Projet L2 Informatique - The Hive

Généré par Doxygen 1.8.13

Table des matières

Chapitre 1

Index des classes

1.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

Chapitre 2

Index des fichiers

2.1 Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

Chapitre 3

Documentation des classes

3.1 Référence de la structure cell_t

Structure pour un hexagone.

`#include <commun.h>`

Attributs publics

- hex_t type
  Type d'hexagone.
- int encounter
  Indicateur de combat sur cet hexagone.
- int quest_id
  Identifiant quête associé à l'hexagone.
- categ_hexa categ
  Catégorie d'hexagone.
- int scavenged
  Indicateur si le joueur a déjà fouillé l'hexagone.

3.1.1 Description détaillée

Structure pour un hexagone.

La documentation de cette structure a été générée à partir du fichier suivant :

- lib/commun.h

3.2 Référence de la structure item_t

Structure pour un item.

`#include <commun.h>`

Attributs publics

- char name [20]
  Nom item.
- type_t type
  Type item.
- int attack [3]
  Valeurs d'attaque.
- int hitchance [3]
  Chances d'attaquer.
- float defense
  Valeur de défense.
- equip_t equipable
  Indicateur si le joueur peut s'équiper avec cet item et où
- int pc_nature
  Pourcentage de chance de trouver cet item dans un hexagone de catégorie nature.
- int pc_urban
  Pourcentage de chance de trouver cet item dans un hexagone de catégorie urbain.
- int pc_military
  Pourcentage de chance de trouver cet item dans un hexagone de catégorie militaire.
- int index
  Position de l'item dans l'inventaire du joueur (-1 si absent)

3.2.1 Description détaillée

Structure pour un item.

La documentation de cette structure a été générée à partir du fichier suivant :

- lib/commun.h

3.3 Référence de la structure npc_t

Structure pour ennemi (NPC)

```
#include <commun.h>
```

Graphe de collaboration de npc_t :

Attributs publics

- char name [10]
  Nom ennemi.
- int pv
  Points de vie ennemi.
- item_t * weapon
  Pointeur sur arme.
- item_t * armor
  Pointeur sur armure.

3.3.1 Description détaillée

Structure pour ennemi (NPC)

La documentation de cette structure a été générée à partir du fichier suivant :

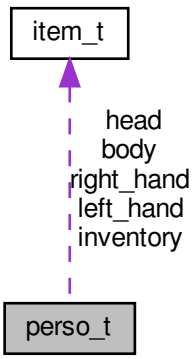
- lib/commun.h

3.4 Référence de la structure perso_t

Structure pour un personnage.

```
#include <commun.h>
```

Graphe de collaboration de perso_t :



Attributs publics

- int pv
  Points de vie.
- int pa
  Points d'action.
- int pe
  Points d'énergie.
- int posY
  Position Y (ligne) sur la carte.
- int posX
  Position X (colonne) sur la carte.
- int turns
  Nombre de tours restants.
- item_t inventory [INVENTORY_CAPACITY]
  Inventaire.
- int nb_items_inventory
  Nombre d'items dans l'inventaire.
- item_t * left_hand
  Pointeur sur item porté à la main gauche.
- item_t * right_hand
  Pointeur sur item porté à la main droite.
- item_t * body
  Pointeur sur item porté le corps.
- item_t * head
  Pointeur sur item porté sur la tête.

3.4.1 Description détaillée

Structure pour un personnage.

La documentation de cette structure a été générée à partir du fichier suivant :

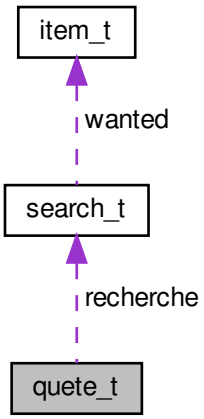
- lib/commun.h

3.5 Référence de la structure quete_t

Structure de suivi des quêtes.

`#include <commun.h>`

Graphe de collaboration de quete_t :



Attributs publics

- int soin
  Indicateur pour savoir l'avancée du joueur dans la quête "soin", si le joueur a aidé l'homme blessé
- search_t recherche
  Indicateur de type search_t pour savoir l'avancée du joueur dans la quête "recheche".
- int bunker
  Indicateur pour savoir l'avancée du joueur dans la quête "bunker".
- int montagne
  Indicateur pour savoir l'avancée du joueur dans la quête "montagne".
- int frontiere
  Indicateur pour savoir l'avancée du joueur dans la quête "frontiere".
- int bandits
  Indicateur pour savoir l'avancée du joueur dans la quête "bandits".

3.5.1 Description détaillée

Structure de suivi des quêtes.

- int = -1 : quête encore non jouée
- int = 0 : quête en cours
- int = 1 : quête déja jouée/finie
- soin = 2 : quête jouée le joueur a aide l'homme blesse
- soin = 3 : quête jouée le joueur a voulu aider l'homme blesse mais sans succes

La documentation de cette structure a été générée à partir du fichier suivant :
- lib/commun.h

3.6 Référence de la structure sauv_t

Structure pour sauvegarder les parties.

```
#include <commun.h>
```

Attributs publics

- int numPartie
  Numéro de la partie jouée.
- int sauv1_existe
  Indicateur si sauvegarde 1 existe.
- int sauv2_existe
  Indicateur si sauvegarde 2 existe.
- int sauv3_existe
  Indicateur si sauvegarde 3 existe.
- char nomPartie1 [21]
  Nom de la partie 1.
- char nomPartie2 [21]
  Nom de la partie 2.
- char nomPartie3 [21]
  Nom de la partie 3.

3.6.1 Description détaillée

Structure pour sauvegarder les parties.

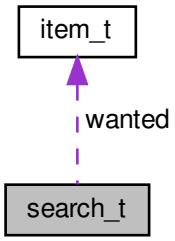
La documentation de cette structure a été générée à partir du fichier suivant :
- lib/commun.h

3.7 Référence de la structure search_t

Structure de suivi de la quête "recherche".

`#include <commun.h>`

Graphe de collaboration de search_t :



Attributs publics

- int situation
  Indicateur pour savoir l'avancée du joueur dans la quête "recherche".
- item_t wanted
  Item que le joueur doit trouver.
- int trouve
  Indicateur afin de savoir si l'item a été trouvé
- int bunkerY
  Coordonnée Y (ligne) du bunker.
- int bunkerX
  Coordonnée X (colonne) du bunker.

3.7.1 Description détaillée

Structure de suivi de la quête "recherche".

- situation = -1 : quête encore non jouée
- situation = 0 : quête en cours
- situation = 1 : quête déjà jouée/finie
- trouve = -1 : la recherche de l'item n'est pas encore activée
- trouve = 0 : la recherche de l'item est en cours
- trouve = 1 : l'item recherché est trouvé

La documentation de cette structure a été générée à partir du fichier suivant :
- lib/commun.h

3.8 Référence de la structure stat_t

Structure pour un champ de bataille.

`#include <commun.h>`

Attributs publics

- int posA
  Position joueur.
- int posB
  Position l'ennemi.
- int coverA
  Couverture joueur.
- int coverB
  Couverture l'ennemi.
- int distance
  Distance entre le joueur et l'ennemi.

3.8.1 Description détaillée

Structure pour un champ de bataille.

La documentation de cette structure a été générée à partir du fichier suivant :
- lib/commun.h

Chapitre 4

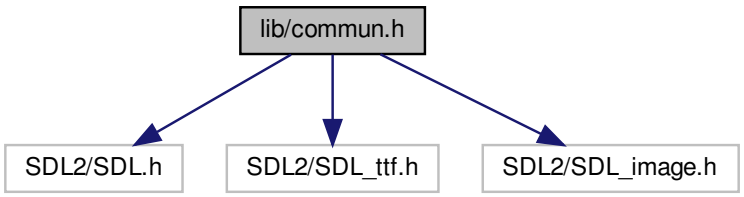
Documentation des fichiers

4.1 Référence du fichier lib/commun.h

Tous les prototypes de fonctions, énumérations, structures, macros appelés dans plusieurs fichiers du projet.

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_image.h>
```

Graphe des dépendances par inclusion de commun.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Classes

- struct item_t
  Structure pour un item.
- struct perso_t
  Structure pour un personnage.
- struct cell_t
  Structure pour un hexagone.
- struct stat_t
  Structure pour un champ de bataille.

- struct npc_t
  Structure pour ennemi (NPC)
- struct search_t
  Structure de suivi de la quête "recherche".
- struct quete_t
  Structure de suivi des quêtes.
- struct sauv_t
  Structure pour sauvegarder les parties.

Macros

- #define INVENTORY_CAPACITY 10
  INVENTORY_CAPACITY représente le nombre maximum d'items que peut contenir l'inventaire du joueur (10 items dans la version 1.0)
- #define ITEMS_MAX 5
  ITEMS_MAX représente le nombre maximum d'items générés sur un hexagone de la carte.
- #define D 15
  D représente la dimension de la carte.
- #define NOT_EQUIPPED 0
  NOT_EQUIPPED indique qu'un item n'est pas équipé
- #define LEFT_HAND 1
  LEFT_HAND indique qu'un item est équipé sur la main gauche du joueur.
- #define RIGHT_HAND 2
  RIGHT_HAND indique qu'un item est équipé sur la main droite du joueur.
- #define BODY 3
  BODY indique qu'un item est équipé sur le corps du joueur.
- #define HEAD 4
  HEAD indique qu'un item est équipé sur la tête du joueur.
- #define **N** 13

Énumérations

- enum type_t { armor, weapon, misc, food }
  Type d'un item.
- enum equip_t { none, hand, body, head }
  Zone où un item est équipable.
- enum hex_t {
  prairie = 1, foret, ville, lac,
  camp_mil, camp_ban, market, favela,
  montagne, frontiere, mer, wasteland }
  Type d'un hexagone.
- enum categ_hexa { other, nature, urbain, militaire }
  Catégorie hexagone.

Fonctions

- void entree_pour_continuer ()
  Actionne la continuité du jeu par la touche "entrée".
- void clrscr ()
  Efface la sortie du terminal, le remet à zéro.
- int range (int a, int b)
  Génère un nombre aléatoire compris entre deux entiers passés en paramètre.
- int rng (int prob)
  Génère 0 ou 1 en fonction de la probabilité en pourcentage passée en paramètre.
- item_t * creer_item (char *chaine, type_t type, int attack0, int attack1, int attack2, int hitchance0, int hitchance1, int hitchance2, float defense, int equipable, int pc_nature, int pc_urban, int pc_military)
  Crée un item à partir des informations passées en paramètres.
- int creation_tab_item (item_t *Tab_Items, int *nb_items)
  Récupère les items du fichier 'data/items.csv' et les stocke dans le tableau passé en paramètres.
- void display_item (item_t item)

Affiche toutes les caractéristiques d'un item (nom, type, valeur d'attaque si arme, valeur de défense si armure, équipable ou non, pourcentage de chance de trouver cet item sur un hexagone de catégorie nature, urbain et militaire)

- void afficher_type_categ_hexa (cell_t map[D][D], int l, int c)
  Affiche le type et la catégorie de l'hexagone de la carte dont les coordonnées sont passées en paramètres.
- void informations_map (cell_t map[D][D])
  Affiche les informations de la carte.
- int coordonnees_valides (int l, int c)
  Vérifie si les coordonnées sont valides.
- void quest_init (cell_t map[D][D], int quest_map[6][2])
  Initialise les quêtes (quest_id) sur la carte.
- void display_TEXT (int l, int c, cell_t map[D][D])
  Affiche la map en version texte avec la légende.
- void map_init (cell_t map[D][D], int quest_map[6][2])
  Initialise la carte au début de chaque partie.
- void portable_switch (int i, int j, cell_t map[D][D])
  Affiche le code de la cellule.
- void init_player (perso_t *player, cell_t map[D][D])
  Initialise les paramètres du joueur quand il commence le jeu.
- void display_player_characteristics (cell_t map[D][D], perso_t player)
  Affiche les paramètres du joueur (points de vie, points d'énergie, points d'action, position sur la carte, nombre de tours restants)
- npc_t * init_npc (item_t *Tab_Items)
  Initialise un non-player character (NPC) ennemi (nom, arme, armure et points de vie)
- stat_t * init_field ()
  Initialise le champ de bataille du combat (positions et couvertures des adversaires, distance qui les sépare)
- void combat (perso_t *player, npc_t *enemy, stat_t *field, cell_t map[D][D], item_t *Tab_Items, int nb_↩ items_available)
  Fonction principale du combat qui gère les actions du joueur et ennemi.
- void move (perso_t *player, cell_t map[D][D])
  Déplace le joueur où il le souhaite, si cela est possible (conditions : coordonnées valides et assez de points d'action)
- void random_move (perso_t *player, cell_t map[D][D])
  Déplace aléatoirement le joueur sur un des hexagones qui l'entoure.
- void eat_or_drink (perso_t *player, item_t item)
  Permet au joueur de boire ou manger un item de type food et récupérer des points d'énergie ou action (si cela est possible).
- void check_the_map (perso_t player, cell_t map[D][D])
  Affiche la carte, si le joueur en possède une dans son inventaire.
- int item_in_inventory (perso_t player, char *nom_item)
  Recherche si l'item dont le nom est passé en paramètre est présent ou non dans l'inventaire.
- void display_inventory (perso_t player)
  Affiche l'inventaire du joueur.
- void delete_item_in_inventory (perso_t *player, item_t item)
  Retire l'item passé en paramètre de l'inventaire (et de l'équipement si besoin) du joueur.
- int add_item_to_inventory (perso_t *player, item_t item)
  Ajoute un item à l'inventaire du joueur.
- void manage_inventory (perso_t *player)
  Fonction centrale du fichier inventory.c permettant au joueur de gérer son inventaire.
- void display_equipment_player (perso_t player)
  Affiche l'équipement du joueur.
- int is_equipped (perso_t player, item_t item)
  Indique si le joueur est équipé de l'item passé en paramètre.
- int nb_equipement (perso_t player)
  Compte le nombre d'item(s) équipé(s) sur le joueur.
- void manage_equipment (perso_t *player)
  Fonction centrale du fichier equipment.c permettant au joueur de gérer son équipement.
- void fish (perso_t *player, cell_t map[D][D])
  Permet au joueur de pêcher s'il se situe sur un hexagone de type lac ou mer et s'il a une canne à pêche dans son inventaire.
- void next_turn (perso_t *player)
  Calcule le nombre de points d'action récupérés à partir de la valeur des points d'énergie du joueur puis passe au tour suivant.
- void rest_and_heal (perso_t *player)
  Permet au joueur de se reposer et récupérer des points de vie et points d'énergie (proportionnellement au nombre de points d'action)
- int exit_game ()
  Propose au joueur de quitter ou non la carte lorsqu'il vient de trouver la sortie.
- void informations_quetes (cell_t map[D][D], int quest_map[6][2], quete_t quete)
  Affiche les informations des quêtes (coordonnées et états)

- void init_quete (quete_t *quete, int quest_map[6][2], item_t *Tab_Items, int nb_items_available)
  Initialisation d'une variable de type quete_t.
- int quetes (perso_t *player, cell_t map[D][D], int quest_map[6][2], quete_t *quete, item_t *Tab_Items, int nb_items_available)
  Récupère le numéro de la quête pour accéder à la quête correspondante.
- int quete_montagne (perso_t *player, quete_t *quete)
  Accès à la quete "montagne".
- int quete_frontiere (perso_t *player, quete_t *quete)
  Accès à la quete "frontière".
- int quete_bunker (perso_t *player, quete_t *quete)
  Accès à la quête "bunker".
- int quete_bandits (perso_t *player, quete_t *quete, item_t *Tab_Items, int nb_items_available, cell_t map[D][D])
  Accès à la quete "bandits".
- int quete_soin (perso_t *player, quete_t *quete, item_t *Tab_Items)
  Déroulement de la quête "soin" : rencontre d'un individu blessé.
- int quete_recherche (perso_t *player, cell_t map[D][D], quete_t *quete)
  Accès à la quete "recherche".
- void affichage_quete_search_en_cours (quete_t quete, cell_t map[D][D], perso_t player)
  Affiche certaines informations selon l'état de la quête recherche.
- int compter_items_urbain (item_t *Tab_Items, int nb_items_available)
  Compte le nombre d'items pouvant être trouvés dans un hexagone de catégorie urbain (pc_urban > 0).
- void init_Tab_Items_urbain (item_t *Tab_items_search, item_t *Tab_Items, int nb_items_urbain)
  Initialise un tableau contenant tous les items pouvant être trouvés dans un hexagone de catégorie urbain.
- void scavenge (cell_t map[D][D], perso_t *player, item_t *Tab_Items, int nb_items_available, quete_t quete)
  Permet au joueur de fouiller l'hexagone sur lequel il se trouve pour récupérer des items.
- void sauvegarder_progression (perso_t player, cell_t map[D][D], int quest_map[6][2], quete_t quete, sauv_t sauv)
  Sauvegarde la progression de la partie actuellement en cours et propose au joueur de continuer ou quitter le jeu.
- int sauvegarde_existante (sauv_t sauv)
  Indique s'il existe une sauvegarde pour la partie choisie par l'utilisateur.
- void affichage_parties (sauv_t sauv)
  Affiche les parties sauvegardées et disponibles.
- void update_etat_sauvegarde (sauv_t *sauv)
  Met à jour l'état des sauvegardes (si elles existent et leurs noms)
- void effacer_partie (sauv_t sauv)
  Efface une partie choisie par l'utilisateur.
- void save (perso_t player, cell_t map[D][D], int quest_map[6][2], quete_t quete, sauv_t sauv)
  Sauvegarde les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie.
- void load (perso_t *player, cell_t map[D][D], int quest_map[6][2], quete_t *quete, sauv_t sauv)
  Charge les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie.
- void demo_afficher_items (perso_t *player, item_t *Tab_Items, int nb_items_available)
  Permet au joueur de choisir les items qu'il souhaite ajouter à son inventaire parmi ceux disponibles.
- void **init_map_essai** (int mapint[N][N])
- void map_correspondance (cell_t map_cell[D][D], int mapaff[N][N], int position_x, int position_y)
  Met en relation la map sur laquelle évolue le personnage et la amtrice gérant l'affichage de celle-ci en prenant en compte les coordonées où se situe le personnage.
- void **relation_hexa_char** (char *mapchar[ ], int mapint[ ][N])
- void affichage_case_centrale (SDL_Renderer *renderer)
  Affiche dans la case centrale de l'écran un hilight et le personnage.
- void affichage_map (SDL_Renderer *renderer, char *map[ ], int maptest[N][N], cell_t map1[D][D], perso_t player)
  Affiche la map, c'est à dire la partie composée d'hexagones à partir de la position du personnage.
- int interface ()
  Affiche l'interface en elle-même.
- void affichage_personnage (SDL_Renderer *renderer, char *img_perso, int x, int y)
  Affiche un personnage dont l'image est en paramètre aux coordonées passées en paramètres.
- int **combat_bis** (TTF_Font *police)
- void affichage_help ()
  Affiche la document d'aide pour le joueur.
- int **monscanf** (char *c)
- int **testscanf** ()

4.1.1 Description détaillée

Tous les prototypes de fonctions, énumérations, structures, macros appelés dans plusieurs fichiers du projet.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.1.2 Documentation du type de l'énumération

4.1.2.1 categ_hexa

`enum categ_hexa`

Catégorie hexagone.

Valeurs énumérées

other	Autre : camp des bandits, frontière et wasteland
nature	Nature : prairie, forêt, lac, montagne, mer
urbain	Urbain : ville, marché, favela
militaire	Militaire : camp militaire

4.1.2.2 equip_t

`enum equip_t`

Zone où un item est équipable.

Valeurs énumérées

none	Pas équipable
hand	Main
body	Corps
head	Tête

4.1.2.3 hex_t

```
enum hex_t
```

Type d'un hexagone.

Valeurs énumérées

prairie	Prairie
foret	Forêt
ville	Ville
lac	Lac
camp_mil	Camp militaire
camp_ban	Camp des bandits
market	Marché
favela	Favela
montagne	Montagne
frontiere	Frontière
mer	Mer
wasteland	Wasteland (terres abandonnées)

4.1.2.4 type_t

```
enum type_t
```

Type d'un item.

Valeurs énumérées

armor	Armure
weapon	Arme
misc	Objet divers
food	Nourriture

4.1.3 Documentation des fonctions

4.1.3.1 add_item_to_inventory()

```
int add_item_to_inventory (
            perso_t * player,
            item_t item )
```

Ajoute un item à l'inventaire du joueur.

Si son inventaire est plein, propose un échange.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
item	Item à ajouter à l'inventaire

Renvoie

Un *int* : 1 si ajout effectué. 0 sinon.

4.1.3.2 affichage_case_centrale()

```
void affichage_case_centrale (
            SDL_Renderer * renderer )
```

Affiche dans la case centrale de l'écran un hilight et le personnage.

Paramètres

SDL_Renderer	∗renderer

Renvoie

Rien

4.1.3.3 affichage_help()

```
void affichage_help ( )
```

Affiche la document d'aide pour le joueur.

Renvoie

Rien

4.1.3.4 affichage_map()

```
void affichage_map (
            SDL_Renderer * renderer,
            char * map[],
            int maptest[N][N],
            cell_t map1[D][D],
            perso_t player )
```

Affiche la map, c'est à dire la partie composée d'hexagones à partir de la position du personnage.

Paramètres

SDL_Renderer	∗renderer
char	∗map[]
int	maptest[N][N]
cell_t	map1[D][D]
perso_t	player

Renvoie

Rien

4.1.3.5 affichage_parties()

```
void affichage_parties (
            sauv_t sauv )
```

Affiche les parties sauvegardées et disponibles.

Paramètres

sauv	Etat des sauvegardes

Renvoie

Rien

4.1.3.6 affichage_personnage()

```
void affichage_personnage (
            SDL_Renderer * renderer,
            char * img_perso,
            int x,
            int y )
```

Affiche un personnage dont l'image est en paramètre aux coordonées passées en paramètres.

Paramètres

SDL_Renderer	∗renderer
char	∗img_perso
int	x
int	y

Renvoie

Rien

4.1.3.7 affichage_quete_search_en_cours()

```
void affichage_quete_search_en_cours (
            quete_t quete,
            cell_t map[D][D],
            perso_t player )
```

Affiche certaines informations selon l'état de la quête recherche.

Paramètres

quete	Etat des quêtes
map[D][D]	Matrice de la carte
player	Joueur

Renvoie

Rien

4.1.3.8 afficher_type_categ_hexa()

```
void afficher_type_categ_hexa (
            cell_t map[D][D],
            int l,
            int c )
```

Affiche le type et la catégorie de l'hexagone de la carte dont les coordonnées sont passées en paramètres.

Paramètres

map[D][D]	Matrice de la carte
l	Coordonnée ligne de l'hexagone qu'on souhaite afficher
c	Coordonnée colonne de l'hexagone qu'on souhaite afficher

Renvoie

Rien

4.1.3.9 check_the_map()

```
void check_the_map (
            perso_t player,
            cell_t map[D][D] )
```

Affiche la carte, si le joueur en possède une dans son inventaire.

Paramètres

player	Joueur
map[D][D]	Matrice de la carte

Renvoie

Rien

4.1.3.10 clrscr()

```
void clrscr ( )
```

Efface la sortie du terminal, le remet à zéro.

Renvoie

Rien

4.1.3.11 combat()

```
void combat (
            perso_t * player,
            npc_t * enemy,
            stat_t * field,
            cell_t map[D][D],
            item_t * Tab_Items,
            int nb_items_available )
```

Fonction principale du combat qui gère les actions du joueur et ennemi.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
enemy	Pointeur sur un objet de type npc_t correspondant à l'ennemi
field	Pointeur sur un objet de type stat_t correspond au champ de bataille
map[D][D]	Matrice de la carte
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Rien

4.1.3.12 compter_items_urbain()

```
int compter_items_urbain (
            item_t * Tab_Items,
            int nb_items_available )
```

Compte le nombre d'items pouvant être trouvés dans un hexagone de catégorie urbain (pc_urban $>$ 0).

Paramètres

Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Retourne un *int* : nombre d'items pouvant être trouvés sur des hexagones de catégorie urbain

4.1.3.13 coordonnees_valides()

```
int coordonnees_valides (
            int l,
            int c )
```

Vérifie si les coordonnées sont valides.

Paramètres

l	Coordonnée ligne
c	Coordonnée colonne

Renvoie

Retourne un *int* : 1 si les coordonnées sont valides, 0 si non

4.1.3.14 creation_tab_item()

```
int creation_tab_item (
            item_t * Tab_Items,
            int * nb_items )
```

Récupère les items du fichier 'data/items.csv' et les stocke dans le tableau passé en paramètres.

Affiche un message d'erreur si fichier 'items.csv' introuvable

Paramètres

Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items	Pointeur sur un entier correspondant au nombre d'items extraits du fichier externe

Renvoie

Un *int* : 1 si récupération des items réalisée avec succès. 0 sinon.

4.1.3.15 creer_item()

```
item_t* creer_item (
            char * chaine,
            type_t type,
            int attack0,
            int attack1,
            int attack2,
            int hitchance0,
            int hitchance1,
            int hitchance2,
            float defense,
            int equipable,
            int pc_nature,
            int pc_urban,
            int pc_military )
```

Crée un item à partir des informations passées en paramètres.

Paramètres

chaine	Nom de l'item
type	Type de l'item
attack0	Valeur d'attaque si distance égale à 0 pendant un combat
attack1	Valeur d'attaque si distance égale à 1 pendant un combat
attack2	Valeur d'attaque si distance égale à 2 pendant un combat
hitchance0	Probabilité en pourcentage de toucher l'ennemi si distance égale à 0 pendant un combat
hitchance1	Probabilité en pourcentage de toucher l'ennemi si distance égale à 1 pendant un combat
hitchance2	Probabilité en pourcentage de toucher l'ennemi si distance égale à 2 pendant un combat
defense	Valeur de défense
equipable	Indicateur si l'item est équipable et où
pc_nature	Probabilité en pourcentage de trouver l'item créé dans un hexagone de catégorie nature
pc_urban	Probabilité en pourcentage de trouver l'item créé dans un hexagone de catégorie urbaine
pc_military	Probabilité en pourcentage de trouver l'item créé dans un hexagone de catégorie militaire

Renvoie

Un pointeur sur un objet de type item_t correspondant à l'item créé

4.1.3.16 delete_item_in_inventory()

```
void delete_item_in_inventory (
            perso_t * player,
            item_t item )
```

Retire l'item passé en paramètre de l'inventaire (et de l'équipement si besoin) du joueur.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
item	Item à retirer de l'inventaire

Renvoie

Rien

4.1.3.17 demo_afficher_items()

```
void demo_afficher_items (
            perso_t * player,
            item_t * Tab_Items,
            int nb_items_available )
```

Permet au joueur de choisir les items qu'il souhaite ajouter à son inventaire parmi ceux disponibles.

Cette fonction est appelée UNIQUEMENT en mode demo.

Mode demo : ./game –demo ou ./game -d

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Rien

4.1.3.18 display_equipment_player()

```
void display_equipment_player (
            perso_t player )
```

Affiche l'équipement du joueur.

Si la tête, la main gauche, la main droite ou le corps du joueur sont équipés, indique avec quels items en précisant leurs positions dans l'inventaire.

Paramètres

player	Joueur

Renvoie

Rien

4.1.3.19 display_inventory()

```
void display_inventory (
            perso_t player )
```

Affiche l'inventaire du joueur.

Affichage des items de l'inventaire avec leurs positions, par catégorie (armes, armures, divers, nourriture). Indique si item équipé pour les armes et armures.

Paramètres

player	Joueur

Renvoie

Rien

4.1.3.20 display_item()

```
void display_item (
            item_t item )
```

Affiche toutes les caractéristiques d'un item (nom, type, valeur d'attaque si arme, valeur de défense si armure, équipable ou non, pourcentage de chance de trouver cet item sur un hexagone de catégorie nature, urbain et militaire)

Paramètres

item	Item à afficher

Renvoie

Rien

4.1.3.21 display_player_characteristics()

```
void display_player_characteristics (
            cell_t map[D][D],
            perso_t player )
```

Affiche les paramètres du joueur (points de vie, points d'énergie, points d'action, position sur la carte, nombre de tours restants)

Paramètres

map[D][D]	Matrice de la carte
player	Joueur

Renvoie

Rien

4.1.3.22 display_TEXT()

```
void display_TEXT (
            int l,
            int c,
            cell_t map[D][D] )
```

Affiche la map en version texte avec la légende.

Paramètres

l	Coordonnée ligne
c	Coordonnée colonne
map[D][D]	Matrice de la carte

Renvoie

Rien

4.1.3.23 eat_or_drink()

```
void eat_or_drink (
            perso_t * player,
            item_t item )
```

Permet au joueur de boire ou manger un item de type *food* et récupérer des points d'énergie ou action (si cela est possible).

Retire l'item mangé / bu de l'inventaire

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
item	Item que le joueur désire manger ou boire

Renvoie

Rien

4.1.3.24 effacer_partie()

```
void effacer_partie (
            sauv_t sauv )
```

Efface une partie choisie par l'utilisateur.

Paramètres

sauv	Etat des sauvegardes

Renvoie

Rien

4.1.3.25 entree_pour_continuer()

```
void entree_pour_continuer ( )
```

Actionne la continuité du jeu par la touche "entrée".

Renvoie

Rien

4.1.3.26 exit_game()

```
int exit_game ( )
```

Propose au joueur de quitter ou non la carte lorsqu'il vient de trouver la sortie.

Renvoie

Un *int* : 1 si le joueur décide de quitter la carte. 0 s'il décide de continuer l'aventure.

4.1.3.27 fish()

```
void fish (
            perso_t * player,
            cell_t map[D][D] )
```

Permet au joueur de pêcher s'il se situe sur un hexagone de type *lac* ou *mer* et s'il a une canne à pêche dans son inventaire.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte

Renvoie

Rien

4.1.3.28 informations_map()

```
void informations_map (
            cell_t map[D][D] )
```

Affiche les informations de la carte.

Paramètres

map[D][D]	Matrice de la carte

Renvoie

Rien

4.1.3.29 informations_quetes()

```
void informations_quetes (
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t quete )
```

Affiche les informations des quêtes (coordonnées et états)

Paramètres

map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Etat des quêtes

Renvoie

Rien

4.1.3.30 init_field()

```
stat_t* init_field ( )
```

Initialise le champ de bataille du combat (positions et couvertures des adversaires, distance qui les sépare)

Renvoie

Pointeur sur un objet de type stat_t correspondant au champ de bataille créé

4.1.3.31 init_npc()

```
npc_t* init_npc (
            item_t * Tab_Items )
```

Initialise un non-player character (NPC) ennemi (nom, arme, armure et points de vie)

Paramètres

Tab_Items	Tableau contenant tous les items disponibles dans le jeu

Renvoie

Pointeur sur un objet de type npc_t correspondant à l'ennemi créé

4.1.3.32 init_player()

```
void init_player (
            perso_t * player,
            cell_t map[D][D] )
```

Initialise les paramètres du joueur quand il commence le jeu.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte

Renvoie

Rien

4.1.3.33 init_quete()

```
void init_quete (
            quete_t * quete,
            int quest_map[6][2],
            item_t * Tab_Items,
            int nb_items_available )
```

Initialisation d'une variable de type quete_t.

Paramètres

quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes
quest_map[6][2]	Matrice des coordonnées des quêtes
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Rien

4.1.3.34 init_Tab_Items_urbain()

```
void init_Tab_Items_urbain (
            item_t * Tab_Items_urbain,
            item_t * Tab_Items,
            int nb_items_available )
```

Initialise un tableau contenant tous les items pouvant être trouvés dans un hexagone de catégorie urbain.

Paramètres

Tab_Items_urbain	Tableau contenant les items pouvant être trouvés sur des hexagones de catégorie urbain
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Rien

4.1.3.35 interface()

```
int interface ( )
```

Affiche l'interface en elle-même.

Renvoie

0 si tout c'est bien passé

4.1.3.36 is_equipped()

```
int is_equipped (
            perso_t player,
            item_t item )
```

Indique si le joueur est équipé de l'item passé en paramètre.

Paramètres

player	Joueur
item	Item

Renvoie

Un *int* : si le joueur n'est pas équipé de l'item retourne 0 (NOT_EQUIPPED), sinon retourne où l'item est équipé sur le joueur (LEFT_HAND = 1, RIGHT_HAND = 2, BODY = 3, HEAD = 4)

4.1.3.37 item_in_inventory()

```
int item_in_inventory (
            perso_t player,
            char * nom_item )
```

Recherche si l'item dont le nom est passé en paramètre est présent ou non dans l'inventaire.

Paramètres

player	Joueur
nom_item	Nom de l'item à rechercher dans l'inventaire

Renvoie

Un *int* : position de l'item dans l'inventaire si présent, -1 si absent

4.1.3.38 load()

```
void load (
            perso_t * player,
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t * quete,
            sauv_t sauv )
```

Charge les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie.

Informations sur le joueur : points de vie, points d'énergie, points d'action, position sur la carte, nombre de tours restants

Informations sur la carte : pour chaque case de la matrice map, chargement de son type, sa catégorie, s'il y a un combat, si le joueur a déjà fouillé la case et si une quête y est positionnée.

Informations sur les quêtes : chargement des coordonnées de chaque quête et de leurs états.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes
sauv	Etat des sauvegardes

Renvoie

Rien

4.1.3.39 manage_equipment()

```
void manage_equipment (
            perso_t * player )
```

Fonction centrale du fichier equipment.c permettant au joueur de gérer son équipement.

Menu équipement : Possibilité pour le joueur de s'équiper d'un item de son inventaire, de retirer un item de son équipement.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

4.1.3.40 manage_inventory()

```
void manage_inventory (
            perso_t * player )
```

Fonction centrale du fichier inventory.c permettant au joueur de gérer son inventaire.

Menu inventaire : Possibilité pour le joueur d'en savoir plus sur un de ses items, de se débarasser d'un item, de manger/boire un item, d'utiliser son kit médical (s'il en possède un)

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

4.1.3.41 map_correspondance()

```
void map_correspondance (
            cell_t map_cell[D][D],
            int mapaff[N][N],
            int position_x,
            int position_y )
```

Met en relation la map sur laquelle évolue le personnage et la amtrice gérant l'affichage de celle-ci en prenant en compte les coordonées où se situe le personnage.

Paramètres

cell_t	map_cell[D][D]
int	mapaff[N][N]
int	position_x
int	position_y

Renvoie

Rien

4.1.3.42 map_init()

```
void map_init (
            cell_t map[D][D],
            int quest_map[6][2] )
```

Initialise la carte au début de chaque partie.

Paramètres

map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
player	Joueur

Renvoie

Rien

4.1.3.43 move()

```
void move (
            perso_t * player,
            cell_t map[D][D] )
```

Déplace le joueur où il le souhaite, si cela est possible (conditions : coordonnées valides et assez de points d'action)

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte

Renvoie

Rien

4.1.3.44 nb_equipement()

```
int nb_equipement (
            perso_t player )
```

Compte le nombre d'item(s) équipé(s) sur le joueur.

Paramètres

player	Joueur

Renvoie

Un *int* correspondant au nombre d'équipement(s) (items) actuellement sur le joueur

4.1.3.45 next_turn()

```
void next_turn (
            perso_t * player )
```

Calcule le nombre de points d'action récupérés à partir de la valeur des points d'énergie du joueur puis passe au tour suivant.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

4.1.3.46 portable_switch()

```
void portable_switch (
            int i,
            int j,
            cell_t map[D][D] )
```

Affiche le code de la cellule.

Cette fonction a été créée pour éviter de refaire le switch dans différentes fonctions d'affichage.

Paramètres

i	Coordonnée ligne
j	Coordonnée colonne
map[D][D]	Matrice de la carte

Renvoie

Rien

4.1.3.47 quest_init()

```
void quest_init (
            cell_t map[D][D],
            int quest_map[6][2] )
```

Initialise les quêtes (quest_id) sur la carte.

Les quêtes sont placées aléatoirement sur la carte.

Paramètres

map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes

Renvoie

Rien

4.1.3.48 quete_bandits()

```
int quete_bandits (
            perso_t * player,
            quete_t * quete,
            item_t * Tab_Items,
            int nb_items_available,
            cell_t map[D][D] )
```

Accès à la quete "bandits".

Le joueur arrive sur un camp de bandits. Il a le choix entre fuir, attendre les hommes ou voler des items. Il en sort vivant ou mort.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu
map[D][D]	Matrice de la carte

Renvoie

Retourne un *int* : 0 si le jeu continue, 1 si le jeu est fini et -1 si problème dans la quête.

4.1.3.49 quete_bunker()

```
int quete_bunker (
            perso_t * player,
            quete_t * quete )
```

Accès à la quête "bunker".

Si le joueur possède le pass_card, il aura le choix de rentrer dans le bunker ou non. Sinon il fait demi-tour.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes

Renvoie

Retourne un *int* : 0 si le jeu continue, 1 si le jeu est fini et -1 si problème dans la quête.

4.1.3.50 quete_frontiere()

```
int quete_frontiere (
            perso_t * player,
            quete_t * quete )
```

Accès à la quete "frontière".

Le joueur trouve la sortie de la frontière, il a le choix de la franchir (finir le jeu) ou non. S'il a joué la quête "soin" et qu'il a aidé l'homme blessé alors ses chances de la franchir sont importantes. Mais s'il ne l'a pas fait, ses chances le sont moins. La chance est donnée par un nombre entier compris entre 0 et 100.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes

Renvoie

Retourne un *int* : 0 si le jeu continue, 1 si le jeu est fini et -1 si problème dans la quête.

4.1.3.51 quete_montagne()

```
int quete_montagne (
            perso_t * player,
            quete_t * quete )
```

Accès à la quete "montagne".

Le joueur a le choix de franchir la montagne (finir le jeu) ou non. S'il a en sa possession l'équipement de montagne alors ses chances de s'échapper sont importantes. Mais s'il ne l'a pas, il est très risqué pour lui de vouloir s'échapper. La chance est donné par un nombre entier compris entre 0 et 100.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes

Renvoie

Retourne un *int* : 0 si le jeu continue, 1 si le jeu est fini et -1 si problème dans la quête.

4.1.3.52 quete_recherche()

```
int quete_recherche (
            perso_t * player,
            cell_t map[D][D],
            quete_t * quete )
```

Accès à la quete "recherche".

Le joueur doit aller à un endroit donné pour trouver un item et le ramener.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes

Renvoie

Retourne un *int* : 0 si le jeu continue et -1 si problème dans la quête.

4.1.3.53 quete_soin()

```
int quete_soin (
            perso_t * player,
            quete_t * quete,
            item_t * Tab_Items )
```

Déroulement de la quête "soin" : rencontre d'un individu blessé.

Le joueur rencontre un individu blesse. Il a le choix entre l'ignorer, lui voler ses items ou l'aider. Cet individu est un homme de l'Etat, en charge de la protection des civiles (policier ou soldat). Pour pouvoir aider le blesse le joueur doit être en possession du medical kit ou de nourriture.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes
Tab_Items	Tableau contenant tous les items disponibles dans le jeu

Renvoie

Retourne un *int* : 0 si le jeu continue et -1 si probleme dans la quete.

4.1.3.54 quetes()

```
int quetes (
            perso_t * player,
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t * quete,
            item_t * Tab_Items,
            int nb_items_available )
```

Récupère le numéro de la quête pour accéder à la quête correspondante.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Pointeur sur un objet de type quete_t correspond à l'état des quêtes
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Retourne un *int* : 0 si le jeu continue, 1 si le jeu est fini et -1 si problème dans la quête.

4.1.3.55 random_move()

```
void random_move (
            perso_t * player,
            cell_t map[D][D] )
```

Déplace aléatoirement le joueur sur un des hexagones qui l'entoure.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte

Renvoie

Rien

4.1.3.56 range()

```
int range (
            int a,
            int b )
```

Génère un nombre aléatoire compris entre deux entiers passés en paramètre.

Paramètres

a	Valeur min de l'intervalle (inclus)
b	Valeur max de l'intervalle (inclus)

Renvoie

Retourne un *int* dans l'intervalle [a ;b]

4.1.3.57 rest_and_heal()

```
void rest_and_heal (
            perso_t * player )
```

Permet au joueur de se reposer et récupérer des points de vie et points d'énergie (proportionnellement au nombre de points d'action)

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

4.1.3.58 rng()

```
int rng (
            int prob )
```

Génère 0 ou 1 en fonction de la probabilité en pourcentage passée en paramètre.

Paramètres

prob	Probabilité en pourcentage

Renvoie

Retourne un *int* : 1 ou 0.

4.1.3.59 sauvegarde_existante()

```
int sauvegarde_existante (
            sauv_t sauv )
```

Indique s'il existe une sauvegarde pour la partie choisie par l'utilisateur.

Paramètres

sauv	Etat des sauvegardes

Renvoie

Un *int* : 1 si une sauvegarde existe. 0 sinon.

4.1.3.60 sauvegarder_progression()

```
void sauvegarder_progression (
            perso_t player,
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t quete,
            sauv_t sauv )
```

Sauvegarde la progression de la partie actuellement en cours et propose au joueur de continuer ou quitter le jeu.

Paramètres

player	Joueur
map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Etat des quêtes
sauv	Etat des sauvegardes

Renvoie

Rien

4.1.3.61 save()

```
void save (
            perso_t player,
```

```
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t quete,
            sauv_t sauv )
```

Sauvegarde les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie.

Informations sur le joueur : points de vie, points d'énergie, points d'action, position sur la carte, nombre de tours restants

Informations sur la carte : pour chaque case de la matrice map, sauvegarde de son type, sa catégorie, s'il y a un combat, si le joueur a déjà fouillé la case et si une quête y est positionnée.

Informations sur les quêtes : sauvegarde des coordonnées de chaque quête et de leurs états.

Paramètres

player	Joueur
map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Etat des quêtes
sauv	Etat des sauvegardes

Renvoie

Rien

4.1.3.62 scavenge()

```
void scavenge (
            cell_t map[D][D],
            perso_t * player,
            item_t * Tab_Items,
            int nb_items_available,
            quete_t quete )
```

Permet au joueur de fouiller l'hexagone sur lequel il se trouve pour récupérer des items.

Paramètres

map[D][D]	Matrice de la carte
player	Pointeur sur un objet de type perso_t correspondant au joueur
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu
quete	Etat des quêtes

Renvoie

Rien

4.1.3.63 update_etat_sauvegarde()

```
void update_etat_sauvegarde (
            sauv_t * sauv )
```

Met à jour l'état des sauvegardes (si elles existent et leurs noms)

Paramètres

sauv	Pointeur sur un objet de type sauv_t correspondant à l'état des sauvegardes

Renvoie

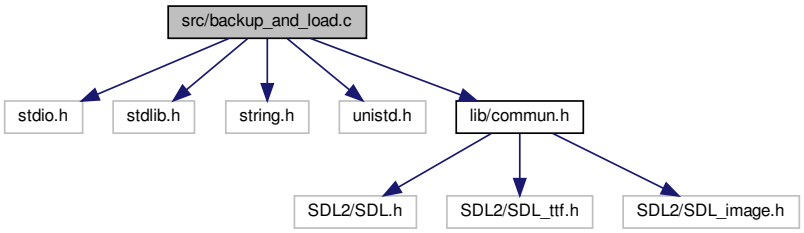
Rien

4.2 Référence du fichier src/backup_and_load.c

Sauvegarde et chargement d'une partie.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de backup_and_load.c :



Fonctions

- void sauvegarder_progression (perso_t player, cell_t map[D][D], int quest_map[6][2], quete_t quete, sauv_t sauv)
  Sauvegarde la progression de la partie actuellement en cours et propose au joueur de continuer ou quitter le jeu.
- int sauvegarde_existante (sauv_t sauv)
  Indique s'il existe une sauvegarde pour la partie choisie par l'utilisateur.

- void affichage_parties (sauv_t sauv)
  Affiche les parties sauvegardées et disponibles.
- void update_etat_sauvegarde (sauv_t *sauv)
  Met à jour l'état des sauvegardes (si elles existent et leurs noms)
- void effacer_partie (sauv_t sauv)
  Efface une partie choisie par l'utilisateur.
- void save (perso_t player, cell_t map[D][D], int quest_map[6][2], quete_t quete, sauv_t sauv)
  Sauvegarde les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie.
- void load (perso_t *player, cell_t map[D][D], int quest_map[6][2], quete_t *quete, sauv_t sauv)
  Charge les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie.

4.2.1 Description détaillée

Sauvegarde et chargement d'une partie.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.2.2 Documentation des fonctions

4.2.2.1 affichage_parties()

```
void affichage_parties (
            sauv_t sauv )
```

Affiche les parties sauvegardées et disponibles.

Paramètres

sauv	Etat des sauvegardes

Renvoie

Rien

4.2.2.2 effacer_partie()

```
void effacer_partie (
            sauv_t sauv )
```

Efface une partie choisie par l'utilisateur.

Paramètres

sauv	Etat des sauvegardes

Renvoie

Rien

4.2.2.3 load()

```
void load (
            perso_t * player,
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t * quete,
            sauv_t sauv )
```

Charge les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie.

Informations sur le joueur : points de vie, points d'énergie, points d'action, position sur la carte, nombre de tours restants

Informations sur la carte : pour chaque case de la matrice map, chargement de son type, sa catégorie, s'il y a un combat, si le joueur a déjà fouillé la case et si une quête y est positionnée.

Informations sur les quêtes : chargement des coordonnées de chaque quête et de leurs états.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes
sauv	Etat des sauvegardes

Renvoie

Rien

4.2.2.4 sauvegarde_existante()

```
int sauvegarde_existante (
            sauv_t sauv )
```

Indique s'il existe une sauvegarde pour la partie choisie par l'utilisateur.

Paramètres

sauv	Etat des sauvegardes

Renvoie

Un *int* : 1 si une sauvegarde existe. 0 sinon.

4.2.2.5 sauvegarder_progression()

```
void sauvegarder_progression (
            perso_t player,
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t quete,
            sauv_t sauv )
```

Sauvegarde la progression de la partie actuellement en cours et propose au joueur de continuer ou quitter le jeu.

Paramètres

player	Joueur
map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Etat des quêtes
sauv	Etat des sauvegardes

Renvoie

Rien

4.2.2.6 save()

```
void save (
            perso_t player,
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t quete,
            sauv_t sauv )
```

Sauvegarde les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie.

Informations sur le joueur : points de vie, points d'énergie, points d'action, position sur la carte, nombre de tours restants

Informations sur la carte : pour chaque case de la matrice map, sauvegarde de son type, sa catégorie, s'il y a un combat, si le joueur a déjà fouillé la case et si une quête y est positionnée.

Informations sur les quêtes : sauvegarde des coordonnées de chaque quête et de leurs états.

Paramètres

player	Joueur
map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Etat des quêtes
sauv	Etat des sauvegardes

Renvoie

Rien

4.2.2.7 update_etat_sauvegarde()

```
void update_etat_sauvegarde (
            sauv_t * sauv )
```

Met à jour l'état des sauvegardes (si elles existent et leurs noms)

Paramètres

sauv	Pointeur sur un objet de type sauv_t correspondant à l'état des sauvegardes

Renvoie

Rien

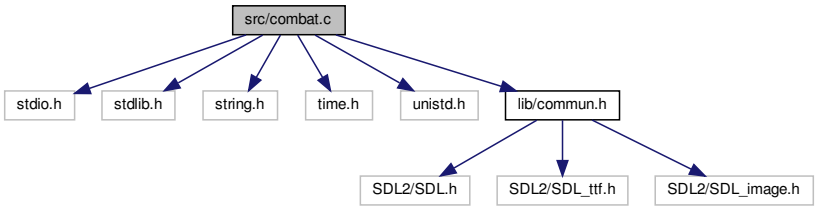
4.3 Référence du fichier src/combat.c

Combat.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
```

```
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de combat.c :



Fonctions

- void retrieve_enemy_items (item_t *Tab_Items, int nb_items_available, perso_t *player)
  Génère aléatoirement 0 à 3 items (correspondant au sac à dos de l'ennemi) que le joueur peut récupérer s'il le souhaite.
- void loot_enemy (item_t *Tab_Items, int nb_items_available, npc_t *enemy, perso_t *player)
  Propose au joueur de récupérer l'arme et/ou l'armure et/ou les items de l'ennemi.
- void damage_calculator (item_t *weapon, item_t *armor, int *hp, int distance, int cover, int scenario)
  Calcule les dommages causés par l'attaquant (scénario 1 : ennemi, scénario 2 : joueur)
- npc_t * init_npc (item_t *Tab_Items)
  Initialise un non-player character (NPC) ennemi (nom, arme, armure et points de vie)
- stat_t * init_field ()
  Initialise le champ de bataille du combat (positions et couvertures des adversaires, distance qui les sépare)
- void turn_npc (npc_t *enemy, stat_t *field, perso_t *player)
  Tour du non-player character (NPC) lors du combat, le comportement du NPC est expliqué sur ce schéma
- int run_away (int position, int distance, cell_t map[D][D], perso_t *player)
  Détermine si le joueur réussit à fuir ou non (selon sa position et la distance qui le sépare de son ennemi)
- void afficher_lettre (int chiffre, stat_t field)
  Affiche les lettres correspondant au joueur et à l'ennemi si le chiffre passé en paramètre correspond à leur position.
- void show_field (stat_t field)
  Affiche le champ de bataille du combat en indiquant où se situent le joueur et l'ennemi.
- void combat_info (int print_type, perso_t player, npc_t enemy, stat_t field)
  Affiche les informations sur le joueur et l'ennemi (pv, armes, armures, distance), le champ de bataille et les actions possibles lors du combat.
- void combat (perso_t *player, npc_t *enemy, stat_t *field, cell_t map[D][D], item_t *Tab_Items, int nb_↩ items_available)
  Fonction principale du combat qui gère les actions du joueur et ennemi.

4.3.1 Description détaillée

Combat.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.3.2 Documentation des fonctions

4.3.2.1 afficher_lettre()

```
void afficher_lettre (
            int chiffre,
            stat_t field )
```

Affiche les lettres correspondant au joueur et à l'ennemi si le chiffre passé en paramètre correspond à leur position.

Paramètres

chiffre	Chiffre correspondant à une des positions possibles sur le champ de bataille
field	Champ de bataille

Renvoie

Rien

4.3.2.2 combat()

```
void combat (
            perso_t * player,
            npc_t * enemy,
            stat_t * field,
            cell_t map[D][D],
            item_t * Tab_Items,
            int nb_items_available )
```

Fonction principale du combat qui gère les actions du joueur et ennemi.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
enemy	Pointeur sur un objet de type npc_t correspondant à l'ennemi
field	Pointeur sur un objet de type stat_t correspond au champ de bataille
map[D][D]	Matrice de la carte
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Rien

4.3.2.3 combat_info()

```
void combat_info (
            int print_type,
            perso_t player,
            npc_t enemy,
            stat_t field )
```

Affiche les informations sur le joueur et l'ennemi (pv, armes, armures, distance), le champ de bataille et les actions possibles lors du combat.

Paramètres

print_type	Configuration du combat
player	Joueur
enemy	Ennemi
field	Champ de bataille

Renvoie

Rien

4.3.2.4 damage_calculator()

```
void damage_calculator (
            item_t * weapon,
            item_t * armor,
            int * hp,
            int distance,
            int cover,
            int scenario )
```

Calcule les dommages causés par l'attaquant (scénario 1 : ennemi, scénario 2 : joueur)

Paramètres

weapon	Pointeur sur un objet de type item_t correspondant à l'arme attaquant
armor	Pointeur sur un objet de type item_t correspondant à l'armure non-attaquant
hp	Pointeur sur un objet de type int correspondant aux points de vie
distance	Distance qui sépare l'ennemi et le joueur sur le champ de bataille
cover	Couverture non-attaquant
scenario	Scénario

Renvoie

Rien

4.3.2.5 init_field()

```
stat_t * init_field ( )
```

Initialise le champ de bataille du combat (positions et couvertures des adversaires, distance qui les sépare)

Renvoie

Pointeur sur un objet de type stat_t correspondant au champ de bataille créé

4.3.2.6 init_npc()

```
npc_t * init_npc (
            item_t * Tab_Items )
```

Initialise un non-player character (NPC) ennemi (nom, arme, armure et points de vie)

Paramètres

Tab_Items	Tableau contenant tous les items disponibles dans le jeu

Renvoie

Pointeur sur un objet de type npc_t correspondant à l'ennemi créé

4.3.2.7 loot_enemy()

```
void loot_enemy (
            item_t * Tab_Items,
            int nb_items_available,
            npc_t * enemy,
            perso_t * player )
```

Propose au joueur de récupérer l'arme et/ou l'armure et/ou les items de l'ennemi.

Cette fonction est appelée uniquement si le joueur gagne au combat.

Paramètres

Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu
enemy	Pointeur sur un objet de type npc_t correspondant à l'ennemi
player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

4.3.2.8 retrieve_enemy_items()

```
void retrieve_enemy_items (
            item_t * Tab_Items,
            int nb_items_available,
            perso_t * player )
```

Génère aléatoirement 0 à 3 items (correspondant au sac à dos de l'ennemi) que le joueur peut récupérer s'il le souhaite.

Cette fonction est appelée uniquement si le joueur gagne au combat.

Paramètres

Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu
player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

4.3.2.9 run_away()

```
int run_away (
            int position,
            int distance,
            cell_t map[D][D],
            perso_t * player )
```

Détermine si le joueur réussit à fuir ou non (selon sa position et la distance qui le sépare de son ennemi)

Paramètres

position	Position du joueur sur le champ de bataille
distance	Distance entre le joueur et l'ennemi sur le champ de bataille
player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Un *int* : 1 si le joueur réussit à fuir. 0 s'il échoue.

4.3.2.10 show_field()

```
void show_field (
            stat_t field )
```

Affiche le champ de bataille du combat en indiquant où se situent le joueur et l'ennemi.

Paramètres

field	Champ de bataille

Renvoie

Rien

4.3.2.11 turn_npc()

```
void turn_npc (
            npc_t * enemy,
            stat_t * field,
            perso_t * player )
```

Tour du non-player character (NPC) lors du combat, le comportement du NPC est expliqué sur ce schéma

Paramètres

enemy	Pointeur sur un objet de type npc_t correspondant à l'ennemi
field	Pointeur sur un objet de type stat_t correspondant au champ de bataille
player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

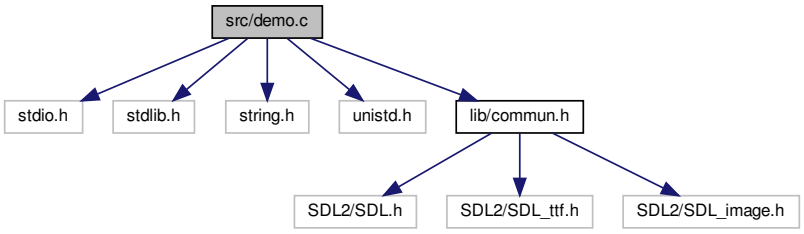
4.4 Référence du fichier src/demo.c

Démo [Mode de jeu pour choisir ses items avant de commencer la partie - utile pour tester certaines fonctionnalités].

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

```
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de demo.c :



Fonctions

— void demo_afficher_items (perso_t *player, item_t *Tab_Items, int nb_items_available)
Permet au joueur de choisir les items qu'il souhaite ajouter à son inventaire parmi ceux disponibles.

4.4.1 Description détaillée

Démo [Mode de jeu pour choisir ses items avant de commencer la partie - utile pour tester certaines fonctionnalités].

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.4.2 Documentation des fonctions

4.4.2.1 demo_afficher_items()

```
void demo_afficher_items (
            perso_t * player,
            item_t * Tab_Items,
            int nb_items_available )
```

Permet au joueur de choisir les items qu'il souhaite ajouter à son inventaire parmi ceux disponibles.

Cette fonction est appelée UNIQUEMENT en mode demo.

Mode demo : ./game –demo ou ./game -d

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

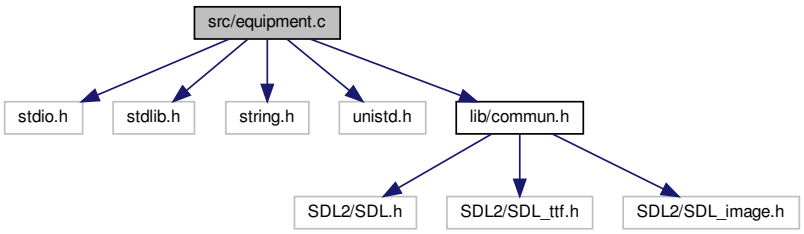
Rien

4.5 Référence du fichier src/equipment.c

Gestion de l'équipement du joueur.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de equipment.c :



Fonctions

- void display_equipment_player (perso_t player)
  Affiche l'équipement du joueur.
- int is_equipped (perso_t player, item_t item)
  Indique si le joueur est équipé de l'item passé en paramètre.
- void swap_equipment_player (perso_t *player, item_t item)
  Echange l'item passé en paramètre avec un item choisi par le joueur figurant dans son équipement.
- void equip_player (perso_t *player)
  Equipe le joueur, au bon endroit, avec un item de son inventaire qu'il choisit.
- int nb_equipement (perso_t player)
  Compte le nombre d'item(s) équipé(s) sur le joueur.
- int nb_items_equipables_non_equipe (perso_t player)
  Compte le nombre d'item(s) équipable(s) (armes et armures) mais non équipé(s) que le joueur a dans son inventaire.
- void remove_equipment_player (perso_t *player)
  Retire un item choisi par le joueur de son équipement.
- void manage_equipment (perso_t *player)
  Fonction centrale du fichier equipment.c permettant au joueur de gérer son équipement.

4.5.1 Description détaillée

Gestion de l'équipement du joueur.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.5.2 Documentation des fonctions

4.5.2.1 display_equipment_player()

```
void display_equipment_player (
            perso_t player )
```

Affiche l'équipement du joueur.

Si la tête, la main gauche, la main droite ou le corps du joueur sont équipés, indique avec quels items en précisant leurs positions dans l'inventaire.

Paramètres

player	Joueur

Renvoie

Rien

4.5.2.2 equip_player()

```
void equip_player (
            perso_t * player )
```

Equipe le joueur, au bon endroit, avec un item de son inventaire qu'il choisit.

L'item doit être équipable.

Paramètres

player	Pointeur sur un objet de type perso_t correspond au joueur

Renvoie

Rien

4.5.2.3 is_equipped()

```
int is_equipped (
            perso_t player,
            item_t item )
```

Indique si le joueur est équipé de l'item passé en paramètre.

Paramètres

player	Joueur
item	Item

Renvoie

Un *int* : si le joueur n'est pas équipé de l'item retourne 0 (NOT_EQUIPPED), sinon retourne où l'item est équipé sur le joueur (LEFT_HAND = 1, RIGHT_HAND = 2, BODY = 3, HEAD = 4)

4.5.2.4 manage_equipment()

```
void manage_equipment (
            perso_t * player )
```

Fonction centrale du fichier equipment.c permettant au joueur de gérer son équipement.

Menu équipement : Possibilité pour le joueur de s'équiper d'un item de son inventaire, de retirer un item de son équipement.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

4.5.2.5 nb_equipement()

```
int nb_equipement (
            perso_t player )
```

Compte le nombre d'item(s) équipé(s) sur le joueur.

Paramètres

player	Joueur

Renvoie

Un *int* correspondant au nombre d'équipement(s) (items) actuellement sur le joueur

4.5.2.6 nb_items_equipables_non_equipe()

```
int nb_items_equipables_non_equipe (
            perso_t player )
```

Compte le nombre d'item(s) équipable(s) (armes et armures) mais non équipé(s) que le joueur a dans son inventaire.

Paramètres

player	Joueur

Renvoie

Un *int* correspondant au nombre d'équipement(s) équipable(s) mais non équipé(s) que le joueur a dans son inventaire

4.5.2.7 remove_equipment_player()

```
void remove_equipment_player (
            perso_t * player )
```

Retire un item choisi par le joueur de son équipement.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

4.5.2.8 swap_equipment_player()

```
void swap_equipment_player (
            perso_t * player,
            item_t item )
```

Echange l'item passé en paramètre avec un item choisi par le joueur figurant dans son équipement.

Cette fonction est appelée lorsque le joueur souhaite s'équiper d'un item sur une zone de son corps déjà équipée.

Paramètres

player	Pointeur sur un objet de type perso_t correspond au joueur
item	Item

Renvoie

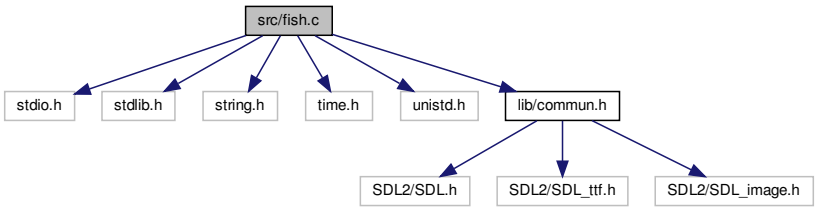
Rien

4.6 Référence du fichier src/fish.c

Pêcher.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de fish.c :



Fonctions

— void fish (perso_t *player, cell_t map[D][D])
Permet au joueur de pêcher s'il se situe sur un hexagone de type lac ou mer et s'il a une canne à pêche dans son inventaire.

4.6.1 Description détaillée

Pêcher.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.6.2 Documentation des fonctions

4.6.2.1 fish()

```
void fish (
            perso_t * player,
            cell_t map[D][D] )
```

Permet au joueur de pêcher s'il se situe sur un hexagone de type *lac* ou *mer* et s'il a une canne à pêche dans son inventaire.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte

Renvoie

Rien

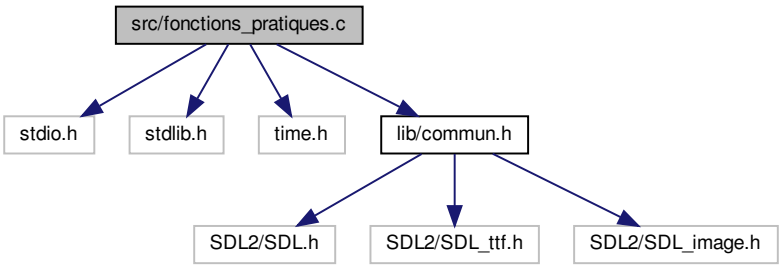
4.7 Référence du fichier src/fonctions_pratiques.c

Fontions pratiques utilisées dans tout le code.

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

```
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de fonctions_pratiques.c :



Fonctions

- void entree_pour_continuer ()
  Actionne la continuité du jeu par la touche "entrée".
- void clrscr ()
  Efface la sortie du terminal, le remet à zéro.
- int range (int a, int b)
  Génère un nombre aléatoire compris entre deux entiers passés en paramètre.
- int rng (int prob)
  Génère 0 ou 1 en fonction de la probabilité en pourcentage passée en paramètre.

4.7.1 Description détaillée

Fontions pratiques utilisées dans tout le code.

Auteur

Mathilde Mottay, Anais Mottier, Clement Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.7.2 Documentation des fonctions

4.7.2.1 clrscr()

```
void clrscr ( )
```

Efface la sortie du terminal, le remet à zéro.

Renvoie

Rien

4.7.2.2 entree_pour_continuer()

```
void entree_pour_continuer ( )
```

Actionne la continuité du jeu par la touche "entrée".

Renvoie

Rien

4.7.2.3 range()

```
int range (
            int a,
            int b )
```

Génère un nombre aléatoire compris entre deux entiers passés en paramètre.

Paramètres

a	Valeur min de l'intervalle (inclus)
b	Valeur max de l'intervalle (inclus)

Renvoie

Retourne un *int* dans l'intervalle [a ;b]

4.7.2.4 rng()

```
int rng (
            int prob )
```

Génère 0 ou 1 en fonction de la probabilité en pourcentage passée en paramètre.

Paramètres

prob	Probabilité en pourcentage

Renvoie

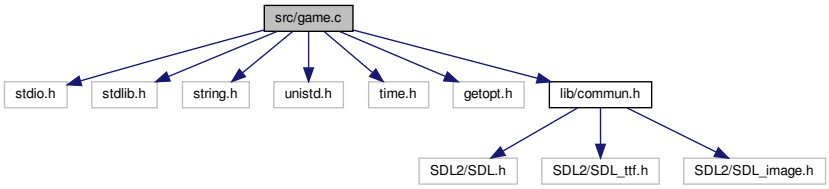
Retourne un *int* : 1 ou 0.

4.8 Référence du fichier src/game.c

Jeu.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <getopt.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de game.c :



Fonctions

- void presentation_regle_jeu ()
  Présentation des règles du jeu quand le joueur commence une nouvelle partie.
- void menu_principal_jeu (perso_t player, cell_t map[D][D], int quest_map[6][2], quete_t quete, sauv_t sauv, item_t *Tab_Items, int nb_items_available)
  Menu principal du jeu.
- void choix_partie (sauv_t *sauv, int demo)
  Propose au joueur de choisir une partie (sauvegardée ou nouvelle partie)
- int main (int argc, char *argv[ ], char *env[ ])
  Programme principal - Menu début de jeu : choisir une partie pour jouer, effacer une partie ou quitter le jeu.

4.8.1 Description détaillée

Jeu.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.8.2 Documentation des fonctions

4.8.2.1 choix_partie()

```
void choix_partie (
            sauv_t * sauv,
            int demo )
```

Propose au joueur de choisir une partie (sauvegardée ou nouvelle partie)

Initialisation différente si le joueur continue une partie précédemment sauvegardée ou en commence une nouvelle.

Paramètres

sauv	Pointeur sur un objet de type sauv_t correspondant à l'état des sauvegardes
demo	Indicateur si mode demo

Renvoie

Rien

4.8.2.2 main()

```
int main (
            int argc,
            char * argv[],
            char * env[] )
```

Programme principal - Menu début de jeu : choisir une partie pour jouer, effacer une partie ou quitter le jeu.

Exécuter ./game –demo ou ./game -d pour mode demo

Paramètres

sauv	Pointeur sur un objet de type sauv_t correspondant à l'état des sauvegardes
demo	Indicateur si mode demo

Renvoie

Rien

4.8.2.3 menu_principal_jeu()

```
void menu_principal_jeu (
            perso_t player,
```

```
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t quete,
            sauv_t sauv,
            item_t * Tab_Items,
            int nb_items_available )
```

Menu principal du jeu.

Le joueur choisit ce qu'il souhaite faire.

Paramètres

player	Joueur
map[D][D]	Matrice de la carte
quete	Etat des quêtes
sauv	Etat des sauvegardes
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Rien

4.8.2.4 presentation_regle_jeu()

```
void presentation_regle_jeu ( )
```

Présentation des règles du jeu quand le joueur commence une nouvelle partie.

Renvoie

Rien

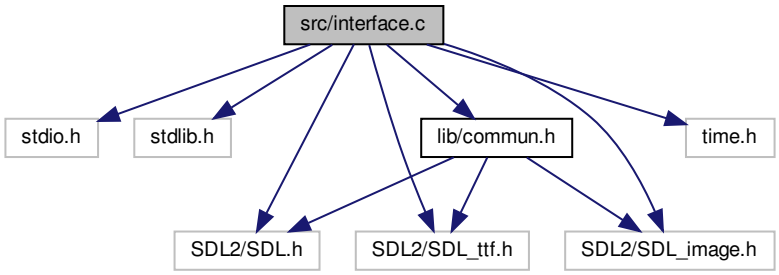
4.9 Référence du fichier src/interface.c

Affichage de l'interface de jeu via SDL2.

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_image.h>
#include <time.h>
```

```
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de interface.c :



Fonctions

- void **init_map_essai** (int mapint[N][N])
- void map_correspondance (cell_t map_cell[D][D], int mapaff[N][N], int position_x, int position_y)
  Met en relation la map sur laquelle évolue le personnage et la amtrice gérant l'affichage de celle-ci en prenant en compte les coordonées où se situe le personnage.
- void relation_hexa_char (char ∗mapchar[ ], int mapint[N][N])
  Remplit un matrice des nom des cases a afficher (images) à partir du chiffre contenu dans la matrice initiale.
- void affichage_case_centrale (SDL_Renderer ∗renderer)
  Affiche dans la case centrale de l'écran un hilight et le personnage.
- void affichage_map (SDL_Renderer ∗renderer, char ∗map[ ], int maptest[N][N], cell_t map1[D][D], perso_t player)
  Affiche la map, c'est à dire la partie composée d'hexagones à partir de la position du personnage.
- int interface ()
  Affiche l'interface en elle-même.
- int **main** ()

Variables

- char ∗ **map** [N ∗N]

4.9.1 Description détaillée

Affichage de l'interface de jeu via SDL2.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.9.2 Documentation des fonctions

4.9.2.1 affichage_case_centrale()

```
void affichage_case_centrale (
            SDL_Renderer * renderer )
```

Affiche dans la case centrale de l'écran un hilight et le personnage.

Paramètres

SDL_Renderer	∗renderer

Renvoie

Rien

4.9.2.2 affichage_map()

```
void void affichage_map (
            SDL_Renderer * renderer,
            char * map[],
            int maptest[N][N],
            cell_t map1[D][D],
            perso_t player )
```

Affiche la map, c'est à dire la partie composée d'hexagones à partir de la position du personnage.

Paramètres

SDL_Renderer	∗renderer
char	∗map[]
int	maptest[N][N]
cell_t	map1[D][D]
perso_t	player

Renvoie

Rien

4.9.2.3 interface()

```
int interface ( )
```

Affiche l'interface en elle-même.

Renvoie

0 si tout c'est bien passé

4.9.2.4 map_correspondance()

```
map_correspondance (
            cell_t map_cell[D][D],
            int mapaff[N][N],
            int position_x,
            int position_y )
```

Met en relation la map sur laquelle évolue le personnage et la amtrice gérant l'affichage de celle-ci en prenant en compte les coordonées où se situe le personnage.

Paramètres

cell_t	map_cell[D][D]
int	mapaff[N][N]
int	position_x
int	position_y

Renvoie

Rien

4.9.2.5 relation_hexa_char()

```
void relation_hexa_char (
            char * mapchar[],
            int mapint[N][N] )
```

Remplit un matrice des nom des cases a afficher (images) à partir du chiffre contenu dans la matrice initiale.

Paramètres

*char**	mapchar[]
int	mapint[N][N]

Renvoie

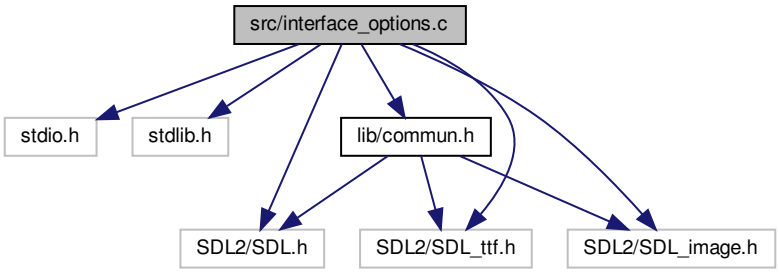
Rien

4.10 Référence du fichier src/interface_options.c

Affichage de l'interface de combat via SDL2.

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_image.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de interface_options.c :



Fonctions

- void affichage_personnage (SDL_Renderer ∗renderer, char ∗img_perso, int x, int y)
  Affiche un personnage dont l'image est en paramètre aux coordonées passées en paramètres.
- int **combat_bis** (TTF_Font ∗police)
- void affichage_help ()
  Affiche la document d'aide pour le joueur.

4.10.1 Description détaillée

Affichage de l'interface de combat via SDL2.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.10.2 Documentation des fonctions

4.10.2.1 affichage_help()

```
void affichage_help ( )
```

Affiche la document d'aide pour le joueur.

Renvoie

Rien

4.10.2.2 affichage_personnage()

```
void affichage_personnage (
            SDL_Renderer * renderer,
            char * img_perso,
            int x,
            int y )
```

Affiche un personnage dont l'image est en paramètre aux coordonées passées en paramètres.

Paramètres

SDL_Renderer	∗renderer
char	∗img_perso
int	x
int	y

Renvoie

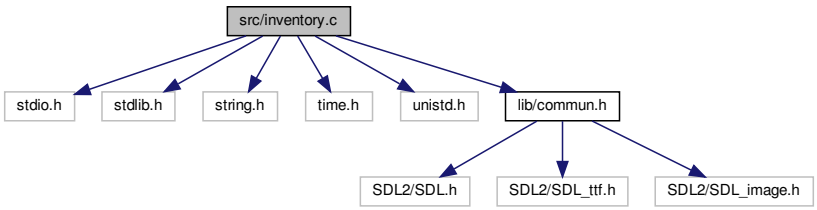
Rien

4.11 Référence du fichier src/inventory.c

Gestion de l'inventaire du joueur.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de inventory.c :



Fonctions

- void gain_energie (perso_t *player, int val_e)
  Calcule et met à jour les points d'énergie du joueur selon la valeur énergétique de l'item mangé / bu.
- void eat_or_drink (perso_t *player, item_t item)
  Permet au joueur de boire ou manger un item de type food et récupérer des points d'énergie ou action (si cela est possible).
- void check_the_map (perso_t player, cell_t map[D][D])
  Affiche la carte, si le joueur en possède une dans son inventaire.
- int item_in_inventory (perso_t player, char *nom_item)
  Recherche si l'item dont le nom est passé en paramètre est présent ou non dans l'inventaire.
- int food_in_inventory (perso_t player)
  Calcule le nombre d'items food dans l'inventaire du joueur.
- int too_much_of_the_same_item (perso_t player, item_t item)
  Indique si l'item passé en paramètre apparaît 2 fois ou plus dans l'inventaire du joueur.
- void display_inventory (perso_t player)
  Affiche l'inventaire du joueur.
- void delete_item_in_inventory (perso_t *player, item_t item)
  Retire l'item passé en paramètre de l'inventaire (et de l'équipement si besoin) du joueur.
- int add_item_to_inventory (perso_t *player, item_t item)
  Ajoute un item à l'inventaire du joueur.
- void manage_inventory (perso_t *player)
  Fonction centrale du fichier inventory.c permettant au joueur de gérer son inventaire.

4.11.1 Description détaillée

Gestion de l'inventaire du joueur.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.11.2 Documentation des fonctions

4.11.2.1 add_item_to_inventory()

```
int add_item_to_inventory (
            perso_t * player,
            item_t item )
```

Ajoute un item à l'inventaire du joueur.

Si son inventaire est plein, propose un échange.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
item	Item à ajouter à l'inventaire

Renvoie

Un *int* : 1 si ajout effectué. 0 sinon.

4.11.2.2 check_the_map()

```
void check_the_map (
            perso_t player,
            cell_t map[D][D] )
```

Affiche la carte, si le joueur en possède une dans son inventaire.

Paramètres

player	Joueur
map[D][D]	Matrice de la carte

Renvoie

Rien

4.11.2.3 delete_item_in_inventory()

```
void delete_item_in_inventory (
            perso_t * player,
            item_t item )
```

Retire l'item passé en paramètre de l'inventaire (et de l'équipement si besoin) du joueur.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
item	Item à retirer de l'inventaire

Renvoie

Rien

4.11.2.4 display_inventory()

```
void display_inventory (
            perso_t player )
```

Affiche l'inventaire du joueur.

Affichage des items de l'inventaire avec leurs positions, par catégorie (armes, armures, divers, nourriture). Indique si item équipé pour les armes et armures.

Paramètres

player	Joueur

Renvoie

Rien

4.11.2.5 eat_or_drink()

```
void eat_or_drink (
            perso_t * player,
            item_t item )
```

Permet au joueur de boire ou manger un item de type *food* et récupérer des points d'énergie ou action (si cela est possible).

Retire l'item mangé / bu de l'inventaire

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
item	Item que le joueur désire manger ou boire

Renvoie

Rien

4.11.2.6 food_in_inventory()

```
int food_in_inventory (
            perso_t player )
```

Calcule le nombre d'items *food* dans l'inventaire du joueur.

Paramètres

player	Joueur

Renvoie

Un *int* : nombre d'items *food* dans l'inventaire du joueur

4.11.2.7 gain_energie()

```
void gain_energie (
            perso_t * player,
            int val_e )
```

Calcule et met à jour les points d'énergie du joueur selon la valeur énergétique de l'item mangé / bu.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
val_e	Valeur énergétique de l'item mangé/bu

Renvoie

Rien

4.11.2.8 item_in_inventory()

```
int item_in_inventory (
            perso_t player,
            char * nom_item )
```

Recherche si l'item dont le nom est passé en paramètre est présent ou non dans l'inventaire.

Paramètres

player	Joueur
nom_item	Nom de l'item à rechercher dans l'inventaire

Renvoie

Un *int* : position de l'item dans l'inventaire si présent, -1 si absent

4.11.2.9 manage_inventory()

```
void manage_inventory (
            perso_t * player )
```

Fonction centrale du fichier inventory.c permettant au joueur de gérer son inventaire.

Menu inventaire : Possibilité pour le joueur d'en savoir plus sur un de ses items, de se débarasser d'un item, de manger/boire un item, d'utiliser son kit médical (s'il en possède un)

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

4.11.2.10 too_much_of_the_same_item()

```
int too_much_of_the_same_item (
            perso_t player,
            item_t item )
```

Indique si l'item passé en paramètre apparaît 2 fois ou plus dans l'inventaire du joueur.

Remarque : Un item peut figurer au maximum 2 fois dans l'inventaire du joueur

Paramètres

player	Joueur
item	Item

Renvoie

Un *int* : retourne 1 si l'item est présent 2 fois ou plus dans l'inventaire. 0 sinon.

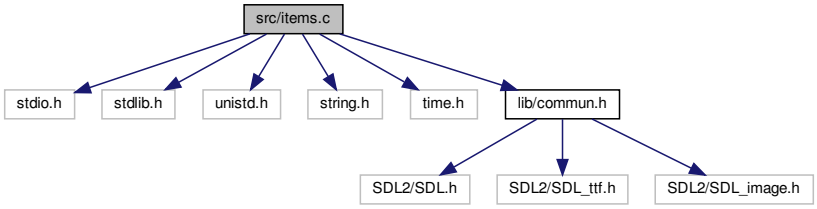
4.12 Référence du fichier src/items.c

Items (création, affichage, génération aléatoire lors de fouille d'hexagones)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
```

```
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de items.c :



Fonctions

- item_t * creer_item (char *chaine, type_t type, int attack0, int attack1, int attack2, int hitchance0, int hitchance1, int hitchance2, float defense, int equipable, int pc_nature, int pc_urban, int pc_military)
  Crée un item à partir des informations passées en paramètres.
- int creation_tab_item (item_t *Tab_Items, int *nb_items)
  Récupère les items du fichier 'data/items.csv' et les stocke dans le tableau passé en paramètres.
- void display_item (item_t item)
  Affiche toutes les caractéristiques d'un item (nom, type, valeur d'attaque si arme, valeur de défense si armure, équipable ou non, pourcentage de chance de trouver cet item sur un hexagone de catégorie nature, urbain et militaire)
- void generate_items (item_t *Tab_Items, int nb_items_available, perso_t *player, categ_hexa categ)
  Génère aléatoirement 0 à ITEMS_MAX items en prenant en compte le pourcentage de chance des items d'apparaître dans un type d'hexagone en particulier.
- void scavenge (cell_t map[D][D], perso_t *player, item_t *Tab_Items, int nb_items_available, quete_t quete)
  Permet au joueur de fouiller l'hexagone sur lequel il se trouve pour récupérer des items.

4.12.1 Description détaillée

Items (création, affichage, génération aléatoire lors de fouille d'hexagones)

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.12.2 Documentation des fonctions

4.12.2.1 creation_tab_item()

```
int creation_tab_item (
            item_t * Tab_Items,
            int * nb_items )
```

Récupère les items du fichier 'data/items.csv' et les stocke dans le tableau passé en paramètres.

Affiche un message d'erreur si fichier 'items.csv' introuvable

Paramètres

Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items	Pointeur sur un entier correspondant au nombre d'items extraits du fichier externe

Renvoie

Un *int* : 1 si récupération des items réalisée avec succès. 0 sinon.

4.12.2.2 creer_item()

```
item_t * creer_item (
            char * chaine,
            type_t type,
            int attack0,
            int attack1,
            int attack2,
            int hitchance0,
            int hitchance1,
            int hitchance2,
            float defense,
            int equipable,
            int pc_nature,
            int pc_urban,
            int pc_military )
```

Crée un item à partir des informations passées en paramètres.

Paramètres

chaine	Nom de l'item
type	Type de l'item
attack0	Valeur d'attaque si distance égale à 0 pendant un combat
attack1	Valeur d'attaque si distance égale à 1 pendant un combat
attack2	Valeur d'attaque si distance égale à 2 pendant un combat
hitchance0	Probabilité en pourcentage de toucher l'ennemi si distance égale à 0 pendant un combat
hitchance1	Probabilité en pourcentage de toucher l'ennemi si distance égale à 1 pendant un combat
hitchance2	Probabilité en pourcentage de toucher l'ennemi si distance égale à 2 pendant un combat
defense	Valeur de défense
equipable	Indicateur si l'item est équipable et où
pc_nature	Probabilité en pourcentage de trouver l'item créé dans un hexagone de catégorie nature
pc_urban	Probabilité en pourcentage de trouver l'item créé dans un hexagone de catégorie urbaine
pc_military	Probabilité en pourcentage de trouver l'item créé dans un hexagone de catégorie militaire

Renvoie

Un pointeur sur un objet de type item_t correspondant à l'item créé

4.12.2.3 display_item()

```
void display_item (
            item_t item )
```

Affiche toutes les caractéristiques d'un item (nom, type, valeur d'attaque si arme, valeur de défense si armure, équipable ou non, pourcentage de chance de trouver cet item sur un hexagone de catégorie nature, urbain et militaire)

Paramètres

item	Item à afficher

Renvoie

Rien

4.12.2.4 generate_items()

```
void generate_items (
            item_t * Tab_Items,
            int nb_items_available,
            perso_t * player,
            categ_hexa categ )
```

Génère aléatoirement 0 à ITEMS_MAX items en prenant en compte le pourcentage de chance des items d'apparaître dans un type d'hexagone en particulier.

Propose au joueur de récupérer les items générés

Paramètres

Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu
player	Pointeur sur un objet de type perso_t correspondant au joueur
categ	Catégorie de l'hexagone inspecté

Renvoie

Rien

4.12.2.5 scavenge()

```
void scavenge (
            cell_t map[D][D],
            perso_t * player,
```

```
item_t * Tab_Items,
int nb_items_available,
quete_t quete )
```

Permet au joueur de fouiller l'hexagone sur lequel il se trouve pour récupérer des items.

Paramètres

map[D][D]	Matrice de la carte
player	Pointeur sur un objet de type perso_t correspondant au joueur
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu
quete	Etat des quêtes

Renvoie

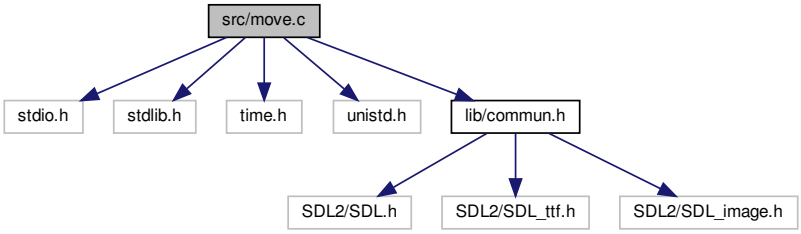
Rien

4.13 Référence du fichier src/move.c

Déplacement du joueur.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de move.c :



Fonctions

- int move_lose_pa (hex_t type_hexa)
  Calcule le nombre de points d'action nécessaires pour se déplacer dans le type d'hexagone passé en paramètre.
- void look_around (int i, int j, cell_t map[D][D])
  Affiche une vue des 8 hexagones qui entourent le joueur et les codes pour choisir où se déplacer.
- void move (perso_t *player, cell_t map[D][D])
  Déplace le joueur où il le souhaite, si cela est possible (conditions : coordonnées valides et assez de points d'action)
- void random_move (perso_t *player, cell_t map[D][D])
  Déplace aléatoirement le joueur sur un des hexagones qui l'entoure.

4.13.1 Description détaillée

Déplacement du joueur.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.13.2 Documentation des fonctions

4.13.2.1 look_around()

```
void look_around (
            int i,
            int j,
            cell_t map[D][D] )
```

Affiche une vue des 8 hexagones qui entourent le joueur et les codes pour choisir où se déplacer.

Paramètres

i	Coordonnée ligne de l'hexagone sur lequel le joueur se trouve
j	Coordonnée colonne de l'hexagone sur lequel le joueur se trouve
map[D][D]	Matrice de la carte

Renvoie

Rien

4.13.2.2 move()

```
void move (
            perso_t * player,
            cell_t map[D][D] )
```

Déplace le joueur où il le souhaite, si cela est possible (conditions : coordonnées valides et assez de points d'action)

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte

Renvoie

Rien

4.13.2.3 move_lose_pa()

```
int move_lose_pa (
            hex_t type_hexa )
```

Calcule le nombre de points d'action nécessaires pour se déplacer dans le type d'hexagone passé en paramètre.

Paramètres

type_hexa	Type de l'hexagone sur lequel le joueur souhaite se déplacer

Renvoie

Un *int* représentant le nombre de points d'action nécessaires (prairie : 1, forêt : 2, ville : 1, lac : 2, camp militaire : 2, camp des bandits : 2, marché : 1, favela : 2, montagne : 3, frontière : 1, mer : 1, wasteland : 1)

4.13.2.4 random_move()

```
void random_move (
            perso_t * player,
            cell_t map[D][D] )
```

Déplace aléatoirement le joueur sur un des hexagones qui l'entoure.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte

Renvoie

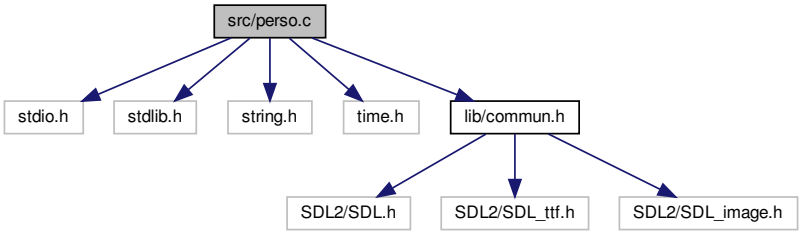
Rien

4.14 Référence du fichier src/perso.c

Initialisation et affichage joueur.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de perso.c :



Fonctions

- void init_player (perso_t *player, cell_t map[D][D])
  Initialise les paramètres du joueur quand il commence le jeu.
- void display_player_characteristics (cell_t map[D][D], perso_t player)
  Affiche les paramètres du joueur (points de vie, points d'énergie, points d'action, position sur la carte, nombre de tours restants)

4.14.1 Description détaillée

Initialisation et affichage joueur.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.14.2 Documentation des fonctions

4.14.2.1 display_player_characteristics()

```
void display_player_characteristics (
            cell_t map[D][D],
            perso_t player )
```

Affiche les paramètres du joueur (points de vie, points d'énergie, points d'action, position sur la carte, nombre de tours restants)

Paramètres

map[D][D]	Matrice de la carte
player	Joueur

Renvoie

Rien

4.14.2.2 init_player()

```
void init_player (
            perso_t * player,
            cell_t map[D][D] )
```

Initialise les paramètres du joueur quand il commence le jeu.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte

Renvoie

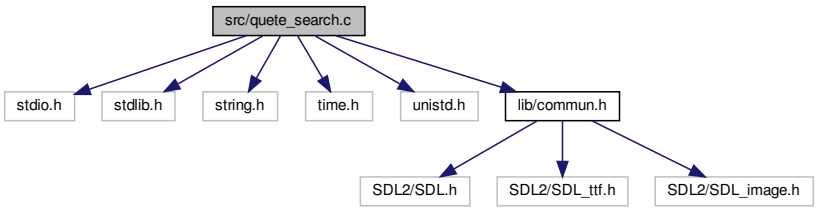
Rien

4.15 Référence du fichier src/quete_search.c

Quête "recherche".

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de quete_search.c :



Fonctions

- void affichage_quete_search_en_cours (quete_t quete, cell_t map[D][D], perso_t player)
  Affiche certaines informations selon l'état de la quête recherche.
- void init_Tab_Items_urbain (item_t *Tab_Items_urbain, item_t *Tab_Items, int nb_items_available)
  Initialise un tableau contenant tous les items pouvant être trouvés dans un hexagone de catégorie urbain.
- int compter_items_urbain (item_t *Tab_Items, int nb_items_available)
  Compte le nombre d'items pouvant être trouvés dans un hexagone de catégorie urbain (pc_urban > 0).
- int quete_recherche (perso_t *player, cell_t map[D][D], quete_t *quete)
  Accès à la quete "recherche".

4.15.1 Description détaillée

Quête "recherche".

Auteur

Mathilde Mottay, Anais Mottier, Clement Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.15.2 Documentation des fonctions

4.15.2.1 affichage_quete_search_en_cours()

```
void affichage_quete_search_en_cours (
            quete_t quete,
            cell_t map[D][D],
            perso_t player )
```

Affiche certaines informations selon l'état de la quête recherche.

Paramètres

quete	Etat des quêtes
map[D][D]	Matrice de la carte
player	Joueur

Renvoie

Rien

4.15.2.2 compter_items_urbain()

```
int compter_items_urbain (
            item_t * Tab_Items,
            int nb_items_available )
```

Compte le nombre d'items pouvant être trouvés dans un hexagone de catégorie urbain (pc_urban $>$ 0).

Paramètres

Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Retourne un *int* : nombre d'items pouvant être trouvés sur des hexagones de catégorie urbain

4.15.2.3 init_Tab_Items_urbain()

```
void init_Tab_Items_urbain (
            item_t * Tab_Items_urbain,
            item_t * Tab_Items,
            int nb_items_available )
```

Initialise un tableau contenant tous les items pouvant être trouvés dans un hexagone de catégorie urbain.

Paramètres

Tab_Items_urbain	Tableau contenant les items pouvant être trouvés sur des hexagones de catégorie urbain
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Rien

4.15.2.4 quete_recherche()

```
int quete_recherche (
            perso_t * player,
            cell_t map[D][D],
            quete_t * quete )
```

Accès à la quete "recherche".

Le joueur doit aller à un endroit donné pour trouver un item et le ramener.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes

Renvoie

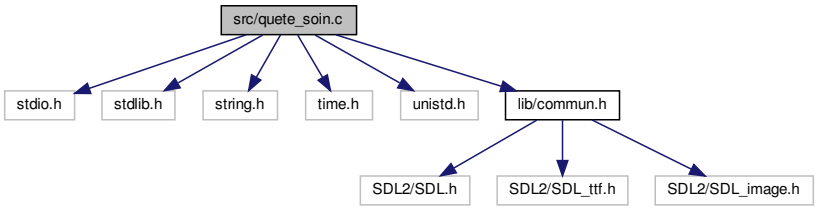
Retourne un *int* : 0 si le jeu continue et -1 si problème dans la quête.

4.16 Référence du fichier src/quete_soin.c

Quête "soin".

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de quete_soin.c :



Fonctions

- npc_t * init_npc_quete (item_t *Tab_Items, int pers)
  Initialisation du personnage de l'homme blessé.
- int recup_items_vole (perso_t *player, int nb_items_vole, npc_t *homme, item_t *pass_card, quete_t *quete)
  Récupération des items voler sur l'homme blessé par le joueur.
- int aider_homme_blesse (perso_t *player, item_t *pass_card, quete_t *quete, int pers)
  Fonction où se déroule la partie aide de la quete "soin".
- int quete_soin (perso_t *player, quete_t *quete, item_t *Tab_Items)
  Déroulement de la quête "soin" : rencontre d'un individu blessé.

4.16.1 Description détaillée

Quête "soin".

Auteur

Mathilde Mottay, Anais Mottier, Clement Mainguy, Moustapha Tsamarayev

Version

Date

2020

4.16.2 Documentation des fonctions

4.16.2.1 aider_homme_blesse()

```
int aider_homme_blesse (
            perso_t * player,
            item_t * pass_card,
            quete_t * quete,
            int pers )
```

Fonction où se déroule la partie aide de la quete "soin".

Le joueur peut aider l'homme de différentes façons, il choisit comment il souhaite le faire ou non.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
pass_card	Pointeur sur un objet de type item_t correspondant au pass
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes
pers	Indicateur si l'homme blessé dans la quête est un soldat ou policier

Renvoie

Retourne un *int* : 0 si le jeu continue et -1 si problème dans la quete.

4.16.2.2 init_npc_quete()

```
npc_t * init_npc_quete (
            item_t * Tab_Items,
            int pers )
```

Initialisation du personnage de l'homme blessé.

L'initialisation est différente en fonction du "métier" de l'homme : soldat, policier. Il n'aura pas la même arme mais aura la même armure.

Paramètres

Tab_Items	Tableau contenant tous les items disponibles dans le jeu
pers	Indicateur si l'homme blessé dans la quête est un soldat ou policier

Renvoie

Un pointeur sur un objet de type npc_t correspondant à l'homme blessé

4.16.2.3 quete_soin()

```
void quete_soin (
            perso_t * player,
            quete_t * quete,
            item_t * Tab_Items )
```

Déroulement de la quête "soin" : rencontre d'un individu blessé.

Le joueur rencontre un individu blesse. Il a le choix entre l'ignorer, lui voler ses items ou l'aider. Cet individu est un homme de l'Etat, en charge de la protection des civiles (policier ou soldat). Pour pouvoir aider le blesse le joueur doit être en possession du medical kit ou de nourriture.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes
Tab_Items	Tableau contenant tous les items disponibles dans le jeu

Renvoie

Retourne un *int* : 0 si le jeu continue et -1 si probleme dans la quete.

4.16.2.4 recup_items_vole()

```
int recup_items_vole (
            perso_t * player,
            int nb_items_vole,
            npc_t * homme,
            item_t * pass_card,
            quete_t * quete )
```

Récupération des items voler sur l'homme blessé par le joueur.

Génération des items volés + décision du joueur de les récupérer ou non dans son inventaire

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
nb_items_vole	Nombre d'items volés par le joueur
homme	Pointeur sur un objet de type npc_t correspondant à l'homme blessé
pass_card	Pointeur sur un objet de type item_t correspondant au pass
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes

Renvoie

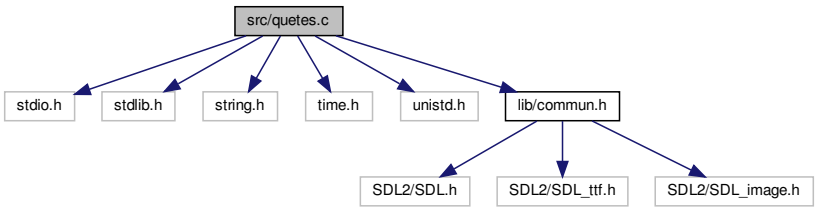
Retourne un *int* : 0 si le jeu continue et -1 si problème dans la quête.

4.17 Référence du fichier src/quetes.c

Fonctions relatives aux quêtes du jeu (initialisation, lancement des quêtes) + 4 quêtes (montagne, frontière, bunker, bandits)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de quetes.c :



Fonctions

- int exit_game ()
  Propose au joueur de quitter ou non la carte lorsqu'il vient de trouver la sortie.
- void informations_quetes (cell_t map[D][D], int quest_map[6][2], quete_t quete)
  Affiche les informations des quêtes (coordonnées et états)
- void init_quete (quete_t *quete, int quest_map[6][2], item_t *Tab_Items, int nb_items_available)
  Initialisation d'une variable de type quete_t.
- int quetes (perso_t *player, cell_t map[D][D], int quest_map[6][2], quete_t *quete, item_t *Tab_Items, int nb_items_available)
  Récupère le numéro de la quête pour accéder à la quête correspondante.
- int quete_montagne (perso_t *player, quete_t *quete)
  Accès à la quete "montagne".
- int quete_frontiere (perso_t *player, quete_t *quete)
  Accès à la quete "frontière".
- int quete_bunker (perso_t *player, quete_t *quete)
  Accès à la quête "bunker".
- int quete_bandits (perso_t *player, quete_t *quete, item_t *Tab_Items, int nb_items_available, cell_t map[D][D])
  Accès à la quete "bandits".

4.17.1 Description détaillée

Fonctions relatives aux quêtes du jeu (initialisation, lancement des quêtes) + 4 quêtes (montagne, frontière, bunker, bandits)

Auteur

Mathilde Mottay, Anais Mottier, Clement Mainguy, Moustapha Tsamarayev

Version

Date

2020

4.17.2 Documentation des fonctions

4.17.2.1 exit_game()

```
int exit_game ( )
```

Propose au joueur de quitter ou non la carte lorsqu'il vient de trouver la sortie.

Renvoie

Un *int* : 1 si le joueur décide de quitter la carte. 0 s'il décide de continuer l'aventure.

4.17.2.2 informations_quetes()

```
void informations_quetes (
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t quete )
```

Affiche les informations des quêtes (coordonnées et états)

Paramètres

map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Etat des quêtes

Renvoie

Rien

4.17.2.3 init_quete()

```
void init_quete (
            quete_t * quete,
            int quest_map[6][2],
            item_t * Tab_Items,
            int nb_items_available )
```

Initialisation d'une variable de type quete_t.

Paramètres

quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes
quest_map[6][2]	Matrice des coordonnées des quêtes
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

Rien

4.17.2.4 quete_bandits()

```
int quete_bandits (
            perso_t * player,
            quete_t * quete,
            item_t * Tab_Items,
            int nb_items_available,
            cell_t map[D][D] )
```

Accès à la quete "bandits".

Le joueur arrive sur un camp de bandits. Il a le choix entre fuir, attendre les hommes ou voler des items. Il en sort vivant ou mort.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu
map[D][D]	Matrice de la carte

Renvoie

Retourne un *int* : 0 si le jeu continue, 1 si le jeu est fini et -1 si problème dans la quête.

4.17.2.5 quete_bunker()

```
void quete_bunker (
            perso_t * player,
            quete_t * quete )
```

Accès à la quête "bunker".

Si le joueur possède le pass_card, il aura le choix de rentrer dans le bunker ou non. Sinon il fait demi-tour.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes

Renvoie

Retourne un *int* : 0 si le jeu continue, 1 si le jeu est fini et -1 si problème dans la quête.

4.17.2.6 quete_frontiere()

```
void quete_frontiere (
            perso_t * player,
            quete_t * quete )
```

Accès à la quete "frontière".

Le joueur trouve la sortie de la frontière, il a le choix de la franchir (finir le jeu) ou non. S'il a joué la quête "soin" et qu'il a aidé l'homme blessé alors ses chances de la franchir sont importantes. Mais s'il ne l'a pas fait, ses chances le sont moins. La chance est donnée par un nombre entier compris entre 0 et 100.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes

Renvoie

Retourne un *int* : 0 si le jeu continue, 1 si le jeu est fini et -1 si problème dans la quête.

4.17.2.7 quete_montagne()

```
void quete_montagne (
            perso_t * player,
            quete_t * quete )
```

Accès à la quete "montagne".

Le joueur a le choix de franchir la montagne (finir le jeu) ou non. S'il a en sa possession l'équipement de montagne alors ses chances de s'échapper sont importantes. Mais s'il ne l'a pas, il est très risqué pour lui de vouloir s'échapper. La chance est donné par un nombre entier compris entre 0 et 100.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes

Renvoie

Retourne un *int* : 0 si le jeu continue, 1 si le jeu est fini et -1 si problème dans la quête.

4.17.2.8 quetes()

```
int quetes (
            perso_t * player,
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t * quete,
            item_t * Tab_Items,
            int nb_items_available )
```

Récupère le numéro de la quête pour accéder à la quête correspondante.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Pointeur sur un objet de type quete_t correspond à l'état des quêtes
Tab_Items	Tableau contenant tous les items disponibles dans le jeu
nb_items_available	Nombre d'items disponibles dans le jeu

Renvoie

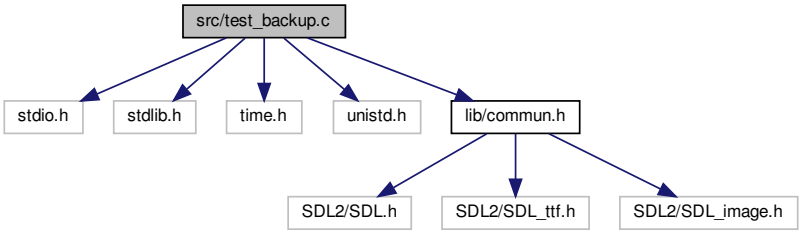
Retourne un *int* : 0 si le jeu continue, 1 si le jeu est fini et -1 si problème dans la quête.

4.18 Référence du fichier src/test_backup.c

Fichier TEST - Sauvegarder une partie.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_backup.c :



Fonctions

- void save_test (perso_t player, cell_t map[D][D], int quest_map[6][2], quete_t quete)
  Sauvegarde les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie test.
- int main ()
  Programme principal pour tester le système de sauvegarde.

4.18.1 Description détaillée

Fichier TEST - Sauvegarder une partie.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.18.2 Documentation des fonctions

4.18.2.1 main()

```
int main ( )
```

Programme principal pour tester le système de sauvegarde.

Crée aléatoirement une carte, des paramètres joueur, remplit l'inventaire du joueur et son équipement puis sauvegarde.

Menu test_backup :

1. Afficher les caractéristiques du joueur
2. Affiche l'inventaire
3. Afficher l'équipement
4. Afficher la carte
5. Afficher les détails de la carte
6. Afficher les informations sur les quêtes

Ce menu permet de comparer avec le résultat obtenu au test de chargement (test_load.c)

4.18.2.2 save_test()

```
void save_test (
            perso_t player,
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t quete )
```

Sauvegarde les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie test.

Informations sur le joueur : points de vie, points d'énergie, points d'action, position sur la carte, nombre de tours restants

Informations sur la carte : pour chaque case de la matrice map, on sauvegarde son type, sa catégorie, s'il y a un combat, si le joueur a déjà fouillé la case et si une quête y est positionnée.

Informations sur les quêtes : on sauvegarde les coordonnées de chaque quête ainsi que leurs états.

Fonction TEST

Paramètres

player	Joueur
map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Etat des quêtes

Renvoie

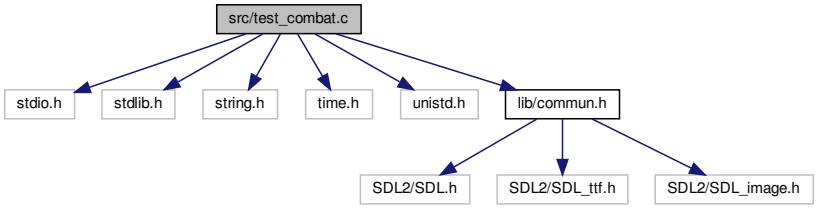
Rien

4.19 Référence du fichier src/test_combat.c

Fichier TEST - Combat.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_combat.c :



Fonctions

— int main ()
Programme principal pour tester le système de combat.

4.19.1 Description détaillée

Fichier TEST - Combat.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.19.2 Documentation des fonctions

4.19.2.1 main()

```
int main ( )
```

Programme principal pour tester le système de combat.

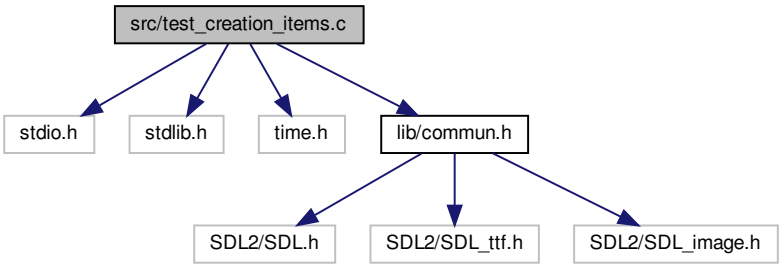
Le menu test_combat permet de tester le système de combat selon tous les cas possibles. L'utilisateur choisit s'il souhaite combattre avec ou sans armure, s'il souhaite porter une arme. Si oui, où ? combien ? lesquelles ?

4.20 Référence du fichier src/test_creation_items.c

Fichier TEST - Extraction des items du fichier externe.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_creation_items.c :



Fonctions

— int main ()
Programme principal pour tester l'extraction des items à partir du fichier externe "items.csv" (dossier data)

4.20.1 Description détaillée

Fichier TEST - Extraction des items du fichier externe.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.20.2 Documentation des fonctions

4.20.2.1 main()

```
int main ( )
```

Programme principal pour tester l'extraction des items à partir du fichier externe "items.csv" (dossier data)

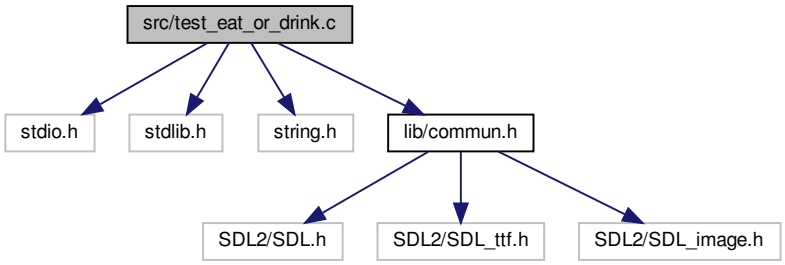
Objectif : Tester la fonction creation_tab_item.

4.21 Référence du fichier src/test_eat_or_drink.c

Fichier TEST - Manger ou boire un item.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_eat_or_drink.c :



Fonctions

— int main ()
Programme principal pour tester le calcul des points d'énergie lorsque le joueur veut manger/boire un item.

4.21.1 Description détaillée

Fichier TEST - Manger ou boire un item.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.21.2 Documentation des fonctions

4.21.2.1 main()

```
int main ( )
```

Programme principal pour tester le calcul des points d'énergie lorsque le joueur veut manger/boire un item.

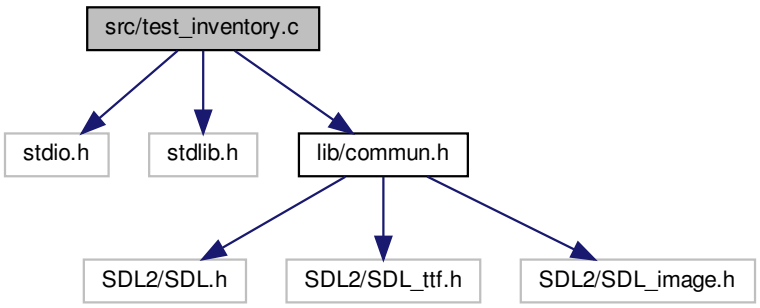
Objectif : Tester les fonctions eat_or_drink et gain_energie.

4.22 Référence du fichier src/test_inventory.c

Fichier TEST - Gestion inventaire.

```
#include <stdio.h>
#include <stdlib.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_inventory.c :



Fonctions

- void test_suivant ()
  void test_suivant()
- int main ()
  Programme principal pour tester la gestion de l'inventaire.

4.22.1 Description détaillée

Fichier TEST - Gestion inventaire.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.22.2 Documentation des fonctions

4.22.2.1 main()

```
int main ( )
```

Programme principal pour tester la gestion de l'inventaire.

Tests effectués :
- Inventaire vide
- Ajout quelques items dans l'inventaire
- Ajout impossible si 2 fois ou plus un même item dans l'inventaire
- Echange d'items
- Suppression item (sans échange)
- Echange avec gestion équipement

4.22.2.2 test_suivant()

```
void test_suivant ( )
```

void test_suivant()

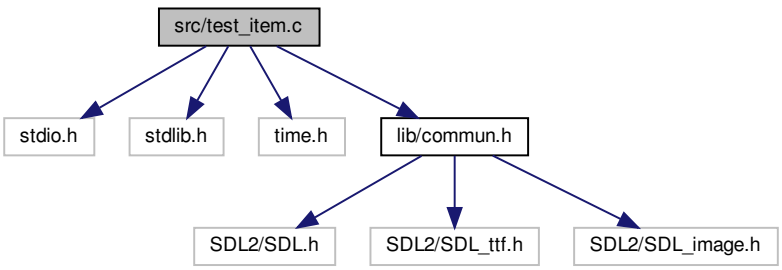
Fonction mineure pour passer au test suivant

4.23 Référence du fichier src/test_item.c

Fichier TEST - Génération aléatoire d'items sur différentes catégories d'hexagone (10 simulations)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_item.c :



Macros

— #define **NB_SIMULATIONS** 10

Fonctions

— void simulation_suivante ()
void simulation_suivante()
— int main ()
Programme principal pour tester la génération d'items sur plusieurs hexagones aléatoires.

4.23.1 Description détaillée

Fichier TEST - Génération aléatoire d'items sur différentes catégories d'hexagone (10 simulations)

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.23.2 Documentation des fonctions

4.23.2.1 main()

```
int main ( )
```

Programme principal pour tester la génération d'items sur plusieurs hexagones aléatoires.

Objectif : Tester la fonction scavenge

4.23.2.2 simulation_suivante()

```
void simulation_suivante ( )
```

void simulation_suivante()

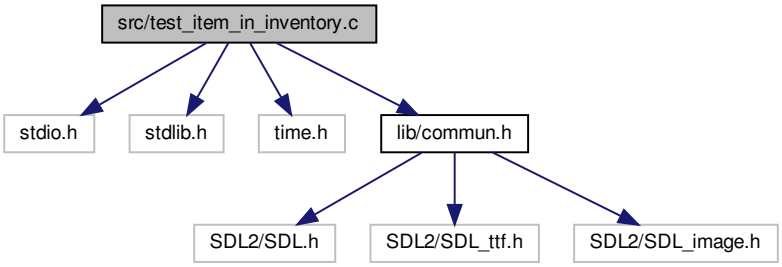
Fonction mineure pour passer à la simulation suivante

4.24 Référence du fichier src/test_item_in_inventory.c

Fichier TEST - Fonction item_in_inventory.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_item_in_inventory.c :



Fonctions

— int main ()
Programme principal pour tester si un item est présent dans l'inventaire du joueur.

4.24.1 Description détaillée

Fichier TEST - Fonction item_in_inventory.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.24.2 Documentation des fonctions

4.24.2.1 main()

```
int main ( )
```

Programme principal pour tester si un item est présent dans l'inventaire du joueur.

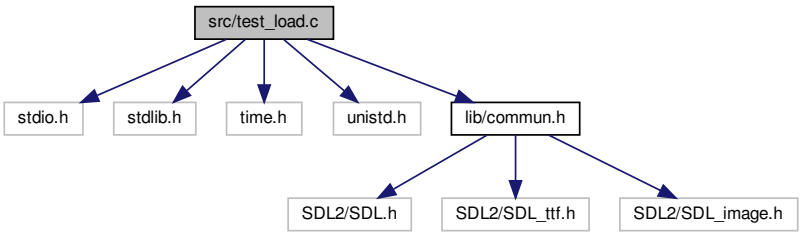
Objectif : Tester la fonction item_in_inventory

4.25 Référence du fichier src/test_load.c

Fichier TEST - Charger une partie.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_load.c :



Fonctions

- void load_test (perso_t *player, cell_t map[D][D], int quest_map[6][2], quete_t *quete)
  Charge les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie test.
- int main ()
  Programme principal pour tester le système de chargement.

4.25.1 Description détaillée

Fichier TEST - Charger une partie.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.25.2 Documentation des fonctions

4.25.2.1 load_test()

```
void load_test (
            perso_t * player,
            cell_t map[D][D],
            int quest_map[6][2],
            quete_t * quete )
```

Charge les informations sur le joueur, son inventaire, son équipement ainsi que les informations sur la carte et les quêtes d'une partie test.

Informations sur le joueur : points de vie, points d'énergie, points d'action, position sur la carte, nombre de tours restants

Informations sur la carte : pour chaque case de la matrice map, on récupère son type, sa catégorie, s'il y a un combat, si le joueur a déjà fouillé la case et si une quête y est positionnée.

Informations sur les quêtes : on récupère les coordonnées de chaque quête ainsi que leurs états.

Fonction TEST

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur
map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
quete	Pointeur sur un objet de type quete_t correspondant à l'état des quêtes

Renvoie

Rien

4.25.2.2 main()

```
int main ( )
```

Programme principal pour tester le système de chargement.

Charge la sauvegarde d'une partie test

Menu test_backup :
1. Afficher les caractéristiques du joueur
2. Affiche l'inventaire
3. Afficher l'équipement
4. Afficher la carte
5. Afficher les détails de la carte
6. Afficher les informations sur les quêtes

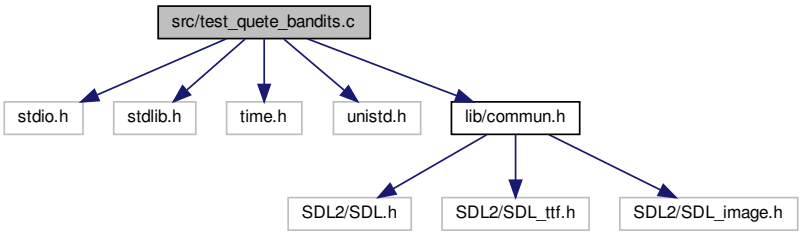
Ce menu permet de comparer avec le résultat obtenu au test de sauvegarde (test_backup.c)

4.26 Référence du fichier src/test_quete_bandits.c

Fichier TEST - Quête bandits.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_quete_bandits.c :



Fonctions

— int main ()
Programme principal pour tester l'affichage et le bon fonctionnement de la quête bandits.

4.26.1 Description détaillée

Fichier TEST - Quête bandits.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.26.2 Documentation des fonctions

4.26.2.1 main()

```
int main ( )
```

Programme principal pour tester l'affichage et le bon fonctionnement de la quête bandits.

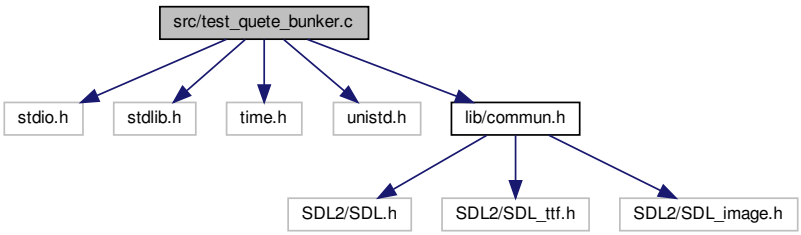
L'utilisateur choisit son équipement avant de commencer la quête.

4.27 Référence du fichier src/test_quete_bunker.c

Fichier TEST - Quête bunker.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_quete_bunker.c :



Fonctions

— int main ()
Programme principal pour tester l'affichage et le bon fonctionnement de la quête bunker.

4.27.1 Description détaillée

Fichier TEST - Quête bunker.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.27.2 Documentation des fonctions

4.27.2.1 main()

```
int main ( )
```

Programme principal pour tester l'affichage et le bon fonctionnement de la quête bunker.

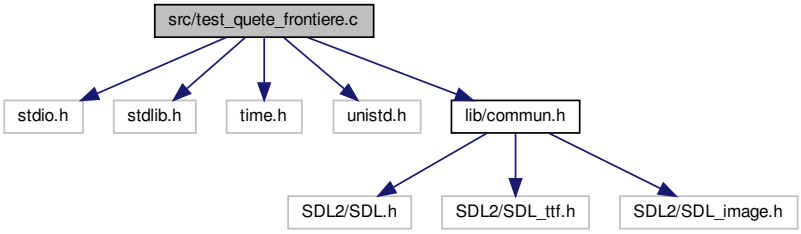
L'utilisateur choisit s'il possède un pass avant de commencer la quête.

4.28 Référence du fichier src/test_quete_frontiere.c

Fichier TEST - Quête frontière.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_quete_frontiere.c :



Fonctions

— int main ()
Programme principal pour tester l'affichage et le bon fonctionnement de la quête frontière.

4.28.1 Description détaillée

Fichier TEST - Quête frontière.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.28.2 Documentation des fonctions

4.28.2.1 main()

```
int main ( )
```

Programme principal pour tester l'affichage et le bon fonctionnement de la quête frontière.

Avant de commencer la quête, l'utilisateur choisit si :

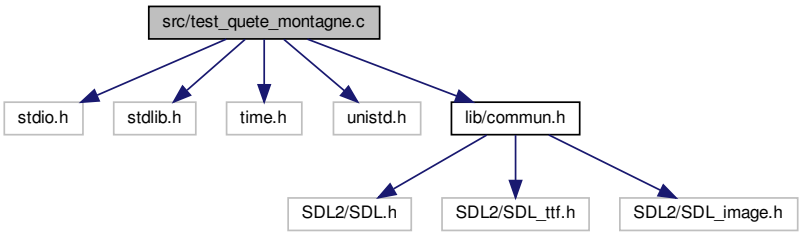
- Le joueur a aidé l'homme blessé (quête soin)
- Le joueur a voulu aider l'homme blessé (quête soin) mais sans succès
- Le joueur ne connait pas de soldat

4.29 Référence du fichier src/test_quete_montagne.c

Fichier TEST - Quête montagne.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_quete_montagne.c :



Fonctions

- int main ()
  Programme principal pour tester l'affichage et le bon fonctionnement de la quête montagne.

4.29.1 Description détaillée

Fichier TEST - Quête montagne.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.29.2 Documentation des fonctions

4.29.2.1 main()

```
int main ( )
```

Programme principal pour tester l'affichage et le bon fonctionnement de la quête montagne.

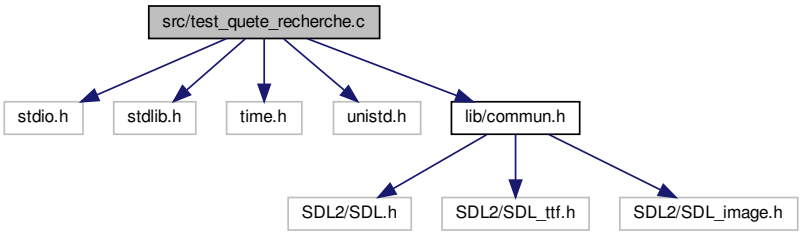
Avant de commencer la quête, l'utilisateur choisit s'il a une corde et/ou un bâton de marche dans son inventaire ou aucun des deux.

4.30 Référence du fichier src/test_quete_recherche.c

Fichier TEST - Quête recherche.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_quete_recherche.c :



Fonctions

— int main ()
Programme principal pour tester l'affichage et le bon fonctionnement de la quête recherche.

4.30.1 Description détaillée

Fichier TEST - Quête recherche.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.30.2 Documentation des fonctions

4.30.2.1 main()

```
int main ( )
```

Programme principal pour tester l'affichage et le bon fonctionnement de la quête recherche.

Avant de commencer la quête, l'utilisateur choisit :

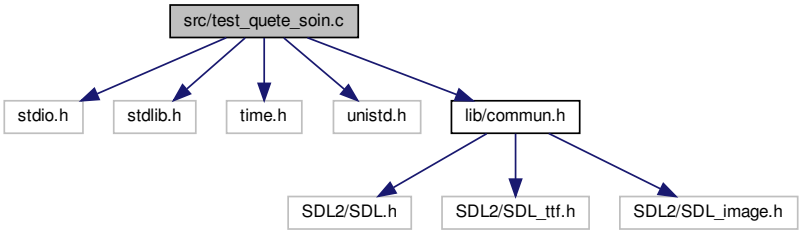
- Première fois sur la quête
- Recherche de l'item (pas sur une case urbain)
- Recherche de l'item (case urbain)
- Donner l'item (case quête)
- Donner l'item (pas case quête)

4.31 Référence du fichier src/test_quete_soin.c

Fichier TEST - Quête soin.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de test_quete_soin.c :



Fonctions

- int main ()
  Programme principal pour tester l'affichage et le bon fonctionnement de la quête soin.

4.31.1 Description détaillée

Fichier TEST - Quête soin.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.31.2 Documentation des fonctions

4.31.2.1 main()

```
int main ( )
```

Programme principal pour tester l'affichage et le bon fonctionnement de la quête soin.

Avant de commencer la quête, l'utilisateur choisit :

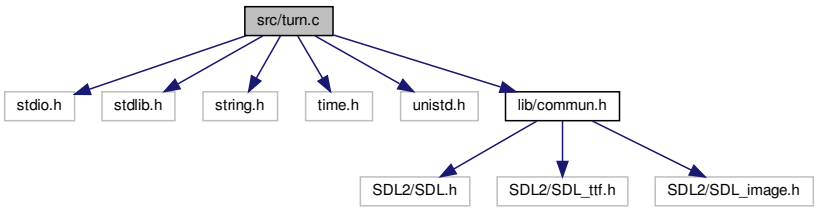
- Inventaire vide
- Inventaire : inclus 1 item food
- Inventaire : inclus 3 items food dans l'inventaire
- Inventaire : inclus 1 medical kit
- Inventaire : inclus 1 medical kit + 1 item food
- Inventaire plein (inclus 1 medical kit + items food)

4.32 Référence du fichier src/turn.c

Fonctions relatives à un tour du jeu.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de turn.c :



Fonctions

— void next_turn (perso_t *player)
Calcule le nombre de points d'action récupérés à partir de la valeur des points d'énergie du joueur puis passe au tour suivant.
— void rest_and_heal (perso_t *player)
Permet au joueur de se reposer et récupérer des points de vie et points d'énergie (proportionnellement au nombre de points d'action)

4.32.1 Description détaillée

Fonctions relatives à un tour du jeu.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.32.2 Documentation des fonctions

4.32.2.1 next_turn()

```
void next_turn (
            perso_t * player )
```

Calcule le nombre de points d'action récupérés à partir de la valeur des points d'énergie du joueur puis passe au tour suivant.

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

Rien

4.32.2.2 rest_and_heal()

```
void rest_and_heal (
            perso_t * player )
```

Permet au joueur de se reposer et récupérer des points de vie et points d'énergie (proportionnellement au nombre de points d'action)

Paramètres

player	Pointeur sur un objet de type perso_t correspondant au joueur

Renvoie

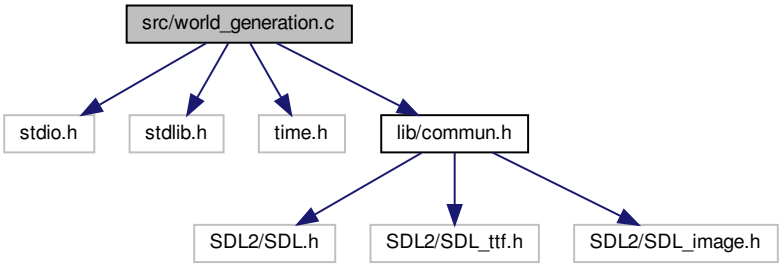
Rien

4.33 Référence du fichier src/world_generation.c

Génération de la carte.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "lib/commun.h"
```

Graphe des dépendances par inclusion de world_generation.c :



Fonctions

- void afficher_type_categ_hexa (cell_t map[D][D], int l, int c)
  Affiche le type et la catégorie de l'hexagone de la carte dont les coordonnées sont passées en paramètres.
- void informations_map (cell_t map[D][D])
  Affiche les informations de la carte.
- void init_border (cell_t map[D][D])
  Initialise les contours de la map.
- void topup (cell_t map[D][D], int quest_map[6][2])
  Crée un nombre déterminé d'hexagones indispensables pour la carte (comme camp militaire/bandit par exemple) sur des coordonnées aléatoires.
- int spawntype (int l, int c, cell_t map[D][D])
  Compte le nombre d'hexagones-voisins similaires et selon ce nombre, après des calculs de probabilité, renvoie le type d'hexagone qui sera créé sur les coordonnées passées en paramètre.
- int categ_switch (int input)
  Donne la catégorie de l'hexagone passé en paramètre.
- void nextgen (cell_t map[D][D])
  Parcours la matrice de la carte passée en paramètre et fait appel à spawntype pour calculer le type d'hexagone qui va apparaître dans la cellule courante.
- int coordonnees_valides (int l, int c)

Vérifie si les coordonnées sont valides.
- void portable_switch (int i, int j, cell_t map[D][D])
  Affiche le code de la cellule.
- void display_TEXT (int l, int c, cell_t map[D][D])
  Affiche la map en version texte avec la légende.
- void init_base (cell_t map[D][D])
  Initialise la base de la carte.
- void count (const cell_t map[D][D])
  Compte le nombre d'occurence de chaque type de cellule (outil de test)
- void encounter_init (cell_t map[D][D])
  Initialise les positions des combats sur la carte.
- void quest_init (cell_t map[D][D], int quest_map[6][2])
  Initialise les quêtes (quest_id) sur la carte.
- void map_init (cell_t map[D][D], int quest_map[6][2])
  Initialise la carte au début de chaque partie.

4.33.1 Description détaillée

Génération de la carte.

Auteur

Mathilde Mottay, Anaïs Mottier, Clément Mainguy, Moustapha Tsamarayev

Version

1.0

Date

2020

4.33.2 Documentation des fonctions

4.33.2.1 afficher_type_categ_hexa()

```
void afficher_type_categ_hexa (
            cell_t map[D][D],
            int l,
            int c )
```

Affiche le type et la catégorie de l'hexagone de la carte dont les coordonnées sont passées en paramètres.

Paramètres

map[D][D]	Matrice de la carte
l	Coordonnée ligne de l'hexagone qu'on souhaite afficher
c	Coordonnée colonne de l'hexagone qu'on souhaite afficher

Renvoie

Rien

4.33.2.2 categ_switch()

```
int categ_switch (
            int input )
```

Donne la catégorie de l'hexagone passé en paramètre.

Paramètres

input	Type d'hexagone

Renvoie

Retourne un *int* correspondant à la catégorie de l'hexagone

4.33.2.3 coordonnees_valides()

```
int coordonnees_valides (
            int l,
            int c )
```

Vérifie si les coordonnées sont valides.

Paramètres

l	Coordonnée ligne
c	Coordonnée colonne

Renvoie

Retourne un *int* : 1 si les coordonnées sont valides, 0 si non

4.33.2.4 count()

```
void count (
            const cell_t map[D][D] )
```

Compte le nombre d'occurence de chaque type de cellule (outil de test)

Paramètres

map[D][D]	Matrice de la carte

Renvoie

Rien

4.33.2.5 display_TEXT()

```
void display_TEXT (
            int l,
            int c,
            cell_t map[D][D] )
```

Affiche la map en version texte avec la légende.

Paramètres

l	Coordonnée ligne
c	Coordonnée colonne
map[D][D]	Matrice de la carte

Renvoie

Rien

4.33.2.6 encounter_init()

```
void encounter_init (
            cell_t map[D][D] )
```

Initialise les positions des combats sur la carte.

Paramètres

map[D][D]	Matrice de la carte

Renvoie

Rien

4.33.2.7 informations_map()

```
void informations_map (
            cell_t map[D][D] )
```

Affiche les informations de la carte.

Paramètres

map[D][D]	Matrice de la carte

Renvoie

Rien

4.33.2.8 init_base()

```
void init_base (
            cell_t map[D][D] )
```

Initialise la base de la carte.

La carte est initialisée par des hexagones de type prairie (catégorie nature), sans combat, sans quête et sans hexagones fouillés.

Paramètres

map[D][D]	Matrice de la carte

Renvoie

Rien

4.33.2.9 init_border()

```
void init_border (
            cell_t map[D][D] )
```

Initialise les contours de la map.

Paramètres

map[D][D]	Matrice de la carte

Renvoie

Rien

4.33.2.10 map_init()

```
void map_init (
             cell_t map[D][D],
             int quest_map[6][2] )
```

Initialise la carte au début de chaque partie.

Paramètres

map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes
player	Joueur

Renvoie

Rien

4.33.2.11 nextgen()

```
void nextgen (
             cell_t map[D][D] )
```

Parcours la matrice de la carte passée en paramètre et fait appel à spawntype pour calculer le type d'hexagone qui va apparaître dans la cellule courante.

Paramètres

map[D][D]	Matrice de la carte

Renvoie

Rien

4.33.2.12 portable_switch()

```
void portable_switch (
             int i,
             int j,
             cell_t map[D][D] )
```

Affiche le code de la cellule.

Cette fonction a été créée pour éviter de refaire le switch dans différentes fonctions d'affichage.

Paramètres

i	Coordonnée ligne
j	Coordonnée colonne
map[D][D]	Matrice de la carte

Renvoie

Rien

4.33.2.13 quest_init()

```
void quest_init (
            cell_t map[D][D],
            int quest_map[6][2] )
```

Initialise les quêtes (quest_id) sur la carte.

Les quêtes sont placées aléatoirement sur la carte.

Paramètres

map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes

Renvoie

Rien

4.33.2.14 spawntype()

```
int spawntype (
            int l,
            int c,
            cell_t map[D][D] )
```

Compte le nombre d'hexagones-voisins similaires et selon ce nombre, après des calculs de probabilité, renvoie le type d'hexagone qui sera créé sur les coordonnées passées en paramètre.

Paramètres

l	Coordonnée ligne
c	Coordonnée colonne
map[D][D]	Matrice de la carte

Renvoie

Retourne un *int* : le type de l'hexagone qui va être créé

4.33.2.15 topup()

```
void topup (
            cell_t map[D][D],
            int quest_map[6][2] )
```

Crée un nombre déterminé d'hexagones indispensables pour la carte (comme camp militaire/bandit par exemple) sur des coordonnées aléatoires.

Paramètres

map[D][D]	Matrice de la carte
quest_map[6][2]	Matrice des coordonnées des quêtes

Renvoie

Rien