# DOKUMENTACJA TECHNICZNA PROJKETU

Symulator pożarów

Autorzy:

YY.XX.2024

**Spis Treści**

1. **Wstęp**
   * Cel projektu
   * Opis ogólny
   * Zakres funkcjonalności
2. **Środowisko i wymagania**
   * Wymagania sprzętowe
   * Wymagania programowe
   * Instalacja i konfiguracja
3. **Struktura projektu**
   * Struktura katalogów
   * Pliki źródłowe i ich funkcje
4. **Szczegóły implementacji**
   * Główne komponenty agregujące dane
     + VectorSim
     + VectorPozarPierwotny
     + VectorPozarRozprzestrzeniony
     + Ubezpieczyciel
   * Struktury danych
     + Data
     + BuforPierwotny
     + BuforRozprz
   * Globalne zmienne i ich rola
   * Funkcje pomocnicze
     + randZeroToOne
     + sample\_vec
     + randBin
     + contains
     + search\_closest
     + percentage\_of\_loss
     + calc\_reas\_bligator
     + reasecuration\_build\_fire
     + index\_spread\_build
     + haversine\_cpp
     + index\_in\_ring
     + haversine\_loop\_cpp\_vec
     + calc\_brutto\_ring
     + reasurance\_risk
     + calc\_reas\_obliga\_event
   * Funkcje wczytujące dane
     + processPrPozaru
     + processPrRozprzestrzenienia
     + processPrWielkoscPozaru
     + processOblig
     + processBudynki
     + processReas
   * Funkcje główne
     + simulateExponsureTEST
     + testALL
5. **Wczytywanie Danych**
   * Opis funkcji do wczytywania danych
   * Struktura i format danych wejściowych
   * Czas wczytywania danych
6. **Wątki i synchronizacja**
   * I nicjalizacja puli wątków i ich wykorzystanie
     + Opis obiektów pool i poolFiles
   * Konfiguracja liczby wątków
     + watekZapisPierwotny
     + watekZapisRozprz
   * Synchronizacja za pomocą mutexów i condition variables
   * Opis wątków roboczych do przetwarzania danych i zapisu CSV
7. **Obsługa CSV**
   * 7.1. Funkcje do zapisu do pliku CSV (np. zapiszDoCSV)
   * 7.2. Opis struktury plików CSV i formatu wyjścia
8. **Simulacja**
   * Opis procesu symulacji
   * Parametry symulacji
   * Przegląd logiki symulacji (np. simulateExponsureTEST)
   * Wyjaśnienie, jak wykonywane są symulacje i jak agregowane są wyniki
   * Rola simulateExponsureTEST
9. **Zapis Wyników**
   * Format wyników (CSV)
   * Struktura katalogów i plików
   * Funkcje pomocnicze do zapisu
10. **Instrukcje użytkowania**
    * 10.1. Jak skompilować i uruchomić program
    * 10.2. Przykładowe argumenty wiersza poleceń (jeśli dotyczy)
11. **Interfejs użytkownika**
    * Konfiguracja ImGui
    * Funkcje renderujące GUI
    * Obsługa zdarzeń
12. **Przykłady użycia**
    * Uruchomienie symulacji
    * Wczytywanie danych
    * Zapis wyników
13. **Testowanie i debugowanie**
    * Metody testowania
    * Narzędzia do debugowania
14. **Podsumowanie**
    * Wnioski
    * Możliwe usprawnienia
15. Wstęp

**1.1 Cel projektu**

Celem projektu jest stworzenie symulatora pożarów, który pozwala na analizę ryzyka wystąpienia pożaru na danym obszarze dla podanych ubezpieczycieli. Symulator wykorzystuje dane dotyczące położenia budynków, ryzyka pożarowego oraz parametrów reasekuracyjnych do przeprowadzenia symulacji, które umożliwiają ocenę potencjalnych strat.

**1.2. Opis ogólny**

Projekt składa się z kilku kluczowych komponentów, w tym klas do przechowywania danych, funkcji do przetwarzania danych wejściowych, funkcji symulacyjnych oraz interfejsu użytkownika opartego na bibliotece ImGui. Klasy takie jak `VectorSim`, `VectorPozarPierwotny`, `VectorPozarRozprzestrzeniony` oraz `Ubezpieczyciel` są odpowiedzialne za organizację i przechowywanie danych związanych z symulacjami pożarów oraz informacjami o ubezpieczycielach. Program zawiera również funkcje, które wczytują i przetwarzają dane wejściowe z plików CSV, co umożliwia załadowanie istotnych informacji o pożarach i ubezpieczeniach. Zestaw funkcji symulacyjnych przeprowadza obliczenia dotyczące ryzyka pożarowego oraz rozprzestrzenienia pożarów, pozwalając na symulację różnych scenariuszy i analizę wyników. Interfejs użytkownika, stworzony z wykorzystaniem biblioteki ImGui, zapewnia łatwy dostęp do funkcji programu, umożliwiając konfigurację parametrów wejściowych, wczytywanie danych, uruchamianie symulacji oraz zapis wyników. Dodatkowo, program został zaprojektowany jako aplikacja wielowątkowa, co pozwala na równoległe przetwarzanie danych, a zastosowanie odpowiednich mechanizmów zapewnia synchronizację wątków, gwarantując bezpieczeństwo danych i efektywność wykonywanych operacji. Całość projektu tworzy złożony system, który umożliwia analizę ryzyka pożarowego w kontekście ubezpieczeń, wspierając użytkowników w podejmowaniu lepszych decyzji na podstawie symulacji i analiz danych.

1. Środowisko i wymagania

**2.1. Wymagania sprzętowe**

* Współczesny komputer z systemem operacyjnym Windows
* Procesor wielordzeniowy
* Minimum 8 GB RAM
* Dysk SSD zalecany

**2. 2. Wymagania programowe**

* Kompilator C++ (G++)
* Biblioteki: Boost, GLFW, ImGui, BS\_thread\_pool (domyślnie są załączone do projektu)

**2. 3. Instrukcja obsługi**

**2.3.1. Instalacja i uruchomienie programu**

* 1. Skopiuj wszystkie pliki projektu, w tym plik rozwiązania .sln, na swój komputer. Upewnij się, że struktura katalogów jest zachowana.
  2. Upewnij się, że masz zainstalowane Visual Studio 2022 Community. Podczas instalacji wybierz opcję „Desktop development with C++”, aby zainstalować wszystkie niezbędne narzędzia i biblioteki.
  3. Otwórz plik .sln projektu w Visual Studio.
  4. Wybierz odpowiednią konfigurację (Release) oraz platformę docelową (x64) z paska narzędzi.
  5. Kliknij „Build” > „Build Solution” lub użyj skrótu klawiszowego Ctrl + Shift + B, aby skompilować projekt.

1. **Konfiguracja parametrów**
   * Uruchom program i skonfiguruj parametry symulacji za pomocą interfejsu użytkownika.
2. **Uruchomienie symulacji**
   * Po skonfigurowaniu parametrów, kliknij przycisk "Wczytaj listę ubezpieczycieli", następnie „Wczytaj dane” oraz „Włącz symulację”, aby rozpocząć proces symulacji.
3. **Monitorowanie postępu**
   * Program wyświetla pasek postępu, który pokazuje, jak daleko jest w procesie symulacji.
4. **Zapis wyników**
   * Po zakończeniu symulacji, wyniki są zapisywane w wybranym katalogu w formie plików CSV.
5. Struktura projektu

**3.1. Struktura katalogów**

* src/ - Katalog zawierający pliki źródłowe programu.
* include/ - Katalog zawierający pliki nagłówkowe.
* data/ - Katalog zawierający dane wejściowe w formacie CSV.
* output/ - Katalog, w którym zapisywane są wyniki symulacji.

**3.2. Pliki źródłowe i ich funkcje**

* main.cpp - Główny plik programu, zawierający funkcje do obsługi interfejsu użytkownika, funkcje do przetwarzania danych wejściowych i funkcje symulacyjne.

1. Architektura programu

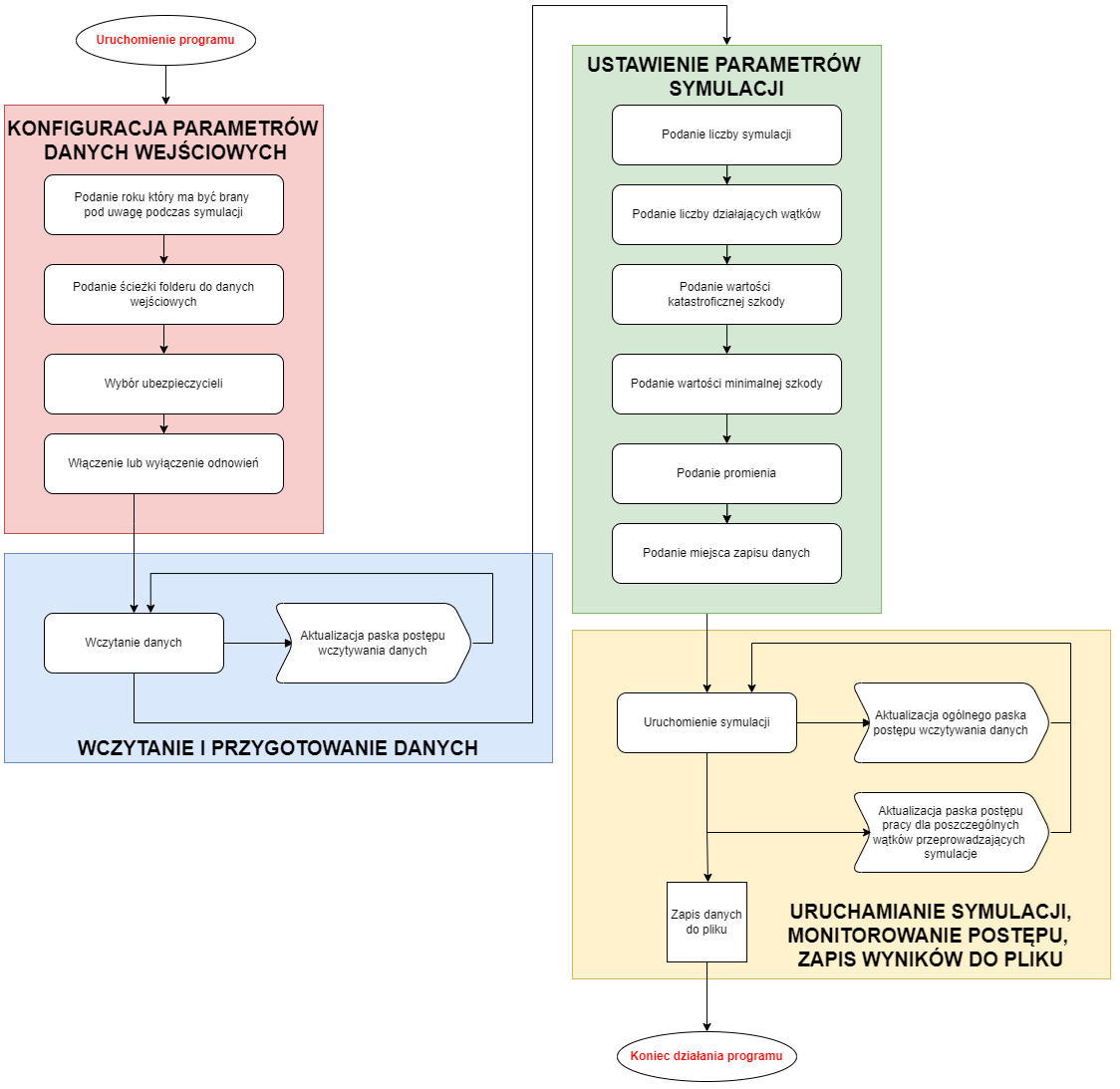
Program jest zbudowany w oparciu o trzy kluczowe komponenty:

* **Zarządzanie danymi** - odpowiedzialne za wczytywanie, przetwarzanie i zapisywanie danych.
* **Symulacja** - wykonuje obliczenia związane z rozprzestrzenianiem się pożarów i ich wpływem na ubezpieczenia.
* **Interfejs użytkownika** - umożliwia użytkownikowi interakcję z programem i konfigurację symulacji.

Program jest zaprojektowany w sposób modularny, co ułatwia jego rozwój i modyfikacje.

1. Graficzne przedstawienie przepływu danych aplikacji

**4.1. Ogólny diagram działania aplikacji pokazujący główne etapy i przepływ pracy w aplikacji.**



**Rycina 1:  Schemat przepływu danych i symulacji**

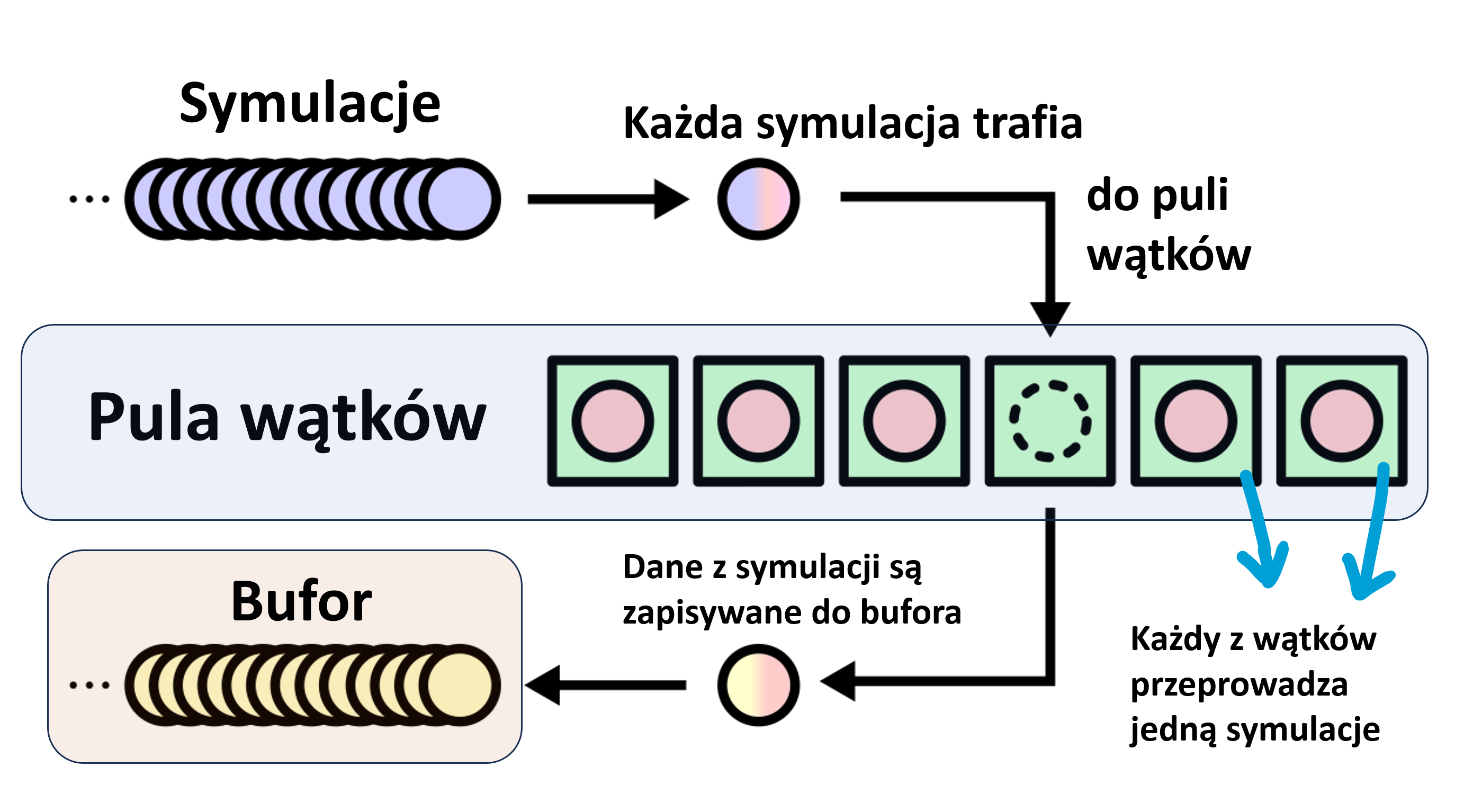
Diagram przedstawia proces korzystania z aplikacji do symulacji. Rozpoczyna się od konfiguracji danych wejściowych, następnie dane są wczytywane i przygotowywane. Kolejnym krokiem jest ustawienie parametrów symulacji i jej uruchomienie z jednoczesnym monitorowaniem postępu. Na końcu wyniki zostają zapisane do pliku, a program kończy działanie.

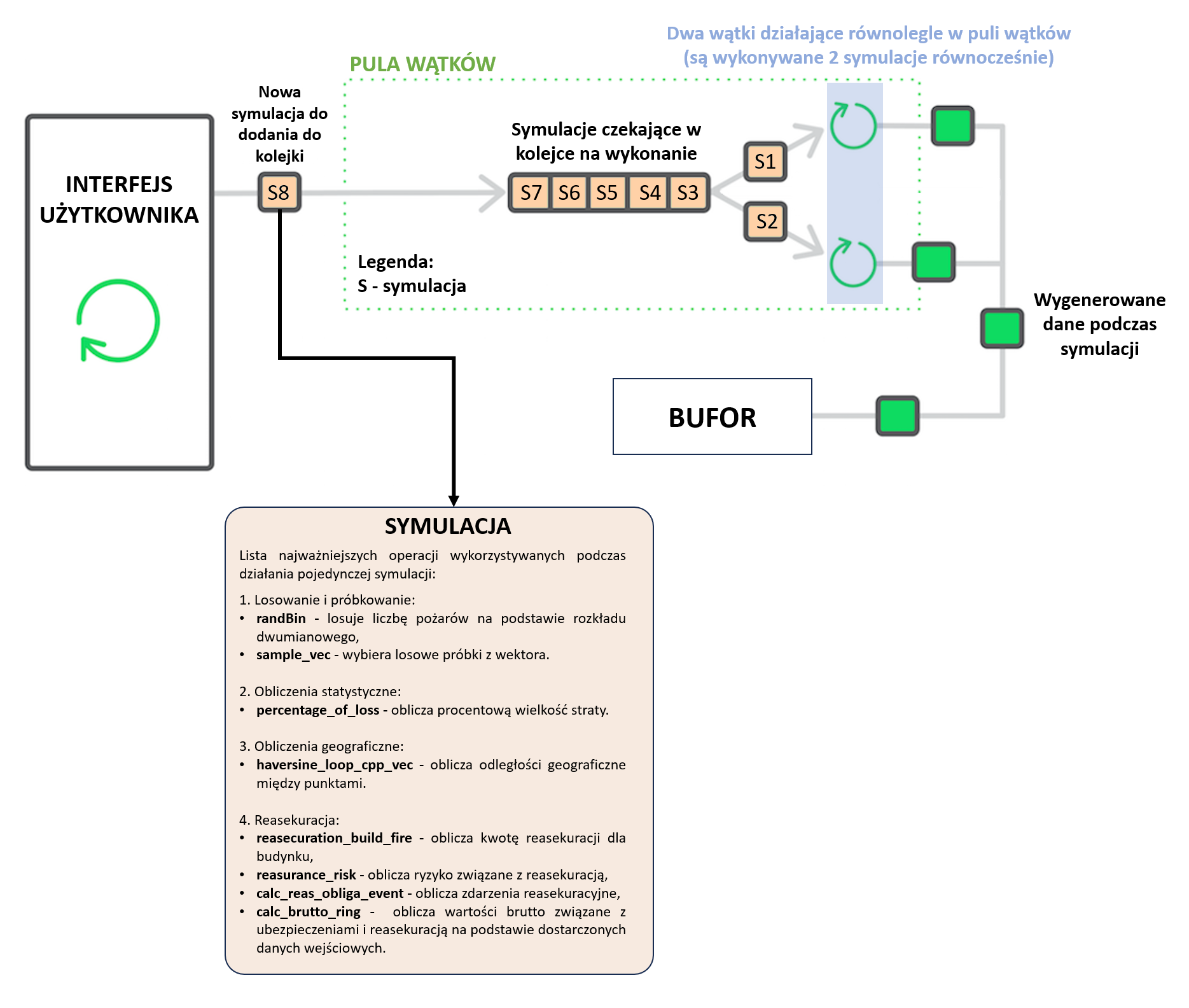
**4.2. Zarządzanie symulacjami przy użyciu puli wątków i bufora.**

Poniższy diagram przedstawia proces zarządzania symulacjami w aplikacji przy użyciu puli wątków i bufora.

1. Na początku widzimy kolejkę **symulacji**, które czekają na przetworzenie.
2. Każda symulacja trafia do **puli wątków**, gdzie każdy wątek jest odpowiedzialny za przeprowadzenie jednej symulacji. Wątki działają równolegle, co pozwala na efektywne przetwarzanie wielu symulacji jednocześnie. Liczba pracujących wątków w puli jest równa liczbie symulacji działających w tym samym czasie.
3. Po zakończeniu symulacji, dane są zapisywane do **bufora**. Bufor przechowuje wyniki, które później są zapisywane.

Diagram ilustruje efektywne zarządzanie zadaniami w systemie wielowątkowym, gdzie użycie puli wątków i bufora optymalizuje przetwarzanie danych.



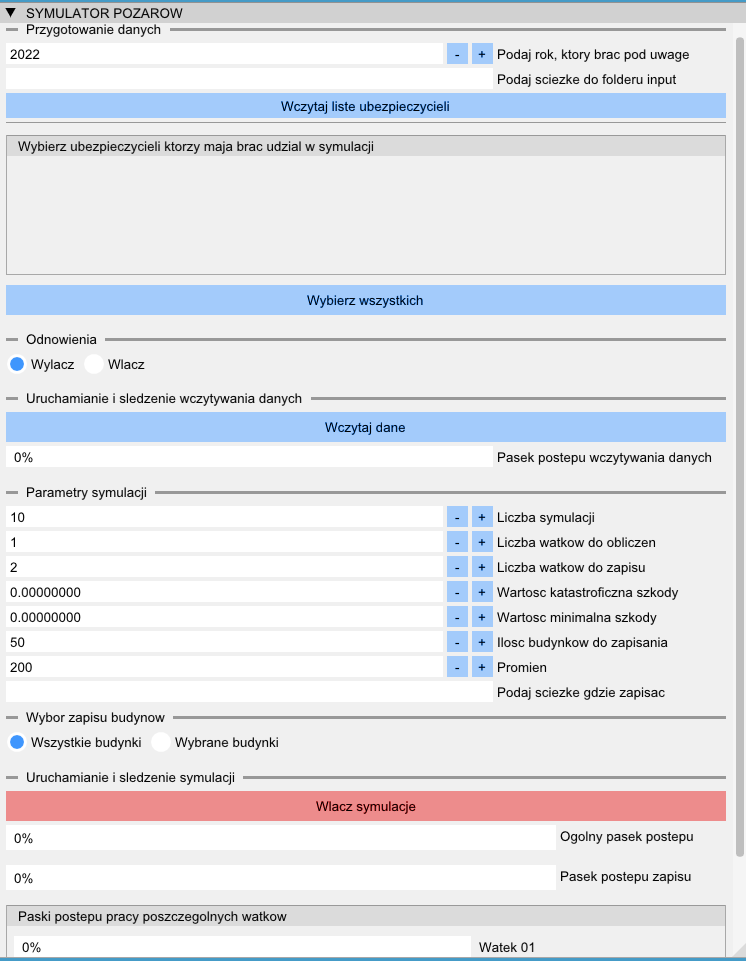


1. Omówienie interfejsu graficznego aplikacji

 Przegląd interfejsu użytkownik:

1. **Sekcja przygotowanie danych**:
   * **Pole tekstowe "Rok"** - umożliwia wprowadzenie roku do analizy.
   * **Pole tekstowe "Ścieżka do folderu input"** - umożliwia podanie ścieżki do folderu z danymi wejściowymi.
   * **Przycisk "Wczytaj listę ubezpieczycieli"** - pozwala załadować listę ubezpieczycieli do symulacji.
   * **Tabela wyboru ubezpieczycieli** - wyświetla listę ubezpieczycieli do wyboru.
   * **Przycisk "Wybierz wszystkich"** - zaznacza wszystkich ubezpieczycieli na liście.
2. **Sekcja odnowienia**:
   * **Przełącznik** - opcja włączenia lub wyłączenia odnowień.
3. **Sekcja uruchamianie i śledzenie wczytywania danych**:
   * **Przycisk "Wczytaj dane"** - inicjuje proces wczytywania danych.
   * **Pasek postępu** - wizualizuje postęp wczytywania danych.
4. **Sekcja parametry symulacji**:
   * **Pola liczby** - pozwalają ustawić parametry, takie jak liczba symulacji, liczba wątków, wartości szkód, itp.
   * **Pole tekstowe "Ścieżka zapisu"** - umożliwia podanie ścieżki do zapisu wyników.
5. **Sekcja wybór zapisu budynków**:
   * **Opcje** - pozwalają wybrać, czy zapisywać wszystkie budynki, czy tylko wybrane.
6. **Sekcja uruchamianie i śledzenie symulacji**:
   * **Przycisk "Włącz symulację"** - rozpoczyna symulację.
   * **Paski postępu** - pokazują ogólny postęp symulacji oraz postęp zapisu.
7. **Sekcja paski postępu pracy poszczególnych wątków**:
   * **Paski postępu** - wyświetlają postęp generowania symulacji dla każdego wątku indywidualnie.

Interfejs jest intuicyjny, zorganizowany w sekcje ułatwiające konfigurację i monitorowanie symulacji.

****

1. Omówienie najwazniejszych funkcji wraz z pseudokodem

**4. Szczegóły implementacji**

**Kluczowe zmienne i ustawienia**

* **ilosc\_budynkow\_do\_zapisania**:
  + Typ: int
  + Opis: Ustala maksymalną liczbę budynków, które mają być zapisane w wynikach symulacji. Domyślnie wynosi 50. Zmienna ta jest konfigurowalna z poziomu interfejsu graficznego.
* **forma\_zapisu\_budynkow**:
  + Typ: int
  + Opis: Określa format zapisu danych budynków. Wartość 0 oznacza zapis wszystkich budynków, a 1 oznacza zapis wybranych budynków. Zmienna ta jest konfigurowalna z poziomu interfejsu graficznego.
* **stanSymulacji**:
  + Typ: std::atomic<double>
  + Opis: Monitoruje postęp symulacji. Umożliwia śledzenie, jak daleko zaszła symulacja w czasie rzeczywistym.
* **stanSymulacjiZapisu**:
  + Typ: std::atomic<double>
  + Opis: Śledzi postęp zapisu wyników symulacji do plików.
* **licznik\_sym**:
  + Typ: std::atomic<int>
  + Opis: Liczy ile pozostało symulacji do wykonania. Umożliwia kontrolowanie, ile symulacji jeszcze się odbywa.
* **wartosc\_minimalna\_szkody**:
  + Typ: double
  + Opis: Ustala minimalną wartość szkody, poniżej której nie są brane pod uwagę wyniki symulacji.
* **save\_step**:
  + Typ: double
  + Opis: Ustala krok zapisu postępu symulacji, co pozwala na aktualizację stanu zapisu.

**Wątki i Zarządzanie Zasobami**

* **BS::thread\_pool pool**:
  + Opis: Pula wątków, która wykonuje zadania przeprowadzenia symulacji, umożliwiając równoległe ich przetwarzanie.
* **BS::thread\_pool poolFiles**:
  + Opis: Pula wątków odpowiedzialnych za zapis wyników do plików podczas trwania symulacji.

**Dane Wejściowe**

* **ubezp\_nazwy**:
  + Typ: std::vector<std::string>
  + Opis: Lista nazw ubezpieczycieli, które są brane pod uwagę w symulacji.
* **flagi**:
  + Typ: std::vector<ImGuiComboFlags>
  + Opis: Flagi, które pozwalają na wybór ubezpieczycieli w interfejsie użytkownika.

**Parametry Symulacji**

* **wybrany\_rok**:
  + Typ: int
  + Opis: Rok, który jest brany pod uwagę w symulacji.
* **sciezka\_input**:
  + Typ: char[512]
  + Opis: Ścieżka do folderu z danymi wejściowymi.
* **liczba\_symulacji**:
  + Typ: int
  + Opis: Liczba symulacji, które mają być przeprowadzone.
* **promien**:
  + Typ: int
  + Opis: Ustala promień, w jakim rozprzestrzeniają się pożary w symulacji.
* **liczba\_dzialajacych\_watkow**:
  + Typ: int
  + Opis: Liczba wątków, które będą używane do obliczeń.
* **liczba\_watkow\_do\_zapisu**:
  + Typ: int
  + Opis: Liczba wątków przeznaczonych do zapisywania wyników symulacji.

**Postęp i Zapis**

* **pasek\_postepu\_wczytywania\_danych**:
  + Typ: float
  + Opis: Wartość postępu wczytywania danych, używana do aktualizacji interfejsu użytkownika.
* **ogolnyprogress**:
  + Typ: float
  + Opis: Ogólny postęp symulacji, który jest aktualizowany w czasie rzeczywistym.
* **gdzie\_zapisac**:
  + Typ: char[512]
  + Opis: Ścieżka, w której mają być zapisywane wyniki symulacji.

**Główne komponenty i klasy**

**VectorSim**

Klasa VectorSim przechowuje dane dotyczące symulacji w formie wektorów i jest używana do przechowywania danych finansowych związanych z ubezpieczeniam. Umożliwia dodawanie danych, zwracanie wektorów oraz czyszczenie danych.

**VectorPozarPierwotny**

Klasa VectorPozarPierwotny przechowuje dane dotyczące pierwotnych pożarów. Umożliwia dodawanie danych, zwracanie wektorów oraz zapisywanie danych do plików CSV.

**VectorPozarRozprzestrzeniony**

Klasa VectorPozarRozprzestrzeniony przechowuje dane dotyczące rozprzestrzenionych pożarów. Umożliwia dodawanie danych, zwracanie wektorów oraz zapisywanie danych do plików CSV.

**Ubezpieczyciel**

Struktura Ubezpieczyciel przechowuje dane dotyczące ubezpieczycieli, w tym wektory pożarów pierwotnych i rozprzestrzenionych oraz sumy wartości.

**Struktury danych**

**Data**

Struktura Data przechowuje dane dotyczące lokalizacji i wartości ubezpieczenia.

**BuforPierwotny**

Struktura BuforPierwotny przechowuje dane dotyczące pierwotnych pożarów oraz informacje o pliku do zapisu.

**BuforRozprz**

Struktura BuforRozprz przechowuje dane dotyczące rozprzestrzenionych pożarów oraz informacje o pliku do zapisu.

**Globalne zmienne i ich rola**

* std::atomic<double> stanSymulacji - Zmienna przechowująca stan symulacji.
* std::atomic<double> stanSymulacjiZapisu - Zmienna przechowująca stan zapisu symulacji.
* std::atomic<int> licznik\_sym - Zmienna przechowująca licznik symulacji.
* static double wartosc\_minimalna\_szkody - Minimalna wartość szkody.
* double save\_step - Krok zapisu.
* BS::thread\_pool pool - Pula wątków do obliczeń.
* BS::thread\_pool poolFiles - Pula wątków do zapisu plików.

**Funkcje pomocnicze**

**randZeroToOne**

Funkcja generująca losową wartość z zakresu [0, 1].

**Definicja**

double randZeroToOne(int a, int b)

**Opis**

Funkcja randZeroToOne generuje losową wartość zmiennoprzecinkową z zakresu [0,1]. Umożliwia to symulację zdarzeń losowych.

**Parametry**

* int a: Minimalna wartość zakresu (zwykle 0).
* int b: Maksymalna wartość zakresu (zwykle 1).

**Zwracana wartość**

* Funkcja zwraca losową wartość typu double z zakresu [0,1]

**Implementacja**

Funkcja wykorzystuje generator liczb losowych, aby wygenerować wartość w zadanym zakresie. Oto przykładowa implementacja:

double randZeroToOne(int a, int b) {

std::random\_device rd; // Ziarno dla generatora

std::mt19937 gen(rd()); // Inicjalizacja generatora

std::uniform\_real\_distribution<double> distribution(a, b); // Rozkład jednostajny

return distribution(gen); // Zwraca losową wartość

}

**Wzór matematyczny**

Wzór matematyczny dla generowania losowej wartości x*x* z rozkładu jednostajnego na przedziale [a,b][*a*,*b*] można zapisać jako:

x=a+(b−a)⋅U*x*=*a*+(*b*−*a*)⋅*U*

gdzie U*U* to losowa wartość z rozkładu jednostajnego w przedziale [0,1][0,1].

Dla funkcji randZeroToOne, przyjmując a=0*a*=0 i b=1*b*=1, wzór upraszcza się do:

x=U*x*=*U*

gdzie U*U* jest generowaną losową wartością z przedziału [0,1][0,1].

**Zastosowanie**

Funkcja randZeroToOne jest używana w różnych kontekstach, takich jak:

* Generowanie losowych zdarzeń w symulacjach.
* Obliczanie prawdopodobieństw w modelach statystycznych.
* Wykonywanie operacji związanych z losowymi próbkami w analizach danych.

Dzięki zastosowaniu rozkładu jednostajnego, funkcja zapewnia, że każda wartość w przedziale [0,1][0,1] jest generowana z równym prawdopodobieństwem, co jest kluczowe w kontekście symulacji i analizy ryzyka.

**sample\_vec**

Funkcja losowo wybierająca elementy z wektora.

**randBin**

Funkcja generująca losową wartość z rozkładu dwumianowego.

**contains**

Funkcja sprawdzająca, czy wektor zawiera dany element.

**search\_closest**

Funkcja wyszukująca najbliższy element w posortowanym wektorze.

**percentage\_of\_loss**

Funkcja obliczająca procentową wartość szkody.

**calc\_reas\_bligator**

Funkcja obliczająca wartość reasekuracji obowiązkowej.

**reasecuration\_build\_fire**

Funkcja obliczająca wartość reasekuracji dla pożaru budynku.

**index\_spread\_build**

Funkcja obliczająca rozprzestrzenianie się pożaru na podstawie danych wejściowych.

std::vector<std::vector<double>> index\_spread\_build(

long double lat\_center,

long double lon\_center,

const std::vector<std::vector<double>>& distance\_res,

const std::vector<std::vector<long double>>& lat\_ring,

const std::vector<std::vector<long double>>& lon\_ring,

const std::vector<std::vector<int>>& insu\_ring,

const std::vector<std::vector<int>>& reas\_ring,

const std::vector<std::vector<double>>& exposure\_sum\_ring);

**haversine\_cpp**

Funkcja obliczająca odległość między dwoma punktami na powierzchni Ziemi za pomocą wzoru Haversine.

double haversine\_cpp(double lat1, double long1, double lat2, double long2, double earth\_radius = 6378137);

**index\_in\_ring**

Funkcja obliczająca indeksy budynków w pierścieniu wokół centrum pożaru.

std::vector<std::vector<double>> index\_in\_ring(

long double lat\_center,

long double lon\_center,

const std::vector<long double>& lat\_sub,

const std::vector<long double>& lon\_sub,

const std::vector<int>& insu\_sub,

const std::vector<int>& reas\_sub,

const std::vector<double>& exponsure\_sum\_value);

**haversine\_loop\_cpp\_vec**

Funkcja obliczająca odległości między budynkami w symulacji.

std::vector<std::vector<double>> haversine\_loop\_cpp\_vec(

double radius,

int n1,

int woj,

int mies);

**calc\_brutto\_ring**

Funkcja obliczająca wartości brutto dla pierścienia budynków.

std::vector<std::vector<std::vector<double>>> calc\_brutto\_ring(

std::vector<double> data\_input,

std::vector<double> insurance,

std::vector<double> reas\_input,

double kat\_val,

int ilosc\_ubezpieczycieli);

**reasurance\_risk**

Funkcja obliczająca ryzyko reasekuracyjne dla budynków.

std::vector<std::vector<double>> reasurance\_risk(

std::vector<std::vector<double>> out\_exp\_sum\_kwota\_insurancers,

std::vector<std::vector<double>> out\_reas\_insurens,

int ilosc\_ubezpieczycieli);

**calc\_reas\_obliga\_event**

Funkcja obliczająca wartości obowiązkowej reasekuracji dla zdarzeń.

std::vector<std::vector<double>> calc\_reas\_obliga\_event(

int ins\_ind,

double fire\_prem,

std::vector<std::vector<double>> num\_reas\_insurances,

std::vector<std::vector<double>> val\_reas\_insurances,

int size\_vec,

int ilosc\_ubezpieczycieli);

**Funkcje główne**

**processPrPozaru**

Funkcja przetwarzająca dane dotyczące prawdopodobieństwa pożaru.

void processPrPozaru(const std::string& filename);

**processPrRozprzestrzenienia**

Funkcja przetwarzająca dane dotyczące rozprzestrzeniania się pożarów.

void processPrRozprzestrzenienia(const std::string& filename);

**processPrWielkoscPozaru**

Funkcja przetwarzająca dane dotyczące wielkości pożaru.

void processPrWielkoscPozaru(const std::string& filename);

**processOblig**

Funkcja przetwarzająca dane dotyczące obowiązkowej reasekuracji.

void processOblig(const std::string& FOLDER\_REAS, const std::vector<std::string>& filename);

**processBudynki**

Funkcja przetwarzająca dane dotyczące budynków.

void processBudynki(const std::string FOLDER\_UBEZP, const std::string ubezp, const std::vector<std::string>& filename, std::string year, std::string odnowienia);

**processReas**

Funkcja przetwarzająca dane dotyczące reasekuracji.

void processReas(const std::string FOLDER\_REAS, const std::vector<std::string>& filename);

**simulateExponsureTEST**

Funkcja przeprowadzająca symulację ekspozycji na pożar.

void simulateExponsureTEST(std::string nazwakatalogu, int sim, int numer\_symulacji, double kat\_val, int ilosc\_ubezpieczycieli, int num\_watku);

**testALL**

Funkcja uruchamiająca cały proces symulacji.

void testALL(int choice);

**5. Wątki i synchronizacja**

**watekZapisPierwotny**

Funkcja wątku zapisującego dane dotyczące pierwotnych pożarów.

void watekZapisPierwotny();

**watekZapisRozprz**

Funkcja wątku zapisującego dane dotyczące rozprzestrzenionych pożarów.

void watekZapisRozprz();

**Synchronizacja za pomocą mutexów i condition variables**

* std::mutex mtx\_pierwotny - Mutex do synchronizacji zapisu pierwotnych pożarów.
* std::mutex mtx\_rozprz - Mutex do synchronizacji zapisu rozprzestrzenionych pożarów.
* std::condition\_variable cv - Condition variable do synchronizacji zapisu pierwotnych pożarów.
* std::condition\_variable cy - Condition variable do synchronizacji zapisu rozprzestrzenionych pożarów.

**6. Interfejs użytkownika**

**Konfiguracja ImGui**

Konfiguracja stylu i czcionek ImGui.

void setup\_imgui\_style(ImGuiIO& io);

**Funkcje renderujące GUI**

Funkcje odpowiedzialne za renderowanie interfejsu użytkownika.

void render\_gui();

**Obsługa zdarzeń**

Funkcje obsługujące zdarzenia w interfejsie użytkownika.

static void glfw\_error\_callback(int error, const char\* description);

**7. Przykłady użycia**

**Uruchomienie symulacji**

1. Skonfiguruj parametry symulacji za pomocą interfejsu użytkownika.
2. Kliknij przycisk "Wlacz symulacje".
3. Monitoruj postęp symulacji na pasku postępu.

**Wczytywanie danych**

1. Skonfiguruj ścieżkę do folderu z danymi wejściowymi.
2. Kliknij przycisk "Wczytaj dane".
3. Monitoruj postęp wczytywania danych na pasku postępu.

**Zapis wyników**

1. Skonfiguruj ścieżkę do folderu, w którym mają być zapisane wyniki.
2. Wyniki symulacji są automatycznie zapisywane po zakończeniu symulacji.