Missile Command

Generated by Doxygen 1.9.6

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Action Namespace Reference

Enumerations

- enum Action {
  LMouse , RMouse , Space , Pause ,
  Exit }

5.1.1 Enumeration Type Documentation

5.1.1.1 Action

enum Action::Action

Enumerator

LMouse	
RMouse	
Space	
Pause	
Exit	

Definition at line 5 of file Action.hpp.

5.2 State Namespace Reference

Enumerations

- enum State {
  InGame = 1 , Menu = 2 , GameOver = 3 , Pause = 4 ,
  Exit = 5 }

5.2.1 Enumeration Type Documentation

5.2.1.1 State

```
enum State::State
```

Enumerator

InGame	
Menu	
GameOver	
Pause	
Exit	

Definition at line 5 of file State.hpp.

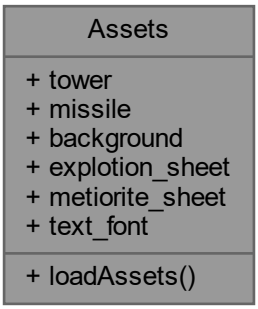
Chapter 6

Class Documentation

6.1 Assets Class Reference

`#include <Assets.hpp>`

Collaboration diagram for Assets:



Static Public Member Functions

- static void loadAssets ()

Static Public Attributes

- static sf::Texture tower
- static sf::Texture missile
- static sf::Texture background
- static sf::Texture explotion_sheet
- static sf::Texture metiorite_sheet
- static sf::Font text_font

6.1.1 Detailed Description

Definition at line 6 of file Assets.hpp.

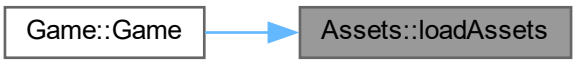
6.1.2 Member Function Documentation

6.1.2.1 loadAssets()

```
static void Assets::loadAssets ( )  [inline], [static]
```

Definition at line 15 of file Assets.hpp.

Here is the caller graph for this function:



6.1.3 Member Data Documentation

6.1.3.1 background

```
sf::Texture Assets::background  [static]
```

Definition at line 11 of file Assets.hpp.

6.1.3.2 explotion_sheet

```
sf::Texture Assets::explotion_sheet  [static]
```

Definition at line 12 of file Assets.hpp.

6.1.3.3 metiorite_sheet

```
sf::Texture Assets::metiorite_sheet  [static]
```

Definition at line 13 of file Assets.hpp.

6.1.3.4 missile

```
sf::Texture Assets::missile  [static]
```

Definition at line 10 of file Assets.hpp.

6.1.3.5 text_font

```
sf::Font Assets::text_font  [static]
```

Definition at line 14 of file Assets.hpp.

6.1.3.6 tower

```
sf::Texture Assets::tower  [static]
```

Definition at line 9 of file Assets.hpp.

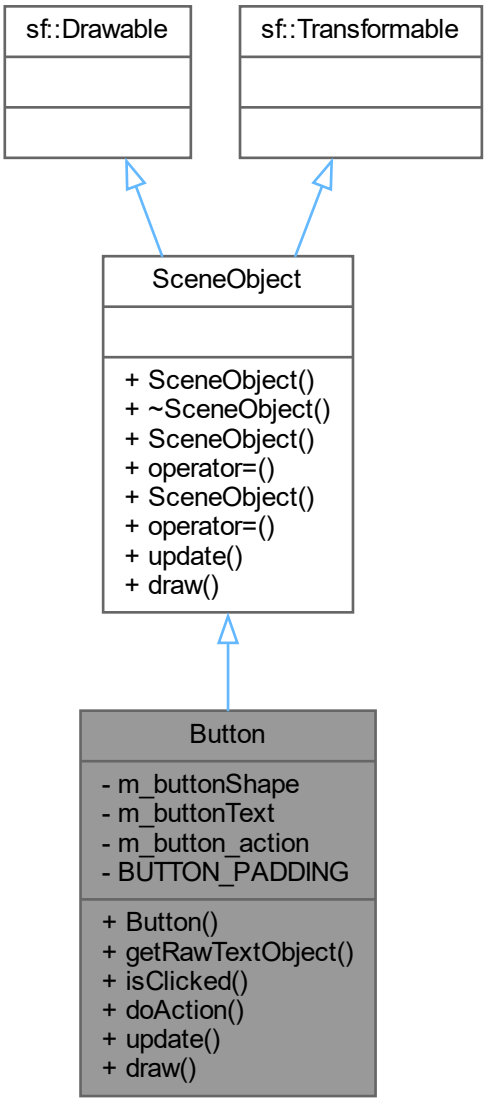
The documentation for this class was generated from the following files:

- tmp/Assets.hpp
- tmp/Assets.cpp

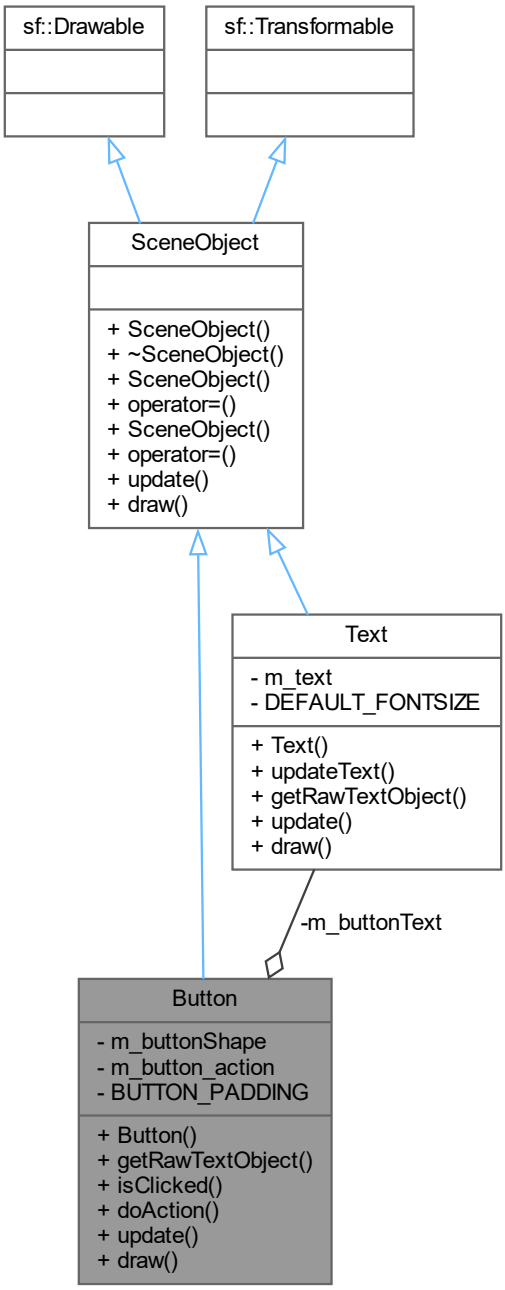
6.2 Button Class Reference

`#include <Button.hpp>`

Inheritance diagram for Button:



Collaboration diagram for Button:



Public Member Functions

- Button (const sf::Vector2f &position, const std::string &_text, std::function< State::State(void)> _button_↩ action)
- auto getRawTextObject () -> sf::Text &
- auto isClicked (const sf::Vector2i &mouse_position, const sf::RenderWindow &window) -> bool
- auto doAction () -> State::State

- auto update (const sf::Time &delta) -> bool override
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- sf::RectangleShape m_buttonShape
- Text m_buttonText
- std::function< State::State(void)> m_button_action

Static Private Attributes

- static constexpr float BUTTON_PADDING = 10.F

6.2.1 Detailed Description

Definition at line 13 of file Button.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Button()

```
Button::Button (
            const sf::Vector2f & position,
            const std::string & _text,
            std::function< State::State(void)> _button_action )
```

Definition at line 6 of file Button.cpp.

Here is the call graph for this function:

6.2.3 Member Function Documentation

6.2.3.1 doAction()

```
auto Button::doAction ( ) -> State::State
```

Definition at line 31 of file Button.cpp.

6.2.3.2 draw()

```
void Button::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [virtual]
```

Implements SceneObject.

Definition at line 46 of file Button.cpp.

6.2.3.3 getRawTextObject()

```
auto Button::getRawTextObject ( ) -> sf::Text&
```

Definition at line 36 of file Button.cpp.

6.2.3.4 isClicked()

```
auto Button::isClicked (
            const sf::Vector2i & mouse_position,
            const sf::RenderWindow & window ) -> bool
```

Definition at line 25 of file Button.cpp.

6.2.3.5 update()

```
auto Button::update (
            const sf::Time & delta ) -> bool  [override], [virtual]
```

Implements SceneObject.

Definition at line 41 of file Button.cpp.

6.2.4 Member Data Documentation

6.2.4.1 BUTTON_PADDING

```
constexpr float Button::BUTTON_PADDING = 10.F  [static], [constexpr], [private]
```

Definition at line 16 of file Button.hpp.

6.2.4.2 m_button_action

```
std::function<State::State(void)> Button::m_button_action  [private]
```

Definition at line 19 of file Button.hpp.

6.2.4.3 m_buttonShape

```
sf::RectangleShape Button::m_buttonShape  [private]
```

Definition at line 17 of file Button.hpp.

6.2.4.4 m_buttonText

```
Text Button::m_buttonText  [private]
```

Definition at line 18 of file Button.hpp.

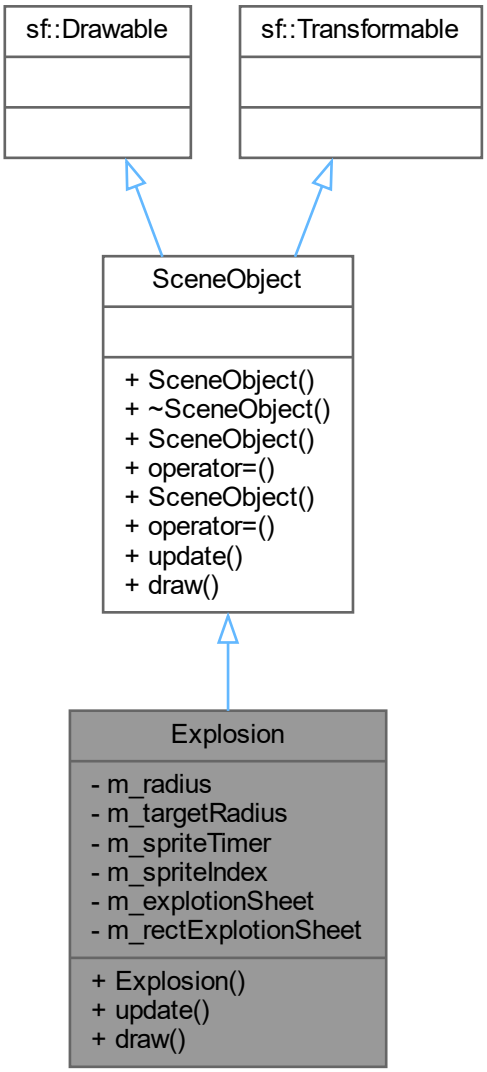
The documentation for this class was generated from the following files:

- tmp/Button.hpp
- tmp/Button.cpp

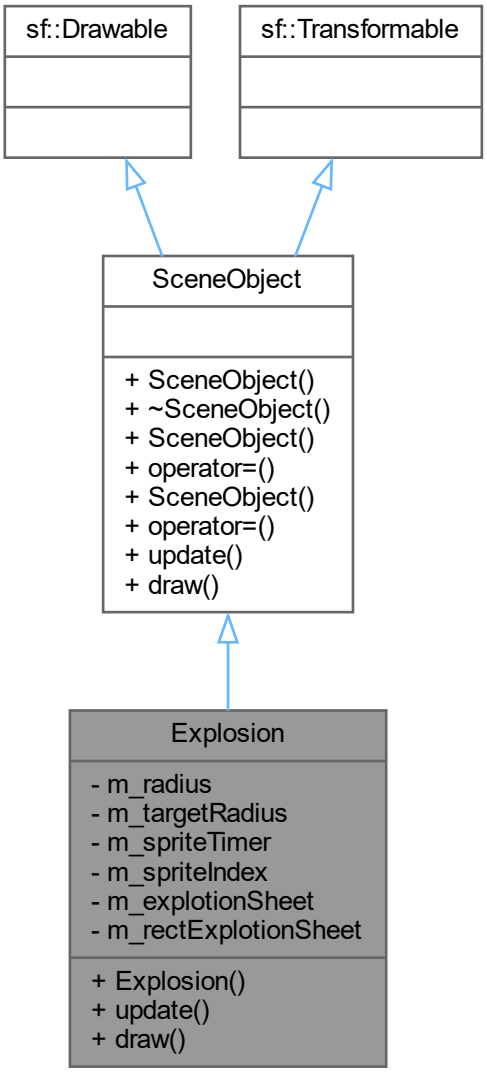
6.3 Explosion Class Reference

`#include <Explosion.hpp>`

Inheritance diagram for Explosion:



Collaboration diagram for Explosion:



Public Member Functions

- Explosion (const sf::Vector2f &position, float radius)
- auto update (const sf::Time &delta) -> bool override
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default

- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- float m_radius
- float m_targetRadius
- float m_spriteTimer = 0.0F
- int m_spriteIndex = 0
- sf::Sprite m_explotionSheet
- sf::IntRect m_rectExplotionSheet

6.3.1 Detailed Description

Definition at line 5 of file Explosion.hpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Explosion()

```
Explosion::Explosion (
            const sf::Vector2f & position,
            float radius )
```

Definition at line 3 of file Explosion.cpp.

6.3.3 Member Function Documentation

6.3.3.1 draw()

```
void Explosion::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [virtual]
```

Implements SceneObject.

Definition at line 38 of file Explosion.cpp.

6.3.3.2 update()

```
auto Explosion::update (
            const sf::Time & delta ) -> bool  [override], [virtual]
```

Implements SceneObject.

Definition at line 18 of file Explosion.cpp.

6.3.4 Member Data Documentation

6.3.4.1 m_explotionSheet

```
sf::Sprite Explosion::m_explotionSheet  [private]
```

Definition at line 11 of file Explosion.hpp.

6.3.4.2 m_radius

```
float Explosion::m_radius  [private]
```

Definition at line 7 of file Explosion.hpp.

6.3.4.3 m_rectExplotionSheet

```
sf::IntRect Explosion::m_rectExplotionSheet  [private]
```

Definition at line 12 of file Explosion.hpp.

6.3.4.4 m_spriteIndex

```
int Explosion::m_spriteIndex = 0  [private]
```

Definition at line 10 of file Explosion.hpp.

6.3.4.5 m_spriteTimer

```
float Explosion::m_spriteTimer = 0.0F  [private]
```

Definition at line 9 of file Explosion.hpp.

6.3.4.6 m_targetRadius

```
float Explosion::m_targetRadius  [private]
```

Definition at line 8 of file Explosion.hpp.

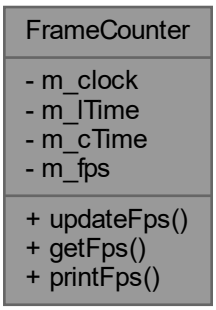
The documentation for this class was generated from the following files:

- tmp/Explosion.hpp
- tmp/Explosion.cpp

6.4 FrameCounter Class Reference

```
#include <FrameCounter.hpp>
```

Collaboration diagram for FrameCounter:



Public Member Functions

- void updateFps ()
- auto getFps () const -> float
- void printFps () const

Private Attributes

- sf::Clock m_clock
- sf::Time m_lTime = m_clock.getElapsedTime()
- sf::Time m_cTime
- float m_fps = 0.0F

6.4.1 Detailed Description

Definition at line 5 of file FrameCounter.hpp.

6.4.2 Member Function Documentation

6.4.2.1 getFps()

```
auto FrameCounter::getFps ( ) const -> float
```

Definition at line 12 of file FrameCounter.cpp.

6.4.2.2 printFps()

```
void FrameCounter::printFps ( ) const
```

Definition at line 17 of file FrameCounter.cpp.

6.4.2.3 updateFps()

```
void FrameCounter::updateFps ( )
```

Definition at line 5 of file FrameCounter.cpp.

6.4.3 Member Data Documentation

6.4.3.1 m_clock

```
sf::Clock FrameCounter::m_clock  [private]
```

Definition at line 7 of file FrameCounter.hpp.

6.4.3.2 m_cTime

```
sf::Time FrameCounter::m_cTime  [private]
```

Definition at line 9 of file FrameCounter.hpp.

6.4.3.3 m_fps

```
float FrameCounter::m_fps = 0.0F  [private]
```

Definition at line 10 of file FrameCounter.hpp.

6.4.3.4 m_lTime

```
sf::Time FrameCounter::m_lTime = m_clock.getElapsedTime()  [private]
```

Definition at line 8 of file FrameCounter.hpp.

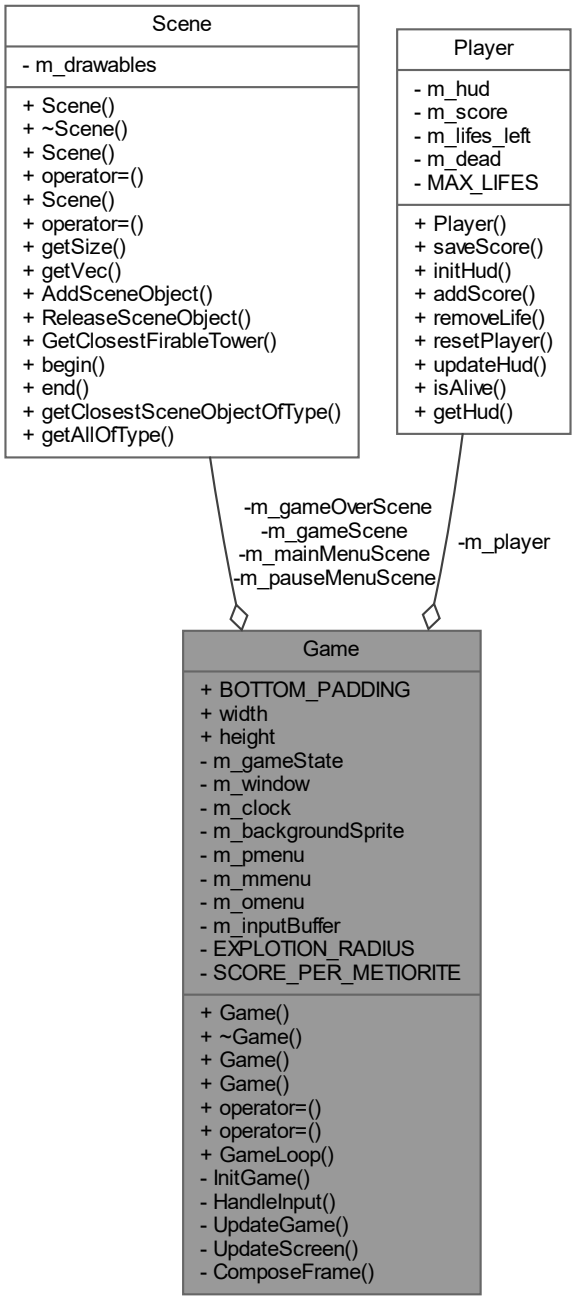
The documentation for this class was generated from the following files:

- tmp/FrameCounter.hpp
- tmp/FrameCounter.cpp

6.5 Game Class Reference

```
#include <Game.hpp>
```

Collaboration diagram for Game:



Public Member Functions

- Game (int width, int height)
- ~Game ()
- Game (const Game &other)=delete
- Game (Game &&other)=delete
- auto operator= (Game &&rhs) -> Game &=delete

- auto operator= (const Game &other) -> Game &=delete
- void GameLoop ()

Static Public Attributes

- static constexpr float BOTTOM_PADDING = 70.F
- static int width
- static int height

Private Member Functions

- void InitGame ()
- void HandleInput ()
- void UpdateGame ()
- void UpdateScreen ()
- void ComposeFrame ()

Private Attributes

- State::State m_gameState {State::Menu}
- sf::RenderWindow m_window
- sf::Clock m_clock
- sf::Sprite m_backgroundSprite
- Scene m_gameScene
- Scene m_pauseMenuScene
- Scene m_mainMenuScene
- Scene m_gameOverScene
- Player m_player
- std::shared_ptr< PauseMenu > m_pmenu
- std::shared_ptr< MainMenu > m_mmenu
- std::shared_ptr< GameOverMenu > m_omenu
- std::multimap< Action::Action, std::any > m_inputBuffer

Static Private Attributes

- static constexpr float EXPLOTION_RADIUS = 60.F
- static constexpr int SCORE_PER_METIORITE = 1000

6.5.1 Detailed Description

Definition at line 31 of file Game.hpp.

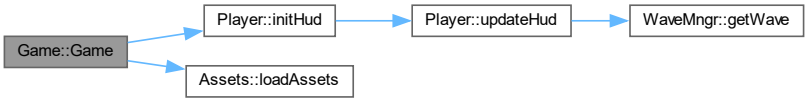
6.5.2 Constructor & Destructor Documentation

6.5.2.1 Game() [1/3]

```
Game::Game (
            int width,
            int height )
```

Definition at line 11 of file Game.cpp.

Here is the call graph for this function:



6.5.2.2 ~Game()

```
Game::~Game ( )  [default]
```

6.5.2.3 Game() [2/3]

```
Game::Game (
            const Game & other )  [delete]
```

6.5.2.4 Game() [3/3]

```
Game::Game (
            Game && other )  [delete]
```

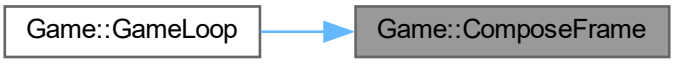
6.5.3 Member Function Documentation

6.5.3.1 ComposeFrame()

```
void Game::ComposeFrame ( )  [private]
```

Definition at line 277 of file Game.cpp.

Here is the caller graph for this function:



6.5.3.2 GameLoop()

```
void Game::GameLoop ( )
```

Definition at line 315 of file Game.cpp.

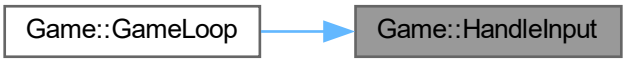
Here is the call graph for this function:

6.5.3.3 HandleInput()

```
void Game::HandleInput ( )  [private]
```

Definition at line 61 of file Game.cpp.

Here is the caller graph for this function:

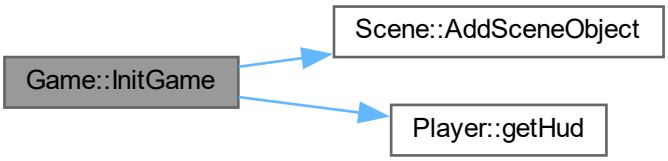


6.5.3.4 InitGame()

```
void Game::InitGame ( )  [private]
```

Definition at line 31 of file Game.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

6.5.3.5 operator=() [1/2]

```
auto Game::operator= (
            const Game & other ) -> Game &=delete  [delete]
```

6.5.3.6 operator=() [2/2]

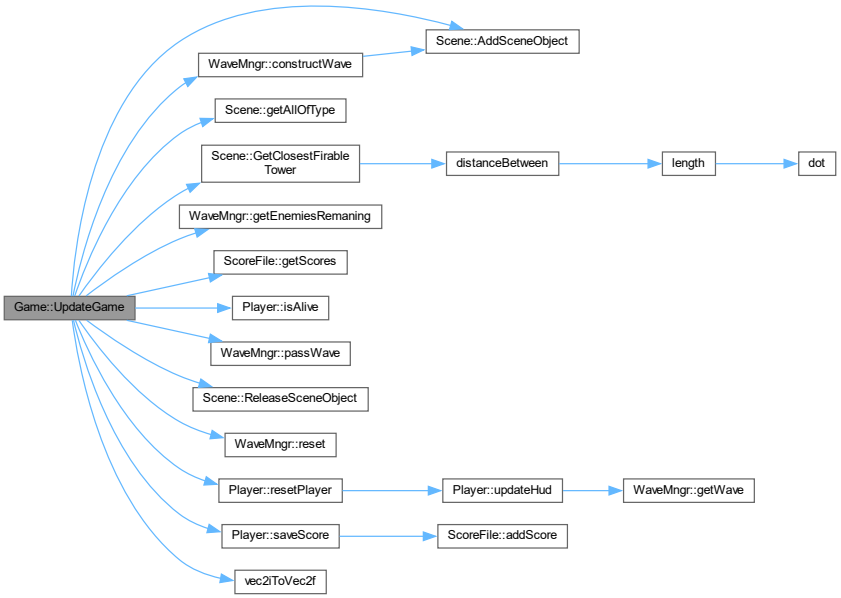
```
auto Game::operator= (
            Game && rhs ) -> Game &=delete  [delete]
```

6.5.3.7 UpdateGame()

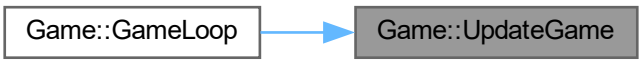
```
void Game::UpdateGame ( )  [private]
```

Definition at line 91 of file Game.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

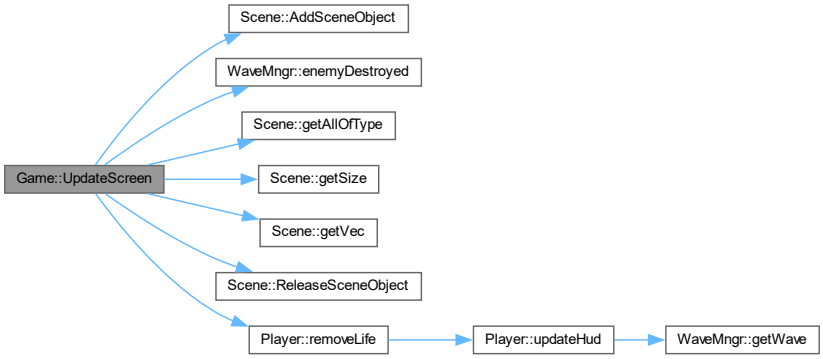


6.5.3.8 UpdateScreen()

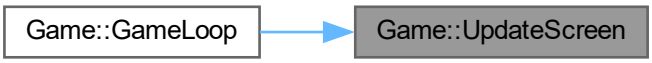
```
void Game::UpdateScreen ( )  [private]
```

Definition at line 206 of file Game.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.4 Member Data Documentation

6.5.4.1 BOTTOM_PADDING

```
constexpr float Game::BOTTOM_PADDING = 70.F  [static], [constexpr]
```

Definition at line 71 of file Game.hpp.

6.5.4.2 EXPLOTION_RADIUS

```
constexpr float Game::EXPLOTION_RADIUS = 60.F  [static], [constexpr], [private]
```

Definition at line 34 of file Game.hpp.

6.5.4.3 height

```
int Game::height  [inline], [static]
```

Definition at line 73 of file Game.hpp.

6.5.4.4 m_backgroundSprite

```
sf::Sprite Game::m_backgroundSprite  [private]
```

Definition at line 41 of file Game.hpp.

6.5.4.5 m_clock

```
sf::Clock Game::m_clock  [private]
```

Definition at line 40 of file Game.hpp.

6.5.4.6 m_gameOverScene

```
Scene Game::m_gameOverScene  [private]
```

Definition at line 45 of file Game.hpp.

6.5.4.7 m_gameScene

```
Scene Game::m_gameScene  [private]
```

Definition at line 42 of file Game.hpp.

6.5.4.8 m_gameState

```
State::State Game::m_gameState {State::Menu}  [private]
```

Definition at line 38 of file Game.hpp.

6.5.4.9 m_inputBuffer

```
std::multimap<Action::Action, std::any> Game::m_inputBuffer  [private]
```

Definition at line 51 of file Game.hpp.

6.5.4.10 m_mainMenuScene

```
Scene Game::m_mainMenuScene  [private]
```

Definition at line 44 of file Game.hpp.

6.5.4.11 m_mmenu

```
std::shared_ptr<MainMenu> Game::m_mmenu  [private]
```

Definition at line 48 of file Game.hpp.

6.5.4.12 m_omenu

```
std::shared_ptr<GameOverMenu> Game::m_omenu  [private]
```

Definition at line 49 of file Game.hpp.

6.5.4.13 m_pauseMenuScene

```
Scene Game::m_pauseMenuScene  [private]
```

Definition at line 43 of file Game.hpp.

6.5.4.14 m_player

```
Player Game::m_player  [private]
```

Definition at line 46 of file Game.hpp.

6.5.4.15 m_pmenu

```
std::shared_ptr<PauseMenu> Game::m_pmenu  [private]
```

Definition at line 47 of file Game.hpp.

6.5.4.16 m_window

```
sf::RenderWindow Game::m_window  [private]
```

Definition at line 39 of file Game.hpp.

6.5.4.17 SCORE_PER_METIORITE

```
constexpr int Game::SCORE_PER_METIORITE = 1000  [static], [constexpr], [private]
```

Definition at line 35 of file Game.hpp.

6.5.4.18 width

```
int Game::width  [inline], [static]
```

Definition at line 72 of file Game.hpp.

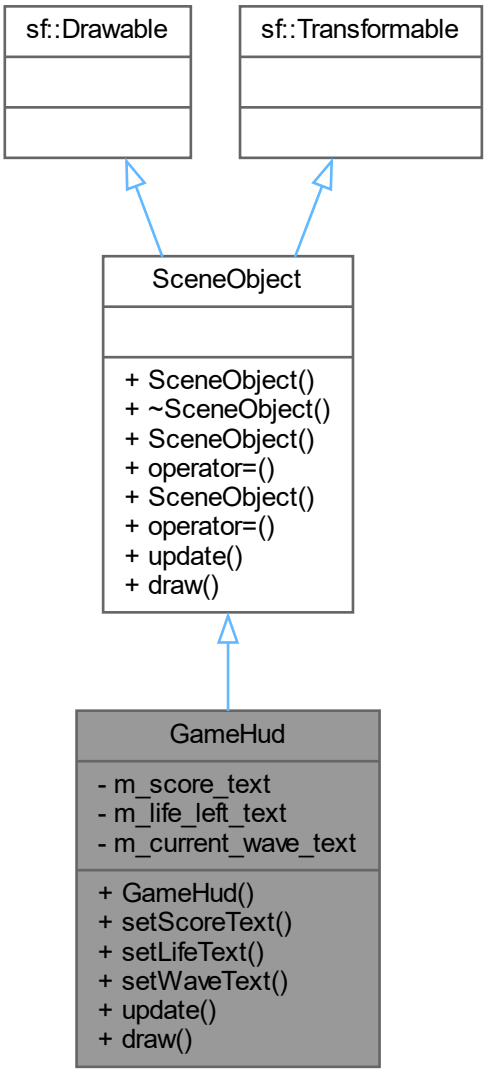
The documentation for this class was generated from the following files:

- tmp/Game.hpp
- tmp/Game.cpp

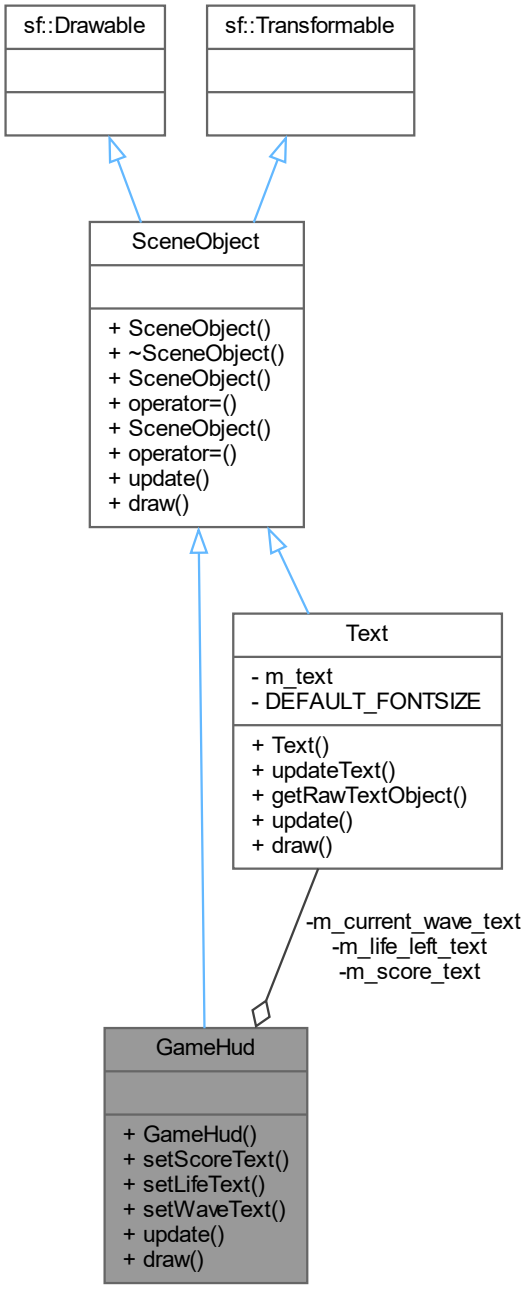
6.6 GameHud Class Reference

`#include <GameHud.hpp>`

Inheritance diagram for GameHud:



Collaboration diagram for GameHud:



Public Member Functions

- GameHud ()
- void setScoreText (const std::string &_text)
- void setLifeText (const std::string &_text)
- void setWaveText (const std::string &_text)
- auto update (const sf::Time &delta) -> bool override
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- Text m_score_text
- Text m_life_left_text
- Text m_current_wave_text

6.6.1 Detailed Description

Definition at line 6 of file GameHud.hpp.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 GameHud()

```
GameHud::GameHud ( )
```

Definition at line 5 of file GameHud.cpp.

6.6.3 Member Function Documentation

6.6.3.1 draw()

```
void GameHud::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [virtual]
```

Implements SceneObject.

Definition at line 35 of file GameHud.cpp.

6.6.3.2 setLifeText()

```
void GameHud::setLifeText (
            const std::string & _text )
```

Definition at line 20 of file GameHud.cpp.

Here is the call graph for this function:

6.6.3.3 setScoreText()

```
void GameHud::setScoreText (
            const std::string & _text )
```

Definition at line 15 of file GameHud.cpp.

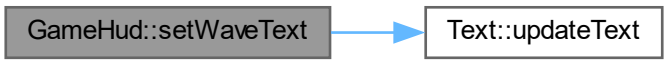
Here is the call graph for this function:

6.6.3.4 setWaveText()

```
void GameHud::setWaveText (
            const std::string & _text )
```

Definition at line 25 of file GameHud.cpp.

Here is the call graph for this function:



6.6.3.5 update()

```
auto GameHud::update (
            const sf::Time & delta ) -> bool  [override], [virtual]
```

Implements SceneObject.

Definition at line 30 of file GameHud.cpp.

6.6.4 Member Data Documentation

6.6.4.1 m_current_wave_text

```
Text GameHud::m_current_wave_text  [private]
```

Definition at line 10 of file GameHud.hpp.

6.6.4.2 m_life_left_text

```
Text GameHud::m_life_left_text  [private]
```

Definition at line 9 of file GameHud.hpp.

6.6.4.3 m_score_text

```
Text GameHud::m_score_text  [private]
```

Definition at line 8 of file GameHud.hpp.

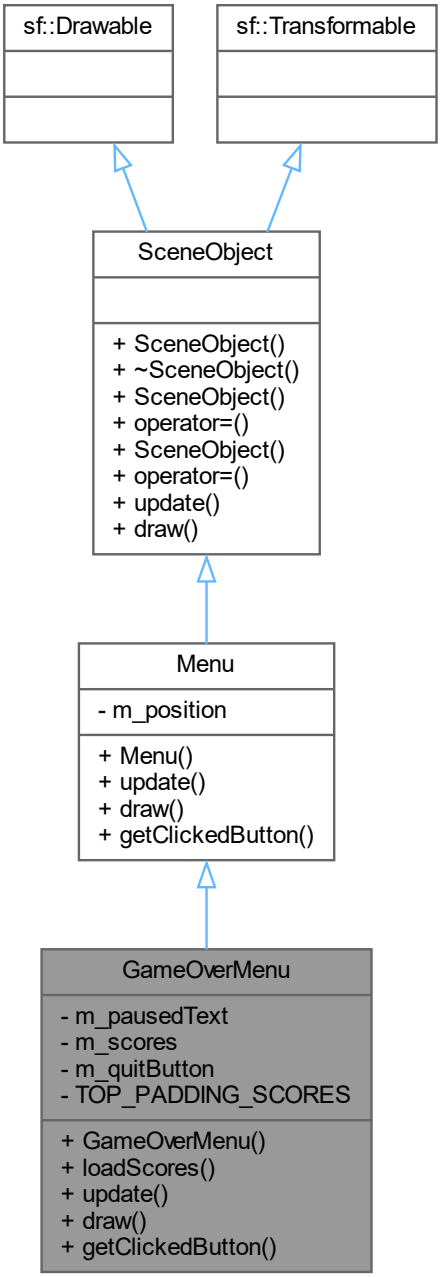
The documentation for this class was generated from the following files:

- tmp/GameHud.hpp
- tmp/GameHud.cpp

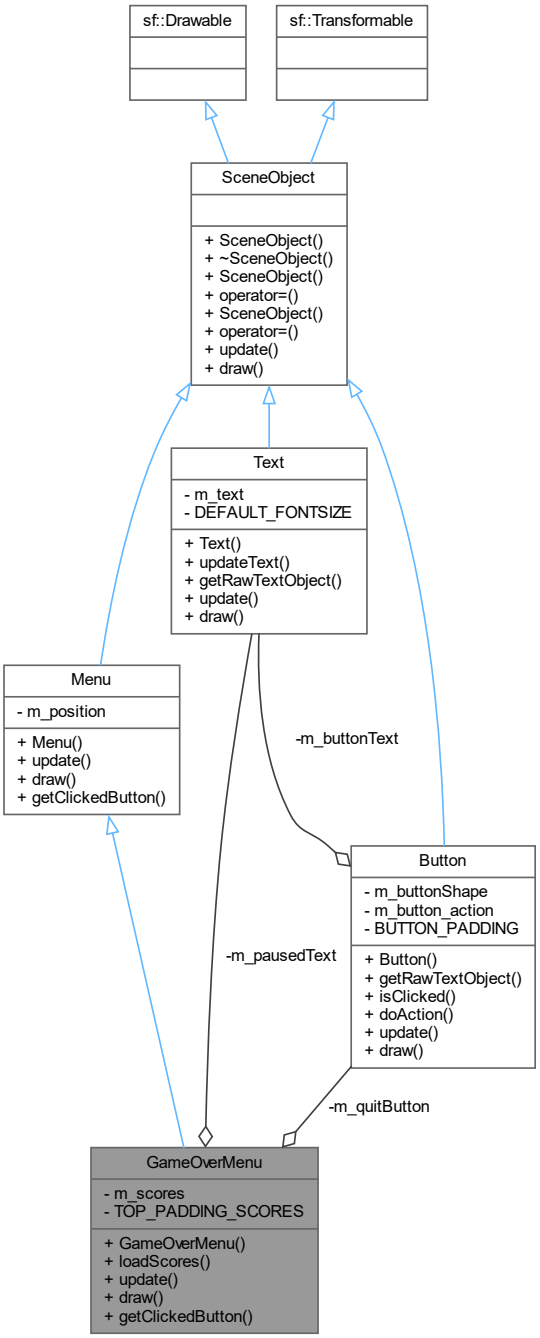
6.7 GameOverMenu Class Reference

`#include <GameOverMenu.hpp>`

Inheritance diagram for GameOverMenu:



Collaboration diagram for GameOverMenu:



Public Member Functions

- GameOverMenu ()
- void loadScores (std::vector< std::string > scores)
- auto update (const sf::Time &delta) -> bool override
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override
- auto getClickedButton (const sf::Vector2i &mouse_position, const sf::RenderWindow &window) -> std::optional< Button > override

Public Member Functions inherited from Menu

- Menu ()
- auto update (const sf::Time &delta) -> bool override=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0
- virtual auto getClickedButton (const sf::Vector2i &mouse_position, const sf::RenderWindow &window) -> std::optional< Button >=0

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- Text m_pausedText
- std::vector< Text > m_scores
- Button m_quitButton

Static Private Attributes

- static constexpr int TOP_PADDING_SCORES = 200

6.7.1 Detailed Description

Definition at line 11 of file GameOverMenu.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 GameOverMenu()

```
GameOverMenu::GameOverMenu ( )
```

Definition at line 3 of file GameOverMenu.cpp.

6.7.3 Member Function Documentation

6.7.3.1 draw()

```
void GameOverMenu::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [virtual]
```

Implements Menu.

Definition at line 27 of file GameOverMenu.cpp.

6.7.3.2 getClickedButton()

```
auto GameOverMenu::getClickedButton (
            const sf::Vector2i & mouse_position,
            const sf::RenderWindow & window ) -> std::optional<Button>  [override], [virtual]
```

Implements Menu.

Definition at line 37 of file GameOverMenu.cpp.

6.7.3.3 loadScores()

```
void GameOverMenu::loadScores (
            std::vector< std::string > scores )
```

Definition at line 10 of file GameOverMenu.cpp.

6.7.3.4 update()

```
auto GameOverMenu::update (
            const sf::Time & delta ) -> bool  [override], [virtual]
```

Implements Menu.

Definition at line 22 of file GameOverMenu.cpp.

6.7.4 Member Data Documentation

6.7.4.1 m_pausedText

```
Text GameOverMenu::m_pausedText  [private]
```

Definition at line 14 of file GameOverMenu.hpp.

6.7.4.2 m_quitButton

```
Button GameOverMenu::m_quitButton  [private]
```

Definition at line 16 of file GameOverMenu.hpp.

6.7.4.3 m_scores

```
std::vector<Text> GameOverMenu::m_scores  [private]
```

Definition at line 15 of file GameOverMenu.hpp.

6.7.4.4 TOP_PADDING_SCORES

```
constexpr int GameOverMenu::TOP_PADDING_SCORES = 200  [static], [constexpr], [private]
```

Definition at line 13 of file GameOverMenu.hpp.

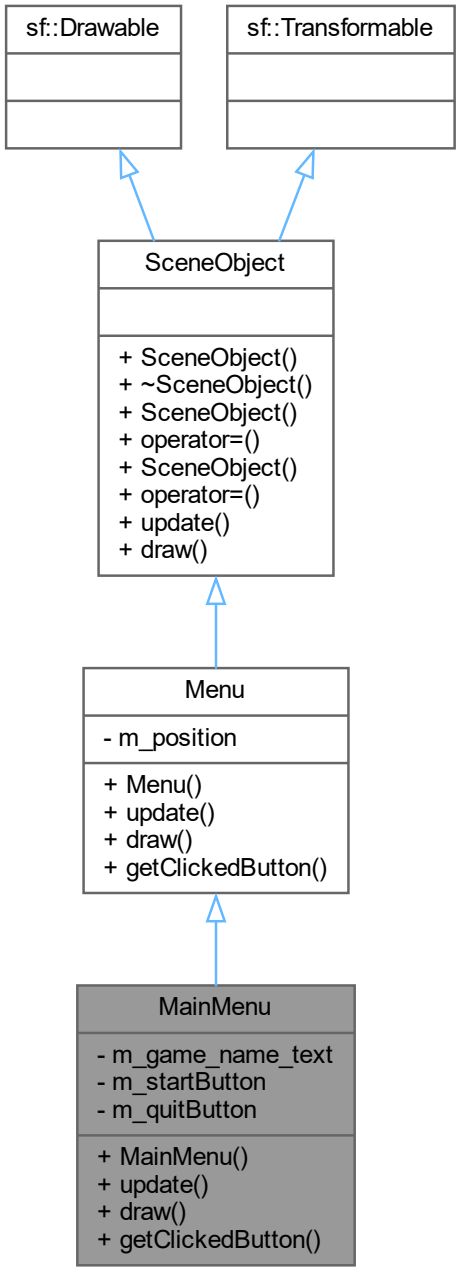
The documentation for this class was generated from the following files:

- tmp/GameOverMenu.hpp
- tmp/GameOverMenu.cpp

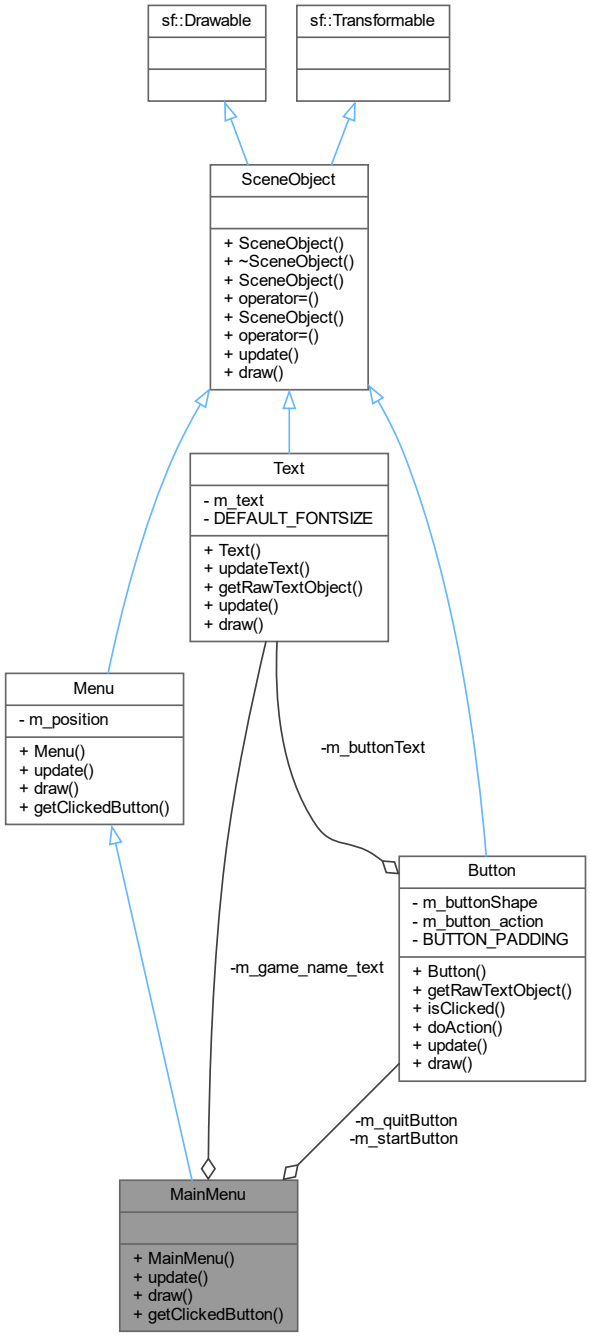
6.8 MainMenu Class Reference

`#include <MainMenu.hpp>`

Inheritance diagram for MainMenu:



Collaboration diagram for MainMenu:



Public Member Functions

- MainMenu ()
- auto update (const sf::Time &delta) -> bool override
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override
- auto getClickedButton (const sf::Vector2i &mouse_position, const sf::RenderWindow &window) -> std::optional< Button > override

Public Member Functions inherited from Menu

- Menu ()
- auto update (const sf::Time &delta) -> bool override=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0
- virtual auto getClickedButton (const sf::Vector2i &mouse_position, const sf::RenderWindow &window) -> std::optional< Button >=0

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- Text m_game_name_text
- Button m_startButton
- Button m_quitButton

6.8.1 Detailed Description

Definition at line 9 of file MainMenu.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 MainMenu()

```
MainMenu::MainMenu ( )
```

Definition at line 8 of file MainMenu.cpp.

6.8.3 Member Function Documentation

6.8.3.1 draw()

```
void MainMenu::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [virtual]
```

Implements Menu.

Definition at line 23 of file MainMenu.cpp.

6.8.3.2 getClickedButton()

```
auto MainMenu::getClickedButton (
            const sf::Vector2i & mouse_position,
            const sf::RenderWindow & window ) -> std::optional<Button>  [override], [virtual]
```

Implements Menu.

Definition at line 29 of file MainMenu.cpp.

6.8.3.3 update()

```
auto MainMenu::update (
            const sf::Time & delta ) -> bool  [override], [virtual]
```

Implements Menu.

Definition at line 18 of file MainMenu.cpp.

6.8.4 Member Data Documentation

6.8.4.1 m_game_name_text

```
Text MainMenu::m_game_name_text  [private]
```

Definition at line 11 of file MainMenu.hpp.

6.8.4.2 m_quitButton

```
Button MainMenu::m_quitButton  [private]
```

Definition at line 13 of file MainMenu.hpp.

6.8.4.3 m_startButton

```
Button MainMenu::m_startButton  [private]
```

Definition at line 12 of file MainMenu.hpp.

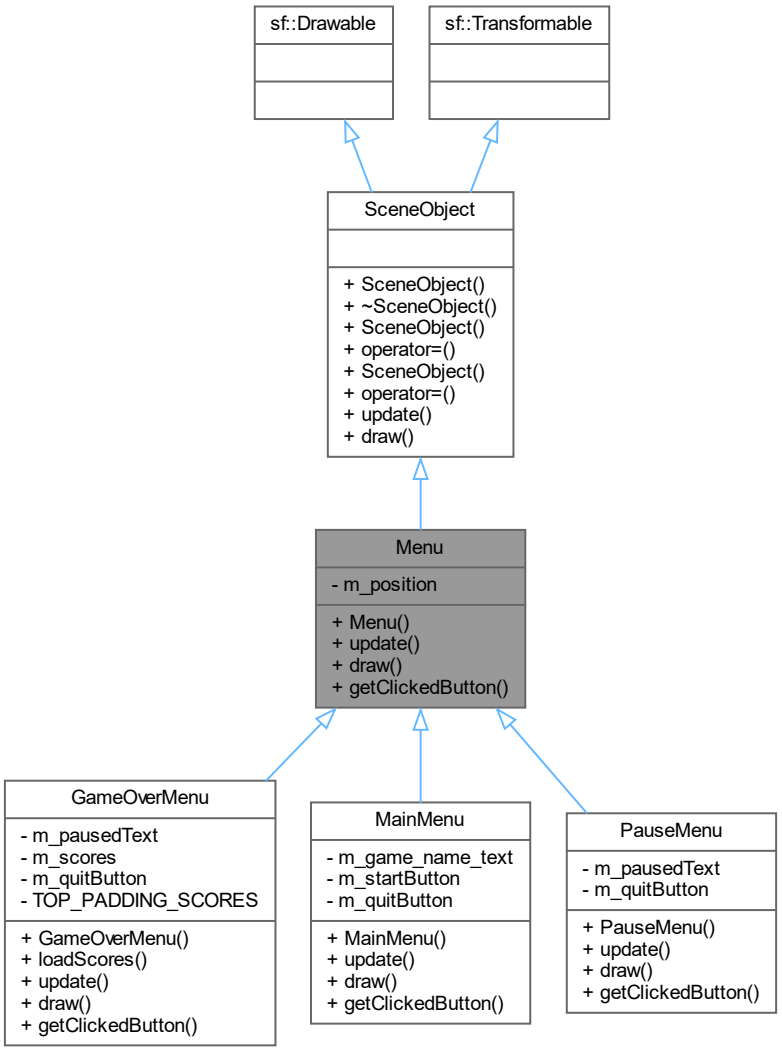
The documentation for this class was generated from the following files:

- tmp/MainMenu.hpp
- tmp/MainMenu.cpp

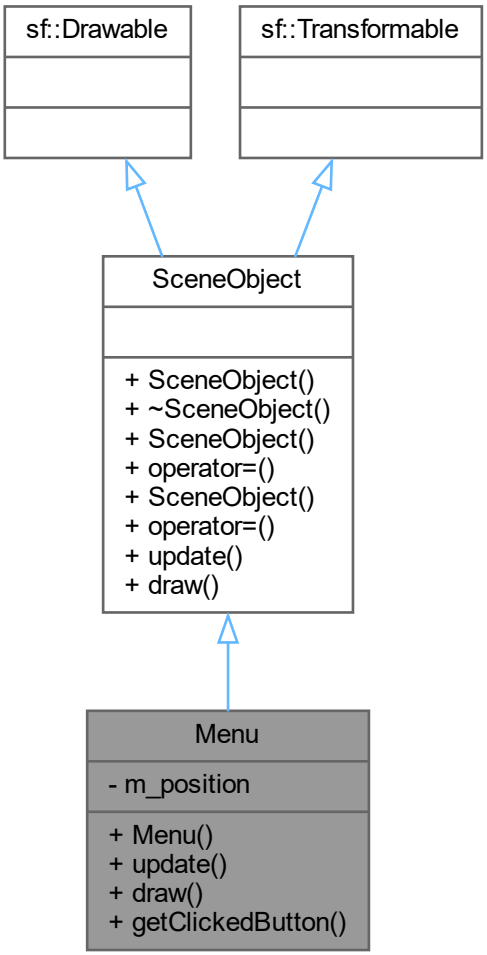
6.9 Menu Class Reference

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

Inheritance diagram for Menu:



Collaboration diagram for Menu:



Public Member Functions

- Menu ()
- auto update (const sf::Time &delta) -> bool override=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0
- virtual auto getClickedButton (const sf::Vector2i &mouse_position, const sf::RenderWindow &window) -> std::optional< Button >=0

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- const sf::Vector2f m_position = sf::Vector2f(0, 0)

6.9.1 Detailed Description

Definition at line 6 of file Menu.hpp.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Menu()

```
Menu::Menu ( )
```

Definition at line 3 of file Menu.cpp.

6.9.3 Member Function Documentation

6.9.3.1 draw()

```
void Menu::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [pure
virtual]
```

Implements SceneObject.

Implemented in GameOverMenu, MainMenu, and PauseMenu.

6.9.3.2 getClickedButton()

```
virtual auto Menu::getClickedButton (
            const sf::Vector2i & mouse_position,
            const sf::RenderWindow & window ) -> std::optional< Button >  [pure virtual]
```

Implemented in GameOverMenu, MainMenu, and PauseMenu.

6.9.3.3 update()

```
auto Menu::update (
            const sf::Time & delta ) -> bool  [override], [pure virtual]
```

Implements SceneObject.

Implemented in GameOverMenu, MainMenu, and PauseMenu.

6.9.4 Member Data Documentation

6.9.4.1 m_position

```
const sf::Vector2f Menu::m_position = sf::Vector2f(0, 0)  [private]
```

Definition at line 8 of file Menu.hpp.

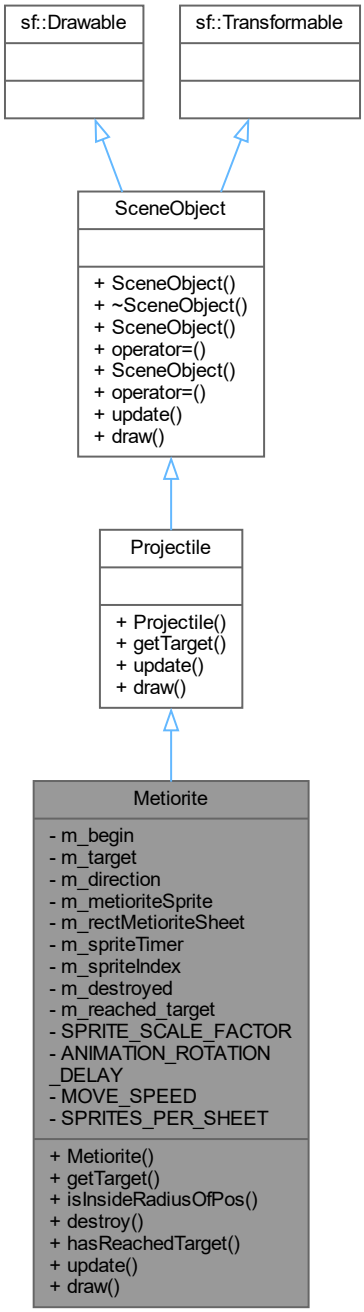
The documentation for this class was generated from the following files:

- tmp/Menu.hpp
- tmp/Menu.cpp

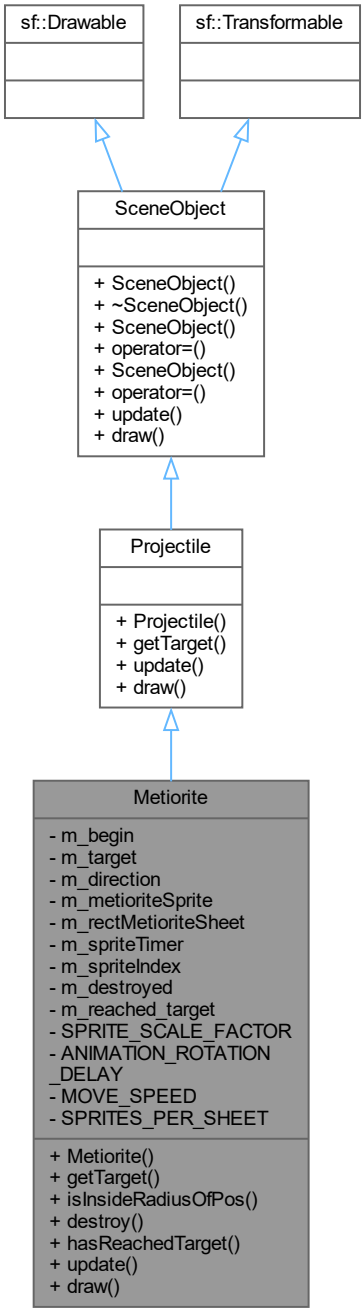
6.10 Metiorite Class Reference

```
#include <Meteorite.hpp>
```

Inheritance diagram for Metiorite:



Collaboration diagram for Metiorite:



Public Member Functions

- Metiorite (const sf::Vector2f &begin, const sf::Vector2f &target)
- auto getTarget () const -> sf::Vector2f override
- auto isInsideRadiusOfPos (const sf::Vector2f &pos, const float &radius) -> bool
- void destroy ()
- auto hasReachedTarget () const -> bool

- auto update (const sf::Time &delta) -> bool override
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override

Public Member Functions inherited from Projectile

- Projectile (const sf::Vector2f &position)
- virtual auto getTarget () const -> sf::Vector2f=0
- auto update (const sf::Time &delta) -> bool override=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- sf::Vector2f m_begin
- sf::Vector2f m_target
- sf::Vector2f m_direction
- sf::Sprite m_metioriteSprite
- sf::IntRect m_rectMetioriteSheet
- float m_spriteTimer = 0.0F
- int m_spriteIndex = 0
- bool m_destroyed = false
- bool m_reached_target = false

Static Private Attributes

- static constexpr float SPRITE_SCALE_FACTOR = 0.5F
- static constexpr float ANIMATION_ROTATION_DELAY = 0.17F
- static constexpr float MOVE_SPEED = 80.0F
- static constexpr int SPRITES_PER_SHEET = 3

6.10.1 Detailed Description

Definition at line 8 of file Meteorite.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 Metiorite()

```
Metiorite::Metiorite (
            const sf::Vector2f & begin,
            const sf::Vector2f & target )
```

Definition at line 4 of file Meteorite.cpp.

6.10.3 Member Function Documentation

6.10.3.1 destroy()

```
void Metiorite::destroy ( )
```

Definition at line 23 of file Meteorite.cpp.

6.10.3.2 draw()

```
void Metiorite::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [virtual]
```

Implements Projectile.

Definition at line 78 of file Meteorite.cpp.

6.10.3.3 getTarget()

```
auto Metiorite::getTarget ( ) const -> sf::Vector2f  [override], [virtual]
```

Implements Projectile.

Definition at line 33 of file Meteorite.cpp.

6.10.3.4 hasReachedTarget()

```
auto Metiorite::hasReachedTarget ( ) const -> bool
```

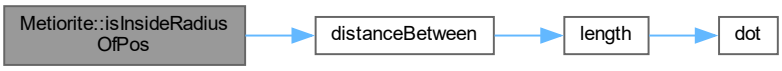
Definition at line 28 of file Meteorite.cpp.

6.10.3.5 isInsideRadiusOfPos()

```
auto Metiorite::isInsideRadiusOfPos (
            const sf::Vector2f & pos,
            const float & radius ) -> bool
```

Definition at line 38 of file Meteorite.cpp.

Here is the call graph for this function:



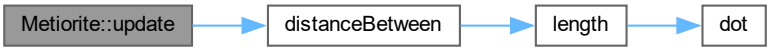
6.10.3.6 update()

```
auto Metiorite::update (
            const sf::Time & delta ) -> bool  [override], [virtual]
```

Implements Projectile.

Definition at line 43 of file Meteorite.cpp.

Here is the call graph for this function:



6.10.4 Member Data Documentation

6.10.4.1 ANIMATION_ROTATION_DELAY

```
constexpr float Metiorite::ANIMATION_ROTATION_DELAY = 0.17F  [static], [constexpr], [private]
```

Definition at line 12 of file Meteorite.hpp.

6.10.4.2 m_begin

```
sf::Vector2f Metiorite::m_begin  [private]
```

Definition at line 17 of file Meteorite.hpp.

6.10.4.3 m_destroyed

```
bool Metiorite::m_destroyed = false  [private]
```

Definition at line 24 of file Meteorite.hpp.

6.10.4.4 m_direction

```
sf::Vector2f Metiorite::m_direction  [private]
```

Definition at line 19 of file Meteorite.hpp.

6.10.4.5 m_metioriteSprite

```
sf::Sprite Metiorite::m_metioriteSprite  [private]
```

Definition at line 20 of file Meteorite.hpp.

6.10.4.6 m_reached_target

```
bool Metiorite::m_reached_target = false  [private]
```

Definition at line 25 of file Meteorite.hpp.

6.10.4.7 m_rectMetioriteSheet

```
sf::IntRect Metiorite::m_rectMetioriteSheet  [private]
```

Definition at line 21 of file Meteorite.hpp.

6.10.4.8 m_spriteIndex

```
int Metiorite::m_spriteIndex = 0  [private]
```

Definition at line 23 of file Meteorite.hpp.

6.10.4.9 m_spriteTimer

```
float Metiorite::m_spriteTimer = 0.0F  [private]
```

Definition at line 22 of file Meteorite.hpp.

6.10.4.10 m_target

```
sf::Vector2f Metiorite::m_target  [private]
```

Definition at line 18 of file Meteorite.hpp.

6.10.4.11 MOVE_SPEED

```
constexpr float Metiorite::MOVE_SPEED = 80.0F  [static], [constexpr], [private]
```

Definition at line 13 of file Meteorite.hpp.

6.10.4.12 SPRITE_SCALE_FACTOR

```
constexpr float Metiorite::SPRITE_SCALE_FACTOR = 0.5F  [static], [constexpr], [private]
```

Definition at line 11 of file Meteorite.hpp.

6.10.4.13 SPRITES_PER_SHEET

```
constexpr int Metiorite::SPRITES_PER_SHEET = 3  [static], [constexpr], [private]
```

Definition at line 14 of file Meteorite.hpp.

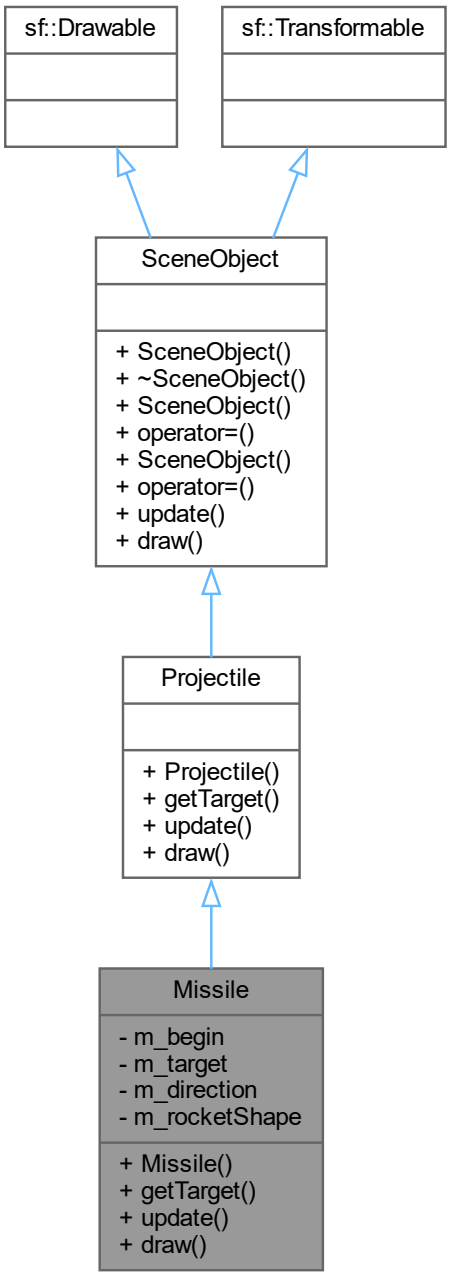
The documentation for this class was generated from the following files:

- tmp/Meteorite.hpp
- tmp/Meteorite.cpp

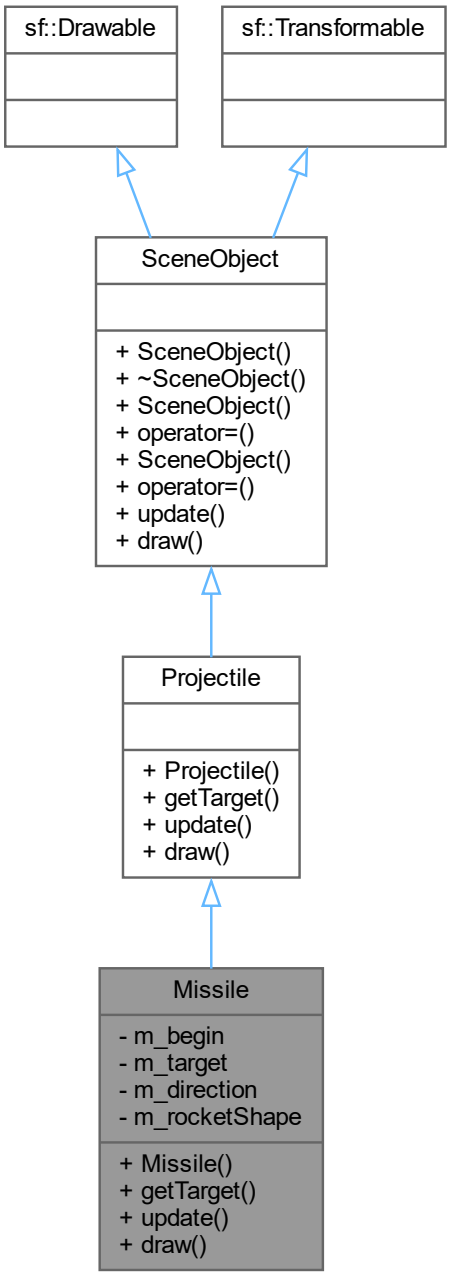
6.11 Missile Class Reference

`#include <Missile.hpp>`

Inheritance diagram for Missile:



Collaboration diagram for Missile:



Public Member Functions

- Missile (const sf::Vector2f &begin, const sf::Vector2f &target)
- auto getTarget () const -> sf::Vector2f override
- auto update (const sf::Time &delta) -> bool override
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override

Public Member Functions inherited from Projectile

- Projectile (const sf::Vector2f &position)
- virtual auto getTarget () const -> sf::Vector2f=0
- auto update (const sf::Time &delta) -> bool override=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- sf::Vector2f m_begin
- sf::Vector2f m_target
- sf::Vector2f m_direction
- sf::Sprite m_rocketShape

6.11.1 Detailed Description

Definition at line 6 of file Missile.hpp.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Missile()

```
Missile::Missile (
            const sf::Vector2f & begin,
            const sf::Vector2f & target )
```

Definition at line 4 of file Missile.cpp.

6.11.3 Member Function Documentation

6.11.3.1 draw()

```
void Missile::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [virtual]
```

Implements Projectile.

Definition at line 32 of file Missile.cpp.

6.11.3.2 getTarget()

```
auto Missile::getTarget ( ) const -> sf::Vector2f  [override], [virtual]
```

Implements Projectile.

Definition at line 13 of file Missile.cpp.

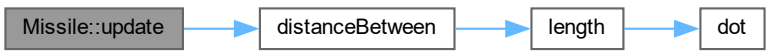
6.11.3.3 update()

```
auto Missile::update (
            const sf::Time & delta ) -> bool  [override], [virtual]
```

Implements Projectile.

Definition at line 18 of file Missile.cpp.

Here is the call graph for this function:



6.11.4 Member Data Documentation

6.11.4.1 m_begin

```
sf::Vector2f Missile::m_begin  [private]
```

Definition at line 8 of file Missile.hpp.

6.11.4.2 m_direction

```
sf::Vector2f Missile::m_direction  [private]
```

Definition at line 10 of file Missile.hpp.

6.11.4.3 m_rocketShape

```
sf::Sprite Missile::m_rocketShape  [private]
```

Definition at line 11 of file Missile.hpp.

6.11.4.4 m_target

```
sf::Vector2f Missile::m_target  [private]
```

Definition at line 9 of file Missile.hpp.

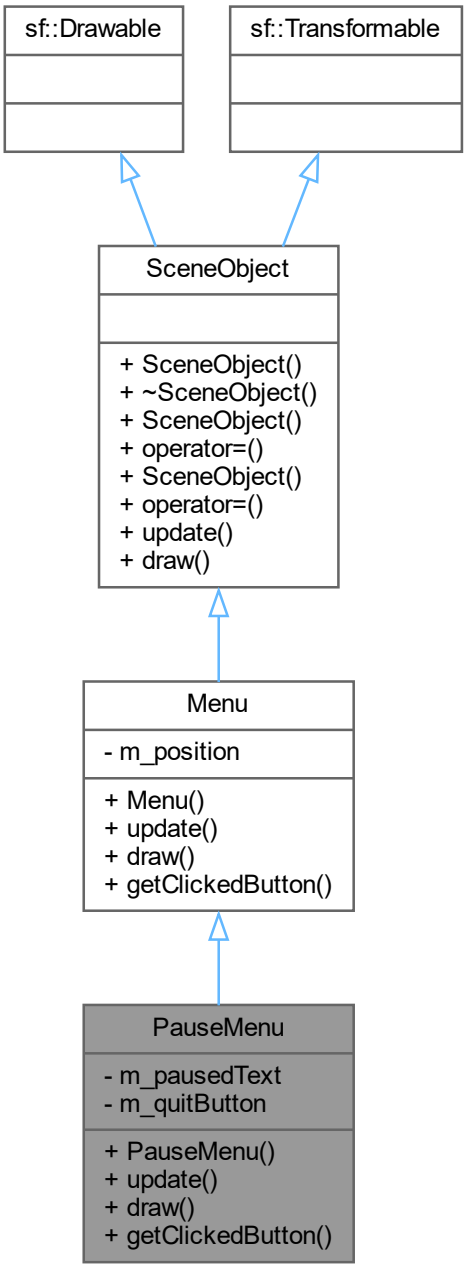
The documentation for this class was generated from the following files:

- tmp/Missile.hpp
- tmp/Missile.cpp

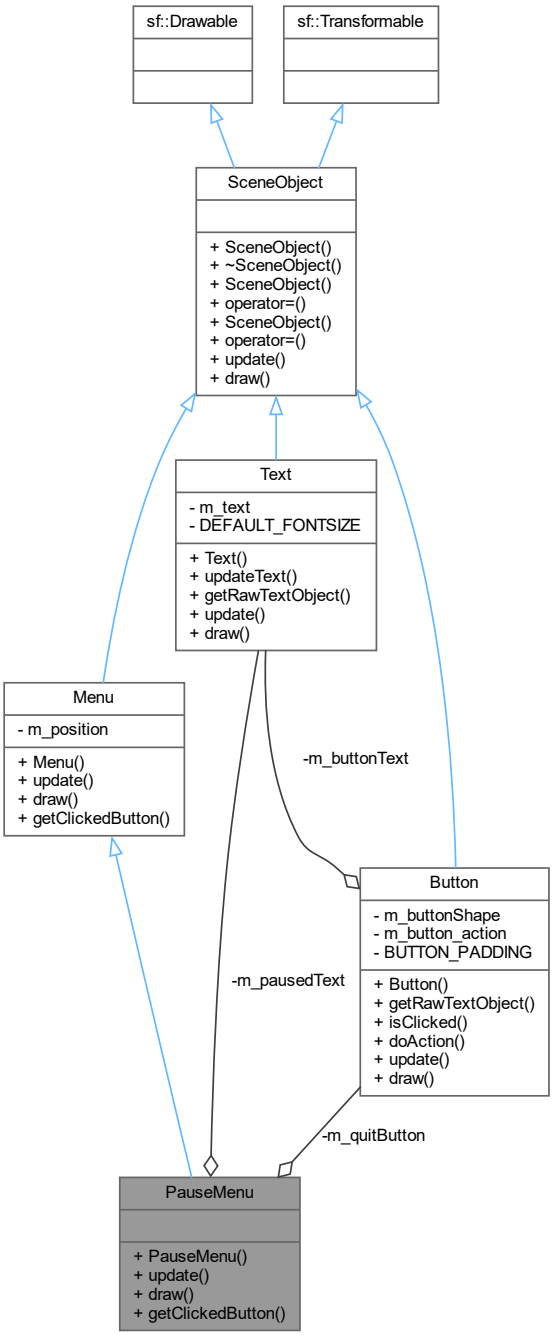
6.12 PauseMenu Class Reference

```
#include <PauseMenu.hpp>
```

Inheritance diagram for PauseMenu:



Collaboration diagram for PauseMenu:



Public Member Functions

- PauseMenu ()
- auto update (const sf::Time &delta) -> bool override
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override
- auto getClickedButton (const sf::Vector2i &mouse_position, const sf::RenderWindow &window) -> std::optional< Button > override

Public Member Functions inherited from Menu

- Menu ()
- auto update (const sf::Time &delta) -> bool override=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0
- virtual auto getClickedButton (const sf::Vector2i &mouse_position, const sf::RenderWindow &window) -> std::optional< Button >=0

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- Text m_pausedText
- Button m_quitButton

6.12.1 Detailed Description

Definition at line 8 of file PauseMenu.hpp.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 PauseMenu()

```
PauseMenu::PauseMenu ( )
```

Definition at line 7 of file PauseMenu.cpp.

6.12.3 Member Function Documentation

6.12.3.1 draw()

```
void PauseMenu::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [virtual]
```

Implements Menu.

Definition at line 18 of file PauseMenu.cpp.

6.12.3.2 getClickedButton()

```
auto PauseMenu::getClickedButton (
            const sf::Vector2i & mouse_position,
            const sf::RenderWindow & window ) -> std::optional<Button>  [override], [virtual]
```

Implements Menu.

Definition at line 24 of file PauseMenu.cpp.

6.12.3.3 update()

```
auto PauseMenu::update (
            const sf::Time & delta ) -> bool  [override], [virtual]
```

Implements Menu.

Definition at line 13 of file PauseMenu.cpp.

6.12.4 Member Data Documentation

6.12.4.1 m_pausedText

```
Text PauseMenu::m_pausedText  [private]
```

Definition at line 10 of file PauseMenu.hpp.

6.12.4.2 m_quitButton

```
Button PauseMenu::m_quitButton  [private]
```

Definition at line 11 of file PauseMenu.hpp.

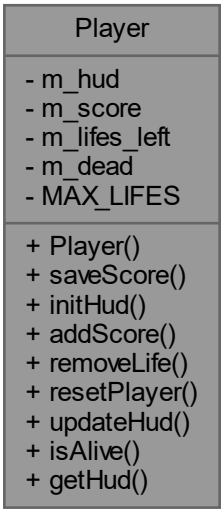
The documentation for this class was generated from the following files:

- tmp/PauseMenu.hpp
- tmp/PauseMenu.cpp

6.13 Player Class Reference

```
#include <Player.hpp>
```

Collaboration diagram for Player:



Public Member Functions

- Player ()
- void saveScore ()
- void initHud ()
- void addScore (int score)
- void removeLife ()
- void resetPlayer ()
- void updateHud ()
- auto isAlive () const -> bool
- auto getHud () -> std::shared_ptr< GameHud >

Private Attributes

- std::shared_ptr< GameHud > m_hud
- int m_score
- int m_lifes_left
- bool m_dead = false

Static Private Attributes

- static constexpr int MAX_LIFES = 3

6.13.1 Detailed Description

Definition at line 10 of file Player.hpp.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 Player()

```
Player::Player ( )
```

Definition at line 6 of file Player.cpp.

6.13.3 Member Function Documentation

6.13.3.1 addScore()

```
void Player::addScore (
            int score )
```

Definition at line 19 of file Player.cpp.

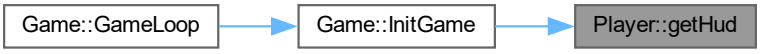
Here is the call graph for this function:

6.13.3.2 getHud()

```
auto Player::getHud ( ) -> std::shared_ptr<GameHud>
```

Definition at line 57 of file Player.cpp.

Here is the caller graph for this function:



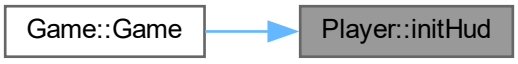
6.13.3.3 initHud()

```
void Player::initHud ( )
```

Definition at line 12 of file Player.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

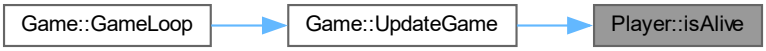


6.13.3.4 isAlive()

```
auto Player::isAlive ( ) const -> bool
```

Definition at line 52 of file Player.cpp.

Here is the caller graph for this function:



6.13.3.5 removeLife()

```
void Player::removeLife ( )
```

Definition at line 25 of file Player.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.3.6 resetPlayer()

```
void Player::resetPlayer ( )
```

Definition at line 38 of file Player.cpp.

Here is the call graph for this function:

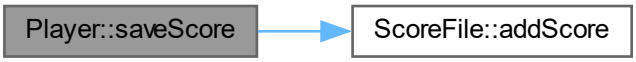
Here is the caller graph for this function:

6.13.3.7 saveScore()

```
void Player::saveScore ( )
```

Definition at line 8 of file Player.cpp.

Here is the call graph for this function:



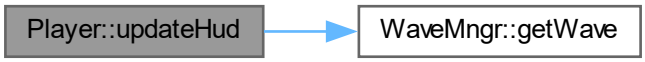
Here is the caller graph for this function:

6.13.3.8 updateHud()

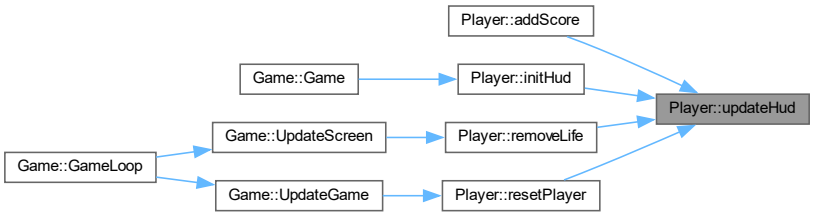
```
void Player::updateHud ( )
```

Definition at line 45 of file Player.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.4 Member Data Documentation

6.13.4.1 m_dead

```
bool Player::m_dead = false  [private]
```

Definition at line 19 of file Player.hpp.

6.13.4.2 m_hud

```
std::shared_ptr<GameHud> Player::m_hud  [private]
```

Definition at line 15 of file Player.hpp.

6.13.4.3 m_lifes_left

```
int Player::m_lifes_left  [private]
```

Definition at line 18 of file Player.hpp.

6.13.4.4 m_score

```
int Player::m_score  [private]
```

Definition at line 17 of file Player.hpp.

6.13.4.5 MAX_LIFES

```
constexpr int Player::MAX_LIFES = 3  [static], [constexpr], [private]
```

Definition at line 12 of file Player.hpp.

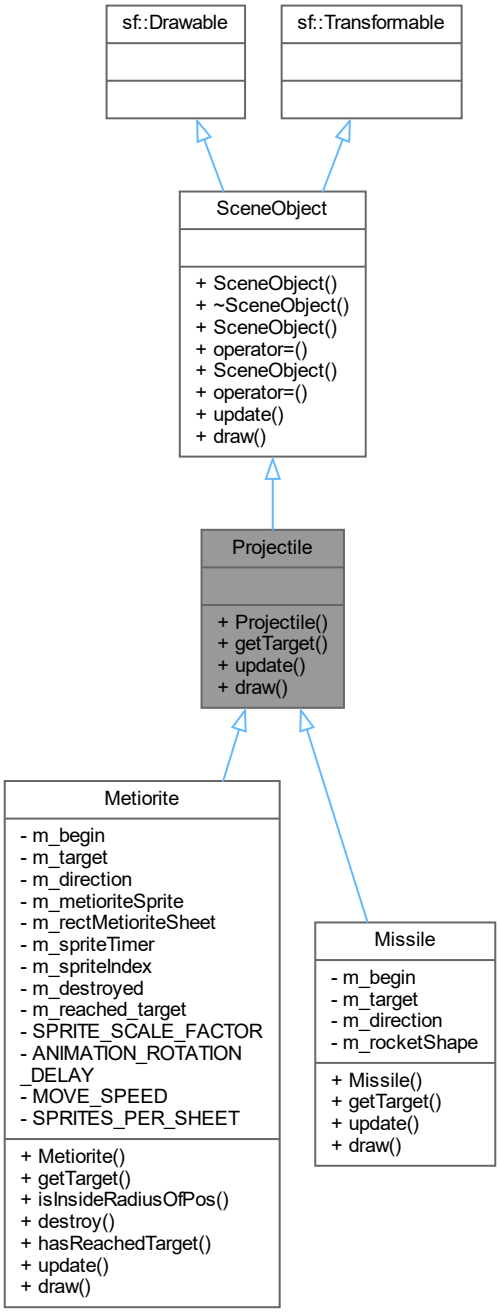
The documentation for this class was generated from the following files:

- tmp/Player.hpp
- tmp/Player.cpp

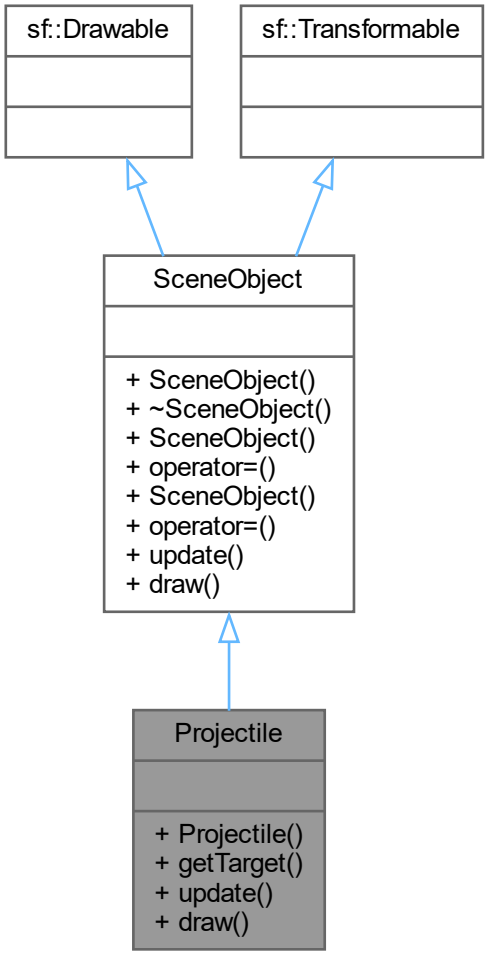
6.14 Projectile Class Reference

```
#include <Projectile.hpp>
```

Inheritance diagram for Projectile:



Collaboration diagram for Projectile:



Public Member Functions

- Projectile (const sf::Vector2f &position)
- virtual auto getTarget () const -> sf::Vector2f=0
- auto update (const sf::Time &delta) -> bool override=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

6.14.1 Detailed Description

Definition at line 4 of file Projectile.hpp.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Projectile()

```
Projectile::Projectile (
            const sf::Vector2f & position )
```

Definition at line 3 of file Projectile.cpp.

6.14.3 Member Function Documentation

6.14.3.1 draw()

```
void Projectile::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [pure
virtual]
```

Implements SceneObject.

Implemented in Metiorite, and Missile.

6.14.3.2 getTarget()

```
virtual auto Projectile::getTarget ( ) const -> sf::Vector2f  [pure virtual]
```

Implemented in Metiorite, and Missile.

6.14.3.3 update()

```
auto Projectile::update (
            const sf::Time & delta ) -> bool  [override], [pure virtual]
```

Implements SceneObject.

Implemented in Metiorite, and Missile.

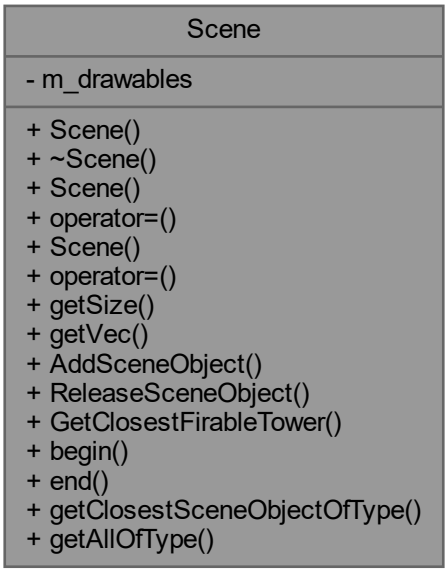
The documentation for this class was generated from the following files:

- tmp/Projectile.hpp
- tmp/Projectile.cpp

6.15 Scene Class Reference

`#include <Scene.hpp>`

Collaboration diagram for Scene:



Public Member Functions

- Scene ()
- ~Scene ()
- Scene (const Scene &other)=delete
- auto operator= (const Scene &other) -> Scene &=delete
- Scene (Scene &&other)=delete
- auto operator= (Scene &&other) -> Scene &=delete
- auto getSize () const -> size_t
- auto getVec () -> std::vector< std::shared_ptr< SceneObject > >
- auto AddSceneObject (std::shared_ptr< SceneObject > Obj) -> bool
- auto ReleaseSceneObject (std::shared_ptr< SceneObject > Obj) -> bool
- auto GetClosestFirableTower (sf::Vector2f pos) -> std::shared_ptr< Tower >
- auto begin () -> std::vector< std::shared_ptr< SceneObject > >::iterator
- auto end () -> std::vector< std::shared_ptr< SceneObject > >::iterator
- template<typename T >
  auto getClosestSceneObjectOfType (sf::Vector2f pos) -> std::shared_ptr< T >
- template<typename T >
  auto getAllOfType () -> std::vector< std::shared_ptr< T > >

Private Attributes

- std::vector< std::shared_ptr< SceneObject > > m_drawables

6.15.1 Detailed Description

Definition at line 11 of file Scene.hpp.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 Scene() [1/3]

```
Scene::Scene ( )  [default]
```

6.15.2.2 ~Scene()

```
Scene::~Scene ( )
```

Definition at line 6 of file Scene.cpp.

6.15.2.3 Scene() [2/3]

```
Scene::Scene (
            const Scene & other )  [delete]
```

6.15.2.4 Scene() [3/3]

```
Scene::Scene (
            Scene && other )  [delete]
```

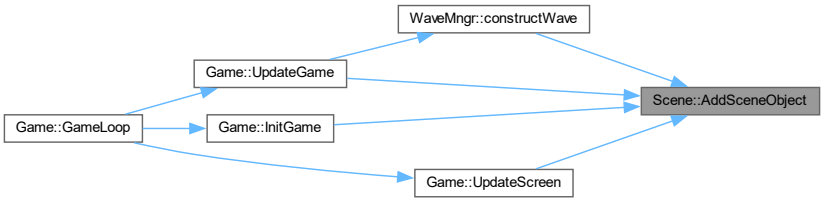
6.15.3 Member Function Documentation

6.15.3.1 AddSceneObject()

```
auto Scene::AddSceneObject (
            std::shared_ptr< SceneObject > Obj ) -> bool
```

Definition at line 18 of file Scene.cpp.

Here is the caller graph for this function:



6.15.3.2 begin()

```
auto Scene::begin ( ) -> std::vector<std::shared_ptr<SceneObject>>::iterator   [inline]
```

Definition at line 31 of file Scene.hpp.

6.15.3.3 end()

```
auto Scene::end ( ) -> std::vector<std::shared_ptr<SceneObject>>::iterator   [inline]
```

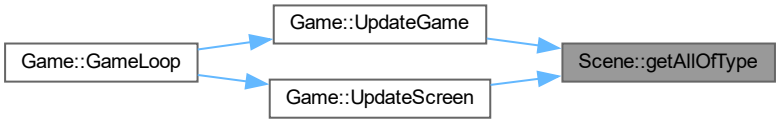
Definition at line 36 of file Scene.hpp.

6.15.3.4 getAllOfType()

```
template<typename T >
auto Scene::getAllOfType ( ) -> std::vector<std::shared_ptr<T>>   [inline]
```

Definition at line 62 of file Scene.hpp.

Here is the caller graph for this function:

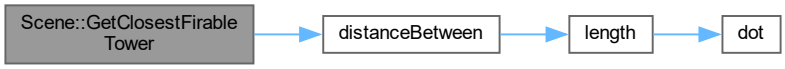


6.15.3.5 GetClosestFirableTower()

```
auto Scene::GetClosestFirableTower (
            sf::Vector2f pos ) -> std::shared_ptr<Tower>
```

Definition at line 50 of file Scene.cpp.

Here is the call graph for this function:



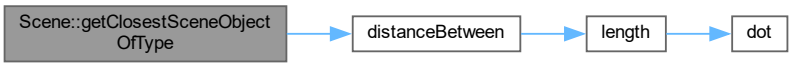
Here is the caller graph for this function:

6.15.3.6 getClosestSceneObjectOfType()

```
template<typename T >
auto Scene::getClosestSceneObjectOfType (
            sf::Vector2f pos ) -> std::shared_ptr<T>   [inline]
```

Definition at line 42 of file Scene.hpp.

Here is the call graph for this function:



6.15.3.7 getSize()

```
auto Scene::getSize ( ) const -> size_t
```

Definition at line 8 of file Scene.cpp.

Here is the caller graph for this function:

6.15.3.8 getVec()

```
auto Scene::getVec ( ) -> std::vector<std::shared_ptr<SceneObject>>
```

Definition at line 13 of file Scene.cpp.

Here is the caller graph for this function:

6.15.3.9 operator=() [1/2]

```
auto Scene::operator= (
            const Scene & other ) -> Scene &=delete  [delete]
```

6.15.3.10 operator=() [2/2]

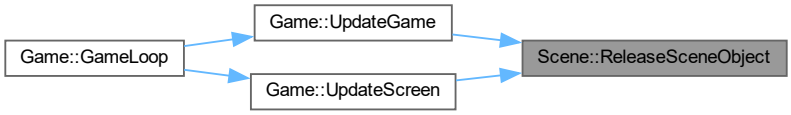
```
auto Scene::operator= (
            Scene && other ) -> Scene &=delete  [delete]
```

6.15.3.11 ReleaseSceneObject()

```
auto Scene::ReleaseSceneObject (
            std::shared_ptr< SceneObject > Obj ) -> bool
```

Definition at line 35 of file Scene.cpp.

Here is the caller graph for this function:



6.15.4 Member Data Documentation

6.15.4.1 m_drawables

```
std::vector<std::shared_ptr<SceneObject> > Scene::m_drawables  [private]
```

Definition at line 13 of file Scene.hpp.

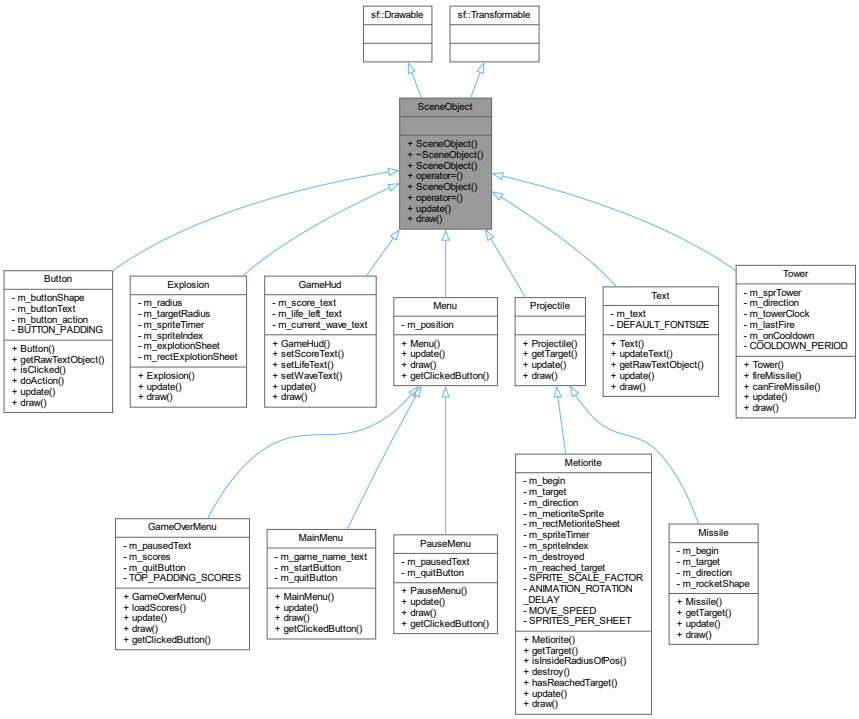
The documentation for this class was generated from the following files:

- tmp/Scene.hpp
- tmp/Scene.cpp

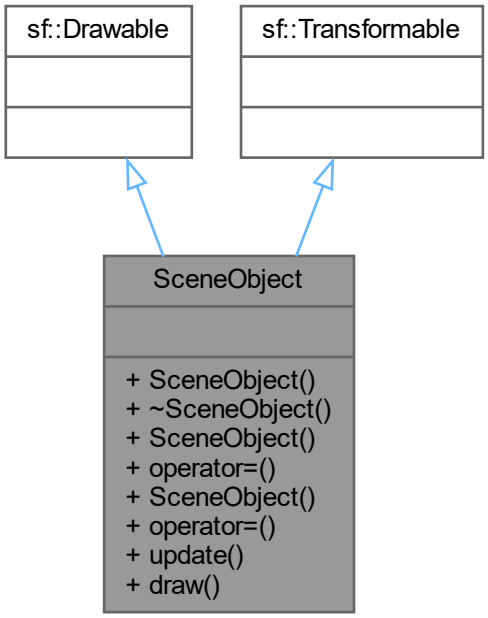
6.16 SceneObject Class Reference

```
#include <SceneObject.hpp>
```

Inheritance diagram for SceneObject:



Collaboration diagram for SceneObject:



Public Member Functions

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

6.16.1 Detailed Description

Definition at line 5 of file SceneObject.hpp.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 SceneObject() [1/3]

```
SceneObject::SceneObject (
            const sf::Vector2f & position )
```

Definition at line 3 of file SceneObject.cpp.

6.16.2.2 ~SceneObject()

```
SceneObject::~SceneObject ( )  [override], [default]
```

6.16.2.3 SceneObject() [2/3]

```
SceneObject::SceneObject (
            SceneObject & object )  [default]
```

6.16.2.4 SceneObject() [3/3]

```
SceneObject::SceneObject (
            SceneObject && object )  [default]
```

6.16.3 Member Function Documentation

6.16.3.1 draw()

```
void SceneObject::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [pure
virtual]
```

Implemented in Button, Explosion, GameHud, GameOverMenu, MainMenu, Metiorite, Missile, PauseMenu, Text, Tower, Menu, and Projectile.

6.16.3.2 operator=() [1/2]

```
auto SceneObject::operator= (
            SceneObject && object ) -> SceneObject &=default  [default]
```

6.16.3.3 operator=() [2/2]

```
auto SceneObject::operator= (
            SceneObject const & object ) -> SceneObject &=default  [default]
```

6.16.3.4 update()

```
virtual auto SceneObject::update (
            const sf::Time & delta ) -> bool  [pure virtual]
```

Implemented in Button, Explosion, GameHud, GameOverMenu, MainMenu, Metiorite, Missile, PauseMenu, Text, Tower, Menu, and Projectile.

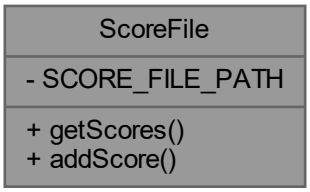
The documentation for this class was generated from the following files:

- tmp/SceneObject.hpp
- tmp/SceneObject.cpp

6.17 ScoreFile Class Reference

`#include <ScoreFile.hpp>`

Collaboration diagram for ScoreFile:



Static Public Member Functions

- static auto getScores () -> std::vector< std::string >
- static void addScore (int score)

Static Private Attributes

- static constexpr char SCORE_FILE_PATH [ ] = "scores.txt"

6.17.1 Detailed Description

Definition at line 7 of file ScoreFile.hpp.

6.17.2 Member Function Documentation

6.17.2.1 addScore()

```
void ScoreFile::addScore (
            int score )  [static]
```

Definition at line 21 of file ScoreFile.cpp.

Here is the caller graph for this function:

6.17.2.2 getScores()

```
auto ScoreFile::getScores ( ) -> std::vector<std::string>  [static]
```

Definition at line 3 of file ScoreFile.cpp.

Here is the caller graph for this function:

6.17.3 Member Data Documentation

6.17.3.1 SCORE_FILE_PATH

```
constexpr char ScoreFile::SCORE_FILE_PATH[] = "scores.txt"  [static], [constexpr], [private]
```

Definition at line 9 of file ScoreFile.hpp.

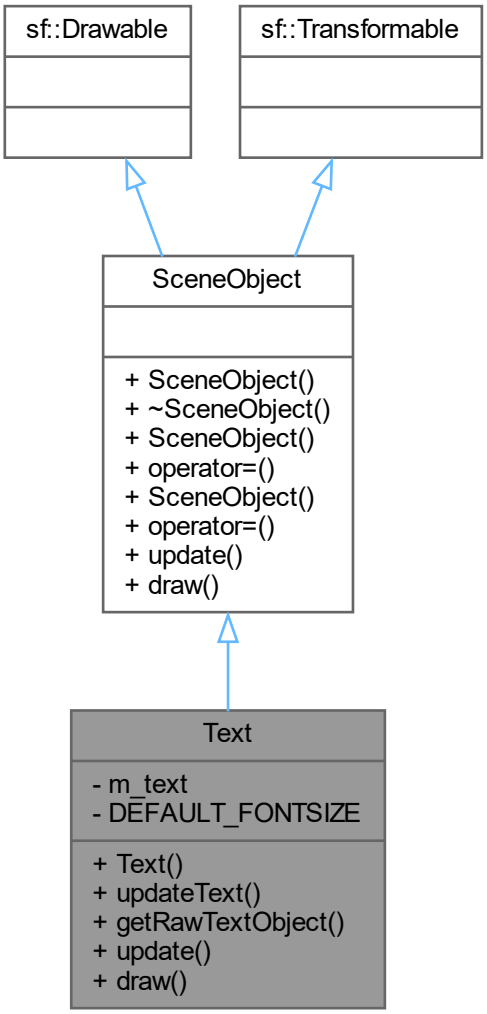
The documentation for this class was generated from the following files:

- tmp/ScoreFile.hpp
- tmp/ScoreFile.cpp

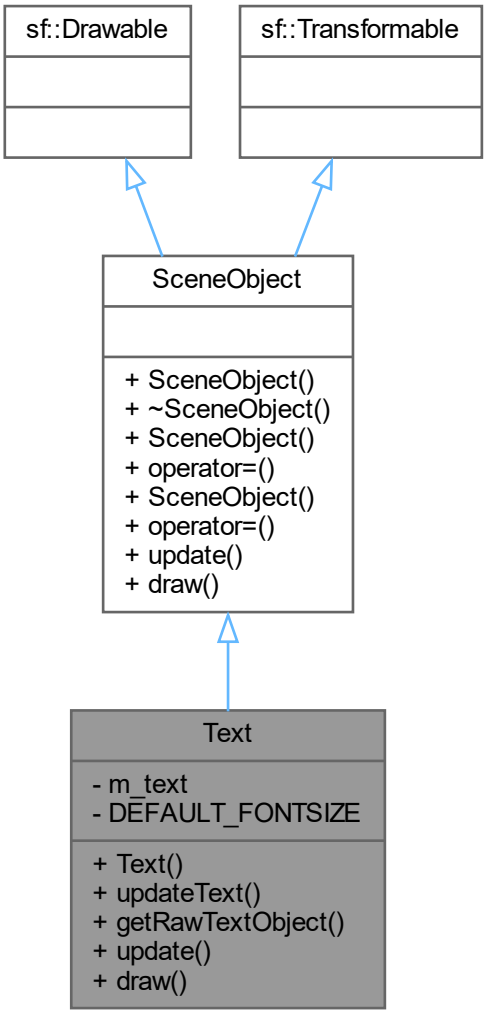
6.18 Text Class Reference

```
#include <Text.hpp>
```

Inheritance diagram for Text:



Collaboration diagram for Text:



Public Member Functions

- Text (const sf::Vector2f &position, const std::string &_text, int fontSize=DEFAULT_FONTSIZE)
- void updateText (const std::string &_text)
- auto getRawTextObject () -> sf::Text &
- auto update (const sf::Time &delta) -> bool override
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)
- ~SceneObject () override
- SceneObject (SceneObject &object)=default

- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- sf::Text m_text

Static Private Attributes

- static constexpr int DEFAULT_FONTSIZE = 24

6.18.1 Detailed Description

Definition at line 5 of file Text.hpp.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Text()

```
Text::Text (
            const sf::Vector2f & position,
            const std::string & _text,
            int fontSize = DEFAULT_FONTSIZE )
```

Definition at line 3 of file Text.cpp.

6.18.3 Member Function Documentation

6.18.3.1 draw()

```
void Text::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [virtual]
```

Implements SceneObject.

Definition at line 39 of file Text.cpp.

6.18.3.2 getRawTextObject()

```
auto Text::getRawTextObject ( ) -> sf::Text&
```

Definition at line 29 of file Text.cpp.

Here is the caller graph for this function:

6.18.3.3 update()

```
auto Text::update (
            const sf::Time & delta ) -> bool  [override], [virtual]
```

Implements SceneObject.

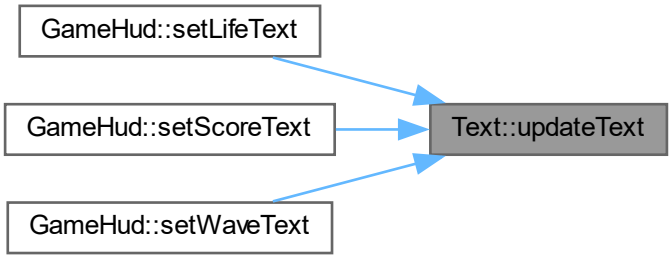
Definition at line 34 of file Text.cpp.

6.18.3.4 updateText()

```
void Text::updateText (
            const std::string & _text )
```

Definition at line 24 of file Text.cpp.

Here is the caller graph for this function:



6.18.4 Member Data Documentation

6.18.4.1 DEFAULT_FONTSIZE

```
constexpr int Text::DEFAULT_FONTSIZE = 24  [static], [constexpr], [private]
```

Definition at line 7 of file Text.hpp.

6.18.4.2 m_text

```
sf::Text Text::m_text  [private]
```

Definition at line 9 of file Text.hpp.

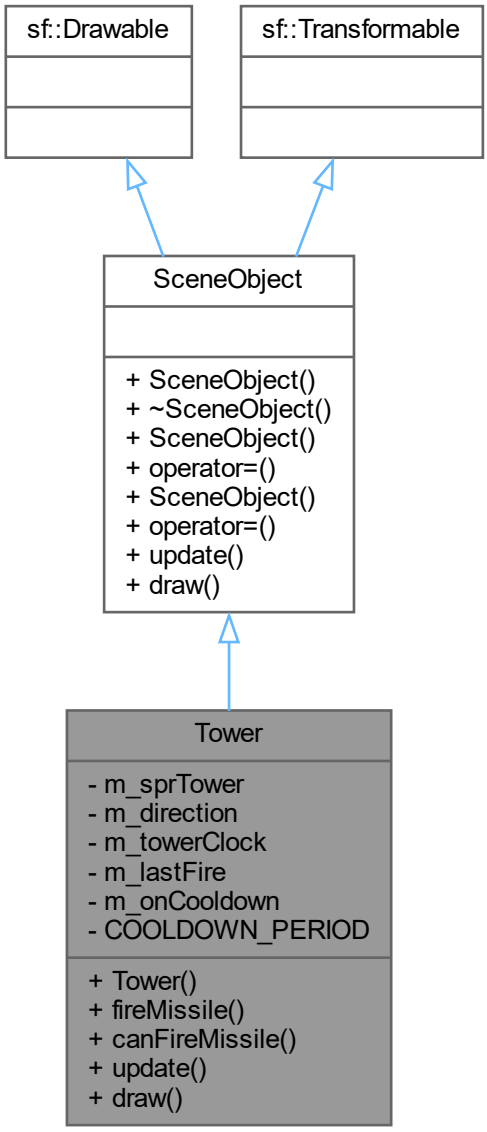
The documentation for this class was generated from the following files:

- tmp/Text.hpp
- tmp/Text.cpp

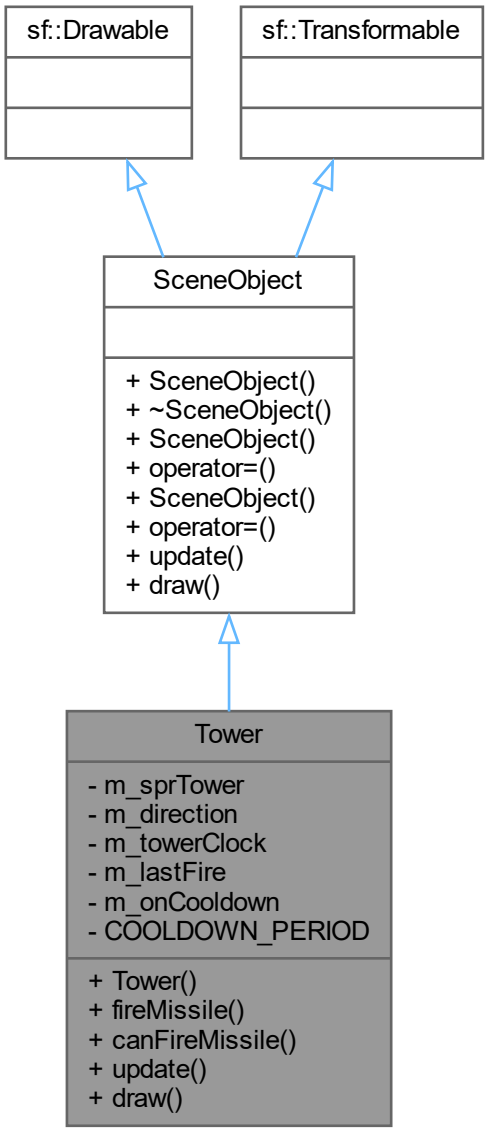
6.19 Tower Class Reference

```
#include <Tower.hpp>
```

Inheritance diagram for Tower:



Collaboration diagram for Tower:



Public Member Functions

- Tower (sf::Vector2f pos)
- void fireMissile ()
- auto canFireMissile () const -> bool
- auto update (const sf::Time &delta) -> bool override
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override

Public Member Functions inherited from SceneObject

- SceneObject (const sf::Vector2f &position)

- ~SceneObject () override
- SceneObject (SceneObject &object)=default
- auto operator= (SceneObject const &object) -> SceneObject &=default
- SceneObject (SceneObject &&object)=default
- auto operator= (SceneObject &&object) -> SceneObject &=default
- virtual auto update (const sf::Time &delta) -> bool=0
- void draw (sf::RenderTarget &target, sf::RenderStates states=sf::RenderStates::Default) const override=0

Private Attributes

- sf::Sprite m_sprTower
- sf::Vector2f m_direction
- sf::Clock m_towerClock
- sf::Time m_lastFire
- bool m_onCooldown = false

Static Private Attributes

- static constexpr float COOLDOWN_PERIOD = 1.0F

6.19.1 Detailed Description

Definition at line 7 of file Tower.hpp.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 Tower()

```
Tower::Tower (
            sf::Vector2f pos )
```

Definition at line 3 of file Tower.cpp.

6.19.3 Member Function Documentation

6.19.3.1 canFireMissile()

```
auto Tower::canFireMissile ( ) const -> bool
```

Definition at line 25 of file Tower.cpp.

6.19.3.2 draw()

```
void Tower::draw (
            sf::RenderTarget & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [override], [virtual]
```

Implements SceneObject.

Definition at line 43 of file Tower.cpp.

6.19.3.3 fireMissile()

```
void Tower::fireMissile ( )
```

Definition at line 15 of file Tower.cpp.

6.19.3.4 update()

```
auto Tower::update (
            const sf::Time & delta ) -> bool  [override], [virtual]
```

Implements SceneObject.

Definition at line 30 of file Tower.cpp.

6.19.4 Member Data Documentation

6.19.4.1 COOLDOWN_PERIOD

```
constexpr float Tower::COOLDOWN_PERIOD = 1.0F  [static], [constexpr], [private]
```

Definition at line 9 of file Tower.hpp.

6.19.4.2 m_direction

```
sf::Vector2f Tower::m_direction  [private]
```

Definition at line 11 of file Tower.hpp.

6.19.4.3 m_lastFire

```
sf::Time Tower::m_lastFire  [private]
```

Definition at line 13 of file Tower.hpp.

6.19.4.4 m_onCooldown

```
bool Tower::m_onCooldown = false  [private]
```

Definition at line 14 of file Tower.hpp.

6.19.4.5 m_sprTower

```
sf::Sprite Tower::m_sprTower  [private]
```

Definition at line 10 of file Tower.hpp.

6.19.4.6 m_towerClock

```
sf::Clock Tower::m_towerClock  [private]
```

Definition at line 12 of file Tower.hpp.

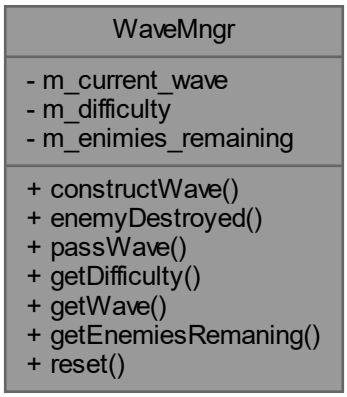
The documentation for this class was generated from the following files:

- tmp/Tower.hpp
- tmp/Tower.cpp

6.20 WaveMngr Class Reference

```
#include <WaveMngr.hpp>
```

Collaboration diagram for WaveMngr:



Static Public Member Functions

- static void constructWave (Scene &gameScene)
- static void enemyDestroyed ()
- static void passWave ()
- static auto getDifficulty () -> int
- static auto getWave () -> int
- static auto getEnemiesRemaning () -> int
- static void reset ()

Static Private Attributes

- static int m_current_wave = 0
- static int m_difficulty = 1
- static int m_enimies_remaining = 0

6.20.1 Detailed Description

Definition at line 9 of file WaveMngr.hpp.

6.20.2 Member Function Documentation

6.20.2.1 constructWave()

```
void WaveMngr::constructWave (
            Scene & gameScene )  [static]
```

Definition at line 11 of file WaveMngr.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.20.2.2 enemyDestroyed()

```
void WaveMngr::enemyDestroyed ( )  [static]
```

Definition at line 24 of file WaveMngr.cpp.

Here is the caller graph for this function:

6.20.2.3 getDifficulty()

```
auto WaveMngr::getDifficulty ( ) -> int  [static]
```

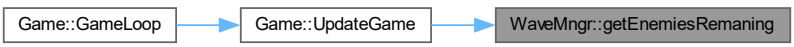
Definition at line 34 of file WaveMngr.cpp.

6.20.2.4 getEnemiesRemaning()

```
auto WaveMngr::getEnemiesRemaning ( ) -> int  [static]
```

Definition at line 44 of file WaveMngr.cpp.

Here is the caller graph for this function:

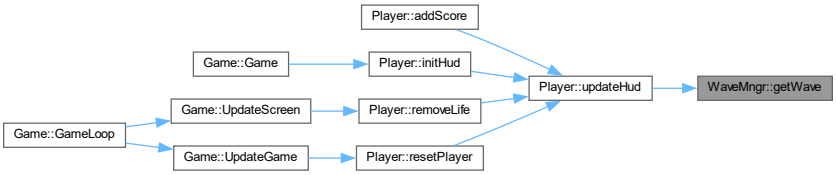


6.20.2.5 getWave()

```
auto WaveMngr::getWave ( ) -> int  [static]
```

Definition at line 39 of file WaveMngr.cpp.

Here is the caller graph for this function:



6.20.2.6 passWave()

```
void WaveMngr::passWave ( )  [static]
```

Definition at line 29 of file WaveMngr.cpp.

Here is the caller graph for this function:

6.20.2.7 reset()

```
void WaveMngr::reset ( )  [static]
```

Definition at line 49 of file WaveMngr.cpp.

Here is the caller graph for this function:

6.20.3 Member Data Documentation

6.20.3.1 m_current_wave

```
int WaveMngr::m_current_wave = 0  [static], [private]
```

Definition at line 11 of file WaveMngr.hpp.

6.20.3.2 m_difficulty

```
int WaveMngr::m_difficulty = 1  [static], [private]
```

Definition at line 12 of file WaveMngr.hpp.

6.20.3.3 m_enimies_remaining

```
int WaveMngr::m_enimies_remaining = 0  [static], [private]
```

Definition at line 13 of file WaveMngr.hpp.

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

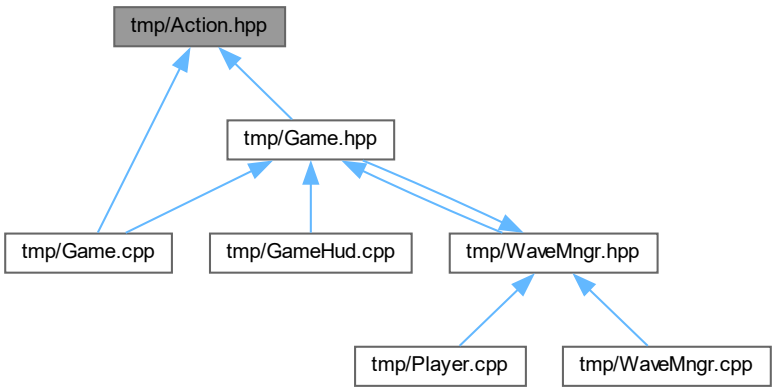
- tmp/WaveMngr.hpp
- tmp/WaveMngr.cpp

Chapter 7

File Documentation

7.1 tmp/Action.hpp File Reference

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



Namespaces

- namespace Action

Enumerations

- enum Action::Action {
  Action::LMouse , Action::RMouse , Action::Space , Action::Pause ,
  Action::Exit }

7.2 Action.hpp

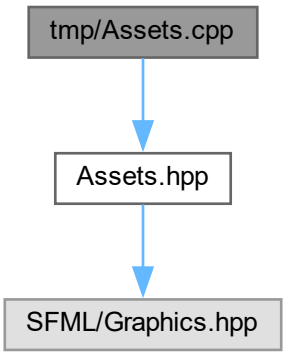
Go to the documentation of this file.

```
00001 #pragma once
00002
00003 namespace Action
00004 {
00005     enum Action
00006     {
00007         LMouse,
00008         RMouse,
00009         Space,
00010         Pause,
00011         Exit
00012     };
00013 }
```

7.3 tmp/Assets.cpp File Reference

`#include "Assets.hpp"`

Include dependency graph for Assets.cpp:



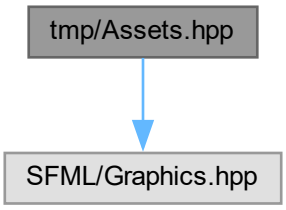
7.4 Assets.cpp

Go to the documentation of this file.

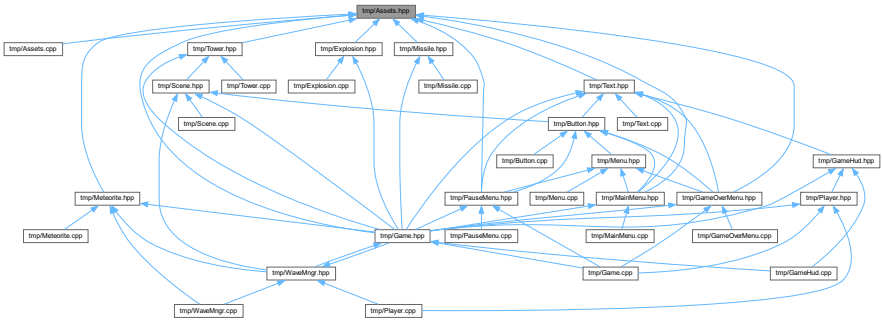
```
00001 #include "Assets.hpp"
00002
00003 sf::Texture Assets::tower; // Init the static Assets
00004 sf::Texture Assets::missile;
00005 sf::Texture Assets::background;
00006 sf::Texture Assets::explotion_sheet;
00007 sf::Texture Assets::metiorite_sheet;
00008 sf::Font Assets::text_font;
```

7.5 tmp/Assets.hpp File Reference

`#include <SFML/Graphics.hpp>`
Include dependency graph for Assets.hpp:



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



Classes

- class Assets

7.6 Assets.hpp

Go to the documentation of this file.

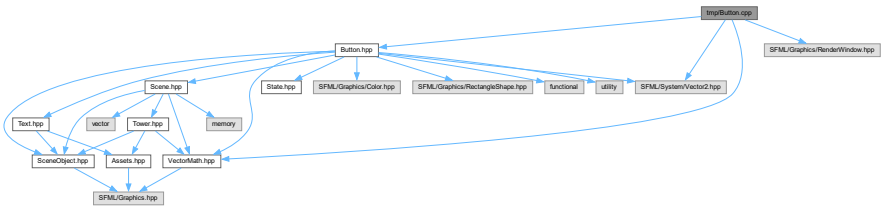
```
00001 #pragma once
00002 #include <SFML/Graphics.hpp>
00003
00004 // This exists to avoid loading the Assets multiple times :D
00005
00006 class Assets
00007 {
00008   public:
00009     static sf::Texture tower;
00010     static sf::Texture missile;
00011     static sf::Texture background;
00012     static sf::Texture explotion_sheet;
00013     static sf::Texture metiorite_sheet;
00014     static sf::Font text_font;
00015     static void loadAssets()
```

```
00016     {
00017 #ifdef __linux__
00018         tower.loadFromFile("../assets/Tower.png"); // Since its compiled from build folder instead
00019                                                    // of root we need to jump one more step out
00020         missile.loadFromFile("../assets/Missile.png");
00021         background.loadFromFile("../assets/background.jpg");
00022         explotionSheet.loadFromFile("../assets/explotionsheet.png");
00023         metioriteSheet.loadFromFile("../assets/metioritesheet.png");
00024         textFont.loadFromFile("../assets/NFagave.ttf");
00025 #elif _WIN32
00026         tower.loadFromFile("assets/Tower.png");
00027         missile.loadFromFile("assets/Missile.png");
00028         background.loadFromFile("assets/background.jpg");
00029         explotion_sheet.loadFromFile("assets/explotionsheet.png");
00030         metiorite_sheet.loadFromFile("assets/metioritesheet.png");
00031         text_font.loadFromFile("assets/NFagave.ttf");
00032 #else
00033         tower.loadFromFile("assets/Tower.png");
00034         missile.loadFromFile("assets/Missile.png");
00035         background.loadFromFile("assets/background.jpg");
00036         explotionSheet.loadFromFile("assets/explotionsheet.png");
00037         metioriteSheet.loadFromFile("assets/metioritesheet.png");
00038         textFont.loadFromFile("assets/NFagave.ttf");
00039 #endif
00040     }
00041 };
```

7.7 tmp/Button.cpp File Reference

```
#include "Button.hpp"
#include "VectorMath.hpp"
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/System/Vector2.hpp>
```

Include dependency graph for Button.cpp:



7.8 Button.cpp

Go to the documentation of this file.

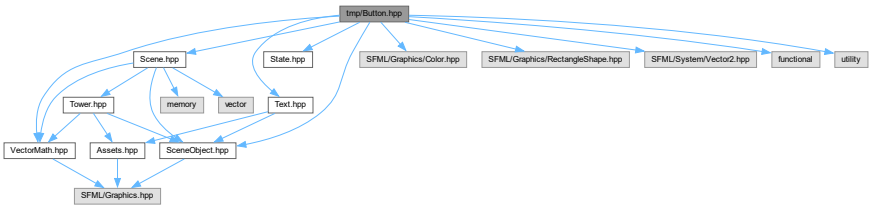
```
00001 #include "Button.hpp"
00002 #include "VectorMath.hpp"
00003 #include <SFML/Graphics/RenderWindow.hpp>
00004 #include <SFML/System/Vector2.hpp>
00005
00006 Button::Button(
00007     const sf::Vector2f& position,
00008     const std::string& _text,
00009     std::function<State::State(void)> _button_action)
00010     : SceneObject(position)
00011     , m_buttonText(position, _text)
00012     , m_button_action(std::move(_button_action))
00013
00014 {
00015     sf::FloatRect text_bounds = m_buttonText.getRawTextObject().getGlobalBounds();
00016     m_buttonShape = sf::RectangleShape(sf::Vector2f(
00017         text_bounds.width + (2 * BUTTON_PADDING),
00018         text_bounds.height + (2 * BUTTON_PADDING)));
00019     m_buttonShape.setOutlineColor(sf::Color::White);
00020     m_buttonShape.setFillColor(sf::Color::Black);
```

```
00021     m_buttonShape.setOutlineThickness(2.F);
00022     m_buttonShape.setPosition(text_bounds.left - BUTTON_PADDING, text_bounds.top - BUTTON_PADDING);
00023 }
00024
00025 auto Button::isClicked(const sf::Vector2i& mouse_position, const sf::RenderWindow& window) -> bool
00026 {
00027     auto translated_pos = window.mapPixelToCoords(mouse_position);
00028     return m_buttonShape.getGlobalBounds().contains(translated_pos);
00029 }
00030
00031 auto Button::doAction() -> State::State
00032 {
00033     return m_button_action();
00034 }
00035
00036 auto Button::getRawTextObject() -> sf::Text&
00037 {
00038     return m_buttonText.getRawTextObject();
00039 }
00040
00041 auto Button::update(const sf::Time& /*delta*/) -> bool
00042 {
00043     return true;
00044 }
00045
00046 void Button::draw(sf::RenderTarget& target, sf::RenderStates states) const
00047 {
00048     target.draw(m_buttonShape, states);
00049     target.draw(m_buttonText, states);
00050 }
```

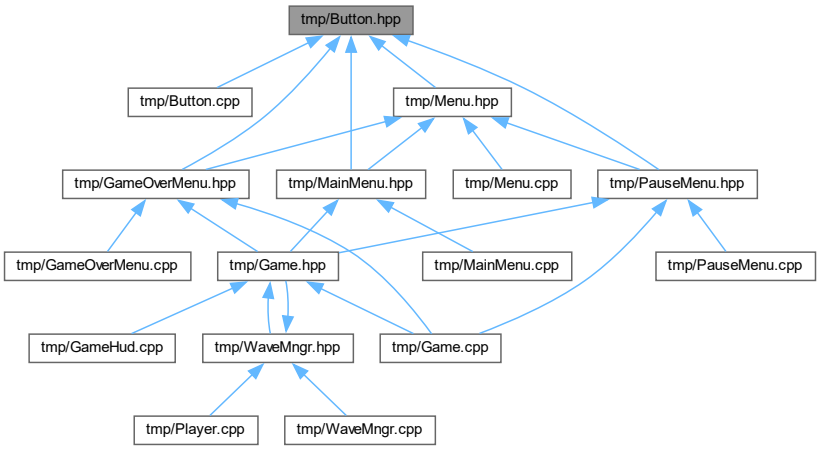
7.9 tmp/Button.hpp File Reference

```
#include "Scene.hpp"
#include "SceneObject.hpp"
#include "State.hpp"
#include "Text.hpp"
#include "VectorMath.hpp"
#include <SFML/Graphics/Color.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/System/Vector2.hpp>
#include <functional>
#include <utility>
```

Include dependency graph for Button.hpp:



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



Classes

- class Button

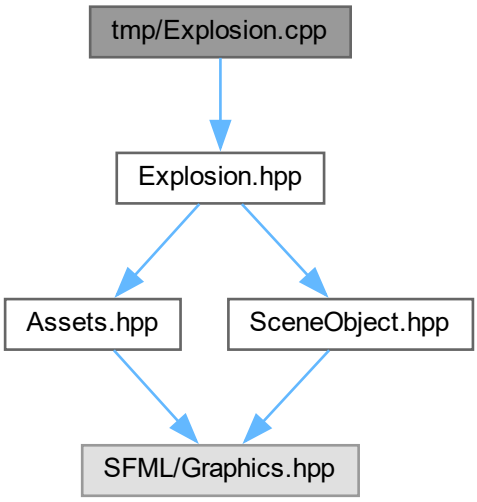
7.10 Button.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "Scene.hpp"
00003 #include "SceneObject.hpp"
00004 #include "State.hpp"
00005 #include "Text.hpp"
00006 #include "VectorMath.hpp"
00007 #include <SFML/Graphics/Color.hpp>
00008 #include <SFML/Graphics/RectangleShape.hpp>
00009 #include <SFML/System/Vector2.hpp>
00010 #include <functional>
00011 #include <utility>
00012
00013 class Button: public SceneObject
00014 {
00015     // WIP Needs implementing
00016     static constexpr float BUTTON_PADDING = 10.F;
00017     sf::RectangleShape m_buttonShape;
00018     Text m_buttonText;
00019     std::function<State::State(void)> m_button_action;
00020
00021   public:
00022     Button(
00023         const sf::Vector2f& position,
00024         const std::string& _text,
00025         std::function<State::State(void)>
00026             _button_action); // Takes in a lambda that defines what the button does
00027
00028     auto getRawTextObject() -> sf::Text&;
00029     auto isClicked(const sf::Vector2i& mouse_position, const sf::RenderWindow& window) -> bool;
00030     auto doAction() -> State::State;
00031
00032     auto update(const sf::Time& delta) -> bool override;
00033     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00034         const override;
00035 };
```

7.11 tmp/Explosion.cpp File Reference

`#include "Explosion.hpp"`
Include dependency graph for Explosion.cpp:



7.12 Explosion.cpp

Go to the documentation of this file.

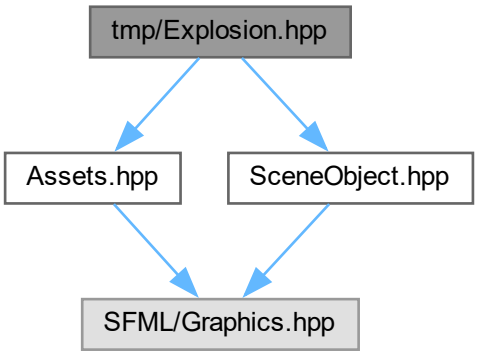
```
00001 #include "Explosion.hpp"
00002
00003 Explosion::Explosion(const sf::Vector2f& position, float radius)
00004     : SceneObject(position)
00005     , m_radius(0)
00006     , m_targetRadius(radius)
00007 {
00008     auto sheet_size = Assets::explotion_sheet.getSize();
00009     m_rectExplotionSheet =
00010         sf::IntRect(0, 0, sheet_size.x / 6, sheet_size.y); // 6 sprites in one sheet
00011     m_explotionSheet = sf::Sprite(Assets::explotion_sheet, m_rectExplotionSheet);
00012     sf::Vector2f adjusted_position;
00013     adjusted_position.x = position.x - ((sheet_size.x / 6.0f) / 2); // texture magic :P
00014     adjusted_position.y = position.y - (sheet_size.y / 2.0f);
00015     m_explotionSheet.setPosition(adjusted_position);
00016 }
00017
00018 auto Explosion::update(const sf::Time& delta) -> bool
00019 {
00020     /*if(m_radius >= m_targetRadius)
00021         return false;*/
00022     m_spriteTimer += delta.asSeconds();
00023     m_radius += 100.0F * delta.asSeconds();
00024
00025     if(m_spriteIndex > 6) // Animation is done
00026         return false;
00027
00028     if(m_spriteTimer > 0.07F)
00029     {
00030         m_rectExplotionSheet.left += Assets::explotion_sheet.getSize().x / 6;
00031         m_spriteTimer = 0.0F;
00032         m_spriteIndex++;
00033     }
00034     m_explotionSheet.setTextureRect(m_rectExplotionSheet);
```

```
00035     return true;
00036 }
00037
00038 void Explosion::draw(sf::RenderTarget& target, sf::RenderStates states) const
00039 {
00040     target.draw(m_explotionSheet, states);
00041 }
```

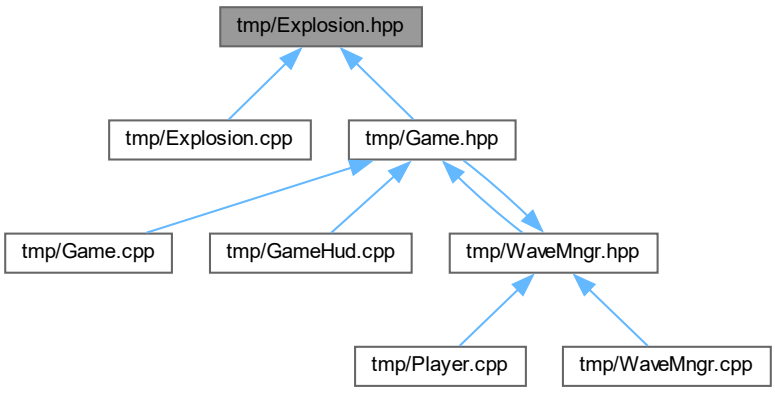
7.13 tmp/Explosion.hpp File Reference

```
#include "Assets.hpp"
#include "SceneObject.hpp"
```

Include dependency graph for Explosion.hpp:



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



Classes

- class Explosion

7.14 Explosion.hpp

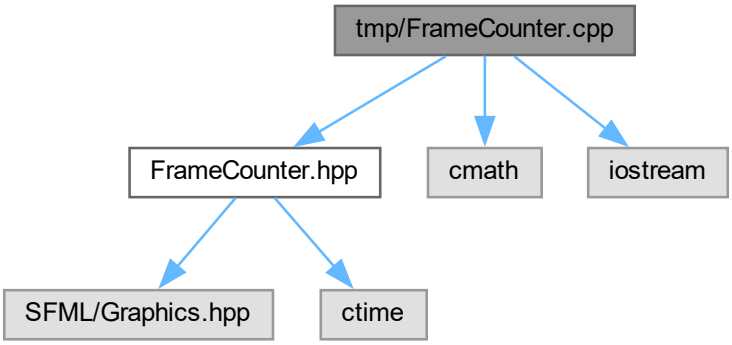
Go to the documentation of this file.

```
00001 #pragma once
00002 #include "Assets.hpp"
00003 #include "SceneObject.hpp"
00004
00005 class Explosion: public SceneObject
00006 {
00007     float m_radius;
00008     float m_targetRadius;
00009     float m_spriteTimer = 0.0F;
00010     int m_spriteIndex = 0;
00011     sf::Sprite m_explotionSheet;
00012     sf::IntRect m_rectExplotionSheet;
00013
00014   public:
00015     Explosion(const sf::Vector2f& position, float radius);
00016
00017     auto update(const sf::Time& delta) -> bool override;
00018     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00019         const override;
00020 };
```

7.15 tmp/FrameCounter.cpp File Reference

```
#include "FrameCounter.hpp"
#include <cmath>
#include <iostream>
```

Include dependency graph for FrameCounter.cpp:



7.16 FrameCounter.cpp

Go to the documentation of this file.

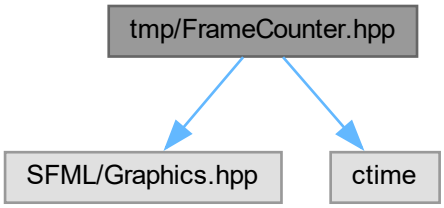
```
00001 #include "FrameCounter.hpp"
00002 #include <cmath>
00003 #include <iostream>
00004
00005 void FrameCounter::updateFps()
00006 {
00007     m_cTime = m_clock.getElapsedTime();
00008     m_fps = 1.0F / (m_cTime.asSeconds() - m_lTime.asSeconds());
00009     m_lTime = m_cTime;
```

```
00010 }
00011
00012 auto FrameCounter::getFps() const -> float
00013 {
00014     return m_fps;
00015 }
00016
00017 void FrameCounter::printFps() const
00018 {
00019     std::cout « "FPS: " « std::floor(m_fps) « std::endl;
00020 }
```

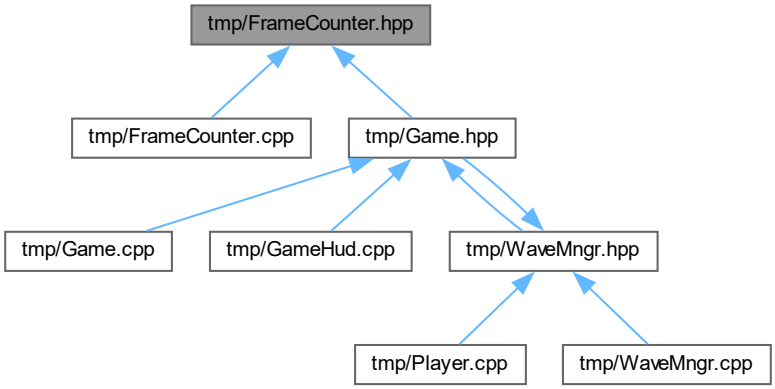
7.17 tmp/FrameCounter.hpp File Reference

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

Include dependency graph for FrameCounter.hpp:



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



Classes

- class FrameCounter

7.18 FrameCounter.hpp

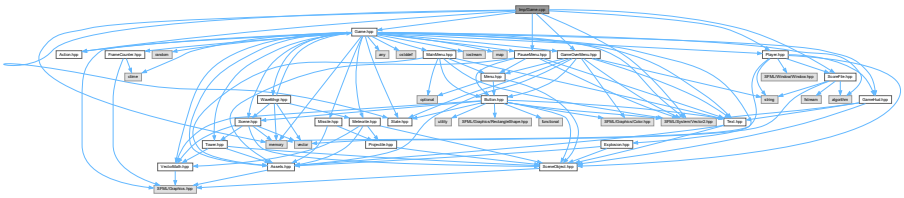
Go to the documentation of this file.

```
00001 #pragma once
00002 #include <SFML/Graphics.hpp>
00003 #include <ctime>
00004
00005 class FrameCounter
00006 {
00007     sf::Clock m_clock;
00008     sf::Time m_lTime = m_clock.getElapsedTime();
00009     sf::Time m_cTime;
00010     float m_fps = 0.0F;
00011
00012   public:
00013     void updateFps();
00014     [[nodiscard]] auto getFps() const -> float;
00015     void printFps() const;
00016 };
```

7.19 tmp/Game.cpp File Reference

```
#include "Action.hpp"
#include "Game.hpp"
#include "GameOverMenu.hpp"
#include "PauseMenu.hpp"
#include "Player.hpp"
#include "ScoreFile.hpp"
#include "State.hpp"
#include <SFML/System/Vector2.hpp>
#include <memory>
```

Include dependency graph for Game.cpp:



7.20 Game.cpp

Go to the documentation of this file.

```
00001 #include "Action.hpp"
00002 #include "Game.hpp"
00003 #include "GameOverMenu.hpp"
00004 #include "PauseMenu.hpp"
00005 #include "Player.hpp"
00006 #include "ScoreFile.hpp"
00007 #include "State.hpp"
00008 #include <SFML/System/Vector2.hpp>
00009 #include <memory>
00010
00011 Game::Game(int _width, int _height)
00012     : m_window(sf::VideoMode(_width, _height, 32), "Missile Command (gradius styled)")
00013 {
00014     height = _height;
00015     width = _width;
00016
00017     Assets::loadAssets(); // Need to load the background before making the sprite
00018     m_backgroundSprite = sf::Sprite(Assets::background);
00019
```

```
00020     m_backgroundSprite.setScale(
00021         width / m_backgroundSprite.getLocalBounds().width,
00022         height / m_backgroundSprite.getLocalBounds().height);
00023     m_backgroundSprite.move(
00024         0,
00025         -30.0F); // Make a black bar at the bottom for text (That looks like gradius)
00026     m_player.initHud();
00027 }
00028
00029 Game::~Game() = default;
00030
00031 void Game::InitGame()
00032 {
00033     srand(time(nullptr)); // Set up the random number gen for use later in WaveMngr.
00034
00035     m_window.setVerticalSyncEnabled(
00036         true); // no need to run the game at lightspeed. this game is a good usecase for vsync.
00037
00038     // Game scene
00039     m_gameScene.AddSceneObject(
00040         std::make_shared<Tower>(sf::Vector2f(width / 6.0F, height - BOTTOM_PADDING))); // Tower left
00041     m_gameScene.AddSceneObject(std::make_shared<Tower>(
00042         sf::Vector2f(width / 2.0F, height - BOTTOM_PADDING))); // Tower middle
00043     m_gameScene.AddSceneObject(std::make_shared<Tower>(
00044         sf::Vector2f(width - (width / 6.0F), height - BOTTOM_PADDING))); // Tower right
00045
00046     m_gameScene.AddSceneObject(m_player.getHud());
00047
00048     // Main menu scene
00049     m_mmenu = std::make_shared<MainMenu>();
00050     m_mainMenuScene.AddSceneObject(m_mmenu);
00051
00052     // Pause menu scene
00053     m_pmenu = std::make_shared<PauseMenu>();
00054     m_pauseMenuScene.AddSceneObject(m_pmenu);
00055
00056     // Game over scene
00057     m_omenu = std::make_shared<GameOverMenu>();
00058     m_gameOverScene.AddSceneObject(m_omenu);
00059 }
00060
00061 void Game::HandleInput()
00062 {
00063     sf::Event event{};
00064     while(m_window.pollEvent(event))
00065     {
00066         if(event.type == sf::Event::Closed)
00067         {
00068             m_window.close();
00069         }
00070         else if(event.type == sf::Event::KeyPressed)
00071         {
00072             if(event.key.code == sf::Keyboard::Escape)
00073             {
00074                 m_gameState = (m_gameState == State::InGame) ? State::Pause : State::InGame;
00075             }
00076             if(event.key.code == sf::Keyboard::Space)
00077             {
00078                 m_inputBuffer.insert(std::pair(Action::Space, nullptr));
00079             }
00080         }
00081         else if(event.type == sf::Event::MouseButtonPressed)
00082         {
00083             if(event.mouseButton.button == sf::Mouse::Left)
00084             {
00085                 m_inputBuffer.insert(std::pair(Action::LMouse, sf::Mouse::getPosition(m_window)));
00086             }
00087         }
00088     }
00089 }
00090
00091 void Game::UpdateGame()
00092 {
00093     switch(m_gameState)
00094     {
00095         case State::State::InGame:
00096             for(auto itr = m_inputBuffer.begin(); itr != m_inputBuffer.end();) // ACtions
00097             {
00098                 if(itr->first == Action::LMouse) // Action is shoot and any is mousePos
00099                 {
00100                     auto mouse_position = vec2iToVec2f(std::any_cast<sf::Vector2i>(itr->second));
00101                     auto closest_tower_ptr = m_gameScene.GetClosestFirableTower(mouse_position);
00102                     if(closest_tower_ptr != nullptr)
00103                     {
00104                         closest_tower_ptr->fireMissile();
00105                         m_gameScene.AddSceneObject(std::make_shared<Missile>(
00106                             closest_tower_ptr->getPosition(),
```

```
00107                             mouse_position));
00108                     }
00109
00110                     itr = m_inputBuffer.erase(itr);
00111                 }
00112                 else
00113                 {
00114                     itr++; // not handled event so ignore
00115                 }
00116             }
00117             // Wave Logic
00118             if(WaveMngr::getEnemiesRemaning() == 0)
00119             {
00120                 WaveMngr::passWave(); // Passing last wave.
00121                 WaveMngr::constructWave(m_gameScene);
00122             }
00123             else if(!m_player.isAlive())
00124             {
00125                 m_gameState = State::GameOver;
00126                 m_player.saveScore();
00127                 m_omenu->loadScores(ScoreFile::getScores());
00128                 auto objects_to_clear = m_gameScene.getAllOfType<Metiorite>();
00129                 for(auto& metiorite : objects_to_clear)
00130                 {
00131                     m_gameScene.ReleaseSceneObject(metiorite);
00132                 }
00133
00134                 WaveMngr::reset();
00135             }
00136
00137             break;
00138         case State::State::Pause:
00139             for(auto itr = m_inputBuffer.begin(); itr != m_inputBuffer.end();) // ACtions
00140             {
00141                 if(itr->first == Action::LMouse) // Action is shoot and any is mousePos
00142                 {
00143                     if(auto button = m_pmenu->getClickedButton(
00144                             std::any_cast<sf::Vector2i>(itr->second),
00145                             m_window))
00146                     {
00147                         m_gameState = button->doAction();
00148                     }
00149                     itr = m_inputBuffer.erase(itr);
00150                 }
00151                 else
00152                 {
00153                     itr++; // not handled event so ignore
00154                 }
00155             }
00156             m_inputBuffer.clear(); // void inputs after handling is done
00157             break;
00158         case State::Menu:
00159             for(auto itr = m_inputBuffer.begin(); itr != m_inputBuffer.end();) // ACtions
00160             {
00161                 if(itr->first == Action::LMouse) // Action is shoot and any is mousePos
00162                 {
00163                     if(auto button = m_mmenu->getClickedButton(
00164                             std::any_cast<sf::Vector2i>(itr->second),
00165                             m_window))
00166                     {
00167                         m_gameState = button->doAction();
00168                         if(m_gameState == State::InGame)
00169                         {
00170                             m_player.resetPlayer();
00171                         }
00172                     }
00173                     itr = m_inputBuffer.erase(itr);
00174                 }
00175                 else
00176                 {
00177                     itr++; // not handled event so ignore
00178                 }
00179             }
00180             m_inputBuffer.clear(); // void inputs after handling is done
00181             break;
00182         case State::GameOver:
00183             for(auto itr = m_inputBuffer.begin(); itr != m_inputBuffer.end();) // ACtions
00184             {
00185                 if(itr->first == Action::LMouse) // Action is shoot and any is mousePos
00186                 {
00187                     if(auto button = m_omenu->getClickedButton(
00188                             std::any_cast<sf::Vector2i>(itr->second),
00189                             m_window))
00190                     {
00191                         m_gameState = button->doAction();
00192                     }
00193                     itr = m_inputBuffer.erase(itr);
```

```
00194                 }
00195                 else
00196                 {
00197                     itr++; // not handled event so ignore
00198                 }
00199             }
00200             m_inputBuffer.clear(); // void inputs after handling is done
00201             break;
00202         case State::Exit: break;
00203     }
00204 }
00205
00206 void Game::UpdateScreen() // Needs a refactor too complex. maybe split it into parts or move code to
00207                           // objects
00208 {
00209     sf::Time const delta = m_clock.restart();
00210
00211     switch(m_gameState)
00212     {
00213         case State::State::InGame:
00214             std::cout << m_gameScene.getSize() << std::endl;
00215             for(const auto& obj : m_gameScene.getVec())
00216             {
00217                 if(!obj->update(delta))
00218                 {
00219                     if(std::shared_ptr<Missile> const p_missile =
00220                            std::dynamic_pointer_cast<Missile>(obj);
00221                        p_missile)
00222                     {
00223                         sf::Vector2f position = p_missile->getPosition();
00224                         auto metiorites = m_gameScene.getAllOfType<Metiorite>();
00225                         int metiorites_destroyed =
00226                             0; // this is used for if a player destroys multiple metiorites with one
00227                                // missile the score will multiply with the amout destroyed
00228                         std::for_each(
00229                             metiorites.begin(),
00230                             metiorites.end(),
00231                             [&position, &metiorites_destroyed, this](auto& elem) {
00232                                 if(elem->isInsideRadiusOfPos(position, EXPLOTION_RADIUS))
00233                                 {
00234                                     metiorites_destroyed++;
00235                                     m_player.addScore(SCORE_PER_METIORITE * metiorites_destroyed);
00236                                     elem->destroy();
00237                                 }
00238                             });
00239                         m_gameScene.ReleaseSceneObject(obj);
00240                         m_gameScene.AddSceneObject(
00241                             std::make_unique<Explosion>(position, EXPLOTION_RADIUS));
00242                     }
00243                     else if(std::shared_ptr<Metiorite> const p_metiorite =
00244                                 std::dynamic_pointer_cast<Metiorite>(obj);
00245                             p_metiorite)
00246                     {
00247                         sf::Vector2f const position = p_metiorite->getPosition();
00248                         m_gameScene.ReleaseSceneObject(obj);
00249
00250                         WaveMngr::enemyDestroyed();
00251
00252                         if(p_metiorite->hasReachedTarget())
00253                         {
00254                             m_player.removeLife();
00255                         }
00256
00257                         m_gameScene.AddSceneObject(
00258                             std::make_unique<Explosion>(position, EXPLOTION_RADIUS));
00259                     }
00260                     else if(std::shared_ptr<Explosion> const p_explotion =
00261                                 std::dynamic_pointer_cast<Explosion>(obj);
00262                             p_explotion)
00263                     {
00264                         m_gameScene.ReleaseSceneObject(obj);
00265                     }
00266                 }
00267             }
00268             break;
00269
00270         case State::GameOver: break;
00271         case State::Pause: break;
00272         case State::Menu: break;
00273         case State::Exit: break;
00274     }
00275 }
00276
00277 void Game::ComposeFrame()
00278 {
00279     // Drawing
00280     m_window.clear();
```

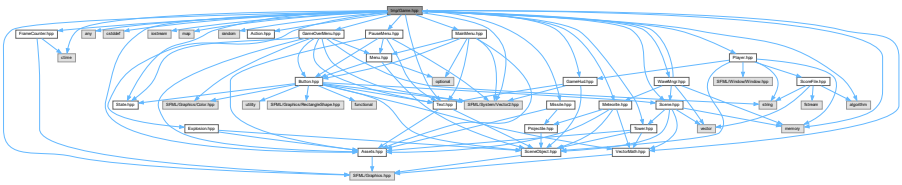
```
00281     switch(m_gameState)
00282     {
00283         case State::State::InGame:
00284             m_window.draw(m_backgroundSprite);
00285             for(auto& itr : m_gameScene)
00286             {
00287                 m_window.draw(*itr);
00288             }
00289             break;
00290         case State::State::Pause:
00291             for(auto& itr : m_pauseMenuScene)
00292             {
00293                 m_window.draw(*itr);
00294             }
00295             break;
00296         case State::Menu:
00297             for(auto& itr : m_mainMenuScene)
00298             {
00299                 m_window.draw(*itr);
00300             }
00301             break;
00302         case State::GameOver:
00303             for(auto& itr : m_gameOverScene)
00304             {
00305                 m_window.draw(*itr);
00306             }
00307             break;
00308 
00309         case State::Exit: break;
00310     }
00311 
00312     m_window.display();
00313 }
00314 
00315 void Game::GameLoop()
00316 {
00317     InitGame();
00318 
00319     while(m_gameState != State::Exit && m_window.isOpen())
00320     {
00321         HandleInput();
00322         UpdateGame();
00323         UpdateScreen();
00324         ComposeFrame();
00325     }
00326 
00327     m_window.close();
00328 }
```

7.21 tmp/Game.hpp File Reference

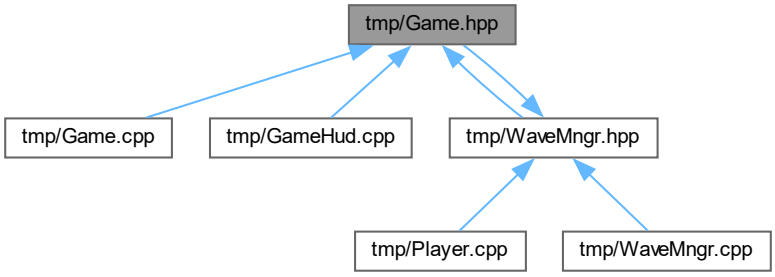
```
#include <SFML/Graphics.hpp>
#include <SFML/System/Vector2.hpp>
#include <algorithm>
#include <any>
#include <cstddef>
#include <ctime>
#include <iostream>
#include <map>
#include <memory>
#include <random>
#include "Action.hpp"
#include "Assets.hpp"
#include "Explosion.hpp"
#include "FrameCounter.hpp"
#include "GameHud.hpp"
#include "GameOverMenu.hpp"
#include "MainMenu.hpp"
#include "Meteorite.hpp"
#include "Missile.hpp"
#include "PauseMenu.hpp"
```

```
#include "Player.hpp"
#include "Scene.hpp"
#include "State.hpp"
#include "Text.hpp"
#include "Tower.hpp"
#include "VectorMath.hpp"
#include "WaveMngr.hpp"
```

Include dependency graph for Game.hpp:



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



Classes

- class Game

7.22 Game.hpp

Go to the documentation of this file.

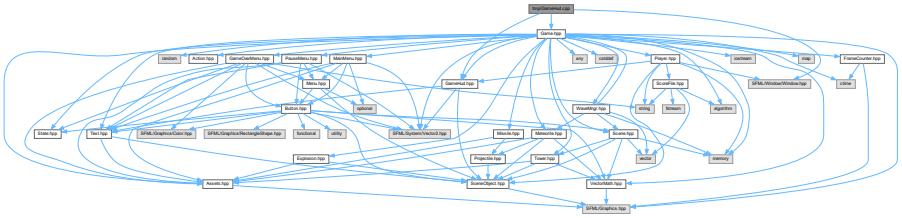
```
00001 #pragma once
00002 #include <SFML/Graphics.hpp>
00003 #include <SFML/System/Vector2.hpp>
00004 #include <algorithm>
00005 #include <any>
00006 #include <cstddef>
00007 #include <ctime>
00008 #include <iostream>
00009 #include <map>
00010 #include <memory>
00011 #include <random>
00012
00013 #include "Action.hpp"
00014 #include "Assets.hpp"
00015 #include "Explosion.hpp"
00016 #include "FrameCounter.hpp"
00017 #include "GameHud.hpp"
```

```
00018 #include "GameOverMenu.hpp"
00019 #include "MainMenu.hpp"
00020 #include "Meteorite.hpp"
00021 #include "Missile.hpp"
00022 #include "PauseMenu.hpp"
00023 #include "Player.hpp"
00024 #include "Scene.hpp"
00025 #include "State.hpp"
00026 #include "Text.hpp"
00027 #include "Tower.hpp"
00028 #include "VectorMath.hpp"
00029 #include "WaveMngr.hpp"
00030
00031 class Game
00032 {
00033     // Game Config constants
00034     static constexpr float EXPLOTION_RADIUS = 60.F;
00035     static constexpr int SCORE_PER_METIORITE = 1000;
00036
00037     // Variables
00038     State::State m_gameState{State::Menu};
00039     sf::RenderWindow m_window; // Main game window
00040     sf::Clock m_clock;
00041     sf::Sprite m_backgroundSprite;
00042     Scene m_gameScene;
00043     Scene m_pauseMenuScene;
00044     Scene m_mainMenuScene;
00045     Scene m_gameOverScene;
00046     Player m_player;
00047     std::shared_ptr<PauseMenu> m_pmenu;
00048     std::shared_ptr<MainMenu> m_mmenu;
00049     std::shared_ptr<GameOverMenu> m_omenu;
00050     std::multimap<Action::Action, std::any>
00051         m_inputBuffer; // A action and any associated data to that action. So if a shoot action is
00052                        // sent any will be a Vec2 with tthe mouse position. If a action cant be
00053                        // completed it will stay in the map to allow input buffering. Shoot actions
00054                        // may be discarded if all towers are on cooldown. otherwise tthe closest
00055                        // tower should shoot.
00056                        //
00057                        // I would like to make the input handling completly async so that even if
00058                        // the screen is not ready to draw a new frame the game still accepts input.
00059                        // Lets se if i have time to implement that :D PS. this is complete overkill
00060                        // but its fun so ill do it anyway
00061
00062     // Functions
00063     void InitGame(); // Ruins once and inits windows and such
00064     void HandleInput(); // Handles input from user
00065     void UpdateGame(); // Updates, runs game logic and checks game state
00066     void UpdateScreen(); // Updates screen objects for drawing
00067     void ComposeFrame(); // Draws to the window
00068
00069   protected:
00070   public:
00071     static constexpr float BOTTOM_PADDING = 70.F;
00072     inline static int width;
00073     inline static int height;
00074
00075     Game(int width, int height);
00076     ~Game();
00077     Game(const Game& other) = delete;
00078     Game(Game&& other) = delete;
00079     auto operator=(Game&& rhs) -> Game& = delete;
00080     auto operator=(const Game& other) -> Game& = delete;
00081     void GameLoop(); // Game Loop
00082 };
```

7.23 tmp/GameHud.cpp File Reference

```
#include "Game.hpp"
#include "GameHud.hpp"
#include <SFML/Window/Window.hpp>
```

Include dependency graph for GameHud.cpp:



7.24 GameHud.cpp

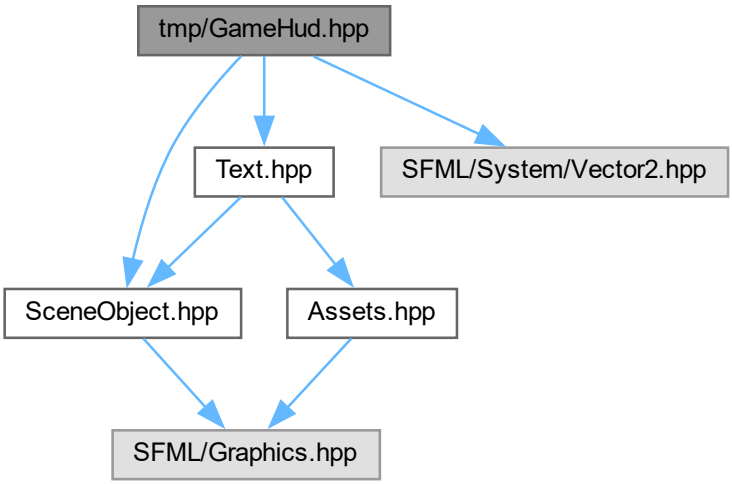
Go to the documentation of this file.

```
00001 #include "Game.hpp"
00002 #include "GameHud.hpp"
00003 #include <SFML/Window/Window.hpp>
00004
00005 GameHud::GameHud()
00006     : SceneObject(sf::Vector2f(0, 0))
00007     , m_score_text(sf::Vector2f(Game::width - 200, Game::height - (Game::BOTTOM_PADDING / 2)), "")
00008     , m_life_left_text(sf::Vector2f(10, Game::height - (Game::BOTTOM_PADDING / 2)), "")
00009     , m_current_wave_text(
00010           sf::Vector2f(Game::width / 2 - 45, Game::height - (Game::BOTTOM_PADDING / 2)),
00011           "")
00012 {
00013 }
00014
00015 void GameHud::setScoreText(const std::string& _text)
00016 {
00017     m_score_text.updateText(_text);
00018 }
00019
00020 void GameHud::setLifeText(const std::string& _text)
00021 {
00022     m_life_left_text.updateText(_text);
00023 }
00024
00025 void GameHud::setWaveText(const std::string& _text)
00026 {
00027     m_current_wave_text.updateText(_text);
00028 }
00029
00030 auto GameHud::update(const sf::Time& /*delta*/) -> bool
00031 {
00032     return true;
00033 }
00034
00035 void GameHud::draw(sf::RenderTarget& target, sf::RenderStates states) const
00036 {
00037     target.draw(m_score_text, states);
00038     target.draw(m_life_left_text, states);
00039     target.draw(m_current_wave_text, states);
00040 }
```

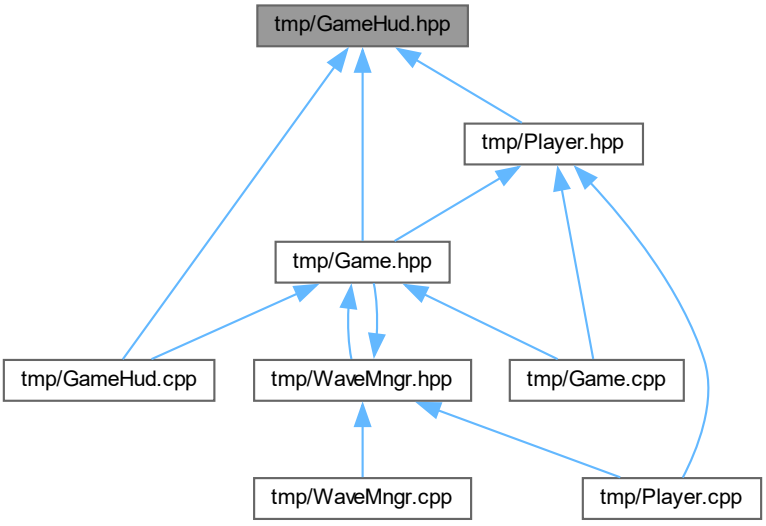
7.25 tmp/GameHud.hpp File Reference

```
#include "SceneObject.hpp"
#include "Text.hpp"
#include <SFML/System/Vector2.hpp>
```

Include dependency graph for GameHud.hpp:



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



Classes

- class GameHud

7.26 GameHud.hpp

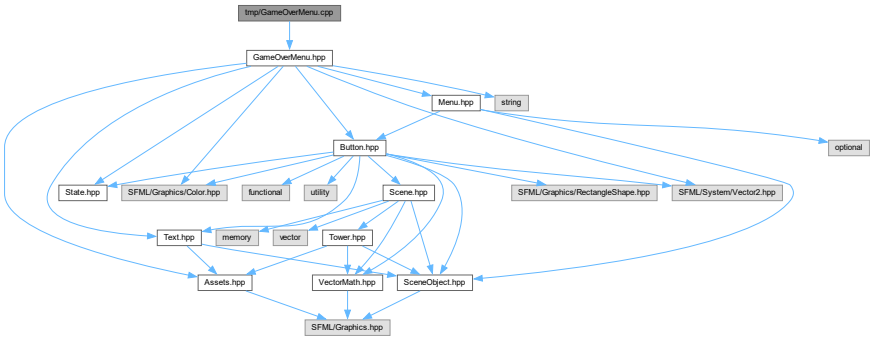
Go to the documentation of this file.

```
00001 #pragma once
00002 #include "SceneObject.hpp"
00003 #include "Text.hpp"
00004 #include <SFML/System/Vector2.hpp>
00005
00006 class GameHud: public SceneObject
00007 {
00008     Text m_score_text;
00009     Text m_life_left_text;
00010     Text m_current_wave_text;
00011
00012   public:
00013     GameHud();
00014
00015     void setScoreText(const std::string& _text);
00016     void setLifeText(const std::string& _text);
00017     void setWaveText(const std::string& _text);
00018
00019     auto update(const sf::Time& delta) -> bool override;
00020     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00021         const override;
00022 };
```

7.27 tmp/GameOverMenu.cpp File Reference

`#include "GameOverMenu.hpp"`

Include dependency graph for GameOverMenu.cpp:



7.28 GameOverMenu.cpp

Go to the documentation of this file.

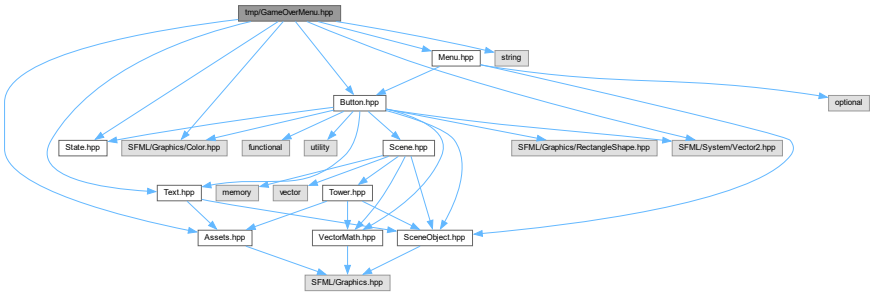
```
00001 #include "GameOverMenu.hpp"
00002
00003 GameOverMenu::GameOverMenu()
00004     : m_pausedText(sf::Vector2f(100, 100), "Game Over", 48)
00005     , m_quitButton(sf::Vector2f(100, 700), "Quit", []() -> State::State {
00006         return State::State::Menu;
00007     })
00008 {
00009 }
00010 void GameOverMenu::loadScores(std::vector<std::string> scores)
00011 {
00012     m_scores.clear();
00013     int current_score = 0;
00014     for(auto score : scores)
00015     {
00016         m_scores.push_back(
00017             Text(sf::Vector2f(100, TOP_PADDING_SCORES + (50 * current_score)), score, 32));
```

```
00018         current_score++;
00019     }
00020 };
00021
00022 auto GameOverMenu::update(const sf::Time& delta) -> bool
00023 {
00024     return true;
00025 }
00026
00027 void GameOverMenu::draw(sf::RenderTarget& target, sf::RenderStates states) const
00028 {
00029     target.draw(m_pausedText, states);
00030     target.draw(m_quitButton, states);
00031     for(auto& score : m_scores)
00032     {
00033         target.draw(score, states);
00034     }
00035 }
00036
00037 auto GameOverMenu::getClickedButton(
00038     const sf::Vector2i& mouse_position,
00039     const sf::RenderWindow& window) -> std::optional<Button>
00040 {
00041     if(m_quitButton.isClicked(mouse_position, window))
00042     {
00043         return m_quitButton;
00044     }
00045     return std::nullopt;
00046 }
```

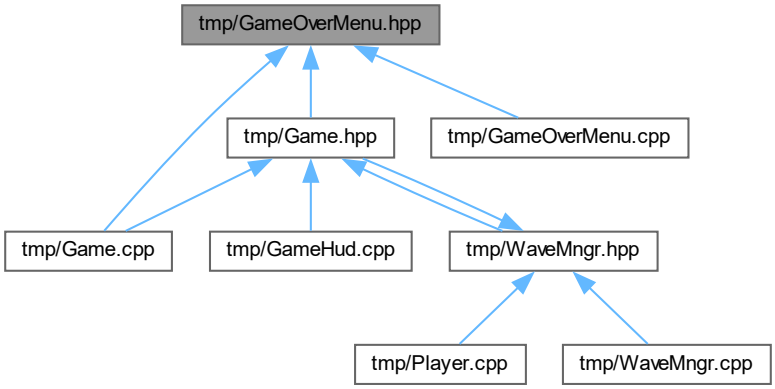
7.29 tmp/GameOverMenu.hpp File Reference

```
#include "Assets.hpp"
#include "Button.hpp"
#include "Menu.hpp"
#include "State.hpp"
#include "Text.hpp"
#include <SFML/Graphics/Color.hpp>
#include <SFML/System/Vector2.hpp>
#include <string>
```

Include dependency graph for GameOverMenu.hpp:



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



Classes

- class GameOverMenu

7.30 GameOverMenu.hpp

Go to the documentation of this file.

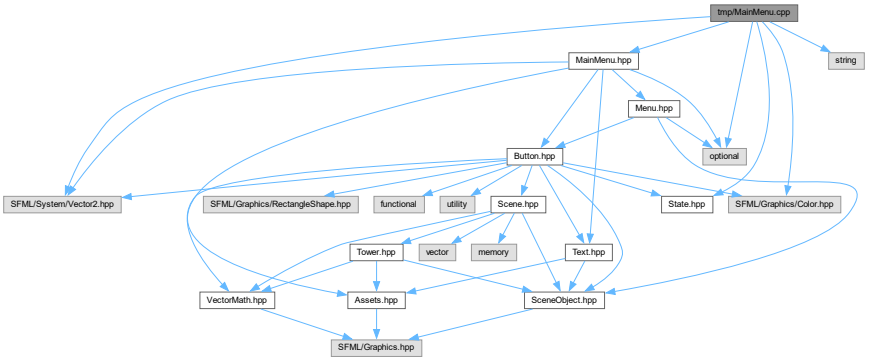
```
00001 #pragma once
00002 #include "Assets.hpp"
00003 #include "Button.hpp"
00004 #include "Menu.hpp"
00005 #include "State.hpp"
00006 #include "Text.hpp"
00007 #include <SFML/Graphics/Color.hpp>
00008 #include <SFML/System/Vector2.hpp>
00009 #include <string>
00010
00011 class GameOverMenu: public Menu
00012 {
00013     static constexpr int TOP_PADDING_SCORES = 200;
00014     Text m_pausedText;
00015     std::vector<Text> m_scores;
00016     Button m_quitButton;
00017
00018   public:
00019     GameOverMenu();
00020
00021     void loadScores(std::vector<std::string> scores);
00022
00023     auto update(const sf::Time& delta) -> bool override;
00024     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00025         const override;
00026     auto getClickedButton(const sf::Vector2i& mouse_position, const sf::RenderWindow& window)
00027         -> std::optional<Button> override;
00028 };
```

7.31 tmp/MainMenu.cpp File Reference

```
#include "MainMenu.hpp"
#include "State.hpp"
```

```
#include <SFML/Graphics/Color.hpp>
#include <SFML/System/Vector2.hpp>
#include <optional>
#include <string>
```

Include dependency graph for MainMenu.cpp:



7.32 MainMenu.cpp

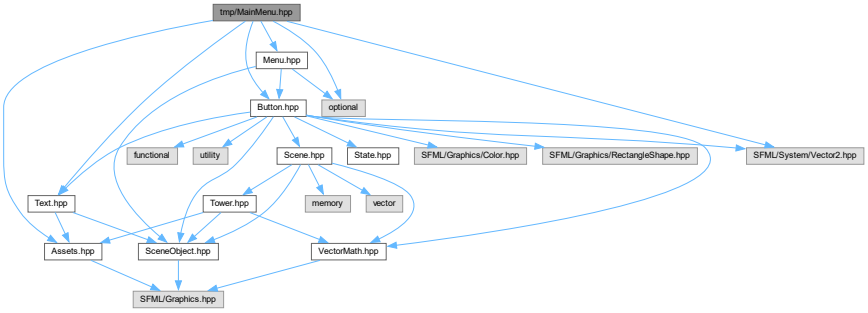
Go to the documentation of this file.

```
00001 #include "MainMenu.hpp"
00002 #include "State.hpp"
00003 #include <SFML/Graphics/Color.hpp>
00004 #include <SFML/System/Vector2.hpp>
00005 #include <optional>
00006 #include <string>
00007
00008 MainMenu::MainMenu()
00009     : m_game_name_text(sf::Vector2f(100, 100), "Main Menu", 48)
00010     , m_startButton(
00011           sf::Vector2f(100, 200),
00012           "Start game",
00013           []() -> State::State { return State::State::InGame; })
00014     , m_quitButton(sf::Vector2f(100, 300), "Quit", []() -> State::State {
00015         return State::State::Exit;
00016     }){};
00017
00018 auto MainMenu::update(const sf::Time& delta) -> bool
00019 {
00020     return true;
00021 }
00022
00023 void MainMenu::draw(sf::RenderTarget& target, sf::RenderStates states) const
00024 {
00025     target.draw(m_game_name_text, states);
00026     target.draw(m_startButton, states);
00027     target.draw(m_quitButton, states);
00028 }
00029 auto MainMenu::getClickedButton(const sf::Vector2i& mouse_position, const sf::RenderWindow& window)
00030     -> std::optional<Button>
00031 {
00032     if(m_startButton.isClicked(mouse_position, window))
00033     {
00034         return m_startButton;
00035     }
00036     if(m_quitButton.isClicked(mouse_position, window))
00037     {
00038         return m_quitButton;
00039     }
00040     return std::nullopt;
00041 }
```

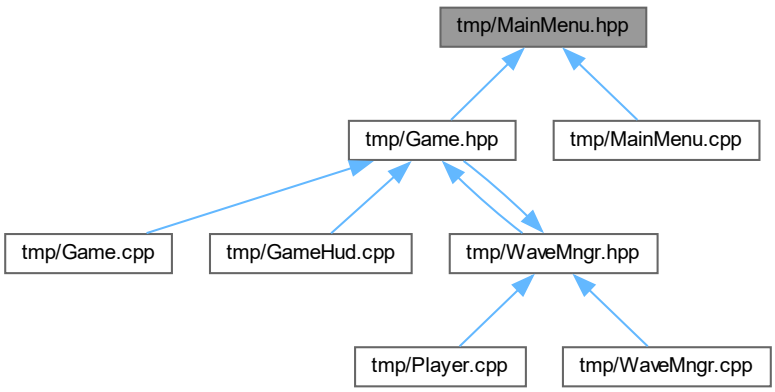
7.33 tmp/MainMenu.hpp File Reference

```
#include "Assets.hpp"
#include "Button.hpp"
#include "Menu.hpp"
#include "Text.hpp"
#include <SFML/System/Vector2.hpp>
#include <optional>
```

Include dependency graph for MainMenu.hpp:



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



Classes

- class MainMenu

7.34 MainMenu.hpp

Go to the documentation of this file.

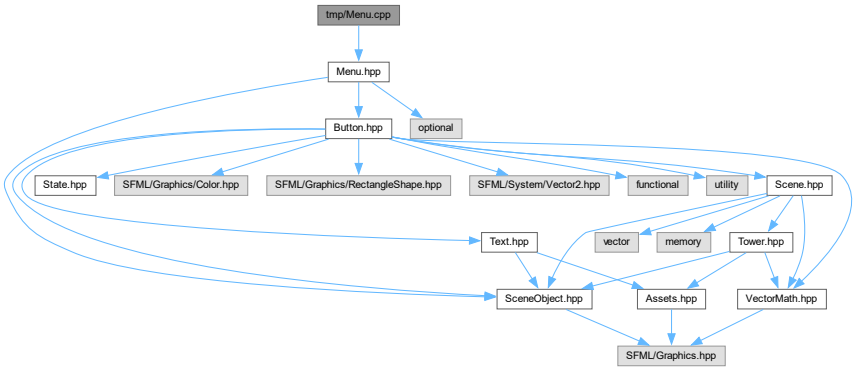
```
00001 #pragma once
```

```
00002 #include "Assets.hpp"
00003 #include "Button.hpp"
00004 #include "Menu.hpp"
00005 #include "Text.hpp"
00006 #include <SFML/System/Vector2.hpp>
00007 #include <optional>
00008
00009 class MainMenu: public Menu
00010 {
00011     Text m_game_name_text;
00012     Button m_startButton;
00013     Button m_quitButton;
00014
00015   public:
00016     MainMenu();
00017
00018     auto update(const sf::Time& delta) -> bool override;
00019     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00020         const override;
00021
00022     auto getClickedButton(const sf::Vector2i& mouse_position, const sf::RenderWindow& window)
00023         -> std::optional<Button> override;
00024 };
```

7.35 tmp/Menu.cpp File Reference

```
#include "Menu.hpp"
```

Include dependency graph for Menu.cpp:



7.36 Menu.cpp

Go to the documentation of this file.

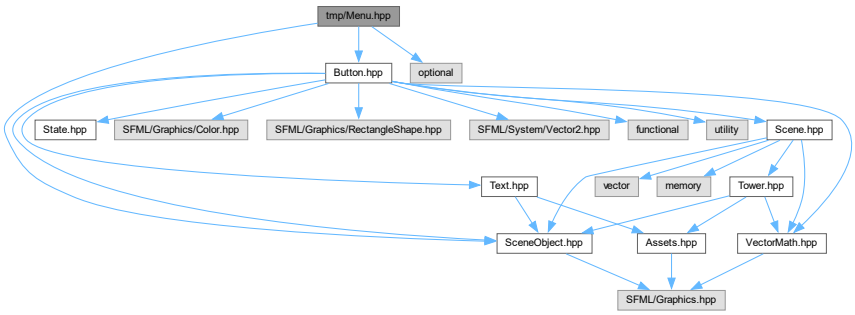
```
00001 #include "Menu.hpp"
00002
00003 Menu::Menu(): SceneObject(m_position){};
```

7.37 tmp/Menu.hpp File Reference

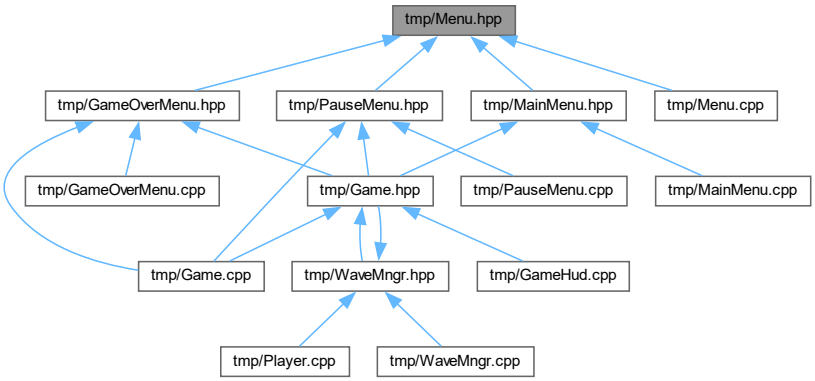
```
#include "Button.hpp"
#include "SceneObject.hpp"
```

`#include <optional>`

Include dependency graph for Menu.hpp:



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



Classes

- class Menu

7.38 Menu.hpp

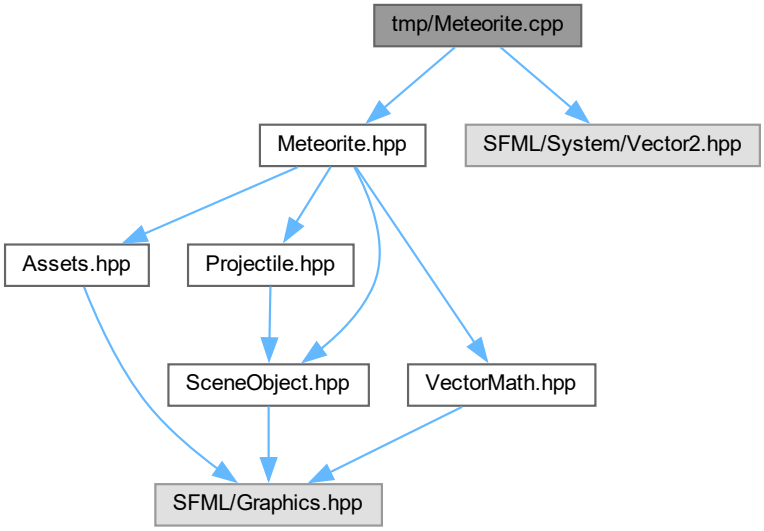
Go to the documentation of this file.

```
00001 #pragma once
00002 #include "Button.hpp"
00003 #include "SceneObject.hpp"
00004 #include <optional>
00005
00006 class Menu: public SceneObject
00007 {
00008     const sf::Vector2f m_position = sf::Vector2f(0, 0); // A menu always starts in 0,0
00009
00010   public:
00011     Menu();
00012
00013     auto update(const sf::Time& delta) -> bool override = 0;
00014     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00015         const override = 0;
00016     virtual auto getClickedButton(
00017         const sf::Vector2i& mouse_position,
00018         const sf::RenderWindow& window) -> std::optional<Button> = 0;
00019 };
```

7.39 tmp/Meteorite.cpp File Reference

```
#include "Meteorite.hpp"
#include <SFML/System/Vector2.hpp>
```

Include dependency graph for Meteorite.cpp:



7.40 Meteorite.cpp

Go to the documentation of this file.

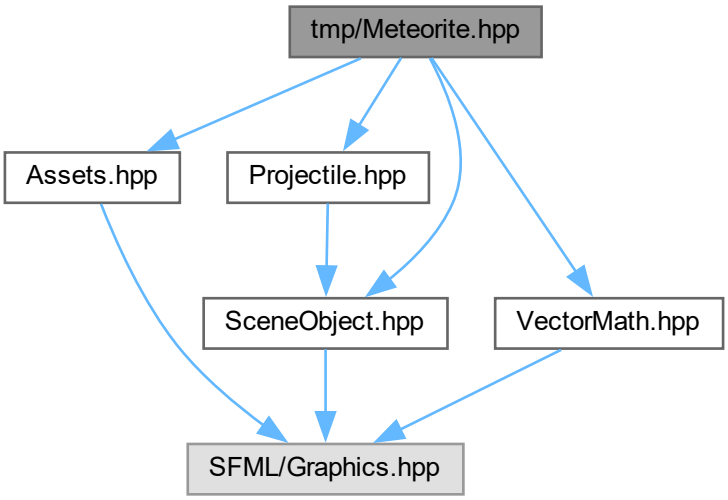
```
00001 #include "Meteorite.hpp"
00002 #include <SFML/System/Vector2.hpp>
00003
00004 Metiorite::Metiorite(const sf::Vector2f& begin, const sf::Vector2f& target)
00005     : Projectile(begin)
00006     , m_begin(begin)
00007     , m_target(target)
00008     , m_direction(normalize(m_target - m_begin))
00009 {
00010     sf::Vector2u sheet_size = Assets::metiorite_sheet.getSize();
00011     m_rectMetioriteSheet =
00012         sf::IntRect(0, 0, sheet_size.x / SPRITES_PER_SHEET, sheet_size.y); // 3 sprites in one sheet
00013     m_metioriteSprite = sf::Sprite(Assets::metiorite_sheet, m_rectMetioriteSheet);
00014
00015     sf::Vector2f adjusted_position;
00016     adjusted_position.x = begin.x - ((sheet_size.x / SPRITES_PER_SHEET) / 2.0F); // texture magic :P
00017     adjusted_position.y = begin.y - (sheet_size.y / 2.0F);
00018
00019     m_metioriteSprite.setPosition(adjusted_position);
00020     m_metioriteSprite.setScale(SPRITE_SCALE_FACTOR, SPRITE_SCALE_FACTOR);
00021 }
00022
00023 void Metiorite::destroy()
00024 {
00025     m_destroyed = true;
00026 }
00027
00028 auto Metiorite::hasReachedTarget() const -> bool
00029 {
00030     return m_reached_target;
00031 };
00032
00033 auto Metiorite::getTarget() const -> sf::Vector2f
```

```
00034 {
00035     return m_target;
00036 }
00037
00038 auto Metiorite::isInsideRadiusOfPos(const sf::Vector2f& pos, const float& radius) -> bool
00039 {
00040     return distanceBetween(pos, this->getPosition()) < radius;
00041 }
00042
00043 auto Metiorite::update(const sf::Time& delta) -> bool
00044 {
00045     if(distanceBetween(m_begin, this->getPosition()) >= distanceBetween(m_begin, m_target))
00046     {
00047         m_reached_target = true;
00048         return false;
00049     }
00050     if(m_destroyed)
00051     {
00052         return false;
00053     }
00054
00055     m_spriteTimer += delta.asSeconds();
00056
00057     if(m_spriteIndex >= 2)
00058     {
00059         m_spriteIndex = 0;
00060         m_rectMetioriteSheet.left = 0;
00061     } // Animation is done
00062
00063     if(m_spriteTimer > ANIMATION_ROTATION_DELAY)
00064     {
00065         m_rectMetioriteSheet.left +=
00066             static_cast<int>(Assets::metiorite_sheet.getSize().x / SPRITES_PER_SHEET);
00067         m_spriteTimer = 0.0F;
00068         m_spriteIndex++;
00069     }
00070     m_metioriteSprite.setTextureRect(m_rectMetioriteSheet);
00071     this->move(m_direction * MOVE_SPEED * delta.asSeconds());
00072
00073     m_metioriteSprite.setPosition(this->getPosition());
00074
00075     return true;
00076 }
00077
00078 void Metiorite::draw(sf::RenderTarget& target, sf::RenderStates states) const
00079 {
00080     target.draw(m_metioriteSprite, states);
00081 }
```

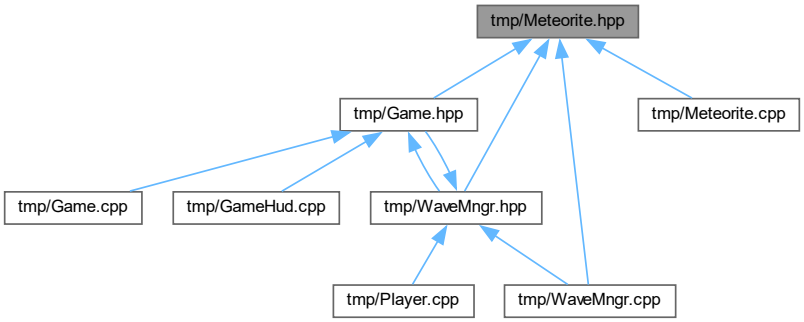
7.41 tmp/Meteorite.hpp File Reference

```
#include "Assets.hpp"
#include "Projectile.hpp"
#include "SceneObject.hpp"
#include "VectorMath.hpp"
```

Include dependency graph for Meteorite.hpp:



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



Classes

- class Metiorite

7.42 Meteorite.hpp

Go to the documentation of this file.

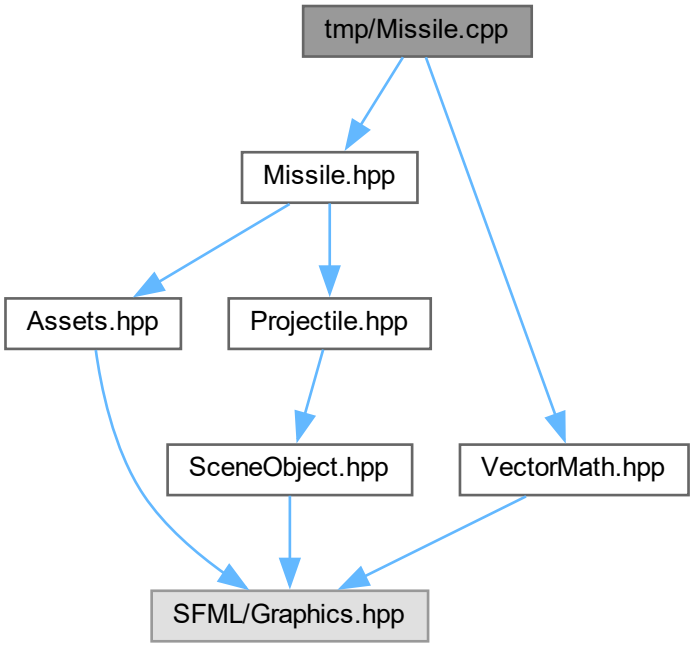
```
00001 #pragma once
00002
00003 #include "Assets.hpp"
00004 #include "Projectile.hpp"
```

```
00005 #include "SceneObject.hpp"
00006 #include "VectorMath.hpp"
00007
00008 class Metiorite: public Projectile
00009 {
00010     // Magic constants
00011     static constexpr float SPRITE_SCALE_FACTOR = 0.5F;
00012     static constexpr float ANIMATION_ROTATION_DELAY = 0.17F;
00013     static constexpr float MOVE_SPEED = 80.0F;
00014     static constexpr int SPRITES_PER_SHEET = 3;
00015
00016     // Variables
00017     sf::Vector2f m_begin;
00018     sf::Vector2f m_target;
00019     sf::Vector2f m_direction;
00020     sf::Sprite m_metioriteSprite;
00021     sf::IntRect m_rectMetioriteSheet;
00022     float m_spriteTimer = 0.0F;
00023     int m_spriteIndex = 0;
00024     bool m_destroyed = false;
00025     bool m_reached_target = false;
00026
00027   public:
00028     Metiorite(const sf::Vector2f& begin, const sf::Vector2f& target);
00029
00030     auto getTarget() const -> sf::Vector2f override;
00031     auto isInsideRadiusOfPos(const sf::Vector2f& pos, const float& radius) -> bool;
00032     void destroy();
00033     auto hasReachedTarget() const -> bool;
00034
00035     auto update(const sf::Time& delta) -> bool override;
00036     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00037         const override;
00038 };
```

7.43 tmp/Missile.cpp File Reference

```
#include "Missile.hpp"
#include "VectorMath.hpp"
```

Include dependency graph for Missile.cpp:



7.44 Missile.cpp

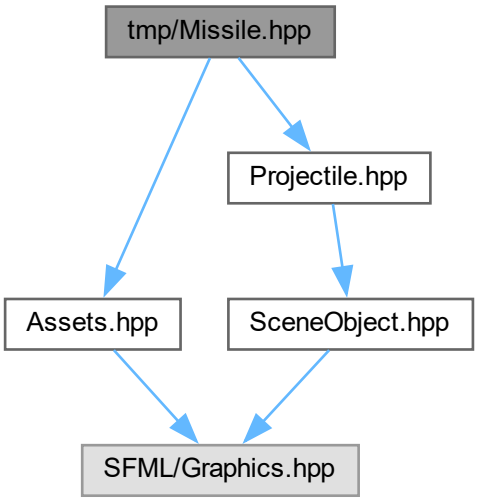
Go to the documentation of this file.

```
00001 #include "Missile.hpp"
00002 #include "VectorMath.hpp"
00003
00004 Missile::Missile(const sf::Vector2f& begin, const sf::Vector2f& target)
00005     : Projectile(begin)
00006     , m_begin(begin)
00007     , m_target(target)
00008     , m_direction(normalize(m_target - m_begin))
00009 {
00010     m_rocketShape = sf::Sprite(Assets::missile);
00011 }
00012
00013 auto Missile::getTarget() const -> sf::Vector2f
00014 {
00015     return m_target;
00016 }
00017
00018 auto Missile::update(const sf::Time& delta) -> bool
00019 {
00020     if(distanceBetween(m_begin, this->getPosition()) >= distanceBetween(m_begin, m_target))
00021     {
00022         return false;
00023     }
00024
00025     this->move(m_direction * 300.0F * delta.asSeconds());
00026
00027     m_rocketShape.setPosition(this->getPosition());
00028
00029     return true;
00030 }
00031
00032 void Missile::draw(sf::RenderTarget& target, sf::RenderStates states) const
00033 {
00034     target.draw(m_rocketShape, states);
00035 }
```

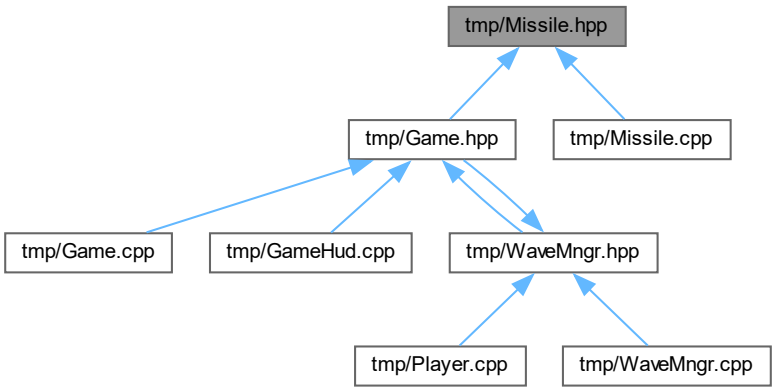
7.45 tmp/Missile.hpp File Reference

```
#include "Assets.hpp"
#include "Projectile.hpp"
```

Include dependency graph for Missile.hpp:



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



Classes

- class Missile

7.46 Missile.hpp

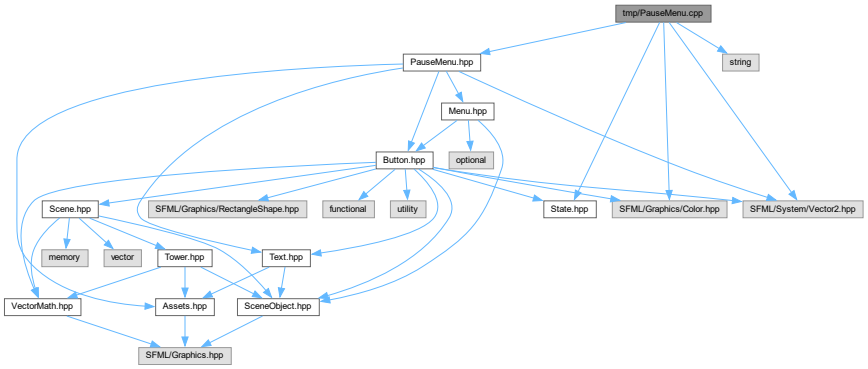
Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Assets.hpp"
00004 #include "Projectile.hpp"
00005
00006 class Missile: public Projectile
00007 {
00008     sf::Vector2f m_begin;
00009     sf::Vector2f m_target;
00010     sf::Vector2f m_direction;
00011     sf::Sprite m_rocketShape;
00012
00013   public:
00014     Missile(const sf::Vector2f& begin, const sf::Vector2f& target);
00015
00016     auto getTarget() const -> sf::Vector2f override;
00017
00018     auto update(const sf::Time& delta) -> bool override;
00019     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00020         const override;
00021 };
```

7.47 tmp/PauseMenu.cpp File Reference

```
#include "PauseMenu.hpp"
#include "State.hpp"
#include <SFML/Graphics/Color.hpp>
#include <SFML/System/Vector2.hpp>
#include <string>
```

Include dependency graph for PauseMenu.cpp:



7.48 PauseMenu.cpp

Go to the documentation of this file.

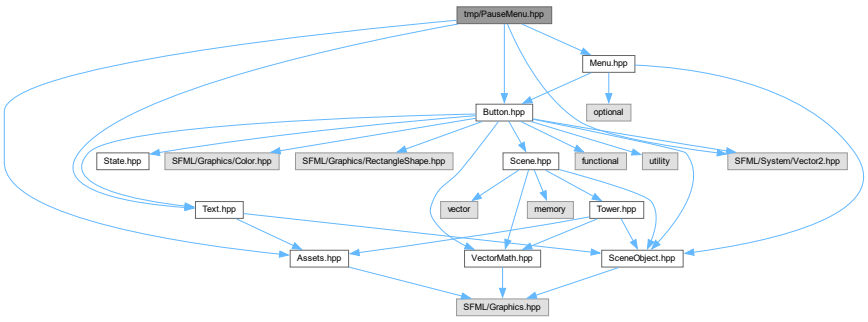
```
00001 #include "PauseMenu.hpp"
00002 #include "State.hpp"
00003 #include <SFML/Graphics/Color.hpp>
00004 #include <SFML/System/Vector2.hpp>
00005 #include <string>
00006
00007 PauseMenu::PauseMenu()
00008     : m_pausedText(sf::Vector2f(100, 100), "Paused", 48)
00009     , m_quitButton(sf::Vector2f(100, 200), "Quit", []() -> State::State {
00010         return State::State::Exit;
```

```
00011     }){};
00012
00013 auto PauseMenu::update(const sf::Time& delta) -> bool
00014 {
00015     return true;
00016 }
00017
00018 void PauseMenu::draw(sf::RenderTarget& target, sf::RenderStates states) const
00019 {
00020     target.draw(m_pausedText, states);
00021     target.draw(m_quitButton, states);
00022 }
00023
00024 auto PauseMenu::getClickedButton(const sf::Vector2i& mouse_position, const sf::RenderWindow& window)
00025     -> std::optional<Button>
00026 {
00027     if(m_quitButton.isClicked(mouse_position, window))
00028     {
00029         return m_quitButton;
00030     }
00031     return std::nullopt;
00032 }
```

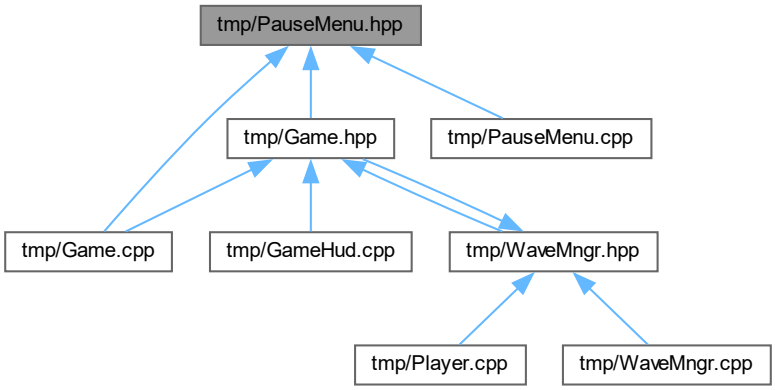
7.49 tmp/PauseMenu.hpp File Reference

```
#include "Assets.hpp"
#include "Button.hpp"
#include "Menu.hpp"
#include "Text.hpp"
#include <SFML/System/Vector2.hpp>
```

Include dependency graph for PauseMenu.hpp:



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



Classes

- class PauseMenu

7.50 PauseMenu.hpp

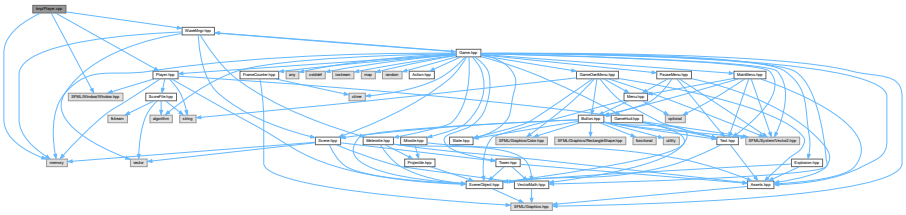
Go to the documentation of this file.

```
00001 #pragma once
00002 #include "Assets.hpp"
00003 #include "Button.hpp"
00004 #include "Menu.hpp"
00005 #include "Text.hpp"
00006 #include <SFML/System/Vector2.hpp>
00007
00008 class PauseMenu: public Menu
00009 {
00010     Text m_pausedText;
00011     Button m_quitButton;
00012
00013   public:
00014     PauseMenu();
00015
00016     auto update(const sf::Time& delta) -> bool override;
00017     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00018         const override;
00019     auto getClickedButton(const sf::Vector2i& mouse_position, const sf::RenderWindow& window)
00020         -> std::optional<Button> override;
00021 };
```

7.51 tmp/Player.cpp File Reference

```
#include "Player.hpp"
#include "WaveMngr.hpp"
#include <SFML/Window/Window.hpp>
```

#include <memory>
Include dependency graph for Player.cpp:



7.52 Player.cpp

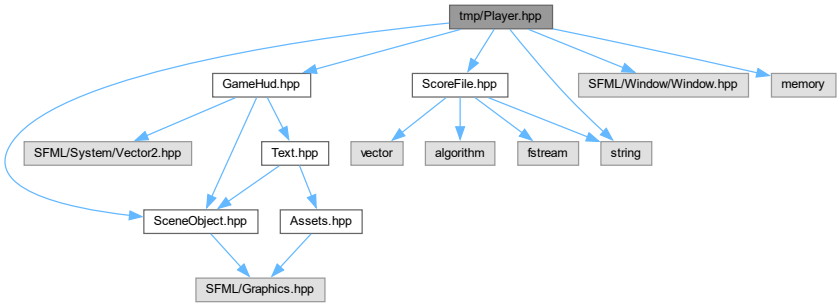
Go to the documentation of this file.

```
00001 #include "Player.hpp"
00002 #include "WaveMngr.hpp"
00003 #include <SFML/Window/Window.hpp>
00004 #include <memory>
00005
00006 Player::Player(): m_score(0), m_lifes_left(MAX_LIFES){};
00007
00008 void Player::saveScore() {
00009     ScoreFile::addScore(m_score);
00010 }
00011
00012 void Player::initHud()
00013 {
00014     m_hud = std::make_shared<GameHud>(); // Needs to be initialized after window and Game static
00015                                          // variables has been set.
00016     updateHud(); // Sets text to the right values
00017 };
00018
00019 void Player::addScore(int score)
00020 {
00021     m_score += score;
00022     updateHud();
00023 }
00024
00025 void Player::removeLife()
00026 {
00027     if(m_lifes_left <= 0)
00028     {
00029         m_dead = true;
00030     }
00031     else
00032     {
00033         m_lifes_left--;
00034     }
00035     updateHud();
00036 }
00037
00038 void Player::resetPlayer()
00039 {
00040     m_lifes_left = MAX_LIFES;
00041     m_score = 0;
00042     updateHud();
00043 }
00044
00045 void Player::updateHud()
00046 {
00047     m_hud->setLifeText("Lifes: " + std::to_string(m_lifes_left));
00048     m_hud->setScoreText("Score: " + std::to_string(m_score));
00049     m_hud->setWaveText("Wave: " + std::to_string(WaveMngr::getWave()));
00050 }
00051
00052 auto Player::isAlive() const -> bool
00053 {
00054     return (m_lifes_left > 0);
00055 }
00056
00057 auto Player::getHud() -> std::shared_ptr<GameHud>
00058 {
00059     return m_hud;
00060 }
```

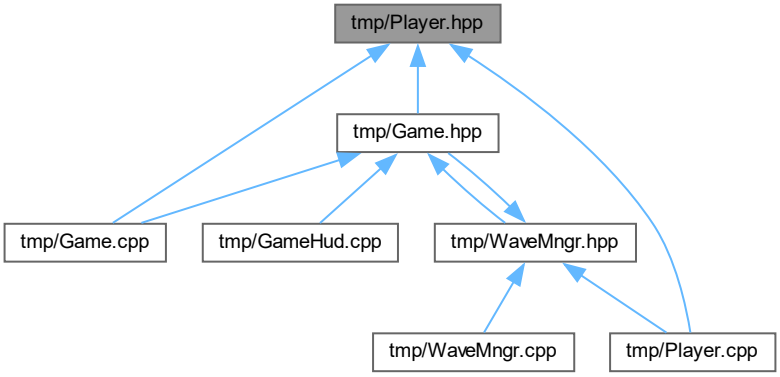
7.53 tmp/Player.hpp File Reference

```
#include "GameHud.hpp"
#include "SceneObject.hpp"
#include "ScoreFile.hpp"
#include <SFML/Window/Window.hpp>
#include <memory>
#include <string>
```

Include dependency graph for Player.hpp:



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



Classes

- class Player

7.54 Player.hpp

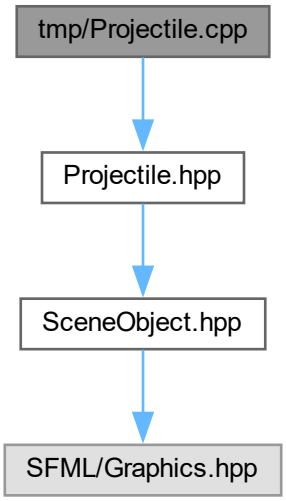
Go to the documentation of this file.

```
00001 #pragma once
```

```
00002
00003 #include "GameHud.hpp"
00004 #include "SceneObject.hpp"
00005 #include "ScoreFile.hpp"
00006 #include <SFML/Window/Window.hpp>
00007 #include <memory>
00008 #include <string>
00009
00010 class Player
00011 {
00012     static constexpr int MAX_LIFES = 3;
00013
00014     std::shared_ptr<GameHud>
00015         m_hud; // will push this onto scene later. since its a shared pointer it *should* be fine.
00016
00017     int m_score;
00018     int m_lifes_left;
00019     bool m_dead = false;
00020
00021   public:
00022     Player();
00023
00024     void saveScore();
00025     void initHud();
00026     void addScore(int score);
00027     void removeLife();
00028     void resetPlayer();
00029     void updateHud();
00030     [[nodiscard]] auto isAlive() const -> bool;
00031     auto getHud() -> std::shared_ptr<GameHud>;
00032 };
```

7.55 tmp/Projectile.cpp File Reference

`#include "Projectile.hpp"`
Include dependency graph for Projectile.cpp:



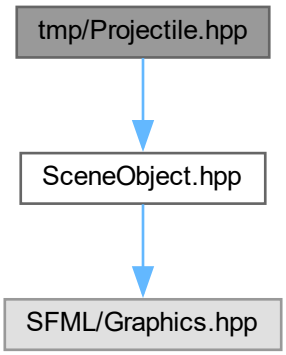
7.56 Projectile.cpp

Go to the documentation of this file.

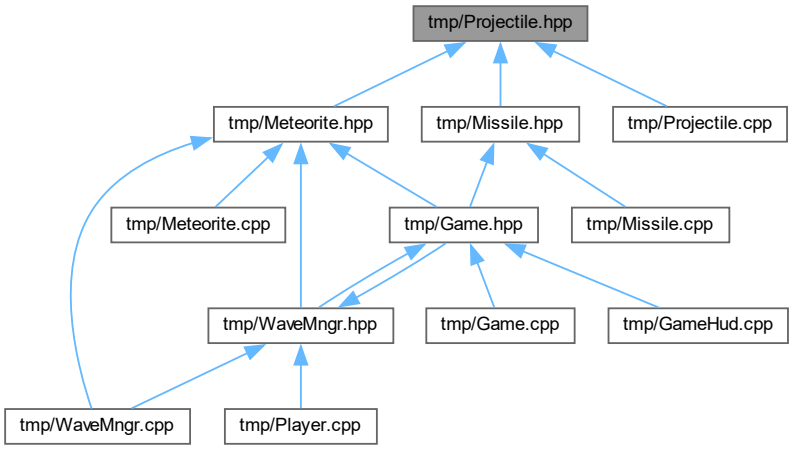
```
00001 #include "Projectile.hpp"
00002
00003 Projectile::Projectile(const sf::Vector2f& position): SceneObject(position) {}
```

7.57 tmp/Projectile.hpp File Reference

`#include "SceneObject.hpp"`
Include dependency graph for Projectile.hpp:



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



Classes

- class Projectile

7.58 Projectile.hpp

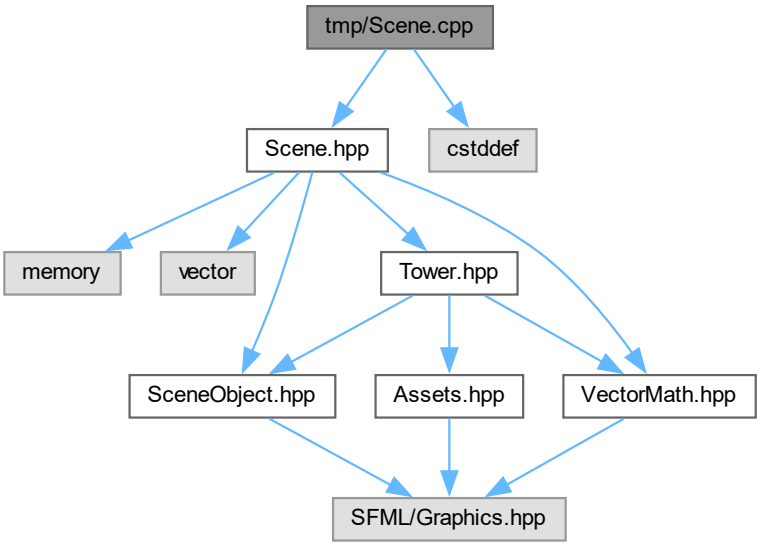
Go to the documentation of this file.

```
00001 #pragma once
00002 #include "SceneObject.hpp"
00003
00004 class Projectile: public SceneObject
00005 {
00006   public:
00007     Projectile(const sf::Vector2f& position);
00008
00009     virtual auto getTarget() const -> sf::Vector2f = 0;
00010
00011     auto update(const sf::Time& delta) -> bool override = 0;
00012     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00013         const override = 0;
00014 };
```

7.59 tmp/Scene.cpp File Reference

```
#include "Scene.hpp"
#include <cstddef>
```

Include dependency graph for Scene.cpp:



7.60 Scene.cpp

Go to the documentation of this file.

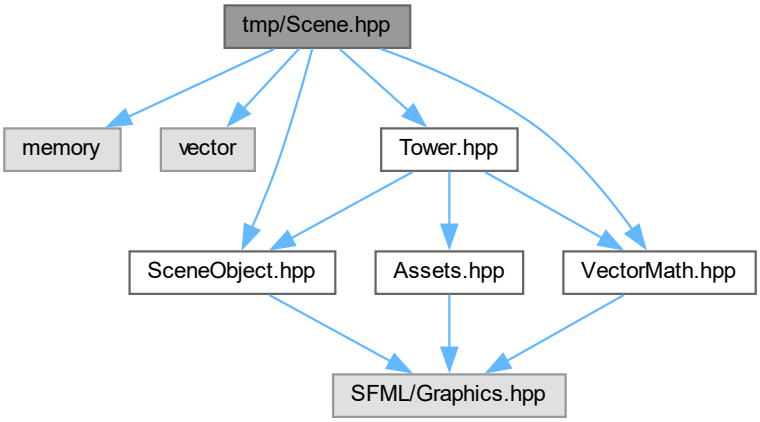
```
00001 #include "Scene.hpp"
00002 #include <cstddef>
00003
00004 Scene::Scene() = default; // Notting to init :V
00005
00006 Scene::~Scene() {} // Notting to remove from heap :feelsgoodman:
00007
00008 auto Scene::getSize() const -> size_t
```

```
00009 {
00010     return m_drawables.size();
00011 }
00012
00013 auto Scene::getVec() -> std::vector<std::shared_ptr<SceneObject»
00014 {
00015     return m_drawables;
00016 }
00017
00018 auto Scene::AddSceneObject(std::shared_ptr<SceneObject> Obj)
00019     -> bool // gib me many copies here :D Refrence counting <3
00020 {
00021     if(Obj != nullptr
00022        && !std::any_of(begin(), end(), [&Obj](const std::shared_ptr<SceneObject>& elem) {
00023               return (elem == Obj);
00024           }))
00025     {
00026         this->m_drawables.push_back(Obj);
00027     }
00028     else
00029     {
00030         return false;
00031     }
00032     return true;
00033 }
00034
00035 auto Scene::ReleaseSceneObject(std::shared_ptr<SceneObject> obj) -> bool
00036 {
00037     bool deleted = false;
00038     m_drawables.erase(
00039         std::remove_if(
00040             begin(),
00041             end(),
00042             [&obj, &deleted](const std::shared_ptr<SceneObject>& elem) {
00043                 deleted = true;
00044                 return (elem == obj);
00045             }),
00046         end());
00047     return deleted;
00048 }
00049
00050 auto Scene::GetClosestFirableTower(sf::Vector2f pos) -> std::shared_ptr<Tower>
00051 {
00052     std::shared_ptr<Tower> c_obj = nullptr; // Clostest
00053     for(auto& elem : m_drawables)
00054     {
00055         if(std::shared_ptr<Tower> ch_obj = std::dynamic_pointer_cast<Tower>(elem);
00056            ch_obj) // not nullptr. Casted correctly
00057         {
00058             if(c_obj != nullptr)
00059             {
00060                 if(distanceBetween(ch_obj->getPosition(), pos)
00061                        < distanceBetween(c_obj->getPosition(), pos)
00062                    && ch_obj->canFireMissile())
00063                 {
00064                     c_obj = ch_obj;
00065                 }
00066             }
00067             else if(ch_obj->canFireMissile())
00068             {
00069                 c_obj = ch_obj;
00070             }
00071         }
00072     };
00073     return c_obj;
00074 }
```

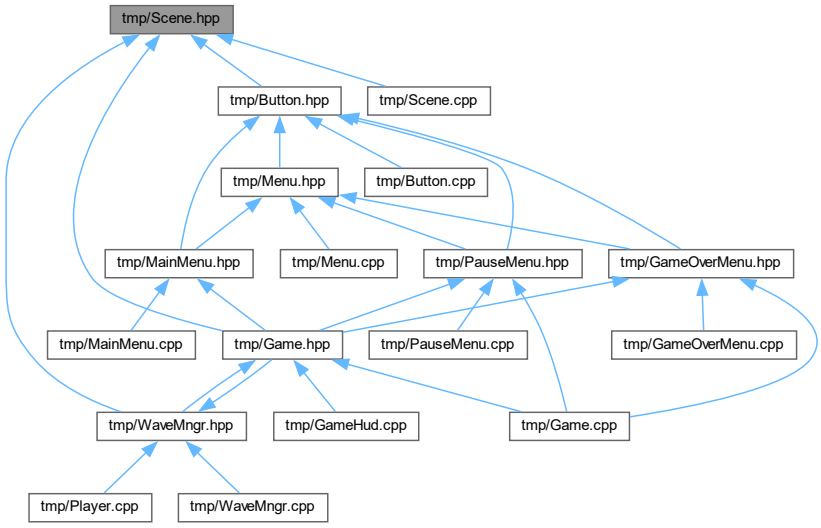
7.61 tmp/Scene.hpp File Reference

```
#include <memory>
#include <vector>
#include "SceneObject.hpp"
#include "Tower.hpp"
#include "VectorMath.hpp"
```

Include dependency graph for Scene.hpp:



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



Classes

- class Scene

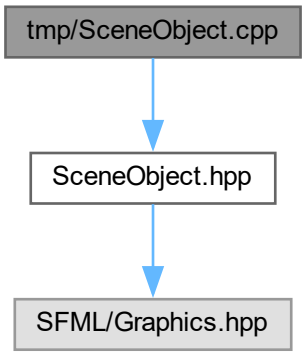
7.62 Scene.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <memory>
00003 #include <vector>
00004
00005 #include "SceneObject.hpp"
00006 #include "Tower.hpp"
00007 #include "VectorMath.hpp"
00008
00009 // Definition of why make class when teacher make good class 4 u :D
00010
00011 class Scene
00012 {
00013     std::vector<std::shared_ptr<SceneObject» m_drawables;
00014
00015   public:
00016     Scene();
00017     ~Scene();
00018     Scene(const Scene& other) = delete; // No copys thanks >:(
00019     auto operator=(const Scene& other) -> Scene& = delete;
00020     Scene(Scene&& other) = delete; // No move thanks >:(
00021     auto operator=(Scene&& other) -> Scene& = delete;
00022
00023     [[nodiscard]] auto getSize() const -> size_t;
00024
00025     auto getVec() -> std::vector<std::shared_ptr<SceneObject»;
00026
00027     auto AddSceneObject(std::shared_ptr<SceneObject> Obj) -> bool;
00028     auto ReleaseSceneObject(std::shared_ptr<SceneObject> Obj) -> bool;
00029     auto GetClosestFirableTower(sf::Vector2f pos) -> std::shared_ptr<Tower>;
00030
00031     inline auto begin()
00032         -> std::vector<std::shared_ptr<SceneObject»::iterator // Thanks for the Module 3 code :D
00033     {
00034         return m_drawables.begin();
00035     }
00036     inline auto end() -> std::vector<std::shared_ptr<SceneObject»::iterator
00037     {
00038         return m_drawables.end();
00039     }
00040
00041     template<typename T>
00042     auto getClosestSceneObjectOfType(sf::Vector2f pos) -> std::shared_ptr<T>
00043     {
00044         std::shared_ptr<T> c_obj = nullptr; // Clostest
00045         for(auto& obj : m_drawables)
00046         {
00047             if(std::shared_ptr<T> ch_obj = std::dynamic_pointer_cast<T>(obj);
00048                ch_obj) // not nullptr. Casted correctly
00049             {
00050                 if(c_obj == nullptr
00051                    || distanceBetween(ch_obj->getPosition(), pos)
00052                           < distanceBetween(c_obj->getPosition(), pos))
00053                 {
00054                     c_obj = ch_obj;
00055                 }
00056             }
00057         };
00058         return c_obj;
00059     }
00060
00061     template<typename T>
00062     auto getAllOfType() -> std::vector<std::shared_ptr<T»
00063     {
00064         std::vector<std::shared_ptr<T» v_type;
00065         // Clostest
00066         for(auto& obj : m_drawables)
00067         {
00068             if(std::shared_ptr<T> const ch_obj = std::dynamic_pointer_cast<T>(obj);
00069                ch_obj) // not nullptr. Casted correctly
00070             {
00071                 v_type.push_back(ch_obj);
00072             }
00073         };
00074         return v_type;
00075     }
00076 };
```

7.63 tmp/SceneObject.cpp File Reference

`#include "SceneObject.hpp"`
Include dependency graph for SceneObject.cpp:



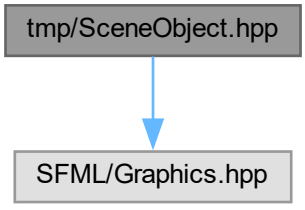
7.64 SceneObject.cpp

Go to the documentation of this file.

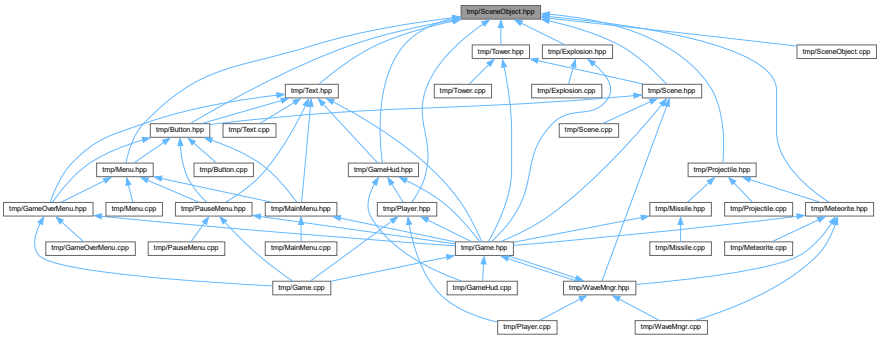
```
00001 #include "SceneObject.hpp"
00002
00003 SceneObject::SceneObject(const sf::Vector2f& position)
00004 {
00005     this->setPosition(position);
00006 }
00007
00008 SceneObject::~SceneObject() = default;
```

7.65 tmp/SceneObject.hpp File Reference

`#include <SFML/Graphics.hpp>`
Include dependency graph for SceneObject.hpp:



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



Classes

- class SceneObject

7.66 SceneObject.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <SFML/Graphics.hpp>
00004
00005 class SceneObject
00006     : public sf::Drawable
00007     , public sf::Transformable
00008 {
00009   public:
00010     SceneObject(const sf::Vector2f& position);
00011     ~SceneObject() override;
00012     SceneObject(SceneObject& object) = default;
00013     auto operator=(SceneObject const& object) -> SceneObject& = default;
00014     SceneObject(SceneObject&& object) = default;
00015     auto operator=(SceneObject&& object) -> SceneObject& = default;
00016
00017     virtual auto update(const sf::Time& delta) -> bool = 0;
00018     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00019         const override = 0;
00020 };
```

7.67 tmp/ScoreBoard.cpp File Reference

7.68 ScoreBoard.cpp

Go to the documentation of this file.

7.69 tmp/ScoreBoard.hpp File Reference

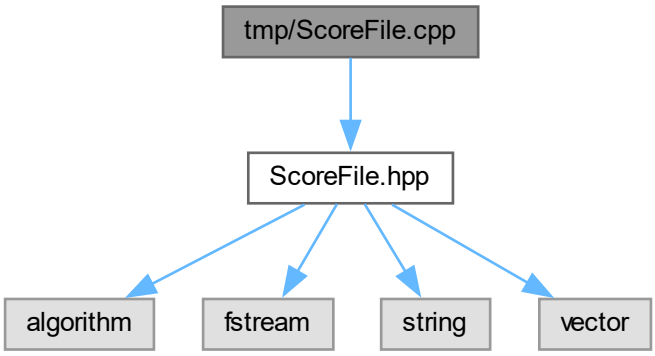
7.70 ScoreBoard.hpp

Go to the documentation of this file.

```
00001 #pragma once
```

7.71 tmp/ScoreFile.cpp File Reference

`#include "ScoreFile.hpp"`
Include dependency graph for ScoreFile.cpp:



7.72 ScoreFile.cpp

Go to the documentation of this file.

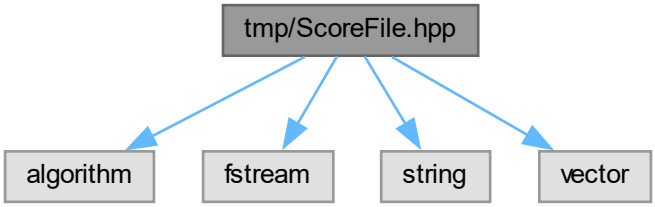
```
00001 #include "ScoreFile.hpp"
00002
00003 auto ScoreFile::getScores() -> std::vector<std::string>
00004 {
00005     std::vector<std::string> ret;
00006     std::fstream file;
00007     file.open(SCORE_FILE_PATH, std::ios::in);
00008     std::string temp_string;
00009     if(!file)
00010     {
00011         return ret; // File might not be created yet.
00012     }
00013     while(file » temp_string && ret.size() <= 4)
00014     {
00015         // Read the number.
00016         ret.push_back(temp_string);
00017     }
00018     return ret;
00019 }
00020
00021 void ScoreFile::addScore(int score)
00022 {
00023     std::vector<int> lines;
00024     std::fstream file;
00025     file.open(SCORE_FILE_PATH, std::ios::in | std::ios::app);
00026     int temp_int;
00027
00028     lines.push_back(score);
00029
00030     while(file » temp_int)
00031     {
00032         lines.push_back(temp_int);
00033     }
00034     file.close();
00035     if(lines.size() > 0)
00036     {
00037         std::sort(lines.begin(), lines.end());
00038         std::reverse(lines.begin(), lines.end());
00039     }
00040     file.open(SCORE_FILE_PATH, std::ios::out | std::ios::trunc);
00041     for(auto& value : lines)
```

```
00042     {
00043         file « value « std::endl;
00044     }
00045 }
```

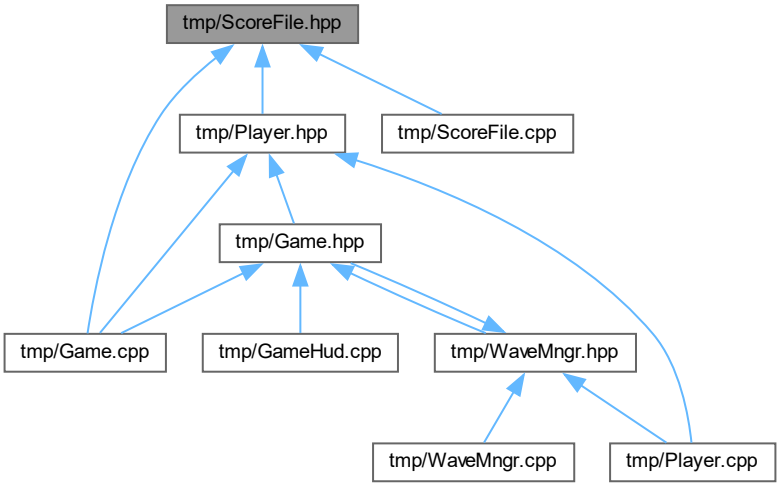
7.73 tmp/ScoreFile.hpp File Reference

```
#include <algorithm>
#include <fstream>
#include <string>
#include <vector>
```

Include dependency graph for ScoreFile.hpp:



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



Classes

- class ScoreFile

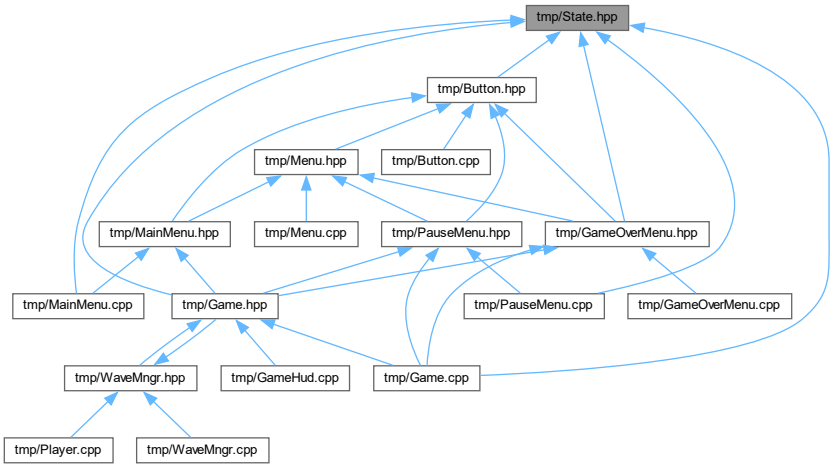
7.74 ScoreFile.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <algorithm>
00003 #include <fstream>
00004 #include <string>
00005 #include <vector>
00006
00007 class ScoreFile
00008 {
00009     static constexpr char SCORE_FILE_PATH[] = "scores.txt";
00010
00011   public:
00012     static auto getScores() -> std::vector<std::string>; // Reads the file if one exists
00013     static void addScore(int score); // appends a score to the score file
00014 };
```

7.75 tmp/State.hpp File Reference

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



Namespaces

- namespace State

Enumerations

- enum State::State {
  State::InGame = 1 , State::Menu = 2 , State::GameOver = 3 , State::Pause = 4 ,
  State::Exit = 5 }

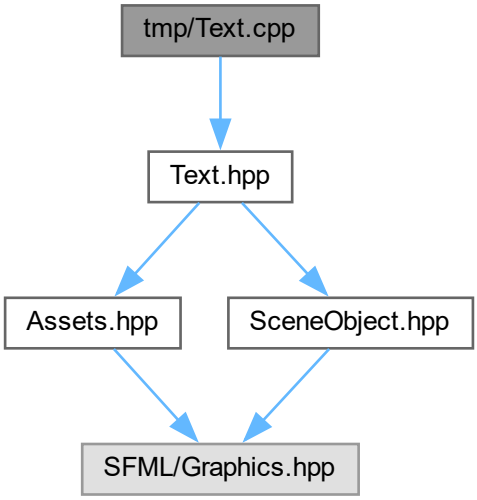
7.76 State.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 namespace State
00004 {
00005     enum State
00006     {
00007         InGame = 1,
00008         Menu = 2,
00009         GameOver = 3,
00010         Pause = 4,
00011         Exit = 5
00012     };
00013 }
```

7.77 tmp/Text.cpp File Reference

`#include "Text.hpp"`
Include dependency graph for Text.cpp:



7.78 Text.cpp

Go to the documentation of this file.

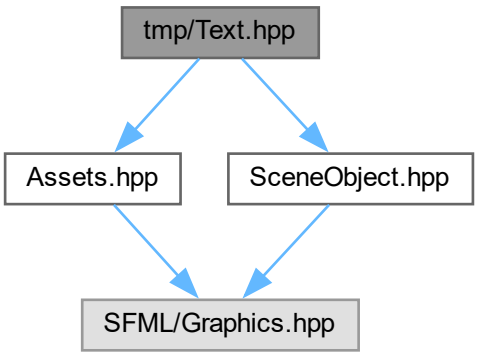
```
00001 #include "Text.hpp"
00002
00003 Text::Text(const sf::Vector2f& position, const std::string& _text, const int fontSize)
00004     : SceneObject(position)
00005 {
00006     // select the font
00007     m_text.setFont(Assets::text_font); // font is a sf::Font
00008
00009     // set the string to display
00010     m_text.setString(_text);
00011
```

```
00012     // set the character size
00013     m_text.setCharacterSize(fontSize); // in pixels, not points!
00014
00015     // set the color
00016     m_text.setFillColor(sf::Color::White);
00017
00018     m_text.setPosition(position.x, position.y);
00019
00020     // set the text style
00021     m_text.setStyle(sf::Text::Bold);
00022 }
00023
00024 void Text::updateText(const std::string& _text)
00025 {
00026     m_text.setString(_text);
00027 }
00028
00029 auto Text::getRawTextObject() -> sf::Text&
00030 {
00031     return m_text;
00032 }
00033
00034 auto Text::update(const sf::Time& /*delta*/) -> bool
00035 {
00036     return true; // no need to update this object
00037 }
00038
00039 void Text::draw(sf::RenderTarget& target, sf::RenderStates states) const
00040 {
00041     target.draw(m_text, states);
00042 }
```

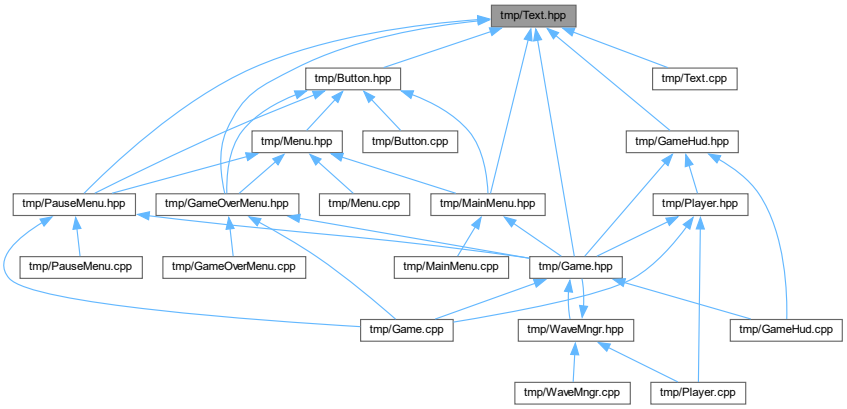
7.79 tmp/Text.hpp File Reference

```
#include "Assets.hpp"
#include "SceneObject.hpp"
```

Include dependency graph for Text.hpp:



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



Classes

- class Text

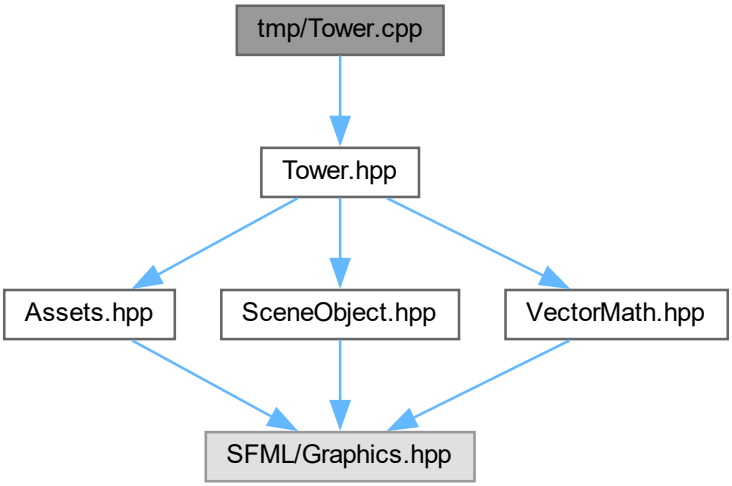
7.80 Text.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "Assets.hpp"
00003 #include "SceneObject.hpp"
00004
00005 class Text: public SceneObject
00006 {
00007     static constexpr int DEFAULT_FONTSIZE = 24;
00008
00009     sf::Text m_text;
00010
00011   public:
00012     Text(const sf::Vector2f& position, const std::string& _text, int fontSize = DEFAULT_FONTSIZE);
00013
00014     void updateText(const std::string& _text);
00015
00016     auto getRawTextObject() -> sf::Text&;
00017
00018     auto update(const sf::Time& delta) -> bool override;
00019     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00020         const override;
00021 };
```

7.81 tmp/Tower.cpp File Reference

`#include "Tower.hpp"`
Include dependency graph for Tower.cpp:



7.82 Tower.cpp

Go to the documentation of this file.

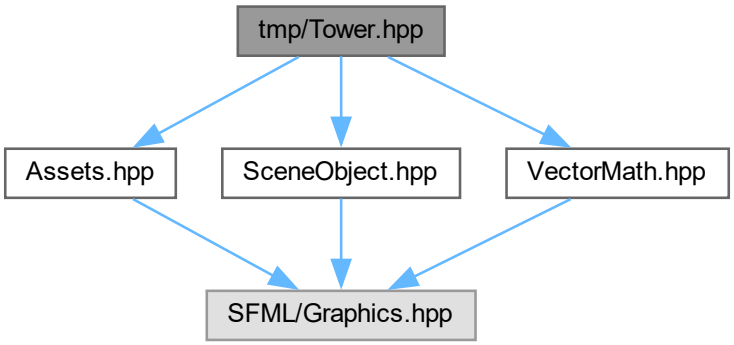
```
00001 #include "Tower.hpp"
00002
00003 Tower::Tower(sf::Vector2f pos): SceneObject(pos)
00004 {
00005     auto texture_size = Assets::tower.getSize();
00006     m_sprTower.setTexture(Assets::tower);
00007
00008     m_sprTower.setColor(sf::Color(255, 255, 255, 200));
00009     pos.x = pos.x - (texture_size.x / 2.0F); // texture magic :P
00010     pos.y = pos.y - (texture_size.y / 2.0F);
00011
00012     m_sprTower.setPosition(pos);
00013 }
00014
00015 void Tower::fireMissile()
00016 {
00017     m_lastFire =
00018         m_towerClock
00019             .getElapsedTime(); // There is a bug here since the timer continues running while
00020                                // game is paused. This can be exploited by spam pausing the
00021                                // game while firing bypassing the timer in game time.
00022     m_onCooldown = true;
00023 }
00024
00025 auto Tower::canFireMissile() const -> bool
00026 {
00027     return !m_onCooldown;
00028 }
00029
00030 auto Tower::update(const sf::Time& delta) -> bool
00031 {
00032     // WIP: Make the tower face the mouse pointer
00033
00034     if(m_onCooldown
00035        && m_lastFire.asSeconds() + COOLDOWN_PERIOD < m_towerClock.getElapsedTime().asSeconds())
```

```
00036     {
00037         m_onCooldown = false;
00038     }
00039
00040     return true;
00041 }
00042
00043 void Tower::draw(sf::RenderTarget& target, sf::RenderStates states) const
00044 {
00045     target.draw(m_sprTower, states);
00046 }
```

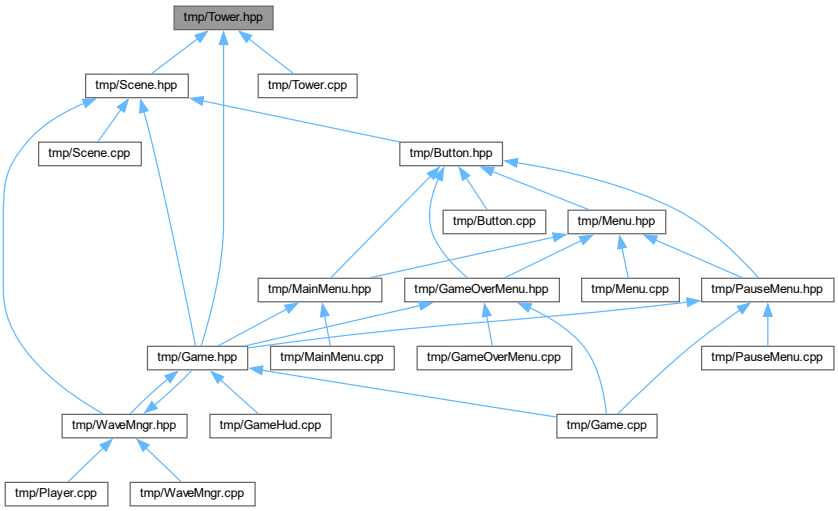
7.83 tmp/Tower.hpp File Reference

```
#include "Assets.hpp"
#include "SceneObject.hpp"
#include "VectorMath.hpp"
```

Include dependency graph for Tower.hpp:



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



Classes

- class Tower

7.84 Tower.hpp

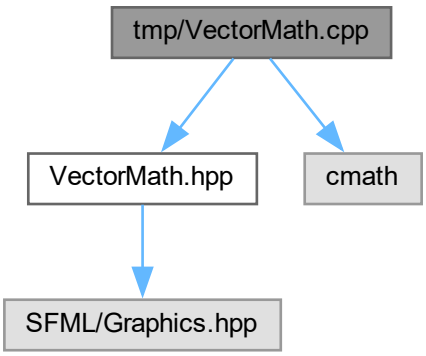
Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Assets.hpp"
00004 #include "SceneObject.hpp"
00005 #include "VectorMath.hpp"
00006
00007 class Tower: public SceneObject
00008 {
00009     static constexpr float COOLDOWN_PERIOD = 1.0F; // Tower cooldown period
00010     sf::Sprite m_sprTower;
00011     sf::Vector2f m_direction;
00012     sf::Clock m_towerClock;
00013     sf::Time m_lastFire;
00014     bool m_onCooldown = false;
00015
00016   public:
00017     Tower(sf::Vector2f pos);
00018     void fireMissile();
00019     auto canFireMissile() const -> bool;
00020     auto update(const sf::Time& delta) -> bool override;
00021     void draw(sf::RenderTarget& target, sf::RenderStates states = sf::RenderStates::Default)
00022         const override;
00023 };
```

7.85 tmp/VectorMath.cpp File Reference

```
#include "VectorMath.hpp"
#include <cmath>
```

Include dependency graph for VectorMath.cpp:



Macros

- #define _USE_MATH_DEFINES

Functions

- auto dot (const sf::Vector2f &vectorA, const sf::Vector2f &vectorB) -> float
- auto length (const sf::Vector2f &vector) -> float
- auto distanceBetween (const sf::Vector2f &pointA, const sf::Vector2f &pointB) -> float
- auto normalize (const sf::Vector2f &vector) -> sf::Vector2f
- auto angleBetween (const sf::Vector2f &directionA, const sf::Vector2f directionB) -> float
- auto vec2iToVec2f (sf::Vector2i rhs) -> sf::Vector2f
- auto vec2fToVec2i (sf::Vector2f rhs) -> sf::Vector2i

7.85.1 Macro Definition Documentation

7.85.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

Definition at line 1 of file VectorMath.cpp.

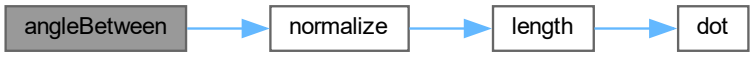
7.85.2 Function Documentation

7.85.2.1 angleBetween()

```
auto angleBetween (
            const sf::Vector2f & directionA,
            const sf::Vector2f directionB ) -> float
```

Definition at line 25 of file VectorMath.cpp.

Here is the call graph for this function:

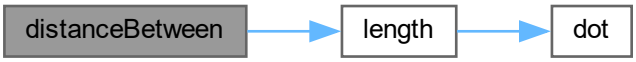


7.85.2.2 distanceBetween()

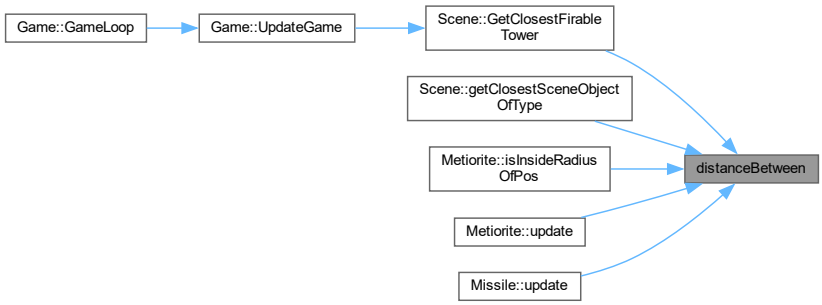
```
auto distanceBetween (
            const sf::Vector2f & pointA,
            const sf::Vector2f & pointB ) -> float
```

Definition at line 15 of file VectorMath.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

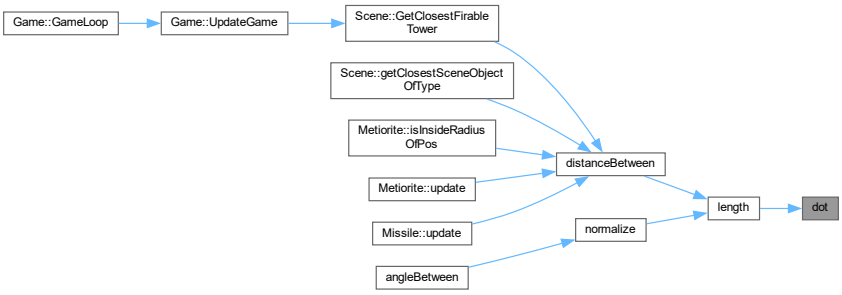


7.85.2.3 dot()

```
auto dot (
            const sf::Vector2f & vectorA,
            const sf::Vector2f & vectorB ) -> float
```

Definition at line 5 of file VectorMath.cpp.

Here is the caller graph for this function:



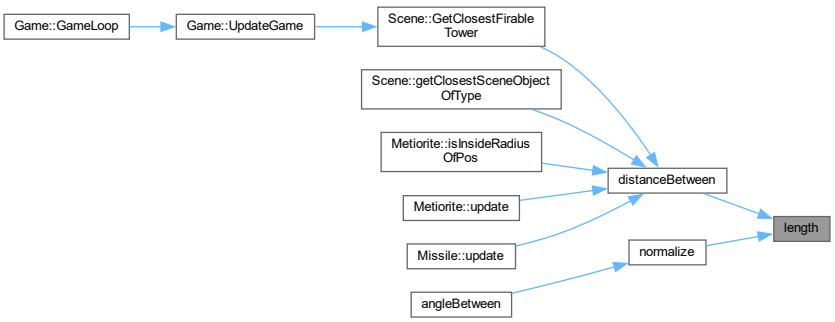
7.85.2.4 length()

```
auto length (
            const sf::Vector2f & vector ) -> float
```

Definition at line 10 of file VectorMath.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



7.85.2.5 normalize()

```
auto normalize (
            const sf::Vector2f & vector ) -> sf::Vector2f
```

Definition at line 20 of file VectorMath.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.85.2.6 vec2fToVec2i()

```
auto vec2fToVec2i (
            sf::Vector2f rhs ) -> sf::Vector2i
```

Definition at line 40 of file VectorMath.cpp.

7.85.2.7 vec2iToVec2f()

```
auto vec2iToVec2f (
            sf::Vector2i rhs ) -> sf::Vector2f
```

Definition at line 35 of file VectorMath.cpp.

Here is the caller graph for this function:

7.86 VectorMath.cpp

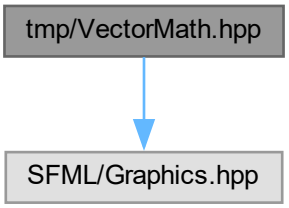
Go to the documentation of this file.

```
00001 #define _USE_MATH_DEFINES
00002 #include "VectorMath.hpp"
00003 #include <cmath>
00004
00005 auto dot(const sf::Vector2f& vectorA, const sf::Vector2f& vectorB) -> float
00006 {
00007     return vectorA.x * vectorB.x + vectorA.y * vectorB.y;
00008 }
00009
00010 auto length(const sf::Vector2f& vector) -> float
00011 {
00012     return std::sqrt(dot(vector, vector));
00013 }
00014
00015 auto distanceBetween(const sf::Vector2f& pointA, const sf::Vector2f& pointB) -> float
00016 {
00017     return length(pointB - pointA);
00018 }
00019
00020 auto normalize(const sf::Vector2f& vector) -> sf::Vector2f
00021 {
00022     return vector / length(vector);
00023 }
00024
00025 auto angleBetween(const sf::Vector2f& directionA, const sf::Vector2f directionB) -> float
00026 {
00027     sf::Vector2f normalized_a = normalize(directionA);
00028     sf::Vector2f normalized_b = normalize(directionB);
00029     float angle_radians =
00030         std::atan2(normalized_b.y, normalized_b.x) - std::atan2(normalized_a.y, normalized_a.x);
00031     float angle_degrees = angle_radians * 180.F / M_PI;
00032     return angle_degrees;
00033 }
00034
00035 auto vec2iToVec2f(sf::Vector2i rhs) -> sf::Vector2f
00036 {
00037     return {static_cast<float>(rhs.x), static_cast<float>(rhs.y)};
00038 }
00039
00040 auto vec2fToVec2i(sf::Vector2f rhs) -> sf::Vector2i
00041 {
00042     return {static_cast<int>(rhs.x), static_cast<int>(rhs.y)};
00043 }
```

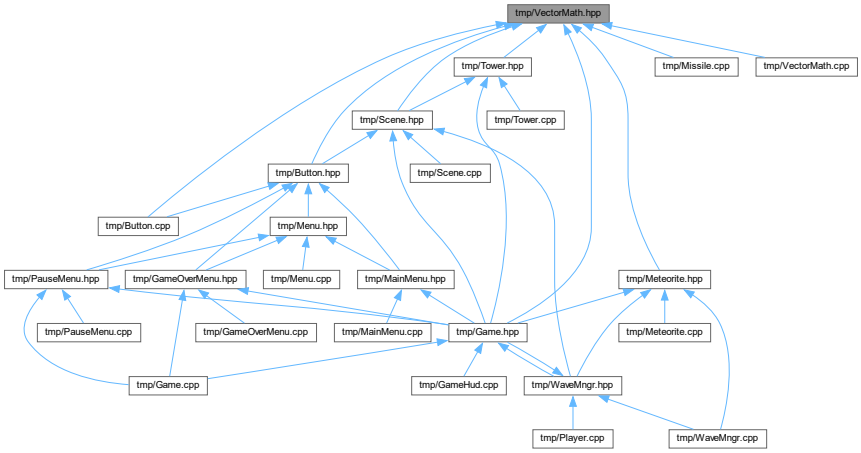
7.87 tmp/VectorMath.hpp File Reference

`#include <SFML/Graphics.hpp>`

Include dependency graph for VectorMath.hpp:



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



Functions

- auto dot (const sf::Vector2f &vectorA, const sf::Vector2f &vectorB) -> float
- auto length (const sf::Vector2f &vector) -> float
- auto distanceBetween (const sf::Vector2f &pointA, const sf::Vector2f &pointB) -> float
- auto normalize (const sf::Vector2f &vector) -> sf::Vector2f
- auto angleBetween (const sf::Vector2f &directionA, sf::Vector2f directionB) -> float
- auto vec2iToVec2f (sf::Vector2i rhs) -> sf::Vector2f
- auto vec2fToVec2i (sf::Vector2f rhs) -> sf::Vector2i

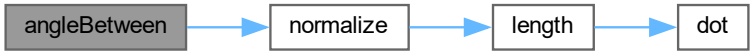
7.87.1 Function Documentation

7.87.1.1 angleBetween()

```
auto angleBetween (
            const sf::Vector2f & directionA,
            sf::Vector2f directionB ) -> float
```

Definition at line 25 of file VectorMath.cpp.

Here is the call graph for this function:

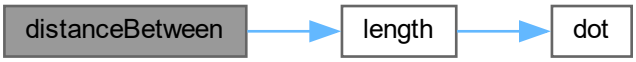


7.87.1.2 distanceBetween()

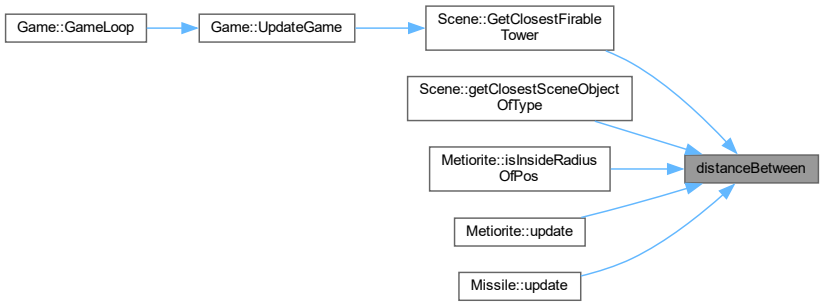
```
auto distanceBetween (
            const sf::Vector2f & pointA,
            const sf::Vector2f & pointB ) -> float
```

Definition at line 15 of file VectorMath.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

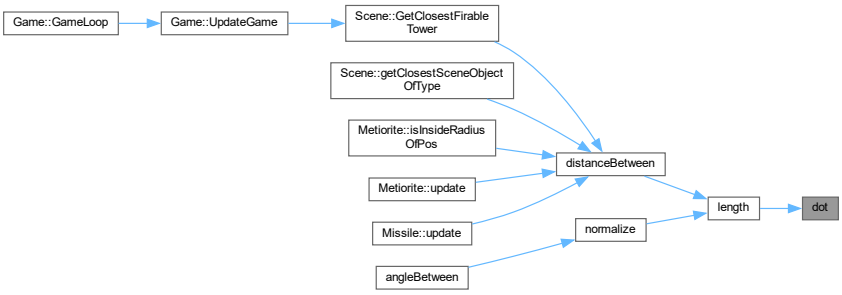


7.87.1.3 dot()

```
auto dot (
            const sf::Vector2f & vectorA,
            const sf::Vector2f & vectorB ) -> float
```

Definition at line 5 of file VectorMath.cpp.

Here is the caller graph for this function:



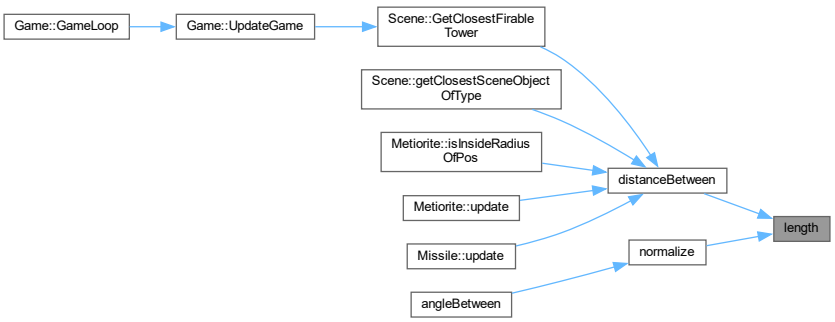
7.87.1.4 length()

```
auto length (
            const sf::Vector2f & vector ) -> float
```

Definition at line 10 of file VectorMath.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



7.87.1.5 normalize()

```
auto normalize (
            const sf::Vector2f & vector ) -> sf::Vector2f
```

Definition at line 20 of file VectorMath.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.87.1.6 vec2fToVec2i()

```
auto vec2fToVec2i (
            sf::Vector2f rhs ) -> sf::Vector2i
```

Definition at line 40 of file VectorMath.cpp.

7.87.1.7 vec2iToVec2f()

```
auto vec2iToVec2f (
            sf::Vector2i rhs ) -> sf::Vector2f
```

Definition at line 35 of file VectorMath.cpp.

Here is the caller graph for this function:

7.88 VectorMath.hpp

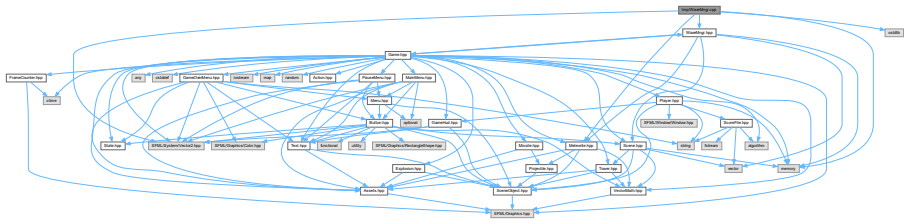
Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <SFML/Graphics.hpp>
00004
00005 auto dot(const sf::Vector2f& vectorA, const sf::Vector2f& vectorB) -> float;
00006 auto length(const sf::Vector2f& vector) -> float;
00007 auto distanceBetween(const sf::Vector2f& pointA, const sf::Vector2f& pointB) -> float;
00008 auto normalize(const sf::Vector2f& vector) -> sf::Vector2f;
00009 auto angleBetween(const sf::Vector2f& directionA, sf::Vector2f directionB) -> float;
00010 auto vec2iToVec2f(sf::Vector2i rhs) -> sf::Vector2f;
00011 auto vec2fToVec2i(sf::Vector2f rhs) -> sf::Vector2i;
```

7.89 tmp/WaveMngr.cpp File Reference

```
#include "Meteorite.hpp"
#include "WaveMngr.hpp"
#include <SFML/System/Vector2.hpp>
#include <cstdlib>
#include <memory>
```

Include dependency graph for WaveMngr.cpp:



7.90 WaveMngr.cpp

Go to the documentation of this file.

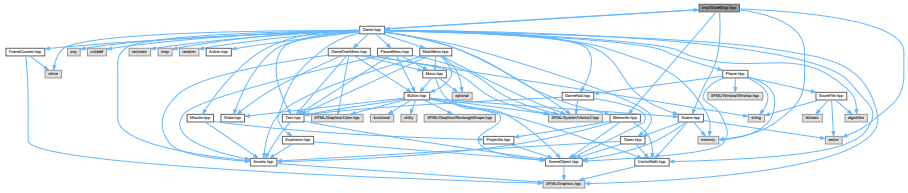
```
00001 #include "Meteorite.hpp"
00002 #include "WaveMngr.hpp"
00003 #include <SFML/System/Vector2.hpp>
00004 #include <cstdlib>
00005 #include <memory>
00006
00007 int WaveMngr::m_current_wave = 0;
00008 int WaveMngr::m_difficulty = 1; // goes from 0 and up. it will scale the amout of enimies
00009 int WaveMngr::m_enimies_remaining = 0;
00010
00011 void WaveMngr::constructWave(Scene& gameScene)
00012 {
00013     int inc = 0;
00014     do
00015     {
00016         gameScene.AddSceneObject(std::make_shared<Metiorite>(
00017             sf::Vector2f(rand() % Game::width, 0),
00018             sf::Vector2f(rand() % Game::width, Game::height - Game::BOTTOM_PADDING)));
00019         m_enimies_remaining++;
00020         inc++;
00021     } while((m_current_wave * m_difficulty) > inc);
00022 }
00023
00024 void WaveMngr::enemyDestroyed()
00025 {
00026     m_enimies_remaining--;
00027 }
00028
00029 void WaveMngr::passWave()
```

```
00030 {
00031     m_current_wave++;
00032 }
00033
00034 auto WaveMngr::getDifficulty() -> int
00035 {
00036     return m_difficulty;
00037 }
00038
00039 auto WaveMngr::getWave() -> int
00040 {
00041     return m_current_wave;
00042 }
00043
00044 auto WaveMngr::getEnemiesRemaning() -> int
00045 {
00046     return m_enimies_remaining;
00047 }
00048
00049 void WaveMngr::reset()
00050 {
00051     m_current_wave = 0;
00052     m_difficulty = 1;
00053     m_enimies_remaining = 0;
00054 }
```

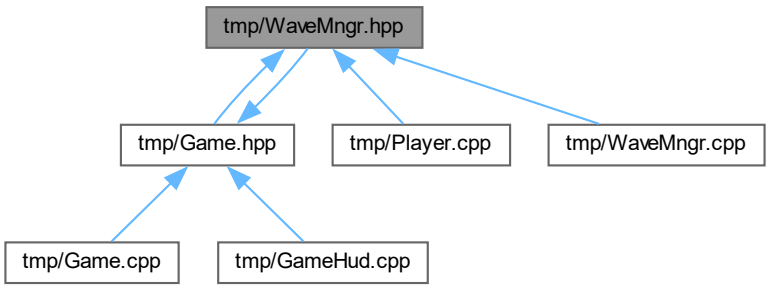
7.91 tmp/WaveMngr.hpp File Reference

```
#include <memory>
#include <vector>
#include "Game.hpp"
#include "Meteorite.hpp"
#include "Scene.hpp"
```

Include dependency graph for WaveMngr.hpp:



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



Classes

- class WaveMngr

7.92 WaveMngr.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <memory>
00003 #include <vector>
00004
00005 #include "Game.hpp"
00006 #include "Meteorite.hpp"
00007 #include "Scene.hpp"
00008
00009 class WaveMngr
00010 {
00011     static int m_current_wave; // Rechable everywhere but private in my eyes ;´()
00012     static int m_difficulty;
00013     static int m_enimies_remaining;
00014
00015   public:
00016     static void constructWave(
00017         Scene& gameScene); // Constructs Enimies and adds them to the gamescene
00018     static void enemyDestroyed();
00019     static void passWave();
00020     static auto getDifficulty() -> int;
00021     static auto getWave() -> int;
00022     static auto getEnemiesRemaning() -> int;
00023     static void reset();
00024 };
```

Index