# Tactics Arena

Généré par Doxygen 1.8.9.1

Vendredi 18 Décembre 2015 10 :22 :01

# Table des matières

# Chapitre 1

# Page principale

Tactics Arena est un jeu de stratégie au tour par tour se jouant sur un plateau. Chaque joueur déplace ses unités respectant des règles de déplacement propre à chaque unité, le but du jeu est d'éliminer ou paralyser toutes les unités ennemies.

### Règles

Les conditions d'égalité sont :

- **Toutes les unités actives sont détruites ou immobilisée**
- **Les joueurs passent leurs tours 3 fois d'affilé**
- **Il n'y a eu aucune rencontre entre les deux unités ennemies depuis les 30 derniers tours.**

Le joueur qui commence est défini aléatoirement par l'ordinateur au début de chaque partie.

Il n'est possible de déplacer qu'une seule unité par tour soit pour :

1. **Se déplacer**
2. **Et / ou attaquer**
3. **Et / ou changer de direction**

Le changement de direction impose d'être fait en dernier.

Chaque tour dure 1 à 2 minutes, dépendant du nombre d'unité restante. Une fois ce temps expiré :

- **Le tour se termine automatiquement si au moins une commande a été effectué**
- **Si aucune commande n'a été effectuée alors abandon automatique**

**Important : toute action est irréversible.**

### Sommaire

**Tactics Arena**

1. **Règles**
2. **Changelog**
   - **Version 0.1**
3. **A faire**
4. **Instructions de compilation**
5. **Utilisation**
6. **Documentation**

## A faire

- **[x] Analyse préliminaire**
- **[ ] Compte rendu mécaniques du jeu**
- **[x] Analyse conceptuelle**
- **[ ] Implémentation**
  - **[x] Grille**
  - **[x] Menu**
  - **[x] Unités**
  - **[x] Statistiques**
  - **[ ] Mouvements**
  - **[ ] Gestion des tours**
  - **[ ] Gestion du temps**
  - **[ ] Capacités spéciales**
  - **[ ] Etat unités**
- **[ ] Test unitaires**
- **[ ] Test d'intégration**
- **[ ] Ajout de fonctionnalités**
  - **[ ] Intégration IA**
  - **[ ] GUI**

## Changelog

### Version 0.1

- Mise en place sommaire des TAD

## Instructions de compilation

## Utilisation

## Documentation

# Chapitre 2

# Index des classes

## 2.1 Liste des classes

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

# Chapitre 3

# Index des fichiers

## 3.1 Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

# Chapitre 4

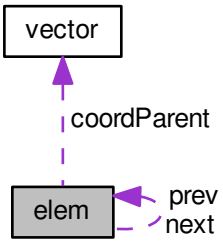
# Documentation des classes

## 4.1 Référence de la structure elem

Représente une case de la grille.

`#include <pathList.c>`

Graphe de collaboration de elem :




### Attributs publics

- vector coordParent
  *Coordonnées de la case.*
- int F
  *Poids de la case.*
- struct elem * prev
  *Elément précédent.*
- struct elem * next
  *Elément suivant.*

### 4.1.1 Description détaillée

Représente une case de la grille.

Définition à la ligne 16 du fichier pathList.c.

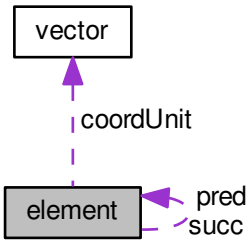
La documentation de cette structure a été générée à partir du fichier suivant :

- src/game/pathList.c

## 4.2 Référence de la structure element

Représente un élément de la liste.

Graphe de collaboration de element :




### Attributs publics

- vector coordUnit
  *Coordonnées de l'unité*
- struct element * pred
  *Élément précédent.*
- struct element * succ
  *Élément suivant.*

### 4.2.1 Description détaillée

Représente un élément de la liste.

Définition à la ligne 19 du fichier listes.c.

La documentation de cette structure a été générée à partir du fichier suivant :
- src/game/listes.c

## 4.3 Référence de la structure targetStat

Représente les informations liées aux cibles.

`#include <engine.h>`

### Attributs publics

- short vertRange
  *Portée verticale.*
- short horizRange
  *Portée horizontale.*
- short ringSize
  *Taille de l'anneau.*
- short line
  *Ciblage en ligne.*

### 4.3.1 Description détaillée

Représente les informations liées aux cibles.

Définition à la ligne 54 du fichier engine.h.

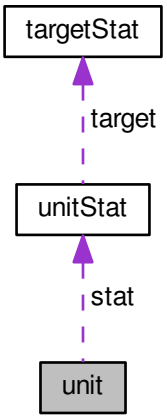
La documentation de cette structure a été générée à partir du fichier suivant :
— include/game/engine.h

## 4.4 Référence de la structure unit

Représente une unité

`#include <engine.h>`

Graphe de collaboration de unit :




### Attributs publics

— unitName name
*Nom de l'unité*
— unitStat stat
*Statistiques de l'unité*
— unitEffect effect [NB_MAX_EFFECT]
*Effets sur l'unité*
— cardinal direct
*Direction de l'unité*
— int noPlayer
*Propriétaire de l'unité*
— int unitColor
*Couleur de l'unité*

### 4.4.1 Description détaillée

Représente une unité

Définition à la ligne 88 du fichier engine.h.

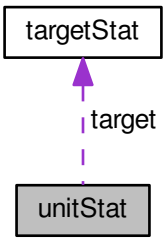
La documentation de cette structure a été générée à partir du fichier suivant :
— include/game/engine.h

## 4.5 Référence de la structure unitStat

Représente les statistiques d'une unité

`#include <engine.h>`

Graphe de collaboration de unitStat :




### Attributs publics

— int HP
*Points de vie.*
— int POWER
*Puissance.*
— float ARMOR
*Armure.*
— int RECOVERY
*Repos.*
— float BLOCK [3]
*Blocage.*
— targetStat target
*Ciblage.*
— int MOVE_RANGE
*Portée mouvement.*

### 4.5.1 Description détaillée

Représente les statistiques d'une unité

Définition à la ligne 65 du fichier engine.h.

La documentation de cette structure a été générée à partir du fichier suivant :
— include/game/engine.h

## 4.6 Référence de la structure vector

Représente les coordonnées d'un vecteur.

```
#include <engine.h>
```

### Attributs publics

— int x
  *Position x.*
— int y
  *Position y.*

### 4.6.1 Description détaillée

Représente les coordonnées d'un vecteur.

Définition à la ligne 79 du fichier engine.h.

La documentation de cette structure a été générée à partir du fichier suivant :
— include/game/engine.h

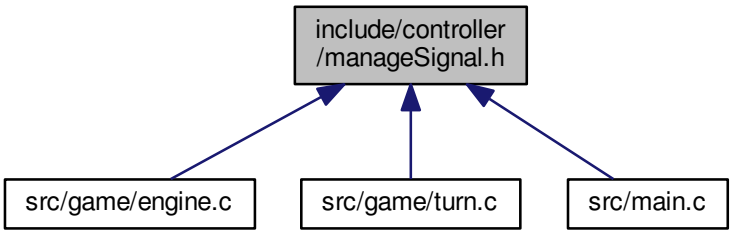
# Chapitre 5

# Documentation des fichiers

## 5.1 Référence du fichier include/controller/manageSignal.h

En-tête gestion des signaux.

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




### Fonctions

- void checkSignal ()
  *Vérifie les signaux du programme.*
- void timeDown ()
  *Message lors du temps écoulé*
- void freeAll ()
  *Libère toute la mémoire.*

### 5.1.1 Description détaillée

En-tête gestion des signaux.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.1.2 Documentation des fonctions

#### 5.1.2.1 void checkSignal ( )

Vérifie les signaux du programme.

Vérifie les signaux du programme.

Définition à la ligne 77 du fichier manageSignal.c.

#### 5.1.2.2 void freeAll ( )

Libère toute la mémoire.

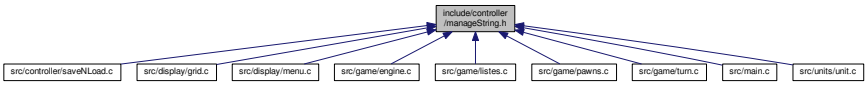
Libère toute la mémoire.

Définition à la ligne 21 du fichier manageSignal.c.

## 5.2 Référence du fichier include/controller/manageString.h

En-tête gestion des chaînes de caractères.

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




### Fonctions

- char * get2Char (char name[ ])
  *Récupère 2 caractères du nom de l'unité*
- char * getNameUnit (unitName unit)
  *Récupère le nom de l'unité*
- void printNameUnit (unitName unit)
  *Affiche le nom de l'unité*
- void getCoordS (char coordString[ ], vector *coordUnit)
  *Récupère les coordonnées d'une chaîne de caractères.*
- bool correctCoord (char *coordString)
  *Vérifie l'authenticité des coordonnées d'une chaîne de caractères.*
- bool rightSide (char *coordString)
  *Vérifie que l'unité est dans le bon camp.*
- int readS (char *string)
  *Lecture sécurisée d'une chaîne.*
- long readLong ()
  *Lecture sécurisée d'un long.*
- double readDouble ()
  *Lecture sécurisée d'un double.*
- void clearBuffer ()
  *Vide la mémoire tampon.*
- char * getDirectionUnit (cardinal direct)
  *Récupère le nom de la direction.*
- char * getNameEffect (unitEffect effect)
  *Récupère le nom de l'effet.*

### 5.2.1 Description détaillée

En-tête gestion des chaînes de caractères.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.2.2 Documentation des fonctions

#### 5.2.2.1 void clearBuffer ( )

Vide la mémoire tampon.

Vide la mémoire tampon.

Définition à la ligne 253 du fichier manageString.c.

#### 5.2.2.2 char* get2Char ( char *name[ ]* )

Récupère 2 caractères du nom de l'unité

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité |

**Renvoie**

Retourne 2 caractères liés au nom de l'unité

Définition à la ligne 47 du fichier manageString.c.

#### 5.2.2.3 void getCoordS ( char *coordString[ ],* vector * *coordUnit* )

Récupère les coordonnées d'une chaîne de caractères.

Récupère les coordonnées d'une chaîne de caractères.

**Paramètres**

| | |
|---|---|
| *coordString* | Coordonnées saisie par l'utilisateur |
| *coordUnit* | Coordonnées de l'unité récupérées de la saisie utilisateur |

Définition à la ligne 19 du fichier manageString.c.

#### 5.2.2.4 char* getDirectionUnit ( cardinal *direct* )

Récupère le nom de la direction.

Récupère le nom de la direction.

**Paramètres**

| | |
|---|---|
| *direct* | Direction de l'unité |

**Renvoie**

Retourne La direction sous forme de chaîne

Définition à la ligne 235 du fichier manageString.c.

#### 5.2.2.5 char* getNameEffect ( unitEffect *effect* )

Récupère le nom de l'effet.

Récupère le nom de l'effet.

**Paramètres**

| | |
|---|---|
| *effect* | Effet provenant de la liste énumérée |

**Renvoie**

Nom de l'effet sous forme de chaîne de caractère

Définition à la ligne 220 du fichier manageString.c.

#### 5.2.2.6 char* getNameUnit ( unitName *name* )

Récupère le nom de l'unité

Récupère le nom de l'unité

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité provenant de la liste énumérée |

**Renvoie**

Nom de l'unité sous forme de chaîne

Définition à la ligne 189 du fichier manageString.c.

#### 5.2.2.7 void printNameUnit ( unitName *unit* )

Affiche le nom de l'unité

**Paramètres**

| | |
|---|---|
| *unit* | Nom de l'unité provenant de la liste énumérée |

Définition à la ligne 246 du fichier manageString.c.

#### 5.2.2.8 double readDouble ( )

Lecture sécurisée d'un double.

Lecture sécurisée d'un double.

**Renvoie**

Retourne le double saisie ou 0.0 en cas d'erreur

Définition à la ligne 305 du fichier manageString.c.

#### 5.2.2.9 long readLong ( )

Lecture sécurisée d'un long.

Lecture sécurisée d'un long.

**Renvoie**

Retourne le long saisie ou 0 en cas d'erreur

Définition à la ligne 291 du fichier manageString.c.

#### 5.2.2.10 int readS ( char * *string* )

Lecture sécurisée d'une chaîne.

Lecture sécurisée d'une chaîne.

**Paramètres**

| | |
|---|---|
| *string* | Chaîne de caractère à vérifier |

**Renvoie**

Retourne 1 si chaîne correcte

Définition à la ligne 265 du fichier manageString.c.

#### 5.2.2.11 bool rightSide ( char * *coordString* )

Vérifie que l'unité est dans le bon camp.

Vérifie que l'unité est dans le bon camp.

**Paramètres**

| | |
|---|---|
| *coordString* | Coordonnées sous forme de chaîne |

**Renvoie**

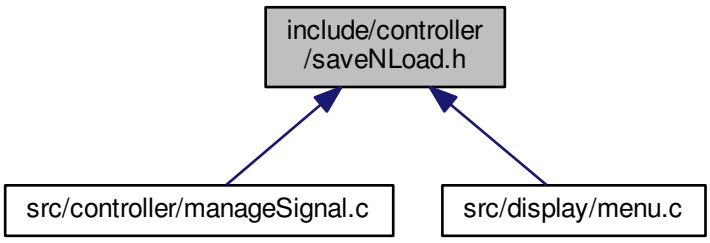
Retourne vrai si du bon côté

Définition à la ligne 165 du fichier manageString.c.

## 5.3 Référence du fichier include/controller/saveNLoad.h

En-tête gestion de la sauvegarde et chargement.

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




### Fonctions

- void save ()
  *Sauvegarde la partie.*
- void load ()
  *Charge une partie.*

### 5.3.1 Description détaillée

En-tête gestion de la sauvegarde et chargement.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.3.2 Documentation des fonctions

#### 5.3.2.1 void load ( )

Charge une partie.

Charge une partie.

Définition à la ligne 201 du fichier saveNLoad.c.

#### 5.3.2.2 void save ( )

Sauvegarde la partie.

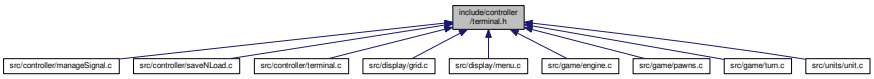
Sauvegarde la partie.

Définition à la ligne 158 du fichier saveNLoad.c.

## 5.4 Référence du fichier include/controller/terminal.h

En-tête gestion du terminal.

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




### Énumérations

- enum terminal {
  **black**, **red**, **green**, **yellow**,
  **blue**, **magenta**, **cyan**, **white** }
  *terminal.h*
- enum screen { **reinit** = 0, **blink** = 5, **invertColor** = 7 }
  *terminal.h*

### Fonctions

- void color (int color, char type[ ])
  *Met en couleurs le texte ou l'écran.*
- void fontColor (int color)
  *Met en couleurs la police.*
- void clearScreen ()
  *Efface l'écran.*
- void reinitColor ()
  *Réinitialise les couleurs.*

### 5.4.1 Description détaillée

En-tête gestion du terminal.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.4.2 Documentation des fonctions

#### 5.4.2.1 void color ( int *color,* char *type[ ]* )

Met en couleurs le texte ou l'écran.

Met en couleurs le texte ou l'écran.

**Paramètres**

| | |
|---|---|
| *color* | Couleur à utiliser |
| *type* | Texte à changer de couleur ou arrière plan |

Définition à la ligne 51 du fichier terminal.c.

#### 5.4.2.2 void fontColor ( int *color* )

Met en couleurs la police.

Met en couleurs la police.

**Paramètres**

| | |
|---|---|
| *color* | Couleur à utiliser |

Définition à la ligne 76 du fichier terminal.c.

#### 5.4.2.3 void reinitColor ( )

Réinitialise les couleurs.

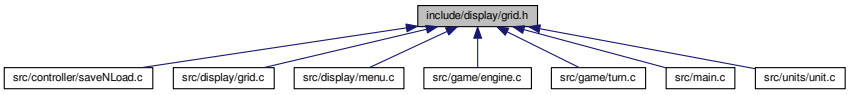
Réinitialise les couleurs.

Définition à la ligne 67 du fichier terminal.c.

## 5.5 Référence du fichier include/display/grid.h

En-tête gestion de la grille.

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




### Macros

— #define RB "\e(0\x6a\e(B"
*188 Right Bottom corner*
— #define RT "\e(0\x6b\e(B"
*187 Right Top corner*
— #define LT "\e(0\x6c\e(B"
*201 Left Top cornet*
— #define LB "\e(0\x6d\e(B"
*200 Left Bottom corner*
— #define VL "\e(0\x78\e(B"

*186 Vertical Line*

— #define HL "\e(0\x71\e(B"
*205 Horizontal Line*

### Fonctions

— void gridDisp ()
*Affiche la grille.*

### 5.5.1 Description détaillée

En-tête gestion de la grille.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.5.2 Documentation des fonctions

#### 5.5.2.1 void gridDisp ( )

Affiche la grille.

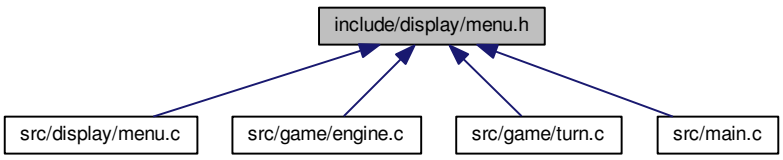
Affiche la grille.

Définition à la ligne 78 du fichier grid.c.

## 5.6 Référence du fichier include/display/menu.h

En-tête gestion des menus.

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




### Fonctions

— void mainMenu ()
*Menu principal.*
— void gameMenu ()

*Menu de jeu.*
- void unitMenu (int choice)
  *Menu de sélection de l'unité*
- void unitList ()
  *Listes des unités du jeu.*
- void mainHelp ()
  *Menu d'aide principal.*
- void helpUnit ()
  *Menu d'aide des unités.*
- void dispDirection ()
  *Affiche la liste des directions.*

### 5.6.1 Description détaillée

En-tête gestion des menus.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.6.2 Documentation des fonctions

#### 5.6.2.1 void dispDirection ( )

Affiche la liste des directions.

Affiche la liste des directions.

Définition à la ligne 458 du fichier menu.c.

#### 5.6.2.2 void gameMenu ( )

Menu de jeu.

Menu de jeu.

Définition à la ligne 470 du fichier menu.c.

#### 5.6.2.3 void helpUnit ( )

Menu d'aide des unités.

Menu d'aide des unités.

Définition à la ligne 222 du fichier menu.c.

#### 5.6.2.4 void mainMenu ( )

Menu principal.

Menu principal.

Définition à la ligne 24 du fichier menu.c.

#### 5.6.2.5 void unitList ( )

Listes des unités du jeu.

Listes des unités du jeu.

Définition à la ligne 593 du fichier menu.c.

#### 5.6.2.6 void unitMenu ( int *choice* )

Menu de sélection de l'unité

**Paramètres**

| | |
|---|---|
| *choice* | Choix de l'action pour l'unité |

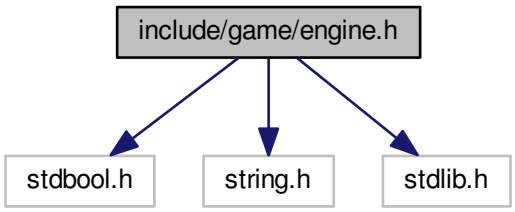
Définition à la ligne 517 du fichier menu.c.

## 5.7 Référence du fichier include/game/engine.h

En-tête moteur de jeu.

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

Graphe des dépendances par inclusion de engine.h :




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

### Classes

- struct targetStat
  *Représente les informations liées aux cibles.*
- struct unitStat
  *Représente les statistiques d'une unité*
- struct vector
  *Représente les coordonnées d'un vecteur.*
- struct unit
  *Représente une unité*

## Macros

- #define N 11
  *Taille de la grille.*
- #define NB_LISTS_ENGINE 2
  *Nombre de liste additionnelles nécessaires pour le jeu.*
- #define NB_PLAYERS 2
  *Nombre de joueurs.*
- #define NB_LINES 3
  *Limite du camp du joueur.*
- #define NB_UNITS 21
  *Nombre d'unités dans le jeu.*
- #define NB_MAX_EFFECT 6
  *Nombre total de status différent.*
- #define MANDATORY_STATS 12
  *Nombre de stats obligatoire.*
- #define FIRST_PLAYER 0
  *Définis la valeur du premier joueur.*
- #define NB_MAX_KN 3
  *Nombre max de Guerrier par joueur.*
- #define NB_MAX_SC 2
  *Nombre max de Recrue par joueur.*
- #define NB_MAX_SG 2
  *Nombre max de Golem de pierre par joueur.*
- #define NB_MAX_LT 1
  *Nombre max de Lightning totem par joueur.*
- #define NB_MAX_DR 2
  *Nombre max de Dragon par joueur.*
- #define NB_MAX_FU 2
  *Nombre max de Furgon par joueur.*
- #define NB_MAX_UNIT 5
  *Nombre max d'unité par joueur.*
- #define NB_MAX_DECOR 7
  *Nombre max de décor.*

## Énumérations

- enum cardinal { **north**, **east**, **south**, **west** }
  *engine.h*
- enum unitName {
  **empty**, **decors**, **knight**, **scout**,
  **assassin**, **cleric**, **pyromancer**, **enchantress**,
  **dragonborn**, **darkWitch**, **lightningTotem**, **barrierTtotem**,
  **mudGolem**, **golemAmbusher**, **frostGolem**, **stoneGolem**,
  **dragonTyrant**, **berserker**, **beastRider**, **poisonWisp**,
  **furgon** }
  *engine.h*
- enum unitEffect {
  **none**, **POWER_BONUS**, **ARMOR_BONUS**, **BARRIER**,
  **POISON**, **PARALYSE**, **FOCUS** }
  *engine.h*

## Fonctions

- bool isSurrounded (vector currentUnit)
  *Vérifie si une unité est entourée.*
- void gameInit ()
  *Initialise le jeu.*
- bool selectUnit (vector *coordUnit)

*Sélectionne une unité*
- void setTarget (unitName name, vector coordUnit, int colorDisp)
  *Définis les cibles.*
- void launchAttack (vector coordSource, vector coordTarget)
  *Lance une attaque.*
- void movable (int colorDisp)
  *Fait la liste des unités déplaçables.*
- void attackable (int colorDisp)
  *Fait la liste des unités pouvant attaquer.*
- void tileWalkable (vector coordUnit, int colorDisp)
  *Fait la liste des cases atteignables par l'unité*
- bool possiblePath (vector coordUnit)
  *Vérifie qu'un chemin est possible.*
- bool pathFind (vector, vector)
  *Trouve une chemin vers la destination.*
- void startGame ()
  *Débute la partie.*
- void lineOfSight (vector, vector)
  *Ligne de vue de l'archer.*

## Variables

- unit grid [N][N]
  *Représentation d'une grille d'unité globale.*
- int noPlayer
  *Représentation du joueur.*

### 5.7.1 Description détaillée

En-tête moteur de jeu.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.7.2 Documentation des fonctions

#### 5.7.2.1 void attackable ( int *colorDisp* )

Fait la liste des unités pouvant attaquer.

**Paramètres**

| | |
|---|---|
| *colorDisp* | Couleur d'affichage |

Définition à la ligne 164 du fichier engine.c.

#### 5.7.2.2 void gameInit ( )

Initialise le jeu.

Initialise le jeu.

Définition à la ligne 750 du fichier engine.c.

#### 5.7.2.3 bool isSurrounded ( vector *currentUnit* )

Vérifie si une unité est entourée.

Vérifie si une unité est entourée.

**Paramètres**

| | |
|---|---|
| *currentUnit* | Unité courante |

**Renvoie**

Retourne vrai si unité entourée

Définition à la ligne 215 du fichier engine.c.

#### 5.7.2.4 void launchAttack ( vector *coordSource,* vector *coordTarget* )

Lance une attaque.

Lance une attaque.

**Paramètres**

| | |
|---|---|
| *coordSource* | Nom de l'unité source |
| *coordTarget* | Coordonnées de la cible |

Définition à la ligne 336 du fichier engine.c.

#### 5.7.2.5 void lineOfSight ( vector *coordSource,* vector *coordTarget* )

Ligne de vue de l'archer.

**Paramètres**

| | |
|---|---|
| *coordSource* | Coordonnées de l'unité attaquante |
| *coordTarget* | Coordonnées de la cible |

Définition à la ligne 117 du fichier engine.c.

#### 5.7.2.6 void movable ( int *colorDisp* )

Fait la liste des unités déplaçables.

**Paramètres**

| | |
|---|---|
| *colorDisp* | Couleur d'affichage |

Définition à la ligne 141 du fichier engine.c.

#### 5.7.2.7 bool pathFind ( vector *coordUnit,* vector *coordTarget* )

Trouve une chemin vers la destination.

Trouve une chemin vers la destination.

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordonnées de l'unité |
| *coordTarget* | Coordonnées de la cible |

**Renvoie**

Retourne vrai si chemin trouvé

Définition à la ligne 60 du fichier engine.c.

#### 5.7.2.8 bool possiblePath ( vector *coordUnit* )

Vérifie qu'un chemin est possible.

Vérifie qu'un chemin est possible.

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordonnées de l'unité |

**Renvoie**

Retourne Vrai si chemin possible vers une quelconque position

Définition à la ligne 31 du fichier engine.c.

#### 5.7.2.9 bool selectUnit ( vector * *coordUnit* )

Sélectionne une unité

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordoonnées de l'unité |

**Renvoie**

Retourne vrai si unité bien sélectionnée

Définition à la ligne 416 du fichier engine.c.

#### 5.7.2.10 void setTarget ( unitName *name,* vector *coordUnit,* int *colorDisp* )

Définis les cibles.

Définis les cibles.

**Paramètres**

| | |
|---|---|
| *name* | Nom du pion |
| *coordUnit* | Coordonnées de l'unité |
| *colorDisp* | Couleur d'affichage |

Définition à la ligne 252 du fichier engine.c.

#### 5.7.2.11 void tileWalkable ( vector *coordUnit,* int *colorDisp* )

Fait la liste des cases atteignables par l'unité

Fait la liste des cases atteignables par l'unité

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordonnées de l'unité |
| *colorDisp* | Couleur d'affichage |

Définition à la ligne 188 du fichier engine.c.

### 5.7.3 Documentation des variables

#### 5.7.3.1 unit grid[N][N]

Représentation d'une grille d'unité globale.

Représentation d'une grille d'unité globale.

Définition à la ligne 23 du fichier engine.c.

#### 5.7.3.2 int noPlayer

Représentation du joueur.

Représentation du joueur.

Définition à la ligne 24 du fichier engine.c.

## 5.8 Référence du fichier include/game/listes.h

En-tête listes de vecteurs.

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


### Macros

— #define MAX_JOUEUR 3
*Nombre max de listes.*

### Fonctions

— void init_liste (int)
*Initialise une liste.*
— void initLists ()
*Initialise les listes.*
— int liste_vide (int)
*Vérifie si une liste est vide.*
— int hors_liste (int)
*Vérifie si l'élément courant est hors de la liste.*
— void en_tete (int)
*Se met en tête de la liste.*
— void en_queue (int)
*Se met en queue de la liste.*
— void precedent (int)

*Se positionne sur l'élément précédent.*
- void suivant (int)
  *Se positionne sur l'élément suivant.*
- void valeur_elt (int, vector *v)
  *Récupère la valeur de l'élément.*
- void modif_elt (int, vector v)
  *Modifie la valeur de l'élément.*
- void oter_elt (int)
  *Supprime l'élément.*
- void ajout_droit (int, vector v)
  *Ajoute à droite l'élément.*
- void ajout_gauche (int, vector v)
  *Ajoute à gauche l'élément.*
- void dumpList (short nbList)
  *Vide une liste.*
- void dumpAllLists ()
  *Vide les listes.*
- void addUnit (vector coordUnit)
  *Ajoute une unité*
- void destroyUnit (vector coordUnit)
  *Détruit une unité*
- void printList (short numList)
  *Affiche la liste.*
- int countUnits ()
  *Compte le nombre d'unités.*
- void addTarget (unitName name, vector coordUnit)
  *Ajoute une cible.*
- bool searchTarget (int numList, vector coordTarget)
  *Cherche une cible.*

## Variables

- int targetList
  *Cibles potentielles.*

### 5.8.1 Description détaillée

En-tête listes de vecteurs.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.8.2 Documentation des fonctions

#### 5.8.2.1 void addTarget ( unitName *name*, vector *coordUnit* )

Ajoute une cible.

Ajoute une cible.

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité |
| *coordUnit* | Coordonnées de l'unité |

Définition à la ligne 211 du fichier listes.c.

#### 5.8.2.2 void addUnit ( vector *coordUnit* )

Ajoute une unité

Ajoute une unité

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordonnées de l'unité |

Définition à la ligne 179 du fichier listes.c.

#### 5.8.2.3 int countUnits ( )

Compte le nombre d'unités.

Compte le nombre d'unités.

**Renvoie**

Retourne le nombre d'unité

Définition à la ligne 280 du fichier listes.c.

#### 5.8.2.4 void destroyUnit ( vector *coordUnit* )

Détruit une unité

Détruit une unité

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordonnées de l'unité à détruire |

Définition à la ligne 260 du fichier listes.c.

#### 5.8.2.5 void dumpAllLists ( )

Vide les listes.

Vide les listes.

Définition à la ligne 167 du fichier listes.c.

#### 5.8.2.6 void dumpList ( short *nbList* )

Vide une liste.

**Paramètres**

| | |
|---|---|
| *nbList* | Numéro de la liste à vider |

Définition à la ligne 155 du fichier listes.c.

#### 5.8.2.7 void printList ( short *numList* )

Affiche la liste.

Affiche la liste.

**Paramètres**

| | |
|---|---|
| *numList* | Numéro de liste |

Définition à la ligne 222 du fichier listes.c.

#### 5.8.2.8 bool searchTarget ( int *numList*, vector *coordTarget* )

Cherche une cible.

Cherche une cible.

**Paramètres**

| | |
|---|---|
| *numList* | Numéro de la liste |
| *coordTarget* | Coordonnées de la cible |

**Renvoie**

Retourne vrai si cible trouvée

Définition à la ligne 301 du fichier listes.c.

### 5.8.3 Documentation des variables

#### 5.8.3.1 int targetList

Cibles potentielles.

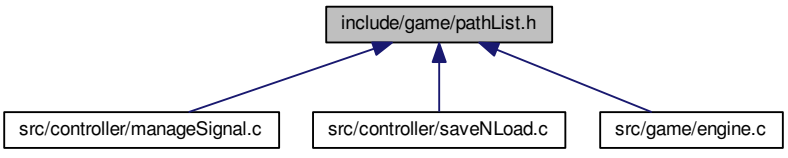
Cibles potentielles.

Définition à la ligne 28 du fichier listes.c.

## 5.9 Référence du fichier include/game/pathList.h

En-tête listes des chemins.

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




### Fonctions

— void initPath (int)

*Initialise le chemin.*

— void initPaths ()

*Initialise les chemins.*

— int emptyPath (int n)

*Vérifie si chemin vide.*

— int outPath (int n)

*Vérifie si l'élément courant est en dehors de la liste.*

— void pathHead (int n)

*Se met en tête de la liste.*

— void pathTail (int n)

*Se met en queue de la liste.*

— void previous (int n)

*Se positionne sur l'élément précédent.*

— void next (int n)

*Se positionne sur l'élément suivant.*

— void getTile (int n, vector ∗v, int ∗F)

*Récupère la valeur d'une case.*

— void setTile (int n, vector v, int F)

*Modifie la valeur d'une case.*

— void eraseTile (int n)

*Efface une case.*

— void toRightPath (int n, vector v, int F)

*Ajoute une case à droite.*

— void dumpPath (short nbList)

*Vide le chemin.*

— void dumpAllPaths ()

*Vide les chemins.*

— bool searchTile (int n, vector)

*Cherche une case.*

— vector getCurrentNode (int n)

*Récupère la case ayant le plus petit poids.*

— void addCloseList (vector, int)

*Ajoute la case à la liste fermée.*

— void addOpenList (vector, int)

*Ajoute la case à la liste ouverte.*

### 5.9.1 Description détaillée

En-tête listes des chemins.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.9.2 Documentation des fonctions

#### 5.9.2.1 void addCloseList ( vector *current,* int *F* )

Ajoute la case à la liste fermée.

Ajoute la case à la liste fermée.

**Paramètres**

| | |
|---|---|
| *current* | Destination |
| *F* | Poids de la case |

Définition à la ligne 262 du fichier pathList.c.

#### 5.9.2.2 void addOpenList ( vector *current*, int *F* )

Ajoute la case à la liste ouverte.

Ajoute la case à la liste ouverte.

**Paramètres**

| | |
|---|---|
| *current* | Destination |
| *F* | Poids de la case |

Définition à la ligne 285 du fichier pathList.c.

#### 5.9.2.3 void dumpAllPaths ( )

Vide les chemins.

Vide les chemins.

Définition à la ligne 226 du fichier pathList.c.

#### 5.9.2.4 void dumpPath ( short *nbList* )

Vide le chemin.

Vide le chemin.

**Paramètres**

| | |
|---|---|
| *nbList* | Numéro de la liste à vider |

Définition à la ligne 214 du fichier pathList.c.

#### 5.9.2.5 int emptyPath ( int *n* )

Vérifie si chemin vide.

Vérifie si chemin vide.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

**Renvoie**

Retourne le drapeau si pas vide

Définition à la ligne 54 du fichier pathList.c.

#### 5.9.2.6 void eraseTile ( int *n* )

Efface une case.

Efface une case.

**Paramètres**

| | |
|---|---|
| *n* | Liste |

Définition à la ligne 170 du fichier pathList.c.

#### 5.9.2.7 vector getCurrentNode ( int *n* )

Récupère la case ayant le plus petit poids.

Récupère la case ayant le plus petit poids.

**Paramètres**

| | |
|---|---|
| *n* | Liste dans laquelle chercher |

Définition à la ligne 130 du fichier pathList.c.

#### 5.9.2.8 void getTile ( int *n*, vector * *v*, int * *F* )

Récupère la valeur d'une case.

Récupère la valeur d'une case.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |
| *v* | Position de la case |
| *F* | Poids de la case |

Définition à la ligne 117 du fichier pathList.c.

#### 5.9.2.9 void initPath ( int *n* )

Initialise le chemin.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

Définition à la ligne 31 du fichier pathList.c.

#### 5.9.2.10 void next ( int *n* )

Se positionne sur l'élément suivant.

Se positionne sur l'élément suivant.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

Définition à la ligne 105 du fichier pathList.c.

#### 5.9.2.11 int outPath ( int *n* )

Vérifie si l'élément courant est en dehors de la liste.

Vérifie si l'élément courant est en dehors de la liste.

**Paramètres**

| | |
|---|---|
| *n* | Liste |

**Renvoie**

Retourne l'élément courant si pas hors liste

Définition à la ligne 65 du fichier pathList.c.

#### 5.9.2.12 void pathHead ( int *n* )

Se met en tête de la liste.

Se met en tête de la liste.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

Définition à la ligne 75 du fichier pathList.c.

#### 5.9.2.13 void pathTail ( int *n* )

Se met en queue de la liste.

Se met en queue de la liste.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

Définition à la ligne 85 du fichier pathList.c.

#### 5.9.2.14 void previous ( int *n* )

Se positionne sur l'élément précédent.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

Définition à la ligne 95 du fichier pathList.c.

#### 5.9.2.15 bool searchTile ( int *n*, vector *coordTile* )

Cherche une case.

Cherche une case.

**Paramètres**

| | |
|---|---|
| *n* | Liste |
| *coordTile* | Position de la case |

Définition à la ligne 239 du fichier pathList.c.

#### 5.9.2.16 void setTile ( int *n*, vector *v*, int *F* )

Modifie la valeur d'une case.

Modifie la valeur d'une case.

**Paramètres**

| | |
|---|---|
| *n* | Liste |
| *v* | Nouvelle position |
| *F* | Nouveau poids |

Définition à la ligne 157 du fichier pathList.c.

#### 5.9.2.17 void toRightPath ( int *n*, vector *v*, int *F* )

Ajoute une case à droite.

**Paramètres**

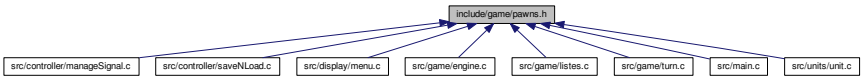
| | |
|---|---|
| *n* | Liste |
| *v* | Position à ajouter |
| *F* | Poids à ajouter |

Définition à la ligne 191 du fichier pathList.c.

## 5.10 Référence du fichier include/game/pawns.h

En-tête gestions des pions.

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




### Fonctions

— void makePawns ()
*Crée les pions.*

### Variables

— unit * pawns
*Tableaux des pions.*
— int sizePawns
*Nombre de pions.*

### 5.10.1 Description détaillée

En-tête gestions des pions.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.10.2 Documentation des fonctions

#### 5.10.2.1 void makePawns ( )

Crée les pions.

Crée les pions.

Définition à la ligne 109 du fichier pawns.c.

### 5.10.3 Documentation des variables

#### 5.10.3.1 unit* pawns

Tableaux des pions.

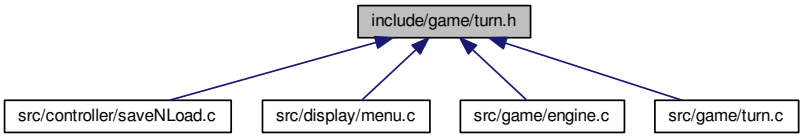
Tableaux des pions.

Définition à la ligne 16 du fichier pawns.c.

## 5.11 Référence du fichier include/game/turn.h

En-tête gestion des tours.

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




### Macros

- #define TIME_BY_UNIT 12
  *Temps par unité possédée par le joueur.*
- #define MIN_TIME 60
  *Temps minimum par tour.*
- #define MAX_TIME 120
  *Temps maximum par tour.*

### Fonctions

- void playTurn ()
  *Joue un tour.*
- void playAttack ()
  *Joue une attaque.*
- void playMove ()
  *Joue une mouvement.*
- void passTurn ()

*Passe le tour.*

— void changeDirection ()

*Change la direction.*

— bool hasPlay ()

*Vérifie si une action a été effectuée.*

— void surrender ()

*Abandonne la partie.*

### Variables

— int hasMoved

*Joueur a joué*

— int hasAttacked

*Joueur a attaqué*

— int hasPassed

*Joueur a passé son tour.*

— int hasSurrender

*Joueur a abandonné la partie.*

## 5.11.1 Description détaillée

En-tête gestion des tours.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

## 5.11.2 Documentation des fonctions

### 5.11.2.1 void changeDirection ( )

Change la direction.

Change la direction.

Définition à la ligne 57 du fichier turn.c.

### 5.11.2.2 bool hasPlay ( )

Vérifie si une action a été effectuée.

Vérifie si une action a été effectuée.

**Renvoie**

Retourne vrai si le joueur a joué sinon faux

Définition à la ligne 245 du fichier turn.c.

#### 5.11.2.3 void passTurn ( )

Passe le tour.

Passe le tour.

Définition à la ligne 235 du fichier turn.c.

#### 5.11.2.4 void playAttack ( )

Joue une attaque.

Joue une attaque.

Définition à la ligne 104 du fichier turn.c.

#### 5.11.2.5 void playMove ( )

Joue une mouvement.

Joue une mouvement.

Définition à la ligne 177 du fichier turn.c.

#### 5.11.2.6 void playTurn ( )

Joue un tour.

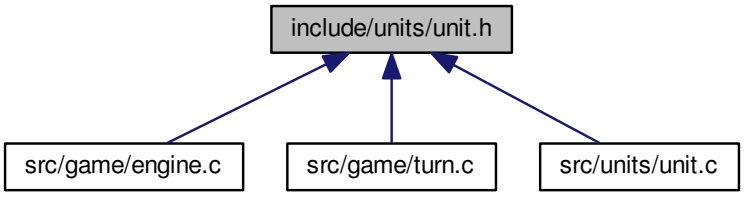
Joue un tour.

Définition à la ligne 272 du fichier turn.c.

## 5.12 Référence du fichier include/units/unit.h

En-tête gestion des unités.

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




### Fonctions

- bool canGetPassed (unit *target)
  *Vérifie si le passage est autorisé*
- bool canBlock (unit *target)
  *Vérifie si l'unité peut bloquer.*
- bool canAttack (unit *target)

*Vérifie si l'unité peut attaquer.*
- bool canMove (unit *target)
  *Vérifie si l'unité peut bouger.*
- bool canTeleport (unitName name)
  *Vérifie si l'unité peut se téléporter.*
- int getSideAttacked (vector source, vector target)
  *Récupère le côté attaqué*
- void heal (unitName name)
  *Soigne les unités.*
- void attack (vector source, vector target)
  *Attaque une unité*
- bool copy (unit *destination, unit *source)
  *Copie la structure source vers destination.*
- void move (vector destination, vector source)
  *Bouge une unité*
- void addEffect (vector target, unitEffect effect)
  *Ajoute un effet à une unité*
- void unitInit (short noPlayer, vector coordUnit)
  *Initialise une unité*
- void setDirection (vector coordUnit, int dir)
  *Définis la direction d'une unité*
- void erase (unit *)
  *Efface une unité de la grille.*
- bool isSleeping (vector)
  *Vérifie si une unité est endormie.*
- void recover ()
  *Se réveille tour par tour.*
- bool allStatic (int numPlayer)
  *Vérifie si toutes les unités sont immobilisées.*
- void minusEffect ()
  *Diminue le temps de l'effet.*
- void poison ()
  *Empoisonne une unité*
- void powerBonus ()
  *Octroie un bonus de puissance selon certaines conditions.*
- void sleep (vector)
  *Endors une unité*

### 5.12.1 Description détaillée

En-tête gestion des unités.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.12.2 Documentation des fonctions

#### 5.12.2.1 void addEffect ( vector *target*, unitEffect *effect* )

Ajoute un effet à une unité

Ajoute un effet à une unité

**Paramètres**

| | |
|---|---|
| *target* | Position unité cible |
| *effect* | Effet à appliquer sur la cible |

Définition à la ligne 437 du fichier unit.c.

#### 5.12.2.2 bool allStatic ( int *numPlayer* )

Vérifie si toutes les unités sont immobilisées.

Vérifie si toutes les unités sont immobilisées.

**Paramètres**

| | |
|---|---|
| *numPlayer* | Numéro du joueur |

**Renvoie**

Retourne vrai si toutes les unités sont immobilisés

Définition à la ligne 529 du fichier unit.c.

#### 5.12.2.3 void attack ( vector *source*, vector *target* )

Attaque une unité

Attaque une unité

**Paramètres**

| | |
|---|---|
| *source* | Unité attaquante |
| *target* | Unité cible |

Définition à la ligne 184 du fichier unit.c.

#### 5.12.2.4 bool canAttack ( unit * *target* )

Vérifie si l'unité peut attaquer.

Vérifie si l'unité peut attaquer.

**Paramètres**

| | |
|---|---|
| *target* | Unité à analyser |

**Renvoie**

Retourne vrai si l'unité peut attaquer

Définition à la ligne 86 du fichier unit.c.

#### 5.12.2.5 bool canBlock ( unit * *target* )

Vérifie si l'unité peut bloquer.

Vérifie si l'unité peut bloquer.

**Paramètres**

| | |
|---|---|
| *target* | Unité à analyser |

**Renvoie**

Retourne vrai si l'unité peut bloquer

Définition à la ligne 66 du fichier unit.c.

#### 5.12.2.6 bool canGetPassed ( unit * *target* )

Vérifie si le passage est autorisé

Vérifie si le passage est autorisé

**Paramètres**

| | |
|---|---|
| *target* | Unité à analyser |

**Renvoie**

Retourne vrai si passage autorisé

Définition à la ligne 46 du fichier unit.c.

#### 5.12.2.7 bool canMove ( unit * *target* )

Vérifie si l'unité peut bouger.

Vérifie si l'unité peut bouger.

**Paramètres**

| | |
|---|---|
| *target* | Unité à analyser |

**Renvoie**

Retourne vraie si l'unité peut se mouvoir

Définition à la ligne 106 du fichier unit.c.

#### 5.12.2.8 bool canTeleport ( unitName *name* )

Vérifie si l'unité peut se téléporter.

Vérifie si l'unité peut se téléporter.

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité |

**Renvoie**

Retourne vraie si l'unité peut se déplacer

Définition à la ligne 125 du fichier unit.c.

#### 5.12.2.9 bool copy ( unit * *destination*, unit * *source* )

Copie la structure source vers destination.

Copie la structure source vers destination.

**Paramètres**

| | |
|---|---|
| *destination* | Structure destination |
| *source* | Structure source |

**Renvoie**

Retourne vrai si copie bien déroulée

Définition à la ligne 236 du fichier unit.c.

#### 5.12.2.10 void erase ( unit * *source* )

Efface une unité de la grille.

**Paramètres**

| | |
|---|---|
| *source* | Source à effacer |

Définition à la ligne 271 du fichier unit.c.

#### 5.12.2.11 int getSideAttacked ( vector *source*, vector *target* )

Récupère le côté attaqué

Récupère le côté attaqué

**Paramètres**

| | |
|---|---|
| *source* | Position unité source |
| *target* | Position unité cible |

**Renvoie**

Le côté attaqué

Définition à la ligne 168 du fichier unit.c.

#### 5.12.2.12 void heal ( unitName *name* )

Soigne les unités.

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité soignant |

Définition à la ligne 137 du fichier unit.c.

#### 5.12.2.13 bool isSleeping ( vector *pos* )

Vérifie si une unité est endormie.

Vérifie si une unité est endormie.

**Paramètres**

| | |
|---|---|
| *pos* | Position de l'unité |

**Renvoie**

Retourne vrai si endormi

Définition à la ligne 490 du fichier unit.c.

#### 5.12.2.14 void minusEffect ( )

Diminue le temps de l'effet.

Diminue le temps de l'effet.

Définition à la ligne 455 du fichier unit.c.

#### 5.12.2.15 void move ( vector *destination,* vector *source* )

Bouge une unité

Bouge une unité

**Paramètres**

| | |
|---|---|
| *destination* | Destination souhaitée |
| *source* | Position de l'unité |

Définition à la ligne 351 du fichier unit.c.

#### 5.12.2.16 void poison ( )

Empoisonne une unité

Empoisonne une unité

Définition à la ligne 386 du fichier unit.c.

#### 5.12.2.17 void powerBonus ( )

Octroie un bonus de puissance selon certaines conditions.

Octroie un bonus de puissance selon certaines conditions.

Définition à la ligne 286 du fichier unit.c.

#### 5.12.2.18 void recover ( )

Se réveille tour par tour.

Se réveille tour par tour.

Définition à la ligne 503 du fichier unit.c.

#### 5.12.2.19 void setDirection ( vector *source,* int *dir* )

Définis la direction d'une unité

Définis la direction d'une unité

**Paramètres**

| | |
|---|---|
| *source* | Unité à tourner |
| *dir* | Direction dans laquelle tourner |

Définition à la ligne 377 du fichier unit.c.

#### 5.12.2.20 void sleep ( vector *pos* )

Endors une unité

Endors une unité

**Paramètres**

| | |
|---|---|
| *pos* | Position de l'unité |

Définition à la ligne 480 du fichier unit.c.

#### 5.12.2.21 void unitInit ( short *noP*, vector *coordUnit* )

Initialise une unité

Initialise une unité

**Paramètres**

| | |
|---|---|
| *noP* | Numéro du joueur |
| *coordUnit* | Coordonnées de l'unité |

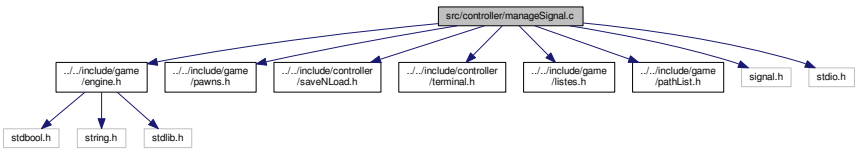
Définition à la ligne 24 du fichier unit.c.

## 5.13 Référence du fichier src/controller/manageSignal.c

Gestion des signaux.

```
#include "../../include/game/engine.h"
#include "../../include/game/pawns.h"
#include "../../include/controller/saveNLoad.h"
#include "../../include/controller/terminal.h"
#include "../../include/game/listes.h"
#include "../../include/game/pathList.h"
#include <signal.h>
#include <stdio.h>
```

Graphe des dépendances par inclusion de manageSignal.c :




### Fonctions

- void freeAll ()
  *Libère tout ce qui reste encore en mémoire.*
- void interrupt (int signal)
  *A l'interruption du programme libère la mémoire et sauvegarde du jeu.*
- void timeDown (int signal)
  *Petit message sympa lors du temps écoulé*
- void terminator (int signal)
  *Libération de la mémoire lors de la fin du programme.*
- void checkSignal ()
  *Vérifie le type de signal envoyé au terminal.*

### 5.13.1 Description détaillée

Gestion des signaux.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.13.2 Documentation des fonctions

#### 5.13.2.1 void checkSignal ( )

Vérifie le type de signal envoyé au terminal.

Vérifie les signaux du programme.

Définition à la ligne 77 du fichier manageSignal.c.

#### 5.13.2.2 void freeAll ( )

Libère tout ce qui reste encore en mémoire.

Libère toute la mémoire.

Définition à la ligne 21 du fichier manageSignal.c.

#### 5.13.2.3 void interrupt ( int *signal* )

A l'interruption du programme libère la mémoire et sauvegarde du jeu.

**Paramètres**

| | |
|---|---|
| *signal* | Signal d'interruption |

Définition à la ligne 35 du fichier manageSignal.c.

#### 5.13.2.4 void terminator ( int *signal* )

Libération de la mémoire lors de la fin du programme.

**Paramètres**

| | |
|---|---|
| *signal* | Signal d'interruption |

Définition à la ligne 68 du fichier manageSignal.c.

#### 5.13.2.5 void timeDown ( int *signal* )

Petit message sympa lors du temps écoulé

**Paramètres**

| | |
|---|---|
| *signal* | Signal d'interruption |

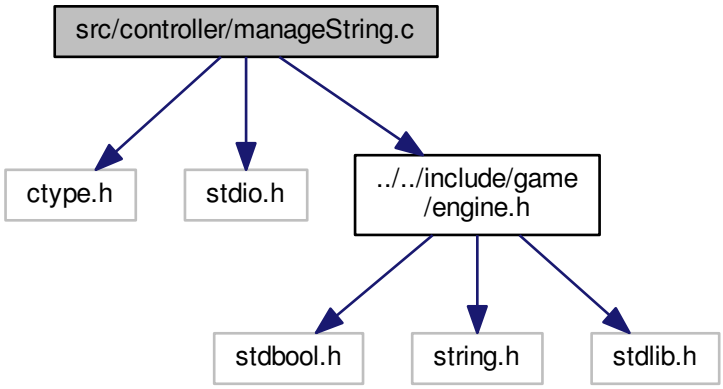
Définition à la ligne 58 du fichier manageSignal.c.

## 5.14 Référence du fichier src/controller/manageString.c

Gestions des chaînes de caractères.

```
#include <ctype.h>
#include <stdio.h>
#include "../../include/game/engine.h"
```

Graphe des dépendances par inclusion de manageString.c :




### Fonctions

- void getCoordS (char coordString[ ], vector ∗coordUnit)
  *Récupère les coordonnées d'une chaîne de caractère sous forme de vecteur x et y.*
- char ∗ get2Char (char name[ ])
  *Récupère 2 caractères du nom de l'unité*
- bool isOutGrid (char coordString[ ])
  *Vérifie que les coordonnées sont dans la grille.*
- bool correctCoord (char coordString[ ])
  *Sélectionne une coordonnée et vérifie son format.*
- bool rightSide (char ∗coordString)
  *Vérifie que l'unité est du bon côté*
- char ∗ getNameUnit (unitName name)
  *Récupère le nom de l'unité à partir de la liste énumérée.*
- char ∗ getNameEffect (unitEffect effect)
  *Récupère le nom de l'effet à partir de la liste énumérée.*
- char ∗ getDirectionUnit (cardinal direct)
  *Retourne la direction.*
- void printNameUnit (unitName unit)
  *Affiche le nom de l'unité*
- void clearBuffer ()
  *Vide le tampon mémoire.*
- int readS (char ∗string)
  *Lit la chaîne de caractère passée en paramètre et vérifie qu'elle ne soit pas trop longue.*
- long readLong ()
  *Lit un long de manière sécurisée.*
- double readDouble ()
  *Lit un double de manière sécurisée.*

### 5.14.1 Description détaillée

Gestions des chaînes de caractères.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.14.2 Documentation des fonctions

#### 5.14.2.1 void clearBuffer ( )

Vide le tampon mémoire.

Vide la mémoire tampon.

Définition à la ligne 253 du fichier manageString.c.

#### 5.14.2.2 bool correctCoord ( char *coordString[ ]* )

Sélectionne une coordonnée et vérifie son format.

**Paramètres**

| | |
|---|---|
| *coordString* | Chaîne de caractère à vérifier |

**Renvoie**

Vraie si les coordonnées saisies sont correctes

Définition à la ligne 138 du fichier manageString.c.

#### 5.14.2.3 char* get2Char ( char *name[ ]* )

Récupère 2 caractères du nom de l'unité

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité |

**Renvoie**

Retourne 2 caractères liés au nom de l'unité

Définition à la ligne 47 du fichier manageString.c.

#### 5.14.2.4 void getCoordS ( char *coordString[ ],* vector * *coordUnit* )

Récupère les coordonnées d'une chaîne de caractère sous forme de vecteur x et y.

Récupère les coordonnées d'une chaîne de caractères.

**Paramètres**

| | |
|---|---|
| *coordString* | Coordonnées saisie par l'utilisateur |
| *coordUnit* | Coordonnées de l'unité récupérées de la saisie utilisateur |

Définition à la ligne 19 du fichier manageString.c.

#### 5.14.2.5 char* getDirectionUnit ( **cardinal** *direct* )

Retourne la direction.

Récupère le nom de la direction.

**Paramètres**

| | |
|---|---|
| *direct* | Direction de l'unité |

**Renvoie**

Retourne La direction sous forme de chaîne

Définition à la ligne 235 du fichier manageString.c.

#### 5.14.2.6 char* getNameEffect ( **unitEffect** *effect* )

Récupère le nom de l'effet à partir de la liste énumérée.

Récupère le nom de l'effet.

**Paramètres**

| | |
|---|---|
| *effect* | Effet provenant de la liste énumérée |

**Renvoie**

Nom de l'effet sous forme de chaîne de caractère

Définition à la ligne 220 du fichier manageString.c.

#### 5.14.2.7 char* getNameUnit ( **unitName** *name* )

Récupère le nom de l'unité à partir de la liste énumérée.

Récupère le nom de l'unité

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité provenant de la liste énumérée |

**Renvoie**

Nom de l'unité sous forme de chaîne

Définition à la ligne 189 du fichier manageString.c.

#### 5.14.2.8 bool isOutGrid ( char *coordString[ ]* )

Vérifie que les coordonnées sont dans la grille.

**Paramètres**

| | |
|---|---|
| *coordString* | Coordonnées sous forme de chaîne |

**Renvoie**

Retourne false si pas en dehors

Définition à la ligne 114 du fichier manageString.c.

#### 5.14.2.9 void printNameUnit ( unitName *unit* )

Affiche le nom de l'unité

**Paramètres**

| | |
|---|---|
| *unit* | Nom de l'unité provenant de la liste énumérée |

Définition à la ligne 246 du fichier manageString.c.

#### 5.14.2.10 double readDouble ( )

Lit un double de manière sécurisée.

Lecture sécurisée d'un double.

**Renvoie**

Retourne le double saisie ou 0.0 en cas d'erreur

Définition à la ligne 305 du fichier manageString.c.

#### 5.14.2.11 long readLong ( )

Lit un long de manière sécurisée.

Lecture sécurisée d'un long.

**Renvoie**

Retourne le long saisie ou 0 en cas d'erreur

Définition à la ligne 291 du fichier manageString.c.

#### 5.14.2.12 int readS ( char * *string* )

Lit la chaîne de caractère passée en paramètre et vérifie qu'elle ne soit pas trop longue.

Lecture sécurisée d'une chaîne.

**Paramètres**

| | |
|---|---|
| *string* | Chaîne de caractère à vérifier |

**Renvoie**

Retourne 1 si chaîne correcte

Définition à la ligne 265 du fichier manageString.c.

#### 5.14.2.13 bool rightSide ( char * *coordString* )

Vérifie que l'unité est du bon côté

Vérifie que l'unité est dans le bon camp.

**Paramètres**

| | |
|---|---|
| *coordString* | Coordonnées sous forme de chaîne |

**Renvoie**

Retourne vrai si du bon côté

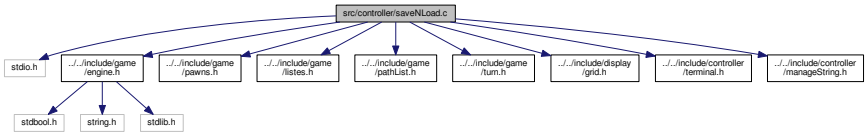
Définition à la ligne 165 du fichier manageString.c.

## 5.15 Référence du fichier src/controller/saveNLoad.c

Gestion de la sauvegarde et du chargement.

```
#include <stdio.h>
#include "../../include/game/engine.h"
#include "../../include/game/pawns.h"
#include "../../include/game/listes.h"
#include "../../include/game/pathList.h"
#include "../../include/game/turn.h"
#include "../../include/display/grid.h"
#include "../../include/controller/terminal.h"
#include "../../include/controller/manageString.h"
```

Graphe des dépendances par inclusion de saveNLoad.c :




### Fonctions

- char getCharKey (char *dynamicKey, int *pos)
  *Récupère un caractère de la clé*
- char * getKey (unitName name)
  *Récupère une clé de décryptage.*
- void crypt (unit *unitSaved)
  *Crypte les statistiques de l'unité pour la sauvegarde.*
- void decrypt (unit *unitLoaded)
  *Décrypte les statistiques de l'unité pour le chargement.*
- bool checkDecrypt (unit *unitLoaded)
  *Vérifie que le décryptage s'est bien passé*
- void save ()
  *Sauvegarde la partie avec l'état actuel de la grille et les données liées au joueur en cours.*
- void load ()
  *Charge la partie précédemment enregistré*

### Variables

- char key [ ] = "SPIBCTBEC"
  *Clé statique à utiliser pour le cryptage.*

#### 5.15.1 Description détaillée

Gestion de la sauvegarde et du chargement.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.15.2 Documentation des fonctions

#### 5.15.2.1 bool checkDecrypt ( unit * *unitLoaded* )

Vérifie que le décryptage s'est bien passé

**Paramètres**

| | |
|---|---|
| *unitLoaded* | Unité chargée |

**Renvoie**

Retourne vrai si données correctes sinon faux

Définition à la ligne 133 du fichier saveNLoad.c.

#### 5.15.2.2 void crypt ( unit * *unitSaved* )

Crypte les statistiques de l'unité pour la sauvegarde.

**Paramètres**

| | |
|---|---|
| *unitSaved* | Unité sauvegardée |

Définition à la ligne 70 du fichier saveNLoad.c.

#### 5.15.2.3 void decrypt ( unit * *unitLoaded* )

Décrypte les statistiques de l'unité pour le chargement.

**Paramètres**

| | |
|---|---|
| *unitLoaded* | Unité chargée |

Définition à la ligne 101 du fichier saveNLoad.c.

#### 5.15.2.4 char getCharKey ( char * *dynamicKey*, int * *pos* )

Récupère un caractère de la clé

**Paramètres**

| | |
|---|---|
| *dynamicKey* | Clé dynamique |
| *pos* | Position dans la clé |

**Renvoie**

Retourne un caractère de la clé

Définition à la ligne 27 du fichier saveNLoad.c.

#### 5.15.2.5 char* getKey ( unitName *name* )

Récupère une clé de décryptage.

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité |

**Renvoie**

Retourne une clé

Définition à la ligne 39 du fichier saveNLoad.c.

#### 5.15.2.6 void load ( )

Charge la partie précédemment enregistré

Charge une partie.

Définition à la ligne 201 du fichier saveNLoad.c.

#### 5.15.2.7 void save ( )

Sauvegarde la partie avec l'état actuel de la grille et les données liées au joueur en cours.

Sauvegarde la partie.

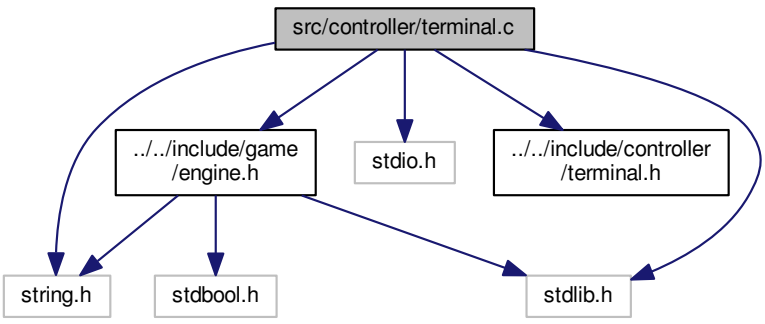
Définition à la ligne 158 du fichier saveNLoad.c.

## 5.16 Référence du fichier src/controller/terminal.c

Gestion du terminal.

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "../../include/controller/terminal.h"
#include "../../include/game/engine.h"
```

Graphe des dépendances par inclusion de terminal.c :




### Fonctions

— char * getColor (int color, char type[ ])
*Récupère le code correspondant à la couleur.*

— void color (int color, char type[ ])
*Change la couleur de l'écran ou de la police.*
— void reinitColor ()
*Réinitialise la couleur de l'écran.*
— void fontColor (int color)
*Change la couleur de la police.*
— void clearScreen ()
*Efface l'écran.*

### 5.16.1 Description détaillée

Gestion du terminal.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.16.2 Documentation des fonctions

#### 5.16.2.1 void color ( int *color,* char *type[ ]* )

Change la couleur de l'écran ou de la police.

Met en couleurs le texte ou l'écran.

**Paramètres**

| | |
|---|---|
| *color* | Couleur à utiliser |
| *type* | Texte à changer de couleur ou arrière plan |

Définition à la ligne 51 du fichier terminal.c.

#### 5.16.2.2 void fontColor ( int *color* )

Change la couleur de la police.

Met en couleurs la police.

**Paramètres**

| | |
|---|---|
| *color* | Couleur à utiliser |

Définition à la ligne 76 du fichier terminal.c.

#### 5.16.2.3 char* getColor ( int *color,* char *type[ ]* )

Récupère le code correspondant à la couleur.

**Paramètres**

| | |
|---|---|
| *color* | Couleur demandée |
| *type* | Type d'objet à modifier |

**Renvoie**

Code correspondant, écran blanc si code incorrect

Définition à la ligne 21 du fichier terminal.c.

#### 5.16.2.4 void reinitColor ( )

Réinitialise la couleur de l'écran.

Réinitialise les couleurs.

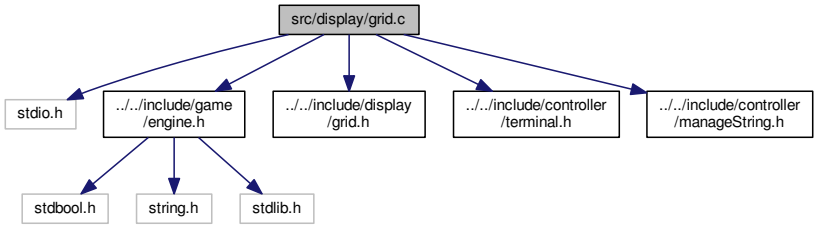
Définition à la ligne 67 du fichier terminal.c.

## 5.17 Référence du fichier src/display/grid.c

Gestion de la grille.

```
#include <stdio.h>
#include "../../include/game/engine.h"
#include "../../include/display/grid.h"
#include "../../include/controller/terminal.h"
#include "../../include/controller/manageString.h"
```

Graphe des dépendances par inclusion de grid.c :




### Fonctions

- void borderRight (short row)
  *Affiche une bordure sur le côté droit sur les lignes vides et utiles.*
- void borderHoriz ()
  *Affiche une bordure horizontale.*
- void dispX ()
  *Affiche les coordonnées verticales.*
- void gridDisp ()
  *Affiche la grille avec les coordonnées.*

### 5.17.1 Description détaillée

Gestion de la grille.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.17.2 Documentation des fonctions

#### 5.17.2.1 void borderRight ( short *row* )

Affiche une bordure sur le côté droit sur les lignes vides et utiles.

**Paramètres**

| | |
|---|---|
| *row* | Ligne actuelle |

Définition à la ligne 19 du fichier grid.c.

#### 5.17.2.2 void gridDisp ( )

Affiche la grille avec les coordonnées.

Affiche la grille.

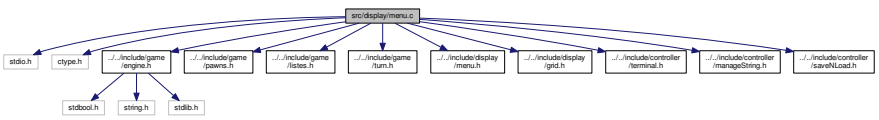
Définition à la ligne 78 du fichier grid.c.

## 5.18 Référence du fichier src/display/menu.c

Gestion des menus.

```
#include <stdio.h>
#include <ctype.h>
#include "../../include/game/engine.h"
#include "../../include/game/pawns.h"
#include "../../include/game/listes.h"
#include "../../include/game/turn.h"
#include "../../include/display/menu.h"
#include "../../include/display/grid.h"
#include "../../include/controller/terminal.h"
#include "../../include/controller/manageString.h"
#include "../../include/controller/saveNLoad.h"
```

Graphe des dépendances par inclusion de menu.c :




### Fonctions

— void mainMenu ()

*Menu principal du jeu.*
— void mainHelp ()
*Menu d'aide principal.*
— void helpUnit ()
*Menu d'aide spéifique aux unités.*
— void dispDirection ()
*Afficher les directions.*
— void gameMenu ()
*Menu du joueur lors de la partie.*
— void unitMenu (int choice)
*Menu de sélection de l'unité*
— void unitList ()
*Affiche la liste des unités inclus dans le jeu.*

### 5.18.1 Description détaillée

Gestion des menus.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.18.2 Documentation des fonctions

#### 5.18.2.1 void dispDirection ( )

Afficher les directions.

Affiche la liste des directions.

Définition à la ligne 458 du fichier menu.c.

#### 5.18.2.2 void gameMenu ( )

Menu du joueur lors de la partie.

Menu de jeu.

Définition à la ligne 470 du fichier menu.c.

#### 5.18.2.3 void helpUnit ( )

Menu d'aide spéifique aux unités.

Menu d'aide des unités.

Définition à la ligne 222 du fichier menu.c.

#### 5.18.2.4 void mainMenu ( )

Menu principal du jeu.

Menu principal.

Définition à la ligne 24 du fichier menu.c.

#### 5.18.2.5 void unitList ( )

Affiche la liste des unités inclus dans le jeu.

Listes des unités du jeu.

Définition à la ligne 593 du fichier menu.c.

#### 5.18.2.6 void unitMenu ( int *choice* )

Menu de sélection de l'unité

**Paramètres**

| | |
|---|---|
| *choice* | Choix de l'action pour l'unité |

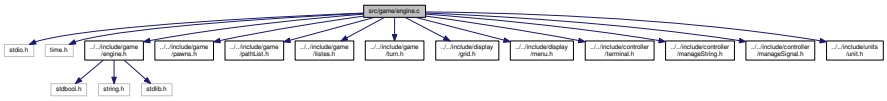
Définition à la ligne 517 du fichier menu.c.

## 5.19 Référence du fichier src/game/engine.c

Moteur de jeu.

```
#include <stdio.h>
#include <time.h>
#include "../../include/game/engine.h"
#include "../../include/game/pawns.h"
#include "../../include/game/pathList.h"
#include "../../include/game/listes.h"
#include "../../include/game/turn.h"
#include "../../include/display/grid.h"
#include "../../include/display/menu.h"
#include "../../include/controller/terminal.h"
#include "../../include/controller/manageString.h"
#include "../../include/controller/manageSignal.h"
#include "../../include/units/unit.h"
```

Graphe des dépendances par inclusion de engine.c :




### Fonctions

- bool possiblePath (vector coordUnit)
  *Trouve un chemin vers une quelconque position pour l'unité*
- bool pathFind (vector coordUnit, vector coordTarget)
  *Trouve un chemin vers la position désirée.*
- void lineOfSight (vector coordSource, vector coordTarget)
  *Ligne de vue de l'archer.*
- void movable (int colorDisp)
  *Fait la liste des unités déplaçables.*
- void attackable (int colorDisp)
  *Fait la liste des unités pouvant attaquer.*
- void tileWalkable (vector coordUnit, int colorDisp)
  *Fait la liste des cases pouvant être atteintes par l'unité*
- bool isSurrounded (vector currentUnit)
  *Permet de savoir si l'unité courante est entourée.*
- void setTarget (unitName name, vector coordUnit, int colorDisp)

*Initialise les cibles du pion.*

— void specialBoons (vector coordSource, vector coordTarget)
*Déclenche une capacité spéciale selon l'unité*

— void launchAttack (vector coordSource, vector coordTarget)
*Lance une attaque selon l'unité*

— bool selectUnit (vector *coordUnit)
*Sélectionne une unité*

— void gridInit ()
*Initialise la grille.*

— bool tooMuchUnit (int unitSelected, int limitUnits[ ])
*Vérifie que l'unité sélectionnée n'est pas en trop grand nombre sur le plateau.*

— void updateLimits (int unitSelected, int limitUnits[ ], vector coordUnit)
*Met à jour les limites d'unités.*

— void askUnit (int *unitSelected, int limitUnits[ ])
*Demande de choisir une unité*

— void askCoord (char coordString[ ])
*Demande les coordonnées de l'unité à placer.*

— void playerAddUnit (int limitUnits[ ], int *nbUnit)
*Placement des unités par le joueur.*

— void playerInit ()
*Initiliase la liste des unités du joueur.*

— bool endGame ()
*Fin de la partie.*

— void startGame ()
*Débute la partie.*

— void gameInit ()
*Initialise la partie.*

## Variables

— unit grid [N][N]
*Grille d'unités.*

— int noPlayer = FIRST_PLAYER
*Joueur en cours.*

### 5.19.1 Description détaillée

Moteur de jeu.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.19.2 Documentation des fonctions

#### 5.19.2.1 void askCoord ( char *coordString[ ]* )

Demande les coordonnées de l'unité à placer.

**Paramètres**

| | |
|---|---|
| *coordString* | Coordonnées sous forme de chaîne |

Définition à la ligne 579 du fichier engine.c.

#### 5.19.2.2 void askUnit ( int * *unitSelected*, int *limitUnits[ ]* )

Demande de choisir une unité

**Paramètres**

| | |
|---|---|
| *unitSelected* | Unité sélectionnée |
| *limitUnits* | Nombre limite pour chaque unité |

Définition à la ligne 551 du fichier engine.c.

#### 5.19.2.3 void attackable ( int *colorDisp* )

Fait la liste des unités pouvant attaquer.

**Paramètres**

| | |
|---|---|
| *colorDisp* | Couleur d'affichage |

Définition à la ligne 164 du fichier engine.c.

#### 5.19.2.4 void gameInit ( )

Initialise la partie.

Initialise le jeu.

Définition à la ligne 750 du fichier engine.c.

#### 5.19.2.5 bool isSurrounded ( vector *currentUnit* )

Permet de savoir si l'unité courante est entourée.

Vérifie si une unité est entourée.

**Paramètres**

| | |
|---|---|
| *currentUnit* | Unité courante |

**Renvoie**

Retourne vrai si unité entourée

Définition à la ligne 215 du fichier engine.c.

#### 5.19.2.6 void launchAttack ( vector *coordSource*, vector *coordTarget* )

Lance une attaque selon l'unité

Lance une attaque.

**Paramètres**

| | |
|---|---|
| *coordSource* | Nom de l'unité source |
| *coordTarget* | Coordonnées de la cible |

Définition à la ligne 336 du fichier engine.c.

#### 5.19.2.7 void lineOfSight ( vector *coordSource*, vector *coordTarget* )

Ligne de vue de l'archer.

**Paramètres**

| | |
|---|---|
| *coordSource* | Coordonnées de l'unité attaquante |
| *coordTarget* | Coordonnées de la cible |

Définition à la ligne 117 du fichier engine.c.

#### 5.19.2.8 void movable ( int *colorDisp* )

Fait la liste des unités déplaçables.

**Paramètres**

| | |
|---|---|
| *colorDisp* | Couleur d'affichage |

Définition à la ligne 141 du fichier engine.c.

#### 5.19.2.9 bool pathFind ( vector *coordUnit*, vector *coordTarget* )

Trouve un chemin vers la position désirée.

Trouve une chemin vers la destination.

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordonnées de l'unité |
| *coordTarget* | Coordonnées de la cible |

**Renvoie**

Retourne vrai si chemin trouvé

Définition à la ligne 60 du fichier engine.c.

#### 5.19.2.10 void playerAddUnit ( int *limitUnits[ ]*, int * *nbUnit* )

Placement des unités par le joueur.

**Paramètres**

| | |
|---|---|
| *limitUnits* | Limites d'unités |
| *nbUnit* | Nombre d'unités restantes à placer |

Définition à la ligne 608 du fichier engine.c.

#### 5.19.2.11 bool possiblePath ( vector *coordUnit* )

Trouve un chemin vers une quelconque position pour l'unité

Vérifie qu'un chemin est possible.

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordonnées de l'unité |

**Renvoie**

Retourne Vrai si chemin possible vers une quelconque position

Définition à la ligne 31 du fichier engine.c.

#### 5.19.2.12 bool selectUnit ( vector * *coordUnit* )

Sélectionne une unité

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordoonnées de l'unité |

**Renvoie**

Retourne vrai si unité bien sélectionnée

Définition à la ligne 416 du fichier engine.c.

#### 5.19.2.13 void setTarget ( unitName *name,* vector *coordUnit,* int *colorDisp* )

Initialise les cibles du pion.

Définis les cibles.

**Paramètres**

| | |
|---|---|
| *name* | Nom du pion |
| *coordUnit* | Coordonnées de l'unité |
| *colorDisp* | Couleur d'affichage |

Définition à la ligne 252 du fichier engine.c.

#### 5.19.2.14 void specialBoons ( vector *coordSource,* vector *coordTarget* )

Déclenche une capacité spéciale selon l'unité

**Paramètres**

| | |
|---|---|
| *coordSource* | Coordonnées de l'unité source (déclenchant une capacité) |
| *coordTarget* | Coordonnées de l'unité affectée par la capacité spéciale |

Définition à la ligne 299 du fichier engine.c.

#### 5.19.2.15 void tileWalkable ( vector *coordUnit,* int *colorDisp* )

Fait la liste des cases pouvant être atteintes par l'unité

Fait la liste des cases atteignables par l'unité

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordonnées de l'unité |

| | |
|---|---|
| *colorDisp* | Couleur d'affichage |

Définition à la ligne 188 du fichier engine.c.

#### 5.19.2.16 bool tooMuchUnit ( int *unitSelected*, int *limitUnits[ ]* )

Vérifie que l'unité sélectionnée n'est pas en trop grand nombre sur le plateau.

**Paramètres**

| | |
|---|---|
| *unitSelected* | Unité sélectionnée |
| *limitUnits* | Tableau des limites pour chaque unité |

**Renvoie**

Retourne vrai si unité en surnombre

Définition à la ligne 468 du fichier engine.c.

#### 5.19.2.17 void updateLimits ( int *unitSelected*, int *limitUnits[ ]*, vector *coordUnit* )

Met à jour les limites d'unités.

**Paramètres**

| | |
|---|---|
| *unitSelected* | Unité sélectionnée |
| *limitUnits* | Limites liées aux unités sur le plateau |
| *coordUnit* | Coordonnées de l'unité |

Définition à la ligne 515 du fichier engine.c.

### 5.19.3 Documentation des variables

#### 5.19.3.1 unit grid[N][N]

Grille d'unités.

Représentation d'une grille d'unité globale.

Définition à la ligne 23 du fichier engine.c.

#### 5.19.3.2 int noPlayer = FIRST_PLAYER

Joueur en cours.

Représentation du joueur.

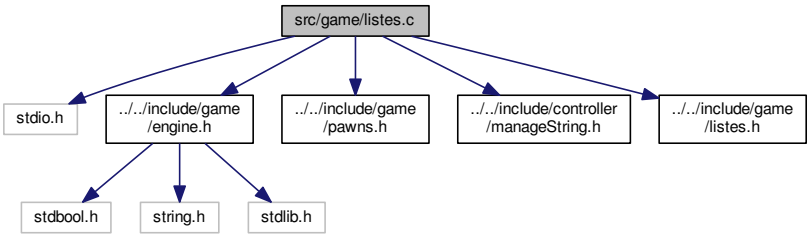
Définition à la ligne 24 du fichier engine.c.

## 5.20 Référence du fichier src/game/listes.c

Listes de vecteurs.

```
#include <stdio.h>
#include "../../include/game/engine.h"
#include "../../include/game/pawns.h"
#include "../../include/controller/manageString.h"
#include "../../include/game/listes.h"
```

Graphe des dépendances par inclusion de listes.c :




## Classes

— struct element
*Représente un élément de la liste.*

## Définitions de type

— typedef struct element t_element
*Définis un élément.*

## Fonctions

— void init_liste (int n)
*Initialise une liste.*
— void initLists ()
*Initialise les listes.*
— int liste_vide (int n)
*Vérifie si une liste est vide.*
— int hors_liste (int n)
*Vérifie si l'élément courant est hors de la liste.*
— void en_tete (int n)
*Se met en tête de la liste.*
— void en_queue (int n)
*Se met en queue de la liste.*
— void precedent (int n)
*Se positionne sur l'élément précédent.*
— void suivant (int n)
*Se positionne sur l'élément suivant.*
— void valeur_elt (int n, vector ∗v)
*Récupère la valeur de l'élément.*
— void modif_elt (int n, vector v)
*Modifie la valeur de l'élément.*
— void oter_elt (int n)
*Supprime l'élément.*
— void ajout_droit (int n, vector v)
*Ajoute à droite l'élément.*
— void ajout_gauche (int n, vector v)
*Ajoute à gauche l'élément.*
— void dumpList (short nbList)
*Vide une liste.*
— void dumpAllLists ()
*Vide toutes les listes.*
— void addUnit (vector coordUnit)

*Ajoute une unité dans la liste des unités du joueur.*
- void addTarget (unitName name, vector coordUnit)
  *Ajoute une cible pour une unité*
- void printList (short numList)
  *Affiche la liste désirée.*
- void destroyUnit (vector coordUnit)
  *Détruit une unité dans la liste.*
- int countUnits ()
  *Compte le nombre d'unité*
- bool searchTarget (int numList, vector coordTarget)
  *Cherche une cible dans la liste.*

## Variables

- t_element * drapeau [NB_UNITS]
  *Tableaux de drapeau.*
- t_element * ec [NB_UNITS]
  *Tableau d'élément courant.*
- int targetList = NB_PLAYERS + 1
  *Numéro de la liste des cibles.*

### 5.20.1 Description détaillée

Listes de vecteurs.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.20.2 Documentation des fonctions

#### 5.20.2.1 void addTarget ( unitName *name*, vector *coordUnit* )

Ajoute une cible pour une unité

Ajoute une cible.

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité |
| *coordUnit* | Coordonnées de l'unité |

Définition à la ligne 211 du fichier listes.c.

#### 5.20.2.2 void addUnit ( vector *coordUnit* )

Ajoute une unité dans la liste des unités du joueur.

Ajoute une unité

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordonnées de l'unité |

Définition à la ligne 179 du fichier listes.c.

#### 5.20.2.3 int countUnits ( )

Compte le nombre d'unité

Compte le nombre d'unités.

**Renvoie**

Retourne le nombre d'unité

Définition à la ligne 280 du fichier listes.c.

#### 5.20.2.4 void destroyUnit ( vector *coordUnit* )

Détruit une unité dans la liste.

Détruit une unité

**Paramètres**

| | |
|---|---|
| *coordUnit* | Coordonnées de l'unité à détruire |

Définition à la ligne 260 du fichier listes.c.

#### 5.20.2.5 void dumpAllLists ( )

Vide toutes les listes.

Vide les listes.

Définition à la ligne 167 du fichier listes.c.

#### 5.20.2.6 void dumpList ( short *nbList* )

Vide une liste.

**Paramètres**

| | |
|---|---|
| *nbList* | Numéro de la liste à vider |

Définition à la ligne 155 du fichier listes.c.

#### 5.20.2.7 void printList ( short *numList* )

Affiche la liste désirée.

Affiche la liste.

**Paramètres**

| | |
|---|---|
| *numList* | Numéro de liste |

Définition à la ligne 222 du fichier listes.c.

#### 5.20.2.8 bool searchTarget ( int *numList*, vector *coordTarget* )

Cherche une cible dans la liste.

Cherche une cible.

**Paramètres**

| | |
|---|---|
| *numList* | Numéro de la liste |
| *coordTarget* | Coordonnées de la cible |

**Renvoie**

Retourne vrai si cible trouvée

Définition à la ligne 301 du fichier listes.c.

### 5.20.3 Documentation des variables

#### 5.20.3.1 int targetList = NB_PLAYERS + 1

Numéro de la liste des cibles.

Cibles potentielles.

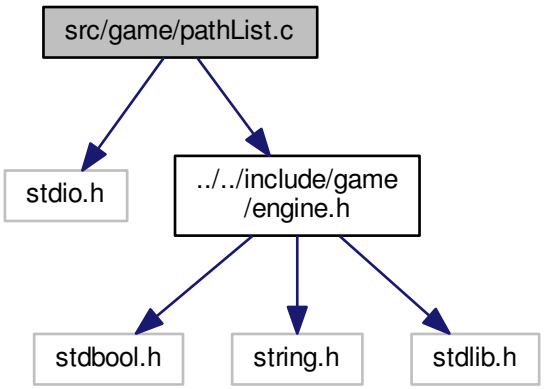
Définition à la ligne 28 du fichier listes.c.

## 5.21 Référence du fichier src/game/pathList.c

Listes de cases sur un chemin définis.

```
#include <stdio.h>
#include "../../include/game/engine.h"
```

Graphe des dépendances par inclusion de pathList.c :




### Classes

— struct elem
*Représente une case de la grille.*

### Définitions de type

— typedef struct elem t_tile
*Définis la case.*

### Fonctions

— void initPath (int n)
*Initialise le chemin.*
— void initPaths ()
*Initialise les chemins.*
— int emptyPath (int n)
*Vérifie si un chemin est vide.*
— int outPath (int n)
*Vérifie si l'élément courant est hors liste.*
— void pathHead (int n)
*Se met en tête du chemin.*
— void pathTail (int n)
*Se met en queue du chemin.*
— void previous (int n)
*Se positionne sur l'élément précédent.*
— void next (int n)
*Se positionne sur l'élémént suivant.*
— void getTile (int n, vector ∗v, int ∗F)
*Récupère une case du chemin.*
— vector getCurrentNode (int n)
*Récupère le noeud ayant le plus petit F.*
— void setTile (int n, vector v, int F)
*Modifie les infos d'une case dans la liste.*
— void eraseTile (int n)
*Efface une case de la grille.*
— void toRightPath (int n, vector v, int F)
*Ajoute une case à droite.*
— void dumpPath (short nbList)
*Vide une liste.*
— void dumpAllPaths ()
*Vide toutes les listes.*
— bool searchTile (int n, vector coordTile)
*Cherche si un vecteur est présent dans la liste fermée ou ouverte.*
— void addCloseList (vector current, int F)
*Ajoute à la liste fermée.*
— void addOpenList (vector current, int F)
*Ajoute à la liste ouverte.*

### Variables

— t_tile ∗ flag [2]
*Tableau de drapeaux.*
— t_tile ∗ elemC [2]
*Tableau d'éléments courants.*

#### 5.21.1 Description détaillée

Listes de cases sur un chemin définis.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.21.2 Documentation des fonctions

#### 5.21.2.1 void addCloseList ( vector *current,* int *F* )

Ajoute à la liste fermée.

Ajoute la case à la liste fermée.

**Paramètres**

| | |
|---|---|
| *current* | Destination |
| *F* | Poids de la case |

Définition à la ligne 262 du fichier pathList.c.

#### 5.21.2.2 void addOpenList ( vector *current,* int *F* )

Ajoute à la liste ouverte.

Ajoute la case à la liste ouverte.

**Paramètres**

| | |
|---|---|
| *current* | Destination |
| *F* | Poids de la case |

Définition à la ligne 285 du fichier pathList.c.

#### 5.21.2.3 void dumpAllPaths ( )

Vide toutes les listes.

Vide les chemins.

Définition à la ligne 226 du fichier pathList.c.

#### 5.21.2.4 void dumpPath ( short *nbList* )

Vide une liste.

Vide le chemin.

**Paramètres**

| | |
|---|---|
| *nbList* | Numéro de la liste à vider |

Définition à la ligne 214 du fichier pathList.c.

#### 5.21.2.5 int emptyPath ( int *n* )

Vérifie si un chemin est vide.

Vérifie si chemin vide.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

**Renvoie**

Retourne le drapeau si pas vide

Définition à la ligne 54 du fichier pathList.c.

#### 5.21.2.6 void eraseTile ( int *n* )

Efface une case de la grille.

Efface une case.

**Paramètres**

| | |
|---|---|
| *n* | Liste |

Définition à la ligne 170 du fichier pathList.c.

#### 5.21.2.7 vector getCurrentNode ( int *n* )

Récupère le noeud ayant le plus petit F.

Récupère la case ayant le plus petit poids.

**Paramètres**

| | |
|---|---|
| *n* | Liste dans laquelle chercher |

Définition à la ligne 130 du fichier pathList.c.

#### 5.21.2.8 void getTile ( int *n*, vector * *v*, int * *F* )

Récupère une case du chemin.

Récupère la valeur d'une case.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |
| *v* | Position de la case |
| *F* | Poids de la case |

Définition à la ligne 117 du fichier pathList.c.

#### 5.21.2.9 void initPath ( int *n* )

Initialise le chemin.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

Définition à la ligne 31 du fichier pathList.c.

#### 5.21.2.10 void next ( int *n* )

Se positionne sur l'élémént suivant.

Se positionne sur l'élément suivant.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

Définition à la ligne 105 du fichier pathList.c.

#### 5.21.2.11 int outPath ( int *n* )

Vérifie si l'élément courant est hors liste.

Vérifie si l'élément courant est en dehors de la liste.

**Paramètres**

| | |
|---|---|
| *n* | Liste |

**Renvoie**

Retourne l'élément courant si pas hors liste

Définition à la ligne 65 du fichier pathList.c.

#### 5.21.2.12 void pathHead ( int *n* )

Se met en tête du chemin.

Se met en tête de la liste.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

Définition à la ligne 75 du fichier pathList.c.

#### 5.21.2.13 void pathTail ( int *n* )

Se met en queue du chemin.

Se met en queue de la liste.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

Définition à la ligne 85 du fichier pathList.c.

#### 5.21.2.14 void previous ( int *n* )

Se positionne sur l'élément précédent.

**Paramètres**

| | |
|---|---|
| *n* | Chemin |

Définition à la ligne 95 du fichier pathList.c.

#### 5.21.2.15 bool searchTile ( int *n*, vector *coordTile* )

Cherche si un vecteur est présent dans la liste fermée ou ouverte.

Cherche une case.

**Paramètres**

| | |
|---|---|
| *n* | Liste |
| *coordTile* | Position de la case |

Définition à la ligne 239 du fichier pathList.c.

#### 5.21.2.16 void setTile ( int *n*, vector *v*, int *F* )

Modifie les infos d'une case dans la liste.

Modifie la valeur d'une case.

**Paramètres**

| | |
|---|---|
| *n* | Liste |
| *v* | Nouvelle position |
| *F* | Nouveau poids |

Définition à la ligne 157 du fichier pathList.c.

#### 5.21.2.17 void toRightPath ( int *n*, vector *v*, int *F* )

Ajoute une case à droite.

**Paramètres**

| | |
|---|---|
| *n* | Liste |
| *v* | Position à ajouter |
| *F* | Poids à ajouter |

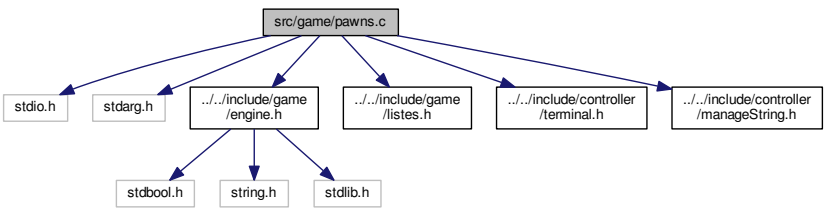
Définition à la ligne 191 du fichier pathList.c.

## 5.22 Référence du fichier src/game/pawns.c

Gestion des pions.

```
#include <stdio.h>
#include <stdarg.h>
#include "../../include/game/engine.h"
#include "../../include/game/listes.h"
#include "../../include/controller/terminal.h"
#include "../../include/controller/manageString.h"
```

Graphe des dépendances par inclusion de pawns.c :



### Fonctions

- void initPawn (unit *pawn)
  *Initialise un pion.*
- bool checkPawn (unit *pawn)
  *Vérifie que le pion est correct.*
- void createPawn (int nbParams, unitName name,...)
  *Crée un pion pour le jeu.*
- void makePawns ()
  *Crée tout les pions utiles au jeu.*

### Variables

- unit * pawns = NULL
  *Pions initialisés avec leur valeurs.*
- int sizePawns = 0
  *Nombre de pions.*

## 5.22.1 Description détaillée

Gestion des pions.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

## 5.22.2 Documentation des fonctions

### 5.22.2.1 bool checkPawn ( unit * *pawn* )

Vérifie que le pion est correct.

**Paramètres**

| | |
|---|---|
| *pawn* | Pion à vérifier |

**Renvoie**

Retourne vrai si le pion est correct

Définition à la ligne 37 du fichier pawns.c.

### 5.22.2.2 void createPawn ( int *nbParams*, unitName *name*, ... )

Crée un pion pour le jeu.

**Paramètres**

| | |
|---|---|
| *nbParams* | Nombre de stats pour le pion |

| | |
|---|---|
| *name* | Nom du pion |

Définition à la ligne 51 du fichier pawns.c.

#### 5.22.2.3 void initPawn ( unit * *pawn* )

Initialise un pion.

**Paramètres**

| | |
|---|---|
| *pawn* | Pion à initialiser |

Définition à la ligne 23 du fichier pawns.c.

#### 5.22.2.4 void makePawns ( )

Crée tout les pions utiles au jeu.

Crée les pions.

Définition à la ligne 109 du fichier pawns.c.

### 5.22.3 Documentation des variables

#### 5.22.3.1 unit* pawns = NULL

Pions initialisés avec leur valeurs.

Tableaux des pions.

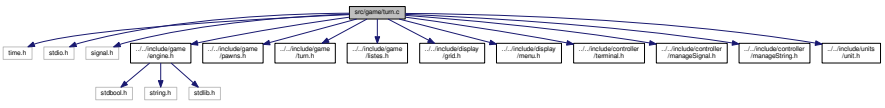
Définition à la ligne 16 du fichier pawns.c.

## 5.23 Référence du fichier src/game/turn.c

Gestion des tours.

```
#include <time.h>
#include <stdio.h>
#include <signal.h>
#include "../../include/game/engine.h"
#include "../../include/game/pawns.h"
#include "../../include/game/turn.h"
#include "../../include/game/listes.h"
#include "../../include/display/grid.h"
#include "../../include/display/menu.h"
#include "../../include/controller/terminal.h"
#include "../../include/controller/manageSignal.h"
#include "../../include/controller/manageString.h"
#include "../../include/units/unit.h"
```

Graphe des dépendances par inclusion de turn.c :



### Fonctions

- int startTurn ()
  *Débute le tour du joueur.*
- int endTurn (time_t start, int totalTime)
  *Renvoie le temps restant avant la fin du tour.*
- void changeDirection ()
  *Change la direction de l'unité*
- void playAttack ()
  *Jouer une attaque.*
- void playMove ()
  *Jouer un mouvement.*
- void passTurn ()
  *Passer tour.*
- bool hasPlay ()
  *Définis si le joueur a joué*
- void surrender ()
  *Abandonne la partie.*
- void setAction (int timeLeft)
  *Définis si l'action est à prendre en compte.*
- void playTurn ()
  *Jouer un tour.*

### Variables

- int hasMoved = 0
  *Joueur a joué*
- int hasAttacked = 0
  *Joueur a attaqué*
- int hasPassed = 0
  *Joueur a passé son tour.*
- int hasSurrender = 0
  *Joueur a abandonné la partie.*

## 5.23.1 Description détaillée

Gestion des tours.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

## 5.23.2 Documentation des fonctions

### 5.23.2.1 void changeDirection ( )

Change la direction de l'unité

Change la direction.

Définition à la ligne 57 du fichier turn.c.

### 5.23.2.2 int endTurn ( time_t *start*, int *totalTime* )

Renvoie le temps restant avant la fin du tour.

**Paramètres**

| | |
|---|---|
| *start* | Temps de début du tour |
| *totalTime* | Temps total du tour |

Définition à la ligne 44 du fichier turn.c.

#### 5.23.2.3 bool hasPlay ( )

Définis si le joueur a joué

Vérifie si une action a été effectuée.

**Renvoie**

Retourne vrai si le joueur a joué sinon faux

Définition à la ligne 245 du fichier turn.c.

#### 5.23.2.4 void passTurn ( )

Passer tour.

Passe le tour.

Définition à la ligne 235 du fichier turn.c.

#### 5.23.2.5 void playAttack ( )

Jouer une attaque.

Joue une attaque.

Définition à la ligne 104 du fichier turn.c.

#### 5.23.2.6 void playMove ( )

Jouer un mouvement.

Joue une mouvement.

Définition à la ligne 177 du fichier turn.c.

#### 5.23.2.7 void playTurn ( )

Jouer un tour.

Joue un tour.

Définition à la ligne 272 du fichier turn.c.

#### 5.23.2.8 void setAction ( int *timeLeft* )

Définis si l'action est à prendre en compte.

**Paramètres**

| | |
|---|---|
| *timeLeft* | Temps restant pour le tour |

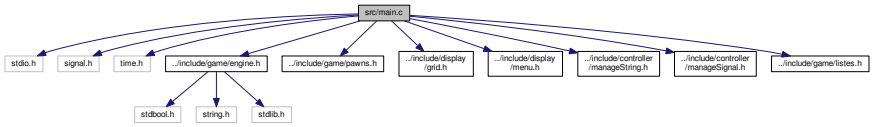
Définition à la ligne 261 du fichier turn.c.

## 5.24 Référence du fichier src/main.c

Programme principal.

```
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include "../include/game/engine.h"
#include "../include/game/pawns.h"
#include "../include/display/grid.h"
#include "../include/display/menu.h"
#include "../include/controller/manageString.h"
#include "../include/controller/manageSignal.h"
#include "../include/game/listes.h"
```

Graphe des dépendances par inclusion de main.c :




### Fonctions

— int main ()

*Programme principal.*

### 5.24.1 Description détaillée

Programme principal.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

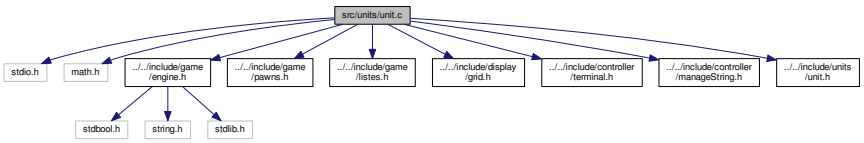
18/12/2015

## 5.25 Référence du fichier src/units/unit.c

Gestion des unités.

```
#include <stdio.h>
#include <math.h>
#include "../../include/game/engine.h"
#include "../../include/game/pawns.h"
#include "../../include/game/listes.h"
#include "../../include/display/grid.h"
#include "../../include/controller/terminal.h"
#include "../../include/controller/manageString.h"
#include "../../include/units/unit.h"
```

Graphe des dépendances par inclusion de unit.c :




## Fonctions

- void unitInit (short noP, vector coordUnit)
  *Initialise l'unité courante.*
- bool canGetPassed (unit ∗target)
  *Définis si le passage est permis.*
- bool canBlock (unit ∗target)
  *Définis si une unité peut bloquer.*
- bool canAttack (unit ∗target)
  *Définis si une unité peut attaquer.*
- bool canMove (unit ∗target)
  *Définis si une unité peut bouger.*
- bool canTeleport (unitName name)
  *Détermine si une unité peut se téléporter.*
- void heal (unitName name)
  *Soigne les unités.*
- int getSideAttacked (vector source, vector target)
  *Retourne le côté attaqué*
- void attack (vector source, vector target)
  *Attaque une unité en prenant en compte son taux de blocage.*
- bool copy (unit ∗destination, unit ∗source)
  *Copie la structure source vers la structure destination.*
- void erase (unit ∗source)
  *Efface une unité de la grille.*
- void powerBonus ()
  *Octroie un bonus de puissance.*
- void move (vector destination, vector source)
  *Déplace une unité vers la destination.*
- void setDirection (vector source, int dir)
  *Définis la direction de l'unité*
- void poison ()
  *Gère le statut empoisonnement.*
- void addEffect (vector target, unitEffect effect)
  *Ajoute un effect sur l'unité cible.*
- void minusEffect ()
  *Décrémente les effets.*
- void sleep (vector pos)
  *Endors l'unité*
- bool isSleeping (vector pos)
  *Vérifie si l'unité est endormie.*
- void recover ()
  *Réveille l'unité tour par tour.*
- bool allStatic (int numPlayer)
  *Vérifie si toutes les unités sont immobilisés.*

### 5.25.1 Description détaillée

Gestion des unités.

**Auteur**

Cousin Brandon Chaudemanche Ewen Biardeau Tristan

**Version**

v1.00

**Date**

18/12/2015

### 5.25.2 Documentation des fonctions

#### 5.25.2.1 void addEffect ( vector *target*, unitEffect *effect* )

Ajoute un effect sur l'unité cible.

Ajoute un effet à une unité

**Paramètres**

| | |
|---|---|
| *target* | Position unité cible |
| *effect* | Effet à appliquer sur la cible |

Définition à la ligne 437 du fichier unit.c.

#### 5.25.2.2 bool allStatic ( int *numPlayer* )

Vérifie si toutes les unités sont immobilisés.

Vérifie si toutes les unités sont immobilisées.

**Paramètres**

| | |
|---|---|
| *numPlayer* | Numéro du joueur |

**Renvoie**

Retourne vrai si toutes les unités sont immobilisés

Définition à la ligne 529 du fichier unit.c.

#### 5.25.2.3 void attack ( vector *source*, vector *target* )

Attaque une unité en prenant en compte son taux de blocage.

Attaque une unité

**Paramètres**

| | |
|---|---|
| *source* | Unité attaquante |
| *target* | Unité cible |

Définition à la ligne 184 du fichier unit.c.

#### 5.25.2.4 bool canAttack ( unit * *target* )

Définis si une unité peut attaquer.

Vérifie si l'unité peut attaquer.

**Paramètres**

| | |
|---|---|
| *target* | Unité à analyser |

**Renvoie**

Retourne vrai si l'unité peut attaquer

Définition à la ligne 86 du fichier unit.c.

#### 5.25.2.5 bool canBlock ( unit * *target* )

Définis si une unité peut bloquer.

Vérifie si l'unité peut bloquer.

**Paramètres**

| | |
|---|---|
| *target* | Unité à analyser |

**Renvoie**

Retourne vrai si l'unité peut bloquer

Définition à la ligne 66 du fichier unit.c.

#### 5.25.2.6 bool canGetPassed ( unit * *target* )

Définis si le passage est permis.

Vérifie si le passage est autorisé

**Paramètres**

| | |
|---|---|
| *target* | Unité à analyser |

**Renvoie**

Retourne vrai si passage autorisé

Définition à la ligne 46 du fichier unit.c.

#### 5.25.2.7 bool canMove ( unit * *target* )

Définis si une unité peut bouger.

Vérifie si l'unité peut bouger.

**Paramètres**

| | |
|---|---|
| *target* | Unité à analyser |

**Renvoie**

Retourne vraie si l'unité peut se mouvoir

Définition à la ligne 106 du fichier unit.c.

#### 5.25.2.8 bool canTeleport ( unitName *name* )

Détermine si une unité peut se téléporter.

Vérifie si l'unité peut se téléporter.

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité |

**Renvoie**

Retourne vraie si l'unité peut se déplacer

Définition à la ligne 125 du fichier unit.c.

#### 5.25.2.9 bool copy ( unit * *destination*, unit * *source* )

Copie la structure source vers la structure destination.

Copie la structure source vers destination.

**Paramètres**

| | |
|---|---|
| *destination* | Structure destination |
| *source* | Structure source |

**Renvoie**

Retourne vrai si copie bien déroulée

Définition à la ligne 236 du fichier unit.c.

#### 5.25.2.10 void erase ( unit * *source* )

Efface une unité de la grille.

**Paramètres**

| | |
|---|---|
| *source* | Source à effacer |

Définition à la ligne 271 du fichier unit.c.

#### 5.25.2.11 int getSideAttacked ( vector *source*, vector *target* )

Retourne le côté attaqué

Récupère le côté attaqué

**Paramètres**

| | |
|---|---|
| *source* | Position unité source |
| *target* | Position unité cible |

**Renvoie**

Le côté attaqué

Définition à la ligne 168 du fichier unit.c.

#### 5.25.2.12 void heal ( unitName *name* )

Soigne les unités.

**Paramètres**

| | |
|---|---|
| *name* | Nom de l'unité soignant |

Définition à la ligne 137 du fichier unit.c.

#### 5.25.2.13 bool isSleeping ( vector *pos* )

Vérifie si l'unité est endormie.

Vérifie si une unité est endormie.

**Paramètres**

| | |
|---|---|
| *pos* | Position de l'unité |

**Renvoie**

Retourne vrai si endormi

Définition à la ligne 490 du fichier unit.c.

#### 5.25.2.14 void minusEffect ( )

Décrémente les effets.

Diminue le temps de l'effet.

Définition à la ligne 455 du fichier unit.c.

#### 5.25.2.15 void move ( vector *destination,* vector *source* )

Déplace une unité vers la destination.

Bouge une unité

**Paramètres**

| | |
|---|---|
| *destination* | Destination souhaitée |
| *source* | Position de l'unité |

Définition à la ligne 351 du fichier unit.c.

#### 5.25.2.16 void poison ( )

Gère le statut empoisonnement.

Empoisonne une unité

Définition à la ligne 386 du fichier unit.c.

#### 5.25.2.17 void powerBonus ( )

Octroie un bonus de puissance.

Octroie un bonus de puissance selon certaines conditions.

Définition à la ligne 286 du fichier unit.c.

#### 5.25.2.18 void recover ( )

Réveille l'unité tour par tour.

Se réveille tour par tour.

Définition à la ligne 503 du fichier unit.c.

#### 5.25.2.19 void setDirection ( vector *source*, int *dir* )

Définis la direction de l'unité

Définis la direction d'une unité

**Paramètres**

| | |
|---|---|
| *source* | Unité à tourner |
| *dir* | Direction dans laquelle tourner |

Définition à la ligne 377 du fichier unit.c.

#### 5.25.2.20 void sleep ( vector *pos* )

Endors l'unité

Endors une unité

**Paramètres**

| | |
|---|---|
| *pos* | Position de l'unité |

Définition à la ligne 480 du fichier unit.c.

#### 5.25.2.21 void unitInit ( short *noP*, vector *coordUnit* )

Initialise l'unité courante.

Initialise une unité

**Paramètres**

| | |
|---|---|
| *noP* | Numéro du joueur |
| *coordUnit* | Coordonnées de l'unité |

Définition à la ligne 24 du fichier unit.c.

# Index