Projekt

Kakuro Game

DOKUMENTACJA

Kielar Kacper

Spis treści

[Button 4](#_Toc156388742)

[Opis klasy Button 4](#_Toc156388743)

[Zmienne prywatne klasy 4](#_Toc156388744)

[Konstruktor i destruktor klasy Button 4](#_Toc156388745)

[void draw(sf::RenderWindow& window) const; 4](#_Toc156388746)

[bool isMouseOver(const sf::RenderWindow& window) const; 4](#_Toc156388747)

[bool isClicked(const sf::Event& event, const sf::RenderWindow& window); 4](#_Toc156388748)

[CotoKakuro 5](#_Toc156388749)

[Opis klasy 5](#_Toc156388750)

[void OpenWindow(); 5](#_Toc156388751)

[NowaGra 5](#_Toc156388752)

[Opis ogólny 5](#_Toc156388753)

[Zmienne prywatne klasy 6](#_Toc156388754)

[std::list<PoleBlanc> getpolaBlanc(); 6](#_Toc156388755)

[void addPoleBlanc(PoleBlanc poleblanc); 6](#_Toc156388756)

[std::list<PoledoWpisania> getpoladoWpisania(); 6](#_Toc156388757)

[void addPoledoWpisania(PoledoWpisania poleblanc); 7](#_Toc156388758)

[void wypiszObjekty(sf::RenderWindow& window); 7](#_Toc156388759)

[void wypiszEventy(const sf::Event& event, sf::RenderWindow& window); 7](#_Toc156388760)

[void CzyscListy(); 7](#_Toc156388761)

[void UtworzListy(int m, sf::Font& font,bool uzupelniaj); 7](#_Toc156388762)

[void UtworzMacierzRozwiazan(const int& przypadek); 8](#_Toc156388763)

[int GeneratorWarunku(const int poczatek, const int koniec); 12](#_Toc156388764)

[void Sprawdz(const sf::Font& font); 13](#_Toc156388765)

[void SprawdzNapis(sf::RenderWindow& window) const; 13](#_Toc156388766)

[void Autouzupelnij(sf::Font& font); 13](#_Toc156388767)

[void CzysclisteMatrix(); 13](#_Toc156388768)

[PoleBlanc 14](#_Toc156388769)

[Opis ogólny 14](#_Toc156388770)

[Zmienne prywatne klasy 14](#_Toc156388771)

[Konstruktor klasy 14](#_Toc156388772)

[void DrawPoleBlanc(sf::RenderWindow& window) const; 14](#_Toc156388773)

[PoledoWpisania 15](#_Toc156388774)

[Opis klasy 15](#_Toc156388775)

[Zmienne prywatne klasy 15](#_Toc156388776)

[Konstruktor klasy 15](#_Toc156388777)

[void handleEvent(const sf::Event& event, sf::RenderWindow& window); 15](#_Toc156388778)

[void draw(sf::RenderWindow& window) const; 15](#_Toc156388779)

[int getValue() const; 15](#_Toc156388780)

[void setvalue(int value); 16](#_Toc156388781)

[int getIdK() const; 16](#_Toc156388782)

[int getIdW() const; 16](#_Toc156388783)

[Text 16](#_Toc156388784)

[Opis ogólny metod zawartych w text 16](#_Toc156388785)

[Zawijanie tekstu 16](#_Toc156388786)

[Wczytanie fontu 17](#_Toc156388787)

[Window 17](#_Toc156388788)

[Opis klasy 17](#_Toc156388789)

[void ShowWindow(); 17](#_Toc156388790)

[AbhayaLibreBoldFont 19](#_Toc156388791)

[Plik wykonywalny 19](#_Toc156388792)

# Button

### Opis klasy Button

Klasa **Button** reprezentuje interaktywny przycisk w interfejsie użytkownika dla aplikacji korzystających z biblioteki SFML. Przycisk składa się z prostokątnego obszaru, który może być naciśnięty, wyświetlającego tekst. Oferuje możliwość interakcji, sprawdzając, czy kursor myszy znajduje się nad nim, oraz czy został kliknięty.

### Zmienne prywatne klasy

* **shape**: **sf::RectangleShape** - Kształt przycisku (prostokąt).
* **buttonText**: **sf::Text** - Tekst wyświetlany na przycisku.
* **position**: **sf::Vector2f** - Pozycja przycisku.
* **size**: **sf::Vector2f** - Rozmiar przycisku.

### Konstruktor i destruktor klasy Button

Konstruktor:

Button(sf::Vector2f position, sf::Vector2f size, sf::Font& font, const std::string& text);

Tworzy obiekt przycisku na podstawie podanej pozycji, rozmiaru, czcionki oraz tekstu.

Destruktor:

~Button();

Opróżnia zasoby zajmowane przez obiekt przycisku.

### void draw(sf::RenderWindow& window) const;

Rysuje przycisk i tekst na oknie.

### bool isMouseOver(const sf::RenderWindow& window) const;

Sprawdza, czy kursor myszy znajduje się nad obszarem przycisku.

### bool isClicked(const sf::Event& event, const sf::RenderWindow& window);

Sprawdza, czy przycisk został kliknięty. Zmienia kolor przycisku po wciśnięciu i przywraca pierwotny kolor po puszczeniu przycisku.

# 

# CotoKakuro

### Opis klasy

Klasa **CotoKakuro** reprezentuje główny moduł aplikacji "Co to Kakuro?". Oferuje jedną publiczną metodę **OpenWindow**, która otwiera okno interfejsu użytkownika, prezentujące informacje dotyczące gry Kakuro.

### void OpenWindow();

Otwiera główne okno aplikacji, w którym prezentowane są zasady gry Kakuro oraz dostępna jest opcja wyjścia. Okno jest renderowane przy użyciu biblioteki SFML. W pętli głównej okna, obsługuje zdarzenia takie jak zamknięcie okna i kliknięcie przycisku "Wyjscie". Okno zawiera również przycisk "Wyjscie" reprezentowany przez obiekt klasy **Button**.

Wewnątrz metody **OpenWindow**:

* Tworzone jest główne okno przy użyciu SFML.
* Wczytywana jest czcionka do wyświetlania tekstu.
* Tworzony jest obiekt klasy **Button** reprezentujący przycisk "Wyjscie".
* Okno jest wypełniane tekstem dotyczącym zasad gry Kakuro.
* Tekst jest odpowiednio formatowany, a następnie rysowany na oknie.
* Przycisk "Wyjscie" jest rysowany na oknie.
* W pętli głównej obsługiwane są zdarzenia, takie jak zamknięcie okna i kliknięcie przycisku "Wyjscie".
* Okno jest odświeżane w każdej iteracji pętli.

# NowaGra

### Opis ogólny

Klasa **NowaGra** jest centralnym elementem w implementacji gry logicznej. Odpowiada za reprezentację i zarządzanie stanem planszy do rozwiązania. Kluczowe elementy tej klasy to listy obiektów reprezentujących puste pola (**polaBlanc**) oraz pola, które można wypełnić (**poladoWpisania**). Dodatkowo, przechowuje macierz rozwiązania, liczby warunków, oraz udostępnia metody do interakcji z planszą, takie jak rysowanie obiektów, obsługa zdarzeń, sprawdzanie poprawności rozwiązania, czy automatyczne uzupełnianie pustych pól.

Klasa ta jest również odpowiedzialna za generowanie warunków planszy, tworzenie macierzy rozwiązania, a także obsługę graficznego interfejsu użytkownika za pomocą biblioteki SFML. Metody klasy pozwalają na dynamiczne tworzenie i modyfikowanie planszy, sprawdzanie poprawności rozwiązania oraz wizualizację informacji na ekranie.

### Zmienne prywatne klasy

polaBlanc (std::list<PoleBlanc>): Lista obiektów reprezentujących puste pola w planszy.

poladoWpisania (std::list<PoledoWpisania>): Lista obiektów reprezentujących pola do wpisania na planszy.

liczba\_warunek (int): Zmienna przechowująca liczbę warunków.

liczba\_do\_matrix (int): Zmienna pomocnicza do operacji na macierzy rozwiązania.

liczba\_do\_mapy\_objektow (int): Zmienna pomocnicza do operacji na macie obiektów planszy.

rozwiazanie (int\*\*): Dwuwymiarowa tablica przechowująca macierz rozwiązania.

liczba (int): Zmienna pomocnicza do generowania warunków.

liczbaWierszy\_przypadek (int): Liczba wierszy w przypadku planszy.

liczbaKolumn (int): Liczba kolumn w przypadku planszy.

czysarozne (bool): Flaga informująca, czy liczby w wierszu i kolumnie są różne.

textSprawdz (sf::Text): Tekst do wyświetlania informacji o poprawności rozwiązania.

### std::list<PoleBlanc> getpolaBlanc();

**getpolaBlanc() -> std::list<PoleBlanc>**

* **Opis**: Zwraca listę pól blank w planszy.
* **Zwraca**: Lista obiektów klasy PoleBlanc.

### void addPoleBlanc(PoleBlanc poleblanc);

**addPoleBlanc(PoleBlanc poleblanc)**

* **Opis**: Dodaje obiekt PoleBlanc do listy pól blank.
* **Parametry**:
  + *poleblanc* (PoleBlanc): Obiekt do dodania do listy.

### std::list<PoledoWpisania> getpoladoWpisania();

**getpoladoWpisania() -> std::list<PoledoWpisania>**

* **Opis**: Zwraca listę pól do wpisania w planszy.
* **Zwraca**: Lista obiektów klasy PoledoWpisania.

### void addPoledoWpisania(PoledoWpisania poleblanc);

**addPoledoWpisania(PoledoWpisania poledowpisania)**

* **Opis**: Dodaje obiekt PoledoWpisania do listy pól do wpisania.
* **Parametry**:
  + *poledowpisania* (PoledoWpisania): Obiekt do dodania do listy.

### void wypiszObjekty(sf::RenderWindow& window);

**wypiszObjekty(sf::RenderWindow& window)**

* **Opis**: Rysuje obiekty planszy na oknie.
* **Parametry**:
  + *window* (sf::RenderWindow&): Okno, na którym mają być rysowane obiekty.

### void wypiszEventy(const sf::Event& event, sf::RenderWindow& window);

**wypiszEventy(const sf::Event& event, sf::RenderWindow& window)**

* **Opis**: Obsługuje zdarzenia dla pól do wpisania na planszy.
* **Parametry**:
  + *event* (const sf::Event&): Zdarzenie do obsłużenia.
  + *window* (sf::RenderWindow&): Okno, na którym ma być obsługiwane zdarzenie

### void CzyscListy();

**CzyscListy()**

* **Opis**: Czyści listy pól blank i pól do wpisania w planszy.

### void UtworzListy(int m, sf::Font& font,bool uzupelniaj);

**UtworzListy(int m, sf::Font& font, bool uzupelniaj)**

* **Opis**: Tworzy listy obiektów planszy na podstawie macierzy rozwiązania.
* **Parametry**:
  + *m* (int): Przypadek planszy do utworzenia.
  + *font* (sf::Font&): Czcionka do wykorzystania w obiektach tekstowych.
  + *uzupelniaj* (bool): Flaga określająca, czy pola do wpisania mają być uzupełniane wartościami.

### void UtworzMacierzRozwiazan(const int& przypadek);

**UtworzMacierzRozwiazan(const int& przypadek)**

* **Opis**: Tworzy macierz rozwiązania na podstawie wybranego przypadku planszy. Występują trzy przypadki plansz: 3x3,4x4, 7x7. Dla rozmiarów 3 i 4 algorytm tworzenia jest identyczny, natomiast dla rozmiaru 7x7 metoda wykorzystuje inne podejście.
* **Parametry**:
  + *przypadek* (const int&): Przypadek planszy do utworzenia.

**Kod tworzenia macierzy dla plansz 3x3 i 4x4:**

//plansza 3x3 i 4x4

for (int i = liczbaWierszy\_przypadek - 1; i >-1; --i)

{

for (int j = liczbaKolumn - 1; j >-1; j-=2)

{

if (i==0 && j==1)

{

rozwiazanie[i][j] = -1;

rozwiazanie[i][j-1] = -1;

}

//suma

else if (i == 0)

{

//wiersz

//pola blank

rozwiazanie[i][j] = -1;

suma = 0;

for (int ii = 1; ii < liczbaWierszy\_przypadek; ii++)

{

suma += rozwiazanie[ii][j];

rozwiazanie[i][j-1] = suma;

}

}

else if (j==1)

{

//kolumna

//pola blank

rozwiazanie[i][j-1] = -1;

suma = 0;

for (int jj = 3; jj < liczbaKolumn; jj += 2)

{

suma += rozwiazanie[i][jj];

rozwiazanie[i][j] = suma;

}

}

else

{

//pole do wpisania

rozwiazanie[i][j-1] = 0;

do

{

liczba = GeneratorWarunku(1, 9);

czysarozne = true;

//sprawdzenie czy jest rozna w wierszu i kolumnie

//wiersz

for (int iii = liczbaWierszy\_przypadek - 1; iii > 0; iii--)

{

if (rozwiazanie[iii][j] == liczba)

{

czysarozne = false;

break;

}

}

//kolumna

for (int jjj = liczbaKolumn - 1; jjj > 0; jjj-=2)

{

if (rozwiazanie[i][jjj] == liczba)

{

czysarozne = false;

break;

}

}

if (czysarozne)

{

rozwiazanie[i][j] = liczba;

}

} while (!czysarozne);

}

}

}

**Kod tworzenia macierzy dla planszy 7x7:**

for (int i = liczbaWierszy\_przypadek - 1; i > -1; --i)

{

for (int j = liczbaKolumn - 1; j > 0; j -= 2)

{

//wiersz 0 lub kolumna 0

if (i == 0 || j == 1)

{

//warunki dla pola blank

//min dwa pola do wpisania

if (

//kolumna

i < liczbaWierszy\_przypadek - 2 &&

rozwiazanie[i + 1][j] != 0 && rozwiazanie[i + 2][j] != 0 &&

rozwiazanie[i + 1][j - 1] == 0 && rozwiazanie[i + 2][j - 1] == 0

)

{

//suma kolumna

suma = 0;

for (int ii = i + 1; ii < liczbaWierszy\_przypadek; ii++)

{

if (rozwiazanie[ii][j - 1]!=0)

{

break;

}

suma += rozwiazanie[ii][j];

}

rozwiazanie[i][j - 1] = suma;

}

else

{

rozwiazanie[i][j - 1] = -1;

}

if (

//wiersz

j < liczbaKolumn - 1 &&

rozwiazanie[i][j + 2] != 0 && rozwiazanie[i][j + 4] != 0 &&

rozwiazanie[i][j + 1] == 0 && rozwiazanie[i][j + 3] == 0

)

{

//suma wiersz

suma = 0;

for (int jj = j + 2; jj < liczbaKolumn; jj += 2)

{

if (rozwiazanie[i][jj - 1] !=0 )

{

break;

}

suma += rozwiazanie[i][jj];

}

rozwiazanie[i][j] = suma;

}

else

{

rozwiazanie[i][j] = -1;

}

}

else if (

(i > liczbaWierszy\_przypadek \* 2 / 3 && j > (liczbaKolumn) \* 2 / 3) //prawy dolny

|| (i > liczbaWierszy\_przypadek \* 2 / 3 && j <= (liczbaKolumn) / 3 + 1) //lewy dolny

|| (i <= liczbaWierszy\_przypadek \* 2 / 3 && i > liczbaWierszy\_przypadek / 3 && j > (liczbaKolumn) / 3 + 1 && j <= (liczbaKolumn) \* 2 / 3) //srodek srodek

|| (i <= liczbaWierszy\_przypadek / 3 && j > (liczbaKolumn) \* 2 / 3)//prawy gorny

|| (i <= liczbaWierszy\_przypadek / 3 && j <= (liczbaKolumn) / 3 + 1)//lewy gorny

) //skrajne rogi

{

//pole do wpisania

rozwiazanie[i][j - 1] = 0;

do

{

liczba = GeneratorWarunku(1, 9);

czysarozne = true;

//kolunma

for (int iii = i; iii <= liczbaWierszy\_przypadek - 1; iii++)

{

if (rozwiazanie[iii][j] == liczba)

{

czysarozne = false;

break;

}

}

//wiersz

for (int jjj = j; jjj <= liczbaKolumn - 1; jjj += 2)

{

if (rozwiazanie[i][jjj] == liczba)

{

czysarozne = false;

break;

}

}

if (czysarozne)

{

rozwiazanie[i][j] = liczba;

}

} while (!czysarozne);

}

//srodki

else

{

if (GeneratorWarunku(0, 1))

{

//warunki dla pola blank

//min dwa pola do wpisania

if (

//kolumna

i < liczbaWierszy\_przypadek - 2 &&

rozwiazanie[i + 1][j] != 0 && rozwiazanie[i + 2][j] != 0 &&

rozwiazanie[i + 1][j - 1] == 0 && rozwiazanie[i + 2][j - 1] == 0

)

{

//suma kolumna

suma = 0;

for (int ii = i+1; ii < liczbaWierszy\_przypadek; ii ++)

{

if (rozwiazanie[ii][j - 1] != 0 )

{

break;

}

suma += rozwiazanie[ii][j];

}

rozwiazanie[i][j - 1] = suma;

}

else

{

rozwiazanie[i][j - 1] = -1;

}

if (

//wiersz

j < liczbaKolumn - 1 &&

rozwiazanie[i][j + 2] != 0 && rozwiazanie[i][j + 4] != 0 &&

rozwiazanie[i][j + 1] == 0 && rozwiazanie[i][j + 3] == 0

)

{

//suma wiersz

suma = 0;

for (int jj = j + 2; jj < liczbaKolumn; jj+=2)

{

if (rozwiazanie[i][jj-1] != 0)

{

break;

}

suma += rozwiazanie[i][jj];

}

rozwiazanie[i][j] = suma;

}

else

{

rozwiazanie[i][j] = -1;

}

}

else

{

//pole do wpisania

rozwiazanie[i][j - 1] = 0;

do

{

liczba = GeneratorWarunku(1, 9);

czysarozne = true;

//kolunma

for (int iii = i; iii <= liczbaWierszy\_przypadek-1; iii++)

{

if (rozwiazanie[iii][j] == liczba)

{

czysarozne = false;

break;

}

}

//wiersz

for (int jjj = j; jjj <= liczbaKolumn-1; jjj+=2)

{

if (rozwiazanie[i][jjj] == liczba)

{

czysarozne = false;

break;

}

}

if (czysarozne)

{

rozwiazanie[i][j] = liczba;

}

} while (!czysarozne);

}

}

}

}

### int GeneratorWarunku(const int poczatek, const int koniec);

**GeneratorWarunku(const int poczatek, const int koniec) -> int**

* **Opis**: Generuje warunek w zakresie od początku do końca.
* **Parametry**:
  + *poczatek* (const int): Dolna granica zakresu generowania warunku.
  + *koniec* (const int): Górna granica zakresu generowania warunku.
* **Zwraca**: Wygenerowany warunek.

### void Sprawdz(const sf::Font& font);

**Sprawdz(const sf::Font& font)**

* **Opis**: Sprawdza poprawność rozwiązania planszy. Weryfikuje czy nie występują powtórzenia, czy sumy wierszy i kolumn są zgodne oraz czy tabela została uzupełniona. Generuje odpowiedni komunikat w zależności od wyniku sprawdzania. Są to komunikaty:

"Uzupelnij tabele"

"SPROBUJ JESZCZE RAZ"

"LAMIGLOWKA ROZWIAZANA POPRAWNIE!"

### void SprawdzNapis(sf::RenderWindow& window) const;

**SprawdzNapis(sf::RenderWindow& window) const**

* **Opis**: Wyświetla informacje o poprawności rozwiązania na ekranie. Wypisuje tekst wygenerowany przez metodę Sprawdz
* **Parametry**:
  + *window* (sf::RenderWindow&): Okno, na którym ma być wyświetlony tekst.

### 

### void Autouzupelnij(sf::Font& font);

**Autouzupelnij(sf::Font& font)**

* **Opis**: Automatycznie uzupełnia puste pola planszy na podstawie rozwiązania wygenerowanego przy tworzeniu macierzy kakuro.

### void CzysclisteMatrix();

**CzysclisteMatrix()**

* **Opis**: Czyści zaalokowaną pamięć dla macierzy rozwiązania.

# PoleBlanc

### Opis ogólny

Klasa **PoleBlanc** reprezentuje pojedyncze pole w grze Kakuro. Każde pole zawiera prostokątny kształt, a także teksty reprezentujące sumy dla wiersza i kolumny. Klasa jest wykorzystywana do reprezentacji pól pustych.

### Zmienne prywatne klasy

* **sf::RectangleShape shape:** Kształt prostokątny reprezentujący pole.
* **sf::Text poleblancText\_sumaWiersz:** Tekst reprezentujący sumę dla wiersza.
* **sf::Text poleblancText\_sumaKolumna:** Tekst reprezentujący sumę dla kolumny.
* **sf::Vector2f position:** Wektor 2D reprezentujący pozycję pola.

### Konstruktor klasy

PoleBlanc(sf::Vector2f position, sf::Font& font, const std::string& suma\_kolumna, const std::string& suma\_wiersz);

Konstruktor przyjmuje pozycję, font, oraz teksty reprezentujące sumy dla kolumny i wiersza. Inicjalizuje kształt, ustawiając jego pozycję, rozmiar i kolor. Ustawia także teksty dla sumy wiersza i kolumny, ich rozmiar, kolor oraz pozycję wewnątrz pola.

### void DrawPoleBlanc(sf::RenderWindow& window) const;

Metoda rysuje pole w oknie **sf::RenderWindow**. Rysuje prostokątny kształt, a także teksty reprezentujące sumy dla wiersza i kolumny.

# PoledoWpisania

### Opis klasy

Klasa **PoledoWpisania** reprezentuje interaktywne pole do wprowadzania tekstu w grze Kakuro. Każde pole zawiera prostokątny kształt, reprezentację tekstu, a także obsługę zdarzeń, takich jak naciśnięcie myszy czy wprowadzanie tekstu.

### Zmienne prywatne klasy

* **sf::RectangleShape rectangle:** Kształt prostokątny reprezentujący pole.
* **sf::Text text:** Tekst reprezentujący wprowadzony tekst.
* **bool isTextInputActive:** Flaga informująca, czy pole jest aktywne i gotowe do wprowadzania tekstu.
* **std::string inputText:** Przechowuje wprowadzony tekst.
* **const int id\_k:** Identyfikator kolumny, do której należy pole.
* **const int id\_w:** Identyfikator wiersza, do którego należy pole.
* **sf::Vector2f position:** Wektor 2D reprezentujący pozycję pola.

### Konstruktor klasy

PoledoWpisania(int k, int w, sf::Vector2f position, sf::Font& font);

Konstruktor przyjmuje identyfikatory kolumny (**k**) i wiersza (**w**), pozycję, font, oraz inicjalizuje kształt prostokątny i tekst wewnątrz pola.

### void handleEvent(const sf::Event& event, sf::RenderWindow& window);

Metoda obsługuje zdarzenia, takie jak naciśnięcie myszy czy wprowadzanie tekstu, dostosowując stan pola i kolor w zależności od interakcji.

### 

### void draw(sf::RenderWindow& window) const;

Metoda rysuje pole w oknie **sf::RenderWindow**. Rysuje prostokątny kształt oraz tekst wprowadzony do pola.

### int getValue() const;

Metoda zwraca wartość liczbową wprowadzoną w polu. Jeśli pole jest puste, zwraca 0.

### void setvalue(int value);

Metoda ustawia wartość liczbową w polu, aktualizując tekst i kolor. Wartość liczbową przekazywaną jako argument.

### int getIdK() const;

Metoda zwraca identyfikator kolumny, do której należy pole.

### int getIdW() const;

Metoda zwraca identyfikator wiersza, do którego należy pole.

# Text

### Opis ogólny metod zawartych w text

Plik **Text.hpp** zawiera dwie funkcje pomocnicze do obsługi tekstu w kontekście aplikacji SFML: **wrapText** i **setFont**. Funkcje te są używane do zawijania tekstu i wczytywania niestandardowego fontu.

### Zawijanie tekstu

**Funkcja wrapText**

std::string wrapText(const std::string& text, sf::Font& font, unsigned int characterSize, float maxWidth);

Opis

Funkcja **wrapText** przyjmuje tekst, obiekt czcionki **sf::Font**, rozmiar znaku **characterSize** oraz maksymalną szerokość **maxWidth**. Jej zadaniem jest zawijanie tekstu w nową linię, gdy szerokość tekstu przekracza podaną maksymalną szerokość.

Parametry

* **text** : **const std::string&** - Tekst, który ma zostać zawinięty.
* **font** : **sf::Font&** - Referencja do obiektu czcionki SFML.
* **characterSize** : **unsigned int** - Rozmiar znaku czcionki.
* **maxWidth** : **float** - Maksymalna szerokość linii tekstu.

Zwracana wartość

Funkcja zwraca zawinięty tekst w formie **std::string**.

### 

### Wczytanie fontu

**Funkcja setFont**

sf::Font setFont();

Opis

Funkcja **setFont** służy do wczytywania niestandardowego fontu, w tym przypadku załadowanie fontu z pamięci (przy użyciu pliku nagłówkowego **AbhayaLibreBoldFont.hpp**).

Parametry

Brak.

Zwracana wartość

Funkcja zwraca obiekt **sf::Font** wczytany z pamięci.

# Window

### Opis klasy

Klasa **Window** reprezentuje główne okno aplikacji Kakuro Game. Zawiera funkcję **ShowWindow**, która obsługuje interakcje użytkownika, renderuje okno oraz komunikuje się z innymi elementami gry.

### void ShowWindow();

Metoda **ShowWindow** inicjalizuje i obsługuje główne okno gry. Tworzy obiekty przycisków, obsługuje zdarzenia, takie jak zamknięcie okna czy kliknięcie przycisku, a także komunikuje się z innymi elementami gry, takimi jak obiekty planszy (**NowaGra**), przekazując odpowiednie zdarzenia.

Kod metody:

void Window::ShowWindow()

{

//crate window object

sf::RenderWindow window(sf::VideoMode(1000, 500), "Kakuro Game");

//font file

sf::Font font = setFont();

//nowa gra

NowaGra nowagra;

//create rectangle button

Button button\_1(sf::Vector2f(780, 20), sf::Vector2f(200, 50), font, "Co to KAKURO?");

Button button\_2(sf::Vector2f(780, 80), sf::Vector2f(200, 50), font, "Nowa gra dla\n planszy 3x3");

Button button\_3(sf::Vector2f(780, 140), sf::Vector2f(200, 50), font, "Nowa gra dla\n planszy 4x4");

Button button\_4(sf::Vector2f(780, 200), sf::Vector2f(200, 50), font, "Nowa gra dla\n planszy 7x7");

Button button\_5(sf::Vector2f(780, 260), sf::Vector2f(200, 50), font, "Sprawdz");

Button button\_6(sf::Vector2f(780, 320), sf::Vector2f(200, 50), font, "Autouzupelnianie");

Button button\_exit(sf::Vector2f(780, 430), sf::Vector2f(200, 50), font, "Wyjscie");

//open window

while (window.isOpen())

{

//exit event

sf::Event event;

while (window.pollEvent(event))

{

//exit

if (event.type == sf::Event::Closed) {

window.close();

}

//if button cicked

if (button\_1.isClicked(event, window))

{

CotoKakuro kakuroepl;

kakuroepl.OpenWindow();

}

if (button\_2.isClicked(event, window))

{

nowagra.UtworzListy(3, font,false);

}

if (button\_3.isClicked(event, window)) {

nowagra.UtworzListy(4, font,false);

}

if (button\_4.isClicked(event, window)) {

nowagra.UtworzListy(7, font,false);

}

if (button\_5.isClicked(event, window)) {

nowagra.Sprawdz(font);

}

if (button\_6.isClicked(event, window)) {

nowagra.Autouzupelnij(font);

}

if (button\_exit.isClicked(event, window)) {

window.close();

}

nowagra.wypiszEventy(event, window);

}

window.clear(sf::Color(128, 128, 128));

//render buttons

button\_1.draw(window);

button\_2.draw(window);

button\_3.draw(window);

button\_4.draw(window);

button\_5.draw(window);

button\_6.draw(window);

button\_exit.draw(window);

nowagra.wypiszObjekty(window);

nowagra.SprawdzNapis(window);

window.display();

}

}

# AbhayaLibreBoldFont

Plik AbhayaLibreBoldFont.hpp zawiera bitmapę czczionki wykorzystywanej w programie.

Zawiera zmienne:

unsigned int AbhayaLibre\_Bold\_ttf\_len – długość bitmapy

unsigned char AbhayaLibre\_Bold\_ttf[] – tablica zawierająca bitmapę czczionki

# Plik wykonywalny

Plik **main.cpp** to główny plik programu, który tworzy obiekt klasy **Window** i wywołuje na nim metodę **ShowWindow** w funkcji **WinMain**.

Głównym celem programu jest stworzenie i wyświetlenie okna gry za pomocą klasy **Window**.

Klasa **Window** zawiera jedną metodę **ShowWindow**, która jest odpowiedzialna za inicjalizację i obsługę głównego okna gry.

Kod pliku wykonywalnego:

#include <SFML/Graphics.hpp>

#include "Window.h"

int WinMain()

{

Window window;

window.ShowWindow();

return 0;

}