Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki

Maciej Bandura, Marcin Ślusarczyk

Aplikacja służąca do tworzenia schematów blokowych algorytmów

Projekt zespołowy

studia stacjonarne
kierunek Informatyka

Opiekun projektu:
Dr inż. Ludomir Tuszyński

Kielce, 2022

SPIS TREŚCI

1. CHARAKTERYSTYKA ZADANIA PROJEKTOWEGO

1.1. Wprowadzenie

Programy służące do kreowania schematów blokowych algorytmów to narzędzia wykorzystywane w dziedzinie informatyki i programowania. Ich celem jest umożliwienie łatwego i intuicyjnego przedstawienia kroków algorytmu w postaci diagramu blokowego.

Schemat blokowy jest używany w celu przedstawienia algorytmu w sposób zrozumiały dla ludzi, ułatwiający wizualizację kolejności działań oraz umożliwiający łatwiejsze debugowanie kodu. Programy do rysowania schematów blokowych algorytmów są używane przez programistów, studentów informatyki, naukowców i inżynierów, którzy potrzebują sposobu na przedstawienie procesów algorytmicznych w przystępny sposób.

1.2. Słownik

- **Diagram blokowy** - graficzne przedstawienie algorytmu w postaci bloków i łączących je strzałek, które przedstawiają kolejność wykonywanych działań.
- **Blok** - graficzny symbol reprezentujący określony krok w algorytmie, np. operację arytmetyczną, warunek lub pętlę.
- **Wejście danych** - blok reprezentujący pobranie danych wejściowych dla algorytmu.
- **Przetwarzanie** - blok reprezentujący wykonywanie operacji na danych.
- **Prosty warunek** - blok reprezentujący wybór alternatywnych ścieżek algorytmu w zależności od spełnienia określonego warunku.
- **Blok funkcji** - blok reprezentujący wywołanie funkcji której schemat również został przedstawiony na diagramie. .
- **Wyjście danych** - blok reprezentujący wyprowadzenie wyniku działania algorytmu.
- **Komentarz** - blok reprezentujący tekstowy komentarz dodawany do diagramu, np. do wyjaśnienia działania poszczególnych kroków.
- **Instrukcja** - pojedynczy krok w algorytmie, reprezentowany przez jeden lub więcej bloków.
- **Strzałka** - graficzny symbol reprezentujący przepływ sterowania w algorytmie, wskazujący kolejność wykonywania instrukcji.

1.3. Ogólny opis systemu

Program do rysowania schematów blokowych algorytmów jest to narzędzie informatyczne, które umożliwia użytkownikowi łatwe tworzenie diagramów blokowych przedstawiających algorytmy w postaci graficznej, a następnie generowanie kodu źródłowego w języku C na podstawie stworzonego diagramu.

Typowy program umożliwia użytkownikowi wybór z zestawu gotowych bloków, takich jak "wejście danych", "przetwarzanie", "warunek", "wyjście danych", "funkcja", a następnie łączenie ich w logiczne sekwencje. Po utworzeniu diagramu blokowego, użytkownik ma możliwość wygenerowania kodu źródłowego w języku C, który odpowiada przedstawionemu algorytmowi.

1.4. Wymagania funkcjonalne

- Tworzenie diagramów blokowych:
  - Program powinien umożliwiać tworzenie diagramów blokowych, w których użytkownik może dodawać bloki symbolizujące poszczególne kroki algorytmu.
  - Użytkownik powinien mieć możliwość połączenia bloków ze sobą, aby utworzyć logiczną sekwencję kroków algorytmu.
  - Program powinien umożliwiać dodawanie komentarzy do diagramów blokowych, aby umożliwić użytkownikowi opisanie algorytmu.
- Generowanie kodu w języku C:
  - Program powinien umożliwiać generowanie kodu źródłowego w języku C na podstawie stworzonego diagramu blokowego.
  - Generowany kod powinien być zgodny z obowiązującą składnią języka C.
  - Użytkownik powinien mieć możliwość wyboru nazwy pliku wyjściowego i lokalizacji zapisu.
  - Program zakłada, że utworzony przez użytkownika schemat blokowy jest poprawny i nie zawiera błędów które mogą prowadzić do błędnego działania wygenerowanego kodu programu
- Import i eksport:
  - Program powinien umożliwiać importowanie diagramów blokowych z pliku.

    - Użytkownik powinien mieć możliwość eksportowania diagramów blokowych do różnych formatów plików, takich jak PNG, JPG.
- Obsługa różnych bloków:
    - Program powinien umożliwiać użytkownikowi wybór z różnych bloków, takich jak "wejście danych", "przetwarzanie", "prosty warunek", "wyjście danych", "funkcja", aby umożliwić stworzenie diagramów blokowych dla różnych rodzajów algorytmów.
- Walidacja diagramów blokowych:
    - Walidacja powinna obejmować sprawdzanie, czy diagram blokowy jest kompletny.

1.5. Wymagania niefunkcjonalne

- Wygląd i interfejs użytkownika:
    - Program powinien mieć intuicyjny interfejs użytkownika, który pozwoli użytkownikom na łatwe korzystanie z aplikacji.
    - Użytkownik powinien mieć możliwość połączenia bloków ze sobą, aby utworzyć logiczną sekwencję kroków algorytmu.
    - Program powinien umożliwiać dodawanie komentarzy do diagramów blokowych, aby umożliwić użytkownikowi opisanie algorytmu.
- Wydajność i stabilność:
    - Program powinien działać sprawnie i szybko, bez opóźnień lub zacinania się.
    - Aplikacja powinna działać stabilnie, bez awarii lub błędów.

1.6. Podział prac

- Maciek
    - Rozbudowa edytora graficznego o funkcjonalności tj.: kopiowanie, wklejanie, wycinanie,
    - manipulacja tekstem na blokach,
    - rozbudowa elementu strzałki o możliwość przełączania tryby wyświetlania grotu,
    - projekt oraz implementacja interfejsu użytkownika: interaktywne elementy interfejsu, ikonki, wyszukiwarka, toolbox z elementami diagramu,

    - zintegrowany menedżer eventów: nasłuchiwanie na przyciski oraz skróty klawiszowe,
    - import diagramów z pliku,
    - narzędzie do sprawdzania spójności diagramów w procesie eksportu do kodu,
- Marcin
    - fundamenty edytora graficznego tj.: stawiane bloków, zaznaczanie bloków, przesuwanie bloków oraz api interakcji z blokami,
    - Implementacja elementu strzałki (łączeń bloków),
    - projekt oraz implementacja interfejsu użytkownika: rozmieszczenie paneli, geometria okien programu oraz prefabrykowane elementy UI,
    - implementacja historii działań użytkownika (tryby cofnij oraz do przodu)
    - eksport diagramów do plików z rozszerzeniem wewnętrznym oraz do plików graficznych tj.: .JPG, .PNG,
    - algorytm konwertujący diagram na kod języka C.

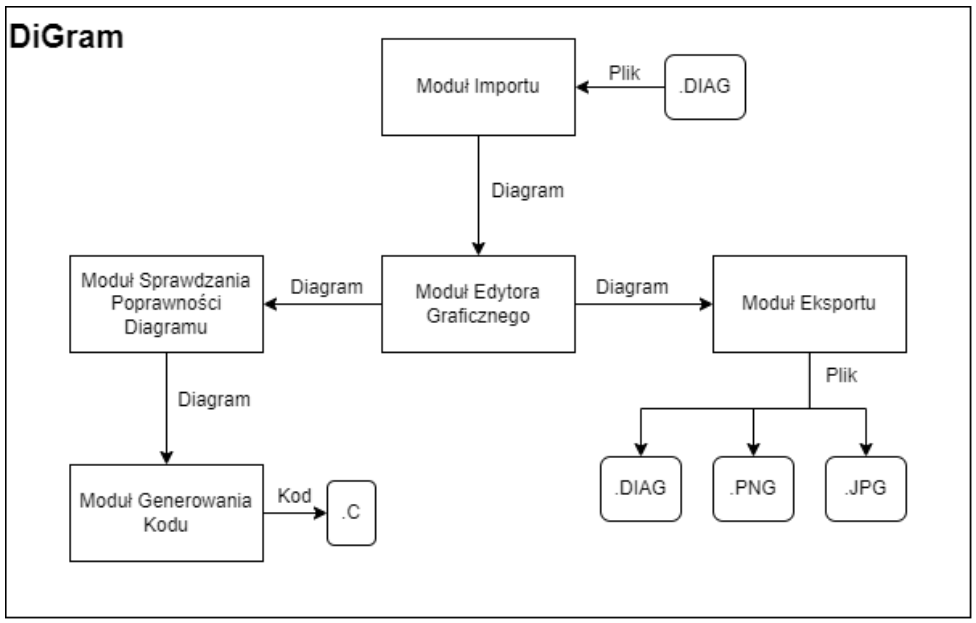
2. STRUKTURA APLIKACJI

2.1. Moduły aplikacji

- **Moduł edytora graficznego** - umożliwia użytkownikowi stawianie bloków i połączeń na obszarze roboczym. Użytkownik może przeciągać i upuszczać bloki, łączyć je liniami oraz zmieniać ich właściwości, takie jak szerokość oraz manipulować właściwościami tekstu wewnątrz bloków. Tymi właściwościami są: pogrubienie, kursywa, podkreślenie oraz rozmiar i krój czcionki. Interfejs edytora graficznego jest intuicyjny i łatwy w obsłudze.
- **Moduł importu** - umożliwia użytkownikowi importowanie utworzonego wcześniej diagramu.
- **Moduł eksportu** - umożliwia eksportowanie utworzonego diagramu do pliku. Program obsługuje popularne formaty plików graficznych, takie jak PNG i JPG, aby umożliwić łatwe udostępnianie oraz dystrybuowanie diagramów.
- **Moduł generowania kodu** - po stworzeniu diagramu, użytkownik może wygenerować kod w języku C na podstawie stworzonego schematu blokowego

algorytmu. Wygenerowany kod powinien być czytelny i zgodny z zasadami programowania w języku C.

- **Moduł sprawdzenia poprawności diagramu** - umożliwia użytkownikowi sprawdzenie poprawności stworzonego schematu blokowego algorytmu. Program powinien wykrywać i informować użytkownika o błędach, takich jak niespójność połączeń czy wolnostojące bloki. W ten sposób użytkownik może szybko wykryć i poprawić błędy w schemacie blokowym. Aby móc przejść do generowania kodu diagram powinien być wolny od błędów.



Rys. 2.1.1. Diagram modułów aplikacji

2.2. Algorytm

Algorytm eksportowania diagramu do kodu w języku C oparty jest o kilka podstawowych reguł i zadań. Głównym założeniem jest powstanie jednego pliku z rozszerzeniem .c, składającego się z 3 części. Struktura tego pliku jest następująca: w pierwszej części zawarta jest lista wykorzystywanych bibliotek do działania programu. Biblioteki te są automatycznie wyselekcjonowane na podstawie funkcji użytych w kodzie. Część ta jest generowana jako ostatnia. W kolejnej części znajdują się predefinicje użytych funkcji. Specyfika języka C wymaga predefiniowania albo logicznego szeregowania definicji funkcji. Została zastosowana pierwsza metoda – predefiniowanie wszystkich

funkcji przed deklaracją ich ciał. Ostatnia składowa pliku zawiera wspomniane definicje ciał funkcji.

W początkowej fazie następuje grupowanie części diagramu reprezentujące konkretne funkcję, poprzez blok startu. Kolejne kroki będą wykonywane dla każdej ze wspomnianych grup. Z bloku startowego wyszukiwane są wszystkie wychodzące ścieżki, jeżeli jest więcej niż jedna – diagram jest uznawany za niepoprawny, użytkownik informowany jest o błędzie. Poprawna ścieżka jest dalej śledzona do kolejnego bloku diagramu. W tym miejscu sprawdzane jest, czy fragment bloku nie został już dodany do bufora kodu źródłowego, jeżeli nie to blok zostaje dopisany do kodu źródłowego uzyskując uprzednio unikalny label. Sprawdzane są wychodzące ścieżki z bloku, podobnie jak w przypadku bloku startowego, jeżeli ścieżka dojdzie do bloku typu: przetwarzanie, IO, funkcja, to do kodu źródłowego zostaje przypisana instrukcja goto, a blok zostaje przetwarzany w podany wyżej sposób. Wyjątkowym przypadkiem jest blok instrukcji warunkowej, z którego może wychodzić więcej niż jedna ścieżka. Dla każdej wychodzącej ścieżki wyszukiwany jest najbliższy blok tekstu, który w połączeniu z zawartością bloku warunku utworzy kompletny warunek, natomiast do labela bloku na końcu ścieżki zostaje utworzona instrukcja goto, wykonywana w przypadku spełnienia konkretnego warunku. Algorytm jest jednowątkowy w związku z tym w danym momencie śledzona jest tylko jedna ścieżka. Natomiast pozostałe rozchodzące się ścieżki są buforowane, zostaną prześledzone w innej chwili czasowej. W momencie gdy, któraś ze ścieżek wskazuje na blok typu koniec, to kontynuowane jest podążanie po jeszcze nie prześledzonych ścieżkach z innych warunków. Jeżeli któraś ze ścieżek nie wskazuje na żaden blok, diagram jest uznawany za niepoprawny i błąd jest zgłaszany użytkownikowi. Po kompletnym przełożeniu diagramu na kod źródłowy funkcji następuje usunięcie niewykorzystywanych labeli – tz. takich z których nie korzystają instrukcje goto. Następnie linker bibliotek korzystając z wbudowanego zbioru definicji funkcji w powszechnych bibliotekach stdc, glibc odnajduje listę wszystkich wykorzystywanych przez kod bibliotek i automatycznie generuje dyrektywy include linkujące wspomniane biblioteki. W ostatniej fazie algorytm łączy wspomniane części w jeden konkretny kod źródłowy. O kolejnych krokach (jak zapisanie do pliku) decyduje inny moduł programu.

3. OPIS APLIKACJI

4.1. Analiza systemowa:

a. kontekst systemu

Aktorzy:

- Użytkownik główny - osoba, która korzysta z aplikacji do tworzenia schematów blokowych algorytmów. Ma on pełnię uprawnień oraz dostęp do każdej funkcjonalności programu.

Procesy:

1. Tworzenie diagramu:

    - Użytkownik wchodzi do aplikacji i rozpoczyna proces tworzenia diagramu.
    - Użytkownik korzysta z edytora graficznego, aby stworzyć schemat blokowy algorytmu, dodając bloki, łącząc je liniami i manipulując ich właściwościami.
    - Użytkownik może zapisać diagram i kontynuować pracę w późniejszym czasie.

2. Importowanie diagramu:

    - Użytkownik korzysta z modułu importu, aby zaimportować utworzony wcześniej diagram do aplikacji.
    - Aplikacja przetwarza plik i wyświetla diagram na ekranie, umożliwiając użytkownikowi jego dalszą edycję.

3. Eksportowanie diagramu:

    - Użytkownik korzysta z modułu eksportu, aby wyeksportować utworzony diagram do pliku w formacie graficznym, takim jak PNG lub JPG.
    - Aplikacja przetwarza diagram i zapisuje go jako plik w wybranym przez użytkownika miejscu na dysku twardym.
    - Aplikacja umożliwia zapis diagramu do wewnętrznego formatu .diag, aby umożliwić jego późniejszą edycję

4. Generowanie kodu:

    - Użytkownik korzysta z modułu generowania kodu, aby wygenerować kod w języku C na podstawie stworzonego schematu blokowego algorytmu.
    - Aplikacja przetwarza diagram i generuje kod w języku C, który może zostać zapisany w wybranym przez użytkownika miejscu.

5. Sprawdzanie poprawności diagramu:

    - Aplikacja korzysta z modułu sprawdzania poprawności diagramu, aby sprawdzić, czy stworzony schemat blokowy algorytmu jest poprawny.
    - Aplikacja przetwarza diagram i wykrywa ewentualne błędy lub niespójności w schemacie blokowym.
    - Aplikacja informuje użytkownika o znalezionych błędach.

Zdarzenia:

- Użytkownik korzysta z modułu edytora graficznego, aby stworzyć schemat blokowy algorytmu.
- Użytkownik może zapisać utworzony schemat blokowy algorytmu w pliku przy użyciu modułu eksportu.
- Użytkownik może wczytać istniejący schemat blokowy algorytmu z pliku za pomocą modułu importu.
- Jeśli schemat blokowy algorytmu jest poprawny, użytkownik może wygenerować kod w języku C na jego podstawie za pomocą modułu generowania kodu.
- Użytkownik może poprawiać schemat blokowy algorytmu, aby usunąć wykryte błędy, a następnie ponownie wykonać proces generowania kodu i eksportowania schematu blokowego algorytmu.

b. scenariusze przypadków użycia (moduły),

Sekcja	**Treść**
Tytuł	Rozpoczęcie tworzenia schematu blokowego algorytmu
Moduł	Moduł edytora graficznego
Warunki wstępne	Użytkownik musi posiadać aplikacje
Warunki końcowe	Użytkownik utworzył schemat blokowy algorytmu
Rezultat	Schemat blokowy jako zdjęcie, plik .diag lub kod w języku C.
Scenariusz główny	1. Uruchomienie aplikacji 2. Rysowanie diagramu w graficznym edytorze 3. Eksportowanie diagramu do pożądanego formatu

Sekcja	**Treść**
Tytuł	Postawienie bloku w edytorze graficznym
Moduł	Moduł edytora graficznego
Warunki wstępne	Użytkownik musi być w trybie edycji graficznej
Warunki końcowe	Użytkownik stawia blok w obszarze roboczym
Rezultat	Postawiony blok w obszarze roboczym aplikacji
Scenariusz główny	1. Wybór lub wyszukanie bloku z toolbox'a. 2. Użytkownik przeciąga wybrany blok na obszar roboczy. 3. Dodany blok zostaje automatycznie zaznaczony, a użytkownik może dowolnie manipulować jego właściwościami za pomocą toolbar'a.

Sekcja	**Treść**
Tytuł	Dodanie treści do postawionego bloku
Moduł	Moduł edytora graficznego
Warunki wstępne	Użytkownik musi być w trybie edycji graficznej Użytkownik musi posiadać postawiony blok w obszarze roboczym
Warunki końcowe	Użytkownik wstawia lub edytuje treść wewnątrz bloku
Rezultat	Treść w bloku zmienia się.
Scenariusz główny	1. Użytkownik klika dwukrotnie lewym przyciskiem myszy na treść znajdującą się w bloku 2. Tekst wewnątrz bloku staje się edytowalny i automatycznie zaznaczony 3. Użytkownik wprowadza żądaną treść do bloku

Sekcja	**Treść**
Tytuł	Nawigowanie po przestrzeni roboczej
Moduł	Moduł edytora graficznego
Warunki wstępne	Użytkownik musi być w trybie edycji graficznej
Warunki końcowe	Użytkownik zmienia orientację obszaru roboczego
Rezultat	Pozycja w przestrzeni roboczej się zmienia
Scenariusz główny	1. Użytkownik przytrzymując PPM wyznacza kierunek oraz odległość przesunięcia, podobnie jak przy grupowym zaznaczaniu obiektów.

Sekcja	**Treść**
Tytuł	Zaznaczenie obiektu
Moduł	Moduł edytora graficznego
Warunki wstępne	Użytkownik musi być w trybie edycji graficznej Użytkownik musi posiadać postawiony blok w obszarze roboczym
Warunki końcowe	Użytkownik posiada zaznaczony obiektu w obszarze roboczym
Rezultat	Zaznaczony blok
Scenariusz główny	1. Użytkownik klikając LPM na blok lub strzałkę zaznacza je. 2. Wokół zaznaczonego obiektu pojawia się żółta obramówka. 3. Obiekt staje się zaznaczony.

Sekcja	**Treść**
Tytuł	Grupowe zaznaczenie obiektów
Moduł	Moduł edytora graficznego
Warunki wstępne	Użytkownik musi być w trybie edycji graficznej Użytkownik musi posiadać kilka elementów w obszarze roboczym
Warunki końcowe	Użytkownik posiada zaznaczone obiekty w obszarze roboczym
Rezultat	Kilka zaznaczonych obiektów

Scenariusz główny	1. Użytkownik przytrzymując LPM na obszarze roboczym wyznacza jego podobszar w którym znajdujące się obiektu zostaną grupowo zaznaczone. 2. Wokół zaznaczonych obiektów pojawia się żółta obramówka. 3. Obiekty stają się zaznaczone.

Sekcja	**Treść**
Tytuł	Wstawienie strzałki
Moduł	Moduł edytora graficznego
Warunki wstępne	Użytkownik musi być w trybie dodawania strzałek (Uruchamiając go przy pomocy klawisza **ENTER** lub klikając odpowiednią ikonkę na toolbarze).
Warunki końcowe	Użytkownik dodaje strzałkę bądź linię
Rezultat	Obiekt strzałki lub linii w obszarze roboczym.
Scenariusz główny	1. Użytkownik przytrzymując LPM rysuje pionową lub poziomą (w zależności od odchylenia pozycji myszki względem pozycji startowej) linię lub strzałkę w zależności od ustawionego w edytorze trybu (Ustawiając go przy pomocy klawisza **SPACJA** lub klikając odpowiednią ikonkę na toolbarze). 2. Strzałka lub linia została dodana.

Sekcja	**Treść**
Tytuł	Zmiana właściwości obiektów
Moduł	Moduł edytora graficznego
Warunki wstępne	Użytkownik musi być w trybie edycji graficznej Użytkownik musi posiadać co najmniej jeden zaznaczony obiekt.
Warunki końcowe	Użytkownik zmienia właściwości obiektu.
Rezultat	Obiekt ze zmienionymi właściwościami.
Scenariusz główny	1. Użytkownik zaznacza jeden lub grupę obiektów. 2. Użytkownik wybiera nowe właściwości dla zaznaczonych obiektów z toolbar'a lub skrótami klawiszowymi. 3. Ewentualnie użytkownik zmienia pozycję zaznaczonych obiektów. 4. Obiekty zostały zmienione.
Scenariusz poboczny	2. Jeżeli obiekty różnią się rodzajami, możliwość edycji niektórych właściwości zostanie wyłączona.

Sekcja	**Treść**
Tytuł	Cofnięcie operacji
Moduł	Moduł edytora graficznego
Warunki wstępne	Użytkownik musi wykonać dowolną operacje w edytorze graficznym
Warunki końcowe	Użytkownik wraca do poprzedniego stanu w obszarze roboczym.
Rezultat	Cofnięcie stanu obiektów na obszarze roboczym do stanu sprzed wykonania ostatniej operacji.
Scenariusz główny	1. Użytkownik klika ikonę cofnij z toolbar'a, lub korzysta ze skrótu klawiszowego (**CTRL+Z**)

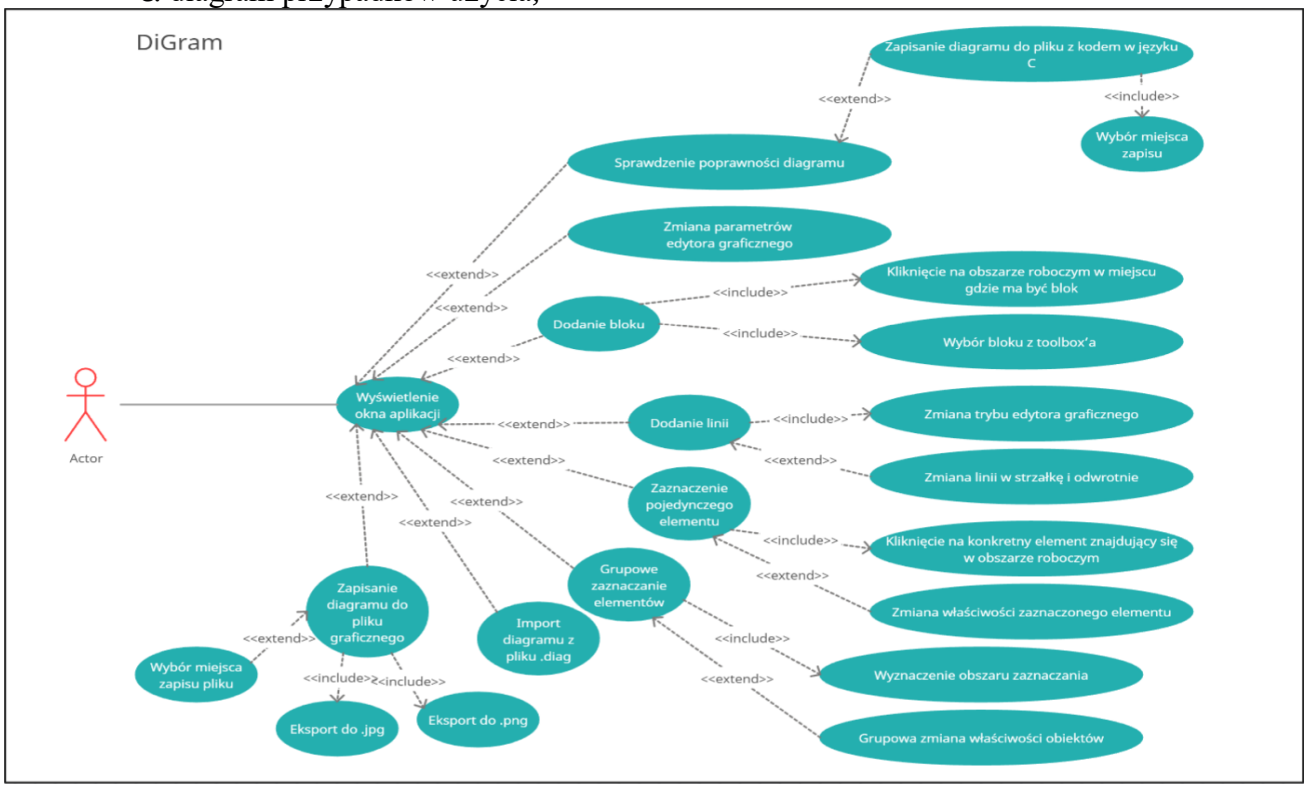
Sekcja	**Treść**
Tytuł	Import diagramu
Moduł	Moduł importu
Warunki wstępne	Użytkownik musi posiadać plik .diag z utworzonym wcześniej diagramem
Warunki końcowe	Użytkownik załadowuje schemat blokowy do programu
Rezultat	Diagram blokowy załadowany w programie
Scenariusz główny	1. Użytkownik wybiera pozycję z paska menu: plik => otwórz. 2. Następnie wskazuje plik z rozszerzeniem .diag do otwarcia 3. Jeżeli plik jest poprawny, diagram zostaje załadowany do programu

Sekcja	**Treść**
Tytuł	Eksport diagramu do wybranego formatu graficznego
Moduł	Moduł eksportu
Warunki wstępne	Diagram który ma zostać wyeksportowany musi być wolny od błędów
Warunki końcowe	Użytkownik eksportuje diagram
Rezultat	Zapisany plik graficzny we wskazanym przez użytkownika miejscu
Scenariusz główny	1. Użytkownik wybiera pozycję z paska menu: plik => eksport. 2. Następnie wybiera format w jakim chce utworzony diagram wyeksportować (dostępne są .PNG oraz .JPG) 3. Wskazuje miejsce zapisu diagramu. 4. Jeżeli schemat blokowy jest poprawny, plik zostaje utworzony
Scenariusz poboczny	4. Jeżeli schemat blokowy nie jest poprawny, użytkownik zostaje o tym fakcie poinformowany a proces eksportu zostaje przerwany.

Sekcja	**Treść**
Tytuł	Zapis diagramu do formatu .diag
Moduł	Moduł eksportu
Warunki wstępne	
Warunki końcowe	Użytkownik eksportuje diagram
Rezultat	Zapisany plik .diag we wskazanym przez użytkownika miejscu
Scenariusz główny	1. Użytkownik wybiera pozycję z paska menu: plik => zapisz. 2. Wskazuje miejsce zapisu diagramu. 3. Plik zostaje utworzony

Sekcja	**Treść**
Tytuł	Zapis diagramu do kodu w języku C
Moduł	Moduł eksportu
Warunki wstępne	Diagram który ma zostać zapisany do języka C musi być wolny od błędów
Warunki końcowe	Użytkownik eksportuje diagram
Rezultat	Zapisany plik z kodem w języku C
Scenariusz główny	1. Użytkownik wybiera pozycję z paska menu: plik => zapisz kod. 2. Wskazuje miejsce zapisu diagramu. 3. Jeżeli schemat blokowy jest poprawny, plik zostaje utworzony
Scenariusz poboczny	3. Jeżeli schemat blokowy nie jest poprawny, użytkownik zostaje o tym fakcie poinformowany a proces zapisu zostaje przerwany.

c. diagram przypadków użycia,



DiGram - diagram sekwencji

4.3. Projekt architektury:

W ramach projektu budowy programu do rysowania schematów blokowych algorytmów z możliwością generacji kodu w języku C zdecydowaliśmy się wykorzystać technologię ElectronJS.

ElectronJS to otwarte narzędzie pozwalające na tworzenie natywnych aplikacji desktopowych przy użyciu webowych technologii, takich jak HTML, CSS i JavaScript. Dzięki wykorzystaniu ElectronJS programiści mogą wykorzystać popularne i znane technologie internetowe do tworzenia natywnych aplikacji desktopowych, co przyspiesza proces tworzenia oprogramowania i umożliwia łatwe dostosowanie aplikacji do różnych platform.

Ponadto, ElectronJS oferuje wsparcie dla wielu platform, takich jak Windows, macOS i Linux, co pozwala na łatwe tworzenie aplikacji, które będą działały na różnych systemach operacyjnych. Wsparcie dla aplikacji internetowych, takich jak aplikacje React i Angular, ułatwia integrację aplikacji internetowych z aplikacjami desktopowymi.

Zdecydowaliśmy się na wykorzystanie technologii ElectronJS ze względu na jej zalety, takie jak łatwość tworzenia aplikacji, wsparcie dla wielu platform i wsparcie dla aplikacji internetowych.

4.4. Implementacja i testowanie aplikacji.

a. scenariusze testowania aplikacji.

Implementacja aplikacji do rysowania schematów blokowych będzie wymagała dokładnego testowania, aby zapewnić jej poprawne działanie. W ramach testowania aplikacji utworzymy kilka prostych schematów blokowych algorytmów i będziemy analizować oraz uruchamiać wygenerowany kod w języku C pod kątem jego poprawności. Będziemy również testować generowanie kodu z bardziej złożonych diagramów, aby upewnić się, że aplikacja działa zgodnie z oczekiwaniami. Dodatkowo, w celu ułatwienia korzystania z aplikacji, do plików aplikacji dołączymy przykładowe schematy blokowe algorytmów wraz z wygenerowanym kodem w języku C. Użytkownicy będą mogli zacząć od tych prostych przykładów i stopniowo rozwijać swoje diagramy i generować kod, co ułatwi im naukę korzystania z aplikacji.

Testowanie aplikacji będzie przeprowadzane w wielu etapach, aby upewnić się, że program działa zgodnie z oczekiwaniami. Na początku będziemy testować poszczególne

moduły aplikacji, a następnie integrować je w całość. Ostatecznie, po wdrożeniu aplikacji, będziemy monitorować jej działanie i w razie potrzeby wprowadzać poprawki.

W ten sposób, nasza aplikacja zostanie dokładnie przetestowana, co pozwoli nam zagwarantować jej poprawne działanie i zapewnić użytkownikom komfort korzystania z niej.

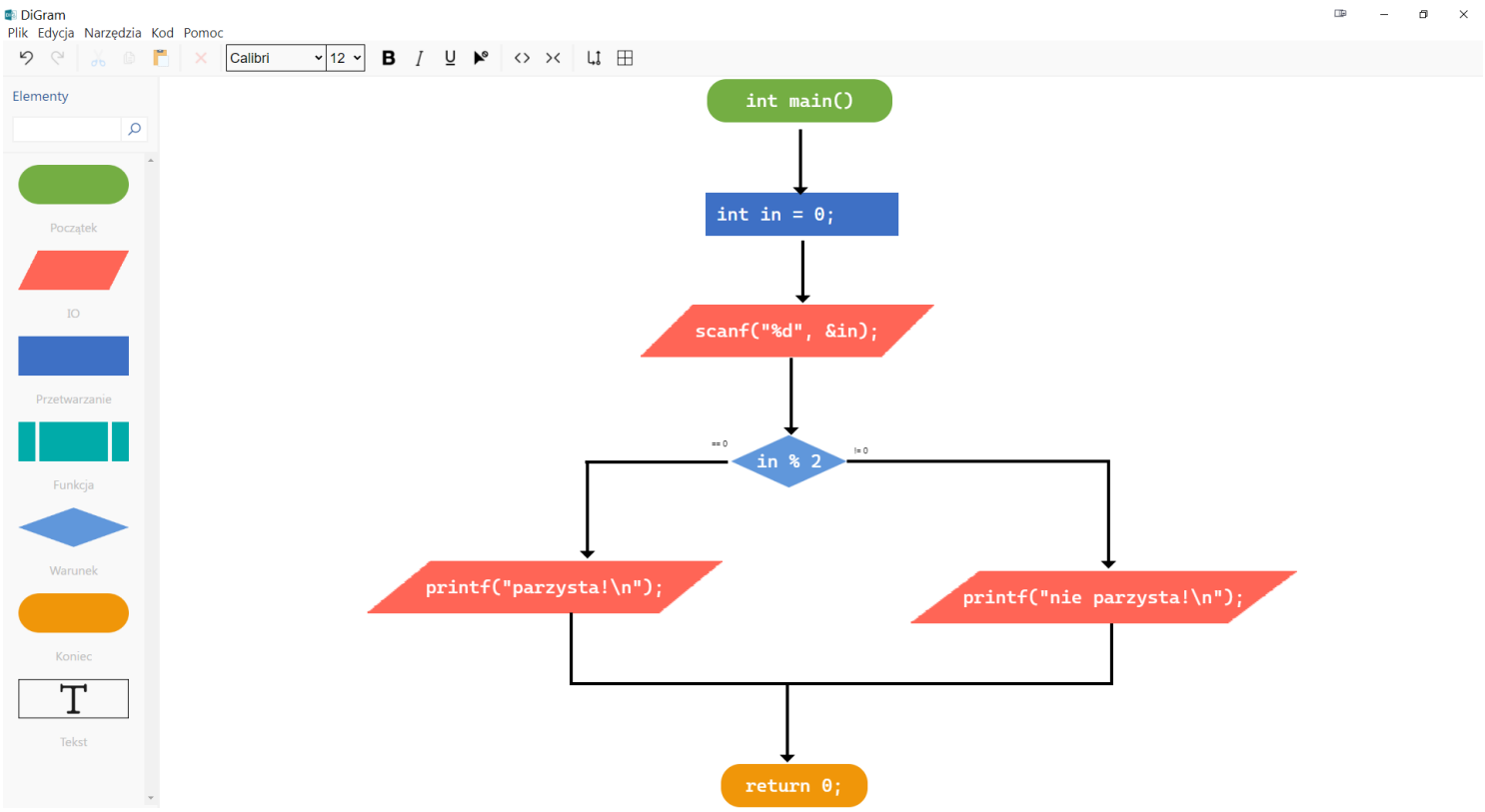
4. PODSUMOWANIE I WNIOSKI

Podsumowując, nasza aplikacja służąca do projektowania schematów blokowych algorytmów z możliwością generowania kodu w języku C jest kompletnym narzędziem, które pozwala użytkownikom w łatwy sposób tworzyć i analizować algorytmy.

Podczas implementacji aplikacji pojawiły się pewne problemy, takie jak konieczność zapewnienia płynnej i responsywnej pracy interfejsu użytkownika w przypadku dużej ilości elementów na schemacie blokowym. Innym problemem była również konieczność opracowania zoptymalizowanej metody generowania kodu w języku C z uwzględnieniem różnych typów algorytmów i operacji.

Jednakże, nasza aplikacja ma potencjał do dalszej rozbudowy. Jednym z możliwych kierunków rozwoju aplikacji jest dodanie funkcjonalności eksportu schematów blokowych algorytmów do większej ilości formatów graficznych, takich jak SVG, BMP, czy PDF. Innym kierunkiem rozwoju jest dodanie możliwości generowania kodu w innych językach programowania, na przykład w języku Python, Java czy C++. Dodatkowo, możemy rozważyć dodanie funkcjonalności, takich jak tworzenie diagramów UML.

Podsumowując, nasza aplikacja do rysowania schematów blokowych algorytmów z możliwością generowania kodu w języku C to narzędzie, które oferuje użytkownikom łatwą i wygodną metodę projektowania algorytmów. Istnieją pewne potencjalne problemy, które należy rozwiązać, ale aplikacja ma również wiele możliwości rozwoju, co pozwoli nam na nieustanne udoskonalanie jej funkcjonalności i zwiększenie komfortu korzystania z niej przez użytkowników.



5. INSTRUKCJA OBSŁUGI APLIKACJI

Instrukcja programu zamieszczona jest w pliku DiGram – instukcja.pdf i dołączona jest do powyższej dokumentacji.