Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Laboratorium
programowania komputerów

Edytor Graficzny

autor	Mateusz Płonka
prowadzący	Grzegorz Wojciech Kwiatkowski
rok akademicki	2019/2020
kierunek	Informatyka
rodzaj studiów	SSI
semestr	4
grupa	3
data sporządzenia sprawozdania	04.05.2020

Link do projektu:
https://drive.google.com/drive/folders/1tzITwFcXYGOe6gbrA5haXjy6lARn86_M?usp=sharing

1.Temat projektu

Zadaniem projektu było utworzenie aplikacji w języku programowania C++ której główną funkcjonalnością jest tworzenia na ekranie rysunków złożonych z podstawowych elementów jak np. linia, kwadrat, prostokąt, elipsa, okrąg oraz trójkąt.

Program został wykonany z uwzględnieniem wszystkich podanych wytycznych oraz przy użyciu zewnętrznej biblioteki graficznej SFML.

2.Analiza zadania

2.1.Struktury danych

W programie wykorzystano podstawową listę jednokierunkową przechowującą w kolejności chronologicznej obiekty graficzne utworzone przez użytkownika w obszarze roboczym. Owa struktura o wdzięcznej nazwie „listManage" oprócz przechowywania wskaźników na narysowany kształt czy linię, jak również na następujący po nim element, przechowuje również wartości współrzędnych myszy zapisanych przy rysowaniu oraz wszystkie parametry danej grafiki. Umożliwia to do łatwy zapis oraz odczyt i ponowne odtworzenie stworzonego wcześniej obrazu.

2.2.Algorytm

Algorytm funkcjonowania aplikacji jest stosunkowo prosty, lecz dobrze przemyślany a opiera się na odpowiedniej reakcji programu na wciśnięcie a następnie puszczenie przycisku myszy w obszarze roboczym przy odpowiednio wybranych narzędziach do rysowania znajdujących się w pasku interface'u w górnej części ekranu.

Dwie pętle While znajdujące się w pliku main odpowiadają kolejno za utrzymanie otwartego okna do momentu zamknięcia programu oraz do wykrycia interakcji użytkownika z danymi funkcjonalnościami.

Druga z wspomnianych pętli zapisuje współrzędne wciśnięcia i puszczenia lewego przycisku myszy do tablicy float points[4]. W połączeniu z danymi z tablicy int click[4] przechowującymi kolejno narzędzie, kolor, grubość linii oraz wypełnienie, zostaje utworzony nowy element listy przechowujący 8 powyższych danych i wskaźnik na utworzony przez interpreter owych danych kształt.

3.Specyfikacja zewnętrzna

Program jest uruchamiany z pliku .exe. Po włączeniu ukaże się okno z dużym, białym obszarem roboczym, służącym do rysowania oraz szarego paska u góry ekranu w którym można dokonać zmiany rysowanego kształtu, koloru, grubości linii, wypełnienia kształtu oraz kilka praktycznych funkcji takich jak cofanie poprzedniego ruchu, zapis do pliku, odczyt z pliku. Całe Gui jest w pełni interaktywne, reaguje na kontakt z użytkownikiem i w przystępny sposób przekazuje wszystkie informacje.

Dodatkowo z okna konsoli wyświetlającej się obok możemy odczytać współrzędne klikanego miejsca i informacje o utworzonym kształcie.

4.Specyfikacja wewnętrzna

Program składa się z czterech plików cpp:

- **main.cpp**- główny plik aplikacji z pętlą programu, i wywołaniem podfunkcji
- **classes.cpp**- zawiera definicje funkcji klas kształtów, zarządzania listą oraz zapisy i odczytu z pliku
- **menu.cpp**- zawiera definicje funkcji klas związanych z gui takich jak przyciski, paski, teksty
- **canvas.cpp**- zawiera definicje funkcji klasy obszaru roboczego

- **classes.h**- zawiera deklaracje klas kształtów oraz ich funkcji, zarządzania listą oraz zapisy i odczytu z pliku
- **menu.h**- zawiera deklaracje klas związanych z gui takich jak przyciski, paski, teksty oraz ich funkcji
- **canvas.h**- zawiera deklaracje klasy obszaru roboczego oraz jego funkcji.

4.1 Klasy i struktury definiowane w programie

a.) Klasy związane z funkcjonowaniem programu

class Button przechowuje informacje o pojedynczym przycisku Gui, jego parametrach, teksturze i aktywności

```
//Klasa pojedynczego przycisku interface'u
class Button {

private:
        sf::RectangleShape button;      //Zmienne kształtu prostokątu biblioteki SFML
        sf::RectangleShape menuImage;   //Kolejno styl i wielkość przycisku oraz tekstura

        int btnWidth;           //Szerokość
        int btnHeight;          //Wysokość
        bool active;            //Czy aktywny
        bool hovered;           //Czy zaznaczony kursorem

public:
        void setBackColor(sf::Color color);             //Ustalanie koloru tła
        void setPosition(sf::Vector2f point, int frame);//Ustalanie pozycji
        void setActive(bool what);                      //Ustalanie aktywności
        void setHovered(bool what);                     //Ustalanie podświetnienia
```

```
        void changeTexture(int which);                  //Zmiana tekstury (ograniczona ze względu na
        wykorzystanie w jednym przypadku)
        void drawTo(sf::RenderWindow& window);          //Narysuj przycisk na ekran
        bool isMouseOver(sf::RenderWindow& window);     //Zwraca prawdę gdy kursor znajduje się na
przycisku

        Button(sf::Vector2f buttonSize, sf::Vector2f buttonPosition, float frame, sf::Color BgColor,
        sf::Texture* menuTexture, int BtnNo, int TextureRow);
```

---

class Gui- klasa konstruująca cały interface. Zawiera vektor wszystkich utworzonych przycisków lub ich grup, funkcje pozwalające na reakcje przycisków na kliknięcia lub podświetlenia kursorem oraz wykorzystanie przeładowanego operatora.

```
//Klasa konstrolująca interface
class Gui {

private:
        std::vector<std::vector<Button>> GuiVector;   //Wektor wektorów przechowujący wszystkie dostępne przyciski
        sf::RectangleShape menuBg;                    //Tło menu
        sf::Text lineText;                            //Tekst grubości linii
        sf::Text fillText;                            //Tekst wypełnienia kształtu

public:
        void drawGui(sf::RenderWindow& window);       //Rysowanie całego Gui

        //Konstruktor rzędu przyciskow powiązanych ze sobą (narzędzie/kolor)
        void makeMenu(std::vector<Button>* menuButtonRow, sf::Vector2f buttonSize,
        sf::Vector2f buttonPosition, float frame, sf::Color BgColor, sf::Texture* menuTexture, int BtnAm, int row);

        void menuReact(sf::RenderWindow& window, int row);     //Podświetlenie przycisku po najechaniu
        bool menuClick(sf::RenderWindow& window, int row);     /Reakcja na klikniecie dla pojedynczych przyciskow
        int menuListClick(sf::RenderWindow& window, int row);  //Reakcja na klinięcie dla list przycisków
        void changeText(int which, int value);                 //Zmiana tesktów w gui

        /*PRZECIĄŻONY OPERATOR*/
        operator int();

        Gui(sf::RenderWindow& window, sf::Texture* menuTexture, sf::Font* font);

};
```

---

class Canvas- klasa posiadająca parametry i funkcje zarządzające obszarem roboczym przeznaczonym do rysowania.

```
//Klasa obszaru roboczego
class Canvas {

private:
        sf::RectangleShape field;   //Zmienna kształtu prostokąta biblioteki SFML
        int fieldWidth;                     //Szerokość
        int fieldHeight;                    //Wysokość

public:
        void drawTo(sf::RenderWindow& window);        //Rysowanie obszaru roboczego
        bool isMouseOver(sf::RenderWindow& window);   //Zwraca prawdę gdy mysz znajduje się w obszarze roboczym
        Canvas();                                     //Domyślny konstruktor
};
```

Struct listManage- element listy dynamicznej przechowującej informacje o utworzonych kształtach oraz ich parametry.

```
//Struktura przechowująca utworzone kształty i informacje o nich
struct listManage
{
        ObiektGraficzny* p;
        listManage* pNext;
        int click[4];
        float points[4];
};
```

---

b.) Klasy związane z rysowaniem

class ObiektGraficzny- w pełni wirtualna klasa główna od której dziedziczą wszystkie klasy związane z rysowaniem, i która podtrzymuje polimorfizm programu. Zawiera funkcje rysowania kształtu na ekran i wyświetlania informacji o nim na konsolę

```
/Główna, w pełni witualna klasa
class ObiektGraficzny {

public:
        virtual void draw(sf::RenderWindow& window) = 0;       //Rysowanie obiektu na ekran
        virtual void showInfo(int click[4]) = 0;               //Wypisywanie informacji o obiekcie na konsole
};
```

Class Shape (Dziedziczy po ObiektGraficzny)- główna klasa kształtu posiadająca współrzędne oraz zmienną kształtu biblioteki SFML. Funkcje owej klasy odpowiadają między innymi za ustalanie pozycji, koloru, wypełnienia czy obwodu figury.

```
//Klasa kształt
class Shape :public ObiektGraficzny {

protected:
        float X1, Y1, X2, Y2;       //Współrzędne
        sf::Shape* shapeSrc;        //Zmienna kształtu z biblioteki SFML
        bool fill;                  //Wypełnienie

public:
        sf::Color chooseColor(int color);    //Wybór koloru kształtu
        void draw(sf::RenderWindow& window); //Rysowanie obiektu na ekran

        void setPoints(float points[4]);              /Ustalanie współrzędnych kształtu
        void setColor(sf::Color color);               //Ustalanie koloru kształtu
        void setFilling(int data);                    //Ustalanie wypełnienia
        void setOutline(sf::Color color, int data);   //Ustalanie grubości obranowania
        void setPosition(float x, float y);           //Ustalanie pozycji kształtu

        virtual void setSize(float a, float b)=0;     //Ustalanie wielkości kształtu
        virtual void specialCalc() = 0;               //Specjalnie obliczenia dla każdej klasy
};
```

Class Line (Dziedziczy po Shape)- klasa zawierająca zmienną kształtu prostokąta biblioteki SFML oraz funkcje określające wielkość, kąt nachylenia linii i specjalne obliczenia z nią związane.

```
//Klasa linia
class Line :public Shape {

protected:
        sf::RectangleShape lineShape;       //Zmienna kształtu prostokątu z biblioteki SFML
        float length, angle;                //Parametry linii

public:
        void setOrigin(float x, float y);   //Ustalanie początku układu współrzędnych
        void setSize(float a, float b);     //Ustalanie wielkości kształtu
        void setAngle();                    //Ustalanie kąta nachylenia linii
        void specialCalc();                 //Obliczenia dla linii
        void showInfo(int click[4]);        //Wypisywanie informacji o obiekcie na konsole

        Line(int click[4], float points[4]);
};
```

Class Rectangle (Dziedziczy po Shape)- klasa zawierająca zmienną kształtu prostokąta biblioteki SFML oraz funkcje określające wielkość, początek układu współrzędnych i obliczenia z nim związane

```
//Klasa prostokąt
class Rectangle :public Shape {

protected:
	sf::RectangleShape rectShape;		//Zmienna kształtu prostokąta z biblioteki SFML
	float a, b, field;			//Parametry prostokąta

public:
	void setSize(float a, float b);		//Ustalanie wielkości kształtu
	void setOrigin();			//Ustalanie początku układu współrzędnych
	void specialCalc();			//Obliczenia dla prostokąta
	void showInfo(int click[4]);		//Wypisywanie informacji o obiekcie na konsole

	Rectangle(int click[4], float points[4]);
}
```

Class Square (Dziedziczy po Rectangle)- klasa modyfikująca prostokąt w celu uzyskania kwadratu. Zawiera funkcje pozwalające na obliczenia z tym związane.

```
//Klasa kwadrat
class Square :public Rectangle {

public:
	void setOrigin();			//Ustalanie początku układu współrzędnych
	void showInfo(int click[4]);		//Wypisywanie informacji o obiekcie na konsole

	Square(int click[4], float points[4]);
};
```

Class Ellipse (Dziedziczy po Shape)- klasa zawierająca zmienną kształtu okręgu biblioteki SFML oraz funkcje określające wielkość, początek układu współrzędnych i obliczenia z ną związane

```
/Klasa elipsa
class Ellipse :public Shape {

protected:
	sf::CircleShape circleShape;		//Zmienna kształtu okręgu z biblioteki SFML
	float a, b, shortR, longR, field;	//Parametry

public:
	void setSize(float a, float b);		//Ustalanie wielkości elipsy
	void specialCalc();			//Obliczenia dla elipsy
	void showInfo(int click[4]);		//Wypisywanie informacji o obiekcie na konsole

	Ellipse(int click[4], float points[4]);
};
```

Class Circle (Dziedziczy po Ellipse)- klasa modyfikująca elipsę w celu uzyskania okręgu. Zawiera funkcje pozwalające na obliczenia z tym związane.

```
//Klasa Okrąg
class Circle :public Ellipse {

public:
	void rotate();				//Obrót i poprawa kształtu względem elipsy
	void showInfo(int click[4]);		//Wypisywanie informacji o obiekcie na konsole

	Circle(int click[4], float points[4]);
};
```

Class Triangle (Dziedziczy po Shape)- klasa zawierająca zmienną kształtu typu ConvexShape biblioteki SFML zmodyfikowaną w celu uzyskania kształtu trójkąta oraz funkcje określające wielkość, początek układu współrzędnych i obliczenia z nim związane.

```
//Klasa Trójkąt
class Triangle :public Shape {

protected:
        sf::ConvexShape triangleShape;          //Zmienna Convex przechowująca kształt
        float a, b, X3, footing, field, height; //Zmienne zawierające parametry trójkąta

public:
        void setSize(float a, float b); //Ustawienie wielkości figury w zależności od kierunku
rysowania
        void specialCalc();                             //Obliczenia matematyczne związane z trójkątem
        void showInfo(int click[4]);    //Wyświetlanie informacji o trójkącie na konsolę

        Triangle(int click[4], float point[4]); //Konstruktor trójkąta
};
```

4.2 Funkcje

a) Funkcje związane z działaniem programu

void addElement – funkcja odpowiedzialna za dodawanie nowego elementu do listy podwieszanej, jednokierunkowej zawierającej narysowane przez użytkownika kształty w kolejności FILO.
Parametry:

- listManage*& pHead- głowa listy
- int click[4]- informacje o wybranych narzędziach i kolorach
- float points[4]- informacje o współrzędnych figury
- bool fake- określa czy informacje o kształcie mają zostać wypisane na konsolę

```
void addElement(listManage*& pHead, int click[4], float points[4], bool fake);
```

---

void drawList – rysuje kształty przechowywane w elementach listy, funkcja rekurencyjna. Wywołuje elementy na ekran w kolejności LIFO.
Parametry:

- listManage* pHead- głowa listy
- sf::RenderWindow& window- zmienna bibl. SFML, przekazuje okno programu

```
void drawList(listManage* pHead, sf::RenderWindow& window);
```

---

void deleteList – funkcja usuwa dynamicznie zaalokowaną listę wykorzystując mechanikę szablonów sprawdzając przy tym zgodność z typem struktury na wypadek wykorzystania listy o innej budowie.
Parametry:

- T*& pHead- głowa listy

```
template <class T>
void deleteList(T*& pHead);
```

---

void deleteLast– funkcja usuwa ostatnio utworzony element z listy
Parametry:

- listManage*& pHead- głowa listy

```
void deleteLast(listManage*& pHead);
```

---

void virtualShape – zarządza i rysuje w czasie rzeczywistym prezentacje rysowanej figury przez puszczeniem przycisku myszy
Parametry:

- int click[4]- informacje o wybranych narzędziach i kolorach
- float points[4]- informacje o współrzędnych figury
- sf::RenderWindow& window- zmienna bibl. SFML, przekazuje okno programu

```
void virtualShape(int click[4], float points[4], sf::RenderWindow& window);
```

---

void writeToFile- funkcja zapisuje kodem informacje o utworzonych figurach do pliku JSON o podanej nazwie.
Parametry:

- listManage* pHead- głowa listy
- std::string adress- adres pliku do zapisu

```
void writeToFile(listManage* pHead, std::string adress);
```

---

void readFile– funkcja odczytuje zakodowane figury z pliku JSON i przenosi je na obszar do rysowania wykorzystując przy tym mechanizm wyjątków.
Parametry:

- listManage*& pHead- głowa listy
- std::string adress- adres pliku do odczytu

```
void readFile(listManage*& pHead, std::string adress);
```

---

bool splitWords – podfunkcja readFile służąca do podziału linijki zakodowanego pliku JSON na odpowiednie parametry oraz rozpoznania niepoprawnej składni.
Parametry:

- listManage*& pHead- głowa listy
- std::string line- linia do pocięcia
- **Return** true/false w zależności czy sprawdzany plik ma poprawną składnię

```
bool splitWords(listManage*& pHead, std::string line);
```

---

void showInfo – funkcja wykorzystuje mechanizm RTTI to poprawnego rozpoznania i wypisania informacji o narysowanym kształcie na okno konsoli.
Parametry:

- ObiektGraficzny* ptr- wskaźnik na kształt który ma zostać opisany
- int click[4]- informacje o wybranych narzędziach i kolorach

```
void showInfo(ObiektGraficzny* ptr, int click[4]);
```

---

void canvas:: / Button:: drawTo- rysuje obszar roboczy w podanym oknie
Parametry:

- sf::RenderWindow& window- zmienna bibl. SFML, przekazuje okno programu

```
void canvas::drawTo(sf::RenderWindow& window);
void Button::drawTo(sf::RenderWindow& window);
```

---

bool canvas:: / Button:: isMouseOver – zwraca wartość true gdy myszka znajduje się na powierzchni obszaru roboczego lub przycisku
Parametry:

- sf::RenderWindow& window- zmienna bibl. SFML, przekazuje okno programu
- **Return** true gdy myszka znajduje się na obszarze roboczym lub na przycisku

```
bool canvas::isMouseOver(sf::RenderWindow& window);
bool Button::isMouseOver(sf::RenderWindow& window);
```

---

void Button::setBackColor– ustala kolor danego przycisku
Parametry:

- sf::Color color- zmienna bilblioteki SFML określająca kolor obiektów SFML

```
void Button::setBackColor(sf::Color color);
```

---

void Button::setPosition – ustala pozycje przycisku z uwzględnieniem grubości obramowania
Parametry:

- sf::Vector2f point- obiekt biblioteki SFML przechowujący dwie wartości typu float. W tym przypadku x i y pozycji.
- int frame- grubość ramki

```
void Button::setPosition(sf::Vector2f point, int frame);
```

---

void Button::setActive – zmienia wartość bool active dla danego przycisku.
Parametry:

- bool what – wartość na jaką ma zostać zmieniona zmienna active

```
void Button::setActive(bool what);
```

---

void Button::setHovered- zmienia wartość bool hovered dla danego przycisku. Wartość ta określa czy przycisk jest aktualnie podświetlony przez kursor.
Parametry:

- bool what- wartość na jaką ma zostać zmieniona zmienna hovered

```
void Button::setHovered(bool what)
```

---

void Button::changeTexture – podmienia teksturę przycisku wypełnienia kształtu. Ogranicza się tylko dla jednej opcji.
Parametry:

- int which- numer tekstury która ma zostać używa

```
void Button::changeTexture(int which);
```

Konstruktor klasy Button – wykorzystuje wszystkie opisane powyżej funkcje klasy Button i finalizuje procedurę utworzenia danej opcji wyboru.
Parametry:

- sf::Vector2f buttonSize- zmienna SFML przechowująca dwie wartości float, szerokość i długość przycisku w pikselach
- sf::Vector2f buttonPosition- współrzędne x i y pozycji przycisku w oknie
- float frame- grubość ramki przycisku
- sf::Color BgColor- zmienna SFML określająca kolor przycisku
- sf::Texture* menuTexture- wskaźnik na plik z obrazkami do wyświetlenia na przyciskach
- int TextureRow- rząd z którego na zostać pobrana tekstura
- int BtnNo- nr tekstury z owego rzędu

```
Button::Button(sf::Vector2f buttonSize, sf::Vector2f buttonPosition, float frame,
sf::Color BgColor, sf::Texture* menuTexture, int BtnNo, int TextureRow);
```

---

void Gui::drawGui- wyświetlna cały wektor wektorów przycisków na ekran, zarządza wyświetlanymi tekstami i tłami.
Parametry:

- sf::RenderWindow& window- zmienna biblioteki SFML, przekazuje okno programu

```
void Gui::drawGui(sf::RenderWindow& window);
```

---

void Gui::makeMenu – konstrukcja rzędu przycisków połączonych ze sobą jak na przykład wybór koloru czy narzędzia
Parametry:

- std::vector<Button>* menuButtonRow- wektor wspólnie zależnych od siebie przycisków
- sf::Vector2f buttonSize- zmienna SFML przechowująca dwie wartości float, szerokość i długość przycisku w pikselach
- sf::Vector2f buttonPosition- współrzędne x i y pozycji przycisku w oknie
- float frame- grubość ramki przycisku
- sf::Color BgColor- zmienna SFML określająca kolor przycisku
- sf::Texture* menuTexture- wskaźnik na plik z obrazkami do wyświetlenia na przyciskach
- int row- rząd z którego na zostać pobrana tekstura
- int BtnAm- nr tekstury z owego rzędu

```
void Gui::makeMenu(std::vector<Button>* menuButtonRow, sf::Vector2f buttonSize,
sf::Vector2f buttonPosition, float frame, sf::Color BgColor, sf::Texture* menuTexture,
int BtnAm, int row);
```

---

void Gui::menuReact – wykonuje reakcję przycisku na kontakt z kursorem w postaci zmiany koloru tła na ciemniejszy.
Parametry:

- sf::RenderWindow& window- zmienna biblioteki SFML, przekazuje okno programu
- int Row- „dział" przycisków do sprawdzenia (np. kolor)

```
void Gui::menuReact(sf::RenderWindow& window, int row);
```

---

int Gui::menuListClick- reakcja na kliknięcie przycisku który jest związany z innymi np. kolor, narzędzia.
Parametry:

- sf::RenderWindow& window- zmienna biblioteki SFML, przekazuje okno programu
- int row- „dział" przycisków do sprawdzenia (np. kolor)
- **Return** zwraca numer przycisku z „działu" który został wciśnięty. Jeśli żaden wtedy zwraca „-1"

```
int Gui::menuListClick(sf::RenderWindow& window, int row);
```

---

bool Gui::menuClick- reakcja na kliknięcie przycisku który działa niezależnie od innych np. zmiana grubości linii, wypełnienie, zapis do pliku
Parametry:

- sf::RenderWindow& window- zmienna biblioteki SFML, przekazuje okno programu
- int row- numer przycisku do sprawdzenia
- **Return** zwraca true/false czy przycisk o którego zapytaliśmy został wciśnięty

```
bool Gui::menuClick(sf::RenderWindow& window, int row)
```

---

void Gui::changeText- zmienia teksty wyświetlane na pasku u góry ekranu względem aktualnego stanu wybranych narzędzi
Parametry:

- int which- numer tekstu do edycji
- int value- wartość do zmiany w tekście (np. wartość grubości linii)

```
void Gui::changeText(int which, int value);
```

Konstruktor klasy Gui- operuje opisanymi powyżej funkcjami klasy Gui i tworzy w pełni funkcjonalny interface.
Parametry:

- sf::RenderWindow& window- zmienna biblioteki SFML, przekazuje okno programu
- sf::Texture* menuTexture- wskaźnik na plik z obrazkami do wyświetlenia na przyciskach
- sf::Font* font- wskaźnik na plik z czcionką arial

```
Gui::Gui(sf::RenderWindow& window, sf::Texture* menuTexture, sf::Font* font);
```

---

b) Funkcje związane z rysowaniem kształtów

sf::Color Shape::chooseColor- zwraca odpowiedni kolor dla rysowanej figury względem podanego parametru
Parametry:

- int color- numer koloru dla kształtu
- **Return-** obiekt SFML przechowujący dane koloru

```
sf::Color Shape::chooseColor(int color);
```

---

void Shape::setPoints- funkcja ustala współrzędne figury względem wartości z parametru. Wykorzystywana przez wszystkie klasy bezpośrednio dziedziczące z Shape.
Parametry:

- float points[4]- współrzędne kliknięcia i puszczenia klawisza myszy

```
void Shape::setPoints(float points[4]);
```

---

void Shape::setColor- funkcja ustala kolor figury względem podanego jako parametr obiektu SFML. Wykorzystywana przez wszystkie klasy bezpośrednio dziedziczące z Shape.
Parametry:

- sf::Color color- obiekt SFML przechowujący informację o kolorze

```
void Shape::setColor(sf::Color color);
```

---

void Shape::setOutline- funkcja ustala grubość oraz kolor obramowania kształtu. Wykorzystywana przez wszystkie klasy bezpośrednio dziedziczące z Shape z wyjątkiem Line.
Parametry:

- sf::Color color- obiekt SFML przechowujący informację o kolorze obwodu
- int data- składowa grubości obramowania

```
void Shape::setOutline(sf::Color color, int data);
```

---

void Shape::setFilling- funkcja wypełnia kształt w zależności od wyboru w Gui przez użytkownika. Wykorzystywana przez wszystkie klasy bezpośrednio dziedziczące z Shape z wyjątkiem Line.
Parametry:

- int data- wartość 0/1 określająca czy kształt ma zostać wypełniony

```
void Shape::setFilling(int data);
```

---

void Shape::setPosition- funkcja ustala pozycję kształtu na obszarze do edycji względem podanych parametrów. Wykorzystywana przez wszystkie klasy bezpośrednio dziedziczące z Shape.

Parametry:

- float x – współrzędna X
- float y – współrzędna Y

```
void Shape::setPosition(float x, float y);
```

---

void Shape::draw- funkcja wyświetla dany obiekt kształtu na przekazany parametrem okno. Wykorzystywana przez wszystkie klasy dziedziczące z Shape. Klasa wirtualna dla klasy ObiektGraficzny dzięki której aplikacja wykorzystuje polimorfizm.

Parametry:

- sf::RenderWindow& window- zmienna biblioteki SFML, przekazuje okno programu

```
void Shape::draw(sf::RenderWindow& window);
```

---

void Line::setOrigin- funkcja ustala punkt początku układu dla danej linii będący zarazem jej środkiem ciężkości i obrotu.

Parametry:

- float x- współrzędna X punktu
- float y- współrzędna Y punktu

```
void Line::setOrigin(float x, float y);
```

---

void Line::setSize- funkcja ustala wielkość linii względem podanych parametrów. Jest to funkcja wirtualna z klasy Shape.

Parametry:

- float a- długość
- float b- szerokość

```
void Line::setSize(float a, float b);
```

---

void Line::setAngle- funkcja ustala kąt nachylenia linii względem punktu określonego przez setOrigin. Nie przyjmuje parametrów gdyż wykorzystuje wartość ze zmiennych klasy.

```
void Line::setAngle();
```

---

void Line::specialCalc- funkcja wykonuje obliczenia długości linii ze wzoru na odległość dwóch punktów oraz kąta nachylenia przy użyciu funkcji cyklometrycznej arctg. Nie przyjmuje parametrów gdyż wykorzystuje wartości ze zmiennych klasy. Jest to funkcja wirtualna z klasy Shape.

```
void Line::specialCalc();
```

---

void Line::showInfo- funkcja wyświetla w konsoli informacje o aktualnie narysowanym kształcie. W przypadku linii podaje również jej długość oraz kąt. Jest to funkcja wirtualna z klasy ObiektGraficzny.

Parametry:

- int click[4]- zakodowane wartości kształtu takie jak kolor czy grubość.

```
void Line::showInfo(int click[4]);
```

Konstruktor linii- funkcja wykorzystuje powyżej opisane funkcje klasy Line i niektóre klasy Shape w celu sfinalizowania procesu tworzenia obiektu linii. Przypisuje również obiekt SFML do wskaźnika znajdującego się w klasie Shape.

Parametry:

- int click[4]- zakodowane wartości kształtu takie jak kolor czy grubość.
- float points[4] – współrzędne punktu kliknięcia i puszczenia myszy.

```
Line::Line(int click[4], float points[4]);
```

---

void Rectangle::setSize- funkcja ustala wielkość prostokątu względem podanych parametrów. Jest to funkcja wirtualna dla klasy bazowej Shape.

Parametry:

- float a- długość
- float b- szerokość

```
void Rectangle::setSize(float a, float b);
```

---

void Rectangle::setOrigin- ustala punkt początku układu dla danego prostokątu, względem kierunku jego tworzenia, będący zarazem jej środkiem ciężkości i obrotu. Nie przyjmuje parametrów gdyż wykorzystuje wartości ze zmiennych klasy.

```
void Rectangle::setOrigin();
```

---

void Rectangle::specialCalc- funkcja wykonuje obliczenia długości, szerokości oraz pola prostokąta. Nie przyjmuje parametrów gdyż wykorzystuje wartości ze zmiennych klasy. Jest to funkcja wirtualna z klasy Shape.

```
void Rectangle::specialCalc();
```

---

void Rectangle:: / Square::showInfo- funkcja wyświetla w konsoli informacje o aktualnie narysowanym kształcie. W przypadku prostokątu lub kwadratu podaje również długości ich boków oraz pole. Jest to funkcja wirtualna z klasy ObiektGraficzny.

Parametry:

- int click[4]- zakodowane wartości kształtu takie jak kolor czy wypełnienie.

```
void Rectangle::showInfo(int click[4]);
void Square::showInfo(int click[4]);
```

Konstruktor prostokąta- funkcja wykorzystuje powyżej opisane funkcje klasy Rectangle i niektóre klasy Shape w celu sfinalizowania procesu tworzenia obiektu Prostokąta. Przypisuje również obiekt SFML do wskaźnika znajdującego się w klasie Shape.
Parametry:

- int click[4]- zakodowane wartości kształtu takie jak kolor czy grubość.
- float points[4] – współrzędne punktu kliknięcia i puszczenia myszy.

```
Rectangle::Rectangle(int click[4], float points[4]);
```

---

void Square::setOrigin- ustala punkt początku układu dla danego kwadratu, względem kierunku jego tworzenia, będący zarazem jej środkiem ciężkości i obrotu. Modyfikuje przy tym utworzony wcześniej prostokąt. Nie przyjmuje parametrów gdyż wykorzystuje wartości ze zmiennych klasy.

```
void Square::setOrigin();
```

Konstruktor kwadratu- klasa Square dziedziczy po klasie Rectangle. Funkcja konstruktora owej klasy wywołuje konstruktor Prostokąta po czym modyfikuje utworzony kształt w celu uzyskania kwadratu.
Parametry:

- int click[4]- zakodowane wartości kształtu takie jak kolor czy grubość.
- float points[4] – współrzędne punktu kliknięcia i puszczenia myszy.

```
Square::Square(int click[4], float points[4]) : Rectangle(click, points);
```

---

void Ellipse::setSize- funkcja ustala promień obiektu SFML oraz proporcje elipsy względem podanych punktów oraz kierunku tworzenia. Jest to funkcja wirtualna dla klasy bazowej Shape.
Parametry:

- float a- długość*
- float b- szerokość*
- *prostokąta który opisywałby daną elipsę

```
void Ellipse::setSize(float a, float b);
```

---

void Ellipse::specialCalc- funkcja wykonuje obliczenia promieni elipsy, jej pola oraz parametrów prostokąta opisującego ją. Nie przyjmuje parametrów gdyż wykorzystuje wartości ze zmiennych klasy. Jest to funkcja wirtualna z klasy Shape.

```
void Ellipse::specialCalc();
```

---

void Ellipse:: / Circle::showInfo- funkcja wyświetla w konsoli informacje o aktualnie narysowanym kształcie. W przypadku elipsy czy okręgu podaje również długości ich promieni oraz pole. Jest to funkcja wirtualna z klasy ObiektGraficzny.
Parametry:

- int click[4]- zakodowane wartości kształtu takie jak kolor czy wypełnienie.

```
void Ellipse::showInfo(int click[4]);
void Circle::showInfo(int click[4]);
```

Konstruktor Elipsy- funkcja wykorzystuje powyżej opisane funkcje klasy Ellipse i niektóre klasy Shape w celu sfinalizowania procesu tworzenia obiektu elipsy. Przypisuje również obiekt SFML do wskaźnika znajdującego się w klasie Shape.
Parametry:

- int click[4]- zakodowane wartości kształtu takie jak kolor czy grubość.
- float points[4] – współrzędne punktu kliknięcia i puszczenia myszy.

```
Rectangle::Rectangle(int click[4], float points[4]);
```

---

void Circle::rotate- funkcja wykorzystuje rotację obiektu elipsy w celu pozbycia się błędu przy rysowaniu. Nie przyjmuje parametrów gdyż wykorzystuje wartości ze zmiennych klasy.

```
void Circle::rotate();
```

Konstruktor okręgu- klasa Circle dziedziczy po klasie Ellipse. Funkcja konstruktora owej klasy wywołuje konstruktor elipsy po czym modyfikuje utworzony kształt w celu uzyskania okręgu/koła.
Parametry:

- int click[4]- zakodowane wartości kształtu takie jak kolor czy grubość.
- float points[4] – współrzędne punktu kliknięcia i puszczenia myszy.

```
Circle::Circle(int click[4], float points[4]): Ellipse(click,points);
```

---

void Triangle::setSize – funkcja dostosowuje wielkość i rozkład punktów trójkąta w obiekcie ConvexShape z pakietu SFML w zależności od kierunku rysowania oraz odległości od punktów drag and drop.
Parametry:

- float a – długość osi X
- float b – długość osi Y

```
void Triangle::setSize(float a, float b);
```

---

void Traingle::specialCalc – funkcja wykonuje obliczenia związane z trójkątem jak wyznaczenie długości wysokości, podstawy czy pola kszatałtu.

```
void Triangle::specialCalc();
```

---

void Triangle::showInfo – wypisuje informacje o trójkącie na konsolę
Parametry:

- int click[4] – zakodowane wartości linii takie jak kolor czy grubość.

```
void Triangle::showInfo(int click[4]);
```

Konstruktor Trójkąta – konstruktor klasy Triangle uruchamiająca wszystkie metody związane z ustaleniem rozłożenia punktów kształtu na płaszczyźnie i dostosowanie kolorów, wypełnienia czy obwodu względem podanych parametrów.
Parametry:

- int click[4]- zakodowane wartości kształtu takie jak kolor czy grubość.
- float points[4] – współrzędne punktu kliknięcia i puszczenia myszy.

```
Triangle::Triangle(int click[4], float points[4])
```

---

*Każda funkcja wymagająca przejścia przez elementy list czy pliki posiada zabezpieczenia przed brakiem elementów listy, podaniem złego id, rozróżnieniem głowy listy od elementów i zabezpieczenia przejścia przez kolejne elementy wykorzystując dodatkowy parametr w celu pozostawienia struktury listy w stanie nienaruszonym.

5.Wykorzystane mechanizmy Pk4

a) RTTI (classes.cpp/395)- funkcja **void showInfo** wykorzystuje operację typeid do poprawnego rozpoznania ostatniego narysowanego kształtu i wypisania przypisanych dla niego informacji na okno konsoli.

b) Wyjątki (classes.cpp/519)- funkcja **void readFile** wykorzystuje mechanikę wyjątków (try/throw/catch) do obsługi błędów związanych z nieistniejącym plikiem wejściowym bądź jego niepoprawną składnią.

c) Szablony (classes.cpp/475)- funkcja usuwa dynamicznie zaalokowaną listę wykorzystując mechanikę szablonów sprawdzając przy tym zgodność z typem struktury na wypadek wykorzystania listy o innej budowie.

d) Kontenery STL (menu.h/38)- klasa Gui wykorzystuje vektor vektorów klasy Button do łatwego zarządzania oraz ustawiania pozycji i współpracy między wszystkimi klawiszami interfaceu tworząc jedno spójne gui.

e) Obsługa plików JSON (classes.cpp/569)- pomimo iż zapis i odczyt w technologii JSON nie był wymagany oraz przedstawiany na laboratorium postanowiłem dołączyć go do projektu w celu bliższego zaznajomienia się z tą często stosowaną konwencją.

6.Testowanie

Program został wielokrotnie przetestowany zarówno pod kątem kompatybilności różnego rodzaju plików (zarówno tekstowych jak i JSON) jak również prób wprowadzania informacji mogących spowodować potencjalne błędy.

Testowanie zostało wielokrotnie przeprowadzone przeze mnie jak i przez osoby które nigdy nie mały z nim styczności z owym kodem w celu znalezienia ewentualnych problemów i błędów w komunikacji program-użytkownik.

Testy przebiegały na różnych komputerach w celu sprawdzenia kompatybilności z większością systemów operacyjnych Windows oraz poprawnego otwierania się okna i funkcjonowania aplikacji.

Podczas testów zostało wyeliminowanych wiele błędów z których najbardziej uciążliwe były:

- niepoprawne określanie punktu odniesienia dla figur symetrycznych takich jak kwadrat, okrąg lub trójkąt.
- odwrotna interpretacja odtworzonej listy kształtów z pliku JSON co powodowało wygenerowanie innego obrazu
- crashowanie aplikacji przy generowaniu „wirtualnego kształtu" pojawiającego się przed puszczeniem przycisku myszy.

Program został przetestowany pod kątem wycieków pamięci.

7.Wnioski

Projektowanie opisanej powyżej aplikacji pomogło mi lepiej zrozumieć wiele zaawansowanych technologii języka C++ takich jak RTTI, szablony czy smart pointery, dokładniej zapoznać się z projektowaniem programu z pełni zorientowanego obiektowo z wykorzystaniem 4-poziomowego dziedziczenia i pełnym polimorfizmem.
Zaaplikowanie zapisu i odczytu plików JSON również pomogło mi lepiej poznać owy sposób opisu danych który z pewnością użyję w moich przyszłych projektach.