Asteroides

1.0

Generated by Doxygen 1.10.0

1 Astéroïdes Documentation 1

1.1 Description . . . 1

2 Hierarchical Index 3

2.1 Class Hierarchy . . . 3

3 Class Index 5

3.1 Class List . . . 5

4 File Index 7

4.1 File List . . . 7

5 Class Documentation 9

5.1 Animation Class Reference . . . 9

5.1.1 Detailed Description . . . 10

5.1.2 Constructor & Destructor Documentation . . . 10

5.1.2.1 Animation() [1/2] . . . 10

5.1.2.2 Animation() [2/2] . . . 10

5.1.3 Member Function Documentation . . . 11

5.1.3.1 isEnd() . . . 11

5.1.3.2 update() . . . 11

5.1.4 Member Data Documentation . . . 12

5.1.4.1 Frame . . . 12

5.1.4.2 frames . . . 12

5.1.4.3 speed . . . 12

5.1.4.4 sprite . . . 12

5.2 asteroide Class Reference . . . 12

5.2.1 Detailed Description . . . 14

5.2.2 Constructor & Destructor Documentation . . . 14

5.2.2.1 asteroide() . . . 14

5.2.3 Member Function Documentation . . . 14

5.2.3.1 update() . . . 14

5.3 Entite Class Reference . . . 15

5.3.1 Detailed Description . . . 16

5.3.2 Constructor & Destructor Documentation . . . 16

5.3.2.1 Entite() . . . 16

5.3.2.2 ~Entite() . . . 17

5.3.3 Member Function Documentation . . . 17

5.3.3.1 draw() . . . 17

5.3.3.2 getangle() . . . 17

5.3.3.3 getAnim() . . . 17

5.3.3.4 getLife() . . . 18

5.3.3.5 getName() . . . 18

5.3.3.6 getR() . . . 18

5.3.3.7 getTeam() . . . 18
5.3.3.8 getvx() . . . 19
5.3.3.9 getvy() . . . 19
5.3.3.10 getx() . . . 19
5.3.3.11 gety() . . . 19
5.3.3.12 setangle() . . . 19
5.3.3.13 setAnim() . . . 20
5.3.3.14 setLife() . . . 20
5.3.3.15 setName() . . . 20
5.3.3.16 setR() . . . 21
5.3.3.17 setTeam() . . . 21
5.3.3.18 settings() . . . 21
5.3.3.19 setvx() . . . 22
5.3.3.20 setvy() . . . 22
5.3.3.21 setx() . . . 22
5.3.3.22 sety() . . . 23
5.3.3.23 update() . . . 23
5.4 GameOverMultiScreen Class Reference . . . 23
5.4.1 Detailed Description . . . 24
5.4.2 Constructor & Destructor Documentation . . . 24
5.4.2.1 GameOverMultiScreen() . . . 24
5.4.3 Member Function Documentation . . . 24
5.4.3.1 draw() . . . 24
5.5 GameOverScreen Class Reference . . . 25
5.5.1 Detailed Description . . . 25
5.5.2 Constructor & Destructor Documentation . . . 25
5.5.2.1 GameOverScreen() . . . 25
5.5.3 Member Function Documentation . . . 26
5.5.3.1 draw() . . . 26
5.6 Menu Class Reference . . . 26
5.6.1 Detailed Description . . . 27
5.6.2 Constructor & Destructor Documentation . . . 27
5.6.2.1 Menu() . . . 27
5.6.3 Member Function Documentation . . . 27
5.6.3.1 run() . . . 27
5.7 player Class Reference . . . 28
5.7.1 Detailed Description . . . 29
5.7.2 Constructor & Destructor Documentation . . . 29
5.7.2.1 player() . . . 29
5.7.3 Member Function Documentation . . . 29
5.7.3.1 draw_bouclier() . . . 29
5.7.3.2 update() . . . 30

5.7.4 Member Data Documentation . . . 30
5.7.4.1 thrust . . . 30
5.8 TableauDesScores Class Reference . . . 30
5.8.1 Detailed Description . . . 31
5.8.2 Constructor & Destructor Documentation . . . 31
5.8.2.1 TableauDesScores() . . . 31
5.8.3 Member Function Documentation . . . 31
5.8.3.1 drawScore() . . . 31
5.8.3.2 getScore() . . . 31
5.8.3.3 increaseScore() . . . 32
5.8.3.4 reset() . . . 32
5.9 tir Class Reference . . . 32
5.9.1 Detailed Description . . . 34
5.9.2 Constructor & Destructor Documentation . . . 34
5.9.2.1 tir() . . . 34
5.9.3 Member Function Documentation . . . 34
5.9.3.1 update() . . . 34

6 File Documentation **35**

6.1 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/Animation.cpp File Reference . . . 35
6.2 Animation.cpp . . . 35
6.3 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/Animation.hpp File Reference . . . 35
6.4 Animation.hpp . . . 36
6.5 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/global_variables.cpp File Reference . . . 36
6.5.1 Variable Documentation . . . 36
6.5.1.1 DEGTORAD . . . 36
6.5.1.2 HauteurFenetre . . . 37
6.5.1.3 hoverColor . . . 37
6.5.1.4 LargeurFenetre . . . 37
6.5.1.5 NombreAsteroideStart . . . 37
6.5.1.6 NombreResidu . . . 37
6.5.1.7 normalColor . . . 37
6.5.1.8 Temps_invincible . . . 37
6.5.1.9 VolumeGeneral . . . 37
6.6 global_variables.cpp . . . 38
6.7 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/global_variables.hpp File Reference . . . 38
6.7.1 Variable Documentation . . . 38
6.7.1.1 DEGTORAD . . . 38
6.7.1.2 HauteurFenetre . . . 38
6.7.1.3 hoverColor . . . 39
6.7.1.4 LargeurFenetre . . . 39
6.7.1.5 NombreAsteroideStart . . . 39

6.7.1.6 NombreResidu . . . 39
6.7.1.7 normalColor . . . 39
6.7.1.8 Temps_invincible . . . 39
6.7.1.9 VolumeGeneral . . . 39
6.8 global_variables.hpp . . . 40
6.9 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOver.cpp File Reference . . . 40
6.10 GameOver.cpp . . . 40
6.11 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOver.hpp File Reference . . . 41
6.12 GameOver.hpp . . . 41
6.13 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOverMulti.cpp File Reference . . . 41
6.14 GameOverMulti.cpp . . . 42
6.15 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOverMulti.hpp File Reference . . . 42
6.16 GameOverMulti.hpp . . . 43
6.17 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/Menu.cpp File Reference . . . 43
6.18 Menu.cpp . . . 43
6.19 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/Menu.hpp File Reference . . . 45
6.20 Menu.hpp . . . 45
6.21 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/TableauDesScores.cpp File Reference 46
6.22 TableauDesScores.cpp . . . 46
6.23 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/TableauDesScores.hpp File Reference 46
6.24 TableauDesScores.hpp . . . 47
6.25 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/main.cpp File Reference . . . 47
6.25.1 Function Documentation . . . 48
6.25.1.1 isCollide() . . . 48
6.25.1.2 main() . . . 48
6.26 main.cpp . . . 48
6.27 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/mainpage.h File Reference . . . 55
6.28 mainpage.h . . . 55
6.29 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/asteroide.cpp File Reference . . . 55
6.30 asteroide.cpp . . . 55
6.31 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/asteroide.hpp File Reference . . . 56
6.32 asteroide.hpp . . . 56
6.33 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/Entite.cpp File Reference . . . 56
6.34 Entite.cpp . . . 57
6.35 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/Entite.hpp File Reference . . . 58
6.36 Entite.hpp . . . 58
6.37 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/player.cpp File Reference . . . 59
6.38 player.cpp . . . 59
6.39 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/player.hpp File Reference . . . 60
6.40 player.hpp . . . 60
6.41 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/tir.cpp File Reference . . . 61
6.42 tir.cpp . . . 61

6.43 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/tir.hpp File Reference 61
6.44 tir.hpp 62

Index **63**

Chapter 1

Astéroïdes Documentation

Authors

Tom Savard et Gaspard Matic

1.1 Description

Voici la documentation du code correspondant à notre projet du jeu Astéroïdes. Ce document permet une meilleur visibilité de la structure globale du projet. Cela favorise également la collaboration en rendant les fonctions annexes plus claires et lisibles. Vous retrouverez chacuns des fichiers codes sources. Ces derniers sont détaillés méthode par méthode. Enfin la documentation permet une arborescence soulignant les liens entre les objets et les héritages.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Animation . . . 9
- Entite . . . 15
  - asteroide . . . 12
  - player . . . 28
  - tir . . . 32
- GameOverMultiScreen . . . 23
- GameOverScreen . . . 25
- Menu . . . 26
- TableauDesScores . . . 30

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Animation
: Gère l'animation d'un sprite à l'aide d'une série d'images (spritesheet) 9

asteroide
: Cette objet regroupe l'ensemble des astres qui volent à l'écran. Ces derniers peuvent être d'apparences, de taille ou encore de vitesse diférentes 12

Entite
: Classe la plus générique regroupant les propriétés des différents objets 15

GameOverMultiScreen
: Classe représentant l'écran de fin de jeu multijoueur 23

GameOverScreen
: Classe représentant l'écran de fin de jeu 25

Menu
: Classe représentant le menu d'accueil du jeu 26

player
: Classe player correspondant au vaisseau du joueur. Il peut se déplacer et tirer avec son blaster 28

TableauDesScores
: Classe représentant un tableau des scores 30

tir
: Ce sont les projectiles du blaster. Ils peuvent entrer en collision avec les entités ennemies (astéroides ou autre vaisseau) 32

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

/Users/savardtom/Desktop/2A/IN204/Asteroids/code/Animation.cpp . . . 35
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/Animation.hpp . . . 35
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/global_variables.cpp . . . 36
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/global_variables.hpp . . . 38
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/main.cpp . . . 47
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/mainpage.h . . . 55
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOver.cpp . . . 40
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOver.hpp . . . 41
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOverMulti.cpp . . . 41
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOverMulti.hpp . . . 42
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/Menu.cpp . . . 43
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/Menu.hpp . . . 45
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/TableauDesScores.cpp . . . 46
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/TableauDesScores.hpp . . . 46
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/asteroide.cpp . . . 55
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/asteroide.hpp . . . 56
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/Entite.cpp . . . 56
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/Entite.hpp . . . 58
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/player.cpp . . . 59
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/player.hpp . . . 60
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/tir.cpp . . . 61
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/tir.hpp . . . 61

Chapter 5

Class Documentation

5.1 Animation Class Reference

Gère l'animation d'un sprite à l'aide d'une série d'images (spritesheet).

`#include <Animation.hpp>`

Public Member Functions

- Animation ()
  Constructeur par défaut.
- Animation (sf::Texture &t, int x, int y, int w, int h, int count, float Speed)
  Constructeur avec paramètres permettant d'initialisées les attributs.
- void update ()
  Met à jour l'animation.
- bool isEnd ()
  Vérifie si l'animation est terminée.

Public Attributes

- float Frame
- float speed
- sf::Sprite sprite
- std::vector< sf::IntRect > frames

Parameters

5.1.1 Detailed Description

Gère l'animation d'un sprite à l'aide d'une série d'images (spritesheet).

Parameters

t	Le paramètre "t" est une référence à un objet sf::Texture. C'est la texture qui sera utilisée pour les images d'animation.
x	Le paramètre "x" dans le constructeur Animation représente la coordonnée x de la position de départ des images d'animation sur la texture.
y	Le paramètre "y" dans le constructeur Animation représente la coordonnée y du coin supérieur gauche de la première image de l'animation dans la texture.
w	Le paramètre "w" dans le constructeur Animation représente la largeur de chaque image de l'animation.
h	Le paramètre "h" dans le constructeur Animation représente la hauteur de chaque image de l'animation. Il est utilisé pour définir la taille des images dans les objets sf::IntRect qui sont stockés dans le vecteur frames.
count	Le paramètre "count" dans le constructeur Animation représente le nombre d'images dans l'animation. Il détermine combien d'images seront ajoutées au vecteur "frames". Chaque cadre est un rectangle défini par sa position (x, y) et sa largeur et hauteur (w, h).
Speed	Le paramètre "Vitesse" du constructeur Animation est utilisé pour contrôler la vitesse à laquelle les images d'animation sont lues. Il détermine la rapidité avec laquelle l'animation progresse d'une image à la suivante. Une valeur plus élevée pour Vitesse accélérera la lecture de l'animation, tandis qu'une valeur inférieure la rendra plus lente.

Definition at line 32 of file Animation.hpp.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Animation() [1/2]

```
Animation::Animation ( )  [inline]
```

Constructeur par défaut.

Definition at line 42 of file Animation.hpp.

5.1.2.2 Animation() [2/2]

```
Animation::Animation (
            sf::Texture & t,
            int x,
            int y,
            int w,
            int h,
            int count,
            float Speed )
```

Constructeur avec paramètres permettant d'initialisées les attributs.

Initialise une animation avec une texture, une position de départ, des dimensions de trame, un nombre de trames, et une vitesse spécifiée.

Parameters

t	La texture à utiliser pour l'animation.
x	La position horizontale de départ de la première trame.
y	La position verticale de départ de la première trame.
w	La largeur de chaque trame.
h	La hauteur de chaque trame.
count	Le nombre total de trames dans l'animation.
Speed	La vitesse de l'animation.

Note

Ce constructeur suppose que la texture est correctement chargée et que les dimensions de la trame et le nombre de trames sont valides.

Definition at line 22 of file Animation.cpp.

5.1.3 Member Function Documentation

5.1.3.1 isEnd()

```
bool Animation::isEnd ( )
```

Vérifie si l'animation est terminée.

Cette méthode vérifie si l'animation est terminée en comparant la position actuelle de la trame avec la taille du vecteur de trames, ajustée en fonction de la vitesse de l'animation.

Returns

true si l'animation est terminée, false sinon.

Note

Cette méthode suppose que le vecteur de trames et la vitesse de l'animation sont correctement définis.

Definition at line 62 of file Animation.cpp.

5.1.3.2 update()

```
void Animation::update ( )
```

Met à jour l'animation.

Cette méthode met à jour l'animation en incrémentant le compteur de trame en fonction de la vitesse de l'animation. Elle assure également que le compteur de trame boucle pour éviter de sortir des limites du vecteur de trames.

Note

Cette méthode suppose que le sprite et le vecteur de trames sont correctement initialisés.

Definition at line 44 of file Animation.cpp.

5.1.4 Member Data Documentation

5.1.4.1 Frame

```
float Animation::Frame
```

Definition at line 35 of file Animation.hpp.

5.1.4.2 frames

```
std::vector<sf::IntRect> Animation::frames
```

Definition at line 37 of file Animation.hpp.

5.1.4.3 speed

```
float Animation::speed
```

Definition at line 35 of file Animation.hpp.

5.1.4.4 sprite

```
sf::Sprite Animation::sprite
```

Definition at line 36 of file Animation.hpp.

The documentation for this class was generated from the following files:

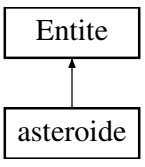
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/Animation.hpp
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/Animation.cpp

5.2 asteroide Class Reference

Cette objet regroupe l'ensemble des astres qui volent à l'écran. Ces derniers peuvent être d'apparences, de taille ou encore de vitesse diférentes.

```
#include <asteroide.hpp>
```

Inheritance diagram for asteroide:



Public Member Functions

- asteroide ()

  Constructeur par défaut.
- void update ()

  Methode permettant la mise à jour de la position de l'astéroïde.

Public Member Functions inherited from Entite

- Entite ()
- void settings (Animation &a, int X, int Y, float Angle=0, int radius=1)

  permet le paramétrage de l'entité en définissant la valeur de ses attributs.
- void draw (sf::RenderWindow &app)

  Dessine une entité sur une fenêtre SFML avec une position et une rotation spécifiques.
- float getx ()

  renvoie la valeur de la variable "x" sous forme de flottant.
- void setx (float newX)

  définit la valeur de la variable x à une nouvelle valeur.
- float gety ()

  renvoie la valeur de la variable "y" sous forme de flottant.
- void sety (float newY)

  définit la valeur de la variable "y" à une nouvelle valeur.
- float getvx ()

  renvoie la valeur de la variable "vx" sous forme de float.
- void setvx (float newVX)

  définit la valeur de la variable "vx" à une nouvelle valeur.
- float getvy ()

  renvoie la valeur de la variable "vy" sous forme de float.
- void setvy (float newVY)

  définit la valeur de la variable vy dans la classe Entite.
- float getR ()

  renvoie la valeur de la variable "R" sous forme de float.
- void setR (float newR)

  définit la valeur de la variable R dans la classe Entite.
- float getangle ()

  renvoie la valeur de la variable "angle" sous forme de float.
- void setangle (float newAngle)

  définit la valeur de la variable angle dans la classe Entite.
- int getTeam ()

  renvoie l'équipe d'une entité.
- void setTeam (int choixTeam)

  définit l'équipe d'une entité à la valeur spécifiée.
- bool getLife ()

  renvoie la valeur de la variable "life".
- void setLife (bool newlife)

  fixe la valeur de la variable "life" à la valeur passée en paramètre.
- std::string getName ()

  renvoie le nom d'une entité.
- void setName (std::string newName)

  définit le nom d'une entité sur une nouvelle valeur.

- Animation * getAnim ()

  renvoie un pointeur sur l'objet "anim" de la classe "Entite".
- void setAnim (Animation newAnim)

  définit l'animation d'une entité sur une nouvelle animation.
- virtual ~Entite ()

  Le destructeur de la classe Entite.

5.2.1 Detailed Description

Cette objet regroupe l'ensemble des astres qui volent à l'écran. Ces derniers peuvent être d'apparences, de taille ou encore de vitesse diférentes.

Definition at line 15 of file asteroide.hpp.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 asteroide()

```
asteroide::asteroide ( )  [inline]
```

Constructeur par défaut.

Le constructeur par défaut de la classe Entite. On initialise aléatoirement les vitesses vx et vy.

Definition at line 23 of file asteroide.hpp.

5.2.3 Member Function Documentation

5.2.3.1 update()

```
void asteroide::update ( )  [virtual]
```

Methode permettant la mise à jour de la position de l'astéroïde.

Met à jour la position d'un astéroïde en ajoutant sa vitesse à sa position actuelle et enroule la position autour de l'écran si elle dépasse les limites de l'écran.

Reimplemented from Entite.

Definition at line 13 of file asteroide.cpp.

The documentation for this class was generated from the following files:

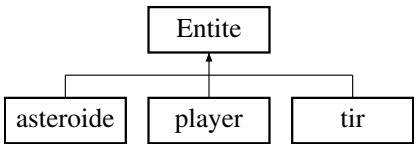
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/asteroide.hpp
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/asteroide.cpp

5.3 Entite Class Reference

Classe la plus générique regroupant les propriétés des différents objets.

`#include <Entite.hpp>`

Inheritance diagram for Entite:



Public Member Functions

- Entite ()
- void settings (Animation &a, int X, int Y, float Angle=0, int radius=1)

  permet le paramétrage de l'entité en définissant la valeur de ses attributs.
- virtual void update ()

  fonction d'espace réservé qui est destinée à être remplacée par les classes dérivées. Elle est utilisée pour mettre à jour l'état de l'entité, comme sa position, sa vitesse ou toute autre propriété susceptible de changer au fil du temps. La mise en œuvre de cette fonction variera en fonction du comportement spécifique de l'entité.
- void draw (sf::RenderWindow &app)

  Dessine une entité sur une fenêtre SFML avec une position et une rotation spécifiques.
- float getx ()

  renvoie la valeur de la variable "x" sous forme de flottant.
- void setx (float newX)

  définit la valeur de la variable x à une nouvelle valeur.
- float gety ()

  renvoie la valeur de la variable "y" sous forme de flottant.
- void sety (float newY)

  définit la valeur de la variable "y" à une nouvelle valeur.
- float getvx ()

  renvoie la valeur de la variable "vx" sous forme de float.
- void setvx (float newVX)

  définit la valeur de la variable "vx" à une nouvelle valeur.
- float getvy ()

  renvoie la valeur de la variable "vy" sous forme de float.
- void setvy (float newVY)

  définit la valeur de la variable vy dans la classe Entite.
- float getR ()

  renvoie la valeur de la variable "R" sous forme de float.
- void setR (float newR)

  définit la valeur de la variable R dans la classe Entite.
- float getangle ()

  renvoie la valeur de la variable "angle" sous forme de float.
- void setangle (float newAngle)

  définit la valeur de la variable angle dans la classe Entite.
- int getTeam ()

  renvoie l'équipe d'une entité.

- void setTeam (int choixTeam)

  définit l'équipe d'une entité à la valeur spécifiée.
- bool getLife ()

  renvoie la valeur de la variable "life".
- void setLife (bool newlife)

  fixe la valeur de la variable "life" à la valeur passée en paramètre.
- std::string getName ()

  renvoie le nom d'une entité.
- void setName (std::string newName)

  définit le nom d'une entité sur une nouvelle valeur.
- Animation * getAnim ()

  renvoie un pointeur sur l'objet "anim" de la classe "Entite".
- void setAnim (Animation newAnim)

  définit l'animation d'une entité sur une nouvelle animation.
- virtual ~Entite ()

  Le destructeur de la classe Entite.

5.3.1 Detailed Description

Classe la plus générique regroupant les propriétés des différents objets.

Cette classe possède de nombreux paramètres tous passés en privée afin de garantir une certaine sécuritée.

Parameters

team	int correspondant au numéro de l'équipe
life	bool indiquant si l'entité est détruite ou non.
name	string permettant la discrimination des entitées.
anim	Animation donnant accès à un sprite.
x	float position horizontale
y	float position verticale
vx	float vitesse horizontale
vy	float vitesse verticale
R	float rayon de la hitbox
angle	float orientation de l'entité dans le plan

Definition at line 27 of file Entite.hpp.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Entite()

```
Entite::Entite ( )  [inline]
```

Le constructeur par défaut de la classe Entite. On initialise l'entité comme étant vivante (life=1;).

Definition at line 41 of file Entite.hpp.

5.3.2.2 ~Entite()

```
Entite::~Entite ( )  [virtual]
```

Le destructeur de la classe Entite.

Ce destructeur est virtuel car il dépend de la structure des classes dérivées.

Definition at line 46 of file Entite.cpp.

5.3.3 Member Function Documentation

5.3.3.1 draw()

```
void Entite::draw (
            sf::RenderWindow & app )
```

Dessine une entité sur une fenêtre SFML avec une position et une rotation spécifiques.

Parameters

app	Le paramètre "app" est une référence à l'objet sf::RenderWindow qui représente la fenêtre de l'application. Il est utilisé pour dessiner le sprite sur la fenêtre.

Definition at line 33 of file Entite.cpp.

5.3.3.2 getangle()

```
float Entite::getangle ( )
```

renvoie la valeur de la variable "angle" sous forme de float.

Returns

la valeur de la variable "angle".

Definition at line 247 of file Entite.cpp.

5.3.3.3 getAnim()

```
Animation * Entite::getAnim ( )
```

renvoie un pointeur sur l'objet "anim" de la classe "Entite".

Returns

un pointeur vers un objet de type Animation.

Definition at line 128 of file Entite.cpp.

5.3.3.4 getLife()

```
bool Entite::getLife ( )
```

renvoie la valeur de la variable "life".

Returns

la valeur de la variable "vie".

Definition at line 87 of file Entite.cpp.

5.3.3.5 getName()

```
std::string Entite::getName ( )
```

renvoie le nom d'une entité.

Returns

une chaîne, qui est la valeur de la variable "name".

Definition at line 108 of file Entite.cpp.

5.3.3.6 getR()

```
float Entite::getR ( )
```

renvoie la valeur de la variable "R" sous forme de float.

Returns

la valeur de la variable "R".

Definition at line 227 of file Entite.cpp.

5.3.3.7 getTeam()

```
int Entite::getTeam ( )
```

renvoie l'équipe d'une entité.

Returns

la valeur de la variable "équipe".

Definition at line 66 of file Entite.cpp.

5.3.3.8 getvx()

```
float Entite::getvx ( )
```

renvoie la valeur de la variable "vx" sous forme de float.

Returns

la valeur de la variable "vx".

Definition at line 188 of file Entite.cpp.

5.3.3.9 getvy()

```
float Entite::getvy ( )
```

renvoie la valeur de la variable "vy" sous forme de float.

Returns

la valeur de la variable "vy".

Definition at line 207 of file Entite.cpp.

5.3.3.10 getx()

```
float Entite::getx ( )
```

renvoie la valeur de la variable "x" sous forme de flottant.

Returns

la valeur de la variable "x".

Definition at line 148 of file Entite.cpp.

5.3.3.11 gety()

```
float Entite::gety ( )
```

renvoie la valeur de la variable "y" sous forme de flottant.

Returns

La valeur y de l'objet Entite.

Definition at line 168 of file Entite.cpp.

5.3.3.12 setangle()

```
void Entite::setangle (
            float newAngle )
```

définit la valeur de la variable angle dans la classe Entite.

Parameters

newAngle	Le paramètre newAngle est un flottant qui représente la nouvelle valeur de la variable angle.

Definition at line 256 of file Entite.cpp.

5.3.3.13 setAnim()

```
void Entite::setAnim (
            Animation newAnim )
```

définit l'animation d'une entité sur une nouvelle animation.

Parameters

newAnim	La nouvelle animation que vous souhaitez définir pour l'entité.

Definition at line 137 of file Entite.cpp.

5.3.3.14 setLife()

```
void Entite::setLife (
            bool newlife )
```

fixe la valeur de la variable "life" à la valeur passée en paramètre.

Parameters

newlife	La nouvelle valeur de la variable "life". C'est une valeur booléenne indiquant si l'entité est vivante ou non.

Definition at line 97 of file Entite.cpp.

5.3.3.15 setName()

```
void Entite::setName (
            std::string newName )
```

définit le nom d'une entité sur une nouvelle valeur.

Parameters

newName	Le nouveau nom qui sera attribué à la variable "name".

Definition at line 117 of file Entite.cpp.

5.3.3.16 setR()

```
void Entite::setR (
            float newR )
```

définit la valeur de la variable R dans la classe Entite.

Parameters

newR	Le paramètre newR est un flottant qui représente la nouvelle valeur de la variable R Rayon de la hitbox de l'objet Entite.

Definition at line 237 of file Entite.cpp.

5.3.3.17 setTeam()

```
void Entite::setTeam (
            int choixTeam )
```

définit l'équipe d'une entité à la valeur spécifiée.

Parameters

choixTeam	Le paramètre « choixTeam » est un entier qui représente le numéro d'équipe ou identifiant de l'entité.

Definition at line 76 of file Entite.cpp.

5.3.3.18 settings()

```
void Entite::settings (
            Animation & a,
            int X,
            int Y,
            float Angle = 0,
            int radius = 1 )
```

permet le paramétrage de l'entité en définissant la valeur de ses attributs.

Parameters

a	animation associée à l'objet
X	position horizontale de l'entité
Y	position verticale
Angle	orientation dans le plan de l'objet
radius	rayon de la hitbox

La fonction « settings » définit l'animation, la position, l'angle et le rayon d'une entité.

Parameters

a	Le paramètre "a" est de type Animation et permet de définir l'animation de l'entité.
X	Le paramètre X représente la coordonnée x de la position de l'entité.
Y	Le paramètre Y représente la position verticale de l'entité sur l'écran.
Angle	Le paramètre angle représente l'angle de rotation de l'entité. Il permet de préciser l'angle de rotation initial de l'entité lors de sa création.
radius	Le paramètre radius représente le rayon de l'entité. Il est utilisé pour déterminer la taille de la hitbox ou de la zone de détection de collision de l'entité.

Definition at line 19 of file Entite.cpp.

5.3.3.19 setvx()

```
void Entite::setvx (
            float newVX )
```

définit la valeur de la variable "vx" à une nouvelle valeur.

Parameters

newVX	La nouvelle valeur de la vitesse x de l'entité.

Definition at line 197 of file Entite.cpp.

5.3.3.20 setvy()

```
void Entite::setvy (
            float newVY )
```

définit la valeur de la variable vy dans la classe Entite.

Parameters

newVY	Le paramètre newVY est un flottant qui représente la nouvelle valeur de la variable vy (vitesse verticale) de l'objet Entite.

Definition at line 217 of file Entite.cpp.

5.3.3.21 setx()

```
void Entite::setx (
            float newX )
```

définit la valeur de la variable x à une nouvelle valeur.

Parameters

newX	Le paramètre "newX" est une variable de type float qui représente la nouvelle valeur de la coordonnée x de l'entité.

Definition at line 158 of file Entite.cpp.

5.3.3.22 sety()

```
void Entite::sety (
            float newY )
```

définit la valeur de la variable "y" à une nouvelle valeur.

Parameters

newY	Le paramètre "newY" est une valeur flottante qui représente la nouvelle coordonnée y de l'entité.

Definition at line 178 of file Entite.cpp.

5.3.3.23 update()

```
void Entite::update ( )  [virtual]
```

fonction d'espace réservé qui est destinée à être remplacée par les classes dérivées. Elle est utilisée pour mettre à jour l'état de l'entité, comme sa position, sa vitesse ou toute autre propriété susceptible de changer au fil du temps. La mise en œuvre de cette fonction variera en fonction du comportement spécifique de l'entité.

Reimplemented in asteroide, player, and tir.

Definition at line 57 of file Entite.cpp.

The documentation for this class was generated from the following files:

- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/Entite.hpp
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/Entite.cpp

5.4 GameOverMultiScreen Class Reference

Classe représentant l'écran de fin de jeu multijoueur.

```
#include <GameOverMulti.hpp>
```

Public Member Functions

- GameOverMultiScreen (int gagnant)

  Constructeur de la classe GameOverMultiScreen.
- void draw (sf::RenderWindow &window)

  Dessine l'écran de fin de jeu multijoueur sur la fenêtre spécifiée.

5.4.1 Detailed Description

Classe représentant l'écran de fin de jeu multijoueur.

Cette classe gère l'affichage de l'écran de fin de jeu multijoueur, incluant le message de fin de partie, le nom du joueur gagnant, ainsi que les instructions pour redémarrer ou quitter le jeu.

Definition at line 15 of file GameOverMulti.hpp.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 GameOverMultiScreen()

```
GameOverMultiScreen::GameOverMultiScreen (
            int gagnant )
```

Constructeur de la classe GameOverMultiScreen.

Initialise l'écran de fin de jeu multijoueur avec le gagnant spécifié et configure les éléments d'affichage.

Parameters

gagnant	Le numéro du joueur gagnant. Si égal à 1, le joueur 1 (équipe rouge) est le gagnant, sinon le joueur 2 (équipe bleue) est le gagnant.

Note

Ce constructeur suppose que le fichier de police Arial est disponible dans le répertoire des ressources.

Definition at line 16 of file GameOverMulti.cpp.

5.4.3 Member Function Documentation

5.4.3.1 draw()

```
void GameOverMultiScreen::draw (
            sf::RenderWindow & window )
```

Dessine l'écran de fin de jeu multijoueur sur la fenêtre spécifiée.

Cette méthode dessine les différents éléments de l'écran de fin de jeu multijoueur (texte de fin de partie, texte indiquant le gagnant, instructions pour redémarrer ou quitter) sur la fenêtre spécifiée.

Parameters

window	La fenêtre SFML sur laquelle dessiner l'écran de fin de jeu multijoueur.

Definition at line 55 of file GameOverMulti.cpp.

The documentation for this class was generated from the following files:

- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOverMulti.hpp
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOverMulti.cpp

5.5 GameOverScreen Class Reference

Classe représentant l'écran de fin de jeu.

`#include <GameOver.hpp>`

Public Member Functions

- GameOverScreen (int score=0)

  Constructeur de la classe GameOverScreen.
- void draw (sf::RenderWindow &window)

  Dessine l'écran de fin de jeu sur la fenêtre spécifiée.

5.5.1 Detailed Description

Classe représentant l'écran de fin de jeu.

Cette classe gère l'affichage de l'écran de fin de jeu, incluant le message de fin de partie, le score du joueur, ainsi que les instructions pour redémarrer ou quitter le jeu.

Definition at line 15 of file GameOver.hpp.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 GameOverScreen()

```
GameOverScreen::GameOverScreen (
            int score = 0 )
```

Constructeur de la classe GameOverScreen.

Le code ci-dessous définit une classe GameOverScreen en C++ qui affiche un jeu sur écran avec le score et les options pour redémarrer ou quitter le jeu.

Parameters

score	Le paramètre « score » est un nombre entier qui représente le score du joueur dans la partie.

Initialise l'écran de fin de jeu avec le score fourni et configure les éléments d'affichage.

Parameters

score	Le score du joueur à afficher.

Note

Ce constructeur suppose que le fichier de police Arial est disponible dans le répertoire des ressources.

Definition at line 23 of file GameOver.cpp.

5.5.3 Member Function Documentation

5.5.3.1 draw()

```
void GameOverScreen::draw (
            sf::RenderWindow & window )
```

Dessine l'écran de fin de jeu sur la fenêtre spécifiée.

Cette méthode dessine les différents éléments de l'écran de fin de jeu (texte de fin de partie, score, instructions pour redémarrer ou quitter) sur la fenêtre spécifiée.

Parameters

window	La fenêtre SFML sur laquelle dessiner l'écran de fin de jeu.

Definition at line 61 of file GameOver.cpp.

The documentation for this class was generated from the following files:

- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOver.hpp
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOver.cpp

5.6 Menu Class Reference

Classe représentant le menu d'accueil du jeu.

```
#include <Menu.hpp>
```

Public Member Functions

- Menu (sf::RenderWindow &window, sf::Music &music)

  Constructeur du Menu. Il fait appel à deux arguments afin de pouvoir effectuer l'affichage et avoir une musique de fond.
- std::tuple< std::string, float > run ()

5.6.1 Detailed Description

Classe représentant le menu d'accueil du jeu.

Cette classe gère l'affichage et l'interaction avec le menu du jeu permettant de sélectionner le mode de jeu ou encore le volume sonore.

Definition at line 16 of file Menu.hpp.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Menu()

```
Menu::Menu (
            sf::RenderWindow & window,
            sf::Music & music )  [inline]
```

Constructeur du Menu. Il fait appel à deux arguments afin de pouvoir effectuer l'affichage et avoir une musique de fond.

Parameters

window	pointeur sf::RenderWindow qui permet l'affichage des éléments.
music	pointeur sf::Music qui donne accès à la musique chargée.

Definition at line 24 of file Menu.hpp.

5.6.3 Member Function Documentation

5.6.3.1 run()

```
std::tuple< std::string, float > Menu::run ( )
```

La fonction run dans la classe Menu affiche un écran de menu avec des boutons et gère les interactions de l'utilisateur telles que cliquer sur des boutons et faire glisser un indicateur de volume.

Returns

La fonction `run()` renvoie un tuple contenant une chaîne et un float. La chaîne représente l'action sélectionnée par l'utilisateur (par exemple, "JouerSolo", "JouerMulti", "Quitter"), et le float représente le niveau de volume sélectionné par l'utilisateur.

Definition at line 19 of file Menu.cpp.

The documentation for this class was generated from the following files:

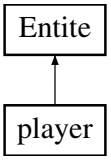
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/Menu.hpp
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/Menu.cpp

5.7 player Class Reference

Classe player correspondant au vaisseau du joueur. Il peut se déplacer et tirer avec son blaster.

`#include <player.hpp>`

Inheritance diagram for player:



Public Member Functions

- player ()
- void update ()

  Mise en oeuvre spécifique à la classe player de la fonction générique update.
- void draw_bouclier (sf::RenderWindow &app, bool invincible)

  permet une invincibilité temporaire au lancement.

Public Member Functions inherited from Entite

- Entite ()
- void settings (Animation &a, int X, int Y, float Angle=0, int radius=1)

  permet le paramétrage de l'entité en définissant la valeur de ses attributs.
- void draw (sf::RenderWindow &app)

  Dessine une entité sur une fenêtre SFML avec une position et une rotation spécifiques.
- float getx ()

  renvoie la valeur de la variable "x" sous forme de flottant.
- void setx (float newX)

  définit la valeur de la variable x à une nouvelle valeur.
- float gety ()

  renvoie la valeur de la variable "y" sous forme de flottant.
- void sety (float newY)

  définit la valeur de la variable "y" à une nouvelle valeur.
- float getvx ()

  renvoie la valeur de la variable "vx" sous forme de float.
- void setvx (float newVX)

  définit la valeur de la variable "vx" à une nouvelle valeur.
- float getvy ()

  renvoie la valeur de la variable "vy" sous forme de float.
- void setvy (float newVY)

  définit la valeur de la variable vy dans la classe Entite.
- float getR ()

  renvoie la valeur de la variable "R" sous forme de float.
- void setR (float newR)

  définit la valeur de la variable R dans la classe Entite.
- float getangle ()

renvoie la valeur de la variable "angle" sous forme de float.

- void setangle (float newAngle)

  définit la valeur de la variable angle dans la classe Entite.
- int getTeam ()

  renvoie l'équipe d'une entité.
- void setTeam (int choixTeam)

  définit l'équipe d'une entité à la valeur spécifiée.
- bool getLife ()

  renvoie la valeur de la variable "life".
- void setLife (bool newlife)

  fixe la valeur de la variable "life" à la valeur passée en paramètre.
- std::string getName ()

  renvoie le nom d'une entité.
- void setName (std::string newName)

  définit le nom d'une entité sur une nouvelle valeur.
- Animation * getAnim ()

  renvoie un pointeur sur l'objet "anim" de la classe "Entite".
- void setAnim (Animation newAnim)

  définit l'animation d'une entité sur une nouvelle animation.
- virtual ~Entite ()

  Le destructeur de la classe Entite.

Public Attributes

- bool thrust

5.7.1 Detailed Description

Classe player correspondant au vaisseau du joueur. Il peut se déplacer et tirer avec son blaster.

Definition at line 15 of file player.hpp.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 player()

```
player::player ( )  [inline]
```

Definition at line 21 of file player.hpp.

5.7.3 Member Function Documentation

5.7.3.1 draw_bouclier()

```
void player::draw_bouclier (
            sf::RenderWindow & app,
            bool invincible )
```

permet une invincibilité temporaire au lancement.

Parameters

app	pointeur sf::RenderWindow qui permet l'affichage
invincible	bool true = invincible, false = vulnérable

Definition at line 41 of file player.cpp.

5.7.3.2 update()

```
void player::update ( )  [virtual]
```

Mise en oeuvre spécifique à la classe player de la fonction générique update.

Actualisation de la vitesse si le boost est activé sinon léger ralentissement

Reimplemented from Entite.

Definition at line 13 of file player.cpp.

5.7.4 Member Data Documentation

5.7.4.1 thrust

```
bool player::thrust
```

Definition at line 19 of file player.hpp.

The documentation for this class was generated from the following files:

- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/player.hpp
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/player.cpp

5.8 TableauDesScores Class Reference

Classe représentant un tableau des scores.

```
#include <TableauDesScores.hpp>
```

Public Member Functions

- TableauDesScores ()
- void increaseScore (int points)

  Incrémente le score de la valeur spécifiée.
- void drawScore (sf::RenderWindow &window)

  Dessine le score sur la fenêtre spécifiée.
- void reset ()

  Réinitialise le score à zéro.
- int getScore ()

  Obtient le score actuel.

5.8.1 Detailed Description

Classe représentant un tableau des scores.

Cette classe gère un tableau des scores dans le jeu, permettant d'incrémenter le score, de le réinitialiser et de le dessiner sur une fenêtre SFML.

Definition at line 15 of file TableauDesScores.hpp.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 TableauDesScores()

```
TableauDesScores::TableauDesScores ( )  [inline]
```

Definition at line 20 of file TableauDesScores.hpp.

5.8.3 Member Function Documentation

5.8.3.1 drawScore()

```
void TableauDesScores::drawScore (
            sf::RenderWindow & window )
```

Dessine le score sur la fenêtre spécifiée.

Cette méthode dessine le score actuel sur la fenêtre SFML spécifiée.

Parameters

window	La fenêtre SFML sur laquelle dessiner le score.

Definition at line 27 of file TableauDesScores.cpp.

5.8.3.2 getScore()

```
int TableauDesScores::getScore ( )
```

Obtient le score actuel.

Returns

Le score actuel.

Definition at line 53 of file TableauDesScores.cpp.

5.8.3.3 increaseScore()

```
void TableauDesScores::increaseScore (
            int points )
```

Incrémente le score de la valeur spécifiée.

Le code définit une classe appelée "TableauDesScores" qui gère un compteur de scores et fournit des fonctions pour augmenter, afficher, réinitialiser et récupérer le score.

Parameters

points	Le paramètre « points » est un nombre entier qui représente le nombre de points à ajouter au score.
points	La valeur à ajouter au score actuel.

Definition at line 18 of file TableauDesScores.cpp.

5.8.3.4 reset()

```
void TableauDesScores::reset ( )
```

Réinitialise le score à zéro.

Cette méthode réinitialise le score à zéro, remettant ainsi le compteur de points à sa valeur initiale.

Definition at line 46 of file TableauDesScores.cpp.

The documentation for this class was generated from the following files:

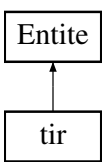
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/TableauDesScores.hpp
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/TableauDesScores.cpp

5.9 tir Class Reference

Ce sont les projectiles du blaster. Ils peuvent entrer en collision avec les entités ennemies (astéroides ou autre vaisseau).

```
#include <tir.hpp>
```

Inheritance diagram for tir:



Public Member Functions

- tir ()

  Constructeur par défaut de la classe tir.
- void update ()

  Mise en oeuvre spécifique à la classe tir de la fonction générique update.

Public Member Functions inherited from Entite

- Entite ()
- void settings (Animation &a, int X, int Y, float Angle=0, int radius=1)

  permet le paramétrage de l'entité en définissant la valeur de ses attributs.
- void draw (sf::RenderWindow &app)

  Dessine une entité sur une fenêtre SFML avec une position et une rotation spécifiques.
- float getx ()

  renvoie la valeur de la variable "x" sous forme de flottant.
- void setx (float newX)

  définit la valeur de la variable x à une nouvelle valeur.
- float gety ()

  renvoie la valeur de la variable "y" sous forme de flottant.
- void sety (float newY)

  définit la valeur de la variable "y" à une nouvelle valeur.
- float getvx ()

  renvoie la valeur de la variable "vx" sous forme de float.
- void setvx (float newVX)

  définit la valeur de la variable "vx" à une nouvelle valeur.
- float getvy ()

  renvoie la valeur de la variable "vy" sous forme de float.
- void setvy (float newVY)

  définit la valeur de la variable vy dans la classe Entite.
- float getR ()

  renvoie la valeur de la variable "R" sous forme de float.
- void setR (float newR)

  définit la valeur de la variable R dans la classe Entite.
- float getangle ()

  renvoie la valeur de la variable "angle" sous forme de float.
- void setangle (float newAngle)

  définit la valeur de la variable angle dans la classe Entite.
- int getTeam ()

  renvoie l'équipe d'une entité.
- void setTeam (int choixTeam)

  définit l'équipe d'une entité à la valeur spécifiée.
- bool getLife ()

  renvoie la valeur de la variable "life".
- void setLife (bool newlife)

  fixe la valeur de la variable "life" à la valeur passée en paramètre.
- std::string getName ()

  renvoie le nom d'une entité.
- void setName (std::string newName)

  définit le nom d'une entité sur une nouvelle valeur.

- Animation * getAnim ()

  renvoie un pointeur sur l'objet "anim" de la classe "Entite".
- void setAnim (Animation newAnim)

  définit l'animation d'une entité sur une nouvelle animation.
- virtual ~Entite ()

  Le destructeur de la classe Entite.

5.9.1 Detailed Description

Ce sont les projectiles du blaster. Ils peuvent entrer en collision avec les entités ennemies (astéroides ou autre vaisseau).

Definition at line 15 of file tir.hpp.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 tir()

```
tir::tir ( )  [inline]
```

Constructeur par défaut de la classe tir.

Definition at line 23 of file tir.hpp.

5.9.3 Member Function Documentation

5.9.3.1 update()

```
void tir::update ( )  [virtual]
```

Mise en oeuvre spécifique à la classe tir de la fonction générique update.

Les tirs ont une trajectoire rectiligne uniforme. Ils sont détruits une fois sortis de l'écran.

Reimplemented from Entite.

Definition at line 13 of file tir.cpp.

The documentation for this class was generated from the following files:

- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/tir.hpp
- /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/tir.cpp

Chapter 6

File Documentation

6.1 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/Animation.cpp File Reference

```
#include "Animation.hpp"
#include "global_variables.hpp"
```

6.2 Animation.cpp

Go to the documentation of this file.

```
00001                                                // Bibliothèques //
00002 #include "Animation.hpp"
00003 #include "global_variables.hpp"
00004
00005                                                // Code Principal //
00022 Animation::Animation (sf::Texture &t, int x, int y, int w, int h, int count, float Speed)
00023 {
00024   Frame = 0;
00025   speed = Speed;
00026
00027   for (int i=0;i<count;i++)
00028   frames.push_back( sf::IntRect(x+i*w, y, w, h)  );
00029
00030   sprite.setTexture(t);
00031   sprite.setOrigin(w/2,h/2);
00032   sprite.setTextureRect(frames[0]);
00033 }
00034
00044 void Animation::update()
00045 {
00046   Frame += speed;
00047   int n = frames.size();
00048   if (Frame >= n) Frame -= n;
00049   if (n>0) sprite.setTextureRect( frames[int(Frame)] );
00050 }
00051
00062 bool Animation::isEnd()
00063 {
00064   return Frame+speed>=frames.size();
00065 }
```

6.3 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/Animation.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
```

Classes

- class Animation

  Gère l'animation d'un sprite à l'aide d'une série d'images (spritesheet).

6.4 Animation.hpp

Go to the documentation of this file.

```
00001 #ifndef ANIMATION_HPP
00002 #define ANIMATION_HPP
00003                                                  // Bibliothéques //
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include <iostream>
00007
00008                                                  // Code Principal //
00032 class Animation
00033 {
00034    public:
00035    float Frame, speed;
00036    sf::Sprite sprite;
00037    std::vector<sf::IntRect> frames;
00042    Animation(){}
00043
00048    Animation (sf::Texture &t, int x, int y, int w, int h, int count, float Speed);
00049
00050
00051    void update();
00052
00053    bool isEnd();
00054
00055 };
00056
00057 #endif // ANIMATION_HPP
```

6.5 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/global_↩ variables.cpp File Reference

```
#include "global_variables.hpp"
```

Variables

- sf::Color normalColor = sf::Color::White
- sf::Color hoverColor = sf::Color(200, 200, 200)
- const int LargeurFenetre = 1200
- const int HauteurFenetre = 800
- float DEGTORAD = 0.0174533f
- float VolumeGeneral = 1.f
- int NombreAsteroideStart = 15
- int NombreResidu = 2
- const unsigned int Temps_invincible = 1500

6.5.1 Variable Documentation

6.5.1.1 DEGTORAD

```
float DEGTORAD = 0.0174533f
```

Definition at line 11 of file global_variables.cpp.

6.5.1.2 HauteurFenetre

```
const int HauteurFenetre = 800
```

Definition at line 9 of file global_variables.cpp.

6.5.1.3 hoverColor

```
sf::Color hoverColor = sf::Color(200, 200, 200)
```

Definition at line 6 of file global_variables.cpp.

6.5.1.4 LargeurFenetre

```
const int LargeurFenetre = 1200
```

Definition at line 8 of file global_variables.cpp.

6.5.1.5 NombreAsteroideStart

```
int NombreAsteroideStart = 15
```

Definition at line 14 of file global_variables.cpp.

6.5.1.6 NombreResidu

```
int NombreResidu = 2
```

Definition at line 15 of file global_variables.cpp.

6.5.1.7 normalColor

```
sf::Color normalColor = sf::Color::White
```

Definition at line 5 of file global_variables.cpp.

6.5.1.8 Temps_invincible

```
const unsigned int Temps_invincible = 1500
```

Definition at line 16 of file global_variables.cpp.

6.5.1.9 VolumeGeneral

```
float VolumeGeneral = 1.f
```

Definition at line 12 of file global_variables.cpp.

6.6 global_variables.cpp

Go to the documentation of this file.

```
00001 /* Ce fichier contient l'ensemble des constantes du jeu. Cela permet une centralisation des données,
00002 supprime la redondance et facilite les modifications*/
00003 #include "global_variables.hpp"
00004
00005 sf::Color normalColor = sf::Color::White;
00006 sf::Color hoverColor = sf::Color(200, 200, 200);
00007
00008 const int LargeurFenetre = 1200;
00009 const int HauteurFenetre = 800;
00010
00011 float DEGTORAD = 0.0174533f;
00012 float VolumeGeneral = 1.f;
00013
00014 int NombreAsteroideStart = 15;
00015 int NombreResidu = 2;
00016 const unsigned int Temps_invincible = 1500;
```

6.7 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/global_↩ variables.hpp File Reference

```
#include <SFML/Graphics.hpp>
```

Variables

- sf::Color normalColor
- sf::Color hoverColor
- const int LargeurFenetre
- const int HauteurFenetre
- float DEGTORAD
- float VolumeGeneral
- int NombreAsteroideStart
- int NombreResidu
- const unsigned int Temps_invincible

6.7.1 Variable Documentation

6.7.1.1 DEGTORAD

```
float DEGTORAD  [extern]
```

Definition at line 11 of file global_variables.cpp.

6.7.1.2 HauteurFenetre

```
const int HauteurFenetre  [extern]
```

Definition at line 9 of file global_variables.cpp.

6.7.1.3 hoverColor

```
sf::Color hoverColor  [extern]
```

Definition at line 6 of file global_variables.cpp.

6.7.1.4 LargeurFenetre

```
const int LargeurFenetre  [extern]
```

Definition at line 8 of file global_variables.cpp.

6.7.1.5 NombreAsteroideStart

```
int NombreAsteroideStart  [extern]
```

Definition at line 14 of file global_variables.cpp.

6.7.1.6 NombreResidu

```
int NombreResidu  [extern]
```

Definition at line 15 of file global_variables.cpp.

6.7.1.7 normalColor

```
sf::Color normalColor  [extern]
```

Definition at line 5 of file global_variables.cpp.

6.7.1.8 Temps_invincible

```
const unsigned int Temps_invincible  [extern]
```

Definition at line 16 of file global_variables.cpp.

6.7.1.9 VolumeGeneral

```
float VolumeGeneral  [extern]
```

Definition at line 12 of file global_variables.cpp.

6.8 global_variables.hpp

Go to the documentation of this file.

```
00001 #ifndef GLOBAL_VARIABLES_HPP
00002 #define GLOBAL_VARIABLES_HPP
00003
00004 #include <SFML/Graphics.hpp>
00005
00006 extern sf::Color normalColor;
00007 extern sf::Color hoverColor;
00008
00009 extern const int LargeurFenetre;
00010 extern const int HauteurFenetre;
00011
00012 extern float DEGTORAD;
00013 extern float VolumeGeneral;
00014
00015 extern int NombreAsteroideStart;
00016 extern int NombreResidu;
00017
00018 extern const unsigned int Temps_invincible;
00019
00020 #endif // GLOBAL_VARIABLES_HPP
```

6.9 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOver.cpp File Reference

```
#include "GameOver.hpp"
#include "../global_variables.hpp"
```

6.10 GameOver.cpp

Go to the documentation of this file.

```
00001
00008                                           // Bibliothèques //
00009 #include "GameOver.hpp"
00010 #include "../global_variables.hpp"
00011
00012                                           // Code Principal //
00013
00023 GameOverScreen::GameOverScreen(int score) {
00024
00025     if (!font.loadFromFile("Ressources/police/arial/arial.ttf")) {
00026     }
00027
00028     gameOverText.setFont(font);
00029     gameOverText.setCharacterSize(50);
00030     gameOverText.setFillColor(sf::Color::Red);
00031     gameOverText.setString("Game Over");
00032     gameOverText.setPosition((LargeurFenetre - gameOverText.getGlobalBounds().width) / 2,
      HauteurFenetre/2 - 100);
00033
00034     scoreText.setFont(font);
00035     scoreText.setCharacterSize(30);
00036     scoreText.setFillColor(sf::Color::White);
00037     scoreText.setString("Score: " + std::to_string(score)); // Convertir le score en chaîne de
      caractères
00038     scoreText.setPosition((LargeurFenetre - scoreText.getGlobalBounds().width) / 2, HauteurFenetre /
      2);
00039
00040     retryText.setFont(font);
00041     retryText.setCharacterSize(30);
00042     retryText.setFillColor(sf::Color::White);
00043     retryText.setString("Appuyer sur R pour redemarrer");
00044     retryText.setPosition((LargeurFenetre - retryText.getGlobalBounds().width) / 2, HauteurFenetre/2 +
      50);
00045
00046     quitText.setFont(font);
00047     quitText.setCharacterSize(30);
00048     quitText.setFillColor(sf::Color::White);
```

```
00049     quitText.setString("Appuyer sur Q pour quitter");
00050     quitText.setPosition((LargeurFenetre - quitText.getGlobalBounds().width) / 2, HauteurFenetre/2 +
      100);
00051 }
00052
00061 void GameOverScreen::draw(sf::RenderWindow &window) {
00062     window.draw(gameOverText);
00063     window.draw(scoreText);
00064     window.draw(retryText);
00065     window.draw(quitText);
00066 }
00067
00068
```

6.11 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOver.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
```

Classes

- class GameOverScreen

  Classe représentant l'écran de fin de jeu.

6.12 GameOver.hpp

Go to the documentation of this file.

```
00001 #ifndef GAMEOVER_HPP
00002 #define GAMEOVER_HPP
00003                                                 // Bibliothéques //
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include <iostream>
00007
00008                                                 // Code Principal //
00015 class GameOverScreen {
00016 private:
00017     sf::Font font;
00018     sf::Text gameOverText;
00019     sf::Text retryText;
00020     sf::Text quitText;
00021     sf::Text scoreText;
00022
00023 public:
00024     GameOverScreen(int score =0);
00025
00026     void draw(sf::RenderWindow &window);
00027 };
00028
00029 #endif // GAMEOVER_HPP
```

6.13 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOverMulti.cpp File Reference

```
#include "GameOverMulti.hpp"
#include "../global_variables.hpp"
```

6.14 GameOverMulti.cpp

Go to the documentation of this file.

```
00001                                                   // Bibliothèques //
00002 #include "GameOverMulti.hpp"
00003 #include "../global_variables.hpp"
00004
00005                                                   // Code Principal //
00006
00016 GameOverMultiScreen::GameOverMultiScreen(int gagnant) {
00017
00018     if (!font.loadFromFile("Ressources/police/arial/arial.ttf")) {
00019     }
00020
00021     gameOverText.setFont(font);
00022     gameOverText.setCharacterSize(50);
00023     gameOverText.setFillColor(sf::Color::Red);
00024     gameOverText.setString("Game Over");
00025     gameOverText.setPosition((LargeurFenetre - gameOverText.getGlobalBounds().width) / 2,
      HauteurFenetre/2 - 100);
00026
00027     gagnantText.setFont(font);
00028     gagnantText.setCharacterSize(30);
00029     gagnantText.setFillColor(sf::Color::White);
00030     if (gagnant == 1) {gagnantText.setString("Victoire du Joueur 1 (Equipe Rouge)");}
00031     else {gagnantText.setString("Victoire du Joueur 2 (Equipe Bleue)");}
00032     gagnantText.setPosition((LargeurFenetre - gagnantText.getGlobalBounds().width) / 2, HauteurFenetre
      / 2);
00033
00034     retryText.setFont(font);
00035     retryText.setCharacterSize(30);
00036     retryText.setFillColor(sf::Color::White);
00037     retryText.setString("Appuyer sur R pour redemarrer");
00038     retryText.setPosition((LargeurFenetre - retryText.getGlobalBounds().width) / 2, HauteurFenetre/2 +
      50);
00039
00040     quitText.setFont(font);
00041     quitText.setCharacterSize(30);
00042     quitText.setFillColor(sf::Color::White);
00043     quitText.setString("Appuyer sur Q pour quitter");
00044     quitText.setPosition((LargeurFenetre - quitText.getGlobalBounds().width) / 2, HauteurFenetre/2 +
      100);
00045 }
00046
00055 void GameOverMultiScreen::draw(sf::RenderWindow &window) {
00056     window.draw(gameOverText);
00057     window.draw(gagnantText);
00058     window.draw(retryText);
00059     window.draw(quitText);
00060 }
00061
00062
```

6.15 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/↩ GameOverMulti.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
```

Classes

- class GameOverMultiScreen

  Classe représentant l'écran de fin de jeu multijoueur.

6.16 GameOverMulti.hpp

Go to the documentation of this file.

```
00001 #ifndef GAMEOVERMULTI_HPP
00002 #define GAMEOVERMULTI_HPP
00003                                                 // Bibliothéques //
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include <iostream>
00007
00008                                                 // Code Principal //
00015 class GameOverMultiScreen {
00016 private:
00017     sf::Font font;
00018     sf::Text gameOverText;
00019     sf::Text retryText;
00020     sf::Text quitText;
00021     sf::Text gagnantText;
00022
00023 public:
00024     GameOverMultiScreen(int gagnant);
00025
00026     void draw(sf::RenderWindow &window);
00027 };
00028
00029 #endif // GAMEOVERMULTI_HPP
```

6.17 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/↩ Menu.cpp File Reference

```
#include "Menu.hpp"
#include "../global_variables.hpp"
#include <tuple>
```

6.18 Menu.cpp

Go to the documentation of this file.

```
00001                                                 // Bibliothèques //
00002 #include "Menu.hpp"
00003 #include "../global_variables.hpp"
00004 #include <tuple>
00005
00006                                                 // Variables //
00007
00008                                                 // Code Principal //
00009
00019 std::tuple <std::string, float> Menu::run() {
00020     sf::Font font;
00021     if (!font.loadFromFile("Ressources/police/arial/arial.ttf")) {
00022         std::cerr « "Failed to load font." « std::endl;
00023         return std::make_tuple("Quitter", 0);
00024     }
00025
00026     sf::Text title("Asteroid Game", font, 50);
00027     title.setFillColor(sf::Color::White);
00028     title.setStyle(sf::Text::Bold);
00029     title.setPosition(200, 100);
00030
00031     sf::Text BoutonSolo("Solo", font, 30);
00032     BoutonSolo.setFillColor(sf::Color::White);
00033     BoutonSolo.setPosition(300, 250);
00034
00035     sf::Text BoutonMultijoueur("Multijoueur", font, 30);
00036     BoutonMultijoueur.setFillColor(sf::Color::White);
00037     BoutonMultijoueur.setPosition(300, 300);
00038
00039     sf::Text BoutonQuitter("Quitter", font, 30);
00040     BoutonQuitter.setFillColor(sf::Color::White);
00041     BoutonQuitter.setPosition(300, 400);
```

```
00042
00043     sf::RectangleShape volumeBar(sf::Vector2f(200, 20));
00044     volumeBar.setFillColor(sf::Color(200, 200, 200)); // Couleur gris clair
00045     volumeBar.setPosition(window.getSize().x - 250, window.getSize().y - 50);
00046
00047     sf::CircleShape volumeIndicator(10);
00048     volumeIndicator.setFillColor(sf::Color::White);
00049     volumeIndicator.setOutlineColor(sf::Color::Black);
00050     volumeIndicator.setOutlineThickness(2);
00051     volumeIndicator.setOrigin(5, 5);
00052     volumeIndicator.setPosition(window.getSize().x - 250 + volume * 200, window.getSize().y - 45);
00053
00054     bool isDragging = false;
00055
00056     // Charger le son pour le clic sur le bouton
00057     sf::SoundBuffer clickSoundBuffer;
00058     if (!clickSoundBuffer.loadFromFile("Ressources/audio/Bouton2.wav")) {std::cerr « "Failed to load
    click sound file" « std::endl;}
00059     sf::Sound clickSound;
00060     clickSound.setBuffer(clickSoundBuffer);
00061
00062     // Test pour le pictogramme de son
00063     sf::Texture SonTexture;
00064     if (!SonTexture.loadFromFile("Ressources/image/Pictogramme_son.png")) {
00065         std::cerr « "Failed to load logo image." « std::endl;
00066         // Gérer l'échec du chargement de l'image
00067     }
00068     sf::Sprite PictogrammeSonSprite(SonTexture);
00069     PictogrammeSonSprite.setPosition(window.getSize().x - 290, window.getSize().y - 55);
00070     float scaleFactor = 0.1f;
00071     PictogrammeSonSprite.setScale(scaleFactor, scaleFactor);
00072
00073     // Test pour le fond d'écran
00074     sf::Texture tbackground;
00075     tbackground.loadFromFile("Ressources/image/Menu_Background.png");
00076     tbackground.setSmooth(true);
00077     sf::Sprite background(tbackground);
00078
00079
00080     while (window.isOpen()) {
00081         sf::Event event;
00082         while (window.pollEvent(event)) {
00083             if (event.type == sf::Event::Closed)
00084                 window.close();
00085             else if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button ==
    sf::Mouse::Left) {
00086                 sf::Vector2f mousePos = window.mapPixelToCoords(sf::Vector2i(event.mouseButton.x,
    event.mouseButton.y));
00087                 if (BoutonSolo.getGlobalBounds().contains(mousePos)) {
00088                     // Action lorsque le bouton "Solo" est cliqué
00089                     std::cout « "Bouton Solo cliqué" « std::endl;
00090                     clickSound.play(); // Jouer le son de clic
00091                     window.close(); // Fermer la fenêtre lorsque le bouton "Jouer" est cliqué
00092                     return std::make_tuple("JouerSolo", volume);
00093                 }
00094                 else if (BoutonMultijoueur.getGlobalBounds().contains(mousePos)) {
00095                     // Action lorsque le bouton "Multijoueur" est cliqué
00096                     std::cout « "Bouton Multijoueur cliqué" « std::endl;
00097                     clickSound.play(); // Jouer le son de clic
00098                     window.close(); // Fermer la fenêtre lorsque le bouton "Jouer" est cliqué
00099                     return std::make_tuple("JouerMulti", volume);
00100                 }
00101                 else if (BoutonQuitter.getGlobalBounds().contains(mousePos)) {
00102                     // Action lorsque le bouton "Quitter" est cliqué
00103                     std::cout « "Bouton quitter cliqué!" « std::endl;
00104                     clickSound.play(); // Jouer le son de clic
00105                     window.close(); // Fermer la fenêtre lorsque le bouton "Quitter" est cliqué
00106                     return std::make_tuple("Quitter", volume);
00107                 }
00108                 else if (volumeIndicator.getGlobalBounds().contains(mousePos)) {
00109                     isDragging = true;
00110                 }
00111             }
00112             else if (event.type == sf::Event::MouseButtonReleased && event.mouseButton.button ==
    sf::Mouse::Left) {
00113                 isDragging = false;
00114             }
00115             else if (event.type == sf::Event::MouseMoved) {
00116                 sf::Vector2f mousePos = window.mapPixelToCoords(sf::Vector2i(event.mouseMove.x,
    event.mouseMove.y));
00117                 // Gestion du survol des boutons
00118                 if (BoutonSolo.getGlobalBounds().contains(mousePos))
    {BoutonSolo.setFillColor(hoverColor);}
00119                 else {BoutonSolo.setFillColor(normalColor);}
00120                 if (BoutonMultijoueur.getGlobalBounds().contains(mousePos))
    {BoutonMultijoueur.setFillColor(hoverColor);}
00121                 else {BoutonMultijoueur.setFillColor(normalColor);}
```

```
00122                 if (BoutonQuitter.getGlobalBounds().contains(mousePos))
      {BoutonQuitter.setFillColor(hoverColor);}
00123                 else {BoutonQuitter.setFillColor(normalColor);}
00124                 // Déplacement du bouton de volume si l'utilisateur est en train de le glisser
00125                 if (isDragging) {
00126                     float newVolume = (mousePos.x - (window.getSize().x - 250)) / 200.0f;
00127                     if (newVolume < 0)
00128                         newVolume = 0;
00129                     else if (newVolume > 1)
00130                         newVolume = 1;
00131                     volume = newVolume;
00132                     music.setVolume(volume * 100); // Réglez le volume sur une échelle de 0 à 100
00133                     volumeIndicator.setPosition(window.getSize().x - 250 + volume * 200,
      window.getSize().y - 45);
00134                 }
00135             }
00136         }
00137         window.clear();
00138         window.draw(background);
00139         window.draw(title);
00140         window.draw(BoutonSolo);
00141         window.draw(BoutonMultijoueur);
00142         window.draw(BoutonQuitter);
00143         window.draw(PictogrammeSonSprite);
00144         window.draw(volumeBar);
00145         window.draw(volumeIndicator);
00146         window.display();
00147     }
00148
00149 return std::make_tuple("Quitter", volume);
00150 }
```

6.19 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/Menu.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include "../global_variables.hpp"
#include <tuple>
```

Classes

- class Menu

  Classe représentant le menu d'accueil du jeu.

6.20 Menu.hpp

Go to the documentation of this file.

```
00001 #ifndef MENU_HPP
00002 #define MENU_HPP
00003                                           // Bibliothèques //
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include <iostream>
00007 #include "../global_variables.hpp"
00008 #include <tuple>
00009
00010                                           // Code Principal //
00016 class Menu {
00017 public:
00024     Menu(sf::RenderWindow& window, sf::Music& music) : window(window), music(music) {}
00025
00026     std::tuple <std::string, float> run();
00027
00028 private:
```

```
00029     sf::RenderWindow& window;
00030     sf::Music& music;
00031     float volume = VolumeGeneral; // Volume sonore initial
00032 };
00033
00034 #endif // MENU_HPP
```

6.21 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/↩ TableauDesScores.cpp File Reference

```
#include "TableauDesScores.hpp"
#include "../global_variables.hpp"
```

6.22 TableauDesScores.cpp

Go to the documentation of this file.

```
00001
00008                                           // Bibliothèques //
00009 #include "TableauDesScores.hpp"
00010 #include "../global_variables.hpp"
00011
00012                                           // Code Principal //
00018 void TableauDesScores::increaseScore(int points) {score += points;}
00019
00027 void TableauDesScores::drawScore(sf::RenderWindow& window) {
00028     sf::Font font;
00029     if (!font.loadFromFile("Ressources/police/arial/arial.ttf")) {
00030         // Gérer l'erreur de chargement de la police
00031         std::cerr « "Failed to load font file" « std::endl;
00032         return;
00033     }
00034     sf::Text scoreText("Score: " + std::to_string(score), font, 30);
00035     scoreText.setFillColor(sf::Color::White);
00036     scoreText.setPosition(10, 10);
00037
00038     window.draw(scoreText); // Dessiner le compteur de points à l'écran
00039 }
00040
00046 void TableauDesScores::reset(){score = 0;}
00047
00053 int TableauDesScores::getScore(){return score;}
00054
```

6.23 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/↩ TableauDesScores.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
```

Classes

- class TableauDesScores

  Classe représentant un tableau des scores.

6.24 TableauDesScores.hpp

Go to the documentation of this file.

```
00001 #ifndef TABLEAUDESSCORES_HPP
00002 #define TABLEAUDESSCORES_HPP
00003                                                 // Bibliothéques //
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include <iostream>
00007
00008                                                 // Code Principal //
00015 class TableauDesScores {
00016     private:
00017         int score;
00018
00019     public:
00020         TableauDesScores() : score(0) {} // Initialiser le compteur de points à zéro
00021
00022         void increaseScore(int points);
00023
00024         void drawScore(sf::RenderWindow& window);
00025
00026         void reset();
00027
00028         int getScore();
00029 };
00030
00031 #endif // TABLEAUDESSCORES_HPP
```

6.25 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/main.cpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include <vector>
#include <memory>
#include <list>
#include <math.h>
#include <time.h>
#include "interface/Menu.hpp"
#include "interface/GameOver.hpp"
#include "interface/GameOverMulti.hpp"
#include "interface/TableauDesScores.hpp"
#include "global_variables.hpp"
#include "Animation.hpp"
#include "objet/Entite.hpp"
#include "objet/asteroide.hpp"
#include "objet/tir.hpp"
#include "objet/player.hpp"
```

Functions

- bool isCollide (Entite *a, Entite *b)

  Prend deux entités et vérifient si il y a collision.

- int main ()

  C'est la boucle principale. Elle gère en particulier l'interaction avec le joueur et fait appel aux autres fichiers sources.

6.25.1 Function Documentation

6.25.1.1 isCollide()

```
bool isCollide (
            Entite * a,
            Entite * b )
```

Prend deux entités et vérifient si il y a collision.

Parameters

a	une entité
b	une autre entité

Returns

true si il y a collision

false sinon

Definition at line 36 of file main.cpp.

6.25.1.2 main()

```
int main ( )
```

C'est la boucle principale. Elle gère en particulier l'interaction avec le joueur et fait appel aux autres fichiers sources.

Returns

int

Definition at line 50 of file main.cpp.

6.26 main.cpp

Go to the documentation of this file.

```
00001
00007
00008 #include <SFML/Graphics.hpp>
00009 #include <SFML/Audio.hpp>
00010 #include <iostream>
00011 #include <vector>
00012 #include <memory>
00013 #include <list>
00014 #include<math.h>
00015 #include<time.h>
00016
00017 #include "interface/Menu.hpp"
00018 #include "interface/GameOver.hpp"
00019 #include "interface/GameOverMulti.hpp"
00020 #include "interface/TableauDesScores.hpp"
00021 #include "global_variables.hpp"
00022 #include "Animation.hpp"
00023 #include "objet/Entite.hpp"
00024 #include "objet/asteroide.hpp"
```

```
00025 #include "objet/tir.hpp"
00026 #include "objet/player.hpp"
00027
00036 bool isCollide(Entite *a,Entite *b)
00037
00038 {
00039   return (b->getx() - a->getx())*(b->getx() - a->getx())+
00040          (b->gety() - a->gety())*(b->gety() - a->gety())<
00041          (a->getR() + b->getR())*(a->getR() + b->getR());
00042 }
00043
00044
00050 int main() {
00051 // Music du menu d'accueil
00052 sf::Music musicHub;
00053 if (!musicHub.openFromFile("Ressources/audio/Dofus2.wav")) {
00054     throw std::runtime_error("Failed to load musicHub file");
00055 }
00056 musicHub.play();
00057 // Création d'une fenetre pour le menu d'accueil et lancement.
00058 sf::RenderWindow window(sf::VideoMode(800, 600), "Asteroid Game");
00059 TableauDesScores LeScore;
00060 Menu menu(window, musicHub);
00061 std::tuple <std::string, float> action = menu.run(); // On récupère les choix effectués par le joueur
      (mode de jeu et volume sonore)
00062 float VolumeSelected=std::get<1>(action);
00063 musicHub.stop();
00064
00065 // Chargement des effets sonores ( Faire un fichier séparé qui gère le son ?) ///
00066 // Blaster
00067 sf::SoundBuffer shootSoundBuffer;
00068 sf::Sound shootSound;
00069 if (!shootSoundBuffer.loadFromFile("Ressources/audio/tir.wav")) {
00070     throw std::runtime_error("Failed to load shootSound");}
00071 shootSound.setBuffer(shootSoundBuffer);
00072 int shootVolume = 8*VolumeSelected;
00073 shootSound.setVolume(shootVolume);
00074 //Explosion Vaisseau
00075 sf::SoundBuffer DeathSoundBuffer;
00076 sf::Sound DeathSound;
00077 if (!DeathSoundBuffer.loadFromFile("Ressources/audio/death.wav")) {
00078     throw std::runtime_error("Failed to load DeathSound");}
00079 DeathSound.setBuffer(DeathSoundBuffer);
00080 int DeathVolume = 100*VolumeSelected;
00081 DeathSound.setVolume(DeathVolume);
00082 // Effet sonore click bouton
00083 sf::SoundBuffer clickSoundBuffer;
00084 if (!clickSoundBuffer.loadFromFile("Ressources/audio/Bouton2.wav")) {
00085     throw std::runtime_error("Failed to load ClickSound");}
00086 sf::Sound clickSound;
00087 clickSound.setBuffer(clickSoundBuffer);
00088
00089 // Boucle de jeu en Solo //
00090 while(std::get<0>(action) == "JouerSolo"){
00091     // Chargement et lancement de la musique du jeu
00092     sf::Music musicGame;
00093     if (!musicGame.openFromFile("Ressources/audio/Glory.wav")) {
00094         throw std::runtime_error("Failed to load musicGame");}
00095     int musicVolume = 70*VolumeSelected;
00096     musicGame.setVolume(musicVolume);
00097     musicGame.play();
00098
00099     srand(time(0));
00100     // Génération de la fenetre de jeu
00101     sf::RenderWindow app(sf::VideoMode(LargeurFenetre, HauteurFenetre), "Asteroides!");
00102     app.setFramerateLimit(60);
00103
00104     //Chargement des textures pour les animations
00105     sf::Texture t1,t2,t3,t4,t5,t10,t11,t12,t13,t14,t21,t22;
00106     try {
00107         if (!t1.loadFromFile("Ressources/animation/spaceship.png")) {
00108             throw std::runtime_error("Failed to load spaceship.png");
00109         }
00110         if (!t2.loadFromFile("Ressources/image/Space_Background.png")) {
00111             throw std::runtime_error("Failed to load Space_Background.png");
00112         }
00113         if (!t3.loadFromFile("Ressources/animation/explosions/type_C.png")) {
00114             throw std::runtime_error("Failed to load type_C.png");
00115         }
00116         if (!t4.loadFromFile("Ressources/animation/explosions/type_B.png")) {
00117             throw std::runtime_error("Failed to load type_B.png");
00118         }
00119         if (!t5.loadFromFile("Ressources/animation/fire_red.png")) {
00120             throw std::runtime_error("Failed to load fire_red.png");
00121         }
00122         if (!t10.loadFromFile("Ressources/animation/TerranWet.png")) {
00123             throw std::runtime_error("Failed to load TerranWet.png");
```

```
00124         }
00125         if (!t11.loadFromFile("Ressources/animation/NoAtmosphere.png")) {
00126             throw std::runtime_error("Failed to load NoAtmosphere.png");
00127         }
00128         if (!t12.loadFromFile("Ressources/animation/LavaPlanet.png")) {
00129             throw std::runtime_error("Failed to load LavaPlanet.png");
00130         }
00131         if (!t13.loadFromFile("Ressources/animation/Star.png")) {
00132             throw std::runtime_error("Failed to load Star.png");
00133         }
00134         if (!t14.loadFromFile("Ressources/animation/Blackhole.png")) {
00135             throw std::runtime_error("Failed to load Blackhole.png");
00136         }
00137         if (!t21.loadFromFile("Ressources/animation/asteroide1.png")) {
00138             throw std::runtime_error("Failed to load asteroide1.png");
00139         }
00140         if (!t22.loadFromFile("Ressources/animation/asteroide2.png")) {
00141             throw std::runtime_error("Failed to load asteroide2.png");
00142         }
00143     } catch (const std::runtime_error& e) {
00144         std::cerr « "Erreur : " « e.what() « std::endl;
00145         return 1;
00146     }
00147     t1.setSmooth(true);
00148     t2.setSmooth(true);
00149     sf::Sprite background(t2);
00150     // Param : (texture, x = position de départ des images sur la texture, y = pareil en vertical,
00151     // w = largeur image, h = hauteur image, count = nbr d'image, speed = vitesse de transition)
00152     Animation sPlayer(t1, 40,0,40,40, 1, 0);
00153     Animation sPlayer_go(t1, 40,40,40,40, 1, 0);
00154     Animation sExplosion(t3, 0,0,256,256, 48, 0.5);
00155     Animation sExplosion_ship(t4, 0,0,192,192, 64, 0.5);
00156     Animation sBullet(t5, 0,0,32,64, 16, 0.8);
00157     Animation sTerranWet(t10, 0, 0, 50, 50, 50, 0.2);
00158     Animation sNoAtmosphere(t11, 0, 0, 50, 50, 50, 0.2);
00159     Animation sLavaPlanet(t12, 0, 0, 50, 50, 50, 0.2);
00160     Animation sStar(t13, 0, 0, 50, 50, 50, 0.2);
00161     Animation sBlackhole(t14, 0, 0, 50, 50, 50, 0.2);
00162     Animation sAsteroide1(t21, 0, 0, 50, 50, 50, 0.2);
00163     Animation sAsteroide2(t22, 0, 0, 50, 50, 50, 0.2);
00164     int asteroideWidth = 25; // à modifier en fonction de la texture choisie
00165     int petitAsteroideWidth = 15; // à modifier en fonction de la texture choisie
00166
00167
00168     // Création d'une liste contenant l'ensemble des entités courantes.
00169     std::list<Entite*> entities;
00170
00171     // Génère un nombre d'astéroïdes désiré au commencement.
00172     for(int i=0;i<NombreAsteroideStart;i++)
00173     {
00174         asteroide *a = new asteroide();
00175         int NumeroTexture = std::rand() % 4;
00176         if (NumeroTexture == 0) {a->settings(sTerranWet, rand()%LargeurFenetre, rand()%HauteurFenetre,
      rand()%360, asteroideWidth);}
00177         else if (NumeroTexture == 1) {a->settings(sNoAtmosphere, rand()%LargeurFenetre,
      rand()%HauteurFenetre, rand()%360, asteroideWidth);}
00178         else if (NumeroTexture == 2) {a->settings(sLavaPlanet, rand()%LargeurFenetre,
      rand()%HauteurFenetre, rand()%360, asteroideWidth);}
00179         else {a->settings(sStar, rand()%LargeurFenetre, rand()%HauteurFenetre, rand()%360,
      petitAsteroideWidth);}
00180         entities.push_back(a);
00181     }
00182
00183     // Création d'une instance player
00184     player *p = new player();
00185     p->settings(sPlayer,200,200,0,20);
00186     entities.push_back(p);
00187
00188     // Variable pour le temps écoulé
00189     auto Temps_start = std::chrono::steady_clock::now();
00190     bool invincible = true;
00191     // Boucle principale du jeu qui tourne tant que le jeu est en cours. Elle gère l'affichage des
      entités à chaque frame //
00192     while (app.isOpen())
00193     {
00194         auto Temps_current = std::chrono::steady_clock::now();
00195         auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(Temps_current -
      Temps_start);
00196         if (duration.count() >= Temps_invincible) {
00197             invincible = false;}
00198
00199         /* Le code ci-dessous gère les événements. Il utilise une boucle while pour
00200         interroger en permanence les événements. */
00201         sf::Event event;
00202         while (app.pollEvent(event))
00203         {
00204             if (event.type == sf::Event::Closed)
```

```
00205                 app.close();
00206
00207             if (event.type == sf::Event::KeyPressed)
00208                 if (event.key.code == sf::Keyboard::Space)
00209                 {
00210                     tir *b = new tir();
00211                     b->settings(sBullet,p->getx(),p->gety(),p->getangle(),10);
00212                     entities.push_back(b);
00213                     shootSound.play();
00214                 }
00215         }
00216         // Contrôles pour le déplacement du joueur
00217         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) p->setangle(p->getangle() + 3);
00218         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) p->setangle(p->getangle() - 3);
00219         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) p->thrust=true;
00220         else p->thrust=false;
00221
00222         // Itération sur les entités pour vérifier les collisions
00223         for(auto a:entities)
00224             for(auto b:entities)
00225             {
00226                 if (a->getName()=="asteroide" && b->getName()=="tir")
00227                     if ( isCollide(a,b) )
00228                         {   // mise à false de leur vie pour une futur suppression de l'entité.
00229                             a->setLife(false);
00230                             b->setLife(false);
00231                             // Génère une entité avec la texture explosion à l'endroit de la collision
00232                             Entite *e = new Entite();
00233                             e->settings(sExplosion,a->getx(),a->gety());
00234                             e->setName("explosion");
00235                             entities.push_back(e);
00236
00237                             // Ici le score correspond au nombre d'astéroïde détruit.
00238                             LeScore.increaseScore(1);
00239
00240                             // On scinde les gros astéroïdes en NombreResidu astéroïdes de petite
      taille.
00241                             for(int i=0; i<NombreResidu; i++)
00242                             {
00243                                 if (a->getR()==asteroideWidth){
00244                                     Entite *e = new asteroide();
00245                                     int NumeroTextureAsteroide = std::rand() % 2;
00246                                     if (NumeroTextureAsteroide == 0)
      {e->settings(sAsteroide1,a->getx(),a->gety(),rand()%360,petitAsteroideWidth);}
00247                                     else
      {e->settings(sAsteroide2,a->getx(),a->gety(),rand()%360,petitAsteroideWidth);}
00248                                     entities.push_back(e);
00249                                 }
00250                             }
00251
00252                         }
00253
00254                     if (a->getName()=="player" && b->getName()=="asteroide" && !invincible)
00255                         if ( isCollide(a,b) )
00256                         {
00257                             b->setLife(false);
00258                             // Génère une entité avec la texture explosion à l'endroit de la collision
00259                             Entite *e = new Entite();
00260                             e->settings(sExplosion_ship,a->getx(),a->gety());
00261                             e->setName("explosion");
00262                             entities.push_back(e);
00263                             // Le vaisseau est détruit. La fenêtre de jeu se ferme.
00264                             app.close();
00265                             DeathSound.play();
00266                             musicGame.stop();
00267                         }
00268                 }
00269
00270             // En fonction de l'entrée de la touche Boost, la texture du vaisseau est modifiée afin
      d'afficher les réacteurs.
00271             if (p->thrust)  p->setAnim(sPlayer_go);
00272             else   p->setAnim(sPlayer);
00273
00274
00275             for(auto e:entities)
00276                 if (e->getName()=="explosion")
00277                 if (e->getAnim()->isEnd()) e->setLife(false);
00278
00279                 if (rand()%150==0)
00280                 {
00281                     asteroide *a = new asteroide();
00282                     int NumeroTexture = std::rand() % 4;
00283                     if (NumeroTexture == 0) {a->settings(sTerranWet, 0, rand()%HauteurFenetre,
      rand()%360, asteroideWidth);}
00284                     else if (NumeroTexture == 1) {a->settings(sNoAtmosphere, 0, rand()%HauteurFenetre,
      rand()%360, asteroideWidth);}
00285                     else if (NumeroTexture == 2) {a->settings(sLavaPlanet, 0, rand()%HauteurFenetre,
```

```
    rand()%360, asteroideWidth);}
00286                     else {a->settings(sStar, 0, rand()%HauteurFenetre, rand()%360,
    petitAsteroideWidth);}
00287                     entities.push_back(a);
00288                 }
00289
00290             for(auto i=entities.begin();i!=entities.end();)
00291             {
00292                 Entite *e = *i;
00293
00294                 e->update();
00295                 e->getAnim()->update();
00296
00297                 if (e->getLife()==false) {i=entities.erase(i); delete e;}
00298                 else i++;
00299             }
00300
00301     // On efface les affichages précédents puis on charge chacune des entitées courantes ainsi que le
    score. On affiche la fenetre.
00302     app.clear();
00303     app.draw(background);
00304     for(auto i:entities) i->draw(app);
00305     p->draw_bouclier(app, invincible);
00306     LeScore.drawScore(app);
00307     app.display();
00308     } // Fin boucle while app.isOpen()
00309
00310     // Le joueur vient de terminer sa partie en modeSolo. Il faut maintenant afficher l'écran GameOver
00311     sf::RenderWindow window(sf::VideoMode(LargeurFenetre, HauteurFenetre), "Asteroid Game Over");
00312     int scoreFinal = LeScore.getScore();
00313     GameOverScreen gameOverScreen(scoreFinal);
00314     bool actionRecu = false;
00315     // On effectue une boucle while pour l'affichage dynamique de la fenêtre.
00316     while (window.isOpen() && !actionRecu) {
00317         sf::Event event;
00318         while (window.pollEvent(event)) {
00319             if (event.type == sf::Event::Closed)
00320             window.close();
00321             else if (event.type == sf::Event::KeyPressed) {
00322                 if (event.key.code == sf::Keyboard::R) { // Le Joueur veut rejouer
00323                     clickSound.play();
00324                     std::get<0>(action) = "JouerSolo";
00325                     actionRecu = true;
00326                     LeScore.reset();
00327                     std::cout « "Restarting the game..." « std::endl;}
00328                 else if (event.key.code == sf::Keyboard::Q) { // Le Joueur veut quitter
00329                     clickSound.play();
00330                     std::get<0>(action) = "Quitter";
00331                     actionRecu = true;
00332                     std::cout « "Exiting the game..." « std::endl;
00333                     window.close();}
00334             }
00335         }
00336         // On nettoie la fenêtre puis génère les nouveaux éléments. On affiche la fenêtre.
00337         window.clear();
00338         gameOverScreen.draw(window);
00339         window.display();
00340     }
00341 }// fin de la boucle while Action==Jouer
00342
00343 // Boucle de jeu en Multi
00344 while(std::get<0>(action) == "JouerMulti"){
00345
00346     // Chargement et lancement de la musique du jeu
00347     sf::Music musicGame;
00348     if (!musicGame.openFromFile("Ressources/audio/Glory.wav")) {
00349         throw std::runtime_error("Failed to load musicGame");}
00350     int musicVolume = 70*VolumeSelected;
00351     musicGame.setVolume(musicVolume);
00352     musicGame.play();
00353
00354     srand(time(0));
00355     // Génération de la fenetre de jeu
00356     sf::RenderWindow app(sf::VideoMode(LargeurFenetre, HauteurFenetre), "Asteroides!");
00357     app.setFramerateLimit(60);
00358
00359
00360     //Chargement des textures pour les animations
00361     sf::Texture t1,t2,t3,t4,t5,t6;
00362     try {
00363         if (!t1.loadFromFile("Ressources/animation/spaceship.png")) {
00364             throw std::runtime_error("Failed to load spaceship.png");
00365         }
00366         if (!t2.loadFromFile("Ressources/image/Space_Background.png")) {
00367             throw std::runtime_error("Failed to load Space_Background.png");
00368         }
00369         if (!t3.loadFromFile("Ressources/animation/spaceshipBlue.png")) {
```

```
00370             throw std::runtime_error("Failed to load spaceshipBlue.png");
00371         }
00372         if (!t4.loadFromFile("Ressources/animation/explosions/type_B.png")) {
00373             throw std::runtime_error("Failed to load type_B.png");
00374         }
00375         if (!t5.loadFromFile("Ressources/animation/fire_red.png")) {
00376             throw std::runtime_error("Failed to load fire_red.png");
00377         }
00378         if (!t6.loadFromFile("Ressources/animation/fire_blue.png")) {
00379             throw std::runtime_error("Failed to load fire_blue.png");
00380         }
00381 
00382     } catch (const std::runtime_error& e) {
00383         std::cerr « "Erreur : " « e.what() « std::endl;
00384         return 1;
00385         }
00386     t1.setSmooth(true);
00387     t2.setSmooth(true);
00388     sf::Sprite background(t2);
00389     Animation sBulletRed(t5, 0,0,32,64, 16, 0.8);
00390     Animation sBulletBlue(t6, 0,0,32,64, 16, 0.8);
00391     Animation sPlayer1(t1, 40,0,40,40, 1, 0);
00392     Animation sPlayer2(t3, 40,0,40,40, 1, 0);
00393     Animation sPlayer1_go(t1, 40,40,40,40, 1, 0);
00394     Animation sPlayer2_go(t3, 40,40,40,40, 1, 0);
00395     Animation sExplosion_ship(t4, 0,0,192,192, 64, 0.5);
00396 
00397 
00398     // Création d'une liste contenant l'ensemble des entités courantes.
00399     std::list<Entite*> entities;
00400 
00401     // Création des deux instances player
00402     player *p1 = new player();
00403     p1->setTeam(1);
00404     p1->settings(sPlayer1,LargeurFenetre-200,HauteurFenetre-200,180,20);
00405     entities.push_back(p1);
00406 
00407     player *p2 = new player();
00408     p2->setTeam(2);
00409     p2->settings(sPlayer2,200,200,0,20);
00410     entities.push_back(p2);
00411 
00412     int gagnant = -1; // Permet de récupérer l'équipe victorieuse
00413 
00414 
00415     // Boucle principale du jeu qui tourne tant que le jeu est en cours. Elle gère l'affichage des
      entités à chaque frame ///
00416     while (app.isOpen())
00417     {
00418         sf::Event event;
00419         while (app.pollEvent(event))
00420         {
00421             if (event.type == sf::Event::Closed){
00422                 app.close();}
00423 
00424             if (event.type == sf::Event::KeyPressed)
00425                 {
00426                 if (event.key.code == sf::Keyboard::SemiColon)
00427                 {   //Tir du joueur 1
00428                     tir *t1 = new tir();
00429                     t1->setTeam(1);
00430                     t1->settings(sBulletRed,p1->getx(),p1->gety(),p1->getangle(),10);
00431                     entities.push_back(t1);
00432                     shootSound.play();
00433                 }
00434                 if (event.key.code == sf::Keyboard::X)
00435                 {   //Tir du joueur 2
00436                     tir *t2 = new tir();
00437                     t2->setTeam(2);
00438                     t2->settings(sBulletBlue,p2->getx(),p2->gety(),p2->getangle(),10);
00439                     entities.push_back(t2);
00440                     shootSound.play();
00441                 }
00442             }
00443         }
00444 
00445         // Contrôles du joueur 1
00446         if(sf::Keyboard::isKeyPressed(sf::Keyboard::M)) {p1->setangle(p1->getangle() + 3);}
00447         if(sf::Keyboard::isKeyPressed(sf::Keyboard::K)) {p1->setangle(p1->getangle() - 3);}
00448         if(sf::Keyboard::isKeyPressed(sf::Keyboard::O)) {p1->thrust=true;}
00449         else {p1->thrust=false;}
00450         // Contrôles du joueur 2
00451         if(sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {p2->setangle(p2->getangle() + 3);}
00452         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {p2->setangle(p2->getangle() - 3);}
00453         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Z)) {p2->thrust=true;}
00454         else {p2->thrust=false;}
00455 
```

```
00456
00457         for(auto a:entities)
00458         {
00459             for(auto b:entities)
00460             {
00461                 if (a->getName()=="player" && b->getName()=="tir")
00462                 {
00463                     if (isCollide(a,b) && a->getTeam()!=b->getTeam())
00464                     {
00465                         b->setLife(false);
00466                         a->setLife(false);
00467
00468                         Entite *e = new Entite();
00469                         e->settings(sExplosion_ship,a->getx(),a->gety());
00470                         e->setName("explosion");
00471                         entities.push_back(e);
00472
00473                         app.close();
00474                         DeathSound.play();
00475                         musicGame.stop();
00476
00477                         gagnant = b->getTeam();
00478                     }
00479                 }
00480             }//fin for b:entities
00481         }
00482
00483         // En fonction de l'entrée de la touche Boost, la texture du vaisseau est modifiée afin
      d'afficher les réacteurs.
00484         if (p1->thrust)  p1->setAnim(sPlayer1_go);
00485         else   p1->setAnim(sPlayer1);
00486         if (p2->thrust)  p2->setAnim(sPlayer2_go);
00487         else   p2->setAnim(sPlayer2);
00488
00489         // Les explosions n'ont qu'une seule apparition. Une fois généré, elles sont immédiatement
      supprimées.
00490         for(auto e:entities)
00491             if (e->getName()=="explosion")
00492             if (e->getAnim()->isEnd()) e->setLife(0);
00493
00494
00495         /* Le code ci-dessus parcourt un conteneur appelé « entités ». Il met à jour
00496         chaque entité du conteneur. Si la variable « vie » d'une entité est fausse, elle est
00497         supprimée du conteneur et est supprimée. */
00498         for(auto i=entities.begin();i!=entities.end();)
00499         {
00500             Entite *e = *i;
00501
00502             e->update();
00503             e->getAnim()->update();
00504
00505             if (e->getLife()==false) {i=entities.erase(i); delete e;}
00506             else i++;
00507         }
00508
00509         // On nettoie la fenêtre puis affiche les entitées courantes.
00510         app.clear();
00511         app.draw(background);
00512         for(auto i:entities) i->draw(app);
00513         app.display();
00514
00515     }//fin while app.isOpen()
00516
00517     // Un des vaisseaux est détruit. Le gagnant peut admirer la preuve de sa victoire sur l'écran
      GameOver
00518     sf::RenderWindow window(sf::VideoMode(LargeurFenetre, HauteurFenetre), "Asteroid Game Over");
00519     GameOverMultiScreen gameOverMultiScreen(gagnant);
00520     bool actionRecu = false;
00521     // On effectue une boucle while pour avoir une fenêtre interactive à travers les entrées claviers.
00522     while (window.isOpen() && !actionRecu)
00523     {
00524         sf::Event event;
00525         while (window.pollEvent(event))
00526         {
00527             if (event.type == sf::Event::Closed)
00528             window.close();
00529             else if (event.type == sf::Event::KeyPressed) {
00530                 if (event.key.code == sf::Keyboard::R) {
00531                     clickSound.play();
00532                     std::get<0>(action) = "JouerMulti";
00533                     actionRecu = true;
00534                     std::cout « "Restarting the game..." « std::endl;}
00535                 else if (event.key.code == sf::Keyboard::Q) {
00536                     clickSound.play();
00537                     std::get<0>(action) = "Quitter";
00538                     actionRecu = true;
00539                     std::cout « "Exiting the game..." « std::endl;
```

```
00540                     window.close();}
00541             }
00542         }
00543         // Nettoyage de la fenêtre et affichage des nouveaux éléments.
00544         window.clear();
00545         gameOverMultiScreen.draw(window);
00546         window.display();
00547     }
00548 }// fin de la boucle while Action==JouerMulti
00549
00550 // ! Les 2 modes ont beaucoup de code en commun. Une optimisation est certainement possible. Une
      partie du code pourrait être contenu directement dans le fichier source GameOver et GameOverMulti. Ces
      deux interfaces pourraient dérivées d'une même classe de base.
00551
00552 return 0;
00553 }
```

6.27 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/mainpage.h File Reference

6.28 mainpage.h

Go to the documentation of this file.

```
00001 // Ce fichier permet d'organiser la page principale de la documentation.
00002
00015
```

6.29 /Users/savardtom/Desktop/2A/IN204/↩ Asteroids/code/objet/asteroide.cpp File Reference

```
#include "asteroide.hpp"
#include "Entite.hpp"
#include "../Animation.hpp"
#include "../global_variables.hpp"
```

6.30 asteroide.cpp

Go to the documentation of this file.

```
00001                                                   // Bibliothèques //
00002 #include "asteroide.hpp"
00003 #include "Entite.hpp"
00004 #include "../Animation.hpp"
00005 #include "../global_variables.hpp"
00006
00007                                                   // Code Principal //
00008
00013 void asteroide::update()
00014 {
00015     this->setx(this->getx() + this->getvx());
00016     this->sety(this->gety() + this->getvy());
00017
00018     if (this->getx()>LargeurFenetre) this->setx(0);  if (this->getx()<0) this->setx(LargeurFenetre);
00019     if (this->gety()>HauteurFenetre) this->sety(0);  if (this->gety()<0) this->sety(HauteurFenetre);
00020 }
00021
```

6.31 /Users/savardtom/Desktop/2A/IN204/↩ Asteroids/code/objet/asteroide.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include "../Animation.hpp"
#include "Entite.hpp"
```

Classes

- class asteroide

  Cette objet regroupe l'ensemble des astres qui volent à l'écran. Ces derniers peuvent être d'apparences, de taille ou encore de vitesse diférentes.

6.32 asteroide.hpp

Go to the documentation of this file.

```
00001 #ifndef ASTEROIDE_HPP
00002 #define ASTEROIDE_HPP
00003                                                   // Bibliothéques //
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include <iostream>
00007 #include "../Animation.hpp"
00008 #include "Entite.hpp"
00009
00010                                                   // Code Principal //
00015 class asteroide: public Entite
00016 {
00017    public:
00023    asteroide()
00024    {
00025      this->setvx(rand()%8-4);
00026      this->setvy(rand()%8-4);
00027      this->setName("asteroide");
00028    }
00033    void update();
00034 };
00035
00036 #endif // ASTEROIDE_HPP
```

6.33 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/↩ Entite.cpp File Reference

```
#include "Entite.hpp"
#include "../Animation.hpp"
#include "../global_variables.hpp"
```

6.34 Entite.cpp

Go to the documentation of this file.

```
00001                                                   // Bibliothèques //
00002 #include "Entite.hpp"
00003 #include "../Animation.hpp"
00004 #include "../global_variables.hpp"
00005
00006                                                   // Code Principal //
00007
00019 void Entite::settings(Animation &a,int X,int Y,float Angle,int radius)
00020 {
00021     anim = a;
00022     x=X; y=Y;
00023     angle = Angle;
00024     R = radius;
00025 }
00026
00033 void Entite::draw(sf::RenderWindow &app)
00034 {
00035     anim.sprite.setPosition(x,y);
00036     anim.sprite.setRotation(angle+90);
00037     app.draw(anim.sprite);
00038 }
00039
00040
00046 Entite::~Entite() {
00047 }
00048
00049
00057 void Entite::update(){
00058 };
00059
00060
00066 int Entite::getTeam()
00067 {
00068     return team;
00069 }
00076 void Entite::setTeam(int choixTeam)
00077 {
00078     team = choixTeam;
00079 }
00080
00081
00087 bool Entite::getLife()
00088 {
00089     return life;
00090 }
00097 void Entite::setLife(bool newlife)
00098 {
00099     life = newlife;
00100 }
00101
00102
00108 std::string Entite::getName()
00109 {
00110     return name;
00111 }
00117 void Entite::setName(std::string newName)
00118 {
00119     name = newName;
00120 }
00121
00122
00128 Animation* Entite::getAnim()
00129 {
00130     return &anim;
00131 }
00137 void Entite::setAnim(Animation newAnim)
00138 {
00139     anim = newAnim;
00140 }
00141
00142
00148 float Entite::getx()
00149 {
00150     return x;
00151 }
00158 void Entite::setx(float newX)
00159 {
00160     x = newX;
00161 }
00162
00168 float Entite::gety()
00169 {
```

```
00170     return y;
00171 }
00178 void Entite::sety(float newY)
00179 {
00180     y = newY;
00181 }
00182
00188 float Entite::getvx()
00189 {
00190     return vx;
00191 }
00197 void Entite::setvx(float newVX)
00198 {
00199     vx = newVX;
00200 }
00201
00207 float Entite::getvy()
00208 {
00209     return vy;
00210 }
00217 void Entite::setvy(float newVY)
00218 {
00219     vy = newVY;
00220 }
00221
00227 float Entite::getR()
00228 {
00229     return R;
00230 }
00237 void Entite::setR(float newR)
00238 {
00239     R = newR;
00240 }
00241
00247 float Entite::getangle()
00248 {
00249     return angle;
00250 }
00256 void Entite::setangle(float newAngle)
00257 {
00258     angle = newAngle;
00259 }
```

6.35 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/↩ Entite.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include "../Animation.hpp"
```

Classes

- class Entite

  Classe la plus générique regroupant les propriétés des différents objets.

6.36 Entite.hpp

Go to the documentation of this file.

```
00001 #ifndef ENTITE_HPP
00002 #define ENTITE_HPP
00003                                           // Bibliothéques //
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include <iostream>
00007 #include "../Animation.hpp"
```

```
00008
00009                                           // Code Principal //
00027 class Entite
00028 {
00029     private :
00030     int team;
00031     bool life;
00032     std::string name;
00033     Animation anim;
00034     float x,y,vx,vy,R,angle;
00035
00036     public:
00037
00041     Entite(){life=1;}
00042
00052     void settings(Animation &a,int X,int Y,float Angle=0,int radius=1);
00053
00054     virtual void update();
00055
00056     void draw(sf::RenderWindow &app);
00057
00058
00059     float getx();
00060     void setx(float newX);
00061
00062     float gety();
00063     void sety(float newY);
00064
00065     float getvx();
00066     void setvx(float newVX);
00067
00068     float getvy();
00069     void setvy(float newVY);
00070
00071     float getR();
00072     void setR(float newR);
00073
00074     float getangle();
00075     void setangle(float newAngle);
00076
00077
00078     int getTeam();
00079     void setTeam(int choixTeam);
00080
00081     bool getLife();
00082     void setLife(bool newlife);
00083
00084     std::string getName();
00085     void setName(std::string newName);
00086
00087     Animation* getAnim(); // passage par pointeur sinon la modification via update n'est pas prise en
      compte.
00088     void setAnim(Animation newAnim);
00089
00090     virtual ~Entite();
00091
00092 };
00093
00094 #endif // ENTITE_HPP
```

6.37 /Users/savardtom/Desktop/2A/IN204/↩ Asteroids/code/objet/player.cpp File Reference

```
#include "Entite.hpp"
#include "player.hpp"
#include "../Animation.hpp"
#include "../global_variables.hpp"
```

6.38 player.cpp

Go to the documentation of this file.

```
00001                                           // Bibliothèques //
00002 #include "Entite.hpp"
00003 #include "player.hpp"
00004 #include "../Animation.hpp"
00005 #include "../global_variables.hpp"
00006
00007                                           // Code Principal //
00013 void player::update()
00014 {
00015     if (thrust)
00016     {
00017         this->setvx(this->getvx() + cos(this->getangle()*DEGTORAD)*0.2);
00018         this->setvy(this->getvy() + sin(this->getangle()*DEGTORAD)*0.2);
00019     }
00020     else
00021     {
00022         this->setvx(this->getvx() * 0.99);
00023         this->setvy(this->getvy() * 0.99);
00024     }
00025
00026     int maxSpeed=15;
00027     float speed = sqrt(this->getvx()*this->getvx()+this->getvy()*this->getvy());
00028     if (speed>maxSpeed)
00029     {
00030         this->setvx(this->getvx() * maxSpeed/speed);
00031         this->setvy(this->getvy() * maxSpeed/speed);
00032     }
00033
00034     this->setx(this->getx() + this->getvx());
00035     this->sety(this->gety() + this->getvy());
00036
00037     if (this->getx()>LargeurFenetre) this->setx(0); if (this->getx()<0) this->setx(LargeurFenetre);
00038     if (this->gety()>HauteurFenetre) this->sety(0); if (this->gety()<0) this->sety(HauteurFenetre);
00039 }
00040
00041 void player::draw_bouclier (sf::RenderWindow &app, bool invincible)
00042 {
00043     if (invincible){
00044         sf::CircleShape circle(this->getR());
00045         circle.setFillColor(sf::Color(255,0,0,170));
00046         circle.setPosition(this->getx(),this->gety());
00047         circle.setOrigin(this->getR(),this->getR());
00048         app.draw(circle);
00049     }
00050 }
```

6.39 /Users/savardtom/Desktop/2A/IN204/↩ Asteroids/code/objet/player.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include "../Animation.hpp"
#include "Entite.hpp"
```

Classes

- class player

  Classe player correspondant au vaisseau du joueur. Il peut se déplacer et tirer avec son blaster.

6.40 player.hpp

Go to the documentation of this file.

```
00001 #ifndef PLAYER_HPP
00002 #define PLAYER_HPP
00003                                           // Bibliothèques //
```

```
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include <iostream>
00007 #include "../Animation.hpp"
00008 #include "Entite.hpp"
00009
00010                                           // Code Principal //
00015 class player: public Entite
00016 {
00017
00018 public:
00019   bool thrust; // à mettre en privée
00020
00021   player()
00022   {
00023     this->setName("player");
00024   }
00025
00026   void update();
00027
00034   void draw_bouclier(sf::RenderWindow &app , bool invincible);
00035
00036 };
00037
00038 #endif // PLAYER_HPP
```

6.41 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/tir.cpp File Reference

```
#include "tir.hpp"
#include "Entite.hpp"
#include "../Animation.hpp"
#include "../global_variables.hpp"
```

6.42 tir.cpp

Go to the documentation of this file.

```
00001                                           // Bibliothèques //
00002 #include "tir.hpp"
00003 #include "Entite.hpp"
00004 #include "../Animation.hpp"
00005 #include "../global_variables.hpp"
00006
00007                                           // Code Principal //
00013 void  tir::update()
00014 {
00015     this->setvx(cos(this->getangle()*DEGTORAD)*6);
00016     this->setvy(sin(this->getangle()*DEGTORAD)*6);
00017     // angle+=rand()%7-3;  /*try this*/
00018     this->setx(this->getx() + this->getvx());
00019     this->sety(this->gety() + this->getvy());
00020
00021     if (this->getx()>LargeurFenetre || this->getx()<0 || this->gety()>HauteurFenetre ||
      this->gety()<0) this->setLife(false);
00022 }
00023
```

6.43 /Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/tir.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include "../Animation.hpp"
#include "Entite.hpp"
```

Classes

- class tir

  Ce sont les projectiles du blaster. Ils peuvent entrer en collision avec les entités ennemies (astéroides ou autre vaisseau).

6.44 tir.hpp

Go to the documentation of this file.

```
00001 #ifndef TIR_HPP
00002 #define TIR_HPP
00003                                                   // Bibliothéques //
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include <iostream>
00007 #include "../Animation.hpp"
00008 #include "Entite.hpp"
00009
00010                                                   // Code Principal //
00015 class tir: public Entite
00016 {
00017
00018   public:
00023   tir()
00024   {
00025     this->setName("tir");
00026   }
00027
00028
00029   void  update();
00030
00031 };
00032
00033 #endif // TIR_HPP
```

Index

/Users/savardtom/Desktop/2A/IN204/Asteroids/code/Animation.cpp, 35
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/Animation.hpp, 35, 36
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/global_variables.cpp, 36, 38
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/global_variables.hpp, 38, 40
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOver.cpp, 40
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOver.hpp, 41
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOverMulti.cpp, 41, 42
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/GameOverMulti.hpp, 42, 43
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/Menu.cpp, 43
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/Menu.hpp, 45
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/TableauDesScores.cpp, 46
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/interface/TableauDesScores.hpp, 46, 47
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/main.cpp, 47, 48
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/mainpage.h, 55
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/Entite.cpp, 56, 57
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/Entite.hpp, 58
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/asteroide.cpp, 55
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/asteroide.hpp, 56
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/player.cpp, 59
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/player.hpp, 60
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/tir.cpp, 61
/Users/savardtom/Desktop/2A/IN204/Asteroids/code/objet/tir.hpp, 61, 62
~Entite
    Entite, 16

Animation, 9
    Animation, 10
    Frame, 12
    frames, 12
    isEnd, 11
    speed, 12
    sprite, 12
    update, 11
asteroide, 12
asteroide.hpp, 14
    update, 14
Asteroides Documentation, 1

DEGTORAD
    global_variables.cpp, 36
    global_variables.hpp, 38
draw
    Entite, 17
    GameOverMultiScreen, 24
    GameOverScreen, 26
draw_bouclier
    player, 29
drawScore
    TableauDesScores, 31

Entite, 15
    ~Entite, 16
    draw, 17
    Entite, 16
    getangle, 17
    getAnim, 17
    getLife, 17
    getName, 18
    getR, 18
    getTeam, 18
    getvx, 18
    getvy, 19
    getx, 19
    gety, 19
    setangle, 19
    setAnim, 20
    setLife, 20
    setName, 20
    setR, 20
    setTeam, 21
    settings, 21
    setvx, 22
    setvy, 22
    setx, 22
    sety, 23
    update, 23

Frame

Animation, 12
frames
Animation, 12

GameOverMultiScreen, 23
draw, 24
GameOverMultiScreen, 24
GameOverScreen, 25
draw, 26
GameOverScreen, 25
getangle
Entite, 17
getAnim
Entite, 17
getLife
Entite, 17
getName
Entite, 18
getR
Entite, 18
getScore
TableauDesScores, 31
getTeam
Entite, 18
getvx
Entite, 18
getvy
Entite, 19
getx
Entite, 19
gety
Entite, 19
global_variables.cpp
DEGTORAD, 36
HauteurFenetre, 36
hoverColor, 37
LargeurFenetre, 37
NombreAsteroideStart, 37
NombreResidu, 37
normalColor, 37
Temps_invincible, 37
VolumeGeneral, 37
global_variables.hpp
DEGTORAD, 38
HauteurFenetre, 38
hoverColor, 38
LargeurFenetre, 39
NombreAsteroideStart, 39
NombreResidu, 39
normalColor, 39
Temps_invincible, 39
VolumeGeneral, 39

HauteurFenetre
global_variables.cpp, 36
global_variables.hpp, 38
hoverColor
global_variables.cpp, 37
global_variables.hpp, 38

increaseScore
TableauDesScores, 31
isCollide
main.cpp, 48
isEnd
Animation, 11

LargeurFenetre
global_variables.cpp, 37
global_variables.hpp, 39

main
main.cpp, 48
main.cpp
isCollide, 48
main, 48
Menu, 26
Menu, 27
run, 27

NombreAsteroideStart
global_variables.cpp, 37
global_variables.hpp, 39
NombreResidu
global_variables.cpp, 37
global_variables.hpp, 39
normalColor
global_variables.cpp, 37
global_variables.hpp, 39

player, 28
draw_bouclier, 29
player, 29
thrust, 30
update, 30

reset
TableauDesScores, 32
run
Menu, 27

setangle
Entite, 19
setAnim
Entite, 20
setLife
Entite, 20
setName
Entite, 20
setR
Entite, 20
setTeam
Entite, 21
settings
Entite, 21
setvx
Entite, 22
setvy
Entite, 22
setx

Entite, 22
sety
Entite, 23
speed
Animation, 12
sprite
Animation, 12

TableauDesScores, 30
drawScore, 31
getScore, 31
increaseScore, 31
reset, 32
TableauDesScores, 31
Temps_invincible
global_variables.cpp, 37
global_variables.hpp, 39
thrust
player, 30
tir, 32
tir, 34
update, 34

update
Animation, 11
asteroide, 14
Entite, 23
player, 30
tir, 34

VolumeGeneral
global_variables.cpp, 37
global_variables.hpp, 39