteur_fichiers++;
00492                          inserer_mot_dans(1 , tempo , liste_commandes[i]);
00493                      }
00494                  }
00495                  if(compteur_fichiers == 1)
00496                  {
00497                      inserer_mot_dans(2 , chaine , tempo);
00498                  }
00499                  else if( ((*nb_Tab) != 0 ) && (compteur_fichiers>0) )
00500                  {
00501                      printf("%s\n" , chaine);
00502                      for(i=0 ; i<nb_commande ; i++)
00503                      {
00504                          j = trouver_avant(liste_commandes[i] , brouillon);
00505                          if(j == strlen(brouillon))
00506                          {
00507                              compteur_fichiers++;
00508                              printf("%s\t" , liste_commandes[i]);
00509                          }
00510                      }
00511                      printf("\n");
00512                      affiche_prompt(repertoire_actuel);
00513                  }
00514                  }
00515                  free(tempo);
00516              }
00517              else if((*nb_Tab) != 0)
00518              {
00519                  (*nb_Tab) = (*nb_Tab) % nb_commande; // Boucler si trop de tabulations
00520                  inserer_mot_dans(2 , chaine , liste_commandes[(*nb_Tab)]);
00521              }
00522          }
00523      }
00524      else if(nbMot == 1)
00525      {
00526          longueur = strlen(chaine);
00527          if(longueur > 0)
00528          {
00529              if(chaine[longueur-1] != ' ')
00530              {
00531                  prendre_mot_dans(1 , chaine , brouillon);
00532                  compteur_fichiers = 0;
00533                  tempo = calloc(MOT , sizeof(char));
00534                  for(i=0 ; i<nb_commande ; i++)
00535                  {
00536                      j = trouver_avant(liste_commandes[i] , brouillon);
00537                      if(j == strlen(brouillon))
00538                      {
00539                          compteur_fichiers++;
00540                          inserer_mot_dans(1 , tempo , liste_commandes[i]);
00541                      }
00542                  }
00543                  if(compteur_fichiers == 1)
00544                  {
00545                      inserer_mot_dans(1 , chaine , tempo);
00546                  }
00547                  else if( ((*nb_Tab) != 0 ) && (compteur_fichiers>0) )
00548                  {
00549                      printf("%s\n" , chaine);
00550                      for(i=0 ; i<nb_commande ; i++)
00551                      {
00552                          j = trouver_avant(liste_commandes[i] , brouillon);
00553                          if(j == strlen(brouillon))
00554                          {
00555                              compteur_fichiers++;
00556                              printf("%s\t" , liste_commandes[i]);
00557                          }
00558                      }
00559                      printf("\n");
00560                      affiche_prompt(repertoire_actuel);
00561                  }

```

```

00562         }
00563         free(tempo);
00564     }
00565     else
00566     {
00567         if((*nb_Tab) != 0)
00568         {
00569             printf("%s\n" , chaine);
00570             tempo = calloc(2 , sizeof(char));
00571             tempo[0] = '\0';
00572             my_ls(racine , fichier_pass , tempo );
00573             affiche_prompt(fichier_pass);
00574             free(tempo);
00575         }
00576     }
00577 }
00578 }
00579 else
00580 {
00581     longueur = strlen(chaine);
00582     if(chaine[longueur-1] != ' ')
00583     {
00584         prend_mot_dans(pos curseur , chaine , brouillon);
00585         if(chaine[longueur-1] != '/')
00586         {
00587             tempo = strchr(brouillon , '/');
00588             if(tempo == NULL)
00589             {
00590                 // Compter les fichiers disponibles
00591                 compteur_fichiers = 0;
00592                 fichier_pass = fichier_pass->premierfils;
00593                 while (fichier_pass != NULL)
00594                 {
00595                     compteur_fichiers++;
00596                     fichier_pass = fichier_pass->frereSuivant;
00597                 }
00598
00599                 if (compteur_fichiers > 0)
00600                 {
00601                     compteur_fichiers = 0;
00602                     fichier_pass = repertoire_actuel->premierfils;
00603                     while(fichier_pass != NULL)
00604                     {
00605                         j = trouver_avant(fichier_pass->nom , brouillon);
00606                         if(j == strlen(brouillon))
00607                         {
00608                             compteur_fichiers++;
00609                             tmp = fichier_pass;
00610                         }
00611                         fichier_pass = fichier_pass->frereSuivant;
00612                     }
00613                     if(compteur_fichiers == 1)
00614                     {
00615                         longueur = strlen(tmp->nom) + 1;
00616                         tempo = calloc( longueur , sizeof(char));
00617                         strcat(tempo , tmp->nom);
00618                         if(tmp->estDossier == 1)
00619                         {
00620                             strcat(tempo , "/");
00621                         }
00622                         inserer_mot_dans(pos curseur , chaine , tempo);
00623                         free(tempo);
00624                     }
00625                     else if( ((*nb_Tab) != 0 ) && (compteur_fichiers>0) )
00626                     {
00627                         printf("%s\n" , chaine);
00628                         fichier_pass = repertoire_actuel->premierfils;
00629                         while(fichier_pass != NULL)
00630                         {
00631                             j = trouver_avant(fichier_pass->nom , brouillon);
00632                             if(j == strlen(brouillon))
00633                             {
00634                                 if (fichier_pass->estDossier)
00635                                 {
00636                                     printf("\033[1;36m%s\033[0m " , fichier_pass->nom);
00637                                 }
00638                                 else
00639                                 {
00640                                     printf("\033[1;37m%s\033[0m " , fichier_pass->nom);
00641                                 }
00642                                 printf("\t");
00643                             }
00644                             fichier_pass = fichier_pass->frereSuivant;
00645                         }
00646                         printf("\n");
00647                         affiche_prompt(repertoire_actuel);
00648                     }

```

```

00649
00650
00651
00652
00653     i = strlen(brouillon);
00654     tempo = calloc(LIGNE , sizeof(char));
00655     if(i > 0)
00656     {
00657         while(brouillon[i] != '/')
00658         {
00659             i--;
00660         }
00661         longueur = i;
00662         j = i + 1;
00663         i = 0;
00664         while(brouillon[j] != '\0')
00665         {
00666             tempo[i] = brouillon[j];
00667             i++;
00668             j++;
00669         }
00670         tempo[i] = '\0';
00671         brouillon[(longueur+1)] = '\0';
00672
00673     }
00674     chemin = calloc(strlen(brouillon) , sizeof(char));
00675     inserer_mot_dans(1 , chemin , brouillon);
00676     tmp = my_cd(racine , repertoire_actuel , chemin);
00677     free(chemin);
00678     if(tmp == NULL)
00679     {
00680         affiche_prompt(repertoire_actuel);
00681     }
00682 else
00683 {
00684     // Compter les fichiers disponibles
00685     compteur_fichiers = 0;
00686     fichier_pass = tmp->premierfils;
00687     while (fichier_pass != NULL)
00688     {
00689         compteur_fichiers++;
00690         fichier_pass = fichier_pass->frereSuivant;
00691     }
00692
00693     if (compteur_fichiers > 0)
00694     {
00695         compteur_fichiers = 0;
00696         fichier_pass = tmp->premierfils;
00697         racine = tmp;
00698         while(fichier_pass != NULL)
00699         {
00700             j = trouver_avant(fichier_pass->nom , tempo);
00701             if(j == strlen(tempo))
00702             {
00703                 compteur_fichiers++;
00704                 tmp = fichier_pass;
00705             }
00706             fichier_pass = fichier_pass->frereSuivant;
00707         }
00708         if(compteur_fichiers == 1)
00709         {
00710             strcat(brouillon , tmp->nom);
00711
00712             if(tmp->estDossier == 1)
00713             {
00714                 strcat(brouillon , "/");
00715             }
00716             inserer_mot_dans(pos curseur , chaine , brouillon);
00717         }
00718         else if( ((*nb_Tab) != 0 ) && (compteur_fichiers>0) )
00719         {
00720             printf("%s\n" , chaine);
00721             fichier_pass = racine->premierfils;
00722             while(fichier_pass != NULL)
00723             {
00724                 j = trouver_avant(fichier_pass->nom , tempo);
00725                 if(j == strlen(tempo))
00726                 {
00727                     if (fichier_pass->estDossier)
00728                     {
00729                         printf("\033[1;36m%s\033[0m " , fichier_pass->nom);
00730                     }
00731                 else
00732                 {
00733                     printf("\033[1;37m%s\033[0m " , fichier_pass->nom);
00734                 }
00735             printf("\t");

```

```

00736                                     }
00737                                     fichier_pass = fichier_pass->frereSuivant;
00738                                 }
00739                                 printf("\n");
00740                                 affiche_prompt(repertoire_actuel);
00741                             }
00742                         }
00743                     }
00744                     free(tempo);
00745                 }
00746             }
00747         }
00748     }
00749     {
00750
00751         prend_mot_dans(pos curseur , chaine , brouillon);
00752         chemin = calloc(strlen(brouillon) , sizeof(char));
00753         inserer_mot_dans(l , chemin , brouillon);
00754         tmp = my_cd(racine , repertoire_actuel , chemin);
00755         free(chemin);
00756
00757         if(tmp == NULL)
00758         {
00759             affiche_prompt(repertoire_actuel);
00760         }
00761         else
00762         {
00763             // Compter les fichiers disponibles
00764             compteur_fichiers = 0;
00765             fichier_pass = tmp->premierfils;
00766             while (fichier_pass != NULL)
00767             {
00768                 compteur_fichiers++;
00769                 fichier_pass = fichier_pass->frereSuivant;
00770             }
00771             if (compteur_fichiers > 0)
00772             {
00773                 if(compteur_fichiers == 1)
00774                 {
00775                     strcat(brouillon , tmp->premierfils->nom);
00776                     inserer_mot_dans(pos curseur , chaine , brouillon);
00777                 }
00778                 else if((*nb_Tab) != 0 )
00779                 {
00780                     printf("%s\n" , chaine);
00781                     fichier_pass = tmp->premierfils;
00782                     while(fichier_pass != NULL)
00783                     {
00784                         if (fichier_pass->estDossier)
00785                         {
00786                             printf("\033[1;36m%s\033[0m " , fichier_pass->nom);
00787                         }
00788                         else
00789                         {
00790                             printf("\033[1;37m%s\033[0m " , fichier_pass->nom);
00791                         }
00792                         printf("\t");
00793
00794                         fichier_pass = fichier_pass->frereSuivant;
00795                     }
00796                     printf("\n");
00797                     affiche_prompt(repertoire_actuel);
00798                 }
00799             }
00800         }
00801     }
00802 }
00803 }
00804 }
00805 }
00806     free(brouillon);
00807 }
00808
00809     (*nb_Tab)++;
00810     strcat(chaine , reste);
00811     free(reste);
00812     longueur = taille_de(chaine);
00813     printf("%s" , chaine);
00814     i = strlen(chaine);
00815
00816     return longueur;
00817 }
00818 void inserer_mot_dans(int position , char *chaine , char *insertion)
00819 {
00820     int i , j , longueur , test;
00821     char *tmp;
00822

```

```

00823     test = compter_mot(chaine);
00824     if(test == 0)
00825     {
00826         position = 1;
00827     }
00828     else if(test < position)
00829     {
00830         position = test + 1;
00831     }
00832     j = 0;
00833     if(position == 1)
00834     {
00835         while(chaine[j] == ' ')
00836         {
00837             j++;
00838         }
00839     }
00840     // se déplacer au début du {postition} ième mot
00841     for(i=0 ; i<(position-1) ; i++)
00842     {
00843         while(chaine[j] == ' ')
00844         {
00845             j++;
00846         }
00847         while(chaine[j] != ' ')
00848         {
00849             j++;
00850         }
00851         while(chaine[j] == ' ')
00852         {
00853             j++;
00854         }
00855     }
00856     i = j;
00857     while((chaine[j] != ' ') && (chaine[j] != '\0'))
00858     {
00859         j++;
00860     }
00861     test = strlen(chaine) - j + 1;
00862     tmp = calloc(test , sizeof(char));
00863
00864     test = strlen(chaine);
00865     longueur = i;
00866     i = 0;
00867     while(j != test)
00868     {
00869         tmp[i] = chaine[j];
00870         i++;
00871         j++;
00872     }
00873     tmp[i] = '\0';
00874     chaine[longueur] = '\0';
00875     if(longueur > 0)
00876     {
00877         if(chaine[longueur-1] != ' ')
00878         {
00879             strcat(chaine , " ");
00880         }
00881     }
00882
00883     strcat(chaine , insertion);
00884     strcat(chaine , tmp);
00885
00886     free(tmp);
00887 }
00888 int trouver_avant(char *chaine , char *recherche)
00889 {
00890     int i , j;
00891
00892     i = 0;
00893     j = 0;
00894     while(chaine[j] == ' ')
00895     {
00896         j++;
00897     }
00898
00899     while( ( recherche[i] == chaine[j] ) && ( i < strlen(recherche) ) && ( j < strlen(chaine) ) )
00900     {
00901         i++;
00902         j++;
00903     }
00904
00905     return (i);
00906 }
00907
00908 int compter_mot(char *chaine)
00909 {

```

```
00910     int i , nbMot;
00911
00912     i = 0;
00913     nbMot = 0;
00914     while(chaine[i] == ' ')
00915     {
00916         i++;
00917     }
00918     while( chaine[i] != '\0' )
00919     {
00920         if(chaine[i] == ' ')
00921         {
00922             nbMot++;
00923             while(chaine[i] == ' ')
00924             {
00925                 i++;
00926             }
00927         }
00928         else
00929         {
00930             i++;
00931         }
00932     }
00933     if(chaine[i-1] != ' ')
00934     {
00935         nbMot++;
00936     }
00937
00938     return (nbMot);
00939 }
00940 int prend_mot_dans(int position , char *chaine , char *emplacement)
00941 {
00942     int i , j;
00943
00944     j = 0;
00945     if(position == 1)
00946     {
00947         while(chaine[j] == ' ')
00948         {
00949             j++;
00950         }
00951     }
00952 // se déplacer au début du {postition} ième mot
00953 for(i=0 ; i<(position-1) ; i++)
00954 {
00955     while(chaine[j] == ' ')
00956     {
00957         j++;
00958     }
00959     while(chaine[j] != ' ')
00960     {
00961         j++;
00962     }
00963     while(chaine[j] == ' ')
00964     {
00965         j++;
00966     }
00967 }
00968
00969 //vider brouillon
00970 i = 0;
00971 while(strlen(emplacement) != 0)
00972 {
00973     i = strlen(emplacement) - 1 ;
00974     emplacement[i] = '\0';
00975 }
00976 while((chaine[j] != '\0') && (chaine[j] != ' '))
00977 {
00978     emplacement[i] = chaine[j];
00979     j++;
00980     i++;
00981 }
00982 // effacer les espaces après le mot
00983 if(i > 1)
00984 {
00985     if(emplacement[i-2] == ' ')
00986     {
00987         i = i - 2;
00988         emplacement[i] = '\0';
00989         if(i != 0)
00990         {
00991             i--;
00992             while(emplacement[i] == ' ')
00993             {
00994                 emplacement[i] = '\0';
00995                 if(i != 0)
00996                 {
```

```

00997             i--;
00998         }
00999     }
01000 }
01001 }
01002 }
01003 }
01004 return (0);
01005 }
01006 void inserer(char caractere , int position , char *chaine)
01007 {
01008     int longueur , j;
01009     longueur = strlen(chaine);
01010     if(longueur == 0)
01011     {
01012         chaine[0] = caractere;
01013         chaine[1] = '\0';
01014     }
01015     else if(position <= longueur)
01016     {
01017         j = (longueur+1) ;
01018         while( (j != position) && (j != 0) )
01019         {
01020             chaine[j] = chaine[j-1];
01021             j--;
01022         }
01023         chaine[position] = caractere;
01024     }
01025 }
01026 }
01027 int prend_ligne_command(char **emplacement, int taille , fichier *repertoire_actuel)
01028 {
01029     int i , j , debut , fin , nbElement , pos , caractere ;
01030     char *ligne , *chemin , *nom;
01031     FILE *histoire;
01032
01033 // Preparer pour l' enregistrement de la ligne de commande
01034 ligne = calloc(taille, sizeof(char));
01035
01036 i = 23;
01037 nom = getenv("USER");
01038 i += strlen(nom);
01039 chemin = calloc(i , sizeof(char));
01040 sprintf(chemin , "../save/%s/histoire.txt" , nom);
01041 histoire = fopen(chemin , "a+");
01042
01043 free(chemin);
01044
01045 // recuperation de brute de la ligne de notre prompt
01046 my_get_ligne(ligne , taille , repertoire_actuel);
01047
01048
01049 nbElement = 0;
01050 // reparation de la ligne pour eviter les erreurs avec les espaces
01051 debut = 0;
01052 while(ligne[debut] == ' ' || ligne[debut] == '\t')
01053 {
01054     debut++;
01055 }
01056
01057 i = debut;
01058 j = 0;
01059 while(ligne[i] != '\0')
01060 {
01061     if(ligne[i] == ' ' || ligne[i] == '\t' )
01062     {
01063         emplacement[nbElement][j] = '\0';
01064         nbElement++;
01065         j = 0;
01066         while(ligne[i] == ' ' || ligne[i] == '\t')
01067         {
01068             i++;
01069         }
01070     }
01071     else
01072     {
01073         emplacement[nbElement][j] = ligne[i];
01074         i++;
01075         j++;
01076     }
01077 }
01078 emplacement[nbElement][j] = '\0';
01079
01080 free(ligne);
01081
01082 if(emplacement[nbElement][0] != '\0')
01083 {

```

```
01084     nbElement = nbElement + 1;
01085 }
01086 // Enregistrement de la ligne ecrite
01087 if(histoire != NULL)
01088 {
01089     fprintf(histoire , "\n" );
01090     for(i=0 ; i<nbElement ; i++)
01091     {
01092         fprintf(histoire , "%s " , emplacement[i]);
01093     }
01094     fclose(histoire);
01095 }
01096
01097 if(emplacement[i][j-1] == '\n' )
01098 {
01099     emplacement[nbElement][j-1] = '\0';
01100 }
01101
01102 return (nbElement);
01103 }
01104 int getNomJoueur(char *emplacement)
01105 {
01106     char *tmp;
01107
01108     tmp = calloc(MOT, sizeof(char));
01109     tmp = getenv("USER");
01110
01111     if(tmp == NULL)
01112     {
01113         return (-1);
01114     }
01115     strcpy(emplacement, tmp);
01116
01117 //free(tmp);
01118
01119 return (0);
01120 }
01121 int get_point()
01122 {
01123     int lec_point;
01124     FILE *ptr_Fic , *save ;
01125     char chemin_point[30] ;
01126     char *tmpFic;
01127
01128     save = fopen( ".../save/temp.txt" , "r");
01129
01130     if(save == NULL)
01131     {
01132         printf("impossible d'ouvrir save !\n");
01133     }
01134     else
01135     {
01136         tmpFic = calloc(100 , sizeof(char));
01137         fgets(tmpFic , 100 , save);
01138         fgets(tmpFic , 100 , save);
01139         free(tmpFic);
01140
01141         fscanf( save , "%s" , chemin_point );
01142         fclose(save);
01143     }
01144
01145     ptr_Fic = fopen( chemin_point , "r");
01146     lec_point = 0;
01147
01148     if( (ptr_Fic != NULL) && !(feof(ptr_Fic)) )
01149     {
01150         fscanf(ptr_Fic , "%d" , &lec_point);
01151     }
01152     else
01153     {
01154         lec_point = 0;
01155         ptr_Fic = fopen( chemin_point , "w+");
01156         fprintf(ptr_Fic , "0" );
01157     }
01158
01159     if(ptr_Fic != NULL)
01160     {
01161         fclose(ptr_Fic);
01162     }
01163     return (lec_point);
01164 }
01165 void increment_score(int actuPTS)
01166 {
01167     int ancien , ecrit;
01168     FILE *ptr_Fic , *save ;
01169     char *tmpFic;
01170     char ligne[10];
```

```

01171     char chemin_point[30] = {""};
01172
01173     save = fopen( ".../save/temp.txt" , "r");
01174
01175     if(save == NULL)
01176     {
01177         printf("impossible d'ouvrir save !\n");
01178     }
01179     else
01180     {
01181         tmpFic = calloc(100 , sizeof(char));
01182         fgets(tmpFic , 100 , save);
01183         fgets(tmpFic , 100 , save);
01184         free(tmpFic);
01185
01186         fscanf( save , "%s" , chemin_point );
01187         fclose(save);
01188     }
01189     ancien = get_point();
01190     ecris = actuPTS;
01191     ptr_Fic = fopen( chemin_point , "w+" );
01192
01193     if(ptr_Fic == NULL)
01194     {
01195         printf("Score NON sauvegardee !\n");
01196     }
01197     else if(ancien == 0)
01198     {
01199         fprintf(ptr_Fic , "%d" , ecris);
01200     }
01201     else if( !(feof(ptr_Fic)) )
01202     {
01203         ecris = ecris + ancien ;
01204         fprintf(ptr_Fic , "%d" , ecris);
01205     }
01206     else
01207     {
01208         fprintf(ptr_Fic , "%d" , ecris);
01209     }
01210
01211     fclose(ptr_Fic);
01212
01213 }
01214 int decrement_score(int penalite)
01215 {
01216     int ancien , ecris;
01217     FILE *ptr_Fic , *save ;
01218     char *tmpFic;
01219     char ligne[10];
01220
01221     char chemin_point[30] = {""};
01222
01223     save = fopen( ".../save/temp.txt" , "r");
01224
01225     if(save == NULL)
01226     {
01227         printf("Impossible d'ouvrir save !\n");
01228     }
01229     else
01230     {
01231         tmpFic = calloc(100 , sizeof(char));
01232         fgets(tmpFic , 100 , save);
01233         fgets(tmpFic , 100 , save);
01234         free(tmpFic);
01235
01236         fscanf( save , "%s" , chemin_point );
01237         fclose(save);
01238     }
01239
01240     ancien = get_point();
01241     ptr_Fic = fopen( chemin_point , "w+" );
01242
01243     if(ptr_Fic == NULL)
01244     {
01245         return (-1);
01246     }
01247     else if(ancien == 0 || (feof(ptr_Fic)))
01248     {
01249         return (-1);
01250     }
01251     else
01252     {
01253         ecris = ecris - 1;
01254         fprintf(ptr_Fic , "%d" , (ancien-1) );
01255     }
01256
01257     fclose(ptr_Fic);

```

```
01258     return (1);
01259 }
01260 void get_pwd(fichier *adresse, char *emplacement)
01261 {
01262     char *tmp;
01263     int i , j , test , taille , position;
01264     fichier *tmpFic;
01265
01266     tmp = calloc(LIGNE, sizeof(char));
01267     strcat(tmp, "//");
01268     tmpFic = adresse;
01269     do
01270     {
01271         strcat(tmp, tmpFic->nom);
01272         test = strcmp(tmpFic->nom , "/");
01273         strcat(tmp, "/");
01274         tmpFic = tmpFic->parent;
01275     }
01276     while(tmpFic != NULL);
01277
01278     strcat(emplacement, "/");
01279     taille = strlen(tmp);
01280     j = 1;
01281     position = taille - 4;
01282     while(position != 0)
01283     {
01284         i = position;
01285         while(tmp[i] != '/')
01286         {
01287             i--;
01288         }
01289         position = i - 1;
01290         i = i + 1;
01291         while(tmp[i] != '/')
01292         {
01293             emplacement[j] = tmp[i];
01294             i++;
01295             j++;
01296         }
01297         emplacement[j] = '/';
01298         j++;
01299     }
01300     emplacement[j] = '\0';
01301
01302     free(tmp);
01303 }
01304 }
01305 int commande_option(ligne_commande* ligne , char **liste , int nb_mot)
01306 {
01307     int i , taille , nbMot , nb_option , nb_argument;
01308
01309     taille = strlen(liste[0]) + 1;
01310     ligne->commande = calloc( taille , sizeof(char));
01311     strcpy(ligne->commande , liste[0]);
01312     nb_option = 0;
01313     nb_argument = 0;
01314     for(i=1 ; i<nb_mot ; i++)
01315     {
01316         if(liste[i][0] == '-')
01317         {
01318             nb_option++;
01319         }
01320         else
01321         {
01322             nb_argument++;
01323         }
01324     }
01325     ligne->option = calloc(nb_option+1 , sizeof(char*));
01326     ligne->argument = calloc(nb_argument+1 , sizeof(char*));
01327     nb_option = 0;
01328     nb_argument = 0;
01329     for(i=1 ; i<nb_mot ; i++)
01330     {
01331         if(liste[i][0] == '-')
01332         {
01333             ligne->option[nb_option] = calloc(strlen(liste[i])+1 , sizeof(char));
01334             strcpy(ligne->option[nb_option] , liste[i]);
01335             nb_option++;
01336         }
01337         else
01338         {
01339             ligne->argument[nb_argument] = calloc(strlen(liste[i])+1 , sizeof(char));
01340             strcpy(ligne->argument[nb_argument] , liste[i]);
01341             nb_argument++;
01342         }
01343     }
01344     ligne->option[nb_option] = NULL;
```

```

01345     ligne->argument[nb_argument] = NULL;
01346
01347     return(0);
01348 }
01349 fichier *execute(char **commande , fichier *repertoire_actuel , int nbMot )
01350 {
01351     int test , i , nb_commande;
01352     fichier *racine , *tmp;
01353     char *tempo , *chemin;
01354     ligne_commande ligne;
01355
01356     // Liste des commandes disponibles
01357     char *liste_commandes[] = {
01358         "ls", "cd", "pwd", "cat", "echo", "mkdir", "rm", "cp", "mv",
01359         "rmdir", "tail", "chmod", "exit", "clear", "aide", "histoire",
01360         "liste_commande", "major", "mission", "head" , "touch" , "favori", "nombre_ouverture", "demos"
01361     , NULL
01362     };
01363     nb_commande = 24;
01364
01365     commande_option(&ligne , commande , nbMot);
01366     racine = repertoire_actuel;
01367     // Recherche de fichier *racine
01368     while( racine->parent != NULL)
01369     {
01370         racine = racine->parent;
01371     }
01372
01373     test = strcmp(commande[0], SORTIR);
01374     if(test == 0)
01375     {
01376         return (NULL);
01377     }
01378
01379     for(i=0 ; ( liste_commandes[i] != NULL) && (i < nb_commande) ) ; i++)
01380     {
01381         test = strcmp(liste_commandes[i] , ligne.commande);
01382         if(test == 0)
01383         {
01384             break;
01385         }
01386     }
01387     nb_commande = i;
01388     switch(nb_commande)
01389     {
01390
01391         case 5:
01392         {
01393             nbMot = 0;
01394             while(ligne.argument[nbMot] != NULL)
01395             {
01396                 nbMot++;
01397             }
01398             if( nbMot == 0 )
01399             {
01400                 printf("(EE) : mkdir a besoin d'argument !\n");
01401             }
01402             else
01403             {
01404                 nbMot = 0;
01405                 while(ligne.argument[nbMot] != NULL)
01406                 {
01407                     my_mkdir(racine , repertoire_actuel , ligne.argument[nbMot] , 0);
01408                     nbMot++;
01409                 }
01410             }
01411             break;
01412         }
01413         case 7:
01414         {
01415             nbMot = 0;
01416             while(ligne.argument[nbMot] != NULL)
01417             {
01418                 if(nbMot == 0)
01419                 {
01420                     nbMot = 1;
01421                 }
01422                 nbMot++;
01423             }
01424             if(nbMot == 2)
01425             {
01426                 my_cp( racine , repertoire_actuel , ligne.argument[0] , ligne.argument[1]);
01427             }
01428             else if(nbMot > 2)
01429             {
01430                 nb_commande = 0;

```

```

01431         while(ligne.argument[(nb_commande)] != NULL)
01432         {
01433             nb_commande++;
01434         }
01435         for(i=0 ; i < (nb_commande-1) ; i++)
01436         {
01437             my_cp( racine , repertoire_actuel , ligne.argument[i] ,
01438                   ligne.argument[(nb_commande-1)]);
01439         }
01440     else
01441     {
01442         printf("\t(EE) : cp besoin d'arguments source et destination !\n\tSaisissez « aide cp
01443 » pour plus d'informations.\n");
01444         return (repertoire_actuel);
01445     }
01446     break;
01447 case 11:
01448 {
01449     nbMot = 0;
01450     while(ligne.argument[nbMot] != NULL)
01451     {
01452         nbMot++;
01453     }
01454     if(nbMot == 2)
01455     {
01456         my_chmod(racine, repertoire_actuel, ligne.argument[0], ligne.argument[1]);
01457     }
01458     else
01459     {
01460         printf("Apprenez a utilisez chmod !\n");
01461     }
01462     break;
01463 }
01464 case 22:
01465 {
01466     afficher_nbr_ouverture(racine, 0) ;
01467     break;
01468 }
01469 case 21:
01470 {
01471     affichage(racine) ;
01472     break;
01473 }
01474 case 0:
01475 {
01476     if(ligne.option[0] != NULL)
01477     {
01478         i = 0;
01479         test = 0;
01480         while(ligne.option[i] != NULL)
01481         {
01482             if(strcmp(ligne.option[i] , "-a") == 0)
01483             {
01484                 if(test == 3)
01485                 {
01486                     test = 2;
01487                 }
01488                 else if(test == 0)
01489                 {
01490                     test = 1;
01491                 }
01492             }
01493             else if(strcmp(ligne.option[i] , "-al") == 0 || strcmp(ligne.option[i] , "-la") ==
01494 0 )
01495             {
01496                 test = 2;
01497             }
01498             else if (strcmp(ligne.option[i] , "-l") == 0)
01499             {
01500                 if(test == 1)
01501                 {
01502                     test = 2;
01503                 }
01504                 else if(test == 0)
01505                 {
01506                     test = 3;
01507                 }
01508             }
01509             else
01510             {
01511                 printf("%s argument non encore connue !\n" , ligne.option[i]);
01512             }
01513             i++;
01514         }
01515     if(ligne.argument[0] != NULL)

```

```

01515         {
01516             i = 0;
01517             while(ligne.argument[i] != NULL)
01518             {
01519                 switch(test)
01520                 {
01521                     case 1:
01522                     {
01523                         my_ls_a(racine , repertoire_actuel , ligne.argument[i]);
01524                         break;
01525                     }
01526                     case 2:
01527                     {
01528                         my_ls_la(racine , repertoire_actuel ,ligne.argument[i]);
01529                         break;
01530                     }
01531                     case 3:
01532                     {
01533                         my_ls_l(racine , repertoire_actuel , ligne.argument[i]);
01534                         break;
01535                     }
01536                     default :
01537                     {
01538                         break;
01539                     }
01540                 }
01541                 i++;
01542             }
01543         }
01544     }
01545     {
01546         switch(test)
01547         {
01548             case 1:
01549             {
01550                 my_ls_a(racine , repertoire_actuel , ".");
01551                 break;
01552             }
01553             case 2:
01554             {
01555                 my_ls_la(racine , repertoire_actuel , ".");
01556                 break;
01557             }
01558             case 3:
01559             {
01560                 my_ls_l(racine , repertoire_actuel , ".");
01561                 break;
01562             }
01563             default :
01564             {
01565                 break;
01566             }
01567         }
01568     }
01569 }
01570 }
01571 else
01572 {
01573     if(nbMot == 1)
01574     {
01575         my_ls(racine , repertoire_actuel , commande[1] );
01576     }
01577 else
01578 {
01579     i = 0;
01580     if(ligne.argument[i] != NULL)
01581     {
01582         while(ligne.argument[i] != NULL)
01583         {
01584             my_ls(racine, repertoire_actuel , ligne.argument[i]);
01585             i++;
01586         }
01587     }
01588 }
01589 }
01590 break;
01591 }
01592 case 20:
01593 {
01594     if( ligne.argument[0] == NULL)
01595     {
01596         printf("(EE) : touch a besoin d'argument !\n");
01597     }
01598 else
01599 {
01600     i = 0;
01601     while(ligne.argument[i] != NULL)

```

```

01602         {
01603             my_touch( racine , repertoire_actuel , ligne.argument[i],0);
01604             i++;
01605         }
01606     }
01607     break;
01608 }
01609 case 16:
01610 {
01611     if( nbMot != 1)
01612     {
01613         printf("Oui liste_commande ignore ton argument !\n");
01614     }
01615     cat_fichier("../data/page_man/mes_commandes.txt" , false);
01616     break;
01617 }
01618 case 1:
01619 {
01620     i = 0;
01621     while(ligne.argument[i] != NULL)
01622     {
01623         i++;
01624     }
01625     if(ligne.option[0] != NULL)
01626     {
01627         printf("Option injuste ! \n");
01628     }
01629     else if(i > 1)
01630     {
01631         printf("Votre destination est indefini !\n");
01632     }
01633     else if(ligne.argument[0] != NULL)
01634     {
01635         tmp = my_cd(racine , repertoire_actuel , ligne.argument[0]);
01636         if(tmp != NULL)
01637         {
01638             repertoire_actuel = tmp;
01639         }
01640     }
01641     else
01642     {
01643         tmp = my_cd(racine , repertoire_actuel , ligne.argument[0]);
01644         if(tmp != NULL)
01645         {
01646             repertoire_actuel = tmp;
01647         }
01648     }
01649     break;
01650 }
01651 case 9:
01652 {
01653     if( ligne.argument[0] == NULL)
01654     {
01655         printf("(EE) : rmdir a besoin d'argument !\n");
01656     }
01657     else
01658     {
01659         i = 0;
01660         while(ligne.argument[i] != NULL)
01661         {
01662             my_rmdir(racine , repertoire_actuel , ligne.argument[i]);
01663             i++;
01664         }
01665     }
01666     break;
01667 }
01668 case 15:
01669 {
01670     if(nbMot == 1)
01671     {
01672         my_history(false);
01673     }
01674     else if(ligne.argument[0] != NULL)
01675     {
01676         printf("(EE) : commande histoire mal compris !\n\n    Utilisation : histoire
[-c]\n\n");
01677     }
01678     else
01679     {
01680         i = 0;
01681         while(ligne.option[i] != NULL)
01682         {
01683             if(strcmp(ligne.option[i], "-c") == 0)
01684             {
01685                 my_history(true);
01686             }
01687             else

```

```

01688             {
01689                 printf("(EE) : Argument invalide ! \n");
01690             }
01691             i++;
01692         }
01693     }
01694     break;
01695 }
01696 case 6:
01697 {
01698     if( nbMot == 1)
01699     {
01700         printf("(EE) : rm a besoin d'argument !\n");
01701     }
01702     else
01703     {
01704         i = 0;
01705         while(ligne.argument[i] != NULL)
01706         {
01707             my_rm(racine , repertoire_actuel , ligne.argument[i]);
01708             i++;
01709         }
01710     }
01711     break;
01712 }
01713 case 2:
01714 {
01715     if ( nbMot == 1 )
01716     {
01717         my_pwd(repertoire_actuel);
01718     }
01719     else if(ligne.argument[0] != NULL)
01720     {
01721         printf("Argument non autorisee !\n");
01722     }
01723     else
01724     {
01725         if(ligne.option[0] != NULL)
01726         {
01727             if( (strcmp(commande[1] , "-L") == 0 ) || (strcmp(commande[1] , "-LP") == 0 ) || ( strcmp(commande[1] , "-PL") == 0 ))
01728             {
01729                 my_pwd(repertoire_actuel);
01730             }
01731             else
01732             {
01733                 printf("(EE) : Argument Invalidé !\n");
01734             }
01735         }
01736     }
01737 }
01738 break;
01739 }
01740 case 13:
01741 {
01742     if ( nbMot == 1)
01743     {
01744         screen_clear();
01745     }
01746     else
01747     {
01748         printf("(EE) : clear ne veut pas d'argument !\n");
01749     }
01750     break;
01751 }
01752 case 3:
01753 {
01754     if( nbMot == 1)
01755     {
01756         printf("(EE) : cat a besoin d'argument pour le moment !\n");
01757     }
01758     else if(ligne.option[0] == NULL)
01759     {
01760         i = 0;
01761         while(ligne.argument[i] != NULL)
01762         {
01763             if(chercher(repertoire_actuel , ligne.argument[i]) != NULL)
01764             {
01765                 chemin = calloc(LIGNE , sizeof(char));
01766                 sprintf(chemin , "../data/cat/%s" , ligne.argument[i]);
01767                 cat_fichier(chemin , false);
01768                 free(chemin);
01769             }
01770             else
01771             {
01772                 printf("Erreur : %s n'existe pas \n", ligne.argument[i]);
01773             }
01774         i++;
01775     }
01776 }
01777

```

```
01775         }
01776     }
01777     else
01778     {
01779         i = 0;
01780         while(ligne.option[i] != NULL)
01781         {
01782             if(strcmp(ligne.option[i] , "-n") == 0)
01783             {
01784                 i = 0;
01785                 while(ligne.argument[i] != NULL)
01786                 {
01787                     if(chercher(repertoire_actuel , ligne.argument[i]) != NULL)
01788                     {
01789                         chemin = calloc(LIGNE , sizeof(char));
01790                         sprintf(chemin , "../data/cat/%s" , ligne.argument[i]);
01791                         cat_fichier(chemin , true);
01792                         free(chemin);
01793                     }
01794                 else
01795                 {
01796                     printf("Erreur : %s n'existe pas \n" , ligne.argument[i]);
01797                 }
01798             }
01799         }
01800     else
01801     {
01802         printf("%s : option inconnue !\n" , ligne.option[i]);
01803     }
01804     i++;
01805 }
01806 }
01807 break;
01808 }
01809 case 19:
01810 {
01811     if( nbMot == 1) || (ligne.argument[0] == NULL)
01812     {
01813         printf("(EE) : head a besoin de fichier comme argument !\n");
01814     }
01815     else if(ligne.option[0] == NULL)
01816     {
01817         for(i=0 ; ligne.argument[i] != NULL ; i++)
01818         {
01819             head_fichier(ligne.argument[i] , 10);
01820         }
01821     }
01822     else
01823     {
01824         nb_commande = 0 ;
01825         for(i=0 ; ligne.option[i] != NULL ; i++)
01826         {
01827             if(strcmp(ligne.option[i] , "-n") == 0)
01828             {
01829                 nb_commande = atoi(ligne.argument[0]);
01830             }
01831             else
01832             {
01833                 printf("%s : option non reconnue !\n" , ligne.option[i]);
01834             }
01835         }
01836         if(nb_commande != 0)
01837         {
01838             test = 1;
01839             while(ligne.argument[test] != NULL)
01840             {
01841                 head_fichier(ligne.argument[test] , i);
01842                 test++;
01843             }
01844         }
01845     }
01846     break;
01847 }
01848 case 10:
01849 {
01850     if( nbMot == 1) || (ligne.argument[0] == NULL)
01851     {
01852         printf("(EE) : tail a besoin de fichier comme argument !\n");
01853     }
01854     else if(ligne.option[0] == NULL)
01855     {
01856         for(i=0 ; ligne.argument[i] != NULL ; i++)
01857         {
01858             tail_fichier(ligne.argument[i] , 10);
01859         }
01860     }
01861 }
```

```

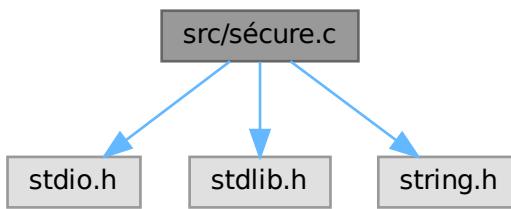
01862     {
01863         nb_commande = 0 ;
01864         for(i=0 ; ligne.option[i] != NULL ; i++)
01865         {
01866             if(strcmp(ligne.option[i] , "-n") == 0)
01867             {
01868                 nb_commande = atoi(ligne.argument[0]);
01869             }
01870             else
01871             {
01872                 printf("%s : option non reconnue !\n", ligne.option[i]);
01873             }
01874         }
01875         if(nb_commande != 0)
01876         {
01877             test = 1;
01878             while(ligne.argument[test] != NULL)
01879             {
01880                 tail_fichier(ligne.argument[test] , i);
01881                 test++;
01882             }
01883         }
01884     }
01885     break;
01886 }
01887 case 4:
01888 {
01889     my_echo( commande );
01890     break;
01891 }
01892 case 8:
01893 {
01894     nbMot = 0;
01895     while(ligne.argument[nbMot] != NULL)
01896     {
01897         if(nbMot == 0)
01898         {
01899             nbMot = 1;
01900         }
01901         nbMot++;
01902     }
01903     if(nbMot == 2)
01904     {
01905         my_mv( racine , repertoire_actuel , ligne.argument[0] , ligne.argument[1]);
01906     }
01907     else if(nbMot > 2)
01908     {
01909         nb_commande = 0;
01910         while(ligne.argument[(nb_commande)] != NULL)
01911         {
01912             nb_commande++;
01913         }
01914         for(i=0 ; i < (nb_commande-1) ; i++)
01915         {
01916             my_mv( racine , repertoire_actuel , ligne.argument[i] ,
01917                   ligne.argument[(nb_commande-1)]);
01918         }
01919     }
01920     else
01921     {
01922         printf("Vous avez besoin d'apprendre mv !\n-----> Tapez aide mv \n");
01923     }
01924     break;
01925 }
01926 case 23:
01927 {
01928     tempo = calloc(LIGNE , sizeof(char));
01929     for(i=1 ; i<nbMot ; i++)
01930     {
01931         strcat(tempo , commande[i]);
01932         strcat(tempo , " ");
01933     }
01934     system(tempo);
01935     free(tempo);
01936     break;
01937 }
01938 case 17:
01939 {
01940     if(nbMot == 1)
01941     {
01942         printf("Eh ! le numero de la mission doit etre en argument !\n");
01943     }
01944     else if(ligne.option[0] != NULL)
01945     {
01946         printf("Pas d'option accepter !\n");
01947     }
01948 }
```

```
01948         {
01949             for(i=0 ; ligne.argument[i] != NULL ; i++)
01950             {
01951                 test = atoi(ligne.argument[i]);
01952                 qui_est_meilleur(test);
01953             }
01954         }
01955     break;
01956 case 14:
01957 {
01958     if(nbMot == 1)
01959     {
01960         my_man("aide");
01961     }
01962     else if( ligne.option[0] != NULL)
01963     {
01964         printf(" Option non accepter !\n\n");
01965     }
01966     else
01967     {
01968         for(i=0 ; ligne.argument[i] != NULL ; i++)
01969         {
01970             my_man(ligne.argument[i]);
01971         }
01972     }
01973 }
01974 break;
01975 }
01976 default:
01977 {
01978     if(nbMot >= 1)
01979     {
01980         printf("(EE) %s : Commande introuvable !\n" , ligne.commande) ;
01981     }
01982     break;
01983 }
01984 }
01985
01986 return (repertoire_actuel);
01987 }
01988 int rapport_mission(int num_mission , bool reussie)
01989 {
01990     int val_mission;
01991     double valeur_temps;
01992     FILE *mission_registre;
01993     char *temps , *nom_joueur;
01994
01995     if(reussie)
01996     {
01997         val_mission = 1;
01998     }
01999     else
02000     {
02001         val_mission = 0;
02002     }
02003
02004     mission_registre = fopen("../save/record.txt" , "a+");
02005     if(mission_registre == NULL)
02006     {
02007         return (-1);
02008     }
02009     else
02010     {
02011         nom_joueur = getenv("USER");
02012         temps = getenv("TEMPS_DERNIER_PROMPT");
02013         if(nom_joueur == NULL || temps == NULL)
02014         {
02015             return (-2);
02016         }
02017         else
02018         {
02019             sscanf(temps , "%lf" , &valeur_temps);
02020             fprintf(mission_registre , "%d\t%d\t%lf\t%s\n" , num_mission , val_mission , valeur_temps
02021             , nom_joueur );
02022         }
02023     }
02024
02025     fclose(mission_registre);
02026     return (1);
02027 }
```

## 4.29 src/sécur.c File Reference

Contenant 04 fonction pour qui lit en toute sécurité les reponses insereés sur "stdin" par l'utilisateur.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
Include dependency graph for sécur.c:
```



### Functions

- void [lireLigne](#) (char \*line, int taille)
- int [lireEntier](#) ()
 

*Lire un entier.*
- float [lireFloat](#) ()
 

*Lit un float.*
- double [lireDouble](#) ()
 

*Lit un double.*

### 4.29.1 Detailed Description

Contenant 04 fonction pour qui lit en toute sécurité les reponses insereés sur "stdin" par l'utilisateur.

#### Date

24 Sept 2025

#### Author

RAKOTOARIVONY Zo Mamitiana Olivier

Definition in file [sécur.c](#).

## 4.29.2 Function Documentation

### 4.29.2.1 lireDouble()

```
double lireDouble ( )
```

Lit un double.

#### Returns

Return un variable de type double

Definition at line 69 of file [sécur.c](#).

Here is the call graph for this function:



### 4.29.2.2 lireEntier()

```
int lireEntier ( )
```

Lire un entier.

#### Returns

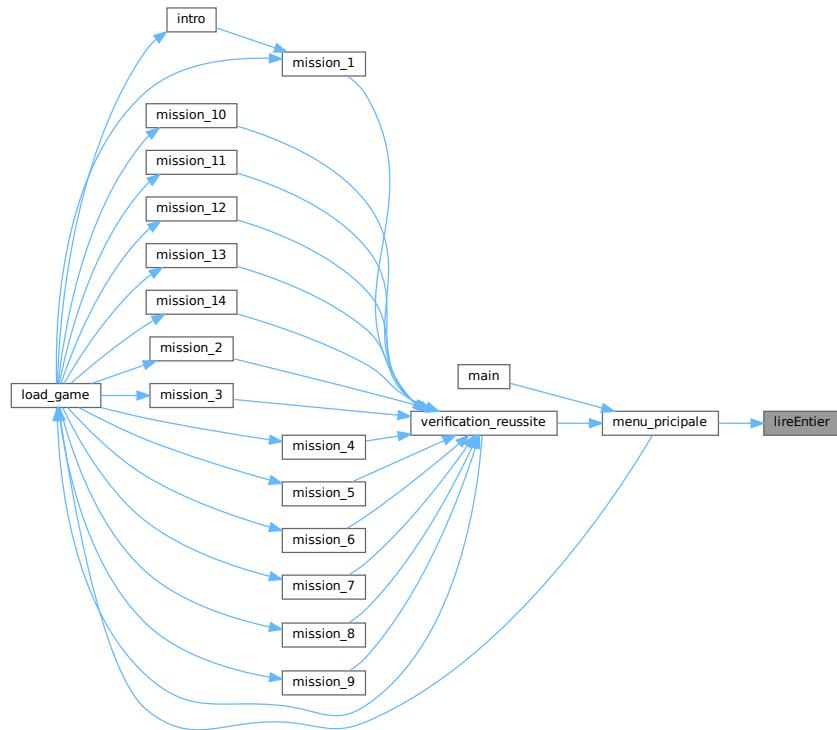
Return l'entier entré

Definition at line 40 of file [sécur.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.29.2.3 lireFloat()

```
float lireFloat ( )
```

Lit un float.

##### Returns

Return le float entré

Definition at line 55 of file [sécure.c](#).

Here is the call graph for this function:

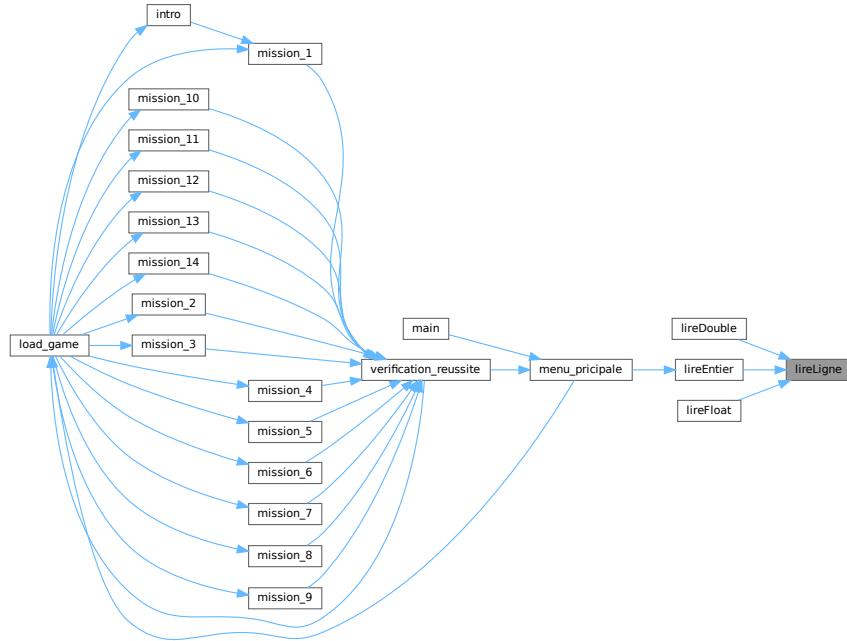


#### 4.29.2.4 lireLigne()

```
void lireLigne (
    char * line,
    int taille )
```

Definition at line 20 of file [sécurisé.c](#).

Here is the caller graph for this function:



## 4.30 sécurisé.c

[Go to the documentation of this file.](#)

```
00001
00009 #include <stdio.h>
00010 #include <stdlib.h>
00011 #include <string.h>
00012
00013
00020 void lireLigne( char *line , int taille)
00021 {
00022     int size = 0;
00023
00024     if (fgets(line , taille , stdin))
00025     {
00026         size = strlen(line);
00027         if ( size > 0 && line[size-1] == '\n' ) // Supprime "\n" inserez automatiquement par
00028         fgets
00029             {
00030                 line[size-1]='\0';
00031             }
00032     }
00033
00034
00040 int lireEntier()
00041 {
00042     char entier[50]="";
00043 }
```

```
00044     lireLigne(entier , sizeof(entier));
00045     return atoi(entier);
00046 }
00048
00049
00055 float lireFloat()
00056 {
00057     char reel[50]="";
00058     lireLigne(reel , sizeof(reel));
00059
00060     return strtod(reel , NULL);
00061 }
00062
00063
00069 double lireDouble()
00070 {
00071     char reel[50]="";
00072     lireLigne(reel , sizeof(reel));
00073
00074     return strtod(reel , NULL);
00075 }
```