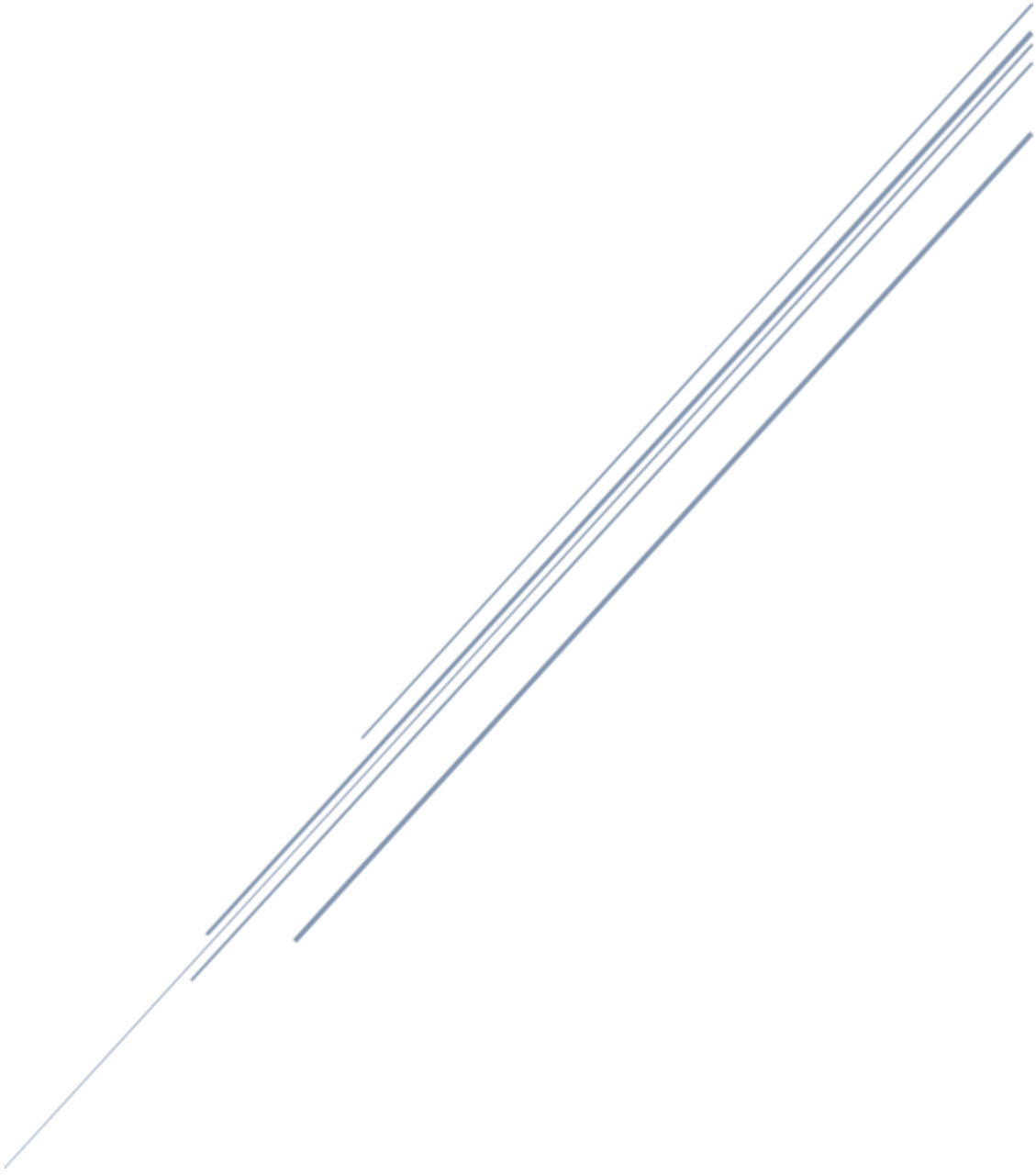
Program przedstawiający symulację apteki



Jakub Kierasiński

Piotr Krzyczkowski

1. Przyjęte założenia

Założyliśmy, że w momencie włączenia symulacji tworzone są wektory składające się z okienek, farmaceutów oraz kolejka klientów. Okienka i farmaceuci tworzą się tylko raz podczas symulacji. Farmaceuci przypisują się po kolei do okienek. Jeden farmaceuta na okienko. Przypisanie farmaceuty do okienka wprowadza je w aktywny status dzięki czemu może przyjmować klientów.

Klienci przychodzący do apteki posiadają wektor symptomów oraz mogą posiadać nazwę leku na który mają receptę. Na podstawie tych danych farmaceuta dobiera odpowiednie leki z bazy.

Żeby doszło do interakcji pomiędzy klientem oraz farmaceutą obydwoje muszą zostać przypisani do okienka. Dopiero w obecności tych dwóch obiektów jest może wykonywania akcji.

Okienko posiada dwa boolowskie atrybuty. Jeden z nich pokazuje nam czy farmaceuta jest w okienku, a drugi czy jest w nim klient. Dodatkowo okienko posiada atrybut statusu operacji klienta, który jest intem. W zależności od tych parametrów program losuje działania wykonywane przez klienta.

Program działa w pętli, której jedna iteracja symbolizuję jedną umowną jednostkę czasu. Dzięki zastosowaniu switcha oraz schematu działań akcji. Poszczególne czynności mogą być wykonywane w różnych jednostkach czasu. Po wykonaniu czynności klient zwalnia dane okienko. Dzięki temu następny klient z początku kolejki ma możliwość wykonać swoją akcje.

2. Struktura programu (podział programu)

Wykorzystane klasy:

Client

Klasa reprezentująca klienta. Klient posiada atrybuty takie jak: imię akcję, lek na receptę, symptomy i koszyk.

Database_meds_reader

Klasa odczytująca dane z pliku txt i na ich podstawie definiuje klasę MDatabase oraz dodaje do niej odpowiednie leki.

Mdatabase

W pełni interaktywna baza danych leków w aptece.

Pharmacist

Klasa reprezentuję farmaceutę, posiadającego identyfikator. Farmaceuta jest odpowiedzialny za działania na lekach oraz wystawianie paragonu, czy faktury. Posiada

metody umożliwiające odpowiednie dobranie leków pod symptomy klient oraz możliwość doboru zamiennika.

Pharmacy

Klasa reprezentuje aptekę. Jej głównym zadaniem jest przeprowadzenie symulacji.

Medicine

Klasa bazowa reprezentująca leki w ogólności, po której dziedziczą podgrupy leków.

Pills, Syrup, Ointment, Drops

Klasy pochodne dziedziczące po klasie Medicine - definiują konkretne grupy leków.

Queue

Klasa reprezentuję kolejkę do apteki. Kolejka składa się z wektora klientów i posiada metody umożliwiające ustawianie liczby osób w kolejce, dodawanie klientów na jej koniec oraz usuwanie klientów z jej początku.

RandomObjectsGenerator

Klasa RandomObjectsGenerator jest odpowiedzialna za generowanie losowych wartości wykorzystywanych w symulacji. Umożliwia generację wektorów składających się z losowej liczby obiektów danej klasy oraz tworzenie stałej liczby obiektów w wektorze potrzebnej przy wywołaniu programu z parametrami użytkownika. Do generacji zmiennych losowych wykorzystuje ziarno, które podczas każdej symulacji jest inne.

TxtFile

Klasa odpowiada za wczytywanie danych potrzebnych do tworzenia losowych obiektów. Przechowuje ścieżki potrzebne do otwierania plików. Z wgranych plików tworzy wektory odpowiednich stringów. Wczytuje takie wartości jak imiona, symptomy, nazwy leków na receptę.

Window

Klasa reprezentuje okienko w aptece. Do okienka mamy możliwość przypisania farmaceuty oraz klienta. Posiada takie atrybuty jak status otwarci, status pokazujący czy okienko jest wolne oraz operację klienta, które są przechowywane jako wartość int.

File_dial_out

Obiekt klasy to operator, któremu w konstruktorze należy podać ścieżkę do pliku - jest on odpowiedzialny za wypisywanie danych symulacji do konsoli (strumienia std::cout) oraz jednocześnie do pliku, którego ścieżkę podaliśmy. Dziedziczy ona po klasie std::streambuf, aby korzystać z metod zdefiniowanych w bibliotece <iomanip> (potrzebne do odpowiedniego formatowania przekazywanego tekstu).

Zdefiniowane wyjątki:

MedicineNotFoundException, NegativeAmountException, NegativePositionException, NegativePriceTagException

Wszystkie zdefiniowane wyjątki dziedziczą po klasie std::invalid_argument.

Program posiada pliki txt:

Med_database_info.txt - plik posiada dane potrzebne do wczytania leków do bazy danych z następującym syntaxem:

Typ_leku Ilość_symptomów(np.3) Nazwa Producent Substancja symptom1 symptom2 symptom3 pojemność_opakowania cena_w_gr recepta typ_unikalny

Pills 3 Aspirin Bayer acetylsalicylic_acid pain fever inflammation 28 7000 0 swallow

Pojemność opakowania wyrażana jest w sztukach lub ml odpowiednio dla każdego typu leku (sztuka dla *Pills*, ml dla reszty typów).

Argumentem recepty może być 1 (lek na receptę) lub 0 (lek bez recepty).

Medicines_prescription.txt - plik składa się tylko z listy nazw leków na receptę

name.txt -plik składa się tylko z listy imion wykorzystywanych przy tworzeniu klientów.

symptoms.txt - plik składa się tylko z listy symptomów wykorzystywanych do tworzenia klientów.

dialog_output.txt - tam zapisuje się cały przebieg programu (po każdym kolejnym uruchomieniu program plik się nadpisuje).

Użyte biblioteki: (wszystkie standardowe)

<iostream> - input/output stream (do wpisywania/wypisywania do konsoli)

<vector> - STL użyty do składowania danych - nie we wszystkich przypadkach lista jest wystarczająco wygodna (łatwiej operować na posortowanych kontenerach)

<windows.h> - używamy w celu zatrzymania działania programu

<fstream> - użyta do obsługi plików (wczytywania z pliku/zapisywania do pliku)

<iomanip> - użyta do łatwiejszego formatowania tekstu

<chrono> - używamy w celu pobrania czasu z procesora, którego używamyh do generowania ziarna

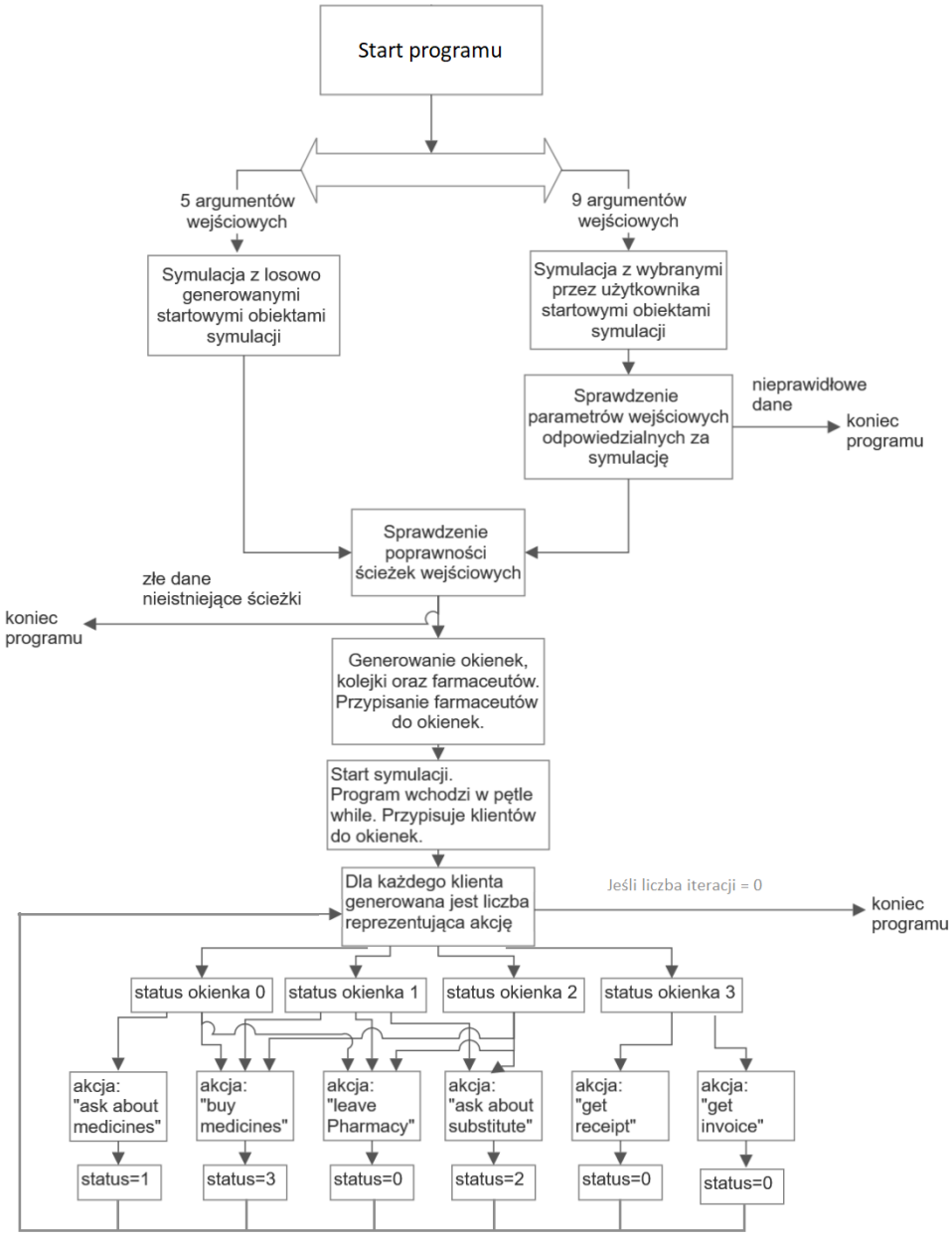
<algorithm> - z tej biblioteki potrzebne były metody do działania na listach/wektorach (np. find_if, if_any itd.)

<memory> - używamy z niej tzw. “smart pointerów” (konkretnie unique_ptr) i operacji na tych pointerach

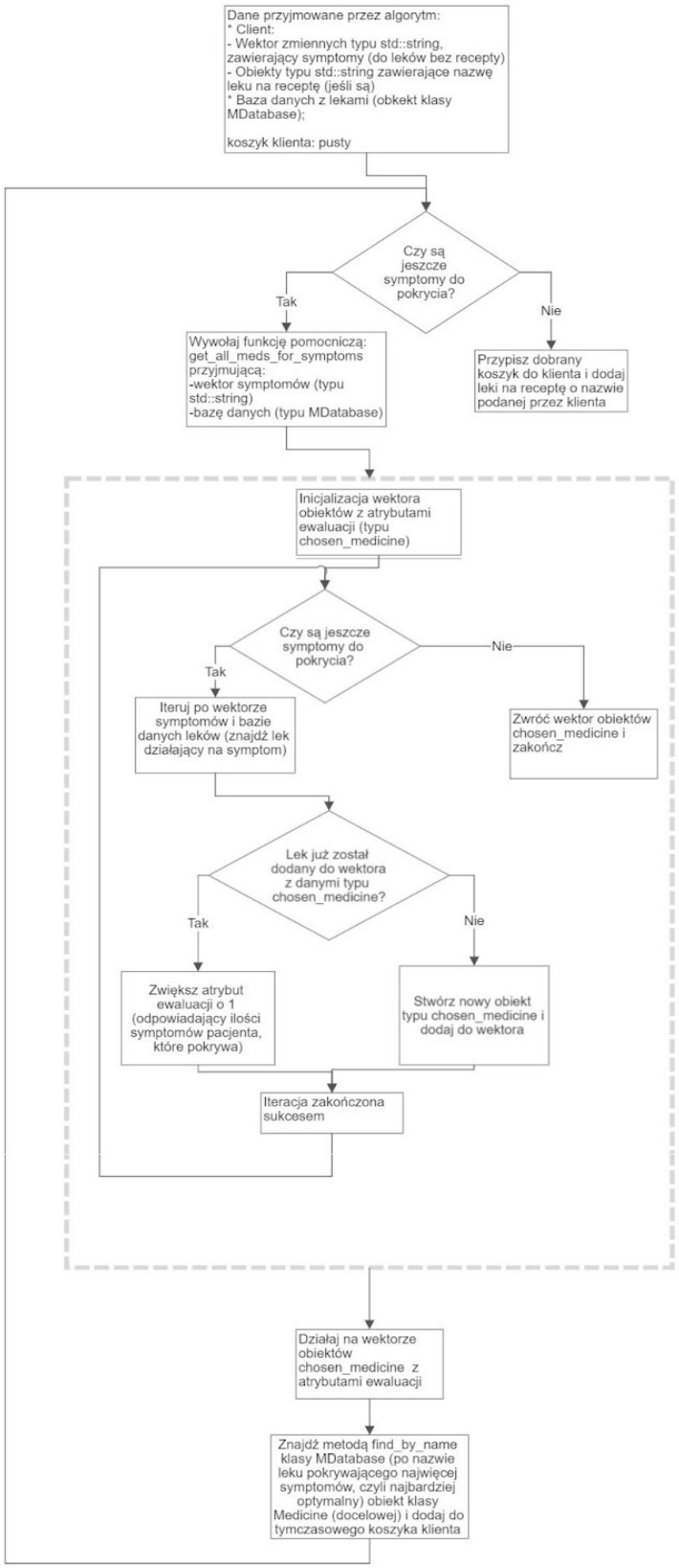
<list> - STL użyty do składowania danych w MDatabase

<random> - używamy do metod oraz obiektów pozwalających generować wartości losowe

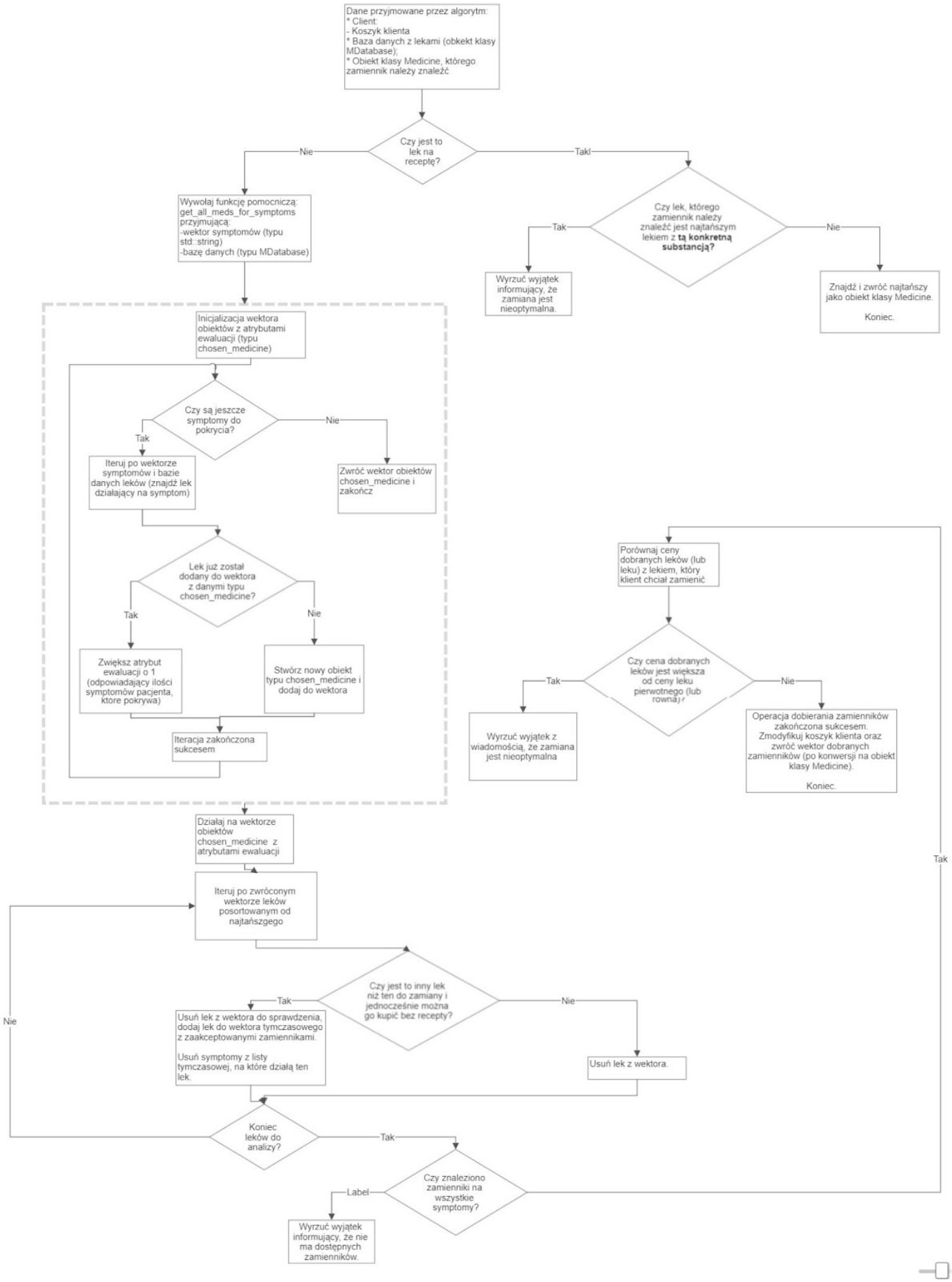
Schemat działania programu:



Uproszczony schemat blokowy działania metody Pharmacist::choose_medicines



Uproszczony schemat blokowy działania metody Pharmacist::choose_cheaper_replacements_and_replace.



3. Instrukcja użytkowania

W celu poprawnego uruchomienia programu użytkownik musi podać 5 lub 9 argumentów wejściowych.

Opis argumentów wejściowych:

pierwszy argument - ścieżka do pliku txt zawierającego bazę danych symptomów,

drugi argument - ścieżka do pliku txt zawierającego bazę danych leków,

trzeci argument - ścieżka do pliku txt zawierającego bazę danych imion,

czwarty argument - ścieżka do pliku txt zawierającego leki na receptę,

piąty argument - ścieżka do pliku txt do którego ma się zapisywać symulacja,

szósty argument - liczba okienek,

siódmy argument - liczba klientów w kolejce na początku programu,

ósmy argument - liczba farmaceutów,

Po podaniu 5 argumentów wejściowych program wylosuję początkową liczbę klientów w kolejcę, liczbę okiene oraz liczbę farmaceutów. Liczba iteracji (czyli naszych jednostek czasu będzie wynosiła 10).

Po podaniu 9 argumentów wejściowych program stworzy startowe parametry zgodnie z naszymi założeniami.

W przypadku podania nieprawidłowych wartości argumentów wywołania takich jak np. nieistniejące ścieżki do plików, czy też wartości które nie reprezentują liczby dla danych wejściowych określających warunki początkowe program poinformuje nas o tym stosownym komunikatem.

Po prawidłowym uruchomieniu programu będziemy mogli zaobserwować wypisywanie się symulacji naszej apteki. Przy każdej iteracji zobaczymy dane takie jak liczba klientów w aptece, liczba farmaceutów oraz liczba okienek. Co trzy sekundy wyświetlają się akcje wykonywane przez klientów.

Program kończy się w momencie skończenie liczby iteracji.

4. Podział pracy

Jakub Kierasiński:

Implementacja klas:

> *Pills, Syrup, Ointment, Drops, Medicine, MDatabase, Pharmacist, File_dial_out, Database_meds_reader*

I metod do nich, w tym wszystkie algorytmy metod w klasie *Pharmacist.*

Część wyjątków.

Część wykonanej pracy - 50%

Piotr Krzyczkowski:

> TxtFile, RandomObjectsGenerator, Client, Window, Pharmacist, Queue, Pharmacy

Metody do wszystkich wyżej wspomnianych klas, plik main oraz część wyjątków.

Część wykonanej pracy - 50%

Nad wieloma klasami/metodami zastanawialiśmy się wspólnie, dużo dyskutowaliśmy na temat różnych implementacji - niemożliwym jest zdefiniowanie podziału obowiązków z dokładnością co do każdej metody. Pełna współpraca to podstawa, a jej owoc Państwu przedstawiamy.

Ostatni commit: d02590b6bbd8cc38fd7544415cb0dcd9cac9e5dd