Icy Tower clone game.
„AEI Tower”

Łukasz Kwiecień

Computer Programming
Semester Project
Teacher: dr inż. Krzysztof Pasterak

Wydział Automatyki, Elektroniki i Informatyki
Politechnika Śląska
Polska
25.01.2020

Contents

1 Topic

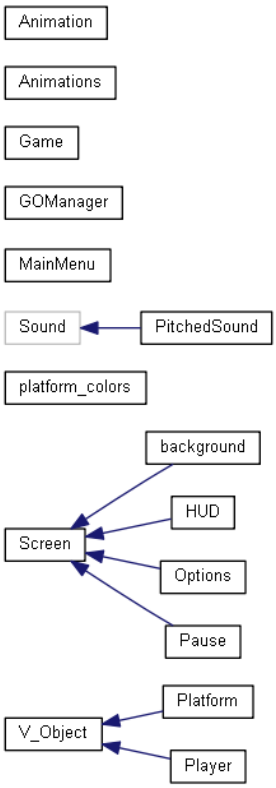
The main goal of the project was to get familiar with using object oriented features and mechanisms of the C++ programming language. I wanted to create a game with simple mechanics but with absorbing gameplay. My choice was to create icy tower clone game. Although the game is relatively simple, the implementation of some mechanics was quite challenging. The main goal of the game is to jump over as many platforms as possible. Platforms are generated endlessly and every 25 platforms their color is changed. Moreover not only the color is changed, platforms are moving down faster and faster, increasing the difficulty for the player.

2 Analysis and Development

The process of the design can be divided into three steps.

The first step was to decide which library will be used to create the game. I wanted to choose library that would be user-friendly and provide tools needed to create a game (for example graphical interface). SFML library had met my requirements, therefore I decided to use it.

The next step concerned the class structure, I wanted to create separate classes responsible for specific features. Finally, the class structure after changes that occurred during the implementation process, looks as follows (more detailed class diagrams, such as dependency graphs etc., are included in the documentation in the appendix) :



The last step was to decide whether game will be frame rate dependent or time dependent. Because the game is not very complex and doesn't require a lot of resources I chose the first option.

2.1 Data structures

Data structures used in the project:

- `std::map<std::string, V_Object*>` collection of std::pair objects containing string and pointer to V_Object.
- `std::list<OptionsItem>` list of type OptionsItem, stores items that compose the options menu.
- `std::list<MenuItem>` list of type MenuItem, stores items that compose the menu.

2.2 Algorithms

Platform has to be generated in the correct position and length to be displayed within the game area. Function that generates position is very simple, provides that the generated position will be inside the game area and fit the shortest platform :

```
int Platform::generatepos()
{
    std::random_device rd;
    std::mt19937 rand(rd());
    std::uniform_int_distribution<> pos(144, 496);
    return pos(rd);
}
```

In order to avoid the generation of platforms that are too long to be displayed in the previously generated position, the algorithm that checks this condition was implemented in the generatelength function:

```
int Platform::generatelength(int pos)
{
    int tmp = pos;
    std::random_device rd;
    std::mt19937 rand(rd());
    std::uniform_int_distribution<> len(7, 13);
    int length = len(rd) * 20;
    // Checking left bound
    tmp -= 74;
    if ((tmp - (length / 2)) < 0)
        length = length - 2*((length/2)-tmp);
    // Checking right bound
    tmp +=  74;
    if ((tmp + (length / 2)) > 570)
        length = length - 2*((tmp + (length / 2))-570);
    if (length < 140)
        return 140;
    else
        return length;
}
```

The part of the function responsible for the generation of the random number is very similar to the previous function. The only difference is of course the range of the generated numbers. In this case the range is between 7 and 13, 140 pixels is the length of the shortest possible platform when 260 is the length of the longest possible one.

Let's take a look at the algorithm mentioned before:

```
    // Checking left bound
    tmp -= 74;
    if ((tmp - (length / 2)) < 0)
        length = length - 2*((length/2)-tmp);
    // Checking right bound
    tmp +=  74;
    if ((tmp + (length / 2)) > 570)
        length = length - 2*((tmp + (length / 2))-570);
    if (length < 140)
        return 140;
    else
        return length;
```

At first the collision with the left bound is checked. When the platform is too long, the excess of pixels is subtracted from its length. After that the correct length of the platform is obtained. Then the collision with the right bound is checked. The process is the same as it is in the left bound case. At the end, after all corrections the length of the platform is returned.

2.3 Physics

2.3.1 Running

Player can run in either left or right direction. After pressing the key, a constant value is added or subtracted (depending on the direction of movement) to the player's x velocity until it reaches maximum speed. In a situation when the keys responsible for horizontal movement are not pressed, the character is decelerated by adding or subtracting a small fixed value. By doing this the "slow down" effect is achieved and the movement seems smoother.
Example code for moving left:

```
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
    {
        if (_velocityX > 0)_velocityX = -0.085f;
        else _velocityX -= 0.085f;
        GetSprite().move(_velocityX, 0);
    }
```

Code for deceleration:

```
        if (_velocityX < -0.125f)

            _velocityX += 0.125f;
        else if (_velocityX > 0.125f)

            _velocityX -= 0.125f;
        else {

            _velocityX = 0;
        }
```

2.3.2 Jumping and Gravity

Jumping

The physics of jumping is very simple. Player can only jump when is standing on the ground. After pressing the key the power of the jump is calculated. Jump power is dependent on the current x velocity of the player.

```
if (_onGround == true) {
            if ((_velocityX > 4.5f) || (-_velocityX > 4.5f))
                _jumpPower = 2;
            else if ((_velocityX > 3.9f) || (-_velocityX > 3.9f))
                _jumpPower = 1;
            else
                _jumpPower = 0;
```

Basing on the power of the jump the y velocity is calculated.

```
if (_onGround == true) {
switch (_jumpPower) {
            case 0:
            {
                _velocityY = -5.75;
                if (_sound == 1) {
                    sound.setBuffer(soundHop);
                    sound.playPitched();
                }

                break;
            }
            case 1:
            {
                _velocityY = fabs(_velocityX) * -0.5 - 5.75;
                if (_sound == 1) {
                    sound.setBuffer(soundHop);
                    sound.playPitched();
                }
                break;
            }
            case 2:
            {
                _velocityY = fabs(_velocityX) * -1 - 5.75;
              // Texture of the rotating animation is 8 pixels higher than normal
                GetSprite().move(0, -8);
                if (_sound == 1) {
                    sound.setBuffer(soundWhoopie);
                    sound.playPitched();
                }
                break;
            }
            }
```

Then the player is moved up by the value of y velocity and the corresponding sound is played.

Gravity

```
    if (_onGround == false) {
        if (_velocityY < _maxvelocityY) {
            _velocityY += _gravity;
        }
    }
```

After each frame a small fixed value is added to the y velocity until it reaches the max value or player lands on the ground. It works like gravity in real world, player is being pulled to the ground. In the first part of the jump player is flying up (y velocity is smaller than zero) but when the value of y velocity becomes greater than zero our character starts to fall. Adding a constant value to the y velocity simulates gravitational acceleration.

3 External specification

User's manual.

3.1 Controls

The following keys are used to control the player and the game:
* **Left arrow key** - move left.
* **Right arrow key** - move right.
* **Up arrow key or spacebar** - jump.
* **Down arrow key** - stop.
* **Escape** - pause the game.
* **Enter** - proceed.

3.2 Gameplay

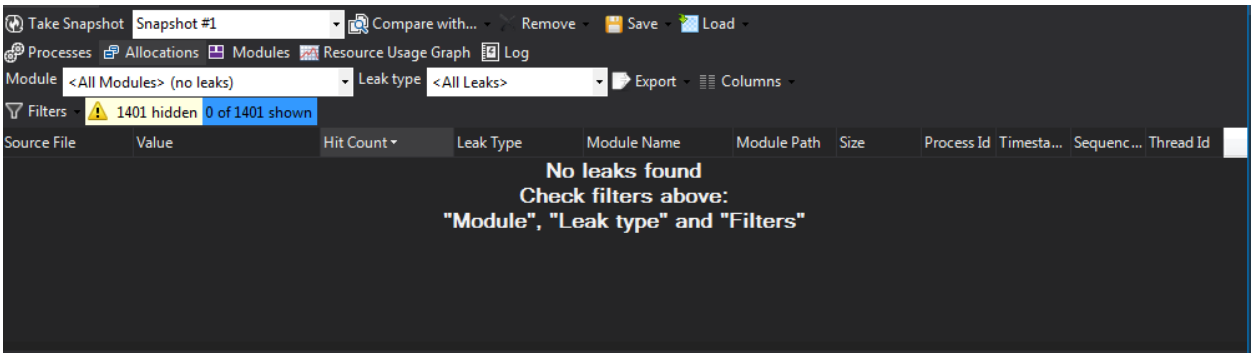
Like in the original Icy Tower, the only goal is to jump over as many platforms as possible to get points. The player (represented by the devil character) goes up using infinitely generated platforms which are moving down. Each platform increases player's score by 10. After every 25 platforms the platform type is changed, it means that next 25 platforms will be generated in a color different than the previous ones. Moreover not only the color is changed, but the speed of the platforms moving down is increased. The higher the player is, the more difficult the game becomes. Player can go up using one of the 3 types of jump. First type is the default jump, character jumps with power enough to land one platform higher. Second type of jump is enabled when the player reaches medium horizontal speed, character jumps with a power that allows him to land two platforms higher. Last type is enabled when the player reaches very large or maximum horizontal speed, the power of the jump is dependent on his current horizontal velocity. This type of jump is the most powerful one, it allows the player to jump over more platforms than in the other two cases. What is more, during this jump a special animation (character starts to rotate in the air) is started and the special combo sound (whoopie) is played. When the player falls outside the lower edge of the screen the game is over. Special game over screen is displayed. On that screen, the player can read his score and the number of the last platform he landed on.

4 Internal specification

Internal specifiaction is moved to appendix. Documentation was generated using the Doxygen.

5 Testing

Game was tested for memory leaks. Deleaker software did not detect any memory leaks.



6 Conclusions

Writing a program using the object oriented paradigm for the first time is difficult. It forces us to change our way of thinking in the design process, however, it gives much more possibilities and bigger control over the code. More time is needed for the design of the class structure, but it pays off during the coding process making the work a lot easier. OOP features such as polymorphism and overloaded operators are very useful.

Appendix
Technical Documentation

AEI Tower

Generated by Doxygen 1.8.17

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 Animation Class Reference

Static Public Attributes

- static sf::IntRect **walk_right_1**
- static sf::IntRect **walk_right_2**
- static sf::IntRect **walk_right_3**
- static sf::IntRect **walk_right_4**
- static sf::IntRect **walk_left_1**
- static sf::IntRect **walk_left_2**
- static sf::IntRect **walk_left_3**
- static sf::IntRect **walk_left_4**
- static sf::IntRect **idle_1**
- static sf::IntRect **idle_2**
- static sf::IntRect **idle_3**
- static sf::IntRect **jump_right_1**
- static sf::IntRect **jump_right_2**
- static sf::IntRect **jump_right_3**
- static sf::IntRect **jump_left_1**
- static sf::IntRect **jump_left_2**
- static sf::IntRect **jump_left_3**
- static sf::IntRect **edge_left_1**
- static sf::IntRect **edge_left_2**
- static sf::IntRect **edge_right_1**
- static sf::IntRect **edge_right_2**
- static sf::IntRect **jump**
- static sf::IntRect **rotate**

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

- Animation.h

4.2 Animations Class Reference

`#include <Animation.h>`

4.2.1 Detailed Description

Holds rectangle coordinates for each appropriate player animation from sheet.png file. Every member of the class is static, so it can be accessed without creating an object.

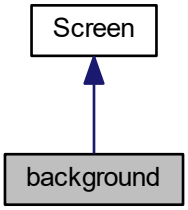
The documentation for this class was generated from the following file:

- Animation.h

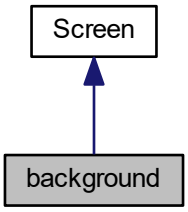
4.3 background Class Reference

`#include <background.h>`

Inheritance diagram for background:



Collaboration diagram for background:



Public Member Functions

- background ()
- ~background ()
- void show (sf::RenderWindow &window)

Additional Inherited Members

4.3.1 Detailed Description

A class that shows the game background. Inherits Screen publicly.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 background()

```
background::background ( )
```

Constructor.

4.3.2.2 ~background()

```
background::~background ( )
```

Destructor.

4.3.3 Member Function Documentation

4.3.3.1 show()

```
void background::show (
            sf::RenderWindow & window )
```

Calls screen show function and displays the sprite in the window.

Parameters

window	render window.

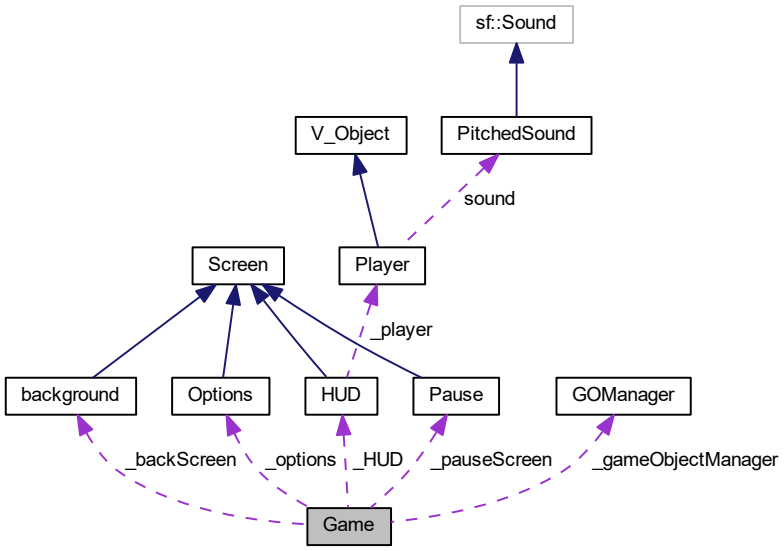
The documentation for this class was generated from the following file:

- background.h

4.4 Game Class Reference

```
#include <Game.h>
```

Collaboration diagram for Game:



Static Public Member Functions

- static void Start ()
- static void initialize ()
- static void Destructing ()

Static Public Attributes

- const static int SCREEN_WIDTH = 640

  width of the screen.
- const static int SCREEN_HEIGHT = 480

  height of the screen.

Private Types

- enum GameState {
  Uninitialized, **ShowingSplash**, **Paused**, **ShowingMenu**,
  Playing, **Exiting**, **Over**, **ShowingOptions** }

Static Private Member Functions

- static bool IsExiting ()
- static void GameLoop ()
- static void ShowPause ()
- static void ShowMenu ()
- static void ShowOver ()
- static void ShowOptions ()

Static Private Attributes

- static GameState **_gameState**
- static sf::RenderWindow _mainWindow

  game window.
- static GOManager **_gameObjectManager**
- static sf::View _mainview

  view for displaying V_Objects objects.
- static sf::View _HUDview

  view for displaying HUD.
- static Pause **_pauseScreen**
- static background **_backScreen**
- static HUD **_HUD**
- static sf::Music _music

  Music.
- static Options **_options**

4.4.1 Detailed Description

The heart of the game. It contains the game loop. Initializes the window and all necessary objects. Calls appropriate functions from other classes.

4.4.2 Member Enumeration Documentation

4.4.2.1 GameState

```
enum Game::GameState  [private]
```

Enum representing the states that game can be in.

4.4.3 Member Function Documentation

4.4.3.1 Destructing()

```
static void Game::Destructing ( )  [static]
```

Deletes player and platforms objects. Function is called when the game is over and main menu is about to be displayed. "Prepares" the game for initialized later.

4.4.3.2 GameLoop()

```
static void Game::GameLoop ( )  [static], [private]
```

Game loop calls functions depending on the current gamestate. For example when gamestate is equal to "Playing" calls all necessary update and draw functions.

4.4.3.3 initialize()

```
static void Game::initialize ( )  [static]
```

Initializes game objects: player and platforms.

4.4.3.4 IsExiting()

```
static bool Game::IsExiting ( )  [static], [private]
```

Checks if the game state is equal to exiting.

Returns

true if yes otherwise false.

4.4.3.5 ShowMenu()

```
static void Game::ShowMenu ( )  [static], [private]
```

Calls function responsible for showing the Main Menu and changes the gamestate based on the user choice in the Menu.

4.4.3.6 ShowOptions()

```
static void Game::ShowOptions ( )  [static], [private]
```

Calls function responsible for showing the Options Menu. Changes the gamestate and enable or disable music based on the user choice.

4.4.3.7 ShowOver()

```
static void Game::ShowOver ( )  [static], [private]
```

Calls function responsible for showing the game over screen.

4.4.3.8 ShowPause()

```
static void Game::ShowPause ( )  [static], [private]
```

Calls function responsible for showing the pause screen.

4.4.3.9 Start()

```
static void Game::Start ( )  [static]
```

Creates game window and views. Sets the gamestate and starts the game loop.

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

- Game.h

4.5 GOManager::GameObjectDeallocator Struct Reference

Public Member Functions

- void operator() (const std::pair< std::string, V_Object * > &p) const

4.5.1 Detailed Description

Struct which holds a functor to delete objects.

4.5.2 Member Function Documentation

4.5.2.1 operator()()

```
void GOManager::GameObjectDeallocator::operator() (
            const std::pair< std::string, V_Object * > & p ) const  [inline]
```

Overloaded function operator.

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

- GOManager.h

4.6 GOManager Class Reference

```
#include <GOManager.h>
```

Classes

- struct GameObjectDeallocator

Public Member Functions

- GOManager ()
- ~GOManager ()
- void Add (std::string name, V_Object *Object)
- void Remove (std::string name)
- V_Object * Get (std::string name) const
- void DrawAll (sf::RenderWindow &renderWindow)
- void UpdateAll ()

Private Attributes

- std::map< std::string, V_Object * > _Objects

  collection of std::pair objects containing string and pointer to V_Object.

4.6.1 Detailed Description

Class responsible for managing game objects. Calls update and draw functions for each V_Object in the game.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 GOManager()

```
GOManager::GOManager ( )
```

Constructor.

4.6.2.2 ~GOManager()

```
GOManager::~GOManager ( )
```

Destructor. Calls for_each function with map and GameObjectDeallocator as parameters. For every object, from the objects map, GameObjectDeallocator functor is called. Every V_Object stored in the map is deleted. This prevents memory leaks.

4.6.3 Member Function Documentation

4.6.3.1 Add()

```
void GOManager::Add (
            std::string name,
            V_Object * Object )
```

Adds V_Object to the map. Map stores std::pair of string and pointer to V_Object.

Parameters

name	a string which holds the name of the object.
Object	pointer to V_Object.

4.6.3.2 DrawAll()

```
void GOManager::DrawAll (
            sf::RenderWindow & renderWindow )
```

Calls draw function for every object stored in the map.

Parameters

renderwindow	render window.

4.6.3.3 Get()

```
V_Object* GOManager::Get (
            std::string name ) const
```

Function goes through entire map until it finds the Object with a given name. If the object is found returns pointer to it, otherwise returns NULL.

Parameters

name	name of the V_Object we want to get.

Returns

pointer to V_Object is returned.

4.6.3.4 Remove()

```
void GOManager::Remove (
            std::string name )
```

Function deletes the pointer to the V_Object referred to in results->second.

Parameters

name	a string which holds the name of the object.

4.6.3.5 UpdateAll()

```
void GOManager::UpdateAll ( )
```

Calls update function for every object stored in the map.

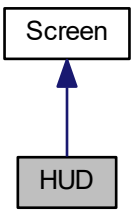
The documentation for this class was generated from the following file:

- GOManager.h

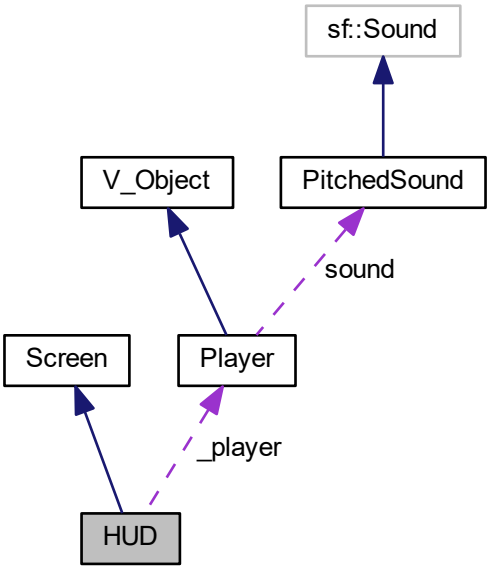
4.7 HUD Class Reference

```
#include <HUD.h>
```

Inheritance diagram for HUD:



Collaboration diagram for HUD:



Public Member Functions

- HUD ()
- ~HUD ()
- void SetPlayer (Player *player)
- void showScore (sf::RenderWindow &window)
- void Show (sf::RenderWindow &window)

Private Attributes

- sf::Font **font**
- sf::Text **text**
- sf::Text **textOver**
- Player * _player

  pointer to a player.

Additional Inherited Members

4.7.1 Detailed Description

Displays score and the game over screen. Inherits Screen publicly.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 HUD()

```
HUD::HUD ( )
```

Constructor.

4.7.2.2 ~HUD()

```
HUD::~HUD ( )
```

Destructor.

4.7.3 Member Function Documentation

4.7.3.1 SetPlayer()

```
void HUD::SetPlayer (
            Player * player )
```

Sets pointer to a player.

Parameters

player	pointer to a player.

4.7.3.2 Show()

```
void HUD::Show (
            sf::RenderWindow & window )  [virtual]
```

Calls screen show function and displays the sprite in the window.

Parameters

window	render window.

Reimplemented from Screen.

4.7.3.3 showScore()

```
void HUD::showScore (
            sf::RenderWindow & window )
```

Displays players score in the window.

Parameters

window	render window.

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

- HUD.h

4.8 MainMenu Class Reference

`#include <MainMenu.h>`

Classes

- struct MenuItem

Public Types

- enum MenuResult { **Nothing**, **Exit**, **Play**, **Options** }

Public Member Functions

- MenuResult Show (sf::RenderWindow &window)

Private Member Functions

- MenuResult HandleClick (int x, int y)
- MenuResult GetMenuResponse (sf::RenderWindow &window)
- MenuResult HandleKey (int x)
- void UpdateButton (int x, sf::RenderWindow &window)

Private Attributes

- sf::Texture **image**
- sf::Texture **play_text**
- sf::Texture **exit_text**
- sf::Texture **options_text**
- sf::Sprite **sprite**
- sf::Sprite **play**
- sf::Sprite **exit**
- sf::Sprite **options**
- std::list< MenuItem > _menuItems

  list of type MenuItem, stores items that compose the menu.

4.8.1 Detailed Description

Responsible for displaying the menu and changing the game state based on user's input.

4.8.2 Member Enumeration Documentation

4.8.2.1 MenuResult

```
enum MainMenu::MenuResult
```

Represents various possible return values the menu could return.

4.8.3 Member Function Documentation

4.8.3.1 GetMenuResponse()

```
MenuResult MainMenu::GetMenuResponse (
            sf::RenderWindow & window )  [private]
```

Handles user mouse and keyboard input. Calls other functions based on the input.

Parameters

window	render window.

Returns

menu state.

4.8.3.2 HandleClick()

```
MenuResult MainMenu::HandleClick (
            int x,
            int y )  [private]
```

Checks if any button was pressed. If not returns nothing.

Parameters

x	position of the mouse on the x axis.
y	position of the mouse on the y axis.

Returns

menu state.

4.8.3.3 HandleKey()

```
MenuResult MainMenu::HandleKey (
            int x )  [private]
```

Responsible for returning menu state according to selected button.

Parameters

x	currently selected button.

Returns

menu state.

4.8.3.4 Show()

```
MenuResult MainMenu::Show (
            sf::RenderWindow & window )
```

Function loads a main-menu and buttons images and creates a sprites to display them.

Parameters

window	render window.

Returns

the main-menu state from the GetMenuResponse function called with window as a parameter.

4.8.3.5 UpdateButton()

```
void MainMenu::UpdateButton (
            int x,
            sf::RenderWindow & window )  [private]
```

Highlight the selected button.

Parameters

x	currently selected button.
window	render window.

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

- MainMenu.h

4.9 MainMenu::MenuItem Struct Reference

```
#include <MainMenu.h>
```

Public Member Functions

- void operator= (sf::Color c)

Public Attributes

- sf::Rect< int > **rect**
- sf::Sprite * sprite

  pointer to a sprite.
- MenuResult **action**
- int **button**

4.9.1 Detailed Description

Struct that represents the individual menu items in the menu. Allows user to use mouse or arrow keys to choose the button.

4.9.2 Member Function Documentation

4.9.2.1 operator=()

```
void MainMenu::MenuItem::operator= (
            sf::Color c )  [inline]
```

Overloaded = operator for setting color of the sprite.

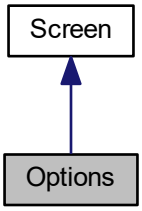
The documentation for this struct was generated from the following file:

- MainMenu.h

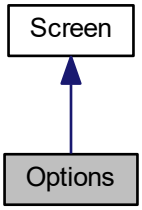
4.10 Options Class Reference

```
#include <Options.h>
```

Inheritance diagram for Options:



Collaboration diagram for Options:



Classes

- struct OptionsItem

Public Types

- enum OptionsResult { **Exit**, **Back** }

Public Member Functions

- Options ()
- ~Options ()
- void Show (sf::RenderWindow &window)
- void UpdateChoice (int x)
- void UpdateOption (int x, int y)
- OptionsResult Option (sf::RenderWindow &window)

Public Attributes

- int **_music**
- int **_sound**

Private Attributes

- sf::Font **font**
- sf::Text **text_sound**
- sf::Text **text_music**
- std::list< OptionsItem > _optionsItems

  list of type OptionsItem, stores items that compose the options menu.

Additional Inherited Members

4.10.1 Detailed Description

A class that shows the options screen. Allows user to disable or enable sounds of the player and the game music. Inherits publicly Screen.

4.10.2 Member Enumeration Documentation

4.10.2.1 OptionsResult

`enum Options::OptionsResult`

Enum representing possible return values the options could return.

4.10.3 Constructor & Destructor Documentation

4.10.3.1 Options()

```
Options::Options ( )
```

Constructor.

4.10.3.2 ~Options()

```
Options::~Options ( )
```

Destructor.

4.10.4 Member Function Documentation

4.10.4.1 Option()

```
OptionsResult Options::Option (
            sf::RenderWindow & window )
```

Handles user input and calls appropriate functions based on user choice.

Parameters

window	render window.

Returns

options state.

4.10.4.2 Show()

```
void Options::Show (
            sf::RenderWindow & window )  [virtual]
```

Calls screen show function and displays the sprite in the window.

Parameters

window	render window.

Reimplemented from Screen.

4.10.4.3 UpdateChoice()

```
void Options::UpdateChoice (
            int x )
```

Highlights the currently selected option.

Parameters

x	currently selected option.

4.10.4.4 UpdateOption()

```
void Options::UpdateOption (
            int x,
            int y )
```

Disables or enables music or sounds depending on selected option and its current value.

Parameters

x	currently selected option.
y	current value of the chosen option, if 1 - enabled if 0 - disabled.

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

- Options.h

4.11 Options::OptionsItem Struct Reference

```
#include <Options.h>
```

Public Member Functions

- void operator= (const sf::Color c)
- void operator= (std::string s)

Public Attributes

- sf::Text * text

  pointer to a text.
- int nb

  number of the option.
- std::string second

  string for storing currently unused string(the opposite of current value of option: on or off).

4.11.1 Detailed Description

Struct representing options menu items.

4.11.2 Member Function Documentation

4.11.2.1 operator=() [1/2]

```
void Options::OptionsItem::operator= (
            const sf::Color c )  [inline]
```

overloaded = operator for changing color of the text.

4.11.2.2 operator=() [2/2]

```
void Options::OptionsItem::operator= (
            std::string s )  [inline]
```

overloaded = operator for swapping the strings of the item.

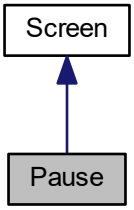
The documentation for this struct was generated from the following file:

- Options.h

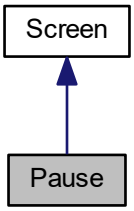
4.12 Pause Class Reference

`#include <Pause.h>`

Inheritance diagram for Pause:



Collaboration diagram for Pause:



Public Member Functions

- Pause ()
- ~Pause ()
- void show (sf::RenderWindow &window)

Additional Inherited Members

4.12.1 Detailed Description

A class that shows the "game paused" screen. Inherits Screen publicly.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Pause()

```
Pause::Pause ( )
```

Constructor.

4.12.2.2 ~Pause()

```
Pause::~Pause ( )
```

Destructor.

4.12.3 Member Function Documentation

4.12.3.1 show()

```
void Pause::show (
            sf::RenderWindow & window )
```

Calls screen show function and displays the sprite in the window.

Parameters

window	render window.

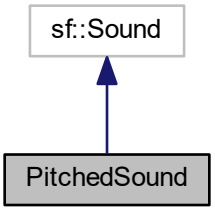
The documentation for this class was generated from the following file:

- Pause.h

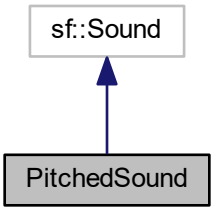
4.13 PitchedSound Class Reference

```
#include <PitchedSound.h>
```

Inheritance diagram for PitchedSound:



Collaboration diagram for PitchedSound:



Public Member Functions

- void playPitched ()

4.13.1 Detailed Description

Responsible for sounds playback at different pitch.

4.13.2 Member Function Documentation

4.13.2.1 playPitched()

```
void PitchedSound::playPitched ( )
```

Plays sound at a random pitch.

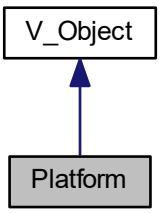
The documentation for this class was generated from the following file:

- PitchedSound.h

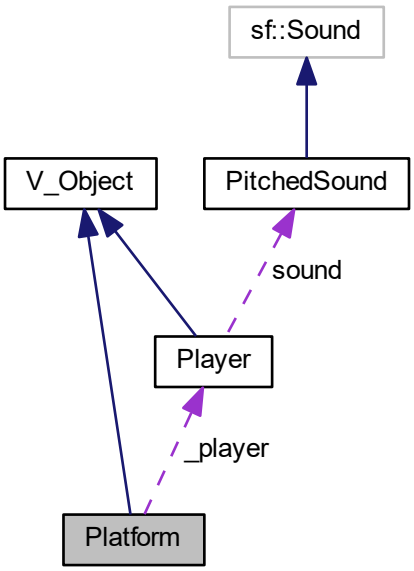
4.14 Platform Class Reference

`#include <Platform.h>`

Inheritance diagram for Platform:



Collaboration diagram for Platform:



Public Member Functions

- Platform (int lvl, Player *player)
- ~Platform ()
- void Update ()
- void Draw (sf::RenderWindow &rw)

- void Color ()
- void Speed ()
- void Collision ()
- bool Collide ()
- void Regenerate ()
- int generatelength (int pos)
- int generatepos ()
- Platform & operator= (Player *p)

Private Attributes

- float **_movespeed**
- int **_distance**
- int **_lvl**
- int **_ground**
- int **_collision**
- float **_gravity**
- Player * _player

  pointer to a Player object

Additional Inherited Members

4.14.1 Detailed Description

Responsible for platforms. Handles collision with the player, moves the entire map and generates platforms endlessly. Because Platform is a visible object in the game, class inherits publicly V_Object.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 Platform()

```
Platform::Platform (
            int lvl,
            Player * player )
```

Platform Constructor.

Parameters

lvl	initialization parameter for _lvl.
player	pointer to Player object.

4.14.2.2 ~Platform()

```
Platform::~Platform ( )
```

Platform destructor.

4.14.3 Member Function Documentation

4.14.3.1 Collide()

```
bool Platform::Collide ( )
```

Checks if player is standing on the platform.

Returns

true if yes false if no.

4.14.3.2 Collision()

```
void Platform::Collision ( )
```

Handles collision with the player.

4.14.3.3 Color()

```
void Platform::Color ( )
```

Sets appropriate color of the platform.

4.14.3.4 Draw()

```
void Platform::Draw (
            sf::RenderWindow & rw )  [virtual]
```

Draws sprite to the render window.

Parameters

rw	render window.

Reimplemented from V_Object.

4.14.3.5 generatelength()

```
int Platform::generatelength (
            int pos )
```

Generates length of the platform based on its position.

Parameters

pos	position of the platform(generated before).

Returns

length of the platform.

4.14.3.6 generatepos()

```
int Platform::generatepos ( )
```

Generates position of the platform on the x axis.

Returns

position of the platform.

4.14.3.7 operator=()

```
Platform& Platform::operator= (
            Player * p )
```

Overloaded operator. Assigns the current platform level to the player level.

4.14.3.8 Regenerate()

```
void Platform::Regenerate ( )
```

When the platform is off screen, the function generates its new position, length and places the platform at the top of the screen.

4.14.3.9 Speed()

```
void Platform::Speed ( )
```

Changes _movespeed based on the level of the platform.

4.14.3.10 Update()

```
void Platform::Update ( )  [virtual]
```

Override V_Object Update() function. Moves platform and calls other functions(responsible for collisions, colors, speed etc.).

Reimplemented from V_Object.

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

- Platform.h

4.15 platform_colors Class Reference

```
#include <platform_colors.h>
```

Static Public Attributes

- static sf::Color **plat_50**
- static sf::Color **plat_100**
- static sf::Color **plat_150**
- static sf::Color **plat_200**
- static sf::Color **plat_250**
- static sf::Color **plat_300**
- static sf::Color **plat_350**
- static sf::Color **plat_400**
- static sf::Color **plat_450**

4.15.1 Detailed Description

Class similar to Animation class. Holds colors constructed from RGBA components for each platform type. Every 50 levels the type of the platform and its color changes. Every member of the class is static, so it can be accessed without creating an object.

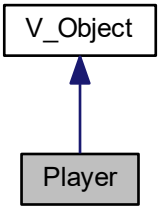
The documentation for this class was generated from the following file:

- platform_colors.h

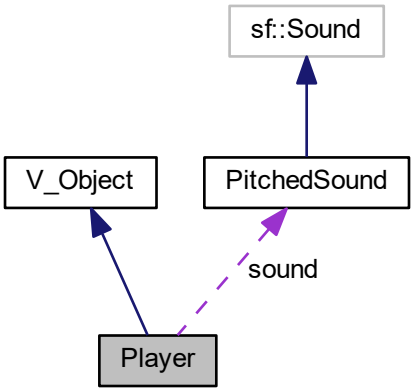
4.16 Player Class Reference

`#include <Player.h>`

Inheritance diagram for Player:



Collaboration diagram for Player:



Public Member Functions

- Player (int a)
- ~Player ()
- void Update ()
- void Draw (sf::RenderWindow &rw)
- void move (float x, float y)
- float **GetVelocityX** () const
- float **GetVelocityY** () const
- int **GetJumpPower** () const
- void **SetJumpPower** (int x)

- int **GetGround** () const
- void **SetGround** (int x)
- int **GetLvl** () const
- void **SetLvl** (int x)
- int **GetOnEdge** () const
- void **SetOnEdge** (int x)
- bool **GetOnGround** () const
- void **SetOnGround** (bool x)
- bool **GetOnPlatform** () const
- void **SetOnPlatform** (bool x)
- bool **GetStarting** () const
- void **SetStarting** (bool x)
- int **GetMaxPlat** () const
- void **SetMaxPlat** (int x)
- float **GetGravity** () const
- void **SetGravity** (float x)

Private Member Functions

- void animation_sound ()
- void checkMove ()
- void boundaries ()

Private Attributes

- sf::SoundBuffer soundHop

  sound buffer for Hop sound.
- sf::SoundBuffer soundWhoopie

  sound buffer for Whoopie sound.
- sf::SoundBuffer soundPrrah

  sound buffer for Prrah sound.
- sf::SoundBuffer soundSkrrt

  sound buffer for Skrrt sound.
- sf::SoundBuffer soundEssa

  sound buffer for Essa sound.
- float **_velocityX**
- float **_velocityY**
- float **_maxvelocityX**
- float **_maxvelocityY**
- int **_wallHit**
- float **_animTime**
- int **_jumpPower**
- int **_ground**
- int **_lvl**
- int **_onEdge**
- int **_wallCounter**
- bool **_onGround**
- bool **_onPlatform**
- bool **_starting**
- int **_maxplat**
- float **_gravity**
- int **_sound**
- bool **_out**

Static Private Attributes

- static PitchedSound sound

  PitchedSound object for playing sounds.

Additional Inherited Members

4.16.1 Detailed Description

Handles Player collision with the walls, user input, movement, physics, animations and sounds. Because Player is a visible object in the game, class inherits publicly V_Object.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Player()

```
Player::Player (
            int a )
```

Player Constructor.

Parameters

a	initialization parameter for _sound.

4.16.2.2 ~Player()

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

Player destructor.

4.16.3 Member Function Documentation

4.16.3.1 animation_sound()

```
void Player::animation_sound ( )  [private]
```

Updates animations and sounds of the player.

4.16.3.2 boundaries()

```
void Player::boundaries ( )  [private]
```

Handles collision with walls.

4.16.3.3 checkMove()

```
void Player::checkMove ( )  [private]
```

Handles movement based on user's input.

4.16.3.4 Draw()

```
void Player::Draw (
            sf::RenderWindow & rw )  [virtual]
```

Draws sprite to the render window.

Parameters

rw	render window.

Reimplemented from V_Object.

4.16.3.5 move()

```
void Player::move (
            float x,
            float y )
```

Moves player by a given amount of pixels on x and y axis.

Parameters

x	offset on the x axis.
y	offset on the y axis.

4.16.3.6 Update()

```
void Player::Update ( )  [virtual]
```

Override V_Object Update() function. Checks velocities and calls other functions(responsible for movement and animations etc.).

Reimplemented from V_Object.

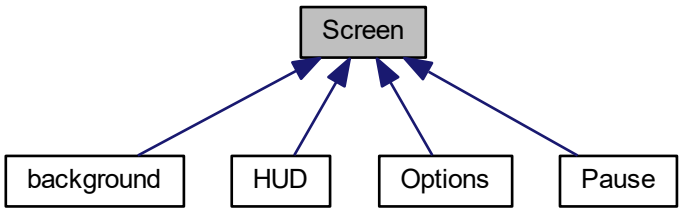
The documentation for this class was generated from the following file:

- Player.h

4.17 Screen Class Reference

`#include <Screen.h>`

Inheritance diagram for Screen:



Public Member Functions

- Screen ()
- virtual ~Screen ()
- virtual void Load (std::string filename)
- virtual void Show (sf::RenderWindow &window)

Protected Member Functions

- sf::Sprite & **GetSprite** ()

Private Attributes

- sf::Sprite **_sprite**
- sf::Texture **_image**
- std::string **_filename**

4.17.1 Detailed Description

Abstract class which loads and displays screens used during the game. Each object displaying screen(except MainMenu) inherits from this class.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 Screen()

```
Screen::Screen ( )  [inline]
```

Screen constructor.

4.17.2.2 ~Screen()

```
virtual Screen::~Screen ( )  [inline], [virtual]
```

Virtual destructor.

4.17.3 Member Function Documentation

4.17.3.1 Load()

```
virtual void Screen::Load (
            std::string filename )  [virtual]
```

If possible loads the image from the file to the sprite.

Parameters

filename	name of the file.

4.17.3.2 Show()

```
virtual void Screen::Show (
            sf::RenderWindow & window )  [virtual]
```

Draws sprite to the render window.

Parameters

window	render window.

Reimplemented in HUD, and Options.

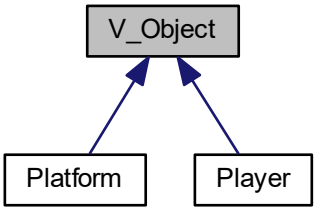
The documentation for this class was generated from the following file:

- Screen.h

4.18 V_Object Class Reference

`#include <V_Object.h>`

Inheritance diagram for V_Object:



Public Member Functions

- V_Object ()
- virtual ~V_Object ()
- virtual void Load (std::string filename)
- virtual void Draw (sf::RenderWindow &window)
- virtual void Update ()
- virtual void SetPosition (float x, float y)
- virtual sf::Vector2f **Getposition** () const
- virtual bool **IsLoaded** () const
- virtual void **SetOut** (bool out)
- virtual bool **IsOut** () const
- virtual float **GetWidth** () const
- virtual float **GetHeight** () const
- virtual sf::Rect< float > **GetBoundingRect** () const

Protected Member Functions

- sf::Sprite & **GetSprite** ()

Private Attributes

- sf::Sprite **_sprite**
- sf::Texture **_image**
- std::string **_filename**
- bool **_isLoaded**
- bool **_out**

4.18.1 Detailed Description

Abstract class for objects visible on the screen. Handles loading images, drawing, updating and changing position of the object. Each displayed object inherits from this class.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 V_Object()

```
V_Object::V_Object ( )
```

Constructor.

4.18.2.2 ~V_Object()

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

Virtual destructor.

4.18.3 Member Function Documentation

4.18.3.1 Draw()

```
virtual void V_Object::Draw (
            sf::RenderWindow & window )  [virtual]
```

If image was loaded draws sprite to the render window.

Parameters

window	render window.

Reimplemented in Platform, and Player.

4.18.3.2 Load()

```
virtual void V_Object::Load (
            std::string filename )  [virtual]
```

If possible loads the image from the file to the sprite.

Parameters

filename	name of the file.

4.18.3.3 SetPosition()

```
virtual void V_Object::SetPosition (
            float x,
            float y )  [virtual]
```

Sets position of the sprite.

Parameters

x	position on the x axis.
y	position on the y axis.

4.18.3.4 Update()

```
virtual void V_Object::Update ( )  [virtual]
```

Calls update function.

Reimplemented in Platform, and Player.

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

- V_Object.h

Chapter 5

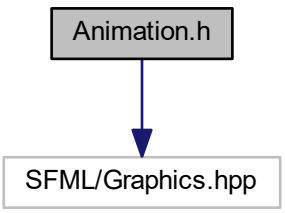
File Documentation

5.1 Animation.h File Reference

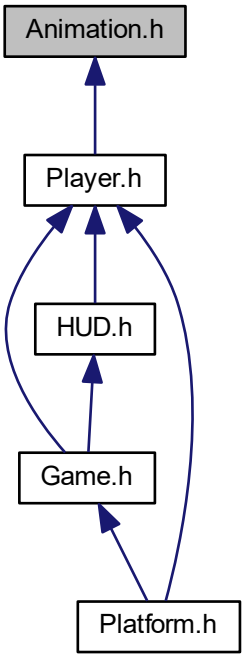
Animation class.

```
#include "SFML/Graphics.hpp"
```

Include dependency graph for Animation.h:



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



Classes

- class Animation

5.1.1 Detailed Description

Animation class.

Author

Lukasz Kwiecien

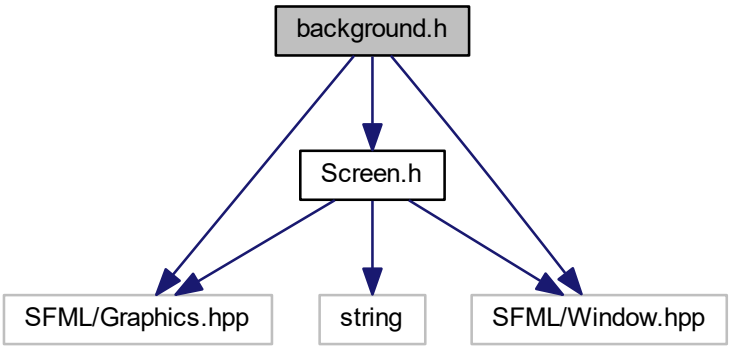
5.2 background.h File Reference

Background screen class.

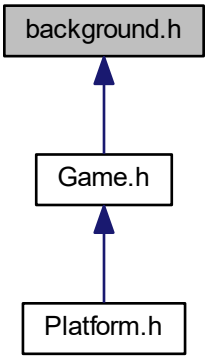
```
#include "SFML/Graphics.hpp"
#include "SFML/Window.hpp"
```

```
#include "Screen.h"
```

Include dependency graph for background.h:



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



Classes

- class background

5.2.1 Detailed Description

Background screen class.

Author

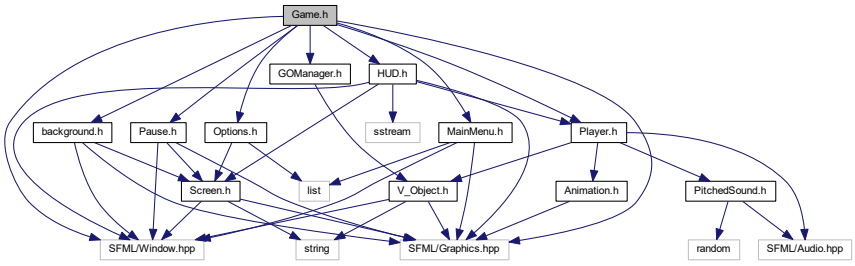
Lukasz Kwiecien

5.3 Game.h File Reference

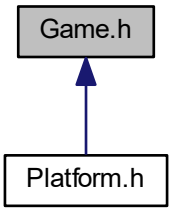
Main Game class.

```
#include "SFML/Window.hpp"
#include "SFML/Graphics.hpp"
#include "Player.h"
#include "GOManager.h"
#include "Pause.h"
#include "background.h"
#include "HUD.h"
#include "Options.h"
#include "MainMenu.h"
```

Include dependency graph for Game.h:



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



Classes

- class Game

5.3.1 Detailed Description

Main Game class.

Author

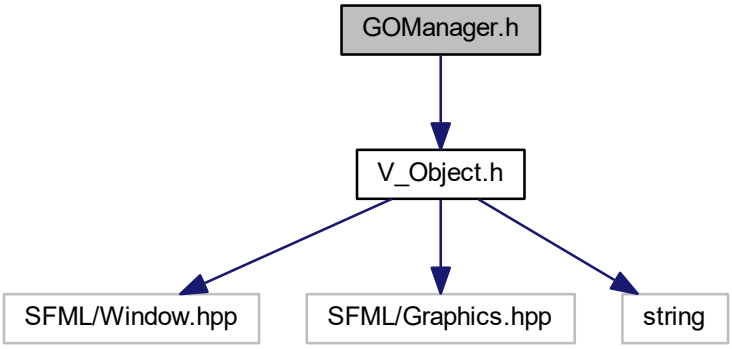
Lukasz Kwiecien

5.4 GOManager.h File Reference

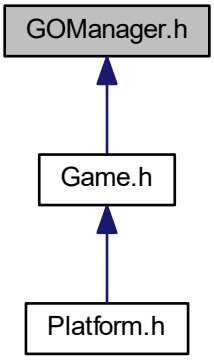
Object Manager.

```
#include "V_Object.h"
```

Include dependency graph for GOManager.h:



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



Classes

- class GOManager
- struct GOManager::GameObjectDeallocator

5.4.1 Detailed Description

Object Manager.

Author

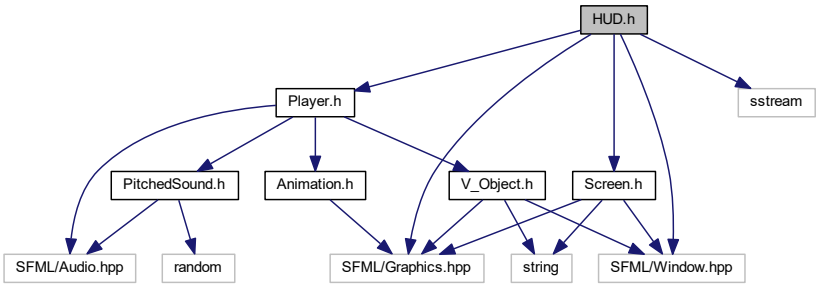
Lukasz Kwiecien

5.5 HUD.h File Reference

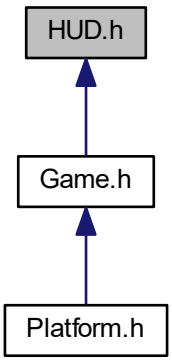
HUD class.

```
#include "SFML/Graphics.hpp"
#include "SFML/Window.hpp"
#include "Player.h"
#include "Screen.h"
#include <sstream>
```

Include dependency graph for HUD.h:



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



Classes

- class HUD

5.5.1 Detailed Description

HUD class.

Author

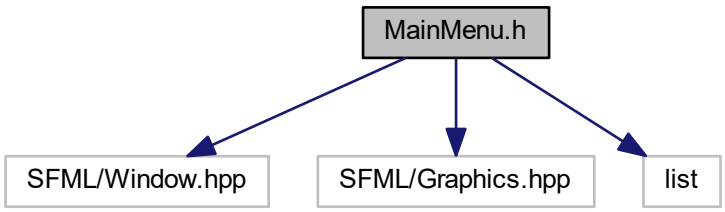
Lukasz Kwiecien

5.6 MainMenu.h File Reference

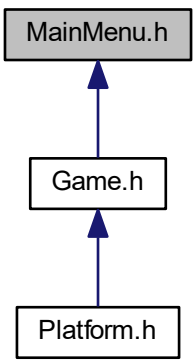
Main Menu class.

```
#include "SFML/Window.hpp"
#include "SFML/Graphics.hpp"
#include <list>
```

Include dependency graph for MainMenu.h:



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



Classes

- class MainMenu
- struct MainMenu::MenuItem

5.6.1 Detailed Description

Main Menu class.

Author

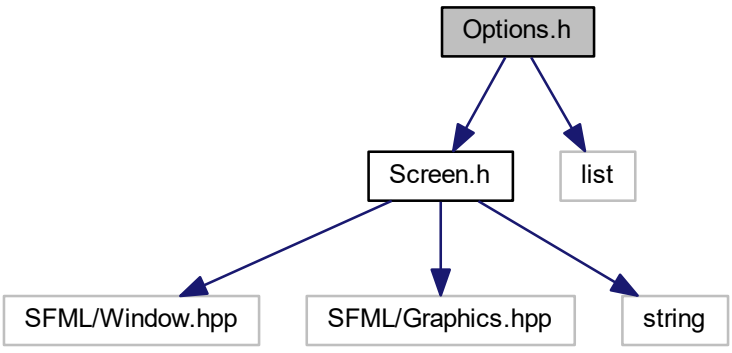
Lukasz Kwiecien

5.7 Options.h File Reference

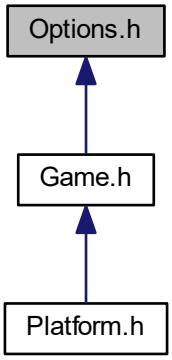
Options class.

```
#include "Screen.h"
#include <list>
```

Include dependency graph for Options.h:



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



Classes

- class Options
- struct Options::OptionsItem

5.7.1 Detailed Description

Options class.

Author

Lukasz Kwiecien

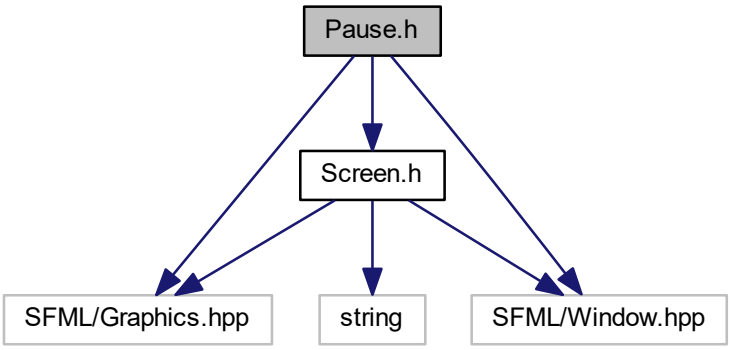
5.8 Pause.h File Reference

Pause screen class.

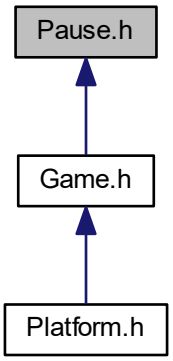
```
#include "SFML/Graphics.hpp"
#include "SFML/Window.hpp"
```

```
#include "Screen.h"
```

Include dependency graph for Pause.h:



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



Classes

- class Pause

5.8.1 Detailed Description

Pause screen class.

Author

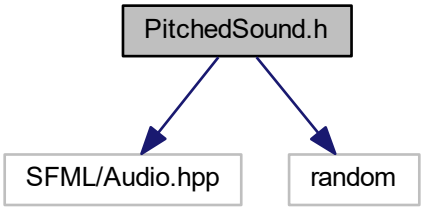
Lukasz Kwiecien

5.9 PitchedSound.h File Reference

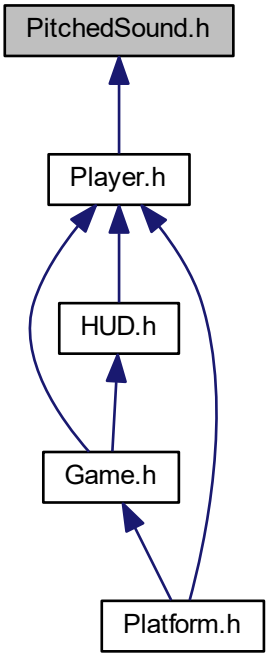
Sounds class.

```
#include <SFML/Audio.hpp>
#include <random>
```

Include dependency graph for PitchedSound.h:



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



Classes

- class PitchedSound

5.9.1 Detailed Description

Sounds class.

Author

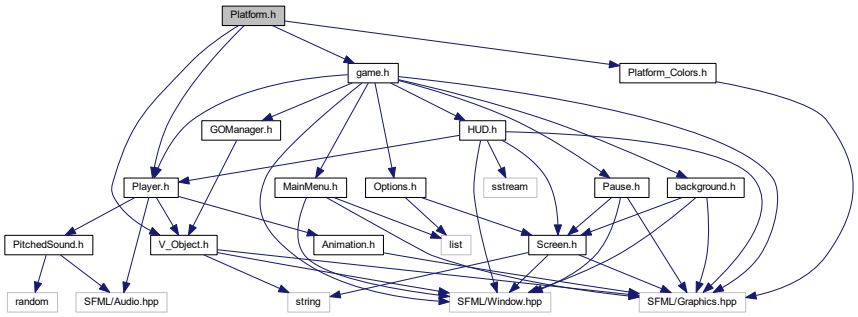
Lukasz Kwiecien

5.10 Platform.h File Reference

Platform class.

```
#include "V_Object.h"
#include "Platform_Colors.h"
#include "Player.h"
#include "game.h"
```

Include dependency graph for Platform.h:



Classes

- class Platform

5.10.1 Detailed Description

Platform class.

Author

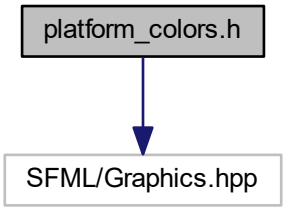
Lukasz Kwiecien

5.11 platform_colors.h File Reference

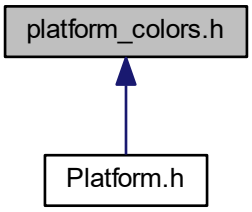
Platform Types class.

```
#include "SFML/Graphics.hpp"
```

Include dependency graph for platform_colors.h:



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



Classes

- class platform_colors

5.11.1 Detailed Description

Platform Types class.

Author

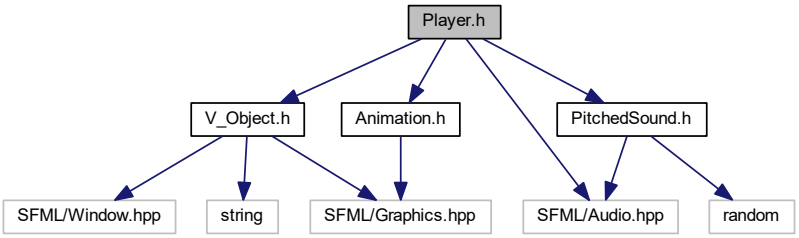
Lukasz Kwiecien

5.12 Player.h File Reference

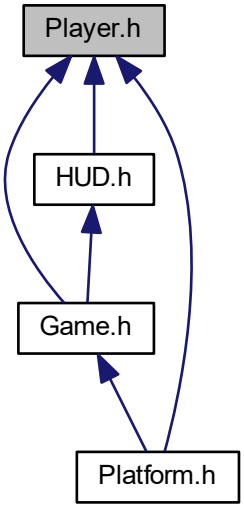
Player class.

```
#include "V_Object.h"
#include "Animation.h"
#include "SFML/Audio.hpp"
#include "PitchedSound.h"
```

Include dependency graph for Player.h:



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



Classes

- class Player

5.12.1 Detailed Description

Player class.

Author

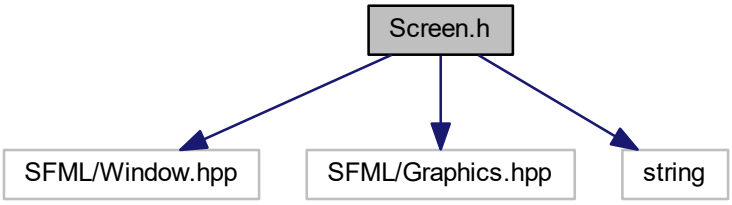
Lukasz Kwiecien

5.13 Screen.h File Reference

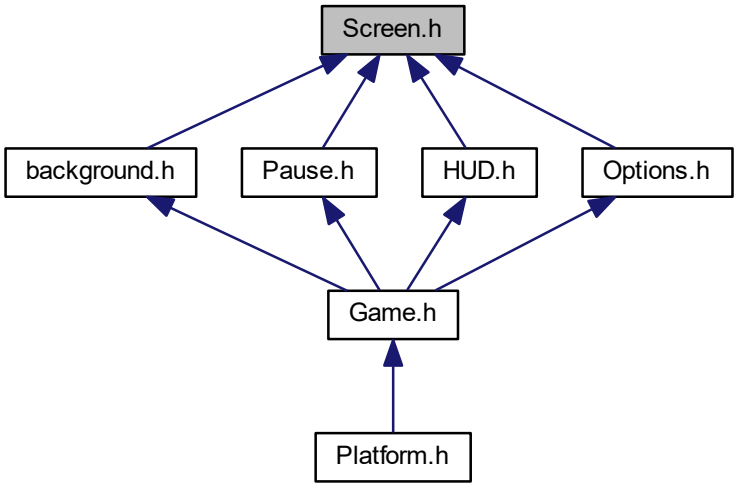
Screen class.

```
#include "SFML/Window.hpp"
#include "SFML/Graphics.hpp"
#include <string>
```

Include dependency graph for Screen.h:



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



Classes

- class Screen

5.13.1 Detailed Description

Screen class.

Author

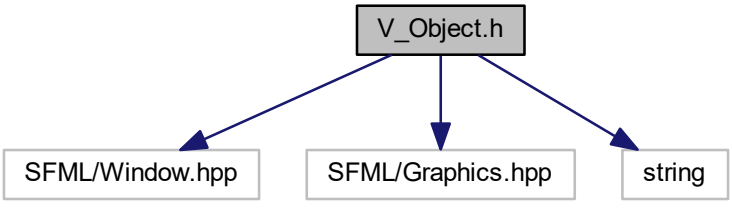
Lukasz Kwiecien

5.14 V_Object.h File Reference

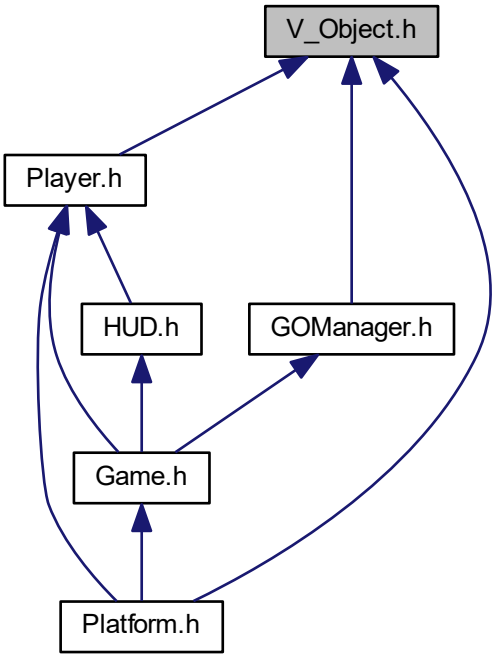
Visible Object Class.

```
#include "SFML/Window.hpp"
#include "SFML/Graphics.hpp"
#include <string>
```

Include dependency graph for V_Object.h:



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



Classes

- class V_Object

5.14.1 Detailed Description

Visible Object Class.

Author

Lukasz Kwiecien

Index