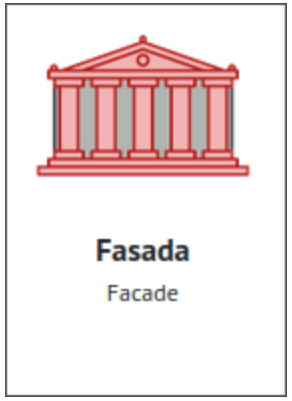
# Labirynty I Quizy

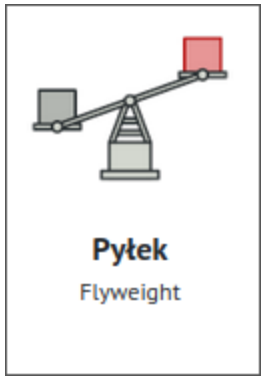
Wzorce projektowe

Projekt na inżynierię oprogramowania

Opracowali:
Aleksander Grudniok
Maciej Gładysiak
Miłosz Fido
Jaromir Gas

# Wybrane wzorce projektowe



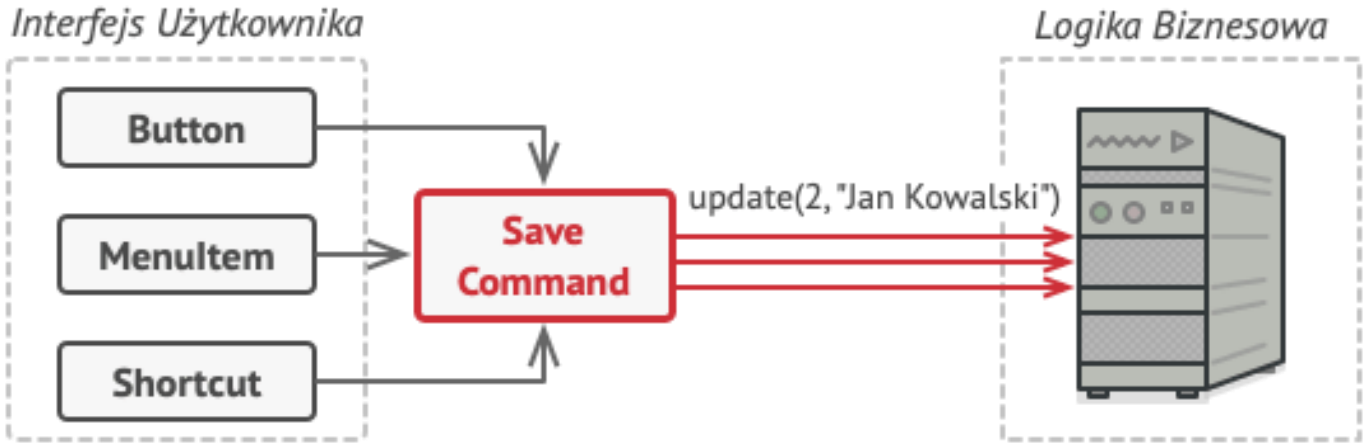


Analizując wzorce projektowe dostępne w katalogu na stronie https://refactoring.guru (z której korzystaliśmy jako katalogu dostępnych wzorców) znaleźliśmy 3, które reprezentowały lub zwierały idea interesujące nas najbardziej pod kątem realizowanego projektu. W większości przypadków poszukiwaliśmy w analizowanych wzorcach przydatnych idei i problemów, których warto unikać, natomiast od strony implementacyjnej nie przywiązywaliśmy tak dużej uwagi do przykładowych, reprezentowanych metod - miały one wysoki poziom podejścia obiektowego do programowania, do którego staramy się nawiązywać raczej w lekki sposób, niż opierać na nim w znaczący sposób naszego programu. Wspomniane 3 wzorce to fasada, pyłek i polecenie.

# Wzorzec projektowy - **Polecenie**


Polecenie jako wzorzec projektowy zakłada stworzenie oddzielnej klasy polecenia, której obiekty będą zawierać logikę niezbędną do zrealizowania danego polecenia, które musi jedynie być wywołane przez inny obiekt.

Nasza klasa przycisku korzysta z podobnego rozwiązania, przechowywując w sobie funkcje która jest wywoływana po naciśnięciu.




Piękna grafika ilustrująca idee.
Źródło: https://refactoring.guru/pl/design-patterns/command

```
// Funkcja aktywacji; najłatwiej podać w konstruktorze jako lambda function
// https://en.cppreference.com/w/cpp/language/lambda
// tldr: coś w stylu
// [&] { /* KOD FUNKCJI */; }
// przykłady w mainMenu.cpp
std::function<void()> onActivation;

_Przycisk(ScreenPos Pozycja, ScreenPos Rozmiar, std::function<void()> FunkcjaAktywacji = [&] {}, int Flagi = BUTTON_NONE) {
        pozycja = Pozycja;
        rozmiar = Rozmiar;
        onActivation = FunkcjaAktywacji;
        flags = Flagi;
        state = NIEAKTYWNY;

        los_pole_x = (float)(rand() % 51) * 0.01f * 1000.0f;//JG+
        los_pole_y = (float)(rand() % 51) * 0.01f * 1000.0f;//JG+
}
```

Implementacja podobna do polecenia funkcji aktywacji przycisku.
Po prawej – miejsce wywołania.

```
void PrzyciskTekst::afterStateChangeLogic() {
        if (state == NAJECHANY && IsMouseButtonReleased(MOUSE_BUTTON_LEFT)) {
                onActivation();
        }
}
```

Różnica - nasz przycisk ma obiekt "na sztywno", a nie wskaźnik / referencje do obiektu. Modyfikacja tego byłaby jednak dosyć prosta.

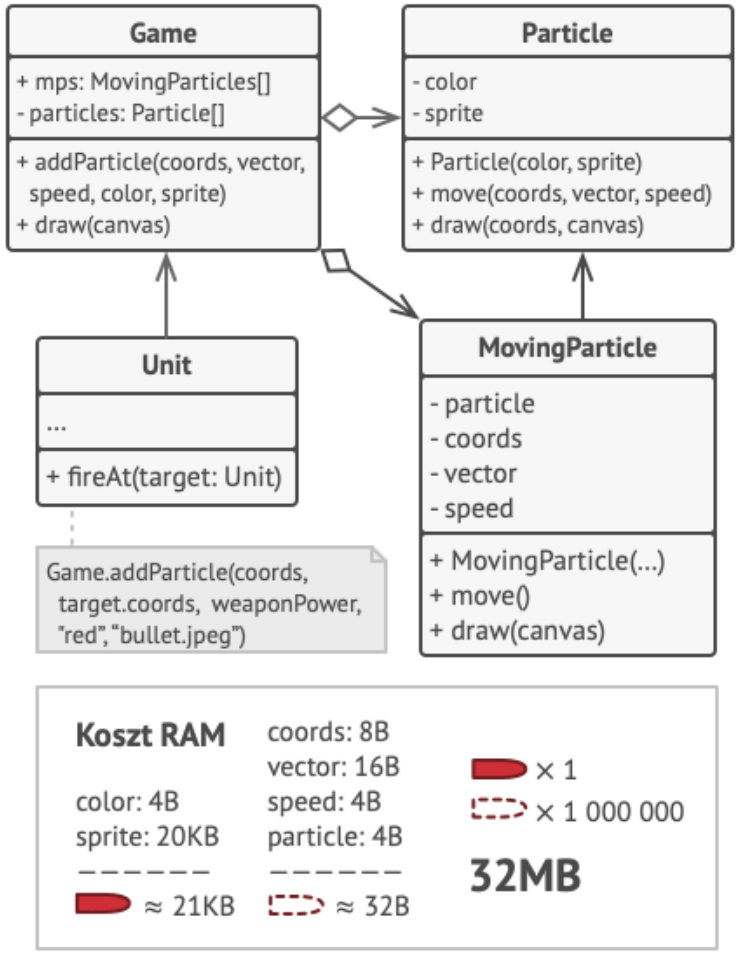
```
        // Definicja Przycisków w menu
        _Przycisk* Buttons[] = {
                new PrzyciskTekst("Wybierz Poziom", {0.5, 0.3, -0.5, 0}, 30, [&] { {stanGry = StanEkranu::WYBOR_POZIOMU; }}),
                new PrzyciskTekst("Instrukcja",{0.5, 0.4, -0.5, 0}, 20, [&] { {stanGry = StanEkranu::INSTRUKCJA; }}),
                new PrzyciskTekst("Zmień Użytkownika", {0.5, 0.5, -0.5, 0}, 20, [&] { {stanGry = StanEkranu::WYBOR_UZYTKOWNIKA; }}),
                new PrzyciskTekst("Ustawienia", {0.5, 0.6, -0.5, 0}, 20, [&] {stanGry = StanEkranu::USTAWIENIA; ustawienia::initSettings();}),
                new PrzyciskTekst("Twórcy", {0.5, 0.7, -0.5, 0}, 20, [&] {stanGry = StanEkranu::TWORCY;}),
                new PrzyciskTekst("Wyjdź", {0.5, 0.8, -0.5, 0}, 20, [&] {stanGry = StanEkranu::EXIT;}),
#ifdef _DEBUG // tego w Release nie będzie
                new PrzyciskTekst("QUIZ", {0.2, 0.2, 0, 0}, 20, [&] {stanGry = StanEkranu::GRA_QUIZ; }),
                new PrzyciskTekst("LABIRYNT", {0.2, 0.3, 0, 0}, 20, [&] {stanGry = StanEkranu::GRA_LABIRYNT; })
#endif // _DEBUG // tego w Release nie będzie
```

Przykład wykorzystania klasy przycisku wraz z zdefiniowaną funkcją która wywoła się, gdy przycisk naciśniemy.

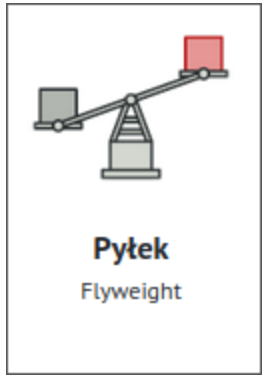
Główna korzyść to wygodna definicja nowych przycisków wraz z zawartą funkcjonalnością bez potrzeby modyfikacji większej ilości kodu.

Główna wada, natomiast, obecnej implementacji jest brak użycia ponownie kodu – obecnie wszystkie funkcje lambda użyte są po prostu przepisywane pomiędzy różnymi menu, nawet gdy ich funkcjonalność jest taka sama.

# Wzorzec projektowy - **Pyłek**




- Jest to wzorzec projektowy jest używany aby zminimalizować zużycie pamięci ram dzięki współdzieleniu części opisu stanów obiektów




Wzorzec "Pyłek" zamiast przechowywać stan zewnętrzny w obiekcie, przekazuje ten stan do metod które go potrzebują. W obrębie obiektu pozostaje jedynie stan wewnętrzny. Ograniczamy tym samym ilość obiektów, ponieważ różnią się one tylko stanem wewnętrznym.

# Możliwe zastosowanie "Pyłka" w projekcie

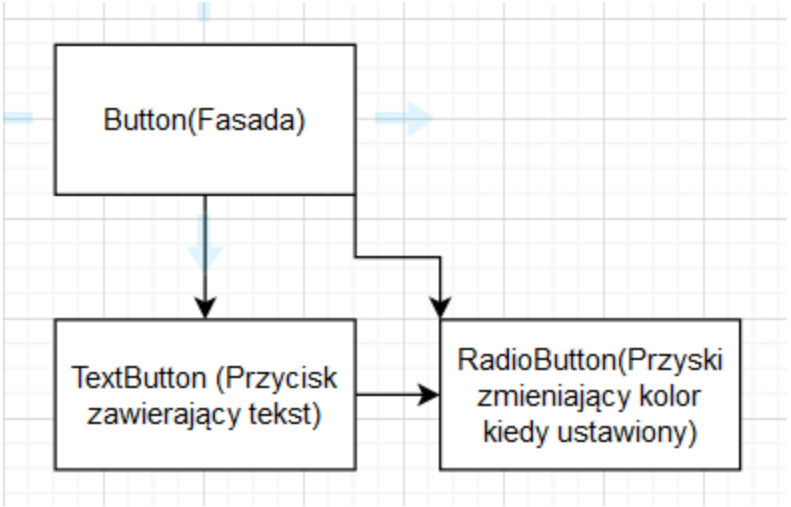
```
namespace wybor_poziomu {
    _Przycisk* Buttons[] = {
        new RadioPrzyciskTekst("Poziom 1", {0.14, 0.3, 0, 0}, 26, [&] {zmienne->poziom = 1;zmienne->epizod_doc = 1;}, [&] {return zmienne->poziom+10*zmienne->epizod_doc == 11;}
        new RadioPrzyciskTekst("Poziom 2", {0.14, 0.4, 0, 0}, 26, [&] {zmienne->poziom = 2;zmienne->epizod_doc = 1;}, [&] {return zmienne->poziom+10*zmienne->epizod_doc == 12;}
        new RadioPrzyciskTekst("Poziom 3", {0.14, 0.5, 0, 0}, 26, [&] {zmienne->poziom = 3;zmienne->epizod_doc = 1;}, [&] {return zmienne->poziom+10*zmienne->epizod_doc == 13;}
        new RadioPrzyciskTekst("Poziom 4", {0.14, 0.6, 0, 0}, 26, [&] {zmienne->poziom = 4;zmienne->epizod_doc = 1;}, [&] {return zmienne->poziom+10*zmienne->epizod_doc == 14;}
        new RadioPrzyciskTekst("Poziom 5", {0.14, 0.7, 0, 0}, 26, [&] {zmienne->poziom = 5;zmienne->epizod_doc = 1;}, [&] {return zmienne->poziom+10*zmienne->epizod_doc == 15;}

        new RadioPrzyciskTekst("Poziom 1", {0.34, 0.3, 0, 0}, 26, [&] {zmienne->poziom = 1;zmienne->epizod_doc = 2;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 2", {0.34, 0.4, 0, 0}, 26, [&] {zmienne->poziom = 2;zmienne->epizod_doc = 2;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 3", {0.34, 0.5, 0, 0}, 26, [&] {zmienne->poziom = 3;zmienne->epizod_doc = 2;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 4", {0.34, 0.6, 0, 0}, 26, [&] {zmienne->poziom = 4;zmienne->epizod_doc = 2;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 5", {0.34, 0.7, 0, 0}, 26, [&] {zmienne->poziom = 5;zmienne->epizod_doc = 2;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==

        new RadioPrzyciskTekst("Poziom 1", {0.54, 0.3, 0, 0}, 26, [&] {zmienne->poziom = 1;zmienne->epizod_doc = 3;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 2", {0.54, 0.4, 0, 0}, 26, [&] {zmienne->poziom = 2;zmienne->epizod_doc = 3;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 3", {0.54, 0.5, 0, 0}, 26, [&] {zmienne->poziom = 3;zmienne->epizod_doc = 3;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 4", {0.54, 0.6, 0, 0}, 26, [&] {zmienne->poziom = 4;zmienne->epizod_doc = 3;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 5", {0.54, 0.7, 0, 0}, 26, [&] {zmienne->poziom = 5;zmienne->epizod_doc = 3;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==

        new RadioPrzyciskTekst("Poziom 1", {0.74, 0.3, 0, 0}, 26, [&] {zmienne->poziom = 1;zmienne->epizod_doc = 4;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 2", {0.74, 0.4, 0, 0}, 26, [&] {zmienne->poziom = 2;zmienne->epizod_doc = 4;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 3", {0.74, 0.5, 0, 0}, 26, [&] {zmienne->poziom = 3;zmienne->epizod_doc = 4;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 4", {0.74, 0.6, 0, 0}, 26, [&] {zmienne->poziom = 4;zmienne->epizod_doc = 4;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==
        new RadioPrzyciskTekst("Poziom 5", {0.74, 0.7, 0, 0}, 26, [&] {zmienne->poziom = 5;zmienne->epizod_doc = 4;}, [&] {return zmienne->poziom + 10 * zmienne->epizod_doc ==

        new PrzyciskTekst("Powrot",{0, 1, 0.1, -1.3}, 30, [&] {stanGry = StanEkranu::MAIN_MENU; }),

        new PrzyciskTekst("QUIZ", {0.37, 0.85, 0, 0}, 20, [&] {stanGry = StanEkranu::GRA_QUIZ; }),
        new PrzyciskTekst("LABIRYNT", {0.54, 0.85, 0, 0}, 20, [&] {stanGry = StanEkranu::GRA_LABIRYNT; })
```

W pliku wybor_poziomu.cpp zamiast stworzenia ponad 20 przycisków z wyborem poziomów, można zastosować wzorzec "pyłek" i kolejno:

1. Wprowadzić fabrykę do zarządzania wspólnymi danymi przycisków.
2. Przekształcić przyciski w instancje, które współdzielą dane.
3. Oddzielić wspólne dane takie jak szerokość czy wysokość przycisku, od specyficznych czyli pozycji lub zwracanego poziomu.

# Wzorzec projektowy - Fasada

- Fasada to klasa umożliwiająca stworzenie prostego interfejsu dla wielu elementów danego typu




Naszą dotychczasową fasadą jest klasa przycisk ułatwiająca korzystanie z wyszczególnionych typów przycisków.

Fasada pozwala na łatwiejsze zarządzanie kodem i uniknięcie kontaktu z całym, obszernym elementem programu, ograniczając go do niezbędnego obszaru - zwiększa to wydajność.

W naszym programie nie ,,widzimy" na razie konieczności wykorzystywania takiego rzowiązania względem bibliotek i frameworków, za to zastosowanie podobnych idei w kodowaniu oraz dostępie do bazy danych mooże być bardzo pomocne.

Klasa PrzyciskTekst zrobiona na podstawie klasy Przycisk, która umożliwia tworzenie różnych typów przycisków.

```
struct _Przycisk {
    ScreenPos pozycja;
    ScreenPos rozmiar;
    StanPrzycisku state;
    int flags;
    std::function<void()> onActivation;
    _Przycisk(ScreenPos Pozycja, ScreenPos Rozmiar, std::function<void()> FunkcjaAktywacji = [&] {}, int Flagi = BUTTON_NONE) {
        pozycja = Pozycja;
        rozmiar = Rozmiar;
        onActivation = FunkcjaAktywacji;
        flags = Flagi;
        state = NIEAKTYWNY;
    }
    _Przycisk() {
        pozycja = { 0,0,0,0 };
        rozmiar = { 0,0,0,0 };
        onActivation = [&] {};
        flags = BUTTON_DISABLED;
        state = NIEAKTYWNY;
    }
    virtual void update() = 0; // Update, przed rysowaniem.
    virtual void draw() const = 0; // Logika sterująca rysowaniem
};
struct PrzyciskTekst : public _Przycisk {
    const char* text;
    int fontSize;
    PrzyciskTekst(
        const char* tekst,
        ScreenPos position,
        int _fontSize,
        std::function<void()> onPress = [&] {},
        int flags = BUTTON_NONE) :
        _Przycisk(position, {0.f,0.f,0.05f,0.2f}, onPress, flags)
    {
        text = tekst;
        fontSize = _fontSize;
    };
```

Dalsze wykorzystanie wzorca fasady może polegać na tworzeniu w grze wielu ruchomych przeszkód w labiryncie korzystających z jednego pierwowzoru. Ideę zbliżoną do fasady wykorzystaliśmy też przy zarządzaaniu danymi poziomów, grafikami i danymi użytkownika - w danym momencie odpowiednie zbiory danych w programie zawierają tylko te znich, które są potrzebne – w razie potrzeby zastępując je innymi wczytanymi z bazy danych lub np. Plików tekstowych.

# Dziękujemy za uwagę!

Wzorce opracowane na podstawie:

- https://refactoring.guru/pl

Opracowali:
Aleksander Grudniok
Maciej Gładysiak
Miłosz Fido
Jaromir Gas