

Dokumentacja systemu VecEdit

Zespół projektowy:

Adam Rogowski, Szymon Sawoń, Bartosz Siemaszkiewicz

13 stycznia 2025

Spis treści

1	Wstęp	1
2	Wzorce projektowe i ich zastosowanie	1
2.1	Kompozyt (<i>Composite</i>)	1
2.2	Metoda wytwórcza (<i>Factory Method</i>)	2
2.3	Strategia (<i>Strategy</i>)	2
2.4	Adapter	3
2.5	Odwiedzający (<i>Visitor</i>)	3
2.6	Prototyp (<i>Prototype</i>)	3
3	Opis użytych bibliotek i narzędzi	4
3.1	Raylib (5.5)	4
3.2	RayGui	4
3.3	C++20 i CMake	4
4	Instrukcja użytkownika	4
4.1	Uruchomienie programu	4
4.2	Interfejs programu — krótki opis	5
4.3	Podstawowe operacje	5
5	Instrukcja instalacji i kompilacji	6
6	Podział pracy w zespole	6
7	Podsumowanie	6

1 Wstęp

VecEdit to aplikacja (napisana w **C++20**) służąca do edycji prostych grafik wektorowych. Umożliwia rysowanie, modyfikowanie, grupowanie oraz klonowanie obiektów graficznych (takich jak prostokąty, okręgi czy wielokąty). Program korzysta z biblioteki **raylib** (w wersji 5.5) do wyświetlania okna, rysowania figur na ekranie oraz pobierania zdarzeń wejściowych od użytkownika (klawiatura, mysz). Do tworzenia interfejsu graficznego w trybie *immediate mode* używana jest biblioteka **RayGui**.

Aplikacja pozwala na:

- rysowanie i edycję kształtów (w tym zmianę koloru, obrysu, przezroczystości),
- przesuwanie i skalowanie figur,
- grupowanie obiektów (`FigureGroup`),
- cofanie (`undo`) i ponawianie (`redo`) operacji dzięki wzorcowi *Command*,
- zapisywanie projektów do plików wektorowych (`SVG`) i *bitmapowych* (`PNG`),
- otwieranie, zamykanie i przełączanie się między wieloma zakładkami dokumentów.

2 Wzorce projektowe i ich zastosowanie

Poniżej przedstawiono zastosowane wewnątrz aplikacji wzorce projektowe, z uwzględnieniem istotnych elementów w kodzie oraz komentarzy dotyczących możliwych zmian (tzw. *wektora zmian*).

2.1 Kompozyt (*Composite*)

- **Cel:** Umożliwienie traktowania pojedynczych obiektów (figury) i grup obiektów (grupy figur) w sposób jednolity, dzięki interfejsowi `Figure`.
- **Struktura:**
 - `Figure` – interfejs *komponentu*. Posiada m.in. metody typu `accept(visitor)`, które w implementacjach *Composite* przekazują wywołanie wszystkim dzieciom.
 - `FigureGroup` – *kompozyt*, zawiera wektor `children` oraz metody `addChild(...)` itp.
 - `RectFigure`, `CircleFigure`, `PolyFigure` – *konkretne komponenty (liście)*, nie posiadają dzieci.
- **Użycie:**
 - W edytorze (`Editor`) przy `groupFigures()` i `ungroupFigures()` łączymy/rozbijamy figury w drzewiastą strukturę.
 - `accept(visitor)` w `FigureGroup` wywołuje `accept` u wszystkich dzieci, co integruje się z *Visitor*.
- **Wektor zmian:** Dodanie kolejnych typów figur nie wymaga zmian w `FigureGroup`, wystarczy zaimplementować te same metody `Figure`.

2.2 Metoda wytwórcza (*Factory Method*)

- **Cel:** Abstrakcja sposobu tworzenia obiektów. U nas służy głównie do tworzenia `PointEditor` w metodzie `Figure::makePointEditor()`.
- **Struktura i użycie:**
 - `Figure` deklaruje wirtualną metodę `makePointEditor()`, która w podklasach (`RectFigure`, `CircleFigure`, ...) zwraca odpowiedni edytor (np. `RectFigurePointEditor`).

- Editor korzysta z tej metody, by pobierać adapter do edycji punktów figury, nie wiedząc *jaki to* konkretnie edytor.
- **Wektor zmian:** Nowy typ figury (np. `StarFigure`) wystarczy wyposażać w swoją implementację `makePointEditor()`, zwracając własny typ `StarFigurePointEditor`.

2.3 Strategia (*Strategy*)

- **Cel:** Definiowanie wymiennych algorytmów/akcji, które można przypisać do przycisków lub skrótów klawiaturowych (np. *undo*, *redo*).
- **Kluczowe elementy:**
 - `Strategy<...>` – interfejs (plik `Strategy.h`),
 - `UndoStrategy`, `RedoStrategy`, `SetSelectStrategy` itp. – konkretne strategie,
 - `FunctorStrategy` – pozwala zdefiniować strategię w oparciu o lambda/domykanie (*closure*), bez tworzenia nowej podklasy.
- **Użycie:**
 - W `AppUi` do `IconButton` lub `KeyboardShortcut` przypisujemy strategię. Np. `auto str = std::make_shared<UndoStrategy>(editor)`, a w innym miejscu `addShortcut(str, KEY_Z, mod)`.
 - W `FunctorStrategy` wystarczy przekazać lambda, np. `FunctorStrategy<>{[this]() { editor->exportDocument("png"); }}`.
- **Wektor zmian:** Aby dodać nową prostą akcję, można utworzyć instancję `FunctorStrategy` z odpowiednim wyrażeniem lambda; dla bardziej złożonych przypadków — osobną klasę dziedziczącą po `Strategy`.

2.4 Adapter

- **Cel:** Udostępnienie wspólnego interfejsu edycji punktów figur (`PointEditor`) mimo iż każda figura ma odmienne właściwości.
- **Struktura i użycie:**
 - `PointEditor` – interfejs z metodami `getEditPoints()` i `updatePointPosition(...)`,
 - `RectFigurePointEditor`, `CircleFigurePointEditor`, `PolyFigurePointEditor` – adaptery konkretnych figur,
 - Każdy adapter *zwraca* (np. w `getEditPoints()`) położenia uchwytów charakterystyczne dla swojej figury (np. promień dla koła).
- **Wektor zmian:** Dodanie nowego kształtu wymaga stworzenia nowego adaptera do edycji, a sama logika `Editor` (obsługująca `PointEditor`) nie musi być zmieniana.

2.5 Odwiedzający (*Visitor*)

- **Cel:** Oddzielenie logiki przetwarzania figur (np. rysowania, zapisu do pliku) od samych klas figur.
- **Struktura i użycie:**
 - `FigureVisitor` – interfejs odwiedzającego,
 - `RendererVisitor` (renderuje figury na ekranie),
 - `SvgSerializerVisitor` (zapisuje wektorowo do pliku SVG),
 - `BitmapRendererVisitor` (obsługa zapisu PNG).
 - Metoda `accept(visitor)` w `Figure` (bądź `FigureGroup`) wywołuje `visitor.visit(...)` odpowiedniego typu.
- **Wektor zmian:** Dodanie kolejnej operacji (np. liczenie pola figur) wymaga utworzenia nowej klasy odwiedzającej, bez modyfikacji istniejących figur.

2.6 Prototyp (*Prototype*)

- **Cel:** Możliwość klonowania obiektów (figur) bez tworzenia zależności pomiędzy kodem wywołującym a ich konkretną klasą.
- **Struktura i użycie:**
 - `FigureBase<FigureType>` posiada wirtualną metodę `clone()`, a w podklasach (`RectFigure`, `CircleFigure` itp.) implementuje tworzenie kopii *danego* typu.
 - W `Editor::processModeInsert()` tworzymy nową figurę poprzez sklonowanie istniejącego prototypu (figura-prototyp jest wcześniej ustawiona przez `SetFigureInsertStra`).
- **Wektor zmian:** Każda klasa dziedzicząca z `FigureBase` automatycznie dziedziczy wzorec klonowania; wystarczy zaimplementować własne `clone()` (często minimalna zmiana).

3 Opis użytych bibliotek i narzędzi

3.1 Raylib (5.5)

W projekcie korzystamy z **raylib** w wersji 5.5, która zapewnia:

- Wyświetlanie okna aplikacji w trybie 2D,
- Funkcje do rysowania podstawowych kształtów (np. `DrawRectangle`, `DrawCircle`),
- Obsługę zdarzeń wejściowych (np. kliknięcia myszą, klawisze),
- Zapisywanie zrenderowanej bitmapy do pliku (np. `ExportImage`).

3.2 RayGui

Używamy **RayGui** (częściowo zintegrowanej z raylib), która wspiera tworzenie interfejsu graficznego w tzw. *immediate mode* — przyciski, suwaki i teksty generowane są bezpośrednio podczas rysowania każdej klatki.

3.3 C++20 i CMake

- **C++20** — W projekcie używamy `std::filesystem` do obsługi plików, `std::format` do wygodnego formatowania tekstu, `std::ranges`. Wspierane są kompilatory GCC (14.2.1) i clang (18.1.8)
- **CMake 3.30** — Plik `CMakeLists.txt` konfiguruje projekt i automatycznie pobiera raylib z GitHuba (jeśli nie jest dostępny lokalnie).

4 Instrukcja użytkownika

4.1 Uruchomienie programu

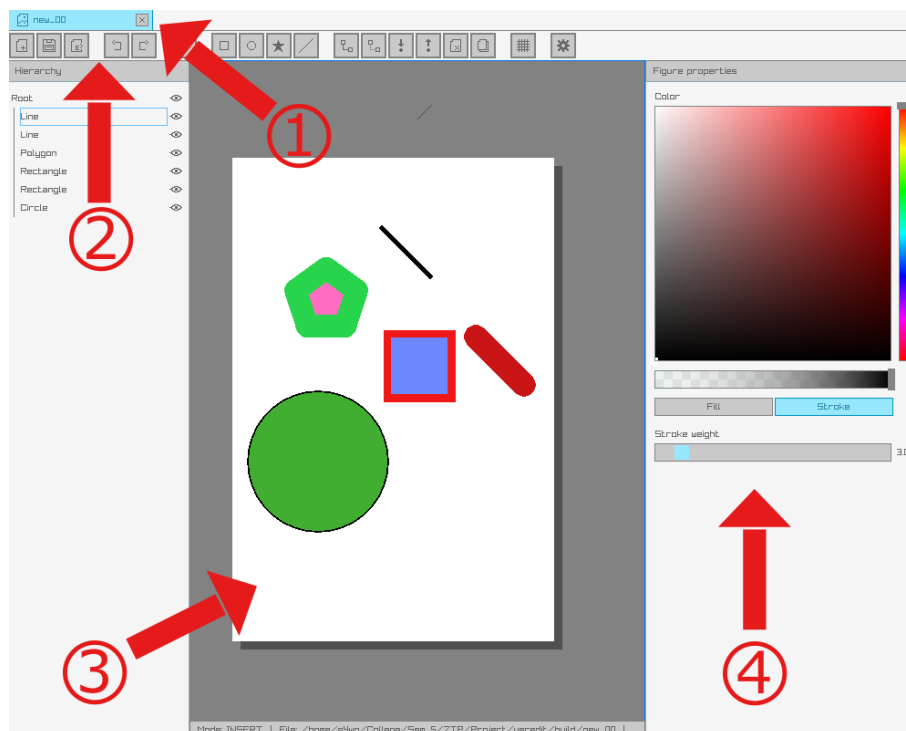
VecEdit można uruchomić:

- **Z gotowego pliku wykonywalnego:** na platformie Windows lub Linux dostarczamy *executable* w folderze `bin/`.
- **Z linii poleceń po kompilacji:** np.

```
cd build
./VecEdit
```

4.2 Interfejs programu — krótki opis

Na rysunku 1 przedstawiono przykładowy zrzut ekranu z zaznaczonymi głównymi elementami interfejsu.



Rysunek 1: Przykładowy widok programu.

TODO: dodać opisy poszczególnych guzików

Oznaczenia na zrzucie:

1. **Pasek zakładek dokumentów** — wybór otwartego projektu, możliwość zamknięcia karty.
2. **Pasek narzędzi:**
 - ikony nowego, zapisu, eksportu,
 - przyciski *undo* i *redo*,
 - przyciski wyboru narzędzia (zaznaczanie, prostokąt, koło itp.),
 - ikony grupowania / rozgrupowywania.
3. **Obszar roboczy** — kliknięcie w tym obszarze wstawia figurę (jeśli wybrano narzędzie *Insert*), zaznacza figurę (jeśli wybrano narzędzie *Select*) itp.
4. **Panel właściwości** (z boku lub w oknie) — służy do zmiany kolorów, grubości obrysu itd. dla wybranej figury.

4.3 Podstawowe operacje

TODO: usunąć/zmienić żeby było dla wszystkich guzików (razem ze skrótami klawiszowymi)

- **Nowy dokument:** Ctrl+N lub ikona *New*.
- **Zapisz:** Ctrl+S lub ikona *Save*.
- **Eksport do PNG:** Ctrl+E lub ikona *Export*.
- **Cofnij/Ponów:** Ctrl+Z / Ctrl+Y.
- **Grupowanie i rozgrupowywanie:** Ctrl+G i Ctrl+U.
- **Klonowanie figury:** zaznacz figurę, wciśnij Ctrl+D — powstanie duplikat w tym samym miejscu.

5 Instrukcja instalacji i kompilacji

1. **Pobranie kodu:** Sklonuj repozytorium (np. `git clone <URL>`).
2. **Kompilacja z użyciem CMake** (proces dokładnie opisany w `README.md`):
3. **TODO - kompilacja w CLion**

```
mkdir build
cd build
cmake ..
make
```
4. **Raylib 5.5:** CMake spróbuje automatycznie pobrać i skompilować odpowiednią wersję biblioteki.
5. **Uruchomienie:** Po kompilacji w folderze `build` znajduje się plik `VecEdit` (Linux/macOS) lub `VecEdit.exe` (Windows). Można go uruchomić z wiersza poleceń lub przez dwuklik.
6. **Problemy w macOS:** Niekiedy wymagane jest potwierdzenie w oknie ostrzeżeń systemu (niepodpisana aplikacja).

6 Podział pracy w zespole

7 Podsumowanie

VecEdit stanowi przykład aplikacji, w której zastosowanie wielu wzorców projektowych (kompozyt, metoda wytwórcza, strategia, adapter, odwiedzający, prototyp i command) umożliwia elastyczną i czytelną strukturę kodu:

- **Composite** — wspólna obsługa figur i grup (hierarchii),
- **Factory Method** — tworzenie edytorów punktów figur,
- **Strategy** — łatwe definiowanie akcji dla GUI,
- **Adapter** — jednolite API do edycji różnych typów figur,
- **Visitor** — rozdzielenie logiki rysowania i serializacji,
- **Prototype** — proste klonowanie figur,
- **Command** — mechanizm cofania i ponawiania operacji.

Dzięki temu kod można łatwo rozwijać (dodawać nowe rodzaje figur, strategie, odwiedzających) i utrzymywać.

Milego korzystania z VecEdit!