Evolving Dungeon

Generated by Doxygen 1.10.0

1 Namespace Index **1**

1.1 Namespace List 1

2 Hierarchical Index **3**

2.1 Class Hierarchy 3

3 Class Index **5**

3.1 Class List 5

4 File Index **7**

4.1 File List 7

5 Namespace Documentation **9**

5.1 ED::Constant Namespace Reference 9

5.1.1 Detailed Description 9

6 Class Documentation **11**

6.1 ED::System::Button Class Reference 11

6.1.1 Detailed Description 12

6.1.2 Constructor & Destructor Documentation 12

6.1.2.1 Button() 12

6.1.3 Member Function Documentation 12

6.1.3.1 clickButton() 12

6.1.3.2 position() 12

6.1.3.3 setPosition() 13

6.1.3.4 setString() 13

6.1.3.5 setType() 13

6.1.3.6 size() 13

6.1.3.7 string() 14

6.1.3.8 type() 14

6.2 ED::System::Game Class Reference 14

6.2.1 Detailed Description 14

6.3 ED::System::GameState Class Reference 15

6.3.1 Detailed Description 15

6.3.2 Member Function Documentation 15

6.3.2.1 handleEvent() 15

6.3.2.2 render() 15

6.3.2.3 update() 16

6.4 ED::System::MenuState Class Reference 16

6.4.1 Detailed Description 16

6.4.2 Constructor & Destructor Documentation 17

6.4.2.1 MenuState() 17

6.4.3 Member Function Documentation 18

6.4.3.1 handleEvent() 18

6.4.3.2 render() 18
6.4.3.3 setPlayingState() 18
6.4.3.4 update() 18
6.5 ED::System::PausedState Class Reference 19
6.5.1 Detailed Description 19
6.5.2 Constructor & Destructor Documentation 19
6.5.2.1 PausedState() 19
6.5.3 Member Function Documentation 20
6.5.3.1 handleEvent() 20
6.5.3.2 render() 20
6.5.3.3 setMenuState() 20
6.5.3.4 setPlayingState() 20
6.5.3.5 update() 21
6.6 ED::System::PlayingState Class Reference 21
6.6.1 Detailed Description 22
6.6.2 Constructor & Destructor Documentation 22
6.6.2.1 PlayingState() 22
6.6.3 Member Function Documentation 22
6.6.3.1 handleEvent() 22
6.6.3.2 render() 22
6.6.3.3 setMenuState() 23
6.6.3.4 setPausedState() 23
6.6.3.5 update() 23
6.7 Resources< Resource, ID > Class Template Reference 23
6.7.1 Detailed Description 24
6.7.2 Member Function Documentation 24
6.7.2.1 load() 24
6.7.2.2 resource() [1/2] 24
6.7.2.3 resource() [2/2] 25

7 File Documentation **27**
7.1 Constants.hpp 27
7.2 Resources.hpp 28
7.3 Resources.inl 28
7.4 Button.hpp 28
7.5 Game.hpp 29
7.6 GameState.hpp 29
7.7 MenuState.hpp 30
7.8 PausedState.hpp 30
7.9 PlayingState.hpp 31

Index **33**

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

ED::Constant
Constants for the evolving dungeon game 9

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

- sf::Drawable
  - ED::System::Button 11
- ED::System::Game 14
- ED::System::GameState 15
  - ED::System::MenuState 16
  - ED::System::PausedState 19
  - ED::System::PlayingState 21
- Resources< Resource, ID > 23
- Resources< sf::Font, ED::Font::ID > 23
- Resources< sf::Texture, ED::Texture::ID > 23

Chapter 3

Class Index

3.1 Class List

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

ED::System::Button
Class for clickable and non-clickable buttons 11
ED::System::Game
The class responsible for running the application 14
ED::System::GameState
Abstract class for state of the game 15
ED::System::MenuState
Class for the state of the game when inside menu 16
ED::System::PausedState
Class for the state of the game when playing the game (unpaused) 19
ED::System::PlayingState
Class for the state of the game when playing the game (unpaused) 21
Resources< Resource, ID >
Class for handling resource allocation 23

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

include/Constants.hpp 27
include/Resources.hpp 28
include/Resources.inl 28
include/system/Button.hpp 28
include/system/Game.hpp 29
include/system/states/GameState.hpp 29
include/system/states/MenuState.hpp 30
include/system/states/PausedState.hpp 30
include/system/states/PlayingState.hpp 31

Chapter 5

Namespace Documentation

5.1 ED::Constant Namespace Reference

Constants for the evolving dungeon game.

Enumerations

- enum class ButtonType {
  none , **startGame** , **unpauseGame** , **quitToMenu** ,
  quitToDesktop }

  Types of buttons.

5.1.1 Detailed Description

Constants for the evolving dungeon game.

Chapter 6

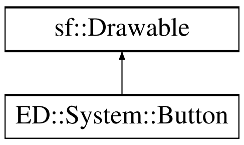
Class Documentation

6.1 ED::System::Button Class Reference

Class for clickable and non-clickable buttons.

`#include <Button.hpp>`

Inheritance diagram for ED::System::Button:



Public Member Functions

- Button (std::string text, sf::Vector2f position, sf::Color color, sf::Color fontColor, Constant::ButtonType type, const Fonts &fonts)

  Construct a new Button object (button is also used for non-clickable objects)
- **Button** ()=default

  Construct a new Button object (button is also used for non-clickable objects)
- ∼**Button** ()

  Destroy the Button object.
- void **calcPosition** ()

  Calculate the position of the button's box and text.
- sf::Vector2f position () const

  Get the position of the button.
- void setPosition (sf::Vector2f)

  Set the position of the button.
- Constant::ButtonType type () const

  Get the type of the button.
- void setType (Constant::ButtonType)

  Set the type of the button.
- sf::Vector2f size () const

  Get the size of the button.
- void setString (std::string)

  Set the button's text's string.
- std::string string () const

  Get the button's text's string.
- bool clickButton (sf::Vector2i mousePosition) const

6.1.1 Detailed Description

Class for clickable and non-clickable buttons.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Button()

```
ED::System::Button::Button (
            std::string text,
            sf::Vector2f position,
            sf::Color color,
            sf::Color fontColor,
            Constant::ButtonType type,
            const Fonts & fonts )
```

Construct a new Button object (button is also used for non-clickable objects)

Parameters

text	Text inside the button
position	Position of the button
color	Color of the button
fontColor	Color of the font
type	Type of the button, e.g. is the button clickable or not
fonts	Fonts used in the game

6.1.3 Member Function Documentation

6.1.3.1 clickButton()

```
bool ED::System::Button::clickButton (
            sf::Vector2i mousePosition ) const
```

Parameters

mousePosition	Position of the mouse

Returns

Whether mouse is over a clickable button or not

6.1.3.2 position()

```
sf::Vector2f ED::System::Button::position ( ) const
```

Get the position of the button.

Returns

Current position of the button

6.1.3.3 setPosition()

```
void ED::System::Button::setPosition (
             sf::Vector2f position )
```

Set the position of the button.

Parameters

position	The new position of the button

6.1.3.4 setString()

```
void ED::System::Button::setString (
             std::string string )
```

Set the button's text's string.

Parameters

string	The new button's text's string

6.1.3.5 setType()

```
void ED::System::Button::setType (
             Constant::ButtonType type )
```

Set the type of the button.

Parameters

type	The new type of the button

6.1.3.6 size()

```
sf::Vector2f ED::System::Button::size ( ) const
```

Get the size of the button.

Returns

Size of the button

6.1.3.7 string()

```
std::string ED::System::Button::string ( ) const
```

Get the button's text's string.

Returns

Button's text's string

6.1.3.8 type()

```
Constant::ButtonType ED::System::Button::type ( ) const
```

Get the type of the button.

Returns

Type of the button

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

- include/system/Button.hpp
- src/system/Button.cpp

6.2 ED::System::Game Class Reference

The class responsible for running the application.

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

Public Member Functions

- **Game** ()

  Construct a new Game object.
- void **run** ()

  Run the game loop.

6.2.1 Detailed Description

The class responsible for running the application.

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

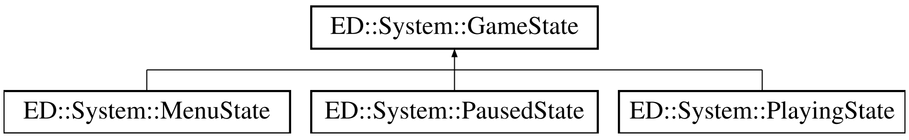
- include/system/Game.hpp
- src/system/Game.cpp

6.3 ED::System::GameState Class Reference

Abstract class for state of the game.

```
#include <GameState.hpp>
```

Inheritance diagram for ED::System::GameState:



Public Member Functions

- virtual GameState ∗ handleEvent (const sf::Event &)=0

    Handle user input events.
- virtual void update (sf::Time)=0

    Update the state.
- virtual void render ()=0

    Render the state.
- virtual ∼**GameState** ()=default

    Destroy the Game State object.

6.3.1 Detailed Description

Abstract class for state of the game.

6.3.2 Member Function Documentation

6.3.2.1 handleEvent()

```
virtual GameState * ED::System::GameState::handleEvent (
            const sf::Event & )  [pure virtual]
```

Handle user input events.

Returns

Pointer to new current state or nullptr if did not change

Implemented in ED::System::MenuState, ED::System::PausedState, and ED::System::PlayingState.

6.3.2.2 render()

```
virtual void ED::System::GameState::render ( )  [pure virtual]
```

Render the state.

Implemented in ED::System::MenuState, ED::System::PausedState, and ED::System::PlayingState.

6.3.2.3 update()

```
virtual void ED::System::GameState::update (
             sf::Time  )  [pure virtual]
```

Update the state.

Implemented in ED::System::MenuState, ED::System::PausedState, and ED::System::PlayingState.

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

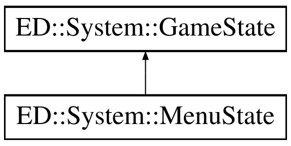
- include/system/states/GameState.hpp

6.4 ED::System::MenuState Class Reference

Class for the state of the game when inside menu.

```
#include <MenuState.hpp>
```

Inheritance diagram for ED::System::MenuState:



Public Member Functions

- MenuState (sf::RenderWindow &window, const Fonts &fonts)

  Construct a new Menu State object.
- GameState ∗ handleEvent (const sf::Event &) final

  Handle user input events.
- void update (sf::Time) final

  Update the state.
- void render () final

  Render the state.
- void setPlayingState (PlayingState &playingState)

  Set the Playing State object.

Public Member Functions inherited from ED::System::GameState

- virtual ∼**GameState** ()=default

  Destroy the Game State object.

6.4.1 Detailed Description

Class for the state of the game when inside menu.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 MenuState()

```
ED::System::MenuState::MenuState (
             sf::RenderWindow & window,
             const Fonts & fonts )
```

Construct a new Menu State object.

Parameters

window	Window of the state
fonts	Fonts of the state

6.4.3 Member Function Documentation

6.4.3.1 handleEvent()

```
GameState * ED::System::MenuState::handleEvent (
             const sf::Event & event )  [final], [virtual]
```

Handle user input events.

Returns

Pointer to new current state or nullptr if did not change

Implements ED::System::GameState.

6.4.3.2 render()

```
void ED::System::MenuState::render ( )  [final], [virtual]
```

Render the state.

Implements ED::System::GameState.

6.4.3.3 setPlayingState()

```
void ED::System::MenuState::setPlayingState (
             PlayingState & playingState )
```

Set the Playing State object.

Parameters

playingState	The new playing state object

6.4.3.4 update()

```
void ED::System::MenuState::update (
             sf::Time  )  [final], [virtual]
```

Update the state.

Implements ED::System::GameState.

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

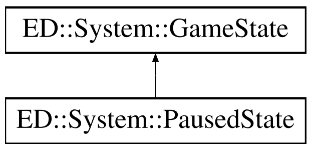
- include/system/states/MenuState.hpp
- src/system/states/MenuState.cpp

6.5 ED::System::PausedState Class Reference

Class for the state of the game when playing the game (unpaused)

`#include <PausedState.hpp>`

Inheritance diagram for ED::System::PausedState:



Public Member Functions

- PausedState (sf::RenderWindow &window, const Fonts &fonts)

  Construct a new Paused State object.
- GameState ∗ handleEvent (const sf::Event &) final

  Handle user input events.
- void update (sf::Time) final

  Update the state.
- void render () final

  Render the state.
- void setMenuState (MenuState &menuState)

  Set the Menu State object.
- void setPlayingState (PlayingState &playingState)

  Set the Playing State object.

Public Member Functions inherited from ED::System::GameState

- virtual ∼**GameState** ()=default

  Destroy the Game State object.

6.5.1 Detailed Description

Class for the state of the game when playing the game (unpaused)

6.5.2 Constructor & Destructor Documentation

6.5.2.1 PausedState()

```
ED::System::PausedState::PausedState (
            sf::RenderWindow & window,
            const Fonts & fonts )
```

Construct a new Paused State object.

Parameters

window	Window of the state
fonts	Fonts of the state

6.5.3 Member Function Documentation

6.5.3.1 handleEvent()

```
GameState * ED::System::PausedState::handleEvent (
            const sf::Event & event )  [final], [virtual]
```

Handle user input events.

Returns

Pointer to new current state or nullptr if did not change

Implements ED::System::GameState.

6.5.3.2 render()

```
void ED::System::PausedState::render ( )  [final], [virtual]
```

Render the state.

Implements ED::System::GameState.

6.5.3.3 setMenuState()

```
void ED::System::PausedState::setMenuState (
            MenuState & menuState )
```

Set the Menu State object.

Parameters

menuState	The new menu state object

6.5.3.4 setPlayingState()

```
void ED::System::PausedState::setPlayingState (
            PlayingState & playingState )
```

Set the Playing State object.

Parameters

playingState	The new playing state object

6.5.3.5 update()

```
void ED::System::PausedState::update (
            sf::Time  )  [final], [virtual]
```

Update the state.

Implements ED::System::GameState.

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

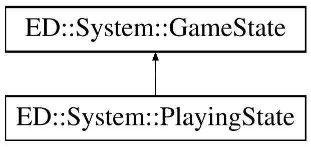
- include/system/states/PausedState.hpp
- src/system/states/PausedState.cpp

6.6 ED::System::PlayingState Class Reference

Class for the state of the game when playing the game (unpaused)

`#include <PlayingState.hpp>`

Inheritance diagram for ED::System::PlayingState:



Public Member Functions

- PlayingState (sf::RenderWindow &window, const Fonts &fonts, const Textures &textures)

    Construct a new Playing State object.
- GameState ∗ handleEvent (const sf::Event &) final

    Handle user input events.
- void update (sf::Time) final

    Update the state.
- void render () final

    Render the state.
- void setMenuState (MenuState &menuState)

    Set the Menu State object.
- void setPausedState (PausedState &pausedState)

    Set the Paused State object.

Public Member Functions inherited from ED::System::GameState

- virtual ∼**GameState** ()=default

  Destroy the Game State object.

6.6.1 Detailed Description

Class for the state of the game when playing the game (unpaused)

6.6.2 Constructor & Destructor Documentation

6.6.2.1 PlayingState()

```
ED::System::PlayingState::PlayingState (
            sf::RenderWindow & window,
            const Fonts & fonts,
            const Textures & textures )
```

Construct a new Playing State object.

Parameters

window	Window of the state
fonts	Fonts of the state
textures	Texture of the state

6.6.3 Member Function Documentation

6.6.3.1 handleEvent()

```
GameState * ED::System::PlayingState::handleEvent (
            const sf::Event & event ) [final], [virtual]
```

Handle user input events.

Returns

Pointer to new current state or nullptr if did not change

Implements ED::System::GameState.

6.6.3.2 render()

```
void ED::System::PlayingState::render ( ) [final], [virtual]
```

Render the state.

Implements ED::System::GameState.

6.6.3.3 setMenuState()

```
void ED::System::PlayingState::setMenuState (
            MenuState & menuState )
```

Set the Menu State object.

Parameters

menuState	The new menu state object

6.6.3.4 setPausedState()

```
void ED::System::PlayingState::setPausedState (
            PausedState & pausedState )
```

Set the Paused State object.

Parameters

pausedState	The new paused state object

6.6.3.5 update()

```
void ED::System::PlayingState::update (
            sf::Time  )  [final], [virtual]
```

Update the state.

Implements ED::System::GameState.

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

- include/system/states/PlayingState.hpp
- src/system/states/PlayingState.cpp

6.7 Resources< Resource, ID > Class Template Reference

Class for handling resource allocation.

`#include <Resources.hpp>`

Public Member Functions

- void load (ID id, const std::string &filename)

  Load specified resource.
- Resource & resource (ID id)

  Get resource object.
- const Resource & resource (ID id) const

  Get constant reference to the resource object.

6.7.1 Detailed Description

template<typename Resource, typename ID>
class Resources< Resource, ID >

Class for handling resource allocation.

Template Parameters

ResourceType	Resource type
ID	Identifier for a resource

Note

Heavily inspired by the book SFML Game Development by Artur Moreira etc.

6.7.2 Member Function Documentation

6.7.2.1 load()

```
template<typename Resource , typename ID >
void Resources< Resource, ID >::load (
             ID id,
             const std::string & filename )
```

Load specified resource.

Parameters

id	Identifier of the resource
filename	File name of the resource

6.7.2.2 resource() [1/2]

```
template<typename Resource , typename ID >
Resource & Resources< Resource, ID >::resource (
             ID id )
```

Get resource object.

Parameters

id	Identifier of the resource

Returns

The resource object

6.7.2.3 resource() `[2/2]`

```
template<typename Resource , typename ID >
const Resource & Resources< Resource, ID >::resource (
              ID id ) const
```

Get constant reference to the resource object.

Parameters

id	Identifier of the resource

Returns

Constant reference to the resource object

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

- include/Constants.hpp
- include/Resources.hpp
- include/Resources.inl

Chapter 7

File Documentation

7.1 Constants.hpp

```
#pragma once

#include <string>

namespace ED {
namespace Constant {
    static unsigned windowHeight = 1000;

    static unsigned windowWidth = 1000;

    static unsigned frameRate = 60;

    static std::string gameName = "Evolving Dungeon";

    enum class ButtonType {
        none,
        startGame,
        unpauseGame,
        quitToMenu,
        quitToDesktop,
    };
} // namespace Constant
namespace Texture {
    enum class ID {
        enemy,
        player,
        tiles,
        weapon,
        item,
    };
} // namespace Texture
namespace Font {
    enum class ID {
        normal,
    };
} // namespace Font
namespace Sound {
    enum class ID {}; // FIXME: Should have id's for different possible sounds
} // namespace Sound
} // namespace ED

// Forward declarations
template <typename Resource, typename ID>
class Resources;

namespace sf {
class Texture;
class Font;
class Sound;
} // namespace sf

typedef Resources<sf::Texture, ED::Texture::ID> Textures;
typedef Resources<sf::Font, ED::Font::ID> Fonts;
typedef Resources<sf::Sound, ED::Sound::ID> Sounds;
```

7.2 Resources.hpp

```
#pragma once

#include <cassert>
#include <map>
#include <memory>
#include <stdexcept>
#include <string>

template <typename Resource, typename ID>
class Resources {
public:
    void load(ID id, const std::string& filename);

    Resource& resource(ID id);

    const Resource& resource(ID id) const;

private:
    std::map<ID, std::unique_ptr<Resource» m_resources;
};

#include "Resources.inl"
```

7.3 Resources.inl

```
template <typename Resource, typename ID>
void Resources<Resource, ID>::load(ID id, const std::string& filename)
{
    std::unique_ptr<Resource> resource(new Resource());
    if (!resource->loadFromFile(filename)) {
        throw std::runtime_error("Resources::load - File loading failed for " + filename);
    }

    auto inserted = m_resources.insert(std::make_pair(id, std::move(resource)));

    // In debug mode, checks that there was no previous inserted value for given id
    assert(inserted.second);
}

template <typename Resource, typename ID>
Resource& Resources<Resource, ID>::resource(ID id)
{
    auto found = m_resources.find(id);
    // In debug mode, checks that we did not reach the end pointer, i.e. check that the resource
exists
    assert(found != m_resources.end());
    return *found->second;
}

template <typename Resource, typename ID>
const Resource& Resources<Resource, ID>::resource(ID id) const
{
    auto found = m_resources.find(id);
    // In debug mode, checks that we did not reach the end pointer, i.e. check that the resource
exists
    assert(found != m_resources.end());
    return *found->second;
}
```

7.4 Button.hpp

```
#pragma once

#include "Constants.hpp"
#include "Resources.hpp"
#include <SFML/Graphics.hpp>

namespace ED::System {
class Button : public sf::Drawable {
public:
    Button(std::string text, sf::Vector2f position, sf::Color color, sf::Color fontColor,
Constant::ButtonType type, const Fonts& fonts);

    Button() = default;

    ~Button();

    void calcPosition();

    sf::Vector2f position() const;

    void setPosition(sf::Vector2f);

    Constant::ButtonType type() const;

    void setType(Constant::ButtonType);

    sf::Vector2f size() const;

    void setString(std::string);

    std::string string() const;

    bool clickButton(sf::Vector2i mousePosition) const;

private:
    sf::Text m_text;

    sf::Vector2f m_position;

    sf::Vector2f m_size;

    sf::RectangleShape m_button;

    Constant::ButtonType m_type;

    void draw(sf::RenderTarget& target, const sf::RenderStates& states) const final;
};
} // namespace ED::System
```

7.5 Game.hpp

```
#pragma once

#include "Constants.hpp"
#include "Resources.hpp"
#include "system/Button.hpp"
#include "system/states/MenuState.hpp"
#include "system/states/PausedState.hpp"
#include "system/states/PlayingState.hpp"
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <algorithm>

namespace ED::System {
class Game {
public:
    Game();

    void run();

private:
    Fonts m_fonts;

    Textures m_textures;

    sf::RenderWindow m_window;

    MenuState m_menuState;

    PausedState m_pausedState;

    PlayingState m_playingState;

    GameState* m_currentState;

    sf::Event m_event;

    void loadResources();

    sf::VideoMode init();
};
};
```

7.6 GameState.hpp

```
#pragma once
```

```

#include <SFML/Graphics.hpp>

namespace ED::System {
class GameState {
public:
    virtual GameState* handleEvent(const sf::Event&) = 0;

    virtual void update(sf::Time) = 0;

    virtual void render() = 0;

    virtual ~GameState() = default;
};
};
```

7.7 MenuState.hpp

```
#pragma once

// #include "../Menu.hpp"
// #include "GameState.hpp"
#include "Constants.hpp"
#include "Resources.hpp"
#include "system/Button.hpp"
#include "system/states/GameState.hpp"
#include "system/states/PlayingState.hpp"
#include <SFML/Graphics.hpp>
#include <algorithm>

namespace ED::System {

class PlayingState;
class MenuState : public GameState {
public:
    MenuState(sf::RenderWindow& window, const Fonts& fonts);

    GameState* handleEvent(const sf::Event&) final;

    void update(sf::Time) final;

    void render() final;

    void setPlayingState(PlayingState& playingState);

private:
    sf::RenderWindow& m_window;

    std::vector<Button> m_UI;

    PlayingState* m_playingState;

    void initializeUI(const Fonts& fonts);
};
};
```

7.8 PausedState.hpp

```
#pragma once

#include "Constants.hpp"
#include "Resources.hpp"
#include "system/Button.hpp"
#include "system/states/GameState.hpp"
#include "system/states/MenuState.hpp"
#include "system/states/PlayingState.hpp"
#include <SFML/Graphics.hpp>

namespace ED::System {

class MenuState;
class PlayingState;

class PausedState : public GameState {
public:
    PausedState(sf::RenderWindow& window, const Fonts& fonts);

    GameState* handleEvent(const sf::Event&) final;

    void update(sf::Time) final;

    void render() final;

    void setMenuState(MenuState& menuState);

    void setPlayingState(PlayingState& playingState);

private:
    sf::RenderWindow& m_window;

    std::vector<Button> m_UI;

    sf::RectangleShape m_background;

    MenuState* m_menuState;

    PlayingState* m_playingState;

    void initializeUI(const Fonts& fonts);
};
};
```

7.9 PlayingState.hpp

```
#pragma once

#include "Constants.hpp"
#include "Resources.hpp"
#include "system/Button.hpp"
#include "system/states/GameState.hpp"
#include "system/states/MenuState.hpp"
#include "system/states/PausedState.hpp"
#include <SFML/Graphics.hpp>
#include <algorithm>

namespace ED::System {

class MenuState;
class PausedState;

class PlayingState : public GameState {
public:
    PlayingState(sf::RenderWindow& window, const Fonts& fonts, const Textures& textures);

    GameState* handleEvent(const sf::Event&) final;

    void update(sf::Time) final;

    void render() final;

    void setMenuState(MenuState& menuState);

    void setPausedState(PausedState& pausedState);

private:
    sf::RenderWindow& m_window;

    MenuState* m_menuState;

    PausedState* m_pausedState;
};
};
```

Index

Button
    ED::System::Button, 12

clickButton
    ED::System::Button, 12

ED::Constant, 9
ED::System::Button, 11
    Button, 12
    clickButton, 12
    position, 12
    setPosition, 13
    setString, 13
    setType, 13
    size, 13
    string, 13
    type, 14
ED::System::Game, 14
ED::System::GameState, 15
    handleEvent, 15
    render, 15
    update, 15
ED::System::MenuState, 16
    handleEvent, 18
    MenuState, 17
    render, 18
    setPlayingState, 18
    update, 18
ED::System::PausedState, 19
    handleEvent, 20
    PausedState, 19
    render, 20
    setMenuState, 20
    setPlayingState, 20
    update, 21
ED::System::PlayingState, 21
    handleEvent, 22
    PlayingState, 22
    render, 22
    setMenuState, 22
    setPausedState, 23
    update, 23

handleEvent
    ED::System::GameState, 15
    ED::System::MenuState, 18
    ED::System::PausedState, 20
    ED::System::PlayingState, 22

include/Constants.hpp, 27
include/Resources.hpp, 28
include/Resources.inl, 28
include/system/Button.hpp, 28
include/system/Game.hpp, 29
include/system/states/GameState.hpp, 29
include/system/states/MenuState.hpp, 30
include/system/states/PausedState.hpp, 30
include/system/states/PlayingState.hpp, 31

load
    Resources< Resource, ID >, 24

MenuState
    ED::System::MenuState, 17

PausedState
    ED::System::PausedState, 19
PlayingState
    ED::System::PlayingState, 22
position
    ED::System::Button, 12

render
    ED::System::GameState, 15
    ED::System::MenuState, 18
    ED::System::PausedState, 20
    ED::System::PlayingState, 22
resource
    Resources< Resource, ID >, 24
Resources< Resource, ID >, 23
    load, 24
    resource, 24

setMenuState
    ED::System::PausedState, 20
    ED::System::PlayingState, 22
setPausedState
    ED::System::PlayingState, 23
setPlayingState
    ED::System::MenuState, 18
    ED::System::PausedState, 20
setPosition
    ED::System::Button, 13
setString
    ED::System::Button, 13
setType
    ED::System::Button, 13
size
    ED::System::Button, 13
string
    ED::System::Button, 13

type
ED::System::Button, 14

update
ED::System::GameState, 15
ED::System::MenuState, 18
ED::System::PausedState, 21
ED::System::PlayingState, 23