Symulacja Cyfrowa
Projekt - poprawa

Metoda interakcji proceców

Paweł Koźmiński
nr albumu 126 240
ESPiO

Poznań, 2018

Raport

1.Punkt krwiodastwa – metodą interakcji procesów
Rozszerzenie zadania: A4
Parametry: D2

2.Treść zadania.

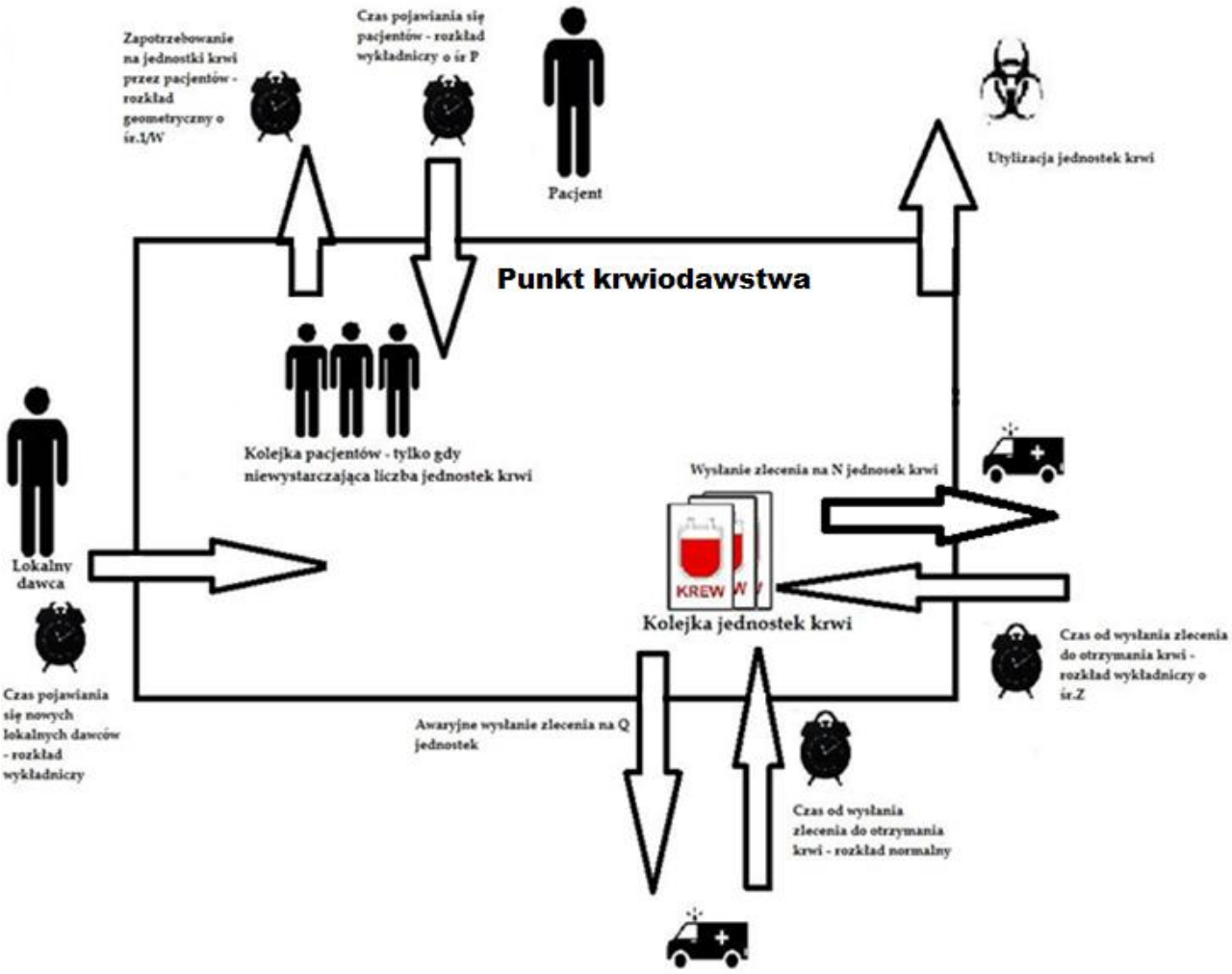
Szpitalny punkt krwiodawstwa korzysta z minitoringu liczby dostępnych jednostek krwi. Jeżeli liczba ta spadnie do poziomu **R** lub niżej, zostaje wysłąne zlecenie na **N** nowych jednostek. Czas od wysłania zamówienia do otrzymania krwi jest zmienną losową o rozkładzie wykładniczym o średniej **Z**. Dostarczona krew musi zostać wykorzystana w ciągu $\mathbf{T_1}$ jednostek czasu. Po tym czasie zostaje zutylizowana. Odstęp czasu pomiędzy pojawieniem się kolejnych pacjentów wymagających transfuzji jest zmienną losową o rozkładzie wykładniczym i średniej **P**. Liczba jednostek krwi podawana pojedynczemu pacjentowi jest zmienną losową o rozkładzie geometrycznym i średniej **1/W**. Jeżeli liczba potrzebnych jednostek jest większa niż aktualny stan zaopatrzenia w punkcie krwiodastwa, zostaje złożone awaryjne zamówienie na **Q** jednostek. Czas dostarczenia takiego zamówienia jest zmienną losową o rozkładzie normalnym, średniej **E** i wariancji $\mathbf{EW^2}$. Dodatkowo, w punkcie krwiodawstwa krew oddają lokalni dawcy. Czas między zgłoszeniem się kolejnych dawców jest zmienną losową o rozkładzie wykładniczym i średniej **L**. Każdy dawca oddaje jedą jednostkę krwi, która musi zostać zużyta w ciągu $\mathbf{T_2}$ jednostek czasu ($\mathbf{T_1}<\mathbf{T_2}$). Celem symulacji jest wyznaczenie wartości **R** oraz **N**, dla których prawdopodobieństwo awaryjnego zamówienia jest mniejsze niż **A**. Dla otrzymanych wartości wyznacz jaki procent krwi jest utylizowany.

Dodatkowo pacjenci oraz dawcy mają jedną z dwóch możliwych grup krwi: A lub B. Pacjenci mogą przyjmować tylko krew swojej grupy. Prawdopodobieństwa występowania grup krwi wynoszą: 60% grupa A oraz 40% grupa B.Prawdopodobieństwa są takie same zarówno dla dawców jak i biorców.Dla każdej grupy krwi występują niezależne zamówienia (t.j. mogą istnieć w jednym momencie cztery zamówienia: dwa awaryjne oraz dwa standardowe, osobne dla każdej grupy krwi).

Rozszerzenie zadania (A4):
Jeśli liczba dostępnych jednostek utrzymuje się przez **TU**=300 jednostek czasu powyżej poziomu **TB**=30, **JB** jednostek zostaje przeznaczonych na badania naukowe.
JB – zmienna losowa o rozkładzie równomiernym w przedziale $\mathbf{[JB_{min}, JB_{max}]}$ **[5,10]**

3.Model symulacji – aby model działał prawidłowo musi zawierać w sobie odpowiednie struktury (klasy) ustalone w należytej hierarchi. Obiekty z ograniczonymi prawami interakcji w strukturę pozostałych obiektów, odpowiednio uporządkowane w programie ,tworzą symulacje odpowiadającą rzeczywistej sytuacji. Model taki pozwala zbadać pewne zjawiska,parametry itp. dużo mniejszym kosztem niż w realnych warunkach.

a)Schemat modelu symulacyjnego



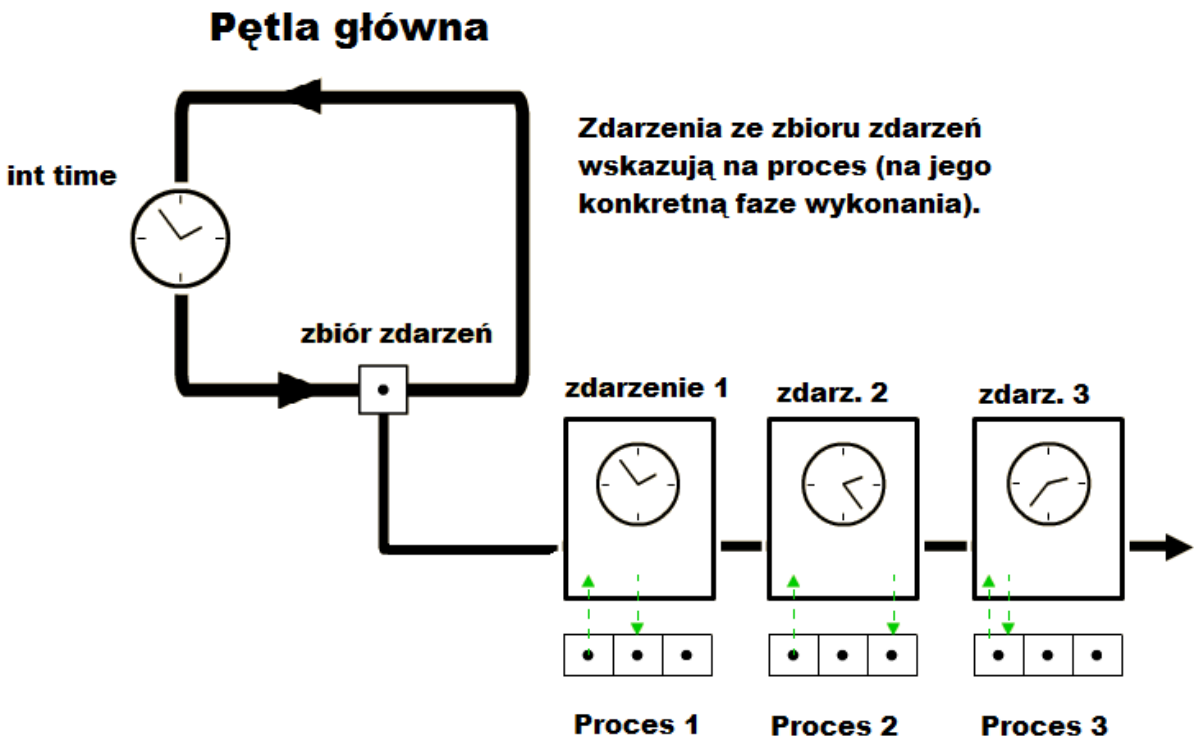
b) Opis obiektów i ich atrybutów

Obiekt	Nazwa klasy implementującej obiekt	Opis	Atrybuty
Punkt krwiodawstwa	BloodDonationPoint	`Klasa zawierająca wszystkie inne elementy sytemu.	- int amountBlood – liczba dostępnych jednostek krwi w punkcie krwiodawstwa - int minAmountBlood – minimalna liczba jednostek krwi zanim zostanie złożone zamówienie na krew - bool order – flaga zamówienia - bool specialOrder – flaga specjalnego zamówienia
Jednostka krwi	UnitBlood	Klasa reprezentująca jednostkę krwi.	- int timeToLive – zmienna przedstawia

			czas usunięcia jednostki krwi - bool group – grupa krwi
Biorca	Recipient	Klasa reprezyntująca pacjentów potrzebujących transfuzji krwi. Pacjenci przybywają do punktu krwodastwa w odstępach czasu o rozkłądzie geometrycznym o śr.1/W	-int unitBloodNeeded zmienna mówi ile jednostek krwi potrzebuje pacjent
Kolejka pacjentów	QueueRecipient	Klasa reprezyntująca kolejke oczekujących pacjentów. W kolejce ustawiają się pacjenci tylko w sytłacji ,gdy zapotrzebowanie na krew przekracza liczbe posiadanych przez punkt jednostek krwi. Kolejka FIFO	lista recipientQueue - kolejka pacjentów
Kolejka jednostek krwi	QueueUnitBlood	Klasa reprezyntująca kolejke jednostek krwi. Kolejka porządkuje jednostki krwi w kolejności takiej ,że na początku są jednostki krwi ,które mają datę utylizacji najbardzej zbliżoną do czasu systemowego.	lista bloodQueue (sortowana malejąco względem zmiennej int timeToLive)
Generator	Generators	Klasa generuje potrzebne rozkłady liczb pseudolosowych.	stałe zmienne typu int potrzebne do rozkładu równomiernego
Statystyki	Statistics	Klasa zbierająca,przetwarzająca oraz drukująca statystyki na konsoli. Także wysyła dane do zewnętrznych plików.	

4.Metoda interakcji procesów – metoda skupia się na procesie, jako sekwencji zdarzeń. Proces zwiazany jest z obiektami modelu i opisuje cykl ich funkcjonowania.

a)Schemat blokowy pętli głównej



b) zidentyfikowane procesy

Proces	Opis	Algorytm
OrderProcess	Proces zajmujący się zamawianiem nowych jednostek krwi, umiejscowieniu ich w kolejce krwi oraz planowaniem ich utylizacji.	1.Gdy liczba dostępnych jednostek krwi spadnie poniżej poziomu R: a) zaplanowanie czasu przyjścia zamówienia 2.Przyjście zamówienia: a) stworzenie N jednostek krwi b) umiejscowienie ich w kolejce krwi c) zaplanowanie czasu usunięcia jednostek krwi z systemu (planowanie utylizacji)
OrderSpecialProcess	Analogiczny proces do powyższego „Order”. Warunk wykonania specjalnego zamówienia spełniony jest ,gdy liczba aktualnego zapotrzebowania jednostek krwi (przez biorców przebywających w danej chwili w systemie) przekroczy liczbę jednostek krwi w punkcie krwiodastwa.	1.Gdy liczba dostępnej krwi jest mniejsza niż liczba potrzebnej krwi: a) zaplanowanie czasu przyjścia specjalnego zamówienia 2.Przyjście specjalnego zamówienia: a) stworzenie N jednostek krwi b) umiejscowienie ich w kolejce krwi c) zaplanowanie czasu usunięcia

		jednostek krwi z systemu (planowanie utylizacji)
RecipientProcess	Proces zajmuję się obsługą biorcy krwi.	1.a) Stworzenie obiektu biorcy w systemie b) umiejscowienie go w kolejce biorców. 2.Jeśli dostępna jest wystarczająca liczba jednostek krwi to pacjent zostaje obsłużony: a)usunięcie biorcy z kolejki biorców b)usunięcie krwi z kolejki krwi oraz usunięcie krwi z systemu c)usunięcie pacjenta z systemu d)zaplanowanie kolejnego biorcy
DonorProcess	Proces zajmujący się obsługą dawcy krwi.	1. a) dodanie 1 jednostki krwi do kolejki krwi b)zaplanowanie utylizacji krwi c) zaplanowanie kolejnej obsługi dawcy krwi.
ScientificResearchProcess	Proces zajmuje się wysłaniem JB jednotek krwi na badania naukowe jeśli przez 300 jednotek czsu poziom krwi w punkcie krwiodawstwa utrzymywał się powyżej 30 jednostek krwi.	1.Sprawdzanie czy liczba jednostek krwi jest powyżej 30.Jeśli tak to punkt 2. 2. Sprawdzanie jednocześnie patrz. punkt 1. oraz czy mineło 300 jednostek czasu. Jeśli tak to punkt 3 3. a)usunięcie JB jednostek krwi z kolejki krwi b) usunięcie JB jednostek krwi z systemu
UtilizationProcess	Proces zajmuje się usuwaniem jednostek krwi z systemu (których czas przydatności minął).	1. a) usuwanie jednostek krwi z kolejki jednostek krwi b) usuwanie jednostek krwi z systemu

5.Generatory liczb pseudolosowych.

W programie symulacyjnym wykorzystałem następujące generatory liczb pseudolosowych:

-generator o rozkładzie równomiernym (multiplikatywny)

-generator o rozkładzie wykładniczym (metoda odwrotnej dystrybuanty)

-generator o rozkładzie normalnym (metoda addytywna)

-generator o rozkładzie geometrycznym (z rozkładu Bernoulliego)

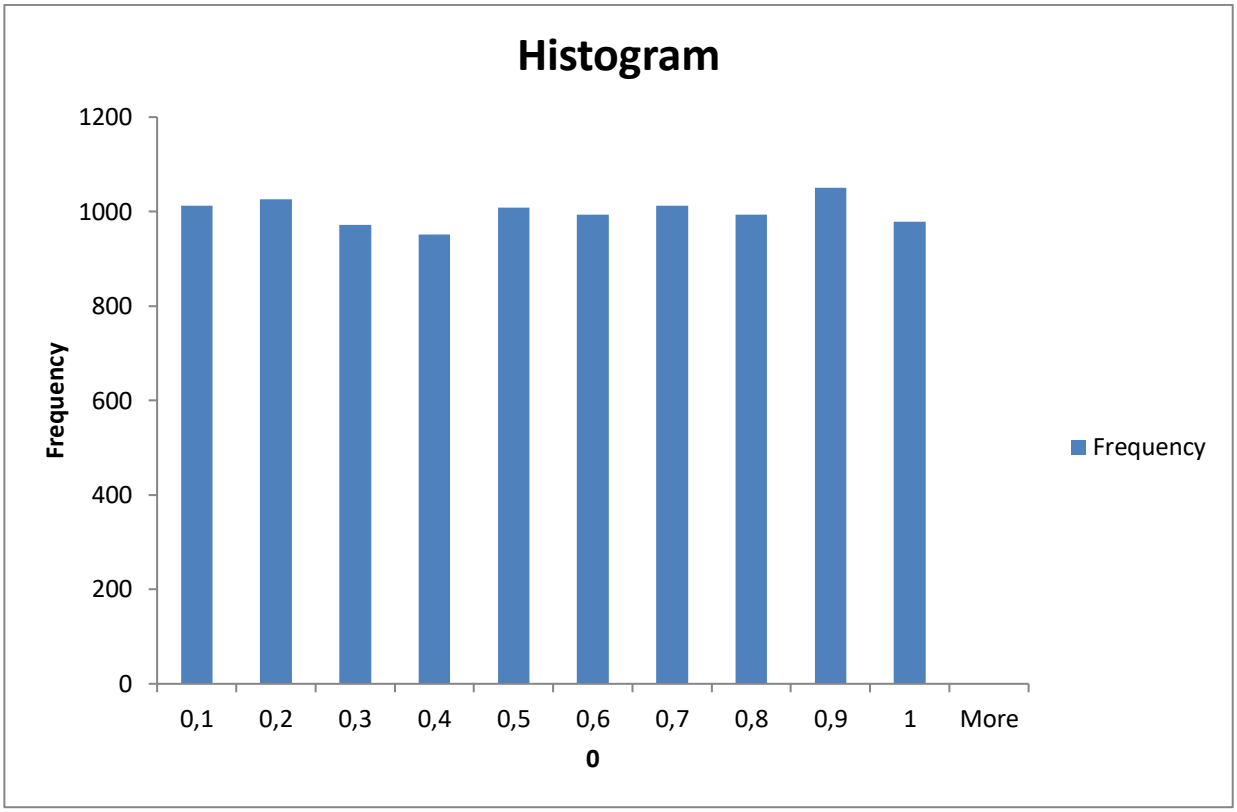
Funkcje generatorów zostały zaczerpnięte z wykładów Symulacji Cyfrowej oraz ze strony internetowej https://www.invocom.et.put.poznan.pl/~invocom/C/P1-2/pl/p1-2/index.htm

Testowanie generatorów odbyło się dla 10 000 wartości. Zapisane zostały one w arkuszu kalkulacyjnym ,a następnie zostały wykreślone histogramy częstotliwości.

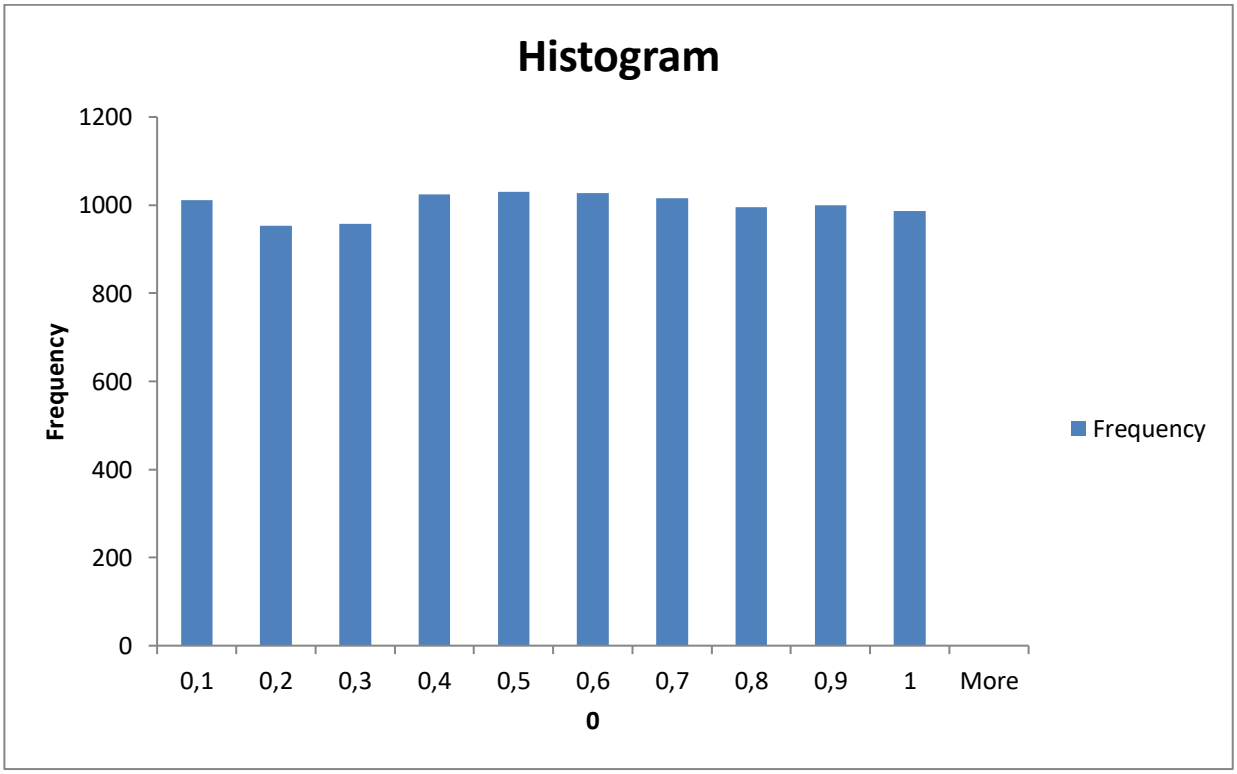
5.1 Generator Równomierny

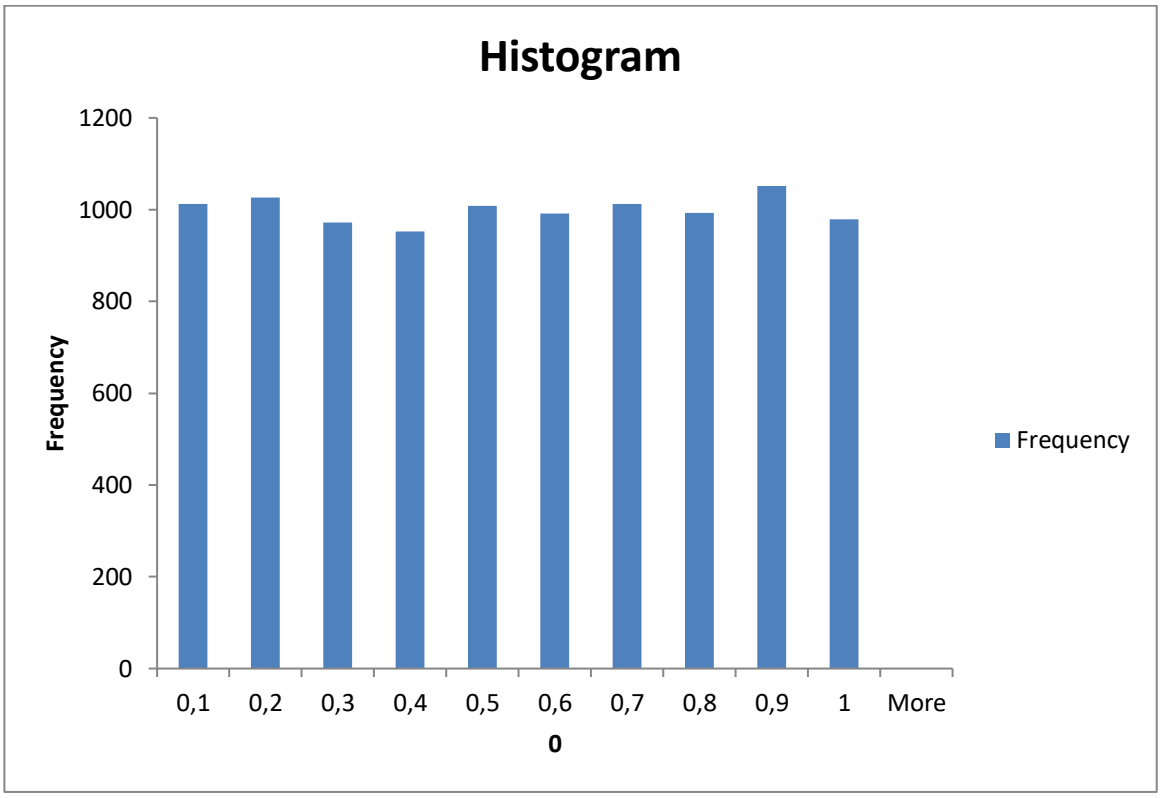
a) Rozkład Równomierny w przedziale [0;1] dla 3 różnych ziaren początkowych

wartość średnia = 0.500364

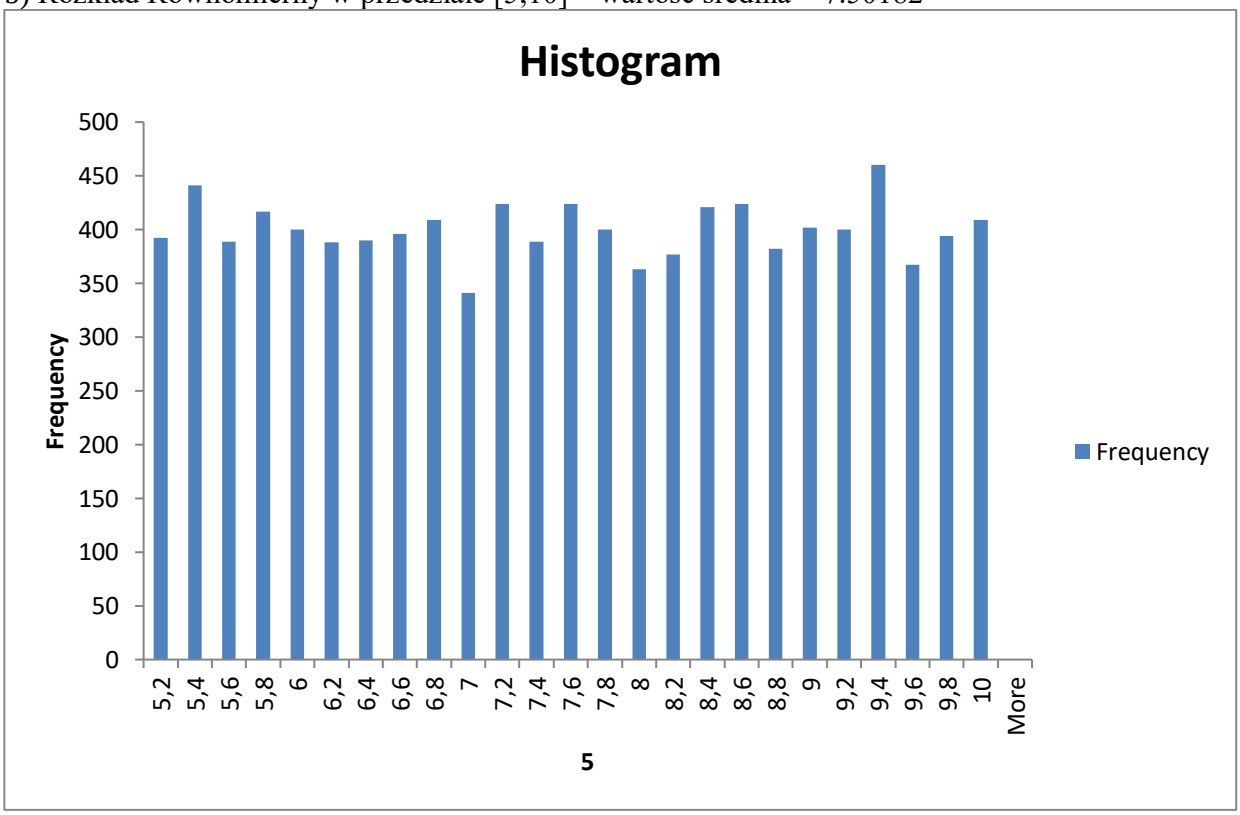


wartość średnia = 0.500415



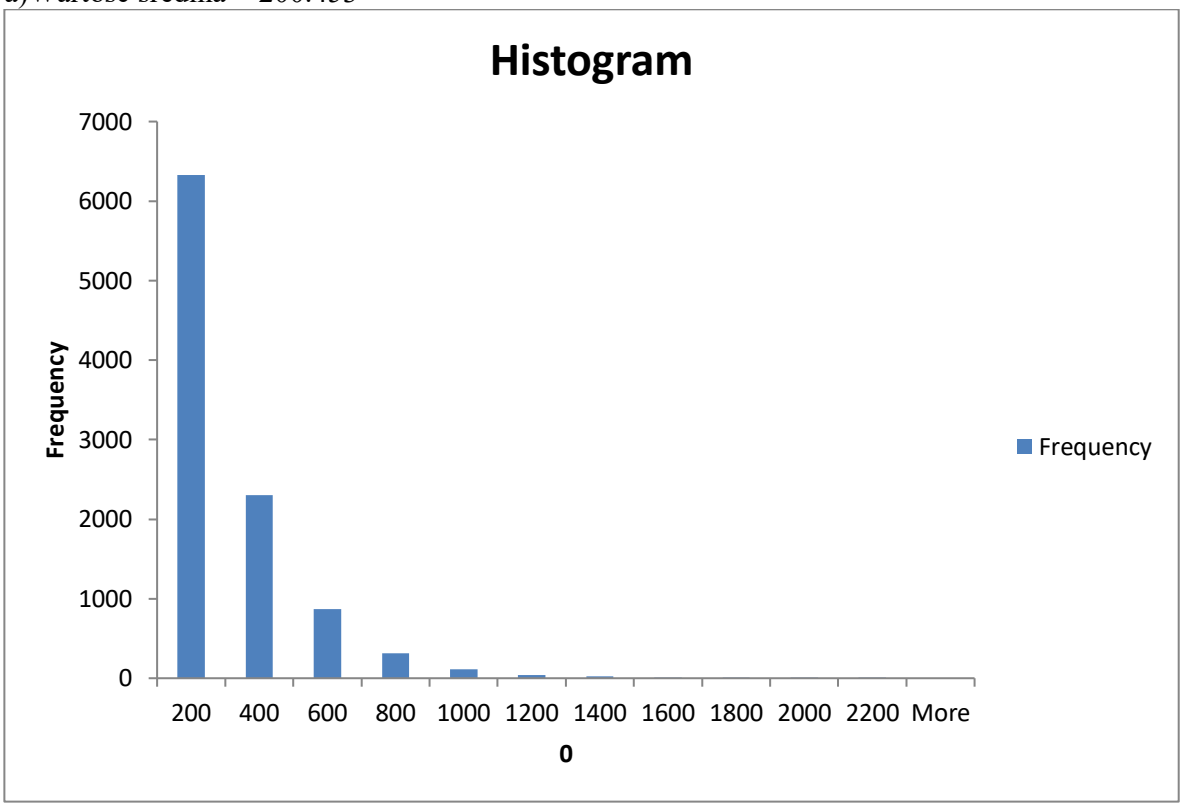


b) Rozkład Równomierny w przedziale [5;10] – wartość średnia = 7.50182

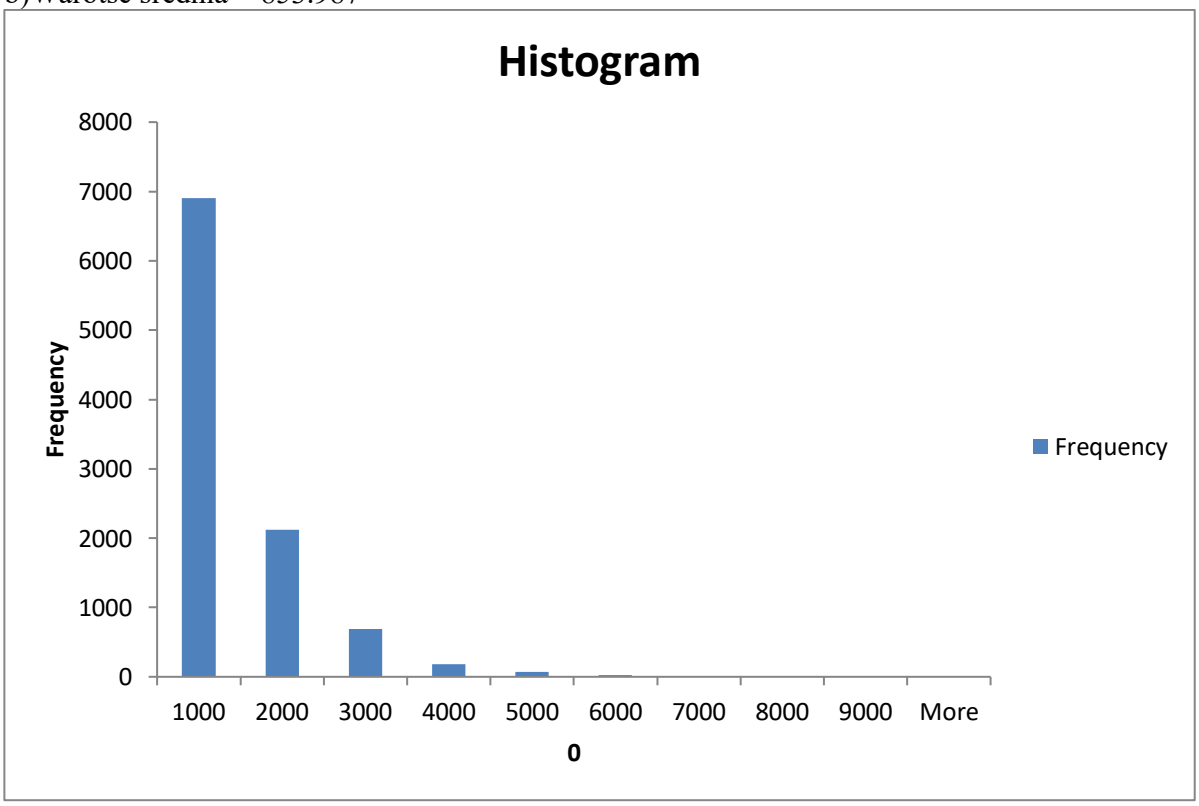


5.2 Generator Wykładniczy

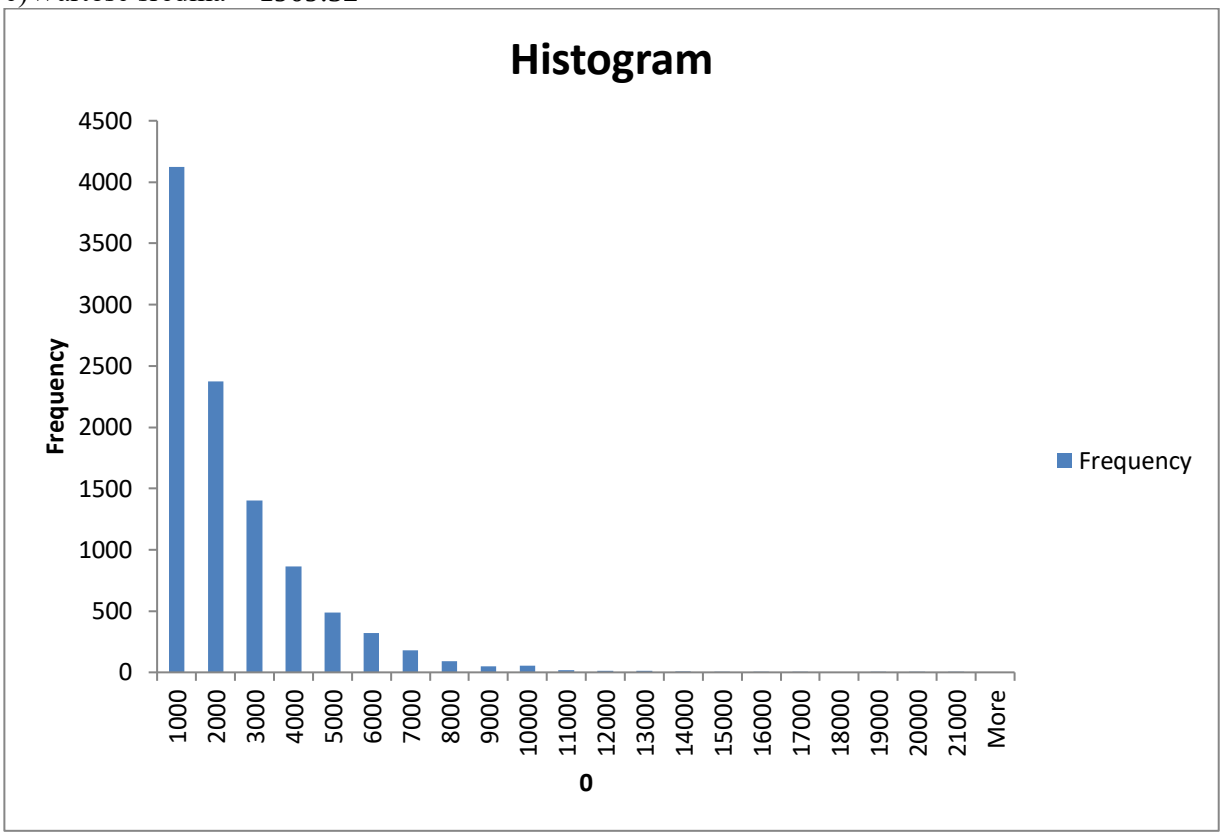
a)Wartość średnia = 200.453



b)Warotść średnia = 853.987

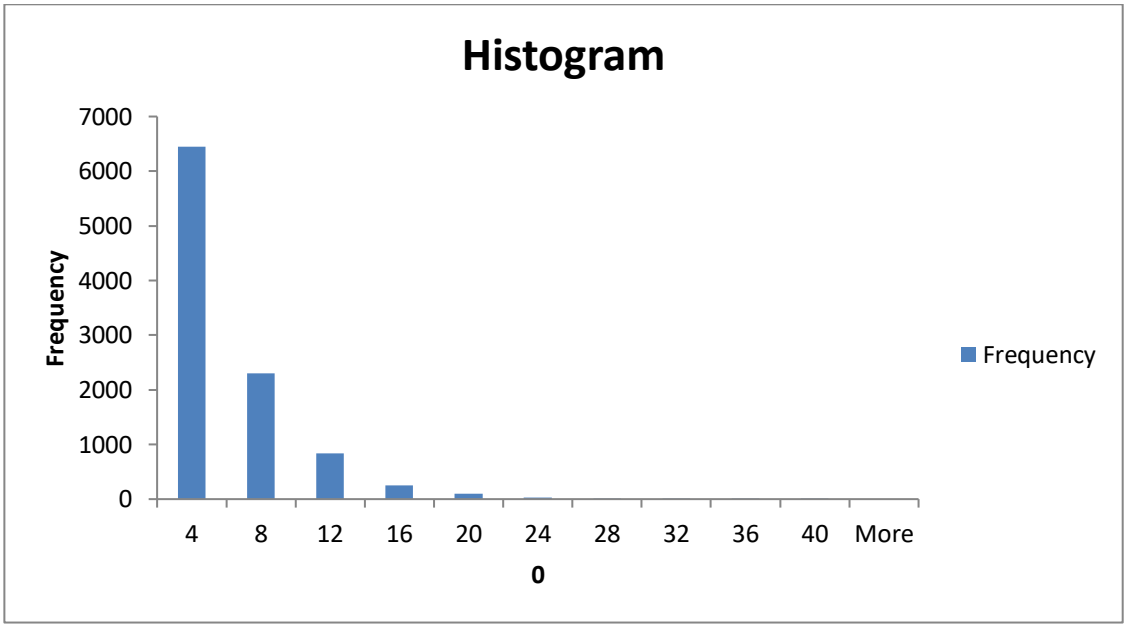


c)Wartość średnia = 1909.52



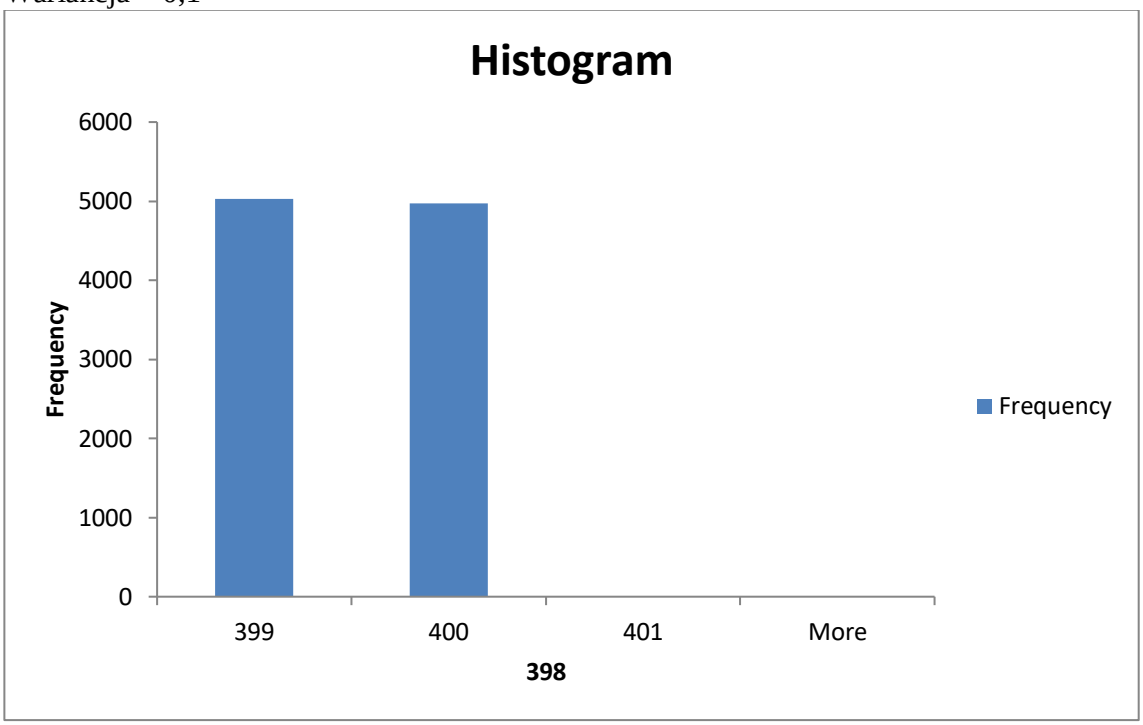
5.3 Generator Geometryczny

Wartość średnia = 4.3578



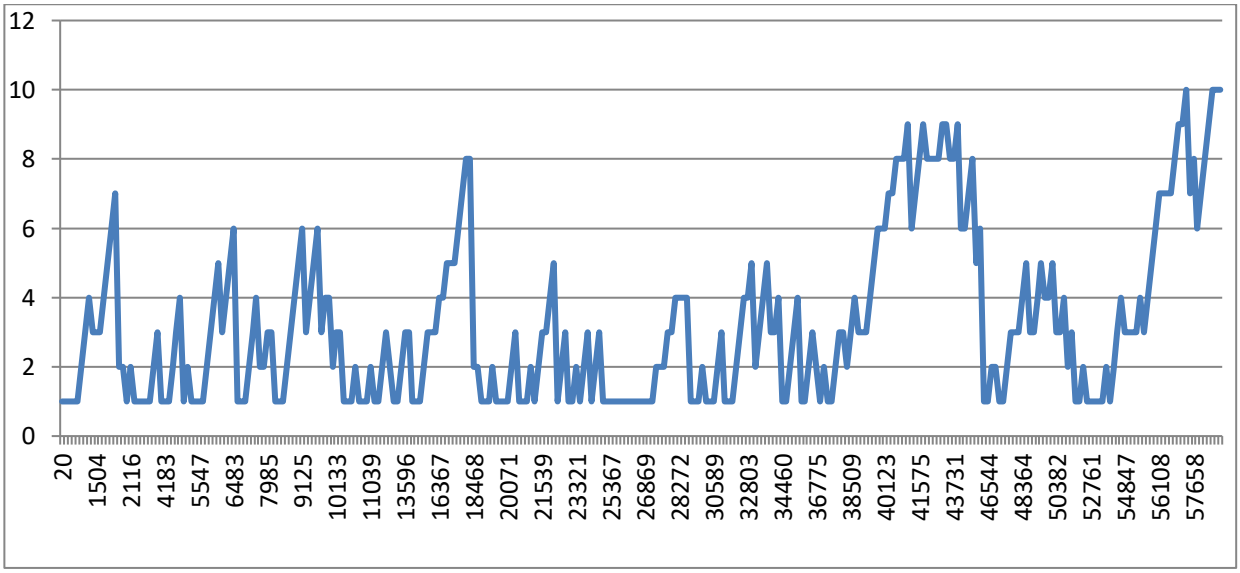
5.4 Generator Normalny

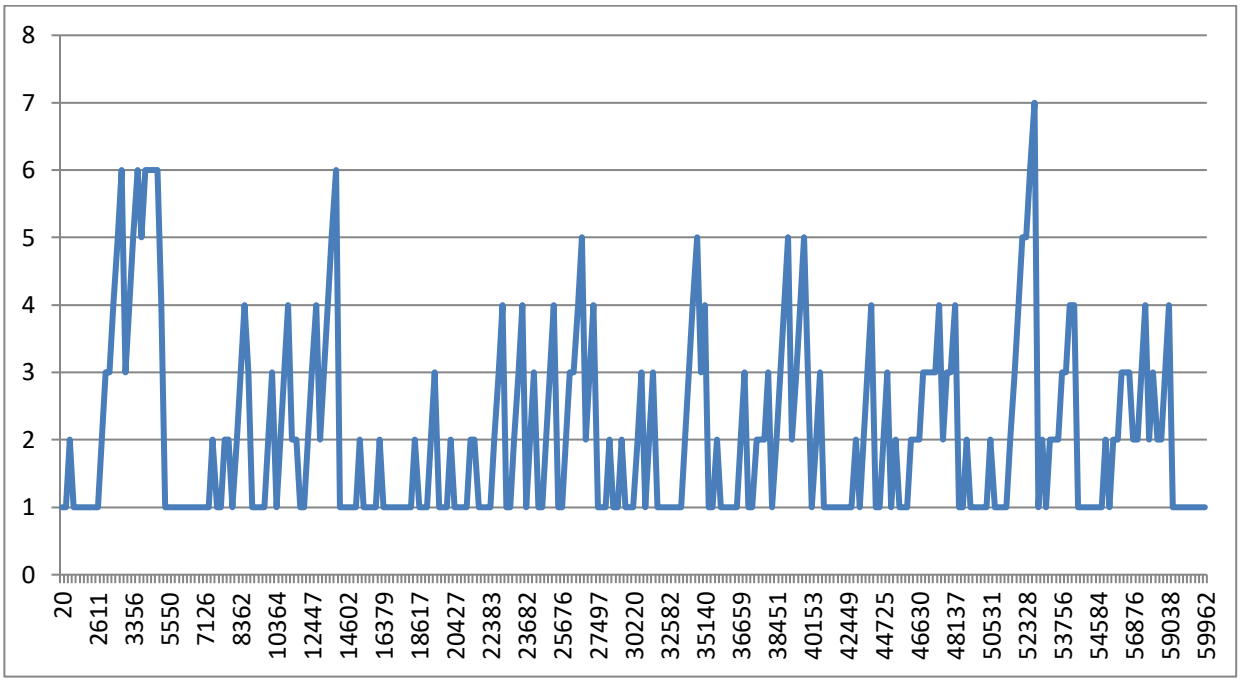
Warotść średnia = 399,67
Wariancja = 0,1

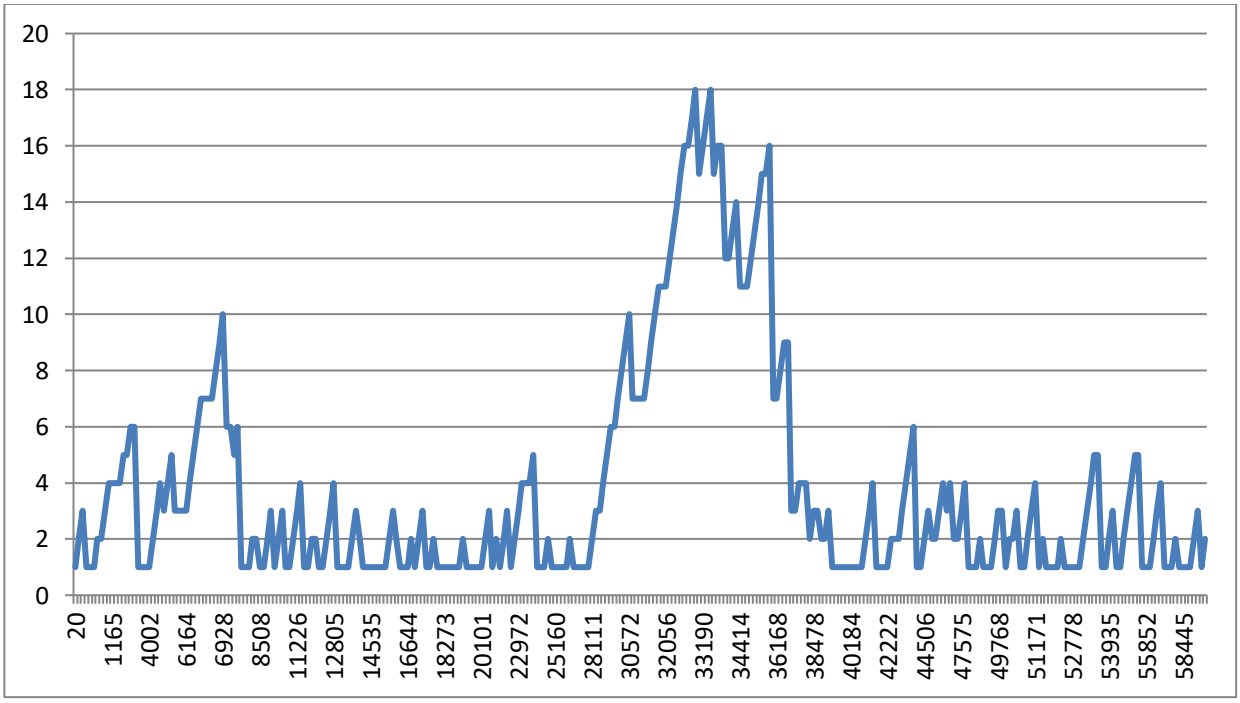


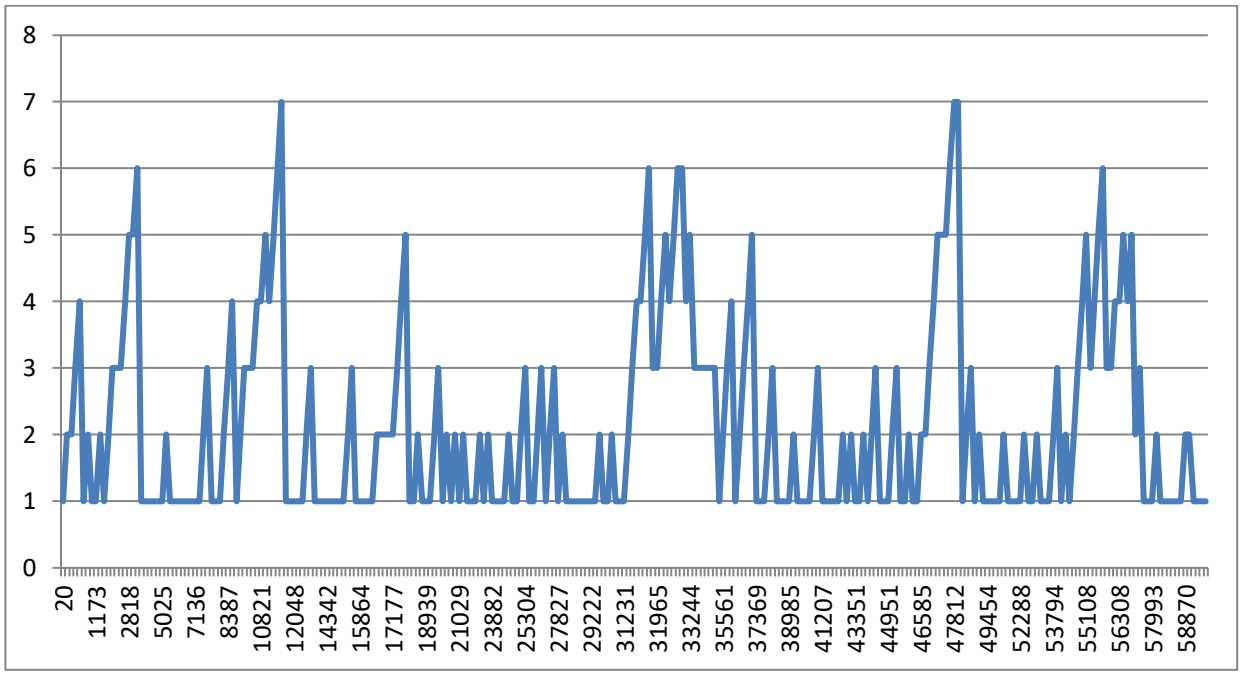
6.Faza początkowa

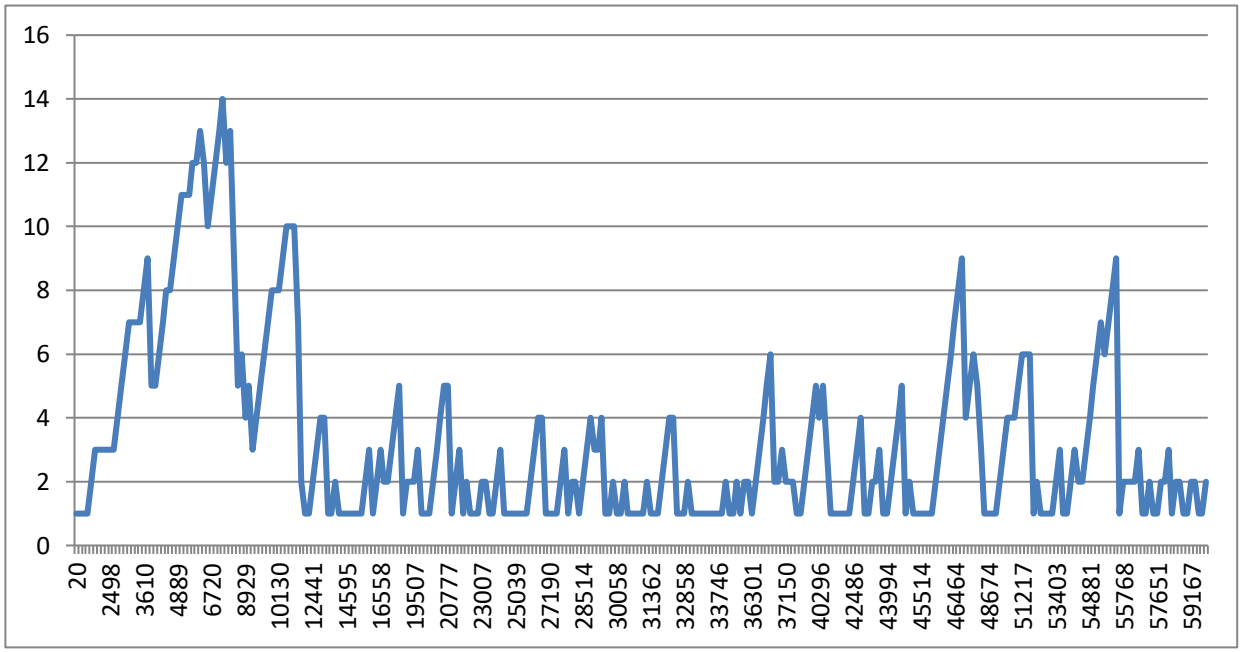
6.1 Wykresy liczby pacjentów w kolejce dla 5 różnych symulacji (różnych ziaren początkowych generatorów) i dla czasu sytemowego 60 000.




8
7
6
5
4
3
2
1
0
20
2611
3356
5550
7126
8362
10364
12447
14602
16379
18617
20427
22383
23682
25676
27497
30220
32582
35140
36659
38451
40153
42449
44725
46630
48137
50531
52328
53756
54584
56876
59038
59962

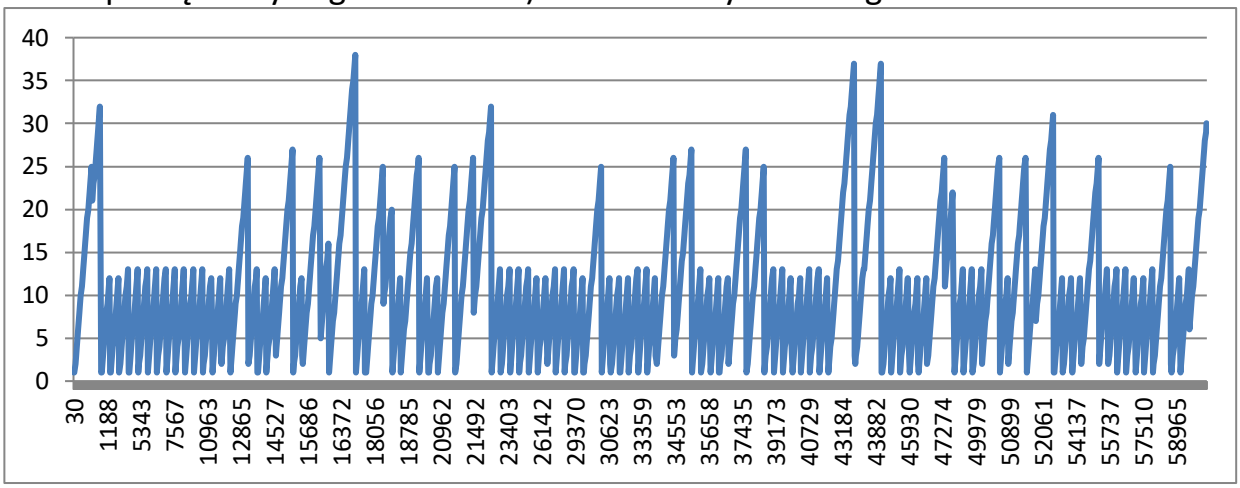

20
18
16
14
12
10
8
6
4
2
0
20
1165
4002
6164
6928
8508
11226
12805
14535
16644
18273
20101
22972
25160
28111
30572
32056
33190
34414
36168
38478
40184
42222
44506
47575
49768
51171
52778
53935
55852
58445

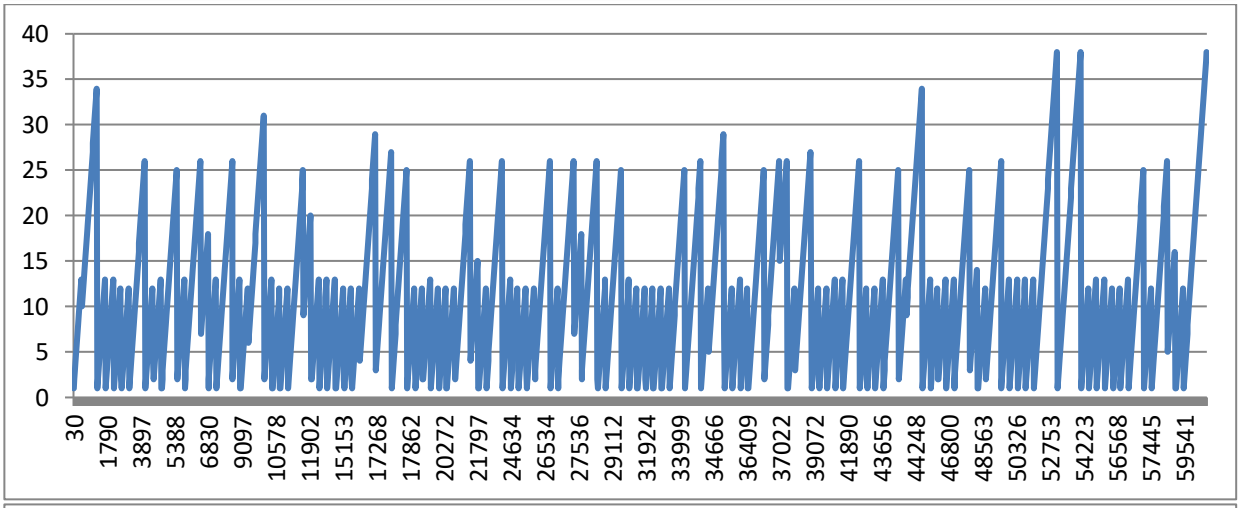


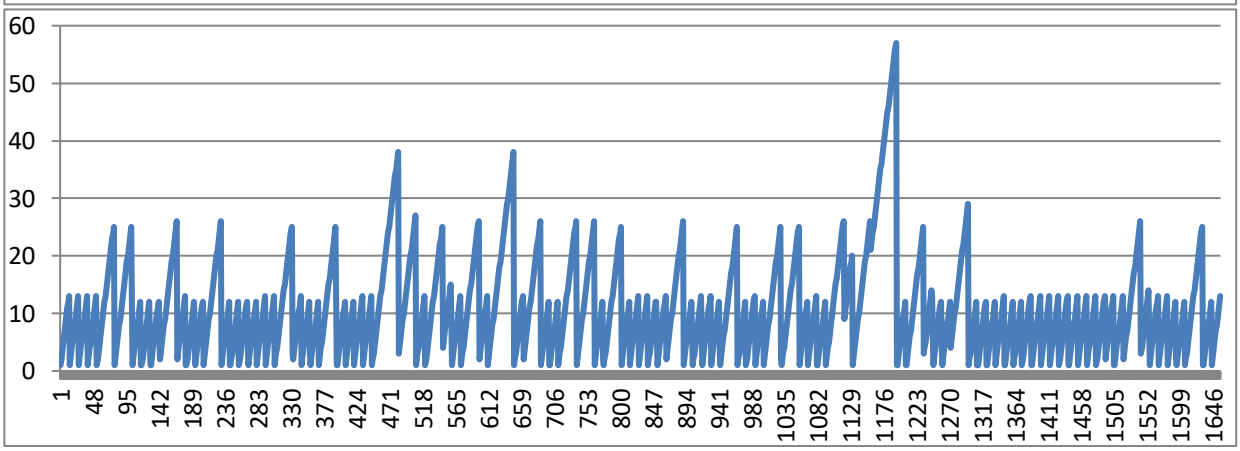


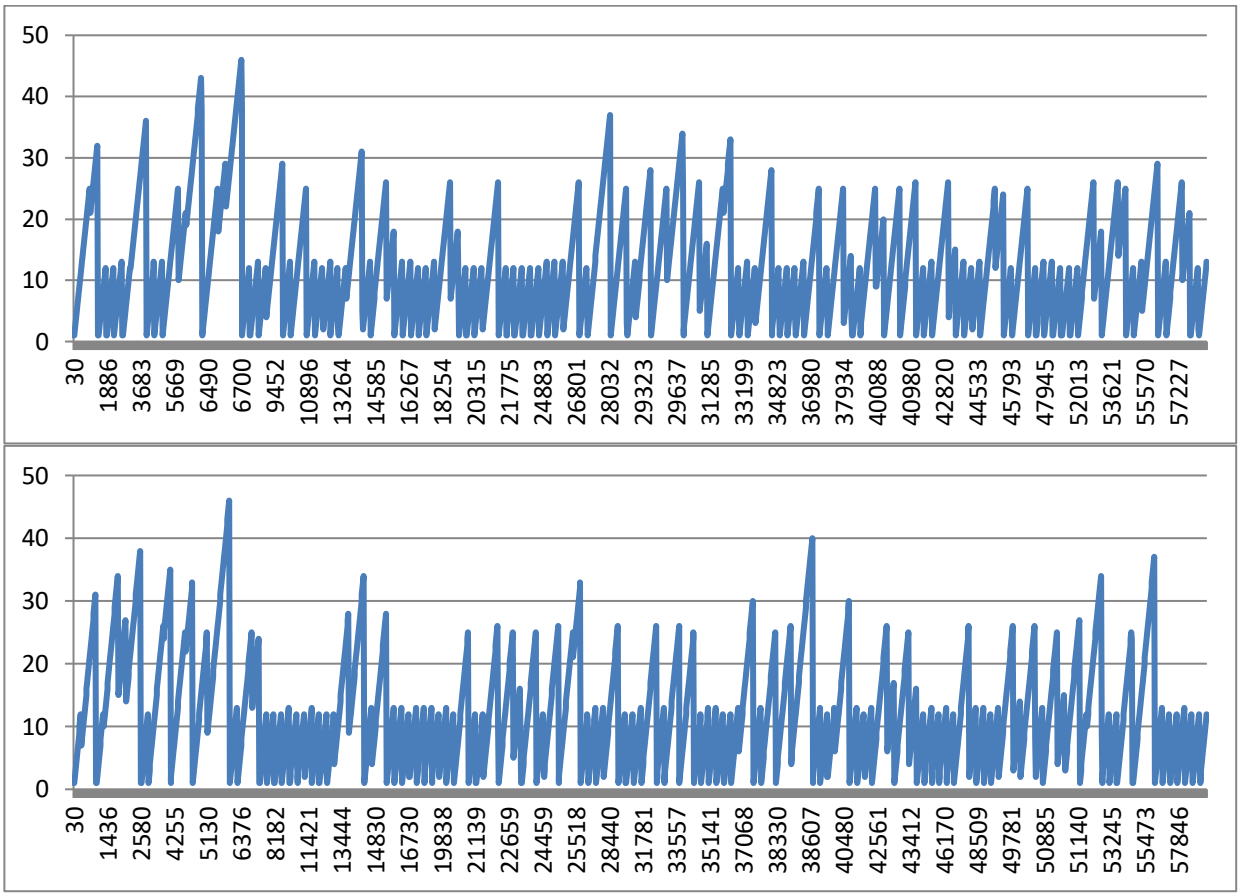
Z powyższych wykresów kolejki pacjentów ciężko wywnioskować dokładny czas zakończenia fazy początkowej. Jest to spowodowane tym ,że wykresy nie są okresowe, kumulacje pacjentów dla różnych ziaren symulacyjnych odbywa się dla innych czasów systemowych np.8000, 35000,52000, 57000.

6.2 Wykresy liczby jednostek krwi w kolejce dla 5 różnych symulacji (różnych ziaren początkowych generatorów) i dla czasu sytemowego 60 000.









Z powyższych wykresów liczby krwi w kolejce krwi także ciężko wyznaczyć koniec fazy początkowej, lecz dla niektórych symulacji można zauważyć pewne ustabilizowanie nie danego parametru. Wnioskując dalej przyjąłem koniec fazy początkowej dla 10 000 czasu systemowego.

7.Wyznaczanie parametrów

Legenda:
% Aw.Zam. - Procent Awaryjnych Zamówień
Odch. Stand. - Odchylenie Standardowe
Dadatni uf. - Dodatnia granica przedziału ufności
Ujemny uf. - Ujemna granica przedziału ufności

N	R	Q	T1	T2	% Aw. Zam.	Odch.Stand.	Dodatni uf.	Ujemny uf.	Przedz.uf
25	10	12	300	500	9,070	0,209	9,360	8,780	0,579
25	15	12	300	500	9,314	0,155	9,529	9,099	0,430
25	20	12	300	500	9,277	0,348	9,760	8,793	0,967
25	25	12	300	500	9,061	0,247	9,404	8,718	0,685
30	10	12	300	500	9,130	0,116	9,291	8,970	0,321
35	10	12	300	500	9,216	0,159	9,437	8,995	0,441
20	5	12	300	500	9,258	0,346	9,738	8,778	0,959
30	15	12	300	500	8,907	0,222	9,216	8,598	0,618

35	20	12	300	500	8,921	0,480	9,588	8,255	1,333
25	10	20	300	500	8,701	0,225	9,013	8,389	0,623
25	10	30	300	500	8,240	0,275	8,622	7,858	0,764
35	10	30	300	500	8,079	0,147	8,283	7,876	0,407
35	5	30	300	500	8,074	0,145	8,275	7,874	0,401
35	5	30	400	500	7,543	0,310	7,973	7,113	0,861
35	5	30	500	500	7,416	0,221	7,722	7,109	0,612
35	5	30	600	600	6,802	0,105	7,145	6,854	0,291

8.Wyniki końcowe

Dla wyznaczonych parametrów
czyli:
N = 35
R = 5
Q = 30
T1 = 600
T2 = 600
Przeprowadziłem 20 symulacji uzyskując średnie wyniki:

Statystyki:
Krew przechodzaca przez punkt = 2583
Krew przechodzaca przez punkt gr.A = 1883
Krew przechodzaca przez punkt gr.B = 700
Krew przechodzaca przez punkt gr.A % = 72
Krew przechodzaca przez punkt gr.B % = 27
Zuzyta krew przez pacjentow = 1058
Krew wyslana na badania naukowe = 185
Krew wyslana na badania naukowe gr.A = 90
Krew wyslana na badania naukowe gr.B= 95
Srednia liczba jednostek krwi w kolejce = 5.74962
Zutylizowana krew = 1321 **Odch.stand**. = 0.4235 **Przedział ufności** = 0.2204
Zutylizowana krew z normalnego zamowienia = 456
Zutylizowana krew ze specjalneg zamowienia = 284
Zutylizowana krew od dawcow = 581
Procent zutylizowanej krwi = 51.1421%
Procent zutylizowanej krwi z normalnego zamowienia = 17.6539%
Procent zutylizowanej krwi ze specjalnego zamowienia = 10.995%
Procent zutylizowanej krwi od dawcow = 0.851723%
Pacjentow przechodzacych przez punkt = 235

Srednia liczba pacjentow w kolejce = 0.358779
Odch.stand. = 0.9135 **Przedział ufności** = 0.865
Liczba normalnych zamowien = 36
Liczba specialnych zamowien = 43
Prawdopodobienstwo wystapienia awaryjnego zamowienia = 6.80389%
Odch.stand. = 0.105 **Przedział ufności** = 0.291

Przedziały ufności liczone za pomocą wzorów oraz rozkładu t-studenta
(zał. 95% przedział ufności)

$$P\left(-t_{\alpha,n-1} < t < t_{\alpha,n-1}\right) = 1-\alpha \qquad t = \frac{\overline{X} - m}{S}\sqrt{n-1}$$

9.Wnioski
W moim programie symulacyjnym należało zmienić wszystkie parametry z możliwością zmiany. To znaczy ,że sama zmiana parametrów N oraz R nie przynosiła oczekiwanego skutku. Przyczyną takiego stanu rzeczy może być niedoskonałość symulacji.
Program symulacyjny jest skomplikowanym projektem w którym wszystkie elementy systemu muszą być odpowiednią zsynchronizowane z pozotałymi. Lecz jeśli program jest napisany prawidłowo staje się niezwykle użytecznym narzędziem symulującym realne zjawiska. Umożliwia dzięki temu optymalizacje wielu procesów nie narażając na ogromne koszty.

10.Kod programu
Symulka_poprawa.cpp

```
#include "pch.h"
#include <iostream>
#include "Event.h"
#include "Process.h"
#include "EventList.h"
#include "QueueRecipient.h"
#include "QueueUnitBlood.h"
#include "DonorProcess.h"
#include "RecipientProcess.h"
#include "OrderSpecialProcess.h"
#include "OrderProcess.h"
#include "ScientificResearchProcess.h"
#include "Statistics.h"
#include "Generators.h"

using namespace std;

int main()
{
```

```
	int amountOfSimulation;
	cout << "Liczba symulacji : ";
	cin >> amountOfSimulation;
	Generators *generator = new Generators();
	for (int i = 0; i < amountOfSimulation; i++)
	{
		BloodDonationPoint *bPoint = new BloodDonationPoint();
		bPoint->setMaxTimeOfSimulation(60000);// max czas symulacji
		bPoint->setMinAmountBlood(5); //prog R
		bPoint->setTimeOfUseBloodFromOrder(600); //czas przydatnosci krwi z
zamowien T1
		bPoint->setTimeOfUseBloodFromDonor(600); //czas przydatnoci krwi od dawcy
T2
		EventList *eList = new EventList();
		QueueUnitBlood *bloodQueueA = new QueueUnitBlood();
		QueueUnitBlood *bloodQueueB = new QueueUnitBlood();
		bloodQueueA->setBloodDonationPoint(bPoint);
		bloodQueueB->setBloodDonationPoint(bPoint);
		QueueRecipient *recipientQueueA = new QueueRecipient();
		recipientQueueA->setBloodDonationPoint(bPoint);
		QueueRecipient *recipientQueueB = new QueueRecipient();
		recipientQueueB->setBloodDonationPoint(bPoint);
		UtilizationProcess *donorUtilizationA = new UtilizationProcess(bPoint,
eList, bloodQueueA);
		UtilizationProcess *donorUtilizationB = new UtilizationProcess(bPoint,
eList, bloodQueueB);
		UtilizationProcess *orderUtilizationA = new UtilizationProcess(bPoint,
eList, bloodQueueA);
		UtilizationProcess *orderUtilizationB = new UtilizationProcess(bPoint,
eList, bloodQueueB);
		UtilizationProcess *specialOrderUtilizationA = new
UtilizationProcess(bPoint, eList, bloodQueueA);
		UtilizationProcess *specialOrderUtilizationB = new
UtilizationProcess(bPoint, eList, bloodQueueB);
		ScientificResearchProcess *scResearchProcessA = new
ScientificResearchProcess(bPoint, eList, bloodQueueA);
		ScientificResearchProcess *scResearchProcessB = new
ScientificResearchProcess(bPoint, eList, bloodQueueB);
		scResearchProcessA->bloodPoint = bPoint;
		scResearchProcessA->eList = eList;
		scResearchProcessA->queueBlood = bloodQueueA;
		scResearchProcessB->bloodPoint = bPoint;
		scResearchProcessB->eList = eList;
		scResearchProcessB->queueBlood = bloodQueueB;
		RecipientProcess *recipientProcess = new RecipientProcess(bPoint, eList,
bloodQueueA, bloodQueueB, recipientQueueA,
			recipientQueueB,scResearchProcessA, scResearchProcessB);
		OrderSpecialProcess *specialOrderProcessA = new
OrderSpecialProcess(bPoint, eList, bloodQueueA, specialOrderUtilizationA,
			recipientProcess, scResearchProcessA,false);
		OrderSpecialProcess *specialOrderProcessB = new
OrderSpecialProcess(bPoint, eList, bloodQueueB, specialOrderUtilizationB,
			recipientProcess, scResearchProcessB,true);
		specialOrderProcessA->setAmountOfSpecialOrder(30); // liczba Q zamowionej
krwi - specjalne zamówienie
		specialOrderProcessB->setAmountOfSpecialOrder(30);
		recipientProcess->setSpecialOrderA(specialOrderProcessA);
		recipientProcess->setSpecialOrderB(specialOrderProcessB);
		OrderProcess *orderProcessA = new OrderProcess(bPoint, eList, bloodQueueA,
orderUtilizationA, recipientProcess, scResearchProcessA,false);
		OrderProcess *orderProcessB = new OrderProcess(bPoint, eList, bloodQueueB,
orderUtilizationB, recipientProcess, scResearchProcessB,true);
```

```
		orderProcessA->setAmountOfOrder(35); // liczba N zamowionej krwi -
normalne zamowienie
		orderProcessB->setAmountOfOrder(35);
		recipientProcess->setOrderA(orderProcessA);
		recipientProcess->setOrderB(orderProcessB);
		DonorProcess *donorProcess = new DonorProcess(bPoint, eList, bloodQueueA,
donorUtilizationA, recipientProcess);
		Statistics *statSimulation = new Statistics(bPoint, donorProcess,
orderProcessA, orderProcessB, recipientQueueA, recipientQueueB,
			bloodQueueA, bloodQueueB, scResearchProcessA,scResearchProcessB,
donorUtilizationA, donorUtilizationB,
			orderUtilizationA, orderUtilizationB, specialOrderUtilizationA,
specialOrderUtilizationB, recipientProcess);
		recipientProcess->setNeededBloodGeometric_0_23(generator-
>generateGeometric(0.23));
		recipientProcess->setTimeOfOrderExponential_1900(generator-
>generateExponentinal(1900));
		recipientProcess->setTimeOfRecipientExponential_200(generator-
>generateExponentinal(200));
		recipientProcess->setTimeOfSpecialOrderNormal_400(generator-
>generateNormal(400, 0.1));
		donorProcess->setTimeOfComingDonorExponential_850(generator-
>generateExponentinal(850));
		scResearchProcessA->setAmountOfBloodUniform_5_10(generator-
>generateUniform(5, 10));
		scResearchProcessB->setAmountOfBloodUniform_5_10(generator-
>generateUniform(5, 10));
		Process *wskProcess = nullptr;
		donorProcess->activate(30);
		recipientProcess->activate(20);
		cout << "Tryb pracy symulacji:" << endl;
		cout << "Wybierz : 0.Ciaglym" << endl;
		cout << "Wybierz : 1.Krokowy" << endl;
		bool mode;
		char a;
		cin >> a;
		if (a == '1')
			mode = true;
		else
			mode = false;
		cout << endl << endl;
		bool beginingPhase = false;
		while (bPoint->getClock() < bPoint->getMaxTimeOfSimulation())
		{
			if(bPoint->getClock() > 10000 && !beginingPhase )
			{
				beginingPhase = true;
				bloodQueueA->clearAmountOfBloodStatistics();
				bloodQueueB->clearAmountOfBloodStatistics();
				recipientProcess->clearBloodPacientStatistics();
				recipientProcess->clearSpecialOrderID();
				scResearchProcessA->clearAmountOfBloodStatistic();
				scResearchProcessB->clearAmountOfBloodStatistic();
				bPoint->clearAmountOfIteracions();
				donorUtilizationA->clearAmountOfUtilizeBloodStatistics();
				orderUtilizationA->clearAmountOfUtilizeBloodStatistics();
				specialOrderUtilizationA-
>clearAmountOfUtilizeBloodStatistics();
				donorUtilizationB->clearAmountOfUtilizeBloodStatistics();
				orderUtilizationB->clearAmountOfUtilizeBloodStatistics();
				specialOrderUtilizationB-
>clearAmountOfUtilizeBloodStatistics();
```

```
                        orderProcessA->clearOrderID();
                        orderProcessB->clearOrderID();
                        recipientQueueA->clearAmountOfPatientStatistics();
                        recipientQueueB->clearAmountOfPatientStatistics();
                }
                for (int i = 0; i < 6 * generator->uniform(); i++)
                {
                        generator->generateGrain();
                }
                recipientProcess->setNeededBloodGeometric_0_23(generator-
>generateGeometric(0.23));
                recipientProcess->setTimeOfOrderExponential_1900(generator-
>generateExponentinal(1900));
                recipientProcess->setTimeOfRecipientExponential_200(generator-
>generateExponentinal(200));
                recipientProcess->setTimeOfSpecialOrderNormal_400(generator-
>generateNormal(400, 0.1));
                donorProcess->setTimeOfComingDonorExponential_850(generator-
>generateExponentinal(850));
                scResearchProcessA->setAmountOfBloodUniform_5_10(generator-
>generateUniform(5, 10));
                scResearchProcessB->setAmountOfBloodUniform_5_10(generator-
>generateUniform(5, 10));
                bPoint->setClock(eList->getFirstTime());
                if (bPoint->getClock() < bPoint->getMaxTimeOfSimulation())
                {
                        cout << "Czas symulacji: " << bPoint->getClock() << endl;
                        wskProcess = eList->getFirstEvent()->getProcess();
                        eList->removeEvent();// usuwamy z listy zdarzeń przed metodą
execute() -bo ona dodaje to samo zdarzenie do listy zdarzeń
                        wskProcess->execute();
                        cout << "Liczba pacjentow w kolejce gr A: " <<
recipientQueueA->howManyRecipients() << endl;
                        cout << "Liczba pacjentow w kolejce gr B: " <<
recipientQueueB->howManyRecipients() << endl;
                        cout << "Liczba jednostek krwi gr A: " << bPoint-
>getAmountBloodA() << endl;
                        cout << "Liczba jednostek krwi gr B: " << bPoint-
>getAmountBloodB() << endl;
                        cout << endl;
                        if (mode)
                        {
                                cout << "Tryb pracy symulacji:" << endl;
                                cout << "Wybierz : 0.Ciaglym" << endl;
                                cout << "Wybierz : Enter.Krokowy" << endl;
                                int a = getchar();
                                if (a != 10)
                                        mode = false;
                                else
                                        mode = true;
                                cout << endl << endl << endl;
                        }
                }
                bPoint->incrementAmoutnOfIteracions();
        }
        statSimulation->viewStat();
        statSimulation->printToFile();
        delete bPoint;
        delete eList;
        delete bloodQueueA;
        delete bloodQueueB;
        delete recipientQueueA;
```

```
            delete recipientQueueB;
            delete donorUtilizationA;
            delete donorUtilizationB;
            delete orderUtilizationA;
            delete orderUtilizationB;
            delete specialOrderUtilizationA;
            delete specialOrderUtilizationB;
            delete scResearchProcessA;
            delete scResearchProcessB;
            delete recipientProcess;
            delete specialOrderProcessA;
            delete specialOrderProcessB;
            delete orderProcessA;
            delete orderProcessB;
            delete donorProcess;
            delete statSimulation;
        }
}
```

BloodDonationPoint.h

```
class BloodDonationPoint
{
public:
        int getClock() const;
        void setClock(int newClock);
        void setAmountBloodA(int);
        int getAmountBloodA();
        void setAmountBloodB(int);
        int getAmountBloodB();
        void setMinAmountBlood(int);
        int getMinAMountBlood();
        void setTimeOfUseBloodFromOrder(int);
        int getTimeOfUseBloodFromOrder();
        void setTimeOfUseBloodFromDonor(int);
        int getTimeOfUseBloodFromDonor();
        void setMaxTimeOfSimulation(int);
        int getMaxTimeOfSimulation();
        void incrementAmoutnOfIteracions();
        int getAmountOfIteracions();
        void clearAmountOfIteracions();
        void setOrderFlagA();
        void clearOrderFlagA();
        bool checkOrderFlagA();
        void setSpecialOrderFlagA();
        void clearSpecialOrderFlagA();
        bool checkSpecialOrderFlagA();
        void setOrderFlagB();
        void clearOrderFlagB();
        bool checkOrderFlagB();
        void setSpecialOrderFlagB();
        void clearSpecialOrderFlagB();
        bool checkSpecialOrderFlagB();
        BloodDonationPoint();
        ~BloodDonationPoint();
private:
        int clock;
        int amountBloodA;
        int amountBloodB;
        int minAmountBlood;
        int timeOfUseBloodFromOrder;
        int timeOfUseBloodFromDonor;
```

```
	int maxTimeOfSimulation;
	bool orderFlagA;
	bool specialOrderFlagA;
	bool orderFlagB;
	bool specialOrderFlagB;
	int amountOfIteracions;
};
```

BloodDonationPoint.cpp

```
#include "pch.h"
#include "BloodDonationPoint.h"

int BloodDonationPoint::getClock() const
{
	return clock;
}

void BloodDonationPoint::setClock(int newClock)
{
	clock = newClock;
}

void BloodDonationPoint::setAmountBloodA(int amout)
{
	amountBloodA = amout;
}

int BloodDonationPoint::getAmountBloodA()
{
	return amountBloodA;
}

void BloodDonationPoint::setAmountBloodB(int number)
{
	amountBloodB = number;
}
int BloodDonationPoint::getAmountBloodB()
{
	return amountBloodB;
}

void BloodDonationPoint::setMinAmountBlood(int amount)
{
	minAmountBlood = amount;
}

int BloodDonationPoint::getMinAMountBlood()
{
	return minAmountBlood;
}

void BloodDonationPoint::setTimeOfUseBloodFromOrder(int number)
{
	timeOfUseBloodFromOrder = number;
}

int BloodDonationPoint::getTimeOfUseBloodFromOrder()
{
	return timeOfUseBloodFromOrder;
}
```

```
void BloodDonationPoint::setTimeOfUseBloodFromDonor(int number)
{
	timeOfUseBloodFromDonor = number;
}

int BloodDonationPoint::getTimeOfUseBloodFromDonor()
{
	return timeOfUseBloodFromDonor;
}

void BloodDonationPoint::setMaxTimeOfSimulation(int number)
{
	maxTimeOfSimulation = number;
}
int BloodDonationPoint::getMaxTimeOfSimulation()
{
	return maxTimeOfSimulation;
}

void BloodDonationPoint::incrementAmoutnOfIteracions()
{
	amountOfIteracions++;
}
int BloodDonationPoint::getAmountOfIteracions()
{
	return amountOfIteracions;
}

void BloodDonationPoint::setOrderFlagA()
{
	orderFlagA = true;
}

void BloodDonationPoint::clearOrderFlagA()
{
	orderFlagA = false;
}

bool BloodDonationPoint::checkOrderFlagA()
{
	return orderFlagA;
}

void BloodDonationPoint::setSpecialOrderFlagA()
{
	specialOrderFlagA = true;
}

void BloodDonationPoint::clearSpecialOrderFlagA()
{
	specialOrderFlagA = false;
}

bool BloodDonationPoint::checkSpecialOrderFlagA()
{
	return specialOrderFlagA;
}

void BloodDonationPoint::setOrderFlagB()
{
	orderFlagB = true;
}
```

```
void BloodDonationPoint::clearOrderFlagB()
{
	orderFlagB = false;
}

bool BloodDonationPoint::checkOrderFlagB()
{
	return orderFlagB;
}

void BloodDonationPoint::setSpecialOrderFlagB()
{
	specialOrderFlagB = true;
}

void BloodDonationPoint::clearSpecialOrderFlagB()
{
	specialOrderFlagB = false;
}

bool BloodDonationPoint::checkSpecialOrderFlagB()
{
	return specialOrderFlagB;
}


void BloodDonationPoint::clearAmountOfIteracions()
{
	amountOfIteracions = 0;
}

BloodDonationPoint::BloodDonationPoint()
{
	clock = 0;
	amountBloodA = 0;
	amountBloodB = 0;
	minAmountBlood = 10;
	timeOfUseBloodFromOrder = 300;
	timeOfUseBloodFromDonor = 500;
	amountOfIteracions = 0;
	maxTimeOfSimulation = 1000;
	orderFlagA = false;
	specialOrderFlagA= false;
	orderFlagB = false;
	specialOrderFlagB = false;
}

BloodDonationPoint::~BloodDonationPoint()
{
}
```

DonorProcess.h

```
#pragma once
#include "Process.h"
#include "UtilizationProcess.h"
#include "RecipientProcess.h"


class RecipientProcess;
class DonorProcess :
```

```
	public Process
{
public:
	void execute() override;
	void setTimeOfComingDonorExponential_850(int);
	DonorProcess(BloodDonationPoint *bPoint, EventList *el, QueueUnitBlood *qBlood,
UtilizationProcess *uP,
		RecipientProcess *rProcess);
	~DonorProcess();
private:
	Event *myEvent;
	UtilizationProcess *uProcess;
	RecipientProcess *recipientProcess;
	int timeOfComingDonorExponential_850;
};
```

DonorProcess.cpp

```
#include "pch.h"
#include "DonorProcess.h"
#include <iostream>

using namespace std;

void DonorProcess::execute()
{
	cout << "'DonorProcess': Obsluga dawcy" << endl;
	int clock = bloodPoint->getClock();
	UnitBlood *blood = new UnitBlood(clock + bloodPoint-
>getTimeOfUseBloodFromDonor());
	queueBlood->addBlood(blood);
	cout << "- dodanie jednostki krwi do kolejki krwi" << endl;
	activate(timeOfComingDonorExponential_850);
	cout << "- zaplanowanie kolejnego procesu 'DonorProcess'" << endl;
	uProcess->setHowManyToRemove(1);
	uProcess->activate(bloodPoint->getTimeOfUseBloodFromDonor());
	cout << "- zaplanowanie usuniecia jednostki krwi 'UtilizationProcess'" << endl;
	bool group = blood->showGroup();
	recipientProcess->reactive(group);
}

void DonorProcess::setTimeOfComingDonorExponential_850(int number)
{
	timeOfComingDonorExponential_850 = number;
}


DonorProcess::DonorProcess(BloodDonationPoint *bPoint, EventList *el, QueueUnitBlood
*qBlood, UtilizationProcess *uP,
	RecipientProcess *rProcess)
	: Process(bPoint, el, qBlood), uProcess(uP), recipientProcess(rProcess)
{

}

DonorProcess::~DonorProcess()
{
}
```

Event.h

```
#pragma once
class Process;
class Event
{
public:
	bool planRecipient;
	int number;
	Event *next;
	int phase;
	void setTime(int time);
	void showTime();
	int getTime();
	Process* getProcess();
	Event(Process *proc);
	Event();
	~Event();
private:
	Process *myProcess;
	int eventTime;
};
```

Event.cpp

```
#include "pch.h"
#include "Event.h"
#include <iostream>

using namespace std;

void Event::setTime(int time)
{
	eventTime = time;
}

void Event::showTime()
{
	cout << "Czas zdarzenia : " << eventTime << endl;
}

int Event::getTime()
{
	return eventTime;
}

Process* Event::getProcess()
{
	return myProcess;
}

Event::Event(Process *proc)
{
	myProcess = proc;
	next = nullptr;
	phase = 3;
	eventTime = 0;
	planRecipient = false;
}

Event::Event()
{
	myProcess = nullptr;
```

```
	next = nullptr;
	phase = 3;
	eventTime = 0;
	planRecipient = false;
}

Event::~Event()
{
}
```

EventList.h

```
#pragma once
#include "Event.h"

class Event;
class EventList
{
public:
	void addEvent(Event *newEvent);
	void addToBegining(Event *newEvent);
	void removeEvent();
	int howMany();
	void sortEvent();
	void showList();
	int getFirstTime();
	void removeSelectedEvent(Event* selected);
	Event* getFirstEvent();
	Event* getLastEvent();
	bool lookForEvent(Event*);
	EventList();
	~EventList();
private:
	Event *root;
};
```

EventList.cpp

```
#include "pch.h"
#include "EventList.h"
#include <iostream>
using namespace std;

void EventList::addEvent(Event *newEvent)
{
	Event *pom = root;
	bool isChild = true;
	while (isChild)
	{
		if (pom->next == nullptr)
		{
			pom->next = newEvent;
			isChild = false;
		}
		else
		{
			isChild = true;
			pom = pom->next;
		}
	}
}
```

```
void EventList::addToBegining(Event *newEvent)
{
	if (root->next)
	{
		Event *pom = root->next;
		root->next = newEvent;
		newEvent->next = pom;
	}
	else
		root->next = newEvent;
}

void EventList::removeEvent()
{
	if (root->next)
	{
		if (root->next->next)
		{
			Event *pom = root->next->next;
			root->next->next = nullptr;
			root->next = pom;
		}
		else
		{
			root->next = nullptr;
		}

	}

}

int EventList::howMany()
{
	Event *pom = root;
	bool isChild = true;
	int howManyChild = 0;
	while (isChild)
	{
		if (pom->next == nullptr)
		{
			return howManyChild;
			isChild = false;
		}
		else
		{
			isChild = true;
			howManyChild++;
			pom = pom->next;
		}
	}
}

void EventList::sortEvent()
{
	Event *pom = root;
	bool sorted = false;
	int finishSort = 0;
	int how = howMany();
	while (!sorted)
	{
		pom = root;
```

```
		finishSort = 0;
		for (int i = 0; i < how - 1; i++)
		{
			if (pom->next->next)
			{
				if (pom->next->getTime() > pom->next->next->getTime())
				{
					Event *pom2, *pom3;
					pom2 = pom->next;
					pom3 = pom->next->next;
					if (pom3->next)
						pom2->next = pom3->next;
					else
						pom2->next = nullptr;
					pom->next = pom3;
					pom3->next = pom2;
					pom = pom->next;
					sorted = false;
					finishSort = 0;

				}
				else
				{
					pom = pom->next;
					finishSort++;
				}
			}
		}
		if (finishSort == howMany() - 1)
		{
			sorted = true;
		}
	}


}

void EventList::showList()
{
	Event *pom = root;
	bool isChild = true;
	while (isChild)
	{
		if (pom->next == nullptr)
		{
			pom->showTime();
			isChild = false;
		}
		else
		{
			isChild = true;
			pom->showTime();
			pom = pom->next;
		}
	}
}

int EventList::getFirstTime()
{
	return root->next->getTime();
}
```

```
void EventList::removeSelectedEvent(Event* selected)
{
	Event *pom = root;
	bool isChild = true;
	if (root->next)
	{
		while (isChild)
		{
			if (pom->next == selected && pom->next)
			{
				if (pom->next->next)
				{
					Event *pom2 = pom->next;
					pom->next = pom->next->next;
					pom2->next = nullptr;
				}
				else
				{
					pom->next = nullptr;
				}

				isChild = false;
			}
			else if (pom->next)
			{
				isChild = true;
				pom = pom->next;
			}
			else
			{
				isChild = false;
			}
		}
	}



}

bool EventList::lookForEvent(Event* a)
{
	Event *pom = root;
	bool isChild = true;
	if (root->next)
	{
		while (isChild)
		{
			if (pom->next == a && pom->next)
			{
				return true;
				isChild = false;
			}
			else if (pom->next)
			{
				isChild = true;
				pom = pom->next;
			}
			else
			{
				isChild = false;
			}
```

```
			}
		}
		return false;

}

Event* EventList::getFirstEvent()
{
		return root->next;
}

Event* EventList::getLastEvent()
{
		Event *pom = root;
		bool isChild = true;
		while (isChild)
		{
			if (pom->next == nullptr)
			{
				return pom;
				isChild = false;
			}
			else
			{
				isChild = true;
				pom = pom->next;
			}
		}
}

EventList::EventList()
{
		root = new Event();
		root->setTime(-1);
}


EventList::~EventList()
{
}
```

Generators.h

```
#pragma once

class Generators
{

		int m;
		int a;
		int q;
		int r;
public:
		int grain;
		double generateGrain();
		double uniform();
		int generateExponentinal(double lambda);
		int generateGeometric(double average);
		int generateNormal(int average,double var);
		int generateUniform(int min,int max);
		/*void testGenerateExponentinal(double lambda, int samples);
		void testGenerateGeometric(double w, int samples);
```

```
	void testGenerateNormal(const double e, const double ew, const int samples);
	void testGenerateUniform(int min, int max,int samples);*/
	Generators();//Statistics *s
	~Generators();
};
```

Generators.cpp

```
#include "pch.h"
#include "Generators.h"
#include <iostream>

using namespace std;


double Generators::generateGrain()
{
	int h =abs(grain / q);
	grain = a * (grain - q * h) - r * h;
	if (grain < 0)
		grain = grain + m;
	return grain;
}


double Generators::uniform()
{
	grain = grain * a % m;
	return abs(static_cast<double>(grain)/m);
}

int Generators::generateUniform(const int min,const int max)
{
	return (int)uniform()*(max - min) + min;
}

int Generators::generateNormal(int average, double var)
{
	double k = 0;
	for(int i=0;i<12;i++)
	{
		k += uniform();
	}
	k = k - 6;
	k = sqrt(var)*k + average;
	return (int)k;
}

int Generators::generateExponentinal(double lambda)
{
	lambda = 1 / lambda;
	return (int)(-(1.0 / lambda)*log(uniform()));
}

int Generators::generateGeometric(double average)
{
	int k = 1;
	/*
	while (uniform()>average)
	{
		++k;
```

```
	}
	return k;
	*/
	while (true)
	{
		if (uniform() > average)
		{
			k++;
			continue;
		}
		break;
	}
	return k;
}
/*
void Generators::testGenerateExponentinal(double lambda,int samples)
{
	double average = 0;
	int result;
	lambda = 1 / lambda;
	cout << "Start Test Exponential Generator." << endl;
	for (int i = 0; i < samples; ++i) {
		result = -1 / lambda * log(uniform());

		stat->saveStatsToFile(result, "exponential.xls");
		average = average + result;
	}

	cout << "average : " << average / samples << endl;
}

void Generators::testGenerateGeometric(double w,int samples)
{
	cout << "Start Test Geometric Generator." << endl;
	double ave = 0;
	int result;
	for (auto i = 0; i < samples; ++i) {
		result = 1;
		while (true)
		{
			if (uniform() > w)
			{
				result++;
				continue;
			}
			break;
		}

		stat->saveStatsToFile(result, "geometric.xls");
		ave = ave + result;
	}

	cout << "w : " << 1 / (ave / samples) << endl;
	cout << "average : " << double(ave / samples) << endl;
}

void Generators::testGenerateNormal(const double e, const double ew, const int samples)
{
	cout << "Start Test Normal Generator." << endl;
	double average = 0;
	int result;
	for (auto i = 0; i < samples; ++i)
```

```
	{
		result = 0;
		double temp = 0;
		for (auto index = 0; index < 12; index++)
		{
			temp += uniform();
		}
		temp -= 6.0;
		result= temp * ew + e;

		stat->saveStatsToFile(result, "normal.xls");
		average = average + result;
	}

	cout << "average : " << average / samples << endl;
}

void Generators::testGenerateUniform(int min, int max,int samples)
{
	double average = 0;
	double numberUniform = 0;
	cout << "Start Test Uniform Generator." << endl;
	generateGrain();
	for (auto i = 0; i < samples; ++i)
	{
		numberUniform = min + (max - min) * uniform();
		stat->saveStatsToFile(numberUniform, "uniform.xls");
		average = average + numberUniform;
	}
	cout << "average : " << average / samples << endl;
}
*/
Generators::Generators()//Statistics *s :stat(s)
{
	m = 2147483647;
	a = 16807;
	q = 127773;
	r = 2836;
	grain = 1179;
}


Generators::~Generators()
{
}
```

OrderProcess.h

```
#pragma once
#include "Process.h"
#include "BloodDonationPoint.h"
#include "UtilizationProcess.h"
#include "RecipientProcess.h"
#include "ScientificResearchProcess.h"

class RecipientProcess;
class OrderProcess :
	public Process
{
public:
	void execute() override;
	void activate(int time,bool);
```

```
	void setAmountOfOrder(int);
	int getAmountOfOrder();
	int getOrderID();
	void clearOrderID();
	OrderProcess(BloodDonationPoint *bPoint, EventList *el, QueueUnitBlood *qBlood,
UtilizationProcess *uP,
		RecipientProcess *rProcess, ScientificResearchProcess *scProc,bool gr);
	~OrderProcess();
private:
	Event *myEvent;
	int amountOfOrder;
	RecipientProcess *recipientProcess;
	UtilizationProcess *uProcess;
	ScientificResearchProcess *scResearchProcess;
	bool group;
	int orderID;
};
```

OrderProcess.cpp

```
#include "pch.h"
#include "OrderProcess.h"
#include <iostream>

using namespace std;

void OrderProcess::execute()
{
	orderID++;
	if(group)
	{
		cout << "'OrderProcess': Standardowe zamowienie gr.B" << endl;
		bloodPoint->clearOrderFlagB();
	}
	else
	{
		cout << "'OrderProcess': Standardowe zamowienie gr.A" << endl;
		bloodPoint->clearOrderFlagA();
	}
	for (int i = 0; i < amountOfOrder; i++)
	{
		UnitBlood *blood = new UnitBlood(bloodPoint->getClock() + bloodPoint-
>getTimeOfUseBloodFromOrder(),group);
		queueBlood->addBlood(blood);
	}
	cout << "- dodanie " << amountOfOrder << " jednostek krwi do koleji krwi" <<
endl;
	cout << "- zaplanowanie utylizacji jednostek krwi" << endl;
	recipientProcess->reactive(group);
	scResearchProcess->checkCondition(group);
	uProcess->setHowManyToRemove(amountOfOrder);
	uProcess->activate(bloodPoint->getTimeOfUseBloodFromOrder());
}

void OrderProcess::activate(int time,bool group)
{
		myEvent->setTime(bloodPoint->getClock() + time);
		eList->addEvent(myEvent);
		eList->sortEvent();
		if (group)
		{
```

```
                bloodPoint->setOrderFlagB();
        }
        else
        {
                bloodPoint->setOrderFlagA();
        }
}

void OrderProcess::setAmountOfOrder(int number)
{
        amountOfOrder = number;
}

int OrderProcess::getAmountOfOrder()
{
        return amountOfOrder;
}

int OrderProcess::getOrderID()
{
        return orderID;
}

void OrderProcess::clearOrderID()
{
        orderID = 0;
}

OrderProcess::OrderProcess(BloodDonationPoint *bPoint, EventList *el, QueueUnitBlood
*qBlood,
        UtilizationProcess *uP, RecipientProcess *rProcess, ScientificResearchProcess
*scProc,bool gr)
        : Process(bPoint, el, qBlood),uProcess(uP), recipientProcess(rProcess),
scResearchProcess(scProc),group(gr)
{
        myEvent = new Event(this);
        amountOfOrder = 40;
}

OrderProcess::~OrderProcess()
{
}
```

OrderSpecialProcess.h

```
#pragma once
#include "Process.h"
#include "BloodDonationPoint.h"
#include "UtilizationProcess.h"
#include "QueueRecipient.h"
#include "RecipientProcess.h"
#include "ScientificResearchProcess.h"

class RecipientProcess;
class QueueRecipient;
class OrderSpecialProcess :
        public Process
{
public:
        void execute() override;
        void activate(int time);
```

```
	void setAmountOfSpecialOrder(int);
	OrderSpecialProcess(BloodDonationPoint *bPoint, EventList *el, QueueUnitBlood
*qBlood, UtilizationProcess *uP,
		RecipientProcess *rProcess, ScientificResearchProcess *scProc,bool gr);
	~OrderSpecialProcess();
private:
	Event*myEvent;
	UtilizationProcess *uProcess;
	QueueRecipient *qRecipient;
	RecipientProcess *recipientProcess;
	ScientificResearchProcess *scResearchProcess;
	int amountOfSpecialOrder;
	bool group;
};
```

OrderSpecialProcess.cpp

```
#include "pch.h"
#include "OrderSpecialProcess.h"
#include <iostream>

using namespace std;

void OrderSpecialProcess::execute()
{
	if (group)
	{
		cout << "'OrderSpecialProcess': Specialne zamowienie gr.B" << endl;
		bloodPoint->clearSpecialOrderFlagB();
	}
	else
	{
		cout << "'OrderSpecialProcess': Specialne zamowienie gr.A" << endl;
		bloodPoint->clearSpecialOrderFlagA();
	}
	for (int i = 0; i < amountOfSpecialOrder; i++)
	{
		UnitBlood *blood = new UnitBlood(bloodPoint->getClock() + bloodPoint-
>getTimeOfUseBloodFromOrder(),group);
		queueBlood->addBlood(blood);
	}
	cout << "- dodanie " << amountOfSpecialOrder << " jednostek krwi do koleji krwi"
<< endl;
	cout << "- zaplanowanie utylizacji jednostek krwi" << endl;
	recipientProcess->reactive(group);
	scResearchProcess->checkCondition(group);
	uProcess->setHowManyToRemove(amountOfSpecialOrder);
	uProcess->activate(bloodPoint->getTimeOfUseBloodFromOrder());
}

void OrderSpecialProcess::activate(int time)
{
	myEvent->setTime(bloodPoint->getClock() + time);
	eList->addEvent(myEvent);
	eList->sortEvent();
	if (group)
	{
		bloodPoint->setSpecialOrderFlagB();
	}
	else
	{
```

```
                bloodPoint->setSpecialOrderFlagA();
        }
}

void OrderSpecialProcess::setAmountOfSpecialOrder(int number)
{
        amountOfSpecialOrder = number;
}

OrderSpecialProcess::OrderSpecialProcess(BloodDonationPoint *bPoint, EventList *el,
QueueUnitBlood *qBlood, UtilizationProcess *uP,
        RecipientProcess *rProcess, ScientificResearchProcess *scProc, bool gr)
        : Process(bPoint, el, qBlood), uProcess(uP), recipientProcess(rProcess),
scResearchProcess(scProc),group(gr)
{
        myEvent = new Event(this);
        amountOfSpecialOrder = 20;
}


OrderSpecialProcess::~OrderSpecialProcess()
{
}
```

Process.h

```
#pragma once
#include "EventList.h"
#include "BloodDonationPoint.h"
#include "QueueUnitBlood.h"

class EventList;
class Event;
class Process
{
public:
        EventList *eList;
        BloodDonationPoint *bloodPoint;
        QueueUnitBlood *queueBlood;
        void virtual execute() = 0;
        void activate(int time);
        int phase;
        Process(BloodDonationPoint *bPoint,EventList *el,QueueUnitBlood *qBlood);
        virtual ~Process();
private:
        Event *myEvent;
};
```

Process.cpp

```
#include "pch.h"
#include "Process.h"
#include <iostream>

using namespace std;

void Process::activate(int time)
{
        myEvent->setTime(bloodPoint->getClock() + time);
        eList->addEvent(myEvent);
        eList->sortEvent();
}
```

```
Process::Process(BloodDonationPoint *bPoint, EventList *el,QueueUnitBlood *qBlood) :
bloodPoint(bPoint),eList(el),queueBlood(qBlood)
{
	myEvent = new Event(this);
	phase = 0;
}

Process::~Process()
{
}
```

QueueRecipient.h

```
#pragma once
#include "Recipient.h"
#include "Statistics.h"

class Statistics;
class QueueRecipient
{
public:
	void addRecipient(Recipient *add);
	void removeRecipient();
	void showRecipient();
	int howManyRecipients();
	int getAmountOfPatientStatistics();
	void clearAmountOfPatientStatistics();
	void removeSelectedRecipient(Recipient* selected);
	void setBloodDonationPoint(BloodDonationPoint*);
	Recipient* getFirstRecipient();
	Recipient* getRecipient(int number);
	QueueRecipient();
	~QueueRecipient();
private:
	Recipient *root;
	Statistics *stat;
	BloodDonationPoint *bloodPoint;
	int amountOfPatientStatistics;
};
```

QueueRecipient.cpp

```
#include "pch.h"
#include "QueueRecipient.h"
#include <iostream>

using namespace std;

void QueueRecipient::addRecipient(Recipient *add)
{
	Recipient *pom = root;
	bool isChild = false;
	do
	{
		if (pom->next == nullptr)
		{
			pom->next = add;

			isChild = false;
```

```
		}
		else
		{
			isChild = true;
			pom = pom->next;
		}
	} while (isChild);
	//stat.saveStatsToFile2(howManyRecipients(),bloodPoint->getClock(), "kolejka");
	amountOfPatientStatistics++;
}

void QueueRecipient::removeRecipient()
{
	Recipient *pom;
	if (root->next->next == nullptr)
	{
		pom = root->next;
		root->next = nullptr;
		delete pom;
	}
	else
	{
		Recipient *pom2 = root->next->next;
		pom = root->next;
		root->next = pom2;
		delete pom;
	}
}

void QueueRecipient::showRecipient()
{
	Recipient *pom = root;
	bool isChild = false;
	cout << "Kolejka pacjentow : " << endl;
	do
	{
		if (pom->next == nullptr)
		{
			cout << "nr = " << pom->number_wsk << " potrzebna krew: " << pom-
>getNeededBlood() << endl;
			isChild = false;
		}
		else
		{
			isChild = true;
			cout << "nr = " << pom->number_wsk << " potrzebna krew: " << pom-
>getNeededBlood() << endl;
			pom = pom->next;
		}
	} while (isChild);
}

int QueueRecipient::howManyRecipients()
{
	Recipient *pom = root;
	bool isChild = false;
	int howMany = 0;
	do
	{
		if (pom->next == nullptr)
		{
```

```
                        return howMany;
                        isChild = false;
                }
                else
                {
                        isChild = true;
                        howMany++;
                        pom = pom->next;
                }
        } while (isChild);
}

int QueueRecipient::getAmountOfPatientStatistics()
{
        return amountOfPatientStatistics;
}

Recipient* QueueRecipient::getFirstRecipient()
{
        return root->next;
}

Recipient* QueueRecipient::getRecipient(int number)
{
        Recipient *pom = root;
        bool isChild = false;
        int pom_number = 0;
        do
        {
                pom_number++;
                if (pom->next == nullptr)
                {
                        isChild = false;
                }
                else
                {
                        isChild = true;
                        pom = pom->next;
                }
                if (pom_number == number)
                        return pom;
        } while (isChild);
}


void QueueRecipient::removeSelectedRecipient(Recipient* selected)
{
        Recipient *pom = root;
        bool isChild = true;
        while (isChild)
        {
                if (pom->next == nullptr)
                {
                        isChild = false;
                }
                else
                {
                        isChild = true;
                        if (pom->next == selected)
                        {
                                if (pom->next->next)
                                {
```

```
                                        Recipient *pom2 = pom->next;
                                        pom->next = pom->next->next;
                                        pom2->next = nullptr;
                                }
                                else
                                        pom->next = nullptr;
                                isChild = false;
                        }
                        pom = pom->next;
                }
        }
}

void QueueRecipient::setBloodDonationPoint(BloodDonationPoint* bP)
{
        bloodPoint = bP;
}

void QueueRecipient::clearAmountOfPatientStatistics()
{
        amountOfPatientStatistics = 0;
}

QueueRecipient::QueueRecipient()
{
        root = new Recipient(-1);
        root->number_wsk = -1;
        amountOfPatientStatistics = 0;
        stat = new Statistics();
}

QueueRecipient::~QueueRecipient()
{
}
```

QueueUnitBlood.h

```
#pragma once
#include "UnitBlood.h"
#include "BloodDonationPoint.h"

class Statistics;
class QueueUnitBlood
{
public:
        void addBlood(UnitBlood *newBlood);
        void removeBlood();
        int howMany();
        void sortBlood();
        void showBlood();
        int showTimeBlood();
        int getAmoutnOfBloodStatistics();
        void clearAmountOfBloodStatistics();
        void setBloodDonationPoint(BloodDonationPoint*);
        QueueUnitBlood();
        ~QueueUnitBlood();
private:
        BloodDonationPoint *bPoint;
        UnitBlood *root;
        int amoutnOfBloodStatistics;
        Statistics *stat;
```

```
};
```

QueueUnitBlood.cpp

```
#include "pch.h"
#include "QueueUnitBlood.h"
#include <iostream>
#include <fstream>

using namespace std;

void QueueUnitBlood::addBlood(UnitBlood *newBlood)
{
	UnitBlood *pom = root;
	bool isChild = false;
	do
	{
		if (pom->next == nullptr)
		{
			pom->next = newBlood;
			isChild = false;
		}
		else
		{
			isChild = true;
			pom = pom->next;
		}
	} while (isChild);
	if(newBlood->showGroup())
		bPoint->setAmountBloodB(bPoint->getAmountBloodB() + 1);
	else
		bPoint->setAmountBloodA(bPoint->getAmountBloodA() + 1);
	sortBlood();
	amoutnOfBloodStatistics++;

	ofstream result;
	result.open("kolejka_krwi_time.xls", ios::out | ios::app);
	result << bPoint->getClock() << endl;
	result.close();
	result.open("kolejka_krwi.xls", ios::out | ios::app);
	result << howMany() << endl;
	result.close();
}

void QueueUnitBlood::removeBlood()
{
	if (root->next)
	{
		UnitBlood *pom = root->next;
		root->next = root->next->next;
		if(pom->showGroup())
			bPoint->setAmountBloodB(bPoint->getAmountBloodB() - 1);
		else
			bPoint->setAmountBloodA(bPoint->getAmountBloodA() - 1);
		delete pom;
	}

}

int QueueUnitBlood::howMany()
{
```

```
	UnitBlood *pom = root;
	bool isChild = true;
	int howManyChild = 0;
	while (isChild)
	{
		if (pom->next == nullptr)
		{
			return howManyChild;
			isChild = false;
		}
		else
		{
			isChild = true;
			howManyChild++;
			pom = pom->next;
		}
	}
}

void QueueUnitBlood::sortBlood()
{
	UnitBlood *pom = root;
	bool sorted = false;
	int finishSort = 0;
	while (!sorted)
	{
		pom = root;
		finishSort = 0;
		for (int i = 0; i < howMany() - 1; i++)
		{
			if (pom->next->next)
			{
				if (pom->next->getTimeOfBlood() > pom->next->next-
>getTimeOfBlood())
				{
					UnitBlood *pom2, *pom3;
					pom2 = pom->next;
					pom3 = pom->next->next;
					if (pom3->next)
						pom2->next = pom3->next;
					else
						pom2->next = nullptr;
					pom->next = pom3;
					pom3->next = pom2;
					pom = pom->next;
					sorted = false;
					finishSort = 0;

				}
				else
				{
					pom = pom->next;
					finishSort++;
				}
			}
		}
		if (finishSort == howMany() - 1)
			sorted = true;
	}


}
```

```
void QueueUnitBlood::showBlood()
{
	UnitBlood *pom = root;
	bool isChild = true;
	while (isChild)
	{
		if (pom->next == nullptr)
		{
			cout << "Krew - termin: " << pom->getTimeOfBlood() << endl;
			isChild = false;
		}
		else
		{
			isChild = true;
			cout << "Krew - termin: " << pom->getTimeOfBlood() << endl;
			pom = pom->next;
		}
	}
	cout << endl;
}

int QueueUnitBlood::showTimeBlood()
{
	if (root->next)
		return root->next->getTimeOfBlood();
	else
		return 0;
}

int QueueUnitBlood::getAmoutnOfBloodStatistics()
{
	return amoutnOfBloodStatistics;
}

void QueueUnitBlood::clearAmountOfBloodStatistics()
{
	amoutnOfBloodStatistics = 0;
}

void QueueUnitBlood::setBloodDonationPoint(BloodDonationPoint *bP)
{
	bPoint = bP;
}

QueueUnitBlood::QueueUnitBlood()
{
	root = new UnitBlood(-1);
	amoutnOfBloodStatistics = 0;
}


QueueUnitBlood::~QueueUnitBlood()
{
}
```

Recipient.h

```
#pragma once
#include "Generators.h"
```

```
class Recipient
{
public:
	Recipient *next;
	int number_wsk;
	int getNeededBlood();
	bool showGruop();
	Recipient(int amount);
	~Recipient();
private:
	Generators gen;
	int unitBloodNeeded;
	bool group;
};
```

Recipient.cpp

```
#include "pch.h"
#include "Recipient.h"

int Recipient::getNeededBlood()
{
	return unitBloodNeeded;
}

bool Recipient::showGruop()
{
	return group;
}


Recipient::Recipient(int amount)
{
	unitBloodNeeded = amount;
	next = nullptr;
	number_wsk = 0;
	if (gen.uniform() < 0.4)
		group = false;
	else
		group = true;
}

Recipient::~Recipient()
{
}
```

RecipientProcess.h

```
#pragma once
#include "Process.h"
#include "Recipient.h"
#include "QueueRecipient.h"
#include "OrderSpecialProcess.h"
#include "OrderProcess.h"
#include "ScientificResearchProcess.h"


class OrderSpecialProcess;
class OrderProcess;
class QueueRecipient;
class RecipientProcess :
```

```
	public Process
{
public:
	void setOrderA(OrderProcess*);
	void setSpecialOrderA(OrderSpecialProcess*);
	void setOrderB(OrderProcess*);
	void setSpecialOrderB(OrderSpecialProcess*);
	void execute() override;
	void activate(int time);
	void reactive(bool);
	int getBloodPacientStatistics();
	void clearSpecialOrderID();
	void clearBloodPacientStatistics();
	int getSpecialOrderID();
	void setTimeOfOrderExponential_1900(int);
	void setTimeOfRecipientExponential_200(int);
	void setTimeOfSpecialOrderNormal_400(int);
	void setNeededBloodGeometric_0_23(int);
	RecipientProcess(BloodDonationPoint *bPoint, EventList *el, QueueUnitBlood
*qBloodA, QueueUnitBlood *qBloodB,
		QueueRecipient *qReciA, QueueRecipient *qReciB,ScientificResearchProcess
*scProcA, ScientificResearchProcess *scProcB);
	~RecipientProcess();
private:
	EventList *events;
	Event** wsk_tab;
	Event** wsk_tab2;
	QueueUnitBlood *queueBloodB;
	Recipient *nRecipient;
	QueueRecipient *qRecipientA;
	QueueRecipient *qRecipientB;
	ScientificResearchProcess *scResearchProcessA;
	ScientificResearchProcess *scResearchProcessB;
	OrderSpecialProcess *specialOrderA;
	OrderSpecialProcess *specialOrderB;
	OrderProcess *orderA;
	OrderProcess *orderB;
	int numberEvent;
	int recipFromQueueA;
	int recipFromQueueB;
	int bloodPacientStatistics;
	int specialOrderID;
	int timeOfOrderExponential_1900;
	int timeOfRecipientExponential_200;
	int timeOfSpecialOrderNormal_400;
	int neededBloodGeometric_0_23;
};
```

RecipientProcess.cpp

```
#include "pch.h"
#include "RecipientProcess.h"
#include <iostream>

using namespace std;


void RecipientProcess::execute()
{
	bool active = true;
	bool actualGroup;
	if (recipFromQueueA > 0)
```

```
	{
		recipFromQueueA--;
		numberEvent = qRecipientA->getRecipient(1)->number_wsk;
		actualGroup = false;
	}
	else if(recipFromQueueB > 0)
	{
		recipFromQueueB--;
		numberEvent = qRecipientB->getRecipient(1)->number_wsk;
		actualGroup = true;
	}
	else if (events->getFirstEvent() && events->getFirstEvent()->getTime() ==
bloodPoint->getClock())
	{
		numberEvent = events->getFirstEvent()->number;
		events->removeEvent();
	}
	phase = wsk_tab[numberEvent]->phase;
	while (active)
	{
		switch (phase)
		{
		case 0:
		{
			cout << "'Recipient Process': Obsluga biorcy 1.etap" << endl;
			nRecipient = new Recipient(neededBloodGeometric_0_23);
			if(nRecipient->showGruop())
				qRecipientB->addRecipient(nRecipient);
			else
				qRecipientA->addRecipient(nRecipient);
			actualGroup = nRecipient->showGruop();
			wsk_tab[numberEvent]->phase = 1;
			cout << "- dodanie pacjenata do koleji pacjentow" << endl;
			if (queueBlood->howMany() < nRecipient->getNeededBlood() &&
!nRecipient->showGruop())
			{
				active = false;
				if (!bloodPoint->checkSpecialOrderFlagA())
				{
					specialOrderA-
>activate(timeOfSpecialOrderNormal_400);
					specialOrderID++;
					cout << "- zaplanowanie przyjscia specjalnego
zamowienia" << endl;
				}
				int num = numberEvent;
				activate(timeOfRecipientExponential_200);
				numberEvent = num;
				wsk_tab[numberEvent]->planRecipient = true;
				nRecipient->number_wsk = numberEvent;
				cout << "- zaplanowanie kolejnego procesu 'Recipient
process' " << endl;
			}
			else if (queueBloodB->howMany() < nRecipient->getNeededBlood() &&
nRecipient->showGruop())
			{
				active = false;
				if (!bloodPoint->checkSpecialOrderFlagB())
				{
					specialOrderB-
>activate(timeOfSpecialOrderNormal_400);
					specialOrderID++;
```

```
                                cout << "- zaplanowanie przyjscia specjalnego
zamowienia" << endl;
                        }
                        int num = numberEvent;
                        activate(timeOfRecipientExponential_200);
                        numberEvent = num;
                        wsk_tab[numberEvent]->planRecipient = true;
                        nRecipient->number_wsk = numberEvent;
                        cout << "- zaplanowanie kolejnego procesu 'Recipient
process' " << endl;
                }
                phase = 1;
                break;
        }
        case 1:
        {
                cout << "'Recipient process': Obsluga biorcy 2.etap" << endl;
                if (wsk_tab[numberEvent]->planRecipient == true)
                {
                        if (!actualGroup)
                                nRecipient = qRecipientA->getFirstRecipient();
                        else
                                nRecipient = qRecipientB->getFirstRecipient();
                }
                if(!nRecipient->showGruop())
                {
                        for (int i = 0; i < nRecipient->getNeededBlood(); i++)
                        {
                                queueBlood->removeBlood();
                                bloodPacientStatistics++;
                        }
                        qRecipientA->removeSelectedRecipient(nRecipient);
                }
                else
                {
                        for (int i = 0; i < nRecipient->getNeededBlood(); i++)
                        {
                                queueBloodB->removeBlood();
                                bloodPacientStatistics++;
                        }
                        qRecipientB->removeSelectedRecipient(nRecipient);
                }
                cout << "- usuniecie pacjenta z koleji pacjentow" << endl;
                cout << "- usuniecie krwi z kolejki krwi" << endl;
                if (wsk_tab[numberEvent]->planRecipient == false)
                {
                        int num = numberEvent;
                        activate(timeOfRecipientExponential_200);
                        numberEvent = num;
                        wsk_tab[numberEvent]->planRecipient = true;
                        cout << "- zaplanowanie kolejnego procesu 'Recipient
process'" << endl;
                }
                active = false;
                wsk_tab[numberEvent]->planRecipient = false;
                wsk_tab[numberEvent]->phase = 3;
                if (actualGroup)
                        scResearchProcessB->checkCondition(actualGroup);
                else
                        scResearchProcessA->checkCondition(actualGroup);
                break;
        }
```

```
			}
		}
		if (bloodPoint->getAmountBloodA() < bloodPoint->getMinAMountBlood() &&
!bloodPoint->checkOrderFlagA())
		{
			orderA->activate(timeOfOrderExponential_1900,false);
			bloodPoint->setOrderFlagA();
			cout << "- zlozenie standardowego zamowienia gr.A" << endl;
		}
		if (bloodPoint->getAmountBloodB() < bloodPoint->getMinAMountBlood() &&
!bloodPoint->checkOrderFlagB())
		{
			orderB->activate(timeOfOrderExponential_1900,true);
			bloodPoint->setOrderFlagB();
			cout << "- zlozenie standardowego zamowienia gr.B" << endl;
		}
}

void RecipientProcess::activate(int time)
{
		bool condition = true;
		int i = 0;
		while (condition)
		{
			if (wsk_tab[i]->phase == 3)
			{
				numberEvent = i;
				condition = false;
			}
			else
			{
				i++;
			}
		}
		wsk_tab[numberEvent]->setTime(bloodPoint->getClock() + time);
		wsk_tab2[numberEvent]->setTime(bloodPoint->getClock() + time);
		wsk_tab[numberEvent]->phase = 0;
		wsk_tab[numberEvent]->planRecipient = false;
		events->addEvent(wsk_tab2[numberEvent]);
		wsk_tab[numberEvent]->number = numberEvent;
		wsk_tab2[numberEvent]->number = numberEvent;
		eList->addEvent(wsk_tab[numberEvent]);
		eList->sortEvent();
}

void RecipientProcess::reactive(bool group)
{
		bool make = true;
		int recip = 1;
		int sumNeeded = 0;
		if(!group)
		{
			while (make)
			{
				make = false;
				if (qRecipientA->howManyRecipients() > 0)
				{
					sumNeeded += qRecipientA->getRecipient(recip)-
>getNeededBlood();
					if (qRecipientA->howManyRecipients() >= recip && queueBlood-
>howMany() >= sumNeeded)
					{
```

```
                                        if (recip == qRecipientA->howManyRecipients())
                                                make = false;
                                        else
                                                make = true;
                                        int nr = qRecipientA->getRecipient(recip)-
>number_wsk;
                                        cout << "- reaktywacja procecu 'RecipientProcess'" <<
endl;
                                        wsk_tab[nr]->setTime(bloodPoint->getClock());
                                        eList->addToBegining(wsk_tab[nr]);
                                        eList->sortEvent();
                                        recipFromQueueA = recip;
                                        recip++;
                                }
                        }

                }
        }
        else
        {
                while (make)
                {
                        make = false;
                        if (qRecipientB->howManyRecipients() > 0)
                        {
                                sumNeeded += qRecipientB->getRecipient(recip)-
>getNeededBlood();
                                if (qRecipientB->howManyRecipients() >= recip && queueBlood-
>howMany() >= sumNeeded)
                                {
                                        if (recip == qRecipientB->howManyRecipients())
                                                make = false;
                                        else
                                                make = true;
                                        int nr = qRecipientB->getRecipient(recip)-
>number_wsk;
                                        cout << "- reaktywacja procecu 'RecipientProcess'" <<
endl;
                                        wsk_tab[nr]->setTime(bloodPoint->getClock());
                                        eList->addToBegining(wsk_tab[nr]);
                                        eList->sortEvent();
                                        recipFromQueueB = recip;
                                        recip++;
                                }
                        }

                }
        }

}

void RecipientProcess::setOrderA(OrderProcess *ord)
{
        orderA = ord;
}

void RecipientProcess::setSpecialOrderA(OrderSpecialProcess *spec)
{
        specialOrderA = spec;
}

void RecipientProcess::setOrderB(OrderProcess *ord)
```

```
{
	orderB = ord;
}

void RecipientProcess::setSpecialOrderB(OrderSpecialProcess *spec)
{
	specialOrderB = spec;
}

int RecipientProcess::getBloodPacientStatistics()
{
	return bloodPacientStatistics;
}

int RecipientProcess::getSpecialOrderID()
{
	return specialOrderID;
}

void RecipientProcess::clearSpecialOrderID()
{
	specialOrderID = 0;
}

void RecipientProcess::clearBloodPacientStatistics()
{
	bloodPacientStatistics = 0;
}

void RecipientProcess::setTimeOfOrderExponential_1900(int number)
{
	timeOfOrderExponential_1900 = number;
}
void RecipientProcess::setTimeOfRecipientExponential_200(int number)
{
	timeOfRecipientExponential_200 = number;
}
void RecipientProcess::setTimeOfSpecialOrderNormal_400(int number)
{
	timeOfSpecialOrderNormal_400 = number;
}
void RecipientProcess::setNeededBloodGeometric_0_23(int number)
{
	neededBloodGeometric_0_23 = number;
}

RecipientProcess::RecipientProcess(BloodDonationPoint *bPoint, EventList *el,
QueueUnitBlood *qBloodA, QueueUnitBlood *qBloodB,
	QueueRecipient *qReciA, QueueRecipient *qReciB, ScientificResearchProcess
*scProcA, ScientificResearchProcess *scProcB)
	:Process(bPoint, el, qBloodA), queueBloodB(qBloodB), qRecipientA(qReciA),
qRecipientB(qReciB), scResearchProcessA(scProcA),
	scResearchProcessB(scProcB)
{
	numberEvent = 0;
	recipFromQueueA = 0;
	recipFromQueueB = 0;
	bloodPacientStatistics = 0;
	specialOrderID = 0;
	wsk_tab = new Event*[100];
	for (int i = 0; i < 100; i++)
	{
```

```
			wsk_tab[i] = new Event(this);
		}
		wsk_tab2 = new Event*[100];
		for (int i = 0; i < 100; i++)
		{
			wsk_tab2[i] = new Event(this);
		}
		events = new EventList();
}


RecipientProcess::~RecipientProcess()
{
}
```

ScientificResearchProcess.h

```
#pragma once
#include "Process.h"
#include "BloodDonationPoint.h"

class ScientificResearchProcess :
		public Process
{
public:
		void execute() override;
		void activate(int time);
		void checkCondition(bool);
		void setAmount(int);
		void setAmountOfBloodUniform_5_10(int);
		int getAmountOfBloodStatistic();
		void clearAmountOfBloodStatistic();
		ScientificResearchProcess(BloodDonationPoint *bPoint, EventList *el,
QueueUnitBlood *qBlood);
		~ScientificResearchProcess();
private:
		Event *myEvent;
		int timeOfStart;
		int amountOfBloodUniform;
		int amountOfBloodStatistic;
};
```

ScientificResearchProcess.cpp

```
#include "pch.h"
#include "ScientificResearchProcess.h"
#include <iostream>

using namespace std;

void ScientificResearchProcess::execute()
{
		cout << "'ScientificResearcheProcess': Badania naukowe" << endl;
		if (timeOfStart + 300 == bloodPoint->getClock())
		{
			for (int i = 0; i < amountOfBloodUniform; i++)
			{
				queueBlood->removeBlood();
				amountOfBloodStatistic++;
```

```
		}
		cout << "- wyslano " << amountOfBloodUniform << " jednostek krwi" << endl;
	}
	else
	{
		cout << "- nie wyslano jednostek krwi" << endl;
	}
	timeOfStart = 0;
}

void ScientificResearchProcess::activate(int time)
{
	cout << "'ScientificResearcheProcess': Badania naukowe" << endl;
	timeOfStart = bloodPoint->getClock();
	myEvent->setTime(bloodPoint->getClock() + time);
	eList->addEvent(myEvent);
	cout << "- zaplanowanie wyslania jednostek krwi" << endl;
	eList->sortEvent();
}

void ScientificResearchProcess::checkCondition(bool group)
{
	if(group)
	{
		if (bloodPoint->getAmountBloodB() < 30)
		{
			eList->removeSelectedEvent(myEvent);
			timeOfStart = 0;
			cout << "'ScientificResearcheProcess': Badania naukowe gr.B" <<
endl;
			cout << "- anulowanie wyslania jednostek krwi na skutek nie
spelnienie warunku" << endl;
		}

		else if (timeOfStart == 0)
		{
			activate(300);
		}
	}
	else
	{
		if (bloodPoint->getAmountBloodA() < 30)
		{
			eList->removeSelectedEvent(myEvent);
			timeOfStart = 0;
			cout << "'ScientificResearcheProcess': Badania naukowe gr.A" <<
endl;
			cout << "- anulowanie wyslania jednostek krwi na skutek nie
spelnienie warunku" << endl;
		}

		else if (timeOfStart == 0)
		{
			activate(300);
		}
	}


}
void ScientificResearchProcess::setAmountOfBloodUniform_5_10(int number)
{
	amountOfBloodUniform = number;
```

```
}

int ScientificResearchProcess::getAmountOfBloodStatistic()
{
	return amountOfBloodStatistic;
}

void ScientificResearchProcess::clearAmountOfBloodStatistic()
{
	amountOfBloodStatistic = 0;
}

ScientificResearchProcess::ScientificResearchProcess(BloodDonationPoint *bPoint,
EventList *el, QueueUnitBlood *qBlood)
	: Process(bPoint, el, qBlood)

{
	amountOfBloodUniform = 7;
	timeOfStart = 0;
	myEvent = new Event(this);
	amountOfBloodStatistic = 0;
}


ScientificResearchProcess::~ScientificResearchProcess()
{
}
```

Statistics.h

```
#pragma once
#include "BloodDonationPoint.h"
#include "DonorProcess.h"
#include "OrderSpecialProcess.h"
#include <string>

class RecipientProces;
class ScientificResearchProcess;
class OrderProcess;
class DonorProcess;

class Statistics
{
public:
	void viewStat();
	void printToFile();
	void saveStatsToFile(double data, std::string name);
	void saveStatsToFile2(double data, int time, std::string name);
	Statistics(BloodDonationPoint *bPoint, DonorProcess *dProc, OrderProcess
*orderPA, OrderProcess *orderPB, QueueRecipient *recQA, QueueRecipient *recQB,
		QueueUnitBlood *bQA, QueueUnitBlood *bQB, ScientificResearchProcess *scPA,
		ScientificResearchProcess *scPB, UtilizationProcess *utilizationDonorA,
UtilizationProcess *utilizationOrderA,
		UtilizationProcess *utilizationSpecialOrderA, UtilizationProcess
*utilizationDonorB, UtilizationProcess *utilizationOrderB,
		UtilizationProcess *utilizationSpecialOrderB, RecipientProcess *recPr);
	Statistics();
	~Statistics();
private:
	BloodDonationPoint *bloodPoint;
	DonorProcess *dProcess;
```

```
	OrderProcess *orderProcA;
	OrderProcess *orderProcB;
	QueueRecipient *recipientQA;
	QueueRecipient *recipientQB;
	QueueUnitBlood *bloodQA;
	QueueUnitBlood *bloodQB;
	ScientificResearchProcess *scProcA;
	ScientificResearchProcess *scProcB;
	UtilizationProcess *utilizationDonorProcA;
	UtilizationProcess *utilizationOrderProcA;
	UtilizationProcess *utilizationSpecialOrderProcA;
	UtilizationProcess *utilizationDonorProcB;
	UtilizationProcess *utilizationOrderProcB;
	UtilizationProcess *utilizationSpecialOrderProcB;
	RecipientProcess *recProcess;
};
```

Statistics.cpp

```
#include "pch.h"
#include "Statistics.h"
#include <iostream>
#include <fstream>

using namespace std;

void Statistics::viewStat()
{
	cout << "Statystyki:" << endl;
	cout << "N - nowych zamowienien (standardowe zamowienie) = " << orderProcA-
>getAmountOfOrder() << endl;
	cout << "R - prog zamawiania = " << bloodPoint->getMinAMountBlood() << endl <<
endl;
	cout << "Krew przechodzaca przez punkt = " << bloodQA-
>getAmoutnOfBloodStatistics() + bloodQB->getAmoutnOfBloodStatistics() << endl;
	cout << "Krew przechodzaca przez punkt gr.A = " << bloodQA-
>getAmoutnOfBloodStatistics()<< endl;
	cout << "Krew przechodzaca przez punkt gr.B = " << bloodQB-
>getAmoutnOfBloodStatistics() << endl;
	cout << "Krew przechodzaca przez punkt gr.A % = " << bloodQA-
>getAmoutnOfBloodStatistics()*100/(bloodQA->getAmoutnOfBloodStatistics()
		+ bloodQB->getAmoutnOfBloodStatistics()) << endl;
	cout << "Krew przechodzaca przez punkt gr.B % = " << bloodQB-
>getAmoutnOfBloodStatistics()*100 / (bloodQA->getAmoutnOfBloodStatistics()
		+ bloodQB->getAmoutnOfBloodStatistics()) << endl;
	cout << "Zuzyta krew przez pacjentow = " << recProcess-
>getBloodPacientStatistics() << endl;
	cout << "Krew wyslana na badania naukowe = " << scProcA-
>getAmountOfBloodStatistic() + scProcB->getAmountOfBloodStatistic() << endl;
	cout << "Krew wyslana na badania naukowe gr.A = " << scProcA-
>getAmountOfBloodStatistic() << endl;
	cout << "Krew wyslana na badania naukowe gr.B= " << scProcB-
>getAmountOfBloodStatistic() << endl;
	cout << "Srednia liczba jednostek krwi w kolejce = " << ((double)bloodQA-
>getAmoutnOfBloodStatistics() + bloodQA->getAmoutnOfBloodStatistics()) /
		bloodPoint->getAmountOfIteracions() << endl;
	int utilizeBlood = utilizationDonorProcA->getAmountOfUtilizeBloodStatistics() +
		utilizationOrderProcA->getAmountOfUtilizeBloodStatistics() +
utilizationSpecialOrderProcA->getAmountOfUtilizeBloodStatistics()
		+ utilizationDonorProcB->getAmountOfUtilizeBloodStatistics() +
		utilizationOrderProcB->getAmountOfUtilizeBloodStatistics() +
utilizationSpecialOrderProcB->getAmountOfUtilizeBloodStatistics();
	cout << "Zutylizowana krew = " << utilizeBlood << endl;
```

```
	cout << "Zutylizowana krew z normalnego zamowienia = " << utilizationOrderProcA-
>getAmountOfUtilizeBloodStatistics() +
		utilizationOrderProcB->getAmountOfUtilizeBloodStatistics() << endl;
	cout << "Zutylizowana krew ze specjalneg zamowienia = " <<
utilizationSpecialOrderProcA->getAmountOfUtilizeBloodStatistics() +
		utilizationSpecialOrderProcB->getAmountOfUtilizeBloodStatistics() << endl;
	cout << "Zutylizowana krew od dawcow = " << utilizationDonorProcA-
>getAmountOfUtilizeBloodStatistics() +
		utilizationDonorProcB->getAmountOfUtilizeBloodStatistics() << endl;
	cout << "Procent zutylizowanej krwi = " << (double)utilizeBlood * 100 / (bloodQA-
>getAmoutnOfBloodStatistics() +
		bloodQB->getAmoutnOfBloodStatistics()) << "%" << endl;
	cout << "Procent zutylizowanej krwi z normalnego zamowienia = " <<
((double)utilizationOrderProcA->getAmountOfUtilizeBloodStatistics()
		+ utilizationOrderProcB->getAmountOfUtilizeBloodStatistics()) * 100 /
(bloodQA->getAmoutnOfBloodStatistics() +
			bloodQB->getAmoutnOfBloodStatistics()) << "%" << endl;
	cout << "Procent zutylizowanej krwi ze specjalnego zamowienia = " <<
((double)utilizationSpecialOrderProcA->getAmountOfUtilizeBloodStatistics()
		+ utilizationSpecialOrderProcB->getAmountOfUtilizeBloodStatistics()) * 100
/ (bloodQA->getAmoutnOfBloodStatistics() +
			bloodQB->getAmoutnOfBloodStatistics()) << "%" << endl;
	cout << "Procent zutylizowanej krwi od dawcow = " <<
((double)utilizationDonorProcA->getAmountOfUtilizeBloodStatistics()
		+ utilizationDonorProcA->getAmountOfUtilizeBloodStatistics()) * 100 /
(bloodQA->getAmoutnOfBloodStatistics() +
			bloodQB->getAmoutnOfBloodStatistics()) << "%" << endl;
	cout << "Pacjentow przechodzacych przez punkt = " << recipientQA-
>getAmountOfPatientStatistics() +
		recipientQB->getAmountOfPatientStatistics() << endl;
	cout << "Srednia liczba pacjentow w kolejce = " << ((double)recipientQA-
>getAmountOfPatientStatistics() +
		recipientQB->getAmountOfPatientStatistics()) / bloodPoint-
>getAmountOfIteracions() << endl;
	cout << "Liczba normalnych zamowien = " << orderProcA->getOrderID() + orderProcB-
>getOrderID()<< endl;
	cout << "Liczba specialnych zamowien = " << recProcess->getSpecialOrderID() <<
endl;
	cout << "Prawdopodobienstwo wystapienia awaryjnego zamowienia = " <<
(double)recProcess->getSpecialOrderID() * 100 /
		bloodPoint->getAmountOfIteracions() << "%" << endl;
	cout << endl << endl;
}

void Statistics::printToFile()
{
	ofstream stat;
	stat.open("satistics.txt", ios::out | ios::app);
	stat << endl;
	stat << "Statystyki:" << endl;
	stat << "N - nowych zamowienien (standardowe zamowienie) = " << orderProcA-
>getAmountOfOrder() << endl;
	stat << "R - prog zamawiania = " << bloodPoint->getMinAMountBlood() << endl <<
endl;
	stat << "Krew przechodzaca przez punkt = " << bloodQA-
>getAmoutnOfBloodStatistics() + bloodQB->getAmoutnOfBloodStatistics() << endl;
	stat << "Krew przechodzaca przez punkt gr.A = " << bloodQA-
>getAmoutnOfBloodStatistics() << endl;
	stat << "Krew przechodzaca przez punkt gr.B = " << bloodQB-
>getAmoutnOfBloodStatistics() << endl;
	stat << "Krew przechodzaca przez punkt gr.A % = " << bloodQA-
>getAmoutnOfBloodStatistics() * 100 / (bloodQA->getAmoutnOfBloodStatistics()
```

```
            + bloodQB->getAmoutnOfBloodStatistics()) << endl;
      stat << "Krew przechodzaca przez punkt gr.B % = " << bloodQB-
>getAmoutnOfBloodStatistics() * 100 / (bloodQA->getAmoutnOfBloodStatistics()
            + bloodQB->getAmoutnOfBloodStatistics()) << endl;
      stat << "Zuzyta krew przez pacjentow = " << recProcess-
>getBloodPacientStatistics() << endl;
      stat << "Krew wyslana na badania naukowe = " << scProcA-
>getAmountOfBloodStatistic() + scProcB->getAmountOfBloodStatistic() << endl;
      stat << "Krew wyslana na badania naukowe gr.A = " << scProcA-
>getAmountOfBloodStatistic() << endl;
      stat << "Krew wyslana na badania naukowe gr.B= " << scProcB-
>getAmountOfBloodStatistic() << endl;
      stat << "Srednia liczba jednostek krwi w kolejce = " << ((double)bloodQA-
>getAmoutnOfBloodStatistics() + bloodQA->getAmoutnOfBloodStatistics()) /
            bloodPoint->getAmountOfIteracions() << endl;
      int utilizeBlood = utilizationDonorProcA->getAmountOfUtilizeBloodStatistics() +
            utilizationOrderProcA->getAmountOfUtilizeBloodStatistics() +
utilizationSpecialOrderProcA->getAmountOfUtilizeBloodStatistics()
            + utilizationDonorProcB->getAmountOfUtilizeBloodStatistics() +
            utilizationOrderProcB->getAmountOfUtilizeBloodStatistics() +
utilizationSpecialOrderProcB->getAmountOfUtilizeBloodStatistics();
      stat << "Zutylizowana krew = " << utilizeBlood << endl;
      stat << "Zutylizowana krew z normalnego zamowienia = " << utilizationOrderProcA-
>getAmountOfUtilizeBloodStatistics() +
            utilizationOrderProcB->getAmountOfUtilizeBloodStatistics() << endl;
      stat << "Zutylizowana krew ze specjalneg zamowienia = " <<
utilizationSpecialOrderProcA->getAmountOfUtilizeBloodStatistics() +
            utilizationSpecialOrderProcB->getAmountOfUtilizeBloodStatistics() << endl;
      stat << "Zutylizowana krew od dawcow = " << utilizationDonorProcA-
>getAmountOfUtilizeBloodStatistics() +
            utilizationDonorProcB->getAmountOfUtilizeBloodStatistics() << endl;
      stat << "Procent zutylizowanej krwi = " << (double)utilizeBlood * 100 / (bloodQA-
>getAmoutnOfBloodStatistics() +
            bloodQB->getAmoutnOfBloodStatistics()) << "%" << endl;
      stat << "Procent zutylizowanej krwi z normalnego zamowienia = " <<
((double)utilizationOrderProcA->getAmountOfUtilizeBloodStatistics()
            + utilizationOrderProcB->getAmountOfUtilizeBloodStatistics()) * 100 /
(bloodQA->getAmoutnOfBloodStatistics() +
                  bloodQB->getAmoutnOfBloodStatistics()) << "%" << endl;
      stat << "Procent zutylizowanej krwi ze specjalnego zamowienia = " <<
((double)utilizationSpecialOrderProcA->getAmountOfUtilizeBloodStatistics()
            + utilizationSpecialOrderProcB->getAmountOfUtilizeBloodStatistics()) * 100
/ (bloodQA->getAmoutnOfBloodStatistics() +
                  bloodQB->getAmoutnOfBloodStatistics()) << "%" << endl;
      stat << "Procent zutylizowanej krwi od dawcow = " <<
((double)utilizationDonorProcA->getAmountOfUtilizeBloodStatistics()
            + utilizationDonorProcA->getAmountOfUtilizeBloodStatistics()) * 100 /
(bloodQA->getAmoutnOfBloodStatistics() +
                  bloodQB->getAmoutnOfBloodStatistics()) << "%" << endl;
      stat << "Pacjentow przechodzacych przez punkt = " << recipientQA-
>getAmountOfPatientStatistics() +
            recipientQB->getAmountOfPatientStatistics() << endl;
      stat << "Srednia liczba pacjentow w kolejce = " << ((double)recipientQA-
>getAmountOfPatientStatistics() +
            recipientQB->getAmountOfPatientStatistics()) / bloodPoint-
>getAmountOfIteracions() << endl;
      stat << "Liczba normalnych zamowien = " << orderProcA->getOrderID() + orderProcB-
>getOrderID() << endl;
      stat << "Liczba specialnych zamowien = " << recProcess->getSpecialOrderID() <<
endl;
      stat << "Prawdopodobienstwo wystapienia awaryjnego zamowienia = " <<
(double)recProcess->getSpecialOrderID() * 100 /
```

```
			bloodPoint->getAmountOfIteracions() << "%" << endl;
	stat << endl << endl;
	stat.close();
}

void Statistics::saveStatsToFile(double data, string name)
{
	ofstream result;
	result.open(name, ios::out | ios::app);
	result << data << endl;
	result.close();
}

void Statistics::saveStatsToFile2(double data, int time, string name)
{
	ofstream result;
	result.open(name, ios::out | ios::app);
	result << data << "    " << time<< endl;
	result.close();
}

Statistics::Statistics(BloodDonationPoint *bPoint, DonorProcess *dProc, OrderProcess
*orderPA, OrderProcess *orderPB,
	QueueRecipient *recQA, QueueRecipient *recQB,QueueUnitBlood *bQA, QueueUnitBlood
*bQB, ScientificResearchProcess *scPA,
	ScientificResearchProcess *scPB, UtilizationProcess *utilizationDonorA,
UtilizationProcess *utilizationOrderA,
	UtilizationProcess *utilizationSpecialOrderA, UtilizationProcess
*utilizationDonorB, UtilizationProcess *utilizationOrderB,
	UtilizationProcess *utilizationSpecialOrderB, RecipientProcess *recPr)
	:bloodPoint(bPoint), dProcess(dProc), orderProcA(orderPA), orderProcB(orderPB),
	recipientQA(recQA), recipientQB(recQB), bloodQA(bQA), bloodQB(bQB),
scProcA(scPA), scProcB(scPB), utilizationDonorProcA(utilizationDonorA),
	utilizationDonorProcB(utilizationDonorB),
utilizationOrderProcA(utilizationOrderA),utilizationOrderProcB(utilizationOrderB),
	utilizationSpecialOrderProcA(utilizationSpecialOrderA),
utilizationSpecialOrderProcB(utilizationSpecialOrderB), recProcess(recPr)
{
}

Statistics::Statistics()
{
}


Statistics::~Statistics()
{
}
```

UnitBlood.h

```
#pragma once
#include "Generators.h"

class UnitBlood
{
public:
	UnitBlood *next;
	int getTimeOfBlood();
	bool showGroup();
	UnitBlood(int tToLive);
	UnitBlood(int tToLive,bool gr);
```

```
	~UnitBlood();
private:
	int timeToLive;
	bool group;
	Generators gen;
};
```

UnitBlood.cpp

```
#include "pch.h"
#include "UnitBlood.h"

int UnitBlood::getTimeOfBlood()
{
	return timeToLive;
}

bool UnitBlood::showGroup()
{
	return group;
}


UnitBlood::UnitBlood(int tToLive)
{
	next = nullptr;
	timeToLive = tToLive;
	if (gen.uniform() > 0.6)
		group = true;
	else
		group = false;
}

UnitBlood::UnitBlood(int tToLive, bool gr)
{
	next = nullptr;
	timeToLive = tToLive;
	group = gr;
}


UnitBlood::~UnitBlood()
{
}
```

UtilizationProcess.h

```
#pragma once
#include "Process.h"

class UtilizationProcess :
	public Process
{
public:
	void execute() override;
	void setHowManyToRemove(int);
	int getHowManyToRemove();
	int getAmountOfUtilizeBloodStatistics();
	void clearAmountOfUtilizeBloodStatistics();
	void activate(int time);
	UtilizationProcess(BloodDonationPoint *bPoint, EventList *el, QueueUnitBlood
*qBlood);
	~UtilizationProcess();
```

```
private:
	int howManyToRemove;
	int amountOfUtilizeBloodStatistics;
	Event *myEvent;
};
```

UtilizationProcess.cpp

```
#include "pch.h"
#include "UtilizationProcess.h"
#include <iostream>

using namespace std;


void UtilizationProcess::execute()
{
	bool utilization = false;
	int howMany = 0;
	for (int i = 0; i < howManyToRemove; i++)
	{
		if (queueBlood->showTimeBlood() == bloodPoint->getClock())
		{
			queueBlood->removeBlood();
			amountOfUtilizeBloodStatistics++;
			utilization = true;
			howMany++;
		}
	}
	cout << "'UtilizationProcess': Utylizacja jednostek krwi" << endl;
	if (utilization)
	{
		cout << "- utylizacja " << howMany << " jednostek krwi z " <<
howManyToRemove << " planowanych" << endl;
	}
	else
	{
		cout << "- utylizacja nie odbyla sie ,poniewaz dane jednostki krwi" <<
endl;
		cout << "  zostaly wykorzystane wczesniej" << endl;
	}
}

void UtilizationProcess::activate(int time)
{
	if (eList->lookForEvent(myEvent))
	{
		Event *secondEvent = new Event(this);
		secondEvent->setTime(bloodPoint->getClock() + time);
		eList->addEvent(secondEvent);
		eList->sortEvent();
	}
	else
	{
		myEvent->setTime(bloodPoint->getClock() + time);
		eList->addEvent(myEvent);
		eList->sortEvent();
	}
}

void UtilizationProcess::setHowManyToRemove(int amount)
{
	howManyToRemove = amount;
```

```
}

int UtilizationProcess::getHowManyToRemove()
{
	return howManyToRemove;
}

int UtilizationProcess::getAmountOfUtilizeBloodStatistics()
{
	return amountOfUtilizeBloodStatistics;
}

void UtilizationProcess::clearAmountOfUtilizeBloodStatistics()
{
	amountOfUtilizeBloodStatistics = 0;
}

UtilizationProcess::UtilizationProcess(BloodDonationPoint *bPoint, EventList *el,
QueueUnitBlood *qBlood)
	: Process(bPoint, el, qBlood)
{
	howManyToRemove = 0;
	amountOfUtilizeBloodStatistics = 0;
	myEvent = new Event(this);
}


UtilizationProcess::~UtilizationProcess()
{
}
```