AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INŻYNIERII METALI I INFORMATYKI PRZEMYSŁOWEJ

KATEDRA *Informatyki Stosowanej i Modelowania*

Projekt dyplomowy

Opracowanie oraz implementacja aplikacji desktopowej pomagającej w nauce elektroniki
Development and implementation of a desktop application helping in learning electronics

Autor: *Bartłomiej Gojowczyk*
Kierunek studiów: *Informatyka Techniczna*
Opiekun projektu: *dr inż. Piotr Kustra*

Kraków, *2024*

Spis treści

1 Idea projektu

1.1 Wprowadzenie

Elektronika, w ogólnym znaczeniu tego słowa, jest dziedziną nauki i techniki, która zajmuje się wykorzystaniem zjawisk związanych z sterowaniem kierunku ruchu elektronów [1]. W dzisiejszych czasach wykorzystanie tej dziedziny nauki jest bardzo powszechne, ukierunkowany ruch elektronów występuje w każdym urządzniu, które wymaga prądu elektrycznego do zasilenia. Przykładami takich urządzeń wykorzystywanych na codzień są: telefony komórkowe, komputery, telewizory, sprzęt RTV, czy sprzęt AGD. Przechodząc niżej elektroniką jest analizowanie oraz przetwarzanie napięć i prądów przez urządzenia.

1.2 Cel pracy

Celem pracy jest opracowanie oraz zaimplementowanie aplikacji pomagającej w nauce elektroniki osobom początkującym poprzez praktyczne łączenie komponentów elektronicznych. Układanie elementów oraz ich łączenie na płytce, w aplikacji odzwierciedla projektowanie płytki drukowanej. Pytanie 1: zdanie zgubiłem coś o układaniu elemnentów, ze mało miejsca, tu coś więcej trzebaby dać

1.3 Wymagania funkcjonalne oraz niefunkcjonalne

Wymaganiami funkcjonalnymi systemu są:

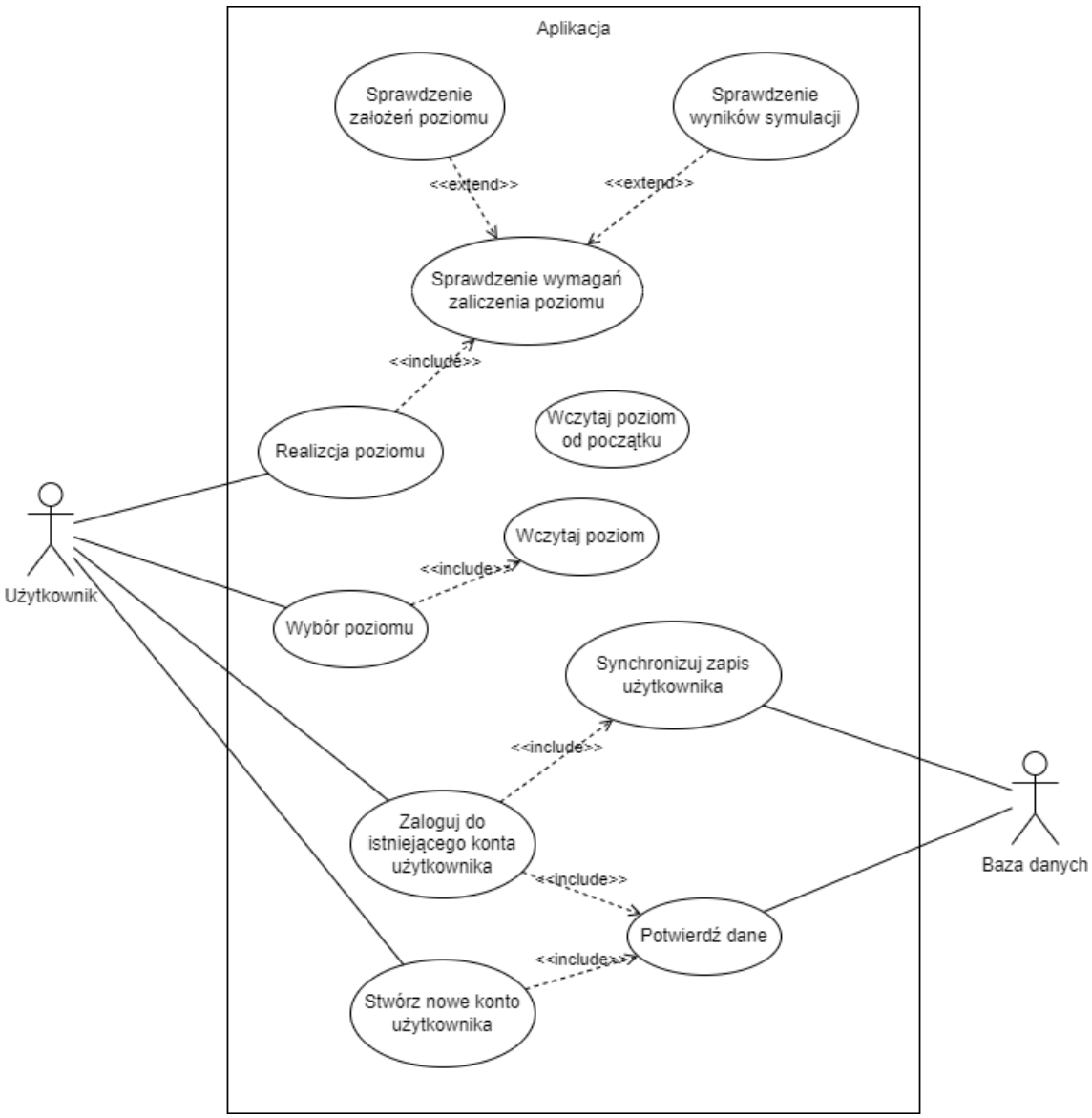
- możliwość utworzenia nowego konta użytkownika
- możliwość zalogowania się na istniejące konto użytkownika
- wyświetlenie opisu komponentu elektronicznego
- wyświetlenie listy zadań do zrealizowania
- wykonanie symulacji układu elektronicznego
- możliwość wybrania poziomu do realizacji
- możliwość połączenia komponentów w dowolny sposób
- możliwość położenia komponentu w dowolnym miejscu na płytce
- synchronizacja z bazą danych

Wymaganiami niefunkcjonalnymi systemu są:

- Połączenie z Internetem (baza danych)

1.4 Przypadki użycia aplikacji

Na rys. 1 przedstawiony został diagram przypadków użycia. Natomiast poniżej rys. 1 przedstowiony został opis poszczególnych przypadków użycia.



Rys. 1: Diagram przypadków użycia

Źródło: opracowanie własne

Przypadek użycia: Stwórz nowe konto użytkownika

Aktor: Użytkownik

Opis: Wprowadzenie danych przez użytkownika do utworzenia konta.

Warunki wstępne: Użytkownik nie jest zalogowany. Widok ekranu początkowego.

Przepływ zdarzeń:

1. Użytkownik przechodzi do ekranu tworzenia konta - kliknięcie przycisku ”Zarejestruj”.

2. Użytkownik wprowadza dane do utworzenia konta.
3. Użytkownik klika przycisk ”Zarejestruj”.
4. Aplikacja sprawdza poprawność danych.
    4.a. Dane wprowadzone przez użytkownika są poprawne.
        4.a.1. Zostaje utworzone nowe konto.
        4.a.2. Zostaje utworzony plik z zapisem.
        4.a.3. Plik z zapisem wysłany jest do bazy danych.
        4.a.4. Wyświetlony zostaje komunikat o utworzeniu konta.
    4.b. Dane wprowadzone przez użytkownika nie są poprawne.
        4.b.1. Wyświetlony zostaje komunikat o błędnych danych.

Przypadek użycia: Sprawdzenie poprawności danych

Aktor: Użytkownik, Baza danych

Opis: Sprawdzenie wprowadoznych danych przez użytkownika

Warunki wstępne: Użytkownik jest nie zalogowany oraz klikną przycisk ”Zaloguj” z widoku logowania lub klikną przycisk ”Zarejestruj” z widoku rejestracji.

Przepływ zdarzeń:

1. Sprawdzenie czy dane nie zawierają niepoprawnych znaków.
2. Dane wprowadzone przez użytkownika są poprawne.
    2.a. Dane wprowadzone przez użytkownika nie są poprawne.
        2.a.1. Zwrócony zostaje komunikat z błędem.
3. Sprawdzenie danych w bazie danych.
    3.a. Dane nie są poprawne.
        3.a.1. Zwrócony zostaje komunikat z błędem.
4. Dane są poprawne.
5. Zwrócona zostaje informacja, że dane są poprawne.

Przypadek użycia: Zaloguj do istniejącego konta użytkownika

Aktor: Użytkownik

Opis: Wprowadzenie danych przez użytkownika do zalogowanie się na istniejące konto.

Warunki wstępne: Użytkownik nie jest zalogowany. Widok ekranu początkowego.

Przepływ zdarzeń:

1. Użytkownik przechodzi do ekranu logowania - kliknięcie przycisku ”Zaloguj”.
2. Użytkownik wprowadza dane do zalogowania.

3. Użytkownik klika przycisk ”Zaloguj”.
4. Aplikacja sprawdza poprawność danych.
    4.a. Dane wprowadzone przez użytkownika są poprawne.
        4.a.1. Użytkonik zostaje zalogowany.
        4.a.2. Wyświetlony zostaje komunikat z pomyślnym zalogowaniem.
    4.b. Dane wprowadzone przez użytkownika nie są poprawne.
        4.b.1. Wyświetlony zostaje komunikat o błędnych danych.

Przypadek użycia: Synchronizuj zapis użytkownika

Aktor: Użytkownik, Baza danych

Opis: Pobranie zapisu z bazy danych.

Warunki wstępne: Użytkonik został pomyślnie zalogowany.

Przepływ zdarzeń:

1. Pobranie pliku z zapisem z bazy danych.

Przypadek użycia: Synchronizuj zapis użytkownika

Aktor: Użytkownik, Baza danych

Opis: Pobranie zapisu z bazy danych.

Warunki wstępne: Użytkownik klikną przycisk zamknięcia aplikacji.

Przepływ zdarzeń:

1. Wysłanie pliku z zapisem do bazy danych.

Przypadek użycia: Wybór poziomu

Aktor: Użytkownik

Opis: Użytkownik wybiera poziom.

Warunki wstępne: Użytkownik znajduje się w widoku początkowym.

Przepływ zdarzeń:

1. Użytkownik klika przycisk ”Rozpoczij”.
    1.a. Użytkownik zalogowany.
        1.a.1. Wyciągnięcie informacji o zrealizowanych poziomach z pliku zapisu.
2. Aplikacja przechodzi do widoku poziomów.
3. Użytkownik klika poziom, który chce zrealizować.
4. Aplikacja wyświetla opis poziomu.
    4.a. Użytkownik klika przycisk ”Rozpocznij od początku”.

4.a.1. Aplikacja wczytuje domyślne ustawienia poziomu.

4.b. Użytkownik klika przycisk ”Wczytaj”.

4.b.1. Aplikacja wczytuje poziom z pliku zapisu.

5. Aplikacja przechodzi do widoku rozwiązywania poziomu.

Przypadek użycia: Wczytaj poziom

Aktor: Użytkownik

Opis: Wczytanie poziomu z pliku.

Warunki wstępne: Użytkownik zalogowany, zapis z bazy danych został zsynchronizowany oraz użytkownik podczas wybierania poziomu kliknął przycisk ”Wczytaj”.

Przepływ zdarzeń:

1. Wyekstrahowanie danych z poziomem (pliku schematu) z pliku zapisu bazodanowego.

1.a. Wyekstrahowany plik jest poprawny.

1.a.1. Wczytanie poziomu z pliku schematu.

1.b. Wyekstrahowany plik nie jest poprawny.

1.b.1. Wyświetlenie komunikatu z błędem.

1.b.2. Wczytanie poziomu z ustawieniami domyślnymi.

Przypadek użycia: Realizcja poziomu

Aktor: Użytkownik

Opis: Użytkownik realizuje poziom

Warunki wstępne: Użytkownik wczytał bądź rozpoczął od nowa poziom.

Przepływ zdarzeń:

1. Aplikacja wyświelta przyciski, które pozwalają sprawdzić poziom, wyświetlić pomoc oraz powrócić do wyboru poziomu.
2. Aplikacja wyświelta opis poziomu oraz cele do zrealizowania.
3. Aplikacja wyświelta przyciski, które pozwalają na dodawanie nowych komponentów.
4. Aplikacja wyświetla przyciski odpowidzlane za dodawanie/usuwanie ścieżki, ukrywanie oraz usuwanie komponentów.
5. Aplikacja wyświetla wczytaną płytkę.

Przypadek użycia: Dodaj ścieżkę

Aktor: Użytkownik

Opis: Użytkownik dodaje ścieżkę na płytce.

Warunki wstępne: Użytkownik wczytał bądź rozpoczął od nowa poziom.

Przepływ zdarzeń:

1. Użytkownik klika przycisk ”Połącz”.
2. Użytkownik trzyma lewy przycisk myszy.
3. Użytkownik przesuwa kursor w miejscu dodania ścieżki.
4. Ścieżka może zostać dodana.
    4.a. Dodanie ścieżki na płytce.

Przypadek użycia: Usuń ścieżkę

Aktor: Użytkownik

Opis: Użytkownik usuwa ścieżkę z płytki.

Warunki wstępne: Użytkownik wczytał bądź rozpoczął od nowa poziom.

Przepływ zdarzeń:

1. Użytkownik klika przycisk ”Połącz”.
2. Użytkownik trzyma prawy przycisk myszy.
3. Użytkownik przesuwa kursor w miejscu usunięcia ścieżki.
4. Ścieżka może zostać usunięta.
    4.a. Usunięcie ścieżki z płytki.

Przypadek użycia: Dodaj komponent

Aktor: Użytkownik

Opis: Użytkownik dodaje nowy komponent elektroniczny na płytce.

Warunki wstępne: Użytkownik wczytał bądź rozpoczął od nowa poziom.

Przepływ zdarzeń:

1. Użytkownik najeżdża myszką na przycisk komponentu, który chce dodać.
2. Aplikacja wyświetla opis komponentu elektronicznego.
3. Użytkownik klika przycisk z komponentem.
4. Aplikacja ukrywa opis komponentu elektronicznego.
5. Użytkownik klika miejsce na płytce.
6. Komponent może zostać dodany.
    6.a. Aplikacja dodaje element na płytkę.

Przypadek użycia: Usuń komponent

Aktor: Użytkownik

Opis: Użytkownik usuwa istniejący komponent elektroniczny z płytki.

Warunki wstępne: Użytkownik wczytał bądź rozpoczął od nowa poziom.

Przepływ zdarzeń:

1. Użytkownik klika przycisk ”Usuń”
2. Użytkownik klika komponent na płytce, który chce usunąć.
3. Komponent może zostać usunięty.
    3.a. Aplikacja usuwa element z płytki.

Przypadek użycia: Zapis poziomu

Aktor: Użytkownik

Opis: Plik schematu zostaje zapisany w pliku z zapisem bazodanowym.

Warunki wstępne: Użytkownik zalogowany oraz wczytał bądź rozpoczął od nowa poziom.

Przepływ zdarzeń:

1. Użytkownik klika przycisk ”Powrót”.
2. Aplikacja tworzy plik schematu.
3. Aplikacja dodaje plik schematu do pliku bazodanowego.

Przypadek użycia: Sprawdzenie wymagań zaliczenia poziomu

Aktor: Użytkownik

Opis: Aplikacja sprawdza czy położone komponenty spełniają wymagania poziomu.

Warunki wstępne: Użytkownik wczytał bądź rozpoczął od nowa poziom.

Przepływ zdarzeń:

1. Użytkownik klika przycisk ”Sprawdź”.
2. Aplikacja sprawdza czy spełnione zostały założenia poziomu.
3. Założenia poziomu zostały spełnione.
    3.a. Założenia poziomu nie zostały spełnione.
        3.a.1. Wyświetlenie komunikatu o błędnym zrealizowaniu poziomu.
        3.a.2. Przerwanie procesu sprawdzania poziomu.
4. Aplikacja sprawdza czy zostały spełnione warunki symulacji
5. Założenia symulacji zostały spełnione.
    5.a. Założenia symulacji nie zostały spełnione.
        5.a.1. Wyświetlenie komunikatu o błędnym zrealizowaniu poziomu.
        5.a.2. Przerwanie procesu sprawdzania poziomu.
6. Aplikacja ustawia flagę, że poziom zostal zrealizowny.

7. Wyświetlenie komunikatu o pomyślnym zrealizowaniu poziomu.

Przypadek użycia: Sprawdzenie wyników symulacji

Aktor: Użytkownik

Opis: Aplikacja wykonuje symulację oraz sprawdza wynik symulacji.

Warunki wstępne: Użytkownik wczytał bądź rozpoczął od nowa poziom, klikną przycisk ”Sprawdź” oraz założenia poziomu zostały spełnione.

Przepływ zdarzeń:

1. Aplikacja tworzy plik schematu.
2. Aplikacja żąda wykonania symulacji.
3. Symulacja została wykonana pomyślnie.
    3.a. Symulacja została zakończona nie powodzeniem.
        3.a.1. Aplikacja zwraca komunikat z błędem.
4. Aplikacja sprawdza warunki symulacji
5. Warunki symulacji zostały spełnione.
    5.a. Warunki symulacji nie zostały spełnione.
        5.a.1. Aplikacja zwraca komunikat z błędem.
6. Aplikacja zwraca komunikat z spełnieniem warunków symulacji.

2 Idea działania aplikacji

2.1 Wprowadzenie

Działanie aplikacji polega na realizowaniu kolejnych poziomów, poprzez układanie elementów elektronicznych na płytce. Płytka jak i elementy elektroniczne zostały przedstawione w rzucie izometrycznym wykorzystując grafikę rastrową w stylu *pixel art*. Każdy poziom zawiera opis działania układu oraz opis połączenia komponentów elektronicznych. Użytkownik ma możliwość zarejestrowania oraz zalogowania się do aplikacji w celu zapisania postępu w ”chmurze”.

Dodatkowym elementem aplikacji jest możliwość wczytania pliku schematu płytki wraz z ułożonymi komponentami do programu symulującego **LTspice**, aby użytkownik miał możliwość dokładniejszej analizy.

Rys. 2: Schemat blokowy pętli głównej

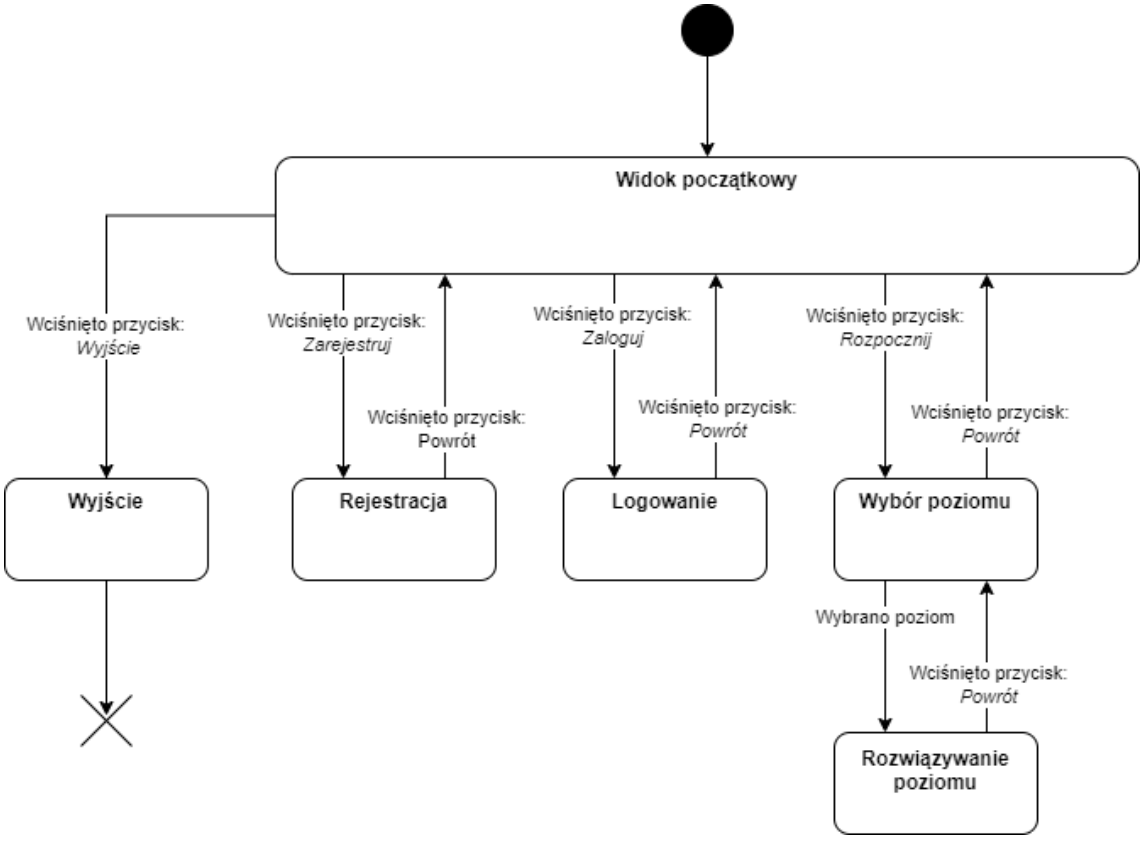
Źródło: opracowanie własne na podstawie [2]

Procesor wykonuje instrukcje w sposób sekwencyjny, więc do zachowania ciągłości renderowania obrazu wymagane jest, by aplikacja zawierała nieskończoną pętlę, w której jest zawarte sprawdzanie zdarzeń, aktualizowanie aplikacji oraz rysowanie okna [2]. Pierwszym etapem pętli jest przetwarzanie zdarzeń, czyli np. sprawdzenie i wykonanie odpowiednich instrukcji np. czy użytkownik wcisnął przycisk zamknięcia okna, czy aplikacja utraciła fokus. Warunek sprawdzania fokusu okna pozwala na nie wykorzystywanie zasobów sprzętowych, w przypadku gdy okno jest nie aktywne,

bądź jest zminimalizowane. W następnym etapie następuje aktualizacja aplikacji, czyli przetworzenie wszytkich instrukcji logicznych. Ostatnim etapem jest czekanie, aby aplikacja nie wykonywała się za szybko. W tej aplikacji zostało przyjęte, że maksymalna liczba klatek na sekundę wynosi 60, więc pętla się wykonuje 60 razy na sekundę. Aplikacja nie wykonuje się szybciej, ponieważ zwiększenie generowania maksymalnej liczby klatek dałoby niezauważalne różnice, a znacznie zwiększyłoby wykorzystanie sprzętu. Schemat blokowy głównej pętli został przedstawiony na rys. 2.

2.2 Widoki aplikacji

Aplikacja składa się z widoków, gdzie w każdym widoku są możliwe do wykonania inne czynności przez użytkownika. Przechodzenie między różnymi widokami odbywa się poprzez kliknięcie odpowiednie kliknięcie przycisku z adekwatną nazwą do danego widoku. W każdym widoku poza widokiem ”Wyjście” oraz ”Widok początkowy” jest przycisk, który pozwala na powrót do poprzedniego widoku. Możliwe widoki zostały przedstawione na rys. 3.



Rys. 3: Diagram maszyny stanów obiektu klasy *Game*

Źródło: opracowanie własne

2.3 Logowanie i rejestracja użytkownika nwm jak nazwać

Użytkownik zanim będzie mógł się zalogować musi utworzyć konto, aby mógł to zrobić musi przejść do widoku rejestracji, a następnie wprowadzić login, e-mail i dwukrotnie hasło oraz nacisnąć przycisk rejestracji. Po kliknięciu przycisku następuje sprawdzenie poprawności wprowadoznych danych, czy dane zawierają puste znaki, czy dane zawierają znaki niedozwolone. Gdy dane są nie poprawne to zostaje zwrócony komunikat z błędnymi danymi. W przeciwnym wypadku aplikacja wysyła zapytanie do bazy danych w celu sprawdzenia czy istnieje konto z takim loginem w bazie danych. Gdy nie istnieje to zostaje dodany nowy użytkownik do bazy danych, a następnie tworzony jest plik bazodanowy i wysyłany do bazy danych. Diagram czynności przedstawiający proces tworzenia nowego konta przedstawiony jest na rys. 5

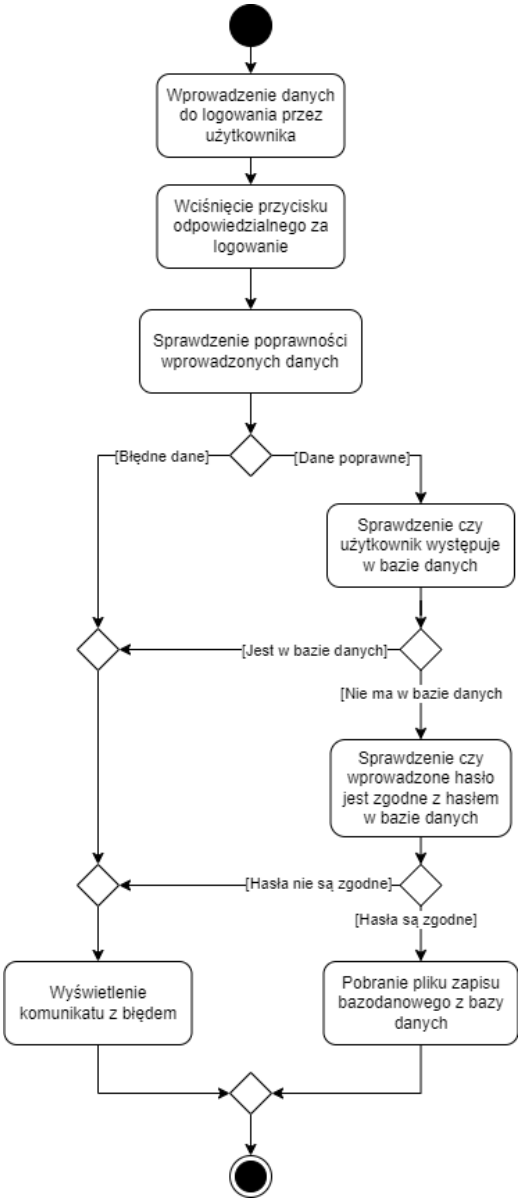
Użytkownik ma możliwość zalogowania, aby było to możliwe uzytkownik musi przejść do widoku logowania, a następnie wprowadzić login i hasło oraz nacisnąć przycisk logowania. Po kliknięciu przycisku aplikacja najpierw sprawdza poprawność wprowadoznych danych, tak jak w przypadku rejestracji. Następnie aplikacja wysyła zapytanie do bazy danych, w celu sprawdzenia czy taki użytkownik istnieje. Gdy login nie zostaje odnaleziony w bazie danych aplikacja wyświetla stosowny komunkat, w przeciwnym wypadku sprawdzane jest hasło w badzie danych. Wyświetlany jest komunikat z błędem w przypadku błędnego hasła. Jeśli dane są poprawne to użytkownik zostaje zalogowany. Diagram czynności przedstawiający proces logowania jest przedstawiony na rys. 4.

2.4 Realizacja poziomu

Aby możliwa była realizajca poziomu użytkownik musi przejść do widoku wybierania poziomu. Jeśli użytkownik jest zalogowany to wczytywane są informacje z pliku bazodanowego o zapisanych poziomach. Pozwalają one określić, które z poziomów zostały zrealizowane. Rys. 6 przedstawia diagram czynności podczas przejścia do widoku wybierania poziomu. Następnie użytkownik wybiera poziom do realizacji po kliknięciu na poziom, wyświetlany jest opis poziomu. Jeśli użytkownik jest zalogowany to ma możliwość wczytania wcześniej zapisanego poziomu bądź rozpoczęcia go od początku. W przypadku jak użytkownik nie jest zalogowany to ma możliwość tylko rozpoczęcia poziomu od początku. Na rys. 7 przedstawiony został diagram czynności, który pokazuje,

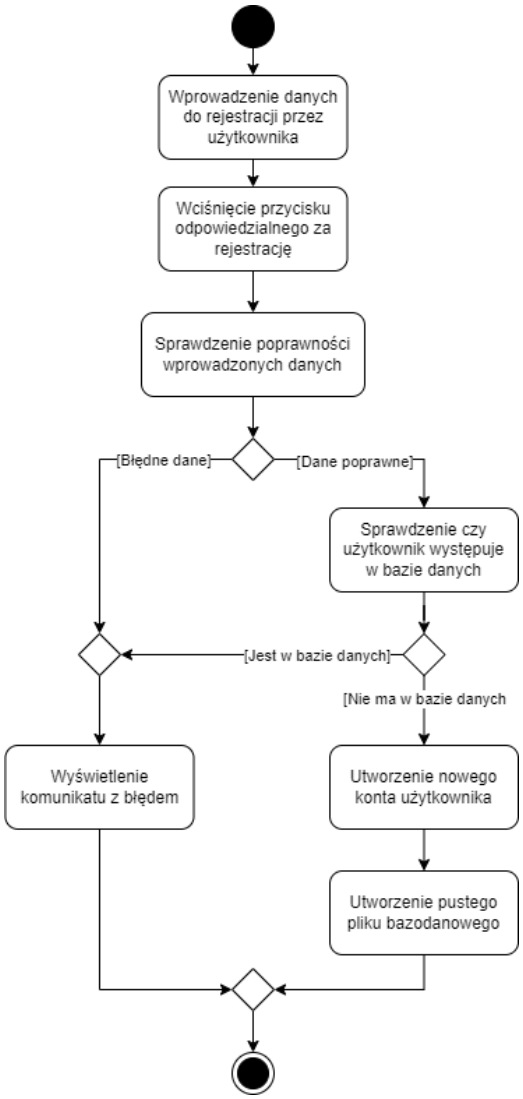
Użytkownik podczas realizacji poziomu dodaje elementy elektroniczne na płytkę, łączy komponenty ścieżkami. Rys. 8 zawiera diagram czynności pokazujący możliwe czynności do wykonania podczas realizowania poziomu oraz jakie warunki muszą zostać zpełnione, by możliwe było dodanie, usunięcie komponentu czy ścieżki.

Po klinkięciu przez użytkownika przycisku sprawdzającego następuje sprawdzenie poziomu.



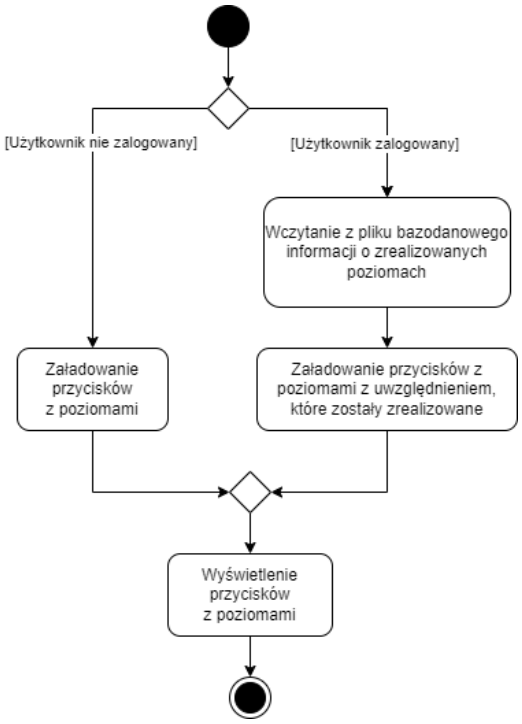
Rys. 4: Diagram czynności podczas logowania użytkownika

Źródło: opracowanie własne



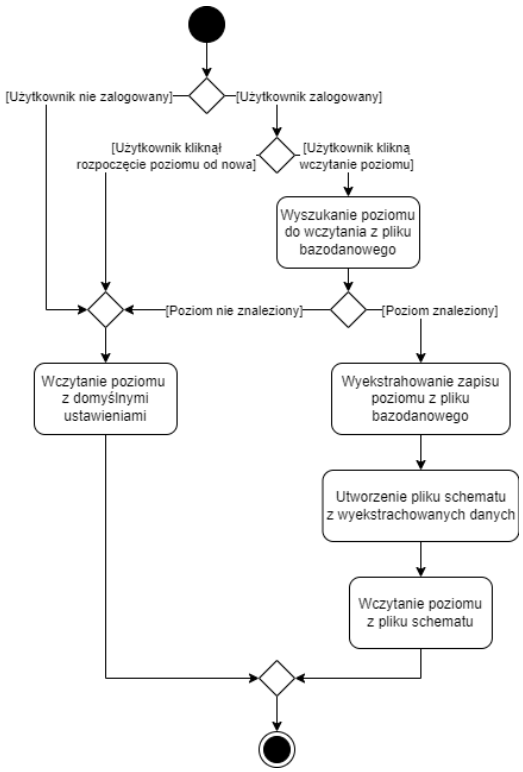
Rys. 5: Diagram czynności podczas rejestracji użytkownika

Źródło: opracowanie własne



Rys. 6: Diagram czynności podczas wybierania poziomu przez użytkownika

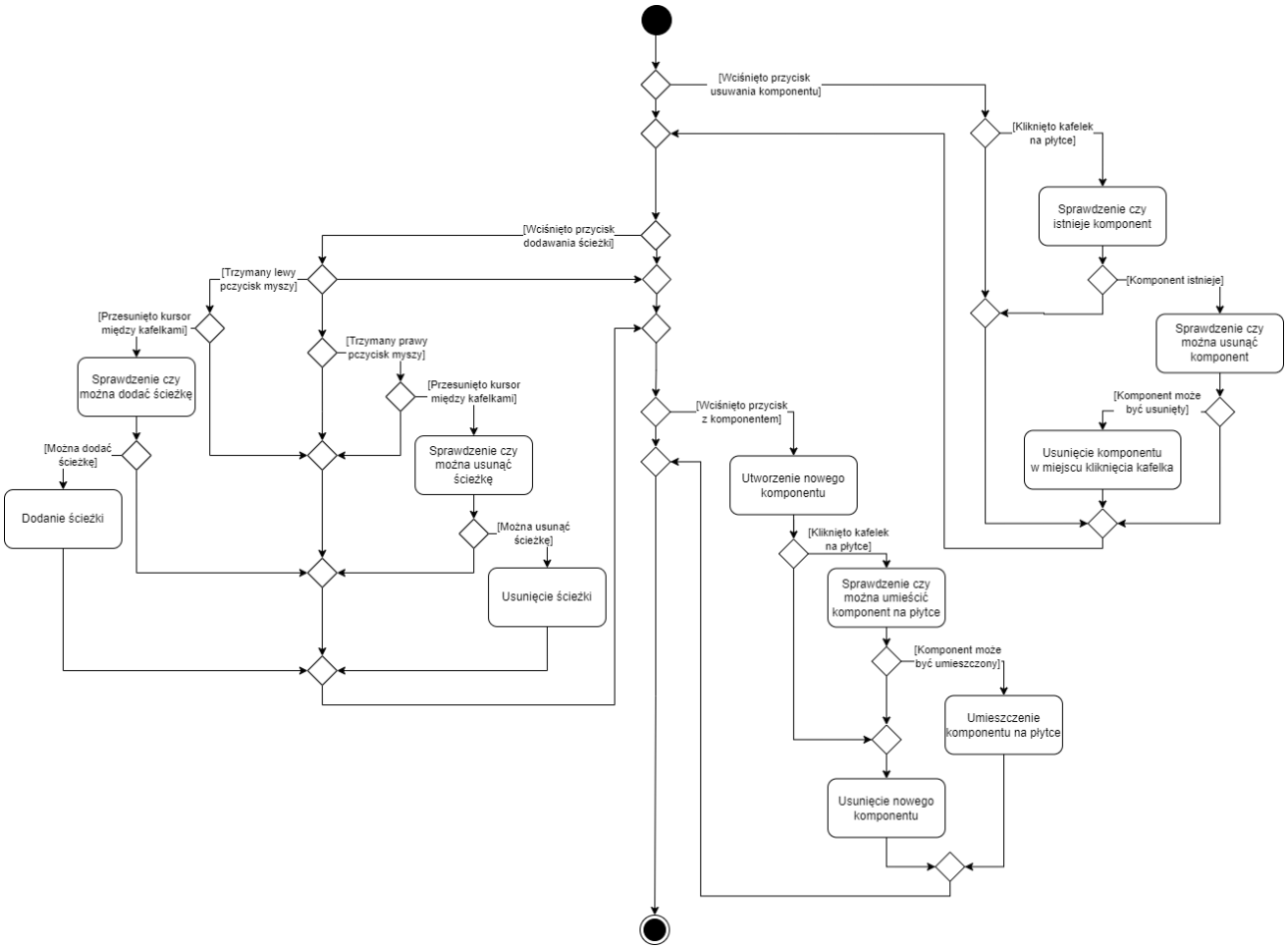
Źródło: opracowanie własne



Rys. 7: Diagram czynności podczas wczytywania poziomu

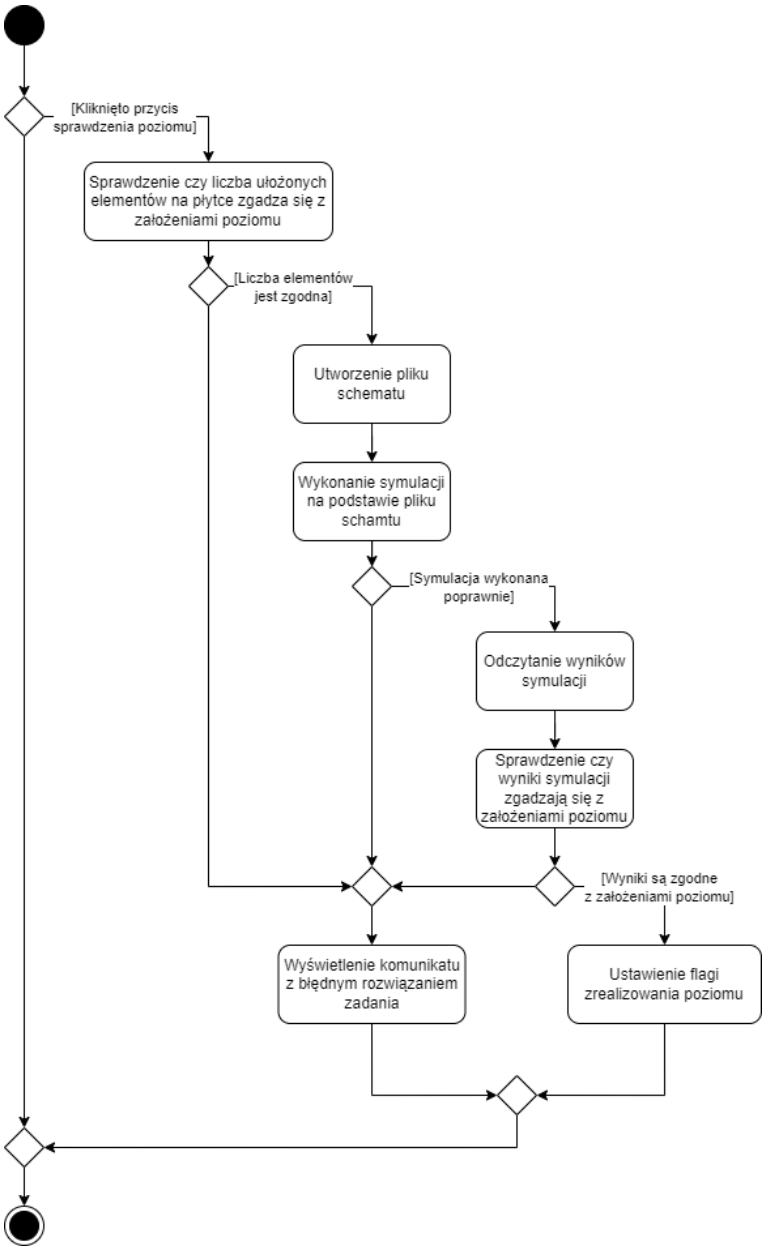
Źródło: opracowanie własne

Pierwsze sprawdzenie polega na weryfikacji czy umieszczone elementy zgadzają się z założeniem zadania. Następnym sprawdzeniem jest sprawdzenie wyników symulacji, aby móc je wykonać aplikacja tworzy plik schematu, następnie wykonuje symulację na podstawie tego pliku. Diagram czynności pokazujący sprawdzanie poziomy jest na rys. 9.



Rys. 8: Diagram czynności podczas realizacji poziomu przez użytkownika

Źródło: opracowanie własne



Rys. 9: Diagram czynności podczas sprawdzania poziomu

Źródło: opracowanie własne

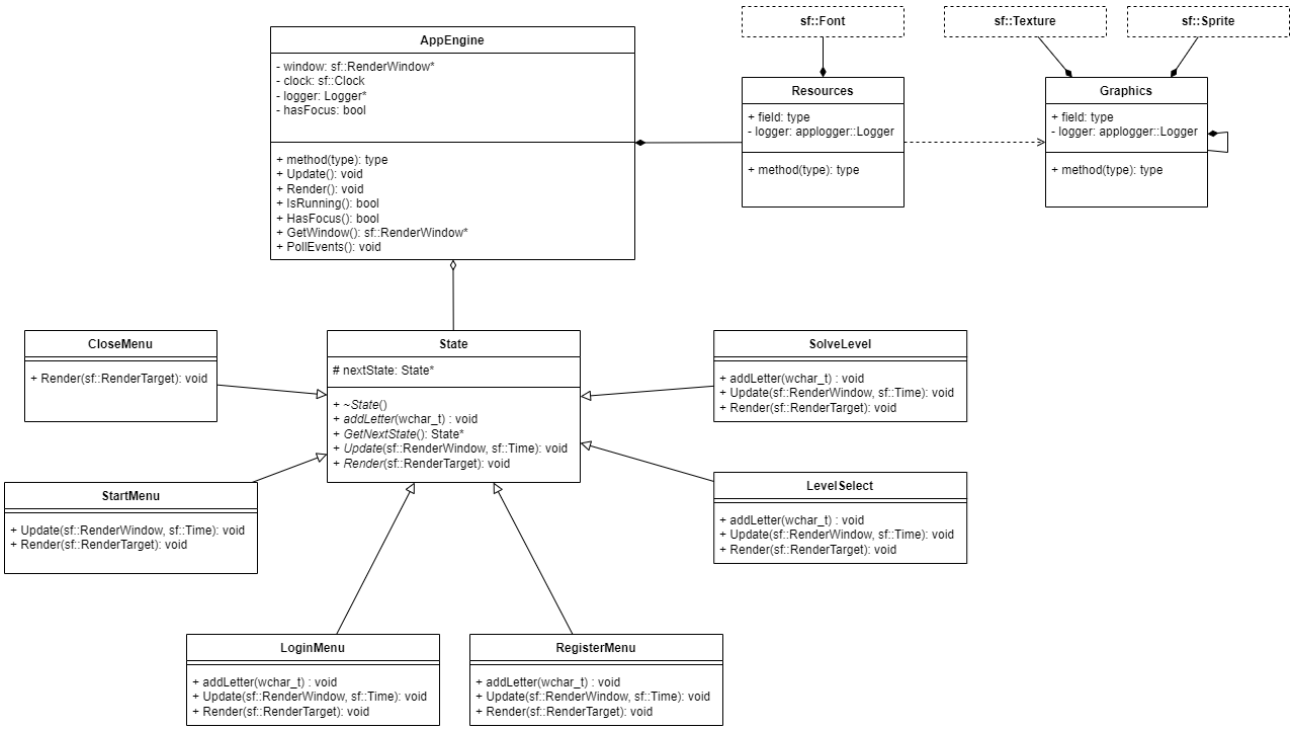
3 Projekt aplikacji

3.1 Przechodzenie między widokami

Pytanie 2: w jaki sposób powinienem odwoływać się do nazw klas/obiektów, kursywą?

Wykorzystany został czynnościowy wzorzec projektowy stan, który pozwala na to, że obiekt może zmieniać swoje zachowanie w zależności od swojego stanu wewnętrznego. Dzięki zastosowaniu tego wzorca projektowego możliwe jest przechodzenie pomiędzy widokami aplikacji (rys. 3). Rys. 10 przedstawia diagram klas, w którym wykorzystany został ten wzorzec. Klasa *AppEngine* jest główną klasą, która zawiera metody niezbędne do wykonywania pętli głównej (rys. 2). W metodach *Update(), Render()* wykonywane są odpowiednie metody z klasy *State*. Klasy dziedziczące po klasie *State* zmieniają odziedziczone pole *nextState*, które pozwala na zmianę zachowania aplikacji bez zmiany klasy w obiekcie klasy *AppEngine*.

Główną klasą odpowiedzialną za



Rys. 10: Uproszczony diagram klas przedstawiający klasy poszczególnych widoków oraz klasy zawierające grafikę

Źródło: opracowanie własne

3.2 Realizowanie poziomu

Po wybraniu przez użytkownika poziomu, aby możliwe było zrealizowanie poziomu użytkownik musi w ułożyć elementy elektroniczne oraz w odpowiedni sposób je połączyć. Dodawanie elementu na płytkę zostało przedstawione na rys. 11. Natomiast dodwanie połączeń zostało przedstawione na rys. **??**. Po naciśnięciu odpowiedniego przycisku aplikacja podczas trzymania wciśniętego lewego klawisza myszy oraz przesuwania po płytce będzie łączyła sąsiadujęce pola.

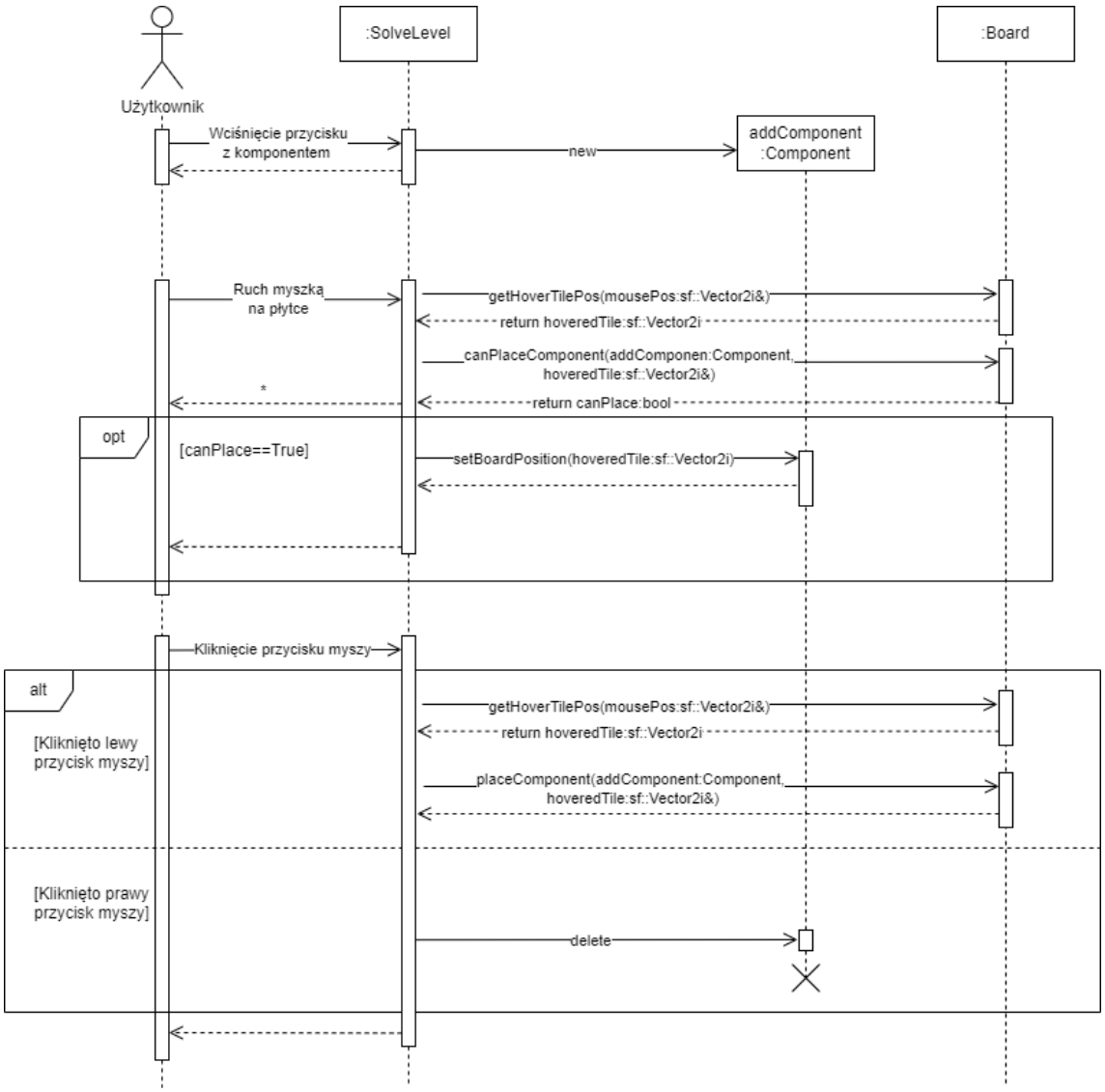
Płytka na, której układane są komponenty elektroniczne składa się z różnej ilości kafelek Pytanie 3: czy dobrze odmieniłem? , w zależności od poziomu. Wielkość kafelka na ekranie, uwzględniając tylko obszar z góry wynosi 64 pixele szerokości oraz 32 pixele wysokości. Do określenia, na który kafelek użytkownik wskazuje należy odczytać pozycję kursora od okna, następnie przeliczyć, uwzględniając przesunięcie płytki, na który kafelek w rzucie izometrycznym wskazuje. Kafelki (obiekty klasy *Tile*) przechowywane są w dynamicznie tworzonej tablicy jednowymiarowej tablicy.

Gdy użytkownik ułoży komponenty na planszy, a następnie wciśnie przycisk sprawdzający zadanie, to aplikacja utworzy plik z schematem. Następnie aplikacja wykona symulację wykorzystując zewnętrzny, darmowy program **LTspice** oraz wygenerowany plik. Rys. 12 przedstawia diagram sekwencji zawierający interakcję między obiektami, gdy aplikacja będzie sprawdzała poziom.

3.3 Baza danych

Gdy zalogowany użytkownik wyjdzie z poziomu, to apliakcja zapisze wygenerowany plik z schematem w pliku z bazodanowym plikiem użytkownika. Struktura zapisanego pliku została przedstowiona na rys. 13. Pierwszy segment pliku składa się z 60 bajtów zawierających dodatkowy identyfikator użytkownika oraz 4 bajtów zawierających liczbę zapisanych schematów w pliku. Kolejnymi segmentami są zapisane schematy wraz z ich identyfikacją. Pierwsze 8 bajtów segmentu z schematem odpowiada identyfikatorowi poziomu, kolejny bajt zawiera flagi dotyczące zapisanego poziomu np. czy poziom został ukończony. Następne 8 bajtów odpowiada liczbie bajtów, które zawierają zapis schematu. Ostatnia część segmentu zawiera bajty z schematem. Zaletą wykorzystania jednego pliku jest zminimalizowanie liczby zapytań do bazy danych. Pierwsze zapytanie następuje podczas logowania użytkownika (założone tutaj zostało, że użytkownik wcześniej utworzył konto), pobrania z bazy danych pliku z zapisami oraz podczas wysłania zaaktualizowanego pliku z zapisami.

Struktura bazy danych jest przedstawiona na rys. 14. Główną tabelą jest tabela *Users*, w której zawarte są dane użytkowników niezbędne do zalogowania. Natomiast w tabeli *Saves* zawarte zostały zapisy użytkowników. Tabelę zostały połączone relacją jeden do jednego, ponieważ jeden użytkownik może mieć tylko jeden zapis postępu wszsytkich poziomów. Powodem, dla którego tabele zostały



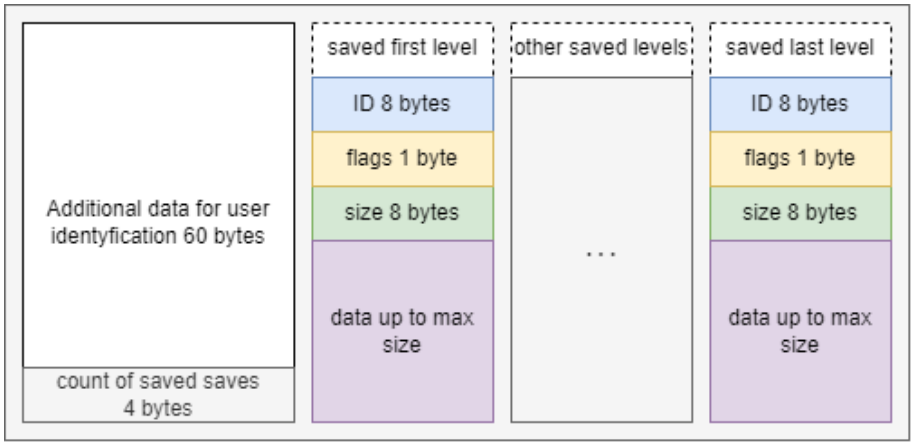
Rys. 11: Diagram sekwecji pokazujący, gdy użytkownik chce dodać element

Źródło: opracowanie własne

tak podzielone jest rozdzielenie zapytań podczas logowania i tworzenia konta od zapytań synchronizujących pliki zapisu. W tym przypadku, gdzie plik zapisu nie jest duży takie rozwiązanie jest wystarczające. Natomiast w przypadku większych plików zapisanych w bazie danych czas odpowiedzi byłby znacznie dłuższy. Dobrym rozwiązaniem byłoby przechowywanie plików zapisu w innej części serwera, a w bazie tylko odniesienie do miejsca, w którym jest plik zapisu.

Rys. 12: Diagram sekwencji, pokazujący interakcję podczas symulacji

Źródło: opracowanie własne



Rys. 13: Struktura bazodanowego pliku zapisu

Źródło: opracowanie własne

Rys. 14: Struktura bazy danych

Źródło: opracowanie własne

4 Implementacja aplikacji

4.1 Wprowadzenie

Główna część aplikacji została zaimplementowana w języku C++, wykorzystując standard ISO C++ 14. Do komunikacji z zewnętrznym programem wykorzystany został język Python w wersji 3.12 osadzony w głównej części aplikacji. Takie rozwiązanie pozwala na bezpośredni dostęp do zasobów wykonywanego kodu w języku Python przez aplikajcę w C++. Aplikacja została zaimplementowana z myślą o systemie operacyjnym Windows, natomiast wykorzystane biblioteki pozwalają na przeniesie na inne systemy operacyjne dystrybucji Linux czy macOS. Pytanie 4: not sure

Wykorzystaną bazą danych jest MySQL.

4.2 Środowisko oraz narzędzia programistyczne

Wykorzystanym środowiskiem programistycznym było Visual Studio 2022. Zostały wykorzystane moduły: *Graphics, System, Window* biblioteki SFML *(Simple and Fast Multimedia Library)* opartej o OpenGL. Wykorzystanie tych modułów pozwala na proste i wydajne rysowanie okna na ekranie oraz przekazywanie własnych parametrów do strumienia przetwarzania potoku grafiki. Zastosowana została również standardowa biblioteka STL.

Aplikacja w celu przekazania i odebrania danych z symulacji wykorzystuje bibliotekę Python. Natomiast w zagnieżdżonej części Python wykorzystuje moduł PyLTSpice do realizacji symulacji oraz moduł ltspice do odczytania obliczonych wartości symulacji.

Do wykonywania zapytań z bazą danych wykorzystana została biblioteka mysqlx. Baza danych była uruchamiana w środowisku kontenerowym Docker, do inicjalizacji tabel oraz relacji wykorzystany został język SQL.

Wykorzystany został program Git oraz serwis github.com, do wersjonowania kodu. Do opracowania diagramów wykorzystany został program draw.io. Grafiki zostały stworzone w programie GIMP.

4.3 Implementacja poziomu

Każdy poziom zawiera funkcję, w której znajduje się tworzenie obieku klasy Level zawierającego identyfikator, nazwę, opis. Zawiera dodanie funkcji tworzenia komponentów oraz funkcji sprawdzających, czy dany poziom jest zrealizowany poprawnie. Implementacja pierwszego poziomu została przedstawiona w fragmencie kodu 1.

```
1 Level* loadLevel0()
```

```
{
  Level* level = new Level(L"P0", L"ąPocztki", L"ąPocztkowy poziom, \nktóry łżsuy
    do zapoznania ęsi z ąaplikacj.", false);
  level->setPathToSave("save.asc");
  level->setGenerateComponents(([](int* componentsCount)->Component** {
    *componentsCount = 2;
    Component** components = new Component * [*componentsCount];
    Vector2i* tmp = new Vector2i[2];
    tmp[0].x = 0; tmp[0].y = 0;
    tmp[1].x = 1; tmp[1].y = 0;
    components[0] = new Resistor(L"Opornik", L"Zamienia ęśćcz energii
   elektrycznje w łciepo", Vector2i(2, 1), 2, tmp, GraphicAll::GetInstance().
   getResistorTexture(), Component::ComponentTypePackage::SMD, true, "id0");
    components[1] = new LedDiode(L"", L"", Vector2i(2, 1), 2, tmp, GraphicAll::
   GetInstance().getDiodeTexture(), Component::ComponentTypePackage::SMD, true,
   "id1");
    delete[] tmp;
    return components;
    }));

  level->setCheckBoard([](Board* board)->bool { ... });

  level->setCheckSimulation([](Board* board)->bool { ... });

  return level;
}
```

Fragment kodu 1: Funkcja zawierająca stworzenie pojedyńczego poziomu

Źródło: opracowanie własne

5 Uruchomienie aplikacji

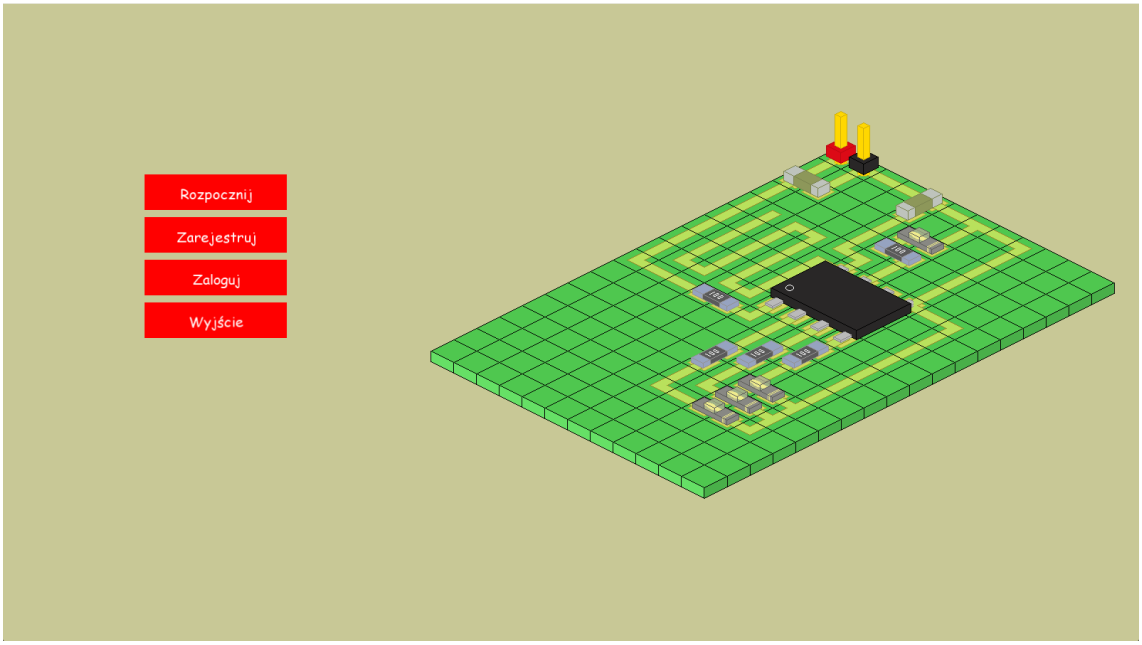
5.1 Wymagania dotyczące uruchamiana aplikacji

Uruchomienie aplikacji wymaga dynamicznie linkownych bibliotek dostarczanych wraz z każdą biblioteką. Dodatkowym wymaganiem jest zainstalowana aplikacja LTspice w domyślnej lokalizacji oraz zainstalowany interpreter języka Python w wersji 3.12.

5.2 Widoki aplikacji

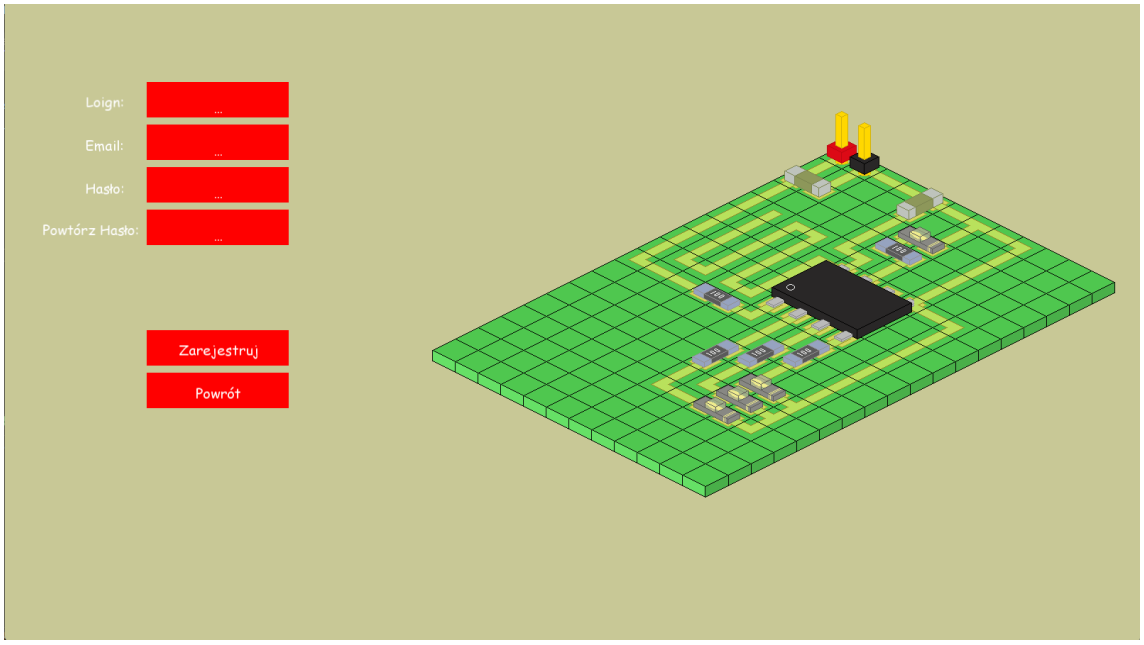
Na rys. 15 przedstawiony został widok po uruchomieniu aplikacji. Za wyjście z aplikacji odpowiada przycisk *Wyście* natomiast pozostałe odpowiadają za przejście do odpowiadających widoków. Widok tworzenia konta został przedstawiony na rys. 16, a widok logowania użytkownika na rys. 17. W przypadku obu widoków, walidacja wprowadzonych danych następuje po wciśnięciu przycisku oraz stosowny komunikat jest wyświetlony.

W przypadku kliknięcia przycisku *Rozpocznij* w widoku początkowym, niezależnie od zalogowania użytkownika następuje przejście do widoku wyboru zadania, widok przedstawiony na rys. **??**.



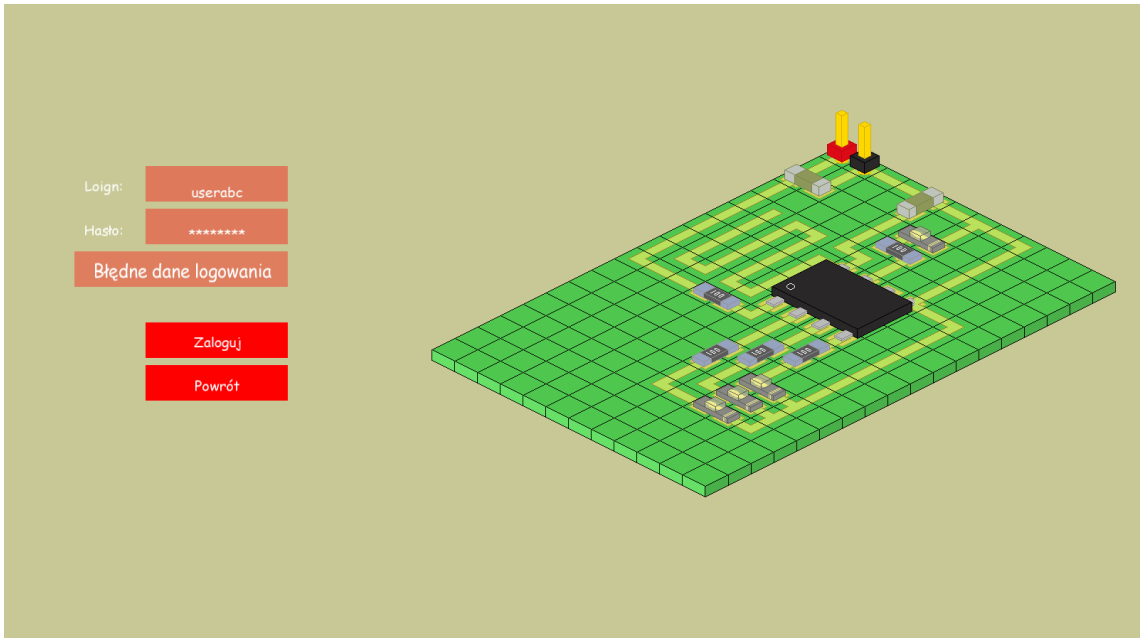
Rys. 15: Widok aplikacji po uruchomieniu

Źródło: opracowanie własne



Rys. 16: Widok aplikacji po przejściu do widoku rejestracji

Źródło: opracowanie własne



Rys. 17: Widok aplikacji po przejściu do widoku logowania po podaniu błędnych danych logowania

Źródło: opracowanie własne

6 Testowanie aplikacji

6.1 Testowanie poziomu

6.2 Testownie komunikacji z bazą danych

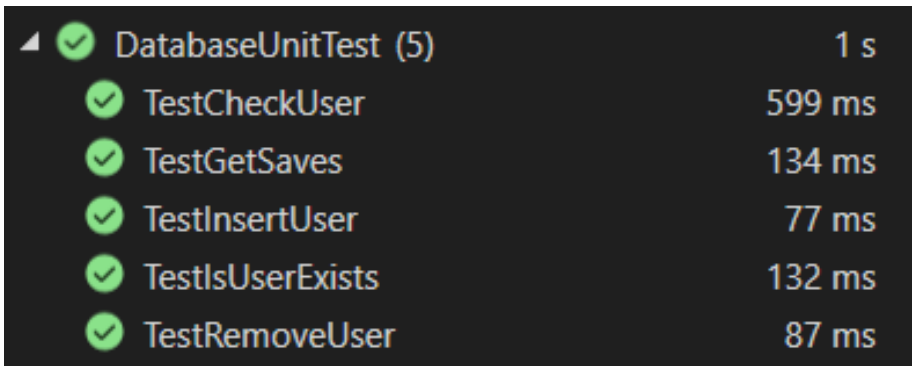
Do przetestowania podstawowej komunikacji z bazą danych utworzone zostały odpowiednie testy. Przetestowane zostało dodawanie nowego użytkownika do bazy danych, usuwanie użytkownika, sprawdzanie czy użytkownik istnieje w bazie, sprawdzanie hasła użytkownika oraz pobieranie pliku zapisu z bazy danych. Fragment kodu 2 zawiera funkcję testująca dodawanie nowego użytkownika do bazy. Rezultat testów sprawdzających komunikcaję z bazą danych widoczny jest na rys. 18.

```
TEST_METHOD(TestInsertUser)
{
  DatabaseConnector dc;
  std::string login = "user1";
  std::string pass = "pass";

  try {
    Assert::IsTrue(dc.insertUser(login, pass));
  }
  catch (const std::string&) {
    Microsoft::VisualStudio::CppUnitTestFramework::Logger::WriteMessage("User
    exists!");
  }

  auto func = [&] {dc.insertUser(login, pass); };
  Assert::ExpectException<std::string>(func);

  login = "us er1";
  pass = "pa ss";
  auto func2 = [&] {dc.insertUser(login, pass); };
  Assert::ExpectException<std::string>(func2);
}
```

Fragment kodu 2: Fragment funkcji testującej zapis, synchronizację z bazą danych oraz odczyt pliku z schematem

Źródło: opracowanie własne

Fragment kodu 3 testuje utworzenie obiektu Board, dodanie komponentu oraz zapisanie pliku schematu. Następnie plik schematu zostaje dodany do pliku bazodanowego oraz wysła do bazy



Rys. 18: Wynik wykonanych testów operacji na bazie danych

Źródło: opracowanie własne

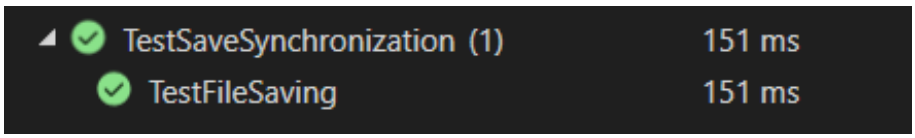
danych. Następnie pobiera plik z bazy danych, wyekstrahuje plik z schematem oraz wczytuje plik z schematem do nowo utworzonego obiektu Board. Porównuje zawartości obiektów przed wysłaniem oraz po synchronizacji. Wynik testu jest widoczny na rys. 19.

```
TEST_CLASS(TestSaveSynchronization)
{
  TEST_METHOD(TestFileSaving)
  {
    // Login user
    ...
    Board* board = new Board(20, 20, 1);
    BoardSave::getInstance()->saveBoard(board, "save.asc");

    Vector2i* tmp = new Vector2i[1];
    tmp[0].x = 0;
    tmp[0].y = 0;
    Vector2i pinPos = {3, 3 };

    Level* level = loadLevelSTART();
    level->load();
    Component** components = level->getComponents();
    board->placeComponent(new Resistor(dynamic_cast<Resistor*>(components[0])),
   pinPos);
    delete[] tmp;

    BoardSave::getInstance()->saveBoard(board, "save.asc");
    Level::saveRealizedLevel(level->getId(), 0);
    User::getInstance().syncSavesFile();

```

```
25      User::getInstance().getSavesFile();
26      Level::extractRelizedLevel(level->getId());
27
28      Board* newBoard = BoardSave::getInstance()->loadBoard("save.asc", level);
29
30      for (int i = 0; i < 20; i++)
31      {
32        for (int j = 0; j < 20; j++)
33        {
34          Component* component = board->getComponentOnBoard({ i,j });
35          Component* newComponent = newBoard->getComponentOnBoard({ i,j });
36          if ((component != nullptr) && (newComponent != nullptr))
37            Assert::IsTrue(board->getComponentOnBoard({ i,j })->getId() == board->
    getComponentOnBoard({ i,j })->getId());
38        }
39      }
40      // Clean and remove user
41      ...
42    }
43  };
```

Fragment kodu 3: Fragment funkcji testującej zapis, synchronizację z bazą danych oraz odczyt pliku z schematem

Źródło: opracowanie własne



Rys. 19: Wynik wykonanego testu fragmentu kodu 3

Źródło: opracowanie własne

Pytanie 5: Czy dla różnych testów mieć różne zrzuty z wynikami testów czy jedne zbiorczy zrzut z wszsytkimi testami?

6.3 Podsumowanie

Wykonane testy

Literatura

[1] Słownik Języka Polskiego PWN https://sjp.pwn.pl/slowniki/elektronika.html

[2] Game Programming Patterns, Robert Nystrom, https://gameprogrammingpatterns.com/game-loop.html