Akademia Górniczo-Hutnicza w Krakowie

Wydział Fizyki i Informatyki Stosowanej

Podstawy Grafiki Komputerowej

Projekt 34: Program do rasteryzacji grafiki wektorowej

Andrzej Świętek
Marcin Knapczyk
Mateusz Wawrzyczek

10.06.2024

Spis treści

1 Opis projektu

Celem projektu jest stworzenie programu do rasteryzacji grafiki wektorowej. Program ten nie będzie pozwalał na tworzenie tejże grafiki lecz tylko wczytywał gotowe prace stworzone przy użyciu innego programu.

2 Założenia wstępne przyjęte w realizacji projektu

Po wczytaniu rysunku jest on wyświetlany w oknie aplikacji. Użytkownik może sprawdzić wszystkie parametry figur w postaci listy, po kliknięciu w odpowiedni przycisk. Klient ma możliwość zmiany koloru linii, koloru wypełniania, obrócenia elementu oraz położenia wierzchołków (odbywa się to poprzez ręczne wpisanie wartości do odpowiednich pól). Dodatkowo możliwe jest zarządzanie położeniem obiektów. Po ewentualnym dokonaniu zmian użytkownik ma możliwość wyboru rozmiaru i zapisania obrazka jako plik BMP.

3 Analiza projektu

3.1 Przechowywanie danych

Dane do wczytania będą zapisane w pliku z rozszerzeniem .xml. Będą tam podane informacje, tj. kolor, kształt, współrzędne wierzchołków (współrzędne środka i promień w przypadku koła). W programie będzie możliwość zapisu danych w podobnej formie, do danych wejściowych, a także wygenerowanie pliku BMP. Informacje na temat kształtów będą zapisywane w klasie Shape, a także w klasach pochodnych odpowiadającym poszczególnym figurom.

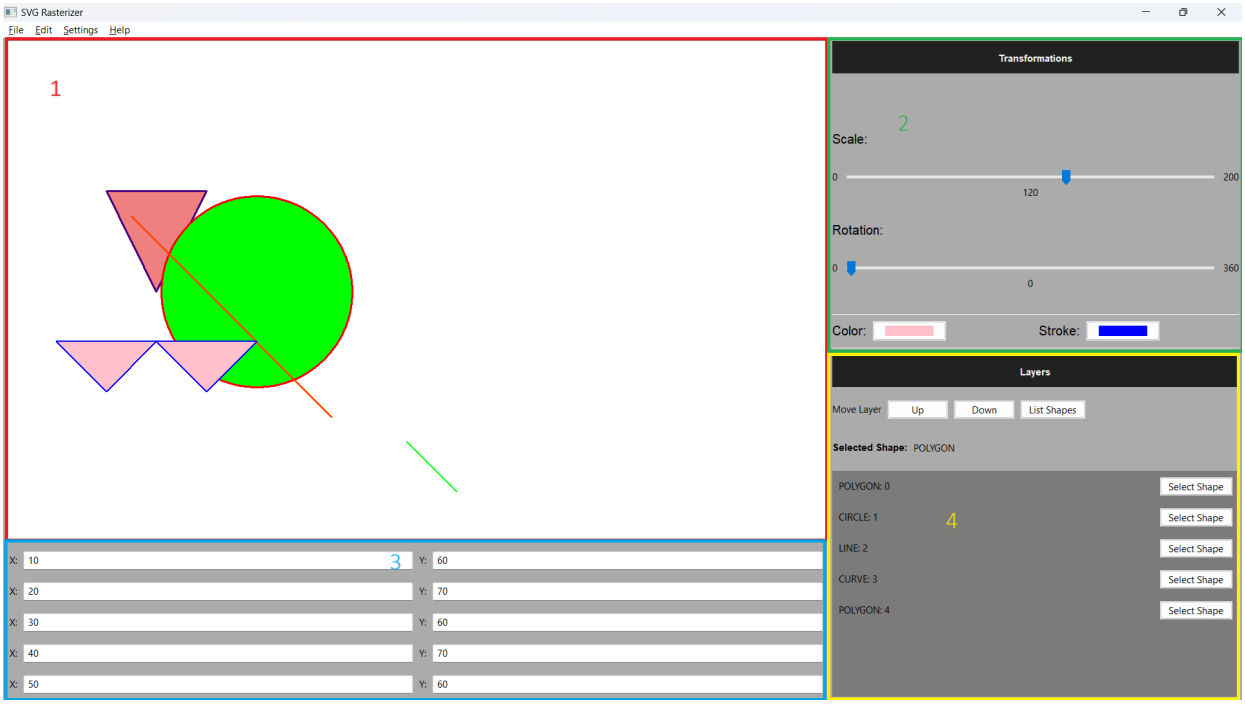
```
<?xml version="1.0" encoding="UTF-8"?>
<image figure-count="5">
    <polygon n="5" stroke="5" outline="255,0,255" fill="66, 135, 245">
        <point x="1" y="3" />
        <point x="2" y="4" />
        <point x="10" y="5" />
        <point x="30" y="6" />
        <point x="50" y="7" />
    </polygon>
    <circle x="5" y="4" r="5" stroke="10" outline="0,255,255" fill="252,186,3" />
    <line x="30" y="30" stroke="5" outline="100,200,50" fill="0,0,0">
        <point x="20" y="20" />
        <point x="40" y="40" />
    </line>
    <curve n="3" stroke="5" outline="255,255,0" fill="165,252,3">
        <point x="10" y="30" />
        <point x="20" y="3" />
        <point x="50" y="10" />
    </curve>
</image>
```

Listing 1: Przykład danych wejściowych w postaci pliku XML

3.2 Struktura projektu

Projekt został podzielony na 3 części. Pierwsza ma za zadanie uploadowanie danych do programu, ich eksportowanie i zapisywanie. Kolejna część odpowiada za przechowywanie danych o konkretnych figurach i ich edycje. Ostatni segment odpowiada za interfejs programu (część do wyświetlania obrazu, odpowiednie slidery, przyciski i pola tekstowe), także ma za zadanie pobieranie danych o zmianach od użytkownika i przekazywanie ich do figur. Dodatkowo w strukturze programu znalazł się folder DOC z dokumentacją, oraz folder EXAMPLE z przykładami.

3.3 Interfejs użytkownika



Rysunek 1: Okno programu

W górnej części programu znajduje się menu. W zakładce file użytkownik ma dostęp do zarządzania plikami (zapis, otwarcie). Edit pozwala na przesuwanie warstw. Settings and help, w przypadku rozwoju programu umożliwią zmianę ustawień oraz dostęp do pomocy i dokumentacji.
W części oznaczonej numerem 1 prezentowana jest grafika. Część 2 odpowiada za zmianę rozmiaru, obrót kształtu, a także zmianę koloru obramowania i wypełnienia. Segment 3 to zmiana położenia wierzchołków kształtu. Funkcja działa tylko w momencie wybrania kształtu. Wybór kształtu odbywa się poprzez użycie przycisku Select Shape. Przciski up, down przesuwają kształty między sobą. List Shape wyświetla wszystkie informacje o kształtach.

3.4 Narzędzie potrzebne do stworzenia projektu

Do realizacji projektu wybrano środowisko programistyczne Visual Studio, które oferuje zaawansowane narzędzia do debugowania, edytowania kodu oraz zarządzania projektem. W projekcie wykorzystano również bibliotekę wxWidgets oraz TinyXML-2.

Biblioteka wxWidgets została użyta do stworzenia interfejsu użytkownika, w tym ikon, suwaków, przycisków, pól tekstowych oraz menu. Dzięki swojej elastyczności i bogatej funkcjonalności, wxWidgets umożliwia tworzenie aplikacji z przyjaznym i intuicyjnym interfejsem.

Biblioteka TinyXML-2 służy do przetwarzania plików XML. Jest wykorzystywana do otwierania i parsowania plików XML, a w przyszłości może być również używana do obsługi plików SVG. TinyXML-2 uczestniczy w procesie parsowania danych wejściowych, zapewniając efektywne zarządzanie strukturą danych i ich integralność.

Takie połączenie narzędzi i bibliotek zapewnia solidną podstawę dla rozwoju projektu, umożliwiając łatwe zarządzanie kodem, efektywne tworzenie interfejsu użytkownika oraz bezproblemowe przetwarzanie danych wejściowych.

4 Podział zadań

- Andrzej Świętek - architektura projektu, wczytywanie danych, obsługa i funkcjonalności związane z warstwami, kluczowe mechanizmy tj. wybieranie kształtu/warstwy, menu itd.
- Marcin Knapczyk - rysowanie obiektów, ich obracanie, zmiana położenia wierzchołków, zapisywanie do pliku .bmp
- Mateusz Wawrzyczek - inne transformacje, obsługa zmiany kolorów tła kształtu i jego obramowania, dokumentacja

Estymacja czasowa zadań

Zadanie	**Odpowiedzialna osoba**	**Podzadania**	**Czas (w dniach)**
Stage 1			
Strukturka plików projektu	Andrzej	• Utworzenie głównego katalogu projektu • Struktura katalogów (src, include, assets, etc.) • Skonfigurowanie plików CMakeLists	2
GUI z form buildera	Andrzej	• Zaprojektowanie podstawowego okna aplikacji • Utworzenie paneli i menu • Integracja z kodem	3
Data Loader - wczytanie danych z pliku	Andrzej	• Implementacja klasy DataLoader • Implementacja XMLDataLoaderAdapter i SVGDataLoaderAdapter • Testy jednostkowe	4
Rysowanie Shape'ów na panelu, Rotate	Marcin	• Implementacja rysowania Shape'ów • Obsługa zdarzeń Rotate • Aktualizacja GUI	4

Kontynuacja na następnej stronie

Kontynuacja z poprzedniej strony

Zadanie	**Odpowiedzialna osoba**	**Podzadania**	**Czas (w dniach)**
Transformacje (Resize, Scale, color, stroke) Funkcje w klasach + eventy	Mateusz	• Implementacja funkcji transformacji w klasach Shape • Obsługa eventów dla transformacji • Integracja z GUI	5
Logger	Andrzej	• Implementacja klasy Logger • Integracja z resztą aplikacji • Testy jednostkowe	2
Stage 2			
Warstwy + zmiana kolejności	Andrzej	• Implementacja systemu warstw • Funkcje do zmiany kolejności warstw • Integracja z GUI	4
Pozycje wertexów	Marcin, Andrzej	• Implementacja funkcji zmiany pozycji wierzchołków • Obsługa eventów • Integracja z GUI	4
Inne opcje menu	Andrzej	• Dodanie dodatkowych opcji w menu • Integracja z resztą aplikacji	3
Dodatkowe funkcjonalności	Andrzej	• Implementacja dodatkowych funkcji aplikacji	3

Kontynuacja na następnej stronie

Kontynuacja z poprzedniej strony

Zadanie	**Odpowiedzialna osoba**	**Podzadania**	**Czas (w dniach)**
Zapisywanie	Andrzej, Marcin	• Implementacja funkcji zapisywania obrazu jako BMP • Testy jednostkowe	3
Przygotowanie kodu na produkcję	Andrzej, Marcin	• Naprawienie błędów • Stworzenie pliku wykonywalnego	3

5 Opracowanie i opis niezbędnych algorytmów

Do obracania obiektów potrzebna jest macierz przekształcenia:

$$\begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Wykorzystano ją w polimorficznych funkcjach Shape::Draw().
Skalowanie kształtów odbywa się w funkcji 'Line::draw' poprzez przeliczenie współrzędnych punktów końcowych z uwzględnieniem współczynnika skalowania oraz rozmiaru okna. Na początku określany jest skaler, czyli mniejszy z wymiarów okna. Następnie obliczany jest współczynnik skalowania jako wartość procentowa podzielona przez 100.
Punkty początkowy i końcowy linii są przesuwane względem środka linii i transformowane za pomocą macierzy rotacji. Współrzędne wynikowe są skalowane przez współczynnik skalowania i przeliczane względem rozmiaru okna. Przeskalowane punkty są używane do rysowania linii na kontekście graficznym 'dc', co zapewnia odpowiednie dopasowanie linii do wymiarów okna.
Do skalowania obiektów w dwóch wymiarach stosujemy macierz skalowania:

$$\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

gdzie S_x i S_y są współczynnikami skalowania w osi x i y odpowiednio.
Do przesuwania (translacji) obiektów w dwóch wymiarach stosujemy macierz translacji:

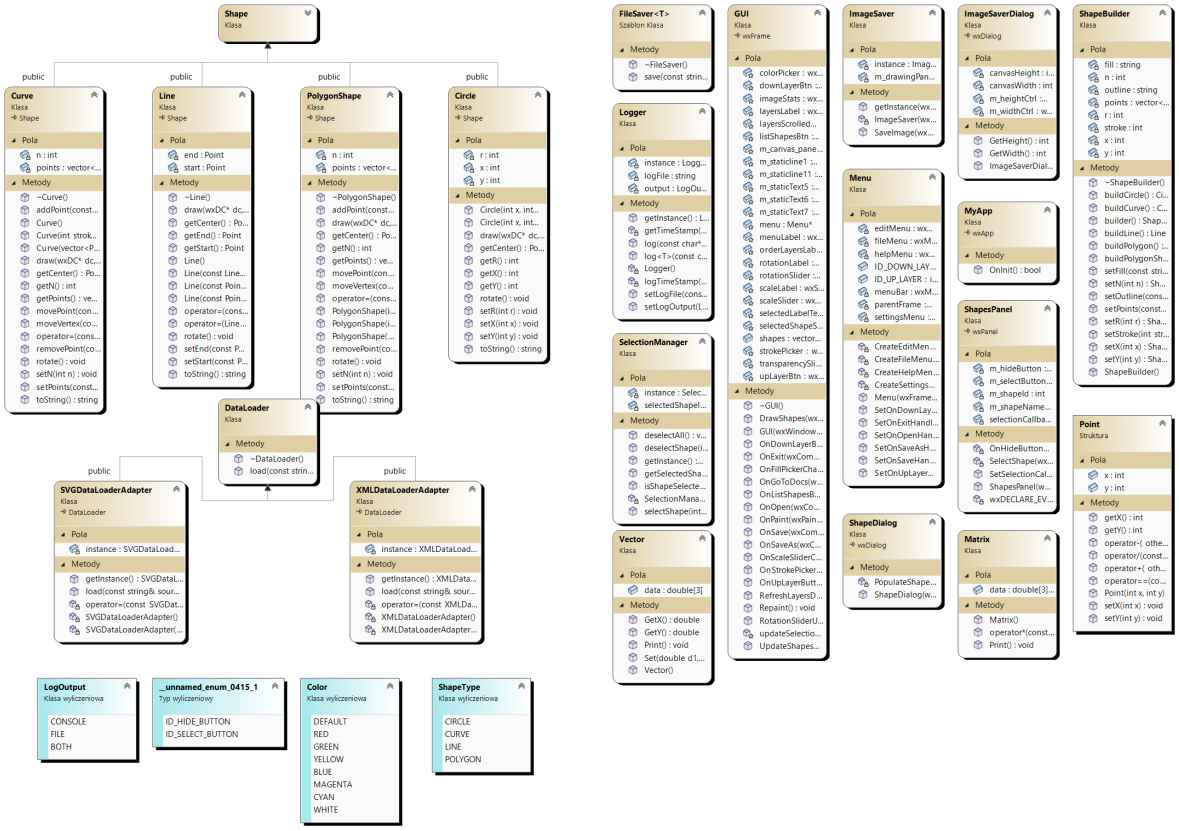
$$\begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{pmatrix}$$

gdzie T_x i T_y są wartościami przesunięcia w osi x i y odpowiednio.
Kombinacja wszystkich tych przekształceń może być uzyskana poprzez ich mnożenie. Na przykład, najpierw skalowanie, potem rotacja, a na końcu translacja:

$$\begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Macierze przekształceń są podstawowymi narzędziami stosowanymi do manipulacji kształtami w grafice komputerowej, umożliwiając obracanie, skalowanie i przesuwanie obiektów w sposób precyzyjny i kontrolowany.

6 Kodowanie



Rysunek 2: Schemat prezentujący hierarchię klas w programie

Na powyższym schemacie zaprezentowano hierarchię klas w programie. Klasa Shape przechowuje dane o kształcie (fill - kolor wypełnienia, outline - kolor obramowania, rotationAngle - kąt obrotu itp.). Klasy kształtów dziedziczące po Shape (Curve, Line, PolygonShape, Circle), przechowują bardziej wyspecjalizowane dane, takie jak lista wierzchołków wielokąta, środek i promień okręgu, czy punkty kontrolne krzywej, oraz definiują polimorficzne metody rysowania kształtów Shape::Draw().
DataLoader, a także klasy pochodne SVGDataLoaderAdapter, XMLDataLoaderAdapter pozwalają na uploadowanie danych do programu.
Klasa GUI, a także Menu odpowiadają za buttony, slidery, a także górne menu aplikacji. SelctionManager pozwala na zarządzanie warstwami przez użytkownika.

Dane z klasy DataLoader są zapisywane w klasie Shape i pochodnych klasach kształtów. Edycja danych (po zmianach wprowadzonych przez użytkowika) odbywa się poprzez zmianę danych przechowywanych w obiektach figur.

SelectionManger:
Kluczowym dla funkcjonowania całej aplikacji była implementacja sprawnego i skalowalnego systemu do obsługi wybierania przez użytkownika figury/warstwym na której w danej chwili chciałby pracować. Nasza implementacja opiera się na klasie singleton, która zapewnia dostęp do tej samej instancji obiektu dla tej konkretnej aplikacji. Obiekt w podstawowej implementacji jaka trafiła do wersji produkcyjnej zapewnia jedynie wybieranie indeksu kształtu w wektorze kształtów przechowywanym w klasie GUI.h. W każdym miejscy w aplikacji gdzie wymagany jest dostęp do wybranej figury następuje pobranie instancji klasy a następnie wykanie metody typu getter czy setter w zależności od potrzeby.

ImageSaver, oraz ImageSaverDialog zarządzają zapisywaniem gotowego obrazu.

```cpp
void ImageSaver::SaveImage(wxWindow* parent) {

    // get the size of the drawing panel
    wxSize panelSize = m_drawingPanel->GetSize();
    int panelWidth = panelSize.GetWidth();
    int panelHeight = panelSize.GetHeight();

    // launch prompt dialog
    ImageSaverDialog dlg(parent, panelWidth, panelHeight);
    if (dlg.ShowModal() == wxID_OK) {
        int targetWidth = dlg.GetWidth();
        int targetHeight = dlg.GetHeight();

        wxFileDialog saveFileDialog(parent, _("Save BMP file"), "", "", "BMP files (*.bmp)|*.bmp", wxFD_SAVE | wxFD_OVERWRITE_PROMPT);
        if (saveFileDialog.ShowModal() == wxID_CANCEL) {
            return;
        }

        wxString filePath = saveFileDialog.GetPath();

        // create a bitmap with the size of the drawing panel
        wxBitmap bitmap(panelWidth, panelHeight);
        wxMemoryDC memDC(bitmap);
        memDC.SetBackground(*wxWHITE_BRUSH);
        memDC.Clear();

        // Blit the content of the drawing panel to the memory DC
        wxClientDC dc(m_drawingPanel);
        m_drawingPanel->PrepareDC(dc);
        memDC.Blit(0, 0, panelWidth, panelHeight, &dc, 0, 0);

        memDC.SelectObject(wxNullBitmap);

        // rescale the bitmap to the target size
        wxImage image = bitmap.ConvertToImage();
        image.Rescale(targetWidth, targetHeight);

        // save the rescaled image as a BMP file
        image.SaveFile(filePath, wxBITMAP_TYPE_BMP);
        wxMessageBox("Image saved successfully", "Info", wxOK | wxICON_INFORMATION);
    }
}
```

Listing 2: Algorytm zapsiu do pliku i ustalenia końcowych wymiarów docelowej mapy bitowej

Funkcja SaveImage klasy ImageSaver jest odpowiedzialna za zapisanie zawartości panelu rysunkowego do pliku BMP. Proces rozpoczyna się od pobrania rozmiaru panelu rysunkowego, co pozwala na utworzenie bitmapy o odpowiednich wymiarach. Następnie wyświetlany jest dialog, który umożliwia użytkownikowi określenie docelowych wymiarów obrazu oraz wybór lokalizacji zapisu pliku.
Po potwierdzeniu przez użytkownika, funkcja tworzy bitmapę i za pomocą kontekstu pamięci kopiuje zawartość panelu rysunkowego. Bitmapa jest następnie konwertowana na obiekt wxImage, który zostaje przeskalowany do wybranych przez użytkownika wymiarów. Końcowym etapem jest zapisanie przeskalowanego obrazu do pliku BMP i poinformowanie użytkownika o sukcesie operacji za pomocą okna komunikatu.
Kod korzysta z istniejących komponentów i funkcji wxWidgets, takich jak wxFileDialog, wxMemoryDC i wxImage, do obsługi zapisywania plików, rysowania i manipulacji obrazami. Dzięki wykorzystaniu tych wbudowanych klas i metod, kod unika ponownego implementowania powszechnych funkcji, przestrzegając zasady "no reinvent the wheel".

OnDownLayerButtonClick i OnUpLayerButtonClick pozwalają na zamianę miejscami warstw (kształtów). Pozycja warstw przechowywana jest w wektorze. Każde użycie przycisku powoduje zamianę miejscami indeksów kształtów w wektorze.

Dokumentacja do programu została stworzona przy pomocy Doxygena. W folderze DOC dostępna jest dokumentacja w html-u.

7 Wzorce projektowe

W naszym projekcie zostały użyte liczne wzorce projektowe mające na celu ustandaryzowanie sposobu pracy i odnoszenia się w kodzie do poszczególnych składników aplikacji. Zabieg ten pozwolił znacząco zredukować ilość nieuporządkowanego kodu na rzecz prostych i schludnych klas zgodnych między innymi z takimi paradygmatami jak KISS, DRY i SOLID.

7.1 Factory-Builder

Factory Builder to wzorzec, który łączy elastyczność konfiguracji (charakterystyczną dla Buildera) z możliwością tworzenia różnych typów obiektów (charakterystyczną dla Factory). Oznacza to, że masz interfejs do ustawiania właściwości obiektu krok po kroku (Builder), a na końcu możesz wywołać odpowiednią metodę fabryczną, aby utworzyć różne typy obiektów (Factory).

Miejsce w Wzorcach Projektowych
Klasa ShapeBuilder wpisuje się w wzorzec Factory Builder, który łączy cechy dwóch innych wzorców projektowych:

- Builder: Umożliwia stopniowe ustawianie właściwości obiektu poprzez metody łańcuchowe (setX, setY, setR, setPoints, setStroke, setOutline, setFill). Dzięki temu można tworzyć złożone obiekty w sposób czytelny i modularny.
- Factory: Dostarcza metody budujące (buildCircle, buildLine, buildPolygonShape), które tworzą różne typy obiektów bez konieczności bezpośredniego używania ich konstruktorów. Pozwala to na elastyczne tworzenie obiektów o różnych typach.

Zalety Wzorca Factory Builder:
Wykorzystanie wzorca Factory Builder w klasie ShapeBuilder niesie za sobą wiele zalet:

1. Czytelność i Modularność: Dzięki metodom łańcuchowym można łatwo i czytelnie konfigurować obiekty. Każda właściwość obiektu może być ustawiona w osobnej metodzie, co poprawia czytelność kodu.
2. Elastyczność: Możliwość tworzenia różnych typów obiektów (takich jak Circle, Line, PolygonShape) za pomocą jednej klasy budującej. To eliminuje potrzebę tworzenia oddzielnych klas fabrycznych dla każdego typu obiektu.
3. Reużywalność: ShapeBuilder może być ponownie użyty do tworzenia różnych obiektów z różnymi konfiguracjami, co zwiększa reużywalność kodu i zmniejsza duplikację.
4. Łatwe Zarządzanie Zmianami: W przypadku dodania nowej właściwości do kształtu, wystarczy dodać nową metodę ustawiającą w klasie ShapeBuilder, bez konieczności modyfikowania konstruktorów poszczególnych klas kształtów.
5. Testowalność: Ułatwia testowanie poprzez umożliwienie łatwego tworzenia i konfigurowania obiektów testowych z różnymi zestawami danych.

Klasa ShapeBuilder jest więc eleganckim i efektywnym rozwiązaniem, które łączy elastyczność i czytelność wzorca Builder z możliwościami wzorca Factory, umożliwiając łatwe i elastyczne tworzenie różnorodnych obiektów graficznych.

```
1     Circle circle = ShapeBuilder().builder()
2                                   .setX(5)
3                                   .setY(5)
4                                   .setR(10)
5                                   .setStroke(2)
6                                   .setOutline("red")
7                                   .setFill("blue")
8                                   .buildCircle();
```

Listing 3: Przykład użycia ShapeBuilder - Koło

```
PolygonShape polygon = ShapeBuilder()
                            .setN(5)
                            .setPoints(points)
                            .setStroke(3)
                            .setOutline("green")
                            .setFill("yellow")
                            .buildPolygonShape();
```

Listing 4: Przykład użycia ShapeBuilder - Wielokąt

7.2 Singleton

Singleton jest kreacyjnym wzorcem projektowym, który pozwala zapewnić istnienie wyłącznie jednej instancji danej klasy. Ponadto daje globalny punkt dostępowy do tejże instancji. Singleton jest używany, gdy trzeba zagwarantować, że w aplikacji będzie tylko jedna instancja obiektu i dostęp do niej musi być łatwy i globalny.

Wzorzec ten jest często stosowany do zarządzania zasobami, konfiguracją aplikacji oraz innymi elementami, które muszą być jednolite w całej aplikacji.

- **ImageSaver**:
  Klasa odpowiedzialna za zapisywanie obrazów. Wzorzec Singleton gwarantuje, że tylko jedna instancja tej klasy będzie zajmować się operacjami zapisu, co zapobiega potencjalnym konfliktom i niespójnościom podczas zapisu obrazów z różnych miejsc w aplikacji.
- **SVGDataLoaderAdapter**:
  Klasa, która ładuje dane w formacie SVG. Dzięki zastosowaniu Singletona, zapewnia jednolity sposób ładowania danych SVG w całej aplikacji. Gwarantuje to spójność oraz łatwiejsze zarządzanie pamięcią i zasobami.
- **XMLDataLoaderAdapter**:
  Podobnie jak w przypadku *SVGDataLoaderAdapter*, ta klasa obsługuje ładowanie danych w formacie XML. Wzorzec Singleton zapewnia, że cała aplikacja korzysta z tej samej instancji do ładowania danych XML, co zwiększa efektywność i minimalizuje ryzyko błędów.
- **Logger**:
  Klasa odpowiedzialna za logowanie informacji, błędów i ostrzeżeń w aplikacji. Singleton zapewnia, że wszystkie części aplikacji logują informacje do tego samego źródła, co ułatwia debugowanie i monitorowanie stanu aplikacji.
- **SelectionManager**:
  Klasa zarządzająca zaznaczeniami w aplikacji (np. zaznaczenie obiektów na ekranie). Singleton gwarantuje, że tylko jedna instancja zarządza wszystkimi zaznaczeniami, co zapewnia spójność i eliminuje problemy związane z wielokrotnymi zaznaczeniami lub konfliktem pomiędzy różnymi instancjami zarządzającymi zaznaczeniami. Przy obecnej implementacji klasa ta służy głównie do obsługi zaznaczanej figury, lecz struktura aplikacji pozwala na modyfikacje i dodatkowe funkcjonalności.

```
const int selected_index = SelectionManager::getInstance()
                                              ->getSelectedShapeIndex();
```

Listing 5: Przykład użycia SelectionManager

7.3 Adapter

Adapter jest strukturalnym wzorcem projektowym, który pozwala obiektom o niekompatybilnych interfejsach współpracować ze sobą. Umożliwia to klasom działanie razem, mimo że ich interfejsy są różne. Adapter opakowuje obiekt o niekompatybilnym interfejsie wewnątrz klasy, która wystawia zgodny interfejs.

W projekcie, klasy *XMLDataLoaderAdapter* i *SVGDataLoaderAdapter* działają jako adaptery, które dostosowują różne formaty danych (XML i SVG) do wspólnego interfejsu DataLoader. Każda z tych klas zaimplementowała metodę load, która wczytuje dane z odpowiedniego źródła, parsuje je i zwraca jako kolekcję obiektów *Shape*.
Zastosowanie wzorca Adapter w połączeniu ze wzorcem Singleton przynosi kilka korzyści:

1. Ujednolicenie interfejsu: Dzięki zastosowaniu wspólnego interfejsu DataLoader, różne formaty danych mogą być ładowane w ten sam sposób, co upraszcza kod aplikacji.
2. Łatwość rozbudowy: Dodanie obsługi nowego formatu danych wymaga jedynie stworzenia nowego adaptera implementującego interfejs DataLoader, co czyni system łatwym do rozbudowy.
3. Efektywne zarządzanie zasobami: Dzięki wzorcowi Singleton każda klasa adaptera ma tylko jedną instancję, co zmniejsza zużycie zasobów i zapobiega potencjalnym konfliktom.
4. Łatwość utrzymania: Oddzielenie kodu parsowania danych od reszty aplikacji sprawia, że kod jest bardziej modularny i łatwiejszy w utrzymaniu oraz testowaniu.

Dzięki tym wzorcom projektowym aplikacja jest bardziej elastyczna, rozszerzalna i łatwiejsza w zarządzaniu, co znacząco poprawia jej jakość i stabilność.

- **XMLDataLoaderAdapter**: Adapter odpowiedzialny za wczytywanie danych z plików XML. Singleton zapewnia, że cała aplikacja korzysta z tej samej instancji do ładowania danych XML, co zwiększa efektywność i minimalizuje ryzyko błędów. Implementacja metody load wykorzystuje bibliotekę TinyXML2 do parsowania danych XML i tworzenia obiektów Shape.
- **SVGDataLoaderAdapter**: Adapter odpowiedzialny za wczytywanie danych z plików SVG. Singleton zapewnia, że cała aplikacja korzysta z tej samej instancji do ładowania danych SVG, co gwarantuje spójność i łatwiejsze zarządzanie zasobami. Implementacja metody ‘load‘ wykorzystuje bibliotekę TinyXML2 do parsowania danych SVG i tworzenia obiektów ‘Shape‘.

Takie podejście pozwala nam wprowadzać dodatkowe formaty plików jakie aplikacja jest w stanie obsłożyć bez jakiekolwiek formy ingerencji w już istniejący kod.

8 Testowanie

- Testy Automatyczne
  Głównym elementem, który wymagał testowania były wczytywanie danych do programu. Należało sprawdzić poprawność wczytywania danych dla różnych plików. Do tego celu w projekcie wykorzystano framework dostarczony przez firmę Mircosoft zintegrowany z IDE Visual Studio. Testy w sposób zakładają załadowanie figury w postaci pliku XML z wszystkimi możliwymi parametrami jakie dana figura posiada by następnie sprawdzić czy jej abstrakcyjna repreznatacja w postaci objektu odpowiedniej klasy ma takie same właściwości.
- Testy manualne
  Przez wzgląd na szybkie tempo rozwoju projektu wraz z przyjętą przez nas metodyką AGILE i częste zmiany początkowego projektu aplikacji na rzecz jej funkcjonalności testy automatyczne okazywały się szybko przedawniać i dezaktualizować.
  Aby mimo tego utrzymać wysoki poziom integralności i odporności na błędy aplikacji przygotowaliśmy zestawy testowe plików .xml. Po każdej znaczącej zmianie i przed każdym pull requestem przeprowadzane były testy regresji mające na celu ustalenie czy nowe zmiany nie spowodowały uszkodzenia już istniejących funkcjonalności. Testowaniem były objęte głównie obsługa plików i obsługa kontrolek interfejsu użytkownika.

9 Wdrożenie, raporty i wnioski

Udało się zrealizować założenia podstawowe projektu. Dodatkowo zrealizowano podstawowe transformacje, takie jak rotacja i skalowanie, zarządzanie warstwami i zmiana położenia warstwy. Program umożliwia w przyszłości dodanie dodatkowych funkcjonalności, takich jak ustawienie przeźroczystości albo chowanie warstw - funkcjonalności które już są częściowo lub całkowicie zaimplmentowane, lecz nie testowane. Użycie wzorców projektowych pozwala na szybkie i intuicyjne wprowadzanie zmian w przyszłości i wzbogacać aplikację o nowe funkcjonalności z możliwie małym nakładem pracy i czasu.

10 Bibliografia

- Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- wxWidgets Documentation. (2023). https://docs.wxwidgets.org/
- Microsoft Visual Studio Documentation. (2023). https://docs.microsoft.com/en-us/visualstudio/
- TinyXML-2 Documentation. (2023). https://leethomason.github.io/tinyxml2/annotated.html