

Pictorial Tetris

1.0.0

Wygenerowano przez Doxygen 1.8.11

Spis treści

1	Indeks hierarchiczny	1
1.1	Hierarchia klas	1
2	Indeks klas	3
2.1	Lista klas	3
3	Dokumentacja klas	5
3.1	Dokumentacja klasy Button	5
3.1.1	Opis szczegółowy	6
3.1.2	Dokumentacja konstruktora i destruktora	6
3.1.2.1	Button(float width, float height, const sf::String &text="","", unsigned int size=TEXT_SIZE)	6
3.1.3	Dokumentacja funkcji składowych	7
3.1.3.1	draw(sf::RenderTarget &target, sf::RenderStates states) const override	7
3.1.3.2	select()	7
3.1.3.3	setPosition(float x, float y)	7
3.1.3.4	setString(const sf::String &string)	7
3.1.3.5	setText(sf::Text &text)	7
3.1.3.6	unselect()	8
3.2	Dokumentacja klasy Config	8
3.2.1	Opis szczegółowy	8
3.2.2	Dokumentacja składowych wyliczanych	9
3.2.2.1	LEVEL	9
3.2.3	Dokumentacja funkcji składowych	9
3.2.3.1	getBlocks() const	9

3.2.3.2	<code>getVelocity() const</code>	9
3.2.3.3	<code>setBlocks(unsigned int x)</code>	9
3.3	Dokumentacja klasy <code>ConfigState</code>	9
3.3.1	Opis szczegółowy	10
3.3.2	Dokumentacja funkcji składowych	11
3.3.2.1	<code>handleInput(sf::Event &event) override</code>	11
3.3.2.2	<code>update(float dt) override</code>	12
3.4	Dokumentacja klasy <code>Game</code>	12
3.4.1	Opis szczegółowy	13
3.4.2	Dokumentacja funkcji składowych	13
3.4.2.1	<code>getConfig()</code>	13
3.4.2.2	<code>getGSM()</code>	13
3.4.2.3	<code>getInstance()</code>	13
3.4.2.4	<code>getView()</code>	13
3.5	Dokumentacja klasy <code>GameOverState</code>	14
3.5.1	Opis szczegółowy	15
3.5.2	Dokumentacja funkcji składowych	15
3.5.2.1	<code>handleInput(sf::Event &event) override</code>	15
3.5.2.2	<code>update(float dt) override</code>	15
3.6	Dokumentacja klasy <code>GameStateManager</code>	15
3.6.1	Opis szczegółowy	16
3.6.2	Dokumentacja funkcji składowych	16
3.6.2.1	<code>handleInput(sf::Event &event)</code>	16
3.6.2.2	<code>push(std::unique_ptr< State > state)</code>	16
3.6.2.3	<code>set(std::unique_ptr< State > state)</code>	16
3.6.2.4	<code>update(float dt)</code>	16
3.7	Dokumentacja klasy <code>HelpState</code>	17
3.7.1	Opis szczegółowy	18
3.7.2	Dokumentacja funkcji składowych	18
3.7.2.1	<code>handleInput(sf::Event &event) override</code>	18

3.7.2.2	<code>update(float dt) override</code>	18
3.8	Dokumentacja klasy <code>MenuState</code>	19
3.8.1	Opis szczegółowy	20
3.8.2	Dokumentacja funkcji składowych	20
3.8.2.1	<code>handleInput(sf::Event &event) override</code>	20
3.8.2.2	<code>update(float dt) override</code>	20
3.9	Dokumentacja klasy <code>MultiOptionField</code>	20
3.9.1	Opis szczegółowy	22
3.9.2	Dokumentacja konstruktora i destruktora	22
3.9.2.1	<code>MultiOptionField(float width, float height, const sf::String &text="", unsigned int size=30, float percent=0.5)</code>	22
3.9.3	Dokumentacja funkcji składowych	22
3.9.3.1	<code>addOption(const sf::String &option)</code>	22
3.9.3.2	<code>draw(sf::RenderTarget &target, sf::RenderStates states) const</code>	22
3.9.3.3	<code>selectNext()</code>	23
3.9.3.4	<code>selectPrevious()</code>	23
3.9.3.5	<code>setCurrent(unsigned int idx)</code>	23
3.9.3.6	<code>setOptionString(unsigned int idx, const sf::String &s)</code>	23
3.9.3.7	<code>setPosition(float x, float y)</code>	23
3.10	Dokumentacja klasy <code>PauseState</code>	24
3.10.1	Opis szczegółowy	25
3.10.2	Dokumentacja funkcji składowych	25
3.10.2.1	<code>handleInput(sf::Event &event) override</code>	25
3.10.2.2	<code>update(float dt) override</code>	25
3.11	Dokumentacja klasy <code>PlayState</code>	25
3.11.1	Opis szczegółowy	26
3.11.2	Dokumentacja funkcji składowych	27
3.11.2.1	<code>handleInput(sf::Event &event) override</code>	27
3.11.2.2	<code>update(float dt) override</code>	28
3.12	Dokumentacja klasy <code>StartState</code>	28
3.12.1	Dokumentacja funkcji składowych	29

3.12.1.1	handleInput(sf::Event &event) override	29
3.12.1.2	update(float dt) override	29
3.13	Dokumentacja klasy State	30
3.13.1	Opis szczegółowy	30
3.13.2	Dokumentacja funkcji składowych	30
3.13.2.1	handleInput(sf::Event &event)=0	30
3.13.2.2	update(float dt)=0	31
3.13.3	Dokumentacja atrybutów składowych	31
3.13.3.1	updated	31
3.14	Dokumentacja klasy View	31
3.14.1	Opis szczegółowy	32
3.14.2	Dokumentacja funkcji składowych	32
3.14.2.1	enableImg(unsigned int idx)	32
3.14.2.2	getBackgroundColor() const	32
3.14.2.3	getFont()	33
3.14.2.4	getHeight() const	33
3.14.2.5	getTexture()	33
3.14.2.6	getWidth() const	33
3.14.2.7	getWindow()	33
3.14.2.8	loadImg2Texture(unsigned int idx)	33
3.14.2.9	resize(sf::Texture &texture, float width, float height)	34
3.14.3	Dokumentacja atrybutów składowych	34
3.14.3.1	IMAGES_COUNT	34
3.14.3.2	time	34
	Indeks	35

Rozdział 1

Indeks hierarchiczny

1.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

Config	8
Game	12
GameStateManager	15
RectangleShape	
Button	5
MultiOptionField	20
State	30
ConfigState	9
GameOverState	14
HelpState	17
MenuState	19
PauseState	24
PlayState	25
StartState	28
View	31

Rozdział 2

Indeks klas

2.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Button	Reprezentuje przycisk wykorzystywany w menu gry	5
Config	Reprezentuje konfigurację i ustawienia gry	8
ConfigState	Reprezentuje stan bycia w konfiguracji gry	9
Game	Reprezentuje gre	12
GameOverState	Reprezentuje stan końca gry	14
GameStateManager	Reprezentuje kontroler stanów aplikacji	15
HelpState	Reprezentuje stan bycia w pomocy	17
MenuState	Reprezentuje stan bycia w menu głównym	19
MultiOptionField	Reprezentuje przycisk z wyborem opcji	20
PauseState	Reprezentuje stan pauzy	24
PlayState	Reprezentuje stan rozpoczęcia gry	25
StartState	28
State	Reprezentuje abstrakcyjny stan	30
View	Reprezentuje widok	31

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja klasy Button

Reprezentuje przycisk wykorzystywany w menu gry.

```
#include <Button.h>
```

Diagram dziedziczenia dla Button

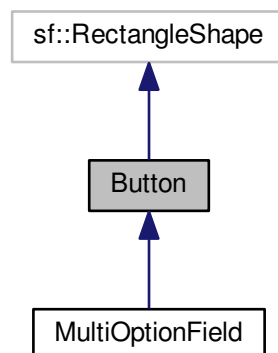
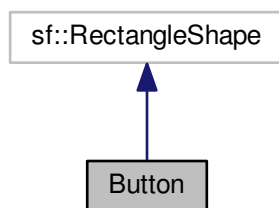


Diagram współpracy dla Button:



Metody publiczne

- **Button** (float width, float height, const sf::String &text="", unsigned int size=TEXT_SIZE)
Konstruuje Button o podanych wymiarach i zawierający określony tekst.
- void **draw** (sf::RenderTarget &target, sf::RenderStates states) const override
Renderuje przycisk.
- void **select** ()
Ustawia przycisk jako zaznaczony w danym momencie.
- void **unselect** ()
Ustawia przycisk jako nie zaznaczony w danym momencie.
- void **setText** (sf::Text &text)
Ustawia obiekt Text widoczny na przycisku.
- void **setString** (const sf::String &string)
Ustawia tekst w zawartym obiekcie klasy Text.
- void **setPosition** (float x, float y)
Zmienia położenie przycisku na ekranie.

3.1.1 Opis szczegółowy

Reprezentuje przycisk wykorzystywany w menu gry.

3.1.2 Dokumentacja konstruktora i destruktora

3.1.2.1 Button::Button (float width, float height, const sf::String & text = " ", unsigned int size = TEXT_SIZE)

Konstruuje Button o podanych wymiarach i zawierający określony tekst.

Parametry

<i>width</i>	Szerokość przycisku w pikselach.
<i>height</i>	Wysokość przycisku w pikselach.
<i>text</i>	Tekst wyświetlany na przycisku.
<i>size</i>	Rozmiar tekstu.

3.1.3 Dokumentacja funkcji składowych

3.1.3.1 void Button::draw (sf::RenderTarget & *target*, sf::RenderStates *states*) const [override]

Renderuje przycisk.

Parametry

<i>target</i>	RenderTarget, do którego przycisk ma być renderowany.
<i>states</i>	Stany renderowania.

3.1.3.2 void Button::select ()

Ustawia przycisk jako zaznaczony w danym momencie.

Zaznaczony przycisk ma inny kolor tła jak i tła tekstu.

3.1.3.3 void Button::setPosition (float *x*, float *y*)

Zmienia położenie przycisku na ekranie.

Parametry

<i>x</i>	Współrzędna na osi OX, w pikselach.
<i>y</i>	Współrzędna na osi OY, w pikselach.

3.1.3.4 void Button::setString (const sf::String & *string*)

Ustawia tekst w zawartym obiekcie klasy Text.

Rozmiar czcionki jest automatycznie dopasowywany do rozmiarów przycisku, jeżeli tekst by wystawał poza przycisk.

Parametry

<i>string</i>	Tekst do ustawienia.
---------------	----------------------

3.1.3.5 void Button::setText (sf::Text & *text*)

Ustawia obiekt Text widoczny na przycisku.

Parametry

<i>text</i>	Text do ustawienia.
-------------	---------------------

3.1.3.6 void Button::unselect ()

Ustawia przycisk jako nie zaznaczony w danym momencie.

Odznaczony przycisk ma domyślny kolor tła jak i tła tekstu.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Button.h

3.2 Dokumentacja klasy Config

Reprezentuje konfigurację i ustawienia gry.

```
#include <Config.h>
```

Typy publiczne

- enum [LEVEL](#) { **EASY**, **NORMAL**, **HARD** }

Metody publiczne

- void [setBlocks](#) (unsigned int x)
Ustawia ilość klocków na bok.
- unsigned int [getBlocks](#) () const
Zwraca ilość klocków na bok.
- double [getVelocity](#) () const
Zwraca szybkość opadania klocków.

Atrybuty publiczne

- [LEVEL level](#) = NORMAL
Przechowuje aktualny poziom trudności gry.

Statyczne atrybuty publiczne

- static const int [MIN_BLOCKS](#) = 2
Przechowuje minimalną możliwą ilość klocków na bok.
- static const int [MAX_BLOCKS](#) = 10
Przechowuje maksymalną możliwą ilość klocków na bok.

3.2.1 Opis szczegółowy

Reprezentuje konfigurację i ustawienia gry.

3.2.2 Dokumentacja składowych wyliczanych

3.2.2.1 enum Config::LEVEL

Reprezentuje poziomy trudności gry.

3.2.3 Dokumentacja funkcji składowych

3.2.3.1 unsigned int Config::getBlocks () const

Zwraca ilość klocków na bok.

Zwraca

Ilość klocków na bok.

3.2.3.2 double Config::getVelocity () const

Zwraca szybkość opadania klocków.

Zwraca

Szybkość opadania klocków.

3.2.3.3 void Config::setBlocks (unsigned int x)

Ustawia ilość klocków na bok.

Parametry

x	Ilość klocków na bok.
---	-----------------------

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Config.h

3.3 Dokumentacja klasy ConfigState

Reprezentuje stan bycia w konfiguracji gry.

```
#include <ConfigState.h>
```

Diagram dziedziczenia dla ConfigState

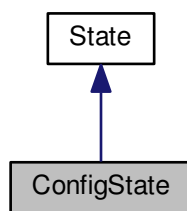
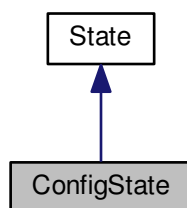


Diagram współpracy dla ConfigState:



Metody publiczne

- `ConfigState ()`
Konstruuje `ConfigState`.
- `void handleInput (sf::Event &event) override`
Przetwarza wejściowe zdarzenia pochodzące od użytkownika.
- `void update (float dt) override`
Aktualizuje aktualny stan.
- `void render () override`
Renderuje stan gry.

Dodatkowe Dziedziczone Składowe

3.3.1 Opis szczegółowy

Reprezentuje stan bycia w konfiguracji gry.

3.3.2 Dokumentacja funkcji składowych

3.3.2.1 void ConfigState::handleInput (sf::Event & *event*) [override],[virtual]

Przetwarza wejściowe zdarzenia pochodzące od użytkownika.

Parametry

<i>event</i>	Zdarzenie pochodzące od użytkownika.
--------------	--------------------------------------

Implementuje [State](#).

3.3.2.2 `void ConfigState::update (float dt) [override],[virtual]`

Aktualizuje aktualny stan.

Parametry

<i>dt</i>	Czas renderowania i przetwarzania poprzedniej klatki gry.
-----------	---

Implementuje [State](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

- ConfigState.h

3.4 Dokumentacja klasy Game

Reprezentuje gre.

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

Metody publiczne

- void [run](#) ()
Uruchamia grę.

Statyczne metody publiczne

- static [Game](#) & [getInstance](#) ()
Zwraca instancję obiektu [Game](#).
- static [View](#) & [getView](#) ()
Zwraca referencję do widoku [View](#).
- static [Config](#) & [getConfig](#) ()
Zwraca referencję do konfiguracji [Config](#).
- static [GameStateManager](#) & [getGSM](#) ()
Zwraca menadżer stanów [GameStateManager](#).

Statyczne atrybuty publiczne

- static const std::string [TITLE](#)
Przechowuje tytuł gry.

3.4.1 Opis szczegółowy

Reprezentuje gre.

Singleton – jednocześnie może występować tylko jedna instancja gry.

3.4.2 Dokumentacja funkcji składowych

3.4.2.1 static Config& Game::getConfig () [static]

Zwraca referencję do konfiguracji [Config](#).

Zwraca

Referencja do [Config](#).

3.4.2.2 static GameStateManager& Game::getGSM () [static]

Zwraca menadżer stanów [GameStateManager](#).

Zwraca

Referencja do [GameStateManager](#).

3.4.2.3 static Game& Game::getInstance () [static]

Zwraca instancję obiektu [Game](#).

Zwraca

Instancja obiektu [Game](#).

3.4.2.4 static View& Game::getView () [static]

Zwraca referencję do widoku [View](#).

Zwraca

Referencja do [View](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Game.h

3.5 Dokumentacja klasy GameState

Reprezentuje stan końca gry.

```
#include <GameOverState.h>
```

Diagram dziedziczenia dla GameState

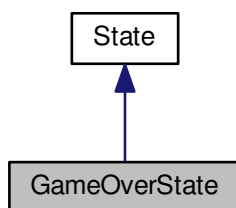
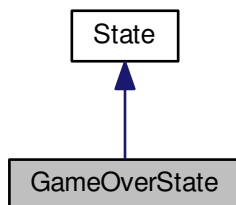


Diagram współpracy dla GameState:



Metody publiczne

- [GameOverState](#) ()
Konstruuje GameState.
- void [handleInput](#) (sf::Event &event) override
Przetwarza wejściowe zdarzenia pochodzące od użytkownika.
- void [update](#) (float dt) override
Aktualizuje aktualny stan.
- void [render](#) () override
Renderuje stan gry.

Dodatkowe Dziedziczone Składowe

3.5.1 Opis szczegółowy

Reprezentuje stan końca gry.

3.5.2 Dokumentacja funkcji składowych

3.5.2.1 void GameState::handleInput (sf::Event & event) [override],[virtual]

Przetwarza wejściowe zdarzenia pochodzące od użytkownika.

Parametry

<i>event</i>	Zdarzenie pochodzące od użytkownika.
--------------	--------------------------------------

Implementuje [State](#).

3.5.2.2 void GameState::update (float dt) [override],[virtual]

Aktualizuje aktualny stan.

Parametry

<i>dt</i>	Czas renderowania i przetwarzania poprzedniej klatki gry.
-----------	---

Implementuje [State](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

- GameState.h

3.6 Dokumentacja klasy GameStateManager

Reprezentuje kontroler stanów aplikacji.

```
#include <GameStateManager.h>
```

Metody publiczne

- void [handleInput](#) (sf::Event &event)
Przetwarza zdarzenie pochodzące od gracza przez aktualny stan aplikacji.
- void [update](#) (float dt)
Aktualizuje aktualny stan aplikacji.

- void `render` ()
Renderuje aktualn stan gry.
- void `push` (std::unique_ptr< `State` > state)
Ustawia aktualny stan aplikacji.
- void `pop` ()
Zamyka ostatni stan aplikacji.
- void `set` (std::unique_ptr< `State` > state)
Zamyka poprzednie stany aplikacji i ustawia nowy.

3.6.1 Opis szczegółowy

Reprezentuje kontroler stanów aplikacji.

3.6.2 Dokumentacja funkcji składowych

3.6.2.1 void GameStateManager::handleInput (sf::Event & event)

Przetwarza zdarzenie pochodzące od gracza przez aktualny stan aplikacji.

Parametry

<i>event</i>	Zdarzenie pochodzące od gracza.
--------------	---------------------------------

3.6.2.2 void GameStateManager::push (std::unique_ptr< `State` > state)

Ustawia aktualny stan aplikacji.

Parametry

<i>state</i>	Nowy stan do ustawienia.
--------------	--------------------------

3.6.2.3 void GameStateManager::set (std::unique_ptr< `State` > state)

Zamyka poprzednie stany aplikacji i ustawia nowy.

Parametry

<i>state</i>	Nowy stan do ustawienia.
--------------	--------------------------

3.6.2.4 void GameStateManager::update (float dt)

Aktualizuje aktualny stan aplikacji.

Parametry

<i>dt</i>	Czas renderowania i przetwarzania poprzedniej klatki gry.
-----------	---

Dokumentacja dla tej klasy została wygenerowana z pliku:

- GameStateManager.h

3.7 Dokumentacja klasy HelpState

Reprezentuje stan bycia w pomocy.

```
#include <HelpState.h>
```

Diagram dziedziczenia dla HelpState

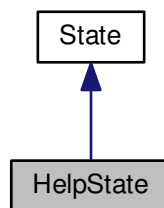
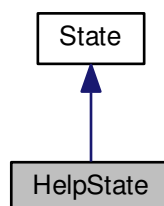


Diagram współpracy dla HelpState:



Metody publiczne

- [HelpState](#) ()=default
Konstruuje [HelpState](#).
- void [handleInput](#) (sf::Event &event) override
Przetwarza wejściowe zdarzenia pochodzące od użytkownika.
- void [update](#) (float dt) override
Aktualizuje aktualny stan.
- void [render](#) () override
Renderuje stan gry.

Dodatkowe Dziedziczone Składowe

3.7.1 Opis szczegółowy

Reprezentuje stan bycia w pomocy.

3.7.2 Dokumentacja funkcji składowych

3.7.2.1 void [HelpState::handleInput](#) (sf::Event & *event*) [override],[virtual]

Przetwarza wejściowe zdarzenia pochodzące od użytkownika.

Parametry

<i>event</i>	Zdarzenie pochodzące od użytkownika.
--------------	--------------------------------------

Implementuje [State](#).

3.7.2.2 void [HelpState::update](#) (float *dt*) [override],[virtual]

Aktualizuje aktualny stan.

Parametry

<i>dt</i>	Czas renderowania i przetwarzania poprzedniej klatki gry.
-----------	---

Implementuje [State](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [HelpState.h](#)

3.8 Dokumentacja klasy MenuState

Reprezentuje stan bycia w menu głównym.

```
#include <MenuState.h>
```

Diagram dziedziczenia dla MenuState

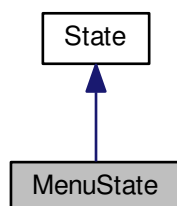
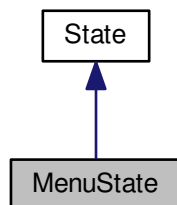


Diagram współpracy dla MenuState:



Metody publiczne

- `MenuState ()`
Konstruuje MainMenu.
- `void handleInput (sf::Event &event) override`
Przetwarza wejściowe zdarzenia pochodzące od użytkownika.
- `void update (float dt) override`
Aktualizuje aktualny stan.
- `void render () override`
Renderuje stan gry.

Dodatkowe Dziedziczone Składowe

3.8.1 Opis szczegółowy

Reprezentuje stan bycia w menu głównym.

3.8.2 Dokumentacja funkcji składowych

3.8.2.1 `void MenuState::handleInput (sf::Event & event) [override],[virtual]`

Przetwarza wejściowe zdarzenia pochodzące od użytkownika.

Parametry

<code>event</code>	Zdarzenie pochodzące od użytkownika.
--------------------	--------------------------------------

Implementuje [State](#).

3.8.2.2 `void MenuState::update (float dt) [override],[virtual]`

Aktualizuje aktualny stan.

Parametry

<code>dt</code>	Czas renderowania i przetwarzania poprzedniej klatki gry.
-----------------	---

Implementuje [State](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

- MenuState.h

3.9 Dokumentacja klasy MultiOptionField

Reprezentuje przycisk z wyborem opcji.

```
#include <MultiOptionField.h>
```

Diagram dziedziczenia dla MultiOptionField

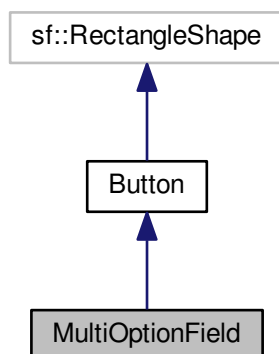
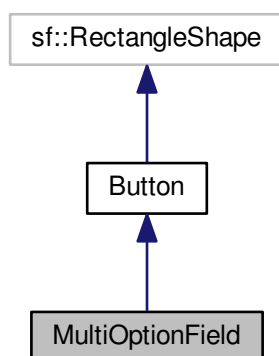


Diagram współpracy dla MultiOptionField:



Metody publiczne

- **MultiOptionField** (float width, float height, const sf::String &text="", unsigned int size=30, float percent=0.5)
Konstruuje MultiOptionField o podanych rozmiarach i pozostałych parametrach.
- void **draw** (sf::RenderTarget &target, sf::RenderStates states) const
Renderuje przycisk z wyborem opcji.
- void **addOption** (const sf::String &option)
Dodaje nową opcję do wyboru.
- void **setCurrent** (unsigned int idx)
Ustawia aktualną opcję.
- void **setPosition** (float x, float y)

- Ustawia pozycję przycisku z wyborem opcji.*
 • void `setOptionString` (unsigned int idx, const sf::String &s)
 - Ustawia tekst opcji o podanym indeksie.*
- unsigned int `selectNext` ()
 - Ustawia następną możliwą opcję.*
- unsigned int `selectPrevious` ()
 - Ustawia poprzednią możliwą opcję.*

3.9.1 Opis szczegółowy

Reprezentuje przycisk z wyborem opcji.

3.9.2 Dokumentacja konstruktora i destruktora

3.9.2.1 `MultiOptionField::MultiOptionField (float width, float height, const sf::String & text = " ", unsigned int size = 30, float percent = 0.5)`

Konstruuje `MultiOptionField` o podanych rozmiarach i pozostałych parametrach.

Parametry

<i>width</i>	Szerokość w pikselach.
<i>height</i>	Wysokość w pikselach. Tekst widoczny na przycisku. Domyślnie pusty tekst. Rozmiar czcionki tekstu. Domyślnie 0.5. Procent zajmowanej szerokości. Domyślnie 0.5 (50%).

3.9.3 Dokumentacja funkcji składowych

3.9.3.1 `void MultiOptionField::addOption (const sf::String & option)`

Dodaje nową opcję do wyboru.

Jeżeli żadna opcja nie istniała, dodaje ją i ustawia ją jako aktualną opcję.

Parametry

<i>option</i>	Opcja do dodania.
---------------	-------------------

3.9.3.2 `void MultiOptionField::draw (sf::RenderTarget & target, sf::RenderStates states) const`

Renderuje przycisk z wyborem opcji.

Parametry

<i>target</i>	RenderTarget, do którego przycisk z wyborem opcji ma być renderowany.
<i>states</i>	Stany renderowania.

3.9.3.3 unsigned int MultiOptionField::selectNext ()

Ustawia następną możliwą opcję.

Zwraca

Indeks nowej opcji.

3.9.3.4 unsigned int MultiOptionField::selectPrevious ()

Ustawia poprzednią możliwą opcję.

Zwraca

Indeks nowej opcji.

3.9.3.5 void MultiOptionField::setCurrent (unsigned int *idx*)

Ustawia aktualną opcję.

Parametry

<i>int</i>	Indeks aktualnej opcji.
------------	-------------------------

3.9.3.6 void MultiOptionField::setOptionString (unsigned int *idx*, const sf::String & *s*)

Ustawia tekst opcji o podanym indeksie.

Parametry

<i>int</i>	Indeks opcji.
<i>s</i>	Tekst do ustawienia.

3.9.3.7 void MultiOptionField::setPosition (float *x*, float *y*)

Ustawia pozycję przycisku z wyborem opcji.

Parametry

<i>x</i>	Współrzędna na osi OX w pikselach.
<i>y</i>	Współrzędna na osi OY w pikselach.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- MultiOptionField.h

3.10 Dokumentacja klasy PauseState

Reprezentuje stan pauzy.

```
#include <PauseState.h>
```

Diagram dziedziczenia dla PauseState

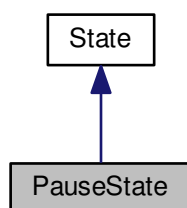
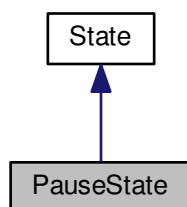


Diagram współpracy dla PauseState:



Metody publiczne

- `PauseState()`=default
Konstruuje `PauseState`.
- void `handleInput` (sf::Event &event) override
Przetwarza wejściowe zdarzenia pochodzące od użytkownika.
- void `update` (float dt) override
Aktualizuje aktualny stan.
- void `render` () override
Renderuje stan gry.

Dodatkowe Dziedziczone Składowe

3.10.1 Opis szczegółowy

Reprezentuje stan pauzy.

3.10.2 Dokumentacja funkcji składowych

3.10.2.1 void PauseState::handleInput (sf::Event & event) [override],[virtual]

Przetwarza wejściowe zdarzenia pochodzące od użytkownika.

Parametry

<i>event</i>	Zdarzenie pochodzące od użytkownika.
--------------	--------------------------------------

Implementuje [State](#).

3.10.2.2 void PauseState::update (float dt) [override],[virtual]

Aktualizuje aktualny stan.

Parametry

<i>dt</i>	Czas renderowania i przetwarzania poprzedniej klatki gry.
-----------	---

Implementuje [State](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

- PauseState.h

3.11 Dokumentacja klasy PlayState

Reprezentuje stan rozpoczęcia gry.

```
#include <PlayState.h>
```

Diagram dziedziczenia dla PlayState

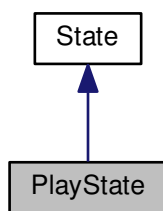
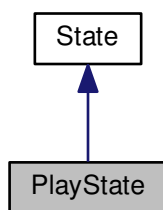


Diagram współpracy dla PlayState:



Metody publiczne

- `PlayState ()`
Konstruuje `PlayState`.
- `void handleInput (sf::Event &event) override`
Przetwarza wejściowe zdarzenia pochodzące od użytkownika.
- `void update (float dt) override`
Aktualizuje aktualny stan.
- `void render () override`
Renderuje stan gry.

Dodatkowe Dziedziczone Składowe

3.11.1 Opis szczegółowy

Reprezentuje stan rozpoczęcia gry.

3.11.2 Dokumentacja funkcji składowych

3.11.2.1 void PlayState::handleInput (sf::Event & *event*) [override],[virtual]

Przetwarza wejściowe zdarzenia pochodzące od użytkownika.

Parametry

<i>event</i>	Zdarzenie pochodzące od użytkownika.
--------------	--------------------------------------

Implementuje [State](#).

3.11.2.2 `void PlayState::update (float dt)` `[override],[virtual]`

Aktualizuje aktualny stan.

Parametry

<i>dt</i>	Czas renderowania i przetwarzania poprzedniej klatki gry.
-----------	---

Implementuje [State](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

- PlayState.h

3.12 Dokumentacja klasy StartState

Diagram dziedziczenia dla StartState

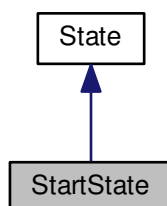
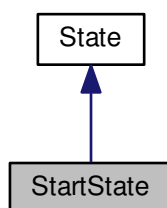


Diagram współpracy dla StartState:



Metody publiczne

- void [handleInput](#) (sf::Event &event) override
Przetwarza wejściowe zdarzenia pochodzące od użytkownika.
- void [update](#) (float dt) override
Aktualizuje aktualny stan.
- void [render](#) () override
Renderuje stan gry.

Dodatkowe Dziedziczone Składowe

3.12.1 Dokumentacja funkcji składowych

3.12.1.1 void StartState::handleInput (sf::Event & *event*) [override],[virtual]

Przetwarza wejściowe zdarzenia pochodzące od użytkownika.

Parametry

<i>event</i>	Zdarzenie pochodzące od użytkownika.
--------------	--------------------------------------

Implementuje [State](#).

3.12.1.2 void StartState::update (float *dt*) [override],[virtual]

Aktualizuje aktualny stan.

Parametry

<i>dt</i>	Czas renderowania i przetwarzania poprzedniej klatki gry.
-----------	---

Implementuje [State](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

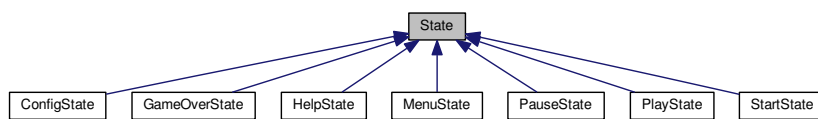
- StartState.h

3.13 Dokumentacja klasy State

Reprezentuje abstrakcyjny stan.

```
#include <State.h>
```

Diagram dziedziczenia dla State



Metody publiczne

- [State](#) ()=default
Konstruuje obiekt abstrakcyjnego stanu. Konstruktor pusty.
- virtual [~State](#) ()
Wirtualny pusty destruktor.
- virtual void [handleInput](#) (sf::Event &event)=0
Przetwarza wejściowe zdarzenia pochodzące od użytkownika. Czysto abstrakcyjna metoda.
- virtual void [update](#) (float dt)=0
Aktualizuje aktualny stan. Czysto abstrakcyjna metoda.
- virtual void [render](#) ()=0
Renderuje stan gry. Czysto abstrakcyjna metoda.

Atrybuty chronione

- bool [updated](#) {true}

3.13.1 Opis szczegółowy

Reprezentuje abstrakcyjny stan.

3.13.2 Dokumentacja funkcji składowych

3.13.2.1 virtual void State::handleInput (sf::Event & event) [pure virtual]

Przetwarza wejściowe zdarzenia pochodzące od użytkownika. Czysto abstrakcyjna metoda.

Parametry

<i>event</i>	Zdarzenie pochodzące od użytkownika.
--------------	--------------------------------------

Implementowany w [PlayState](#), [ConfigState](#), [MenuState](#), [HelpState](#), [PauseState](#), [GameOverState](#) i [StartState](#).

3.13.2.2 `virtual void State::update (float dt) [pure virtual]`

Aktualizuje aktualny stan. Czysto abstrakcyjna metoda.

Parametry

<i>dt</i>	Czas renderowania i przetwarzania poprzedniej klatki gry.
-----------	---

Implementowany w [PlayState](#), [ConfigState](#), [MenuState](#), [HelpState](#), [PauseState](#), [GameOverState](#) i [StartState](#).

3.13.3 Dokumentacja atrybutów składowych

3.13.3.1 `bool State::updated {true} [protected]`

Przechowuje informację o tym, czy stan został zaktualizowany.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [State.h](#)

3.14 Dokumentacja klasy View

Reprezentuje widok.

```
#include <View.h>
```

Metody publiczne

- [View](#) ()
Konstruuje [View](#), tworząc okno gry, ustawiając rozdzielczość i inne niezbędne parametry.
- `sf::RenderWindow & getWindow ()`
Zwraca okno gry.
- `sf::Font & getFont ()`
Zwraca czcionkę wykorzystywaną w grze.
- `sf::Texture & getTexture ()`
Zwraca teksturę zawierającą obrazek gry.
- `const sf::Color & getBackgroundColor () const`
Zwraca kolor tła.
- `void enableImg (unsigned int idx)`

- Ładuje obrazek do tablicy obrazków do podanego indeksu.
- void `loadImg2Texture` (unsigned int idx)
Ustawia aktualny obrazek gry na podstawie indeksu tablicy z obrazkami.
- sf::Sprite `resize` (sf::Texture &texture, float width, float height)
Przeskalowuje obrazek do odpowiednich wymiarów.
- unsigned int `getHeight` () const
Zwraca wysokość okna gry.
- unsigned int `getWidth` () const
Zwraca szerokość okna gry.

Atrybuty publiczne

- float `time` {0}

Statyczne atrybuty publiczne

- static const unsigned int `IMAGES_COUNT` = 4

3.14.1 Opis szczegółowy

Reprezentuje widok.

3.14.2 Dokumentacja funkcji składowych

3.14.2.1 void View::enableImg (unsigned int idx)

Ładuje obrazek do tablicy obrazków do podanego indeksu.

Tablica przyjmuje ściśle określoną maksymalną liczbę obrazków do pamięci. Dodatkowo każdy obrazek musi znajdować się w katalogu "data" i nazywać się "imgX.jpg", gdzie X jest także numerem indeksu w tablicy, gdzie zostanie zapisany.

Parametry

<code>int</code>	Indeks tablicy i końcówka nazwy pliku z obrazkiem.
------------------	--

3.14.2.2 const sf::Color& View::getBackgroundColor () const

Zwraca kolor tła.

Zwraca

Kolor tła.

3.14.2.3 sf::Font& View::getFont ()

Zwraca czcionkę wykorzystywaną w grze.

Zwraca

Czcionka.

3.14.2.4 unsigned int View::getHeight () const

Zwraca wysokość okna gry.

Zwraca

Wysokość okna gry w pikselach.

3.14.2.5 sf::Texture& View::getTexture ()

Zwraca teksturę zawierającą obrazek gry.

Zwraca

Tekstura zawierająca obrazek gry.

3.14.2.6 unsigned int View::getWidth () const

Zwraca szerokość okna gry.

Zwraca

Szerokość okna gry w pikselach.

3.14.2.7 sf::RenderWindow& View::getWindow ()

Zwraca okno gry.

Zwraca

Okno gry.

3.14.2.8 void View::loadImg2Texture (unsigned int *idx*)

Ustawia aktualny obrazek gry na podstawie indeksu tablicy z obrazkami.

Parametry

<i>int</i>	Indeks tablicy.
------------	-----------------

3.14.2.9 `sf::Sprite View::resize (sf::Texture & texture, float width, float height)`

Przeskalowuje obrazek do odpowiednich wymiarów.

Parametry

<i>texture</i>	Tekstura do przeskalowania.
<i>width</i>	Oczekiwana szerokość w pikselach.
<i>height</i>	Oczekiwana wysokość w pikselach.

Zwraca

Obiekt Sprite zawierający przeskalowany obrazek.

3.14.3 Dokumentacja atrybutów składowych

3.14.3.1 `const unsigned int View::IMAGES_COUNT = 4` `[static]`

Przechowuje ilość przechowywanych obrazków w tablicy.

3.14.3.2 `float View::time {0}`

Przechowuje czas gry.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- View.h

Skorowidz

- addOption
 - MultiOptionField, [22](#)
- Button, [5](#)
 - Button, [6](#)
 - draw, [7](#)
 - select, [7](#)
 - setPosition, [7](#)
 - setString, [7](#)
 - setText, [7](#)
 - unselect, [7](#)
- Config, [8](#)
 - getBlocks, [9](#)
 - getVelocity, [9](#)
 - LEVEL, [9](#)
 - setBlocks, [9](#)
- ConfigState, [9](#)
 - handleInput, [11](#)
 - update, [12](#)
- draw
 - Button, [7](#)
 - MultiOptionField, [22](#)
- enableImg
 - View, [32](#)
- Game, [12](#)
 - getConfig, [13](#)
 - getGSM, [13](#)
 - getInstance, [13](#)
 - getView, [13](#)
- GameOverState, [14](#)
 - handleInput, [15](#)
 - update, [15](#)
- GameStateManager, [15](#)
 - handleInput, [16](#)
 - push, [16](#)
 - set, [16](#)
 - update, [16](#)
- getBackgroundColor
 - View, [32](#)
- getBlocks
 - Config, [9](#)
- getConfig
 - Game, [13](#)
- getFont
 - View, [32](#)
- getGSM
 - Game, [13](#)
- getHeight
 - View, [33](#)
- getInstance
 - Game, [13](#)
- getTexture
 - View, [33](#)
- getVelocity
 - Config, [9](#)
- getView
 - Game, [13](#)
- getWidth
 - View, [33](#)
- getWindow
 - View, [33](#)
- handleInput
 - ConfigState, [11](#)
 - GameOverState, [15](#)
 - GameStateManager, [16](#)
 - HelpState, [18](#)
 - MenuState, [20](#)
 - PauseState, [25](#)
 - PlayState, [27](#)
 - StartState, [29](#)
 - State, [30](#)
- HelpState, [17](#)
 - handleInput, [18](#)
 - update, [18](#)
- IMAGES_COUNT
 - View, [34](#)
- LEVEL
 - Config, [9](#)
- loadImg2Texture
 - View, [33](#)
- MenuState, [19](#)
 - handleInput, [20](#)
 - update, [20](#)
- MultiOptionField, [20](#)
 - addOption, [22](#)
 - draw, [22](#)
 - MultiOptionField, [22](#)
 - selectNext, [23](#)
 - selectPrevious, [23](#)
 - setCurrent, [23](#)
 - setOptionString, [23](#)
 - setPosition, [23](#)

- PauseState, [24](#)
 - handleInput, [25](#)
 - update, [25](#)
- PlayState, [25](#)
 - handleInput, [27](#)
 - update, [28](#)
- push
 - GameStateManager, [16](#)
- resize
 - View, [34](#)
- select
 - Button, [7](#)
- selectNext
 - MultiOptionField, [23](#)
- selectPrevious
 - MultiOptionField, [23](#)
- set
 - GameStateManager, [16](#)
- setBlocks
 - Config, [9](#)
- setCurrent
 - MultiOptionField, [23](#)
- setOptionString
 - MultiOptionField, [23](#)
- setPosition
 - Button, [7](#)
 - MultiOptionField, [23](#)
- setString
 - Button, [7](#)
- setText
 - Button, [7](#)
- StartState, [28](#)
 - handleInput, [29](#)
 - update, [29](#)
- State, [30](#)
 - handleInput, [30](#)
 - update, [31](#)
 - updated, [31](#)
- time
 - View, [34](#)
- unselect
 - Button, [7](#)
- update
 - ConfigState, [12](#)
 - GameOverState, [15](#)
 - GameStateManager, [16](#)
 - HelpState, [18](#)
 - MenuState, [20](#)
 - PauseState, [25](#)
 - PlayState, [28](#)
 - StartState, [29](#)
 - State, [31](#)
- updated
 - State, [31](#)
- View, [31](#)
- enableImg, [32](#)
- getBackgroundColor, [32](#)
- getFont, [32](#)
- getHeight, [33](#)
- getTexture, [33](#)
- getWidth, [33](#)
- getWindow, [33](#)
- IMAGES_COUNT, [34](#)
- loadImg2Texture, [33](#)
- resize, [34](#)
- time, [34](#)