WYDZIAŁ MATEMATYCZNO-PRZYRODNICZY, SZKOŁA NAUK ŚCISŁYCH

**UNIWERSYTET KARDYNAŁA STEFANA WYSZYŃSKIEGO
W WARSZAWIE**

Symulator systemów kolejkowych

Autorzy:
Krzysztof Jura
Michał Gawroński

Warszawa 2013

Spis treści

1. Wstęp

Opisywana w niniejszym opracowaniu aplikacja jest symulatorem systemów kolejkowych.

Systemy kolejkowe są i działają wokół nas od dawna. Mają zastosowanie w wielu dziedzinach życia, można je spotkać choćby w sklepach, urzędach, przychodniach, a także w dziedzinie logistyki czy transportu. Podstawowym ich przykładem stała się centrala telefoniczna będąca realizacją zasadniczych założeń dotyczących strumienia zgłoszeń, funkcji rozkładu czasów obsługi oraz źródłem wielu terminów i definicji [1].

W opracowaniu zawarty jest wstęp teoretyczny wraz z opisem pojęć dotyczących teorii kolejek, wyjaśnieniem zasad klasyfikacji systemów kolejkowych oraz ich charakterystykami. Następnie przedstawiona została aplikacja będąca symulacyjną metodą analizy systemów. Niektóre wyniki jej działania są porównane do podobnych obliczeń metodami analitycznymi.

Aplikacja posiada dwa tryby pracy: edukacyjny i naukowy. W pierwszym skupiono się na graficznym zwizualizowaniu zdarzeń zachodzących w małym systemie kolejkowym w czasie rzeczywistym, natomiast w drugim nacisk położony jest na obliczenia wartości charakterystycznych systemu ze znacząco większą ilością zgłoszeń i kanałów obsługi oraz na weryfikacji ich wyników. W obu trybach możliwe jest wstępne określenie przez użytkownika parametrów symulacji przed jej rozpoczęciem.

Program została napisany obiektowo w języku C++ [2] przy użyciu środowiska programistycznego *Microsoft Visual C++ 2008 i 2010.* Do rysowania grafiki użyta została biblioteka *OpenGL* [3].

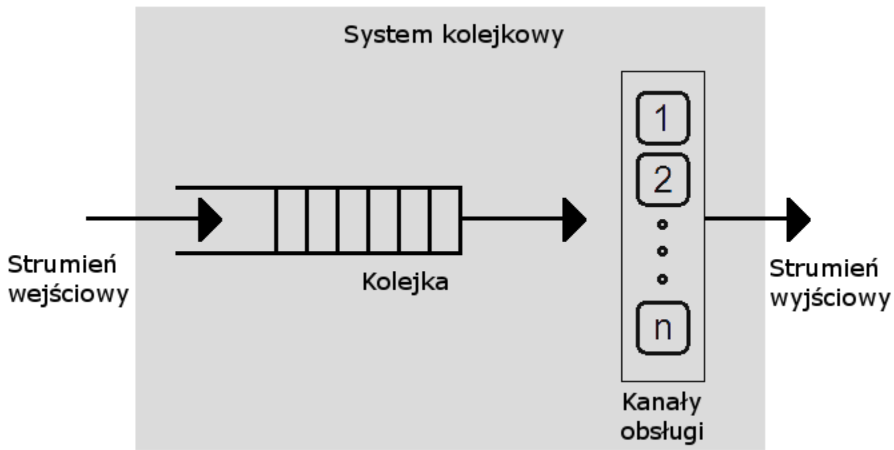
2. Opis teoretyczny

Teoria kolejek, nazywana również *teorią masowej* obsługi, jest dziedziną matematyki, zajmującą się analizowaniem systemów, w których powstają kolejki. W takich systemach zgłoszenia napływają do punktów obsługi i czekają na obsłużenie w poczekalni.

2.1 Pojęcia ogólne

Zgłoszenie	- żądanie wykonania pewnych czynności na rzecz zgłaszającego (np. klienta, abonenta, pacjenta)
Obsługa	- spełnienie żądań danego zgłoszenia w kanale obsługi
System obsługi	- zbiór wszystkich kanałów obsługi
Strumień zdarzeń	- ciąg losowych zdarzeń związanych z przybywaniem nowych zgłoszeń lub kończeniem obsługi zgłoszeń

W ogólnej postaci każdy system kolejkowy można w uproszczeniu przedstawić w poniższy sposób:



Rys. 2.1. Schemat systemu kolejkowego

Proces obsługi jednego zgłoszenia ze względu na ilość stanowisk potrzebnych do realizacji jednego zgłoszenia może być: **jednofazowy** lub **wielofazowy**.

Organizacja systemu może być **uporządkowana** lub **nieuporządkowana**. Zależy to od numeracji kanałów. Uporządkowany system zawsze przydziela zgłoszenia do kanałów w tej samej kolejności.

Ze względu na liczbę kanałów systemy mogą być **jednokanałowe** lub **wielokanałowe**.

Ze względu na zachowanie się zgłoszenia, które w momencie nadejścia natrafia na całkowicie zajęte kanały obsługi, wyróżniamy trzy typy systemów:
ze stratami (bez kolejki) – zgłoszenie nie może czekać na obsługę, gdy wszystkie kanały są zajęte, opuszcza system nieobsłużone;

bez strat (nieograniczona kolejka) – zgłoszenie musi być obsłużone i będzie czekać na to aż do skutku;
mieszane – pośrednie warunki takie jak: ograniczony czas oczekiwania w kolejce (lub łącznie w systemie), ograniczona długość kolejki

Ze względu na rozmiar rozróżniamy systemy z **ograniczonym** lub **nieograniczonym** rozmiarem. Rozmiar to suma miejsc w kolejce i wszystkich kanałów.

Ze względu na ilość źródeł zgłoszeń systemy dzielimy na **otwarte** i **zamknięte.** Pierwsze z nich mogą mieć nieskończoną ilość przychodzących zgłoszeń, drugie zaś obsługują stale tę samą, ograniczoną grupę klientów, np. konserwacja i naprawa urządzeń w parku maszyn lub pojazdów z floty jednego przewoźnika.

Podstawową wielkością charakteryzującą systemy kolejkowe jest **nasycenie** systemu kolejkowego, które można opisać za pomocą kilku charakterystyk [1]:

- statystycznego opisu **strumienia zgłoszeń** (będącego zazwyczaj funkcją rokładu odstępów czasu pomiędzy kolejnymi zgłoszeniami),
- **procesu obsługi** (funkcja rozkładu czasów obsługi i krotność systemu) oraz
- **regulaminu kolejki** opisującego kolejność wybierania zgłoszeń do obsługi.

Gdy strumień zgłoszeń ma stały interwał pomiędzy zgłoszeniami, mówimy że jest **deterministyczny**. W przeciwnym wypadku jest **niedeterministyczny** tj. ma losowy interwał.

Kolejną właściwością strumienia jest **stacjonarność**. Jest to zależność od czasu (np. pory dnia) lub jej brak.

2.2 Zasady klasyfikacji systemów kolejkowych

Systemy kolejkowe można uporządkować wg różnych cech klasyfikacyjnych. Ogólnie przyjętą metodą (także w tej pracy) jest prosty kod zawierający większość najważniejszych informacji określających system wymyślony przez D. Kendalla i rozszerzony przez A. M. Lee. Zapis ww. symboliki wygląda następująco:

$$X/Y/m/d/l/z$$

gdzie:
X - symbol rozkładu wejściowego strumienia (M – wykładniczy, D – regularny)
Y - symbol rozkładu wyjściowego strumienia
m - liczba kanałów obsługi
d - kod dyscypliny kolejki (FIFO – First In First Out, LIFO – Last In First Out)
l - rozmiar systemu
z - zamkniętość: (O – otwarty, F – zamknięty)

Wykładniczy rozkład prawdopodobieństwa długości odstępu czasu między kolejnymi zgłoszeniami do systemu nazywamy strumieniem Poissona [1].

2.3 Podstawowe modele systemów kolejkowych

2.3.1 System kolejkowy ze stratami

Systemy kolejkowe ze stratami są oznaczane przez M/M/m/-/m. W takich systemach kolejkowych nie istnieje poczekalnia, w której mogłaby się tworzyć kolejka. Nowe zgłoszenia przychodzące do systemu zostają odrzucone w przypadku gdy wszystkie kanały obsługi są zajęte, co jest ewidentną stratą dla zarządzającego.

2.3.2 System kolejkowy z oczekiwaniem

Systemy kolejkowe z oczekiwaniem są oznaczane jako M/M/m/FIFO/∞. W takich systemach poczekalnia jest nieskończenie wielka, więc nie ma żadnego ograniczenia co do długości kolejki. Zgłoszenie, które wejdzie do kolejki czeka w niej dopóki nie zostanie obsłużone.

2.3.3 System kolejkowy mieszany

Systemy kolejkowe mieszane są oznaczane przez M/M/m/FIFO/m+N. W takim systemie kolejkowym istnieje ograniczona ilość kanałów oraz ograniczona długość kolejki. Jeżeli wszystkie kanały są zajęte to nowe zgłoszenie jest kierowane do kolejki, pod warunkiem, że jest w niej jeszcze miejsce. Jeżeli miejsca w kolejce już nie ma to nowe zgłoszenie zostaje odrzucone.

2.3.4 System kolejkowy o nieograniczonej liczbie kanałów

Systemy kolejkowe o nieograniczonej licznie kanałów są oznaczane przez M/M/∞. Nieograniczona ilość kanałów oznacza brak możliwości utworzenia się kolejki, ponieważ wszystkie przychodzące do systemu zgłoszenia zostają od razu przydzielone do kanałów. W rzeczywistości takie systemy nie istnieją, możemy jednak potraktować system o nieograniczonej liczbie kanałów tak samo jak system, w którym liczba kanałów jest ograniczona, lecz na tyle duża, że nigdy nie dojdzie do sytuacji, w której wszystkie kanały będą zajęte.

2.3.5 System kolejkowy z niecierpliwymi klientami

Przyjmujemy, że system kolejkowy z niecierpliwymi klientami rozszerza system kolejkowy z oczekiwaniem i oznaczamy go przez M/M/m/FIFO/∞. Długość kolejki w takim systemie jest nieograniczona, natomiast ograniczony jest czas przebywania zgłoszenia w kolejce. Jeżeli przed upływem czasu rezygnacji z obsługi zgłoszenie nie otrzyma obsługi, opuszcza ono system, co stanowi stratę dla zarządzającego.

2.3.6 System kolejkowy zamknięty

Systemy kolejkowe zamknięte są oznaczane przez M/M/m/FIFO/N/F. Liczba klientów zgłaszających się do takie systemu jest ograniczona. Przyjmuje się, że do systemu zgłasza się tyle różnych zgłoszeń ile wynosi długość kolejki, zatem nie ma możliwości zajścia sytuacji, w której przychodzące zgłoszenie zostaje odrzucone z powodu braku miejsca w kolejce. Obsłużone zgłoszenia po upływie ustalonego czasu znów zgłaszają się do systemu.

2.4 Analiza symulowanych systemów kolejkowych

Metody analityczne – istotą jest w nich ułożenie i rozwiązanie układów równań różniczkowych wiążących ze sobą prawdopodobieństwa zdarzeń występujących w procesie obsługi. Przy założeniu, że czas $t \to \infty$ układ równań różniczkowych ulega przekształceniu w odpowiadający mu układ równań algebraicznych.

Legenda:

$\bar{t}_a$ - średnia długość interwału pomiędzy dwoma sąsiadującymi zgłoszeniami

λ - średnie natężenie strumienia zgłoszeń

$\bar{t}_0$ - średni czas obsługi zgłoszenia

μ - parametr

q - względna zdolność obsługi systemu

A - bezwzględna zdolność obsługi systemu

ρ - względna intensywność obsługi

p_{odm} - prawdopodobieństwo odmowy tzn. że nowe zgłoszenie zostanie odrzucone

p_0 - prawdopodobieństwo, że wszystkie kanały obsługi są wolne

p_i - prawdopodobieństwo, że system jest w i-tym stanie

$\overline{v}$ - średnia liczba zgłoszeń oczekujących w kolejce

N - maksymalna długość kolejki

m - ilość kanałów obsługi w systemie

$\overline{m_0}$ - średnia liczba zajętych kanałów

$\overline{n}$ - średnia liczba zgłoszeń przebywających w systemie

$\bar{t}_f$ - średni czas oczekiwania zgłoszenia w kolejce

$\bar{t}_s$ - średni czas przebywania jednostki w systemie

M - symbol średniej wartości,

T_s - czas przebywania jednostki w systemie,

T_f - czas oczekiwania zgłoszenia w kolejce,

T_0 - czas obsługi

Niezmienne dla każdego modelu systemu kolejkowego są następujące elementy:

$$\lambda = \frac{1}{\overline{t_a}} \tag{2.1}$$

$$\mu = \frac{1}{\overline{t_0}} \tag{2.2}$$

$$\rho = \frac{\lambda}{\mu} \tag{2.3}$$

Naszym celem dokonania analizy systemu kolejkowego jest opisanie następujących jego parametrów, aby następnie móc je porównać z wynikami symulacji:

$\overline{v}$ $\overline{m_0}$ $\overline{n}$ $\overline{t}_s$

Wzory i sposoby uzyskania tych elementów mogą się różnić w zależności od rodzajów systemów kolejkowych, dlatego należy wypisać wzory dla każdego systemu kolejkowego z osobna.

System kolejkowy ze stratami (M/M/m/-/m):

Tworzymy równania Chapmana Kołmogorowa z warunkami brzegowymi (początkowymi) i znajdujemy ich stacjonarne rozwiązania przy wykorzystaniu warunku

$$\sum_{i=0}^{m} p_i = 1 \tag{2.4}$$

$$p_i = \frac{1}{i!}\rho^i p_0, \quad i = 1, ..., m \tag{2.5}$$

Gdzie p_0 wyliczamy wykorzystując zależność (2.4) otrzymując

$$p_0 = \frac{1}{\sum_{s=0}^{m} \frac{\rho^s}{s!}}, \quad 0 \le s \le m \tag{2.6}$$

Jeżeli wstawimy zależność (2.6) do (2.5) oraz we wzorze wynikowym podstawimy *i = m* to otrzymamy wyrażenie na *prawdopodobieństwo odmowy obsługi*

$$p_{odm} = p_m = \frac{\frac{\rho^m}{m!}}{\sum_{s=0}^{m} \frac{\rho^s}{s!}} \tag{2.7}$$

$$q = 1 - p_m \tag{2.8}$$

$$A = \lambda q = \lambda(1 - p_m) \tag{2.9}$$

W tym systemie nie ma kolejki (średnia długość kolejki $\overline{v}$ wynosi 0), a średnia ilość zgłoszeń przebywających w systemie - $\overline{n}$ jest równa średniej liczbie zajętych kanałów obsługi $\overline{m}_0$

$$\overline{n} = \overline{m}_0 = 0p_0 + 1p_1 + ... + mp_m = \frac{\sum_{s=1}^{m} \frac{\rho^{s-1}}{(s-1)!}}{\sum_{s=0}^{m} \frac{\rho^s}{s!}} \tag{2.10}$$

Można zauważyć, że

$$\overline{n} = \overline{m}_0 = \frac{A}{\mu} = \frac{\lambda(1 - p_m)}{\mu} = \rho(1 - p_m) \tag{2.11}$$

Przekształcając *II formułę Little'a w stanie ustalonym* otrzymujemy wzór na średni czas przebywania jednostki w systemie

$$\overline{t}_s = \frac{1}{\lambda}\overline{n} \tag{2.12}$$

System kolejkowy z oczekiwaniem (M/M/m/FIFO/∞):

Tworzymy równania Chapmana Kołmogorowa z warunkami brzegowymi (początkowymi) i znajdujemy ich stacjonarne rozwiązania przy wykorzystaniu warunku

$$\sum_{i=0}^{m} p_i = 1 \tag{2.13}$$

$$p_i = \frac{\rho^i}{i!} p_0, \quad 1 \le i \le m-1, \quad \rho = \frac{\lambda}{\mu} \tag{2.14}$$

$$p_j = \frac{1}{m! m^{j-m}} \rho^j p_0, \qquad j \geqslant m \tag{2.15}$$

Stąd:

$$p_0 = \frac{1}{\sum_{i=0}^{m-1} \frac{\rho^i}{i!} + \frac{\rho^m}{m!} \sum_{j=m}^{\infty} (\frac{\rho}{m})^{j-m}} \tag{2.16}$$

Zakładając, że $\frac{\rho}{m} < 1$ (czyli system spełnia warunek ergodycznosci) otrzymujemy prawdopodobieństwo, że wszystkie kanały obsługi są zajęte

$$p_m = \frac{\frac{\rho^i}{i!}}{\sum_{i=0}^{m-1} \frac{\rho^i}{i!} + \frac{\rho^m}{(m-1)!(m-\rho)}} \tag{2.17}$$

Przekształcając wzór

$$\overline{v} = \sum_{r=0}^{\infty} \frac{\rho^m}{m!} p_0 r \left(\frac{\rho}{m}\right)^r \tag{2.18}$$

Otrzymujemy

$$\overline{v} = \frac{\overline{\frac{\rho^{m+1}}{(m-\rho)^2 (m-1)!}}}{\sum_{i=0}^{m-1} \frac{\rho^i}{i!} + \frac{\rho^m}{(m-1)!(m-\rho)}} \tag{2.19}$$

W tym systemie kolejka może być nieskończenie długa, a zatem każde przychodzące zgłoszenie, będzie oczekiwać w kolejce, dopóki nie zostanie obsłużone. Prawdopodobieństwo odmowy wynosi więc 0.

$$p_{odm} = 0 \tag{2.20}$$

$$q = 1 - p_{odm} \tag{2.21}$$

Bezwzględna zdolność obsługi systemu - A w tym systemie równa się średniemu natężeniu strumienia zgłoszeń - λ:

$$A = \lambda q = \lambda \tag{2.22}$$

Posiadając wyniki powyższych równań, jesteśmy w stanie wyliczyć szukane parametry systemu wyprowadzając je z poniższych wzorów:

$$\overline{m}_0 = \frac{A}{\mu} \tag{2.23}$$

Obliczmy średni czasy oczekiwania zgłoszenia w kolejce. Jeżeli nowe zgłoszenie przybywa do systemu, podczas gdy wszystkie kanały obsługi są zajęte i nie ma kolejki, musi zaczekać ono na zwolnienie kanału, a średni czas oczekiwania wyniesie $\frac{1}{m\mu}$. Jeżeli zastała jedno zgłoszenie w kolejce, to średni czas oczekiwania wzrasta do $\frac{2}{m\mu}$ itd. Możemy więc napisać

$$\overline{t_f} = \frac{1}{m\mu}p_m + \frac{2}{m\mu}p_{m+1} + \ldots + \frac{r-1}{m\mu}p_{m+r} + \ldots \tag{2.24}$$

gdzie r – liczba jednostek w kolejce. Po przekształceniach otrzymujemy

$$\bar{t}_f = \frac{\rho^{m+1}}{\lambda(m-1)!(m-\rho)^2}\,p_0 \tag{2.25}$$

Znając wzór na $\overline{v}$ otrzymujemy I formułe Little'a dla stanu ustalonego

$$\bar{t}_f = \frac{\overline{v}}{\lambda} \tag{2.26}$$

Obliczmy teraz średni czas obsługi zgłoszenia. Ponieważ $q = 1$, to

$$\bar{t}_0 = \frac{1}{\mu} \tag{2.27}$$

Suma obu czasów daje nam średni czas przebywania jednostki w systemie

$$\bar{t}_s = \bar{t}_f + q\bar{t}_0 \tag{2.28}$$

Z II formuły Little'a w stanie ustalonym liczymy średnią liczbę zgłoszeń przebywających w systemie

$$\overline{n} = \lambda\bar{t}_s \tag{2.29}$$

System kolejkowy mieszany (M/M/m/FIFO/m+N):

Tworzymy równania Chapmana Kołmogorowa z warunkami brzegowymi (początkowymi) i znajdujemy ich stacjonarne rozwiązania przy wykorzystaniu warunku

$$\sum_{i=0}^{N+m} p_i = 1 \tag{2.30}$$

$$\begin{aligned} p_i &= \frac{\rho^i}{i!} p_0, \quad 1 \le i \le m-1, \quad \rho = \frac{\lambda}{\mu} \\ p_i &= \frac{1}{m^{i-m}} \frac{1}{m!} \rho^i p_0, \quad m \le i \le m+N \end{aligned} \tag{2.31}$$

Prawdopodobieństwo p_0 wyliczamy wykorzystując warunek (2.30) dla stanu ustalonego. Podstawiając formułę (2.31) i odpowiednio przekształcając otrzymujemy dwa przypadki:

$$\begin{aligned} p_0 &= \left[\sum_{k=0}^{m-1} \frac{\rho^k}{k!} + \frac{\rho^m}{m!}(N+1) \right]^{-1} & \frac{\rho}{m} = 1 \\ p_0 &= \left[\sum_{k=0}^{m-1} \frac{\rho^k}{k!} + \frac{\rho^m}{m!} \frac{1-(\frac{\rho}{m})^{N+1}}{1-\frac{\rho}{m}} \right]^{-1} & \frac{\rho}{m} \ne 1 \end{aligned} \tag{2.32}$$

Prawdopodobieństwo odmowy jest równe prawdopodobieństwu tego, że wszystkie kanały w systemie są zajęte oraz kolejka jest pełna, czyli stanu systemu p_{m+N}:

$$p_{odm} = p_{m+N} = \frac{\rho^{m+N}}{m^N m!} p_0 \tag{2.33}$$

$$q = 1 - p_{odm} \tag{2.34}$$

$$A = \lambda q \tag{2.35}$$

Posiadając wyniki powyższych równań, jesteśmy w stanie wyliczyć szukane parametry systemu wyprowadzając je z poniższych wzorów:

$$\overline{m}_0 = \frac{A}{\mu} \tag{2.36}$$

$$\overline{v} = \frac{\rho^{m+1}}{mm!} p_0 (1 + 2a + 3a^2 + \ldots + Na^{N-1}), \quad a = \frac{\rho}{m} \tag{2.37}$$

$$\overline{n} = \overline{v} + \overline{m}_0 \tag{2.38}$$

Średni czas oczekiwania zgłoszenia w kolejce otrzymamy, wykorzystując *I formułę Little'a*

$$\bar{t}_f = \frac{\overline{v}}{\lambda} \tag{2.39}$$

Średni czas przebywania zgłoszenia w systemie wynosi

$\bar{t}_s = M(T_s) = M(T_f) + M(T_0)$ gdzie

$M(T_f) = \bar{t}_f = \dfrac{\overline{v}}{\lambda}$

$M(T_0) = q\bar{t}_0 = \dfrac{q}{\mu}$

A więc

$$\bar{t}_s = \bar{t}_f + q\bar{t}_0 = \frac{\overline{v}}{\lambda} + \frac{q}{\mu} \tag{2.40}$$

System kolejkowy o nieograniczonej ilości kanałów obsługi (M/M/∞):

Jeżeli zastosujemy regułę mnemotechniczną, otrzymamy układ równań różniczkowych z następującym warunkiem normalizującym, przy założeniu, że w chwili początkowej w systemie nie było klientów

$$\sum_{i=0}^{\infty} p_i(t) = 1 \tag{2.41}$$

Wykorzystując metodę funkcji tworzącej, możemy znaleźć rozwiązania tego układu równań różniczkowych oraz przebieg w czasie średniej liczby zajętych kanałów obsługi

$$p_i(t) = \frac{(1-e^{-\mu t})^i}{i!}\rho^i e^{-\rho(1-e^{-\mu t})}, \quad i \geq 0 \tag{2.42}$$

$$\overline{m}_0(t) = \sum_{i=1}^{\infty} i p_i(t) = \rho(1-e^{-\mu t}) \tag{2.43}$$

Wartości $p_i(t)$ oraz $\overline{m}_0(t)$ w stanie ustalonym znajdziemy, przechodząc we wzorach (2.42) i (2.43) do granicy z $t \to \infty$. Otrzymamy wówczas

$$p_i = \lim_{t\to\infty} p_i(t) = \frac{1}{i!}\rho^i e^{-\rho}, \quad s \geq 0 \tag{2.44}$$

$$\overline{m}_0 = \lim_{t\to\infty} \overline{m}_0(t) = \lim_{t\to\infty} \rho(1-e^{-\mu t}) = \rho \tag{2.45}$$

Stan ustalony istnieje zawsze, a prawdopodobieństwo, że system znajduje się w stanie *i* na podstawie (2.44), wynosi

$$p_i = \frac{p_i}{i!}p_0, \quad j \geq 1$$

gdzie

$$p_0 = e^{-\rho}, \quad \rho = \frac{\lambda}{\mu} \tag{2.46}$$

Średnią liczbę zgłoszeń przebywających w systemie wyliczamy z zależności

$$\overline{n} = \sum_{n=1}^{\infty} n p_n = \ldots = \rho e^{-\rho} \sum_{m=1}^{\infty} \frac{\rho^m}{m!} = \rho e^{-\rho} e^{\rho} = \rho \tag{2.47}$$

Pozostałe wielkości charakteryzujące pracę systemu obliczamy na podstawie zależności

$$\overline{t}_s = \frac{\overline{n}}{\lambda} = \frac{1}{\mu} \tag{2.48}$$

$$\overline{t}_f = 0, \qquad \overline{v} = \overline{n} - \rho = \rho - \rho = 0 \tag{2.49}$$

System kolejkowy zamknięty (M/M/m/FIFO/N/F):

Tworzymy równania Chapmana Kołmogorowa z warunkami brzegowymi (początkowymi) i znajdujemy ich stacjonarne rozwiązania przy wykorzystaniu warunku

$$\sum_{i=0}^{N} p_i = 1 \tag{2.50}$$

$$p_i = \frac{N!}{i!(N-i)!}\rho^i p_0, \qquad 1 \le i \le m$$

$$p_i = \frac{N!}{m!m^{i-m}(N-i)!}\rho^i p_0, \qquad m+1 \le i \le N \tag{2.51}$$

Aby wyznaczyć p_0, należy wykorzystać wyrażenie (2.50) oraz formuły (2.51). Otrzymamy wówczas po drobnych przekształceniach

$$p_0 = \left[\sum_{i=0}^{m} \frac{N!}{i!(N-i)!}\rho^i + \sum_{j=m+1}^{N} \frac{N!}{m!(N-j)!m^{j-m}}\rho^j\right]^{-1} \tag{2.52}$$

Średnia liczba zgłoszeń oczekujących w kolejce $\overline{v}$ wynosi

$$\overline{v} = \frac{N!}{m!}p_0 \sum_{r=0}^{N-m} \frac{r}{m^r(N-m-r)!}\rho^{m+r} \tag{2.53}$$

Średnią liczbę zajętych kanałów obsługi uzyskujemy z zależności

$$\overline{m}_0 = \sum_{i=0}^{m-1} ip_i + m\left(1 - \sum_{i=0}^{m-1} p_i\right) \tag{2.54}$$

Średnią liczbę zgłoszeń w systemie obliczamy z poniższego wzoru

$$\overline{n} = N - \frac{\overline{m}_0}{\rho} \tag{2.55}$$

Średni czas przebywania zgłoszeń w systemie wyliczamy, stosując II formułę Little'a

$$\overline{t}_s = \frac{\overline{n}}{\lambda(N-\overline{n})} \tag{2.56}$$

3. Opis programu

3.1 Środowisko programistyczne

Program został napisany obiektowo w języku programowania C++ [2] i skompilowany przy pomocy programu *Microsoft Visual C++ 2008.* Do rysowania elementów graficznych użyto biblioteki *freeGLUT*, która jest zestawem narzędzi ułatwiającym korzystanie z biblioteki *OpenGL* [3]. Wczytywanie plików graficznych z rozszerzeniem png zawdzięczamy wykorzystaniu biblioteki *DevIL*. Warto również wspomnieć o bibliotece *TTMath*, z której pomocy skorzystaliśmy przy przechowywaniu wielkich liczb w zmiennych.

3.2 Opis trybów symulacji

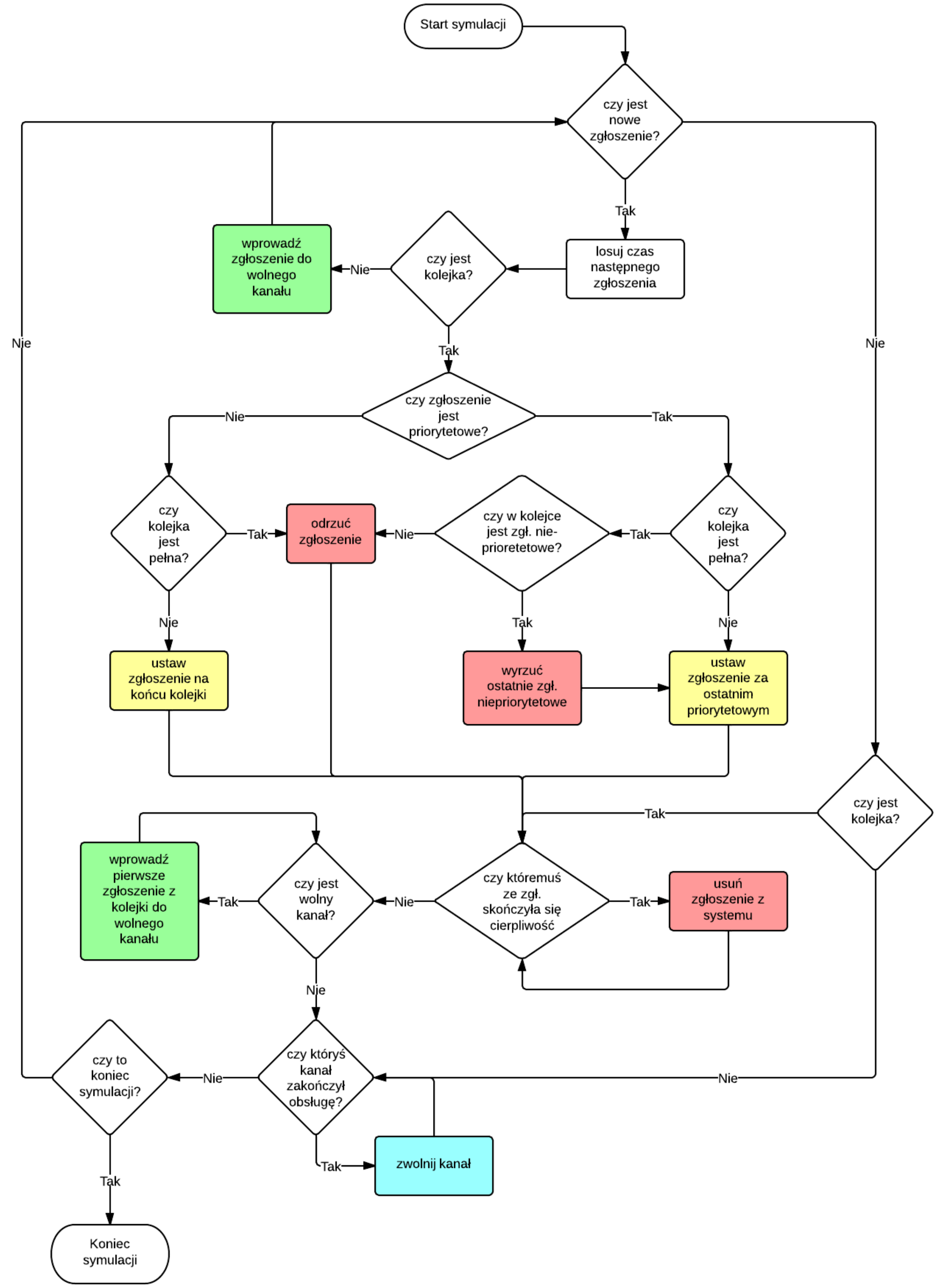
Program został podzielony na dwa tryby: edukacyjny i naukowy. Użytkownik wybiera jeden z nich w menu głównym. Jak wspomniano we wstępie, w trybie edukacyjnym skupiono się na graficznym zilustrowaniu w czasie rzeczywistym zdarzeń zachodzących w stosunkowo małym systemie kolejkowym, natomiast w trybie naukowym nacisk położony jest na obliczenia wartości charakterystycznych systemu ze znacząco większą ilością zgłoszeń i kanałów obsługi oraz na weryfikacji ich wyników.

W obu trybach użytkownik ma podgląd na wykresach takich wartości jak: średni czas pobytu zgłoszenia w systemie i średnia zajętość systemu. Możliwe jest także wstępne określenie parametrów symulacji przed jej rozpoczęciem tj. ilości kanałów, długości kolejki, jej dyscypliny, rozkładów strumieni wejściowych / wyjściowych, niecierpliwości zgłoszeń etc. Ważną różnicą jest ograniczenie nałożone na symulację edukacyjną. W trybie edukacyjnym maksymalna ilość kanałów wynosi 10, zaś maksymalna długość kolejki 40. Ograniczeń tych nie ma w trybie naukowym, w którym ilość kanałów obsługi i długość kolejki są niemal nieograniczone. Ograniczenie to wynika z charakterystyki edukacyjnego trybu, którego celem jest obrazowe przedstawienie systemu kolejkowego w postaci animacji.

3.3 Opis klas

Chart	- klasa reprezentująca wykres, przechowuje jego pozycję na ekranie, tytuł, wyświetlane dane, oraz informacje o stosowanej skali, podziałce wykresu.
Chart_Line	- obiekt tej klasy przedstawia linie, które widzimy na wykresach
Menu	- służy do obsługi menu wyboru trybu symulacji, zawiera informacje o przyciskach, suwakach i polach tekstowych wyświetlanych na ekranie
System	- jest to główna klasa programu, dokonuje potrzebnych do działania programu inicjalizacji
LTexture	- jedynym zadaniem obiektu tej klasy jest przechowywanie tekstury wczytanej z pliku png
Button	- przycisk, obiekty tej klasy są reprezentowane jako przyciski na ekranie
Slider	- suwak, obiekty tej klasy są widoczne na ekranie w menu
Job	- klasa reprezentująca zgłoszenie, potrzebna do poprawnego działania systemu kolejkowego.
Queue	- zawiera informacje o długości kolejki, zgłoszeniach, które aktualnie w niej przebywają oraz czasach, kiedy do owej kolejki weszły
Scientific	- zbiór metod i zmiennych potrzebnych do poprawnego działania symulacji w trybie naukowym
Server	- reprezentuje pojedynczy kanał, zawiera informacje takie jak zgłoszenie, które obecnie obsługuje, czas wejścia danego zgłoszenia do tego kanału oraz swoje współrzędne na ekranie dla edukacyjnego trybu symulacji
Simulation_Box	- zbiór metod i zmiennych potrzebnych do poprawnego działania symulacji w trybie edukacyjnym
Color	- obiekty tej klasy przechowują informacje o kolorach
Label	- obiekty tej klasy przechowują informację o etykietach wyświetlanych na ekranie
Point	- obiekt tej klasy reprezentuje punkt
Rect	- obiekt tej klasy reprezentuje kwadrat
Scale	- obiekt tej klasy zawiera informacje dotyczące podziałki i skali, która ma być zastosowana dla danego wykresu

3.4 Ogólny schemat blokowy



3.5 Opis wybranych procedur

- Procedura *update()* klasy **Scientific** odpowiada za zdarzenia zachodzące w systemie kolejkowym w naukowym trybie symulacji. Metoda ta wywoływana jest co iterację w głównej pętli programu. Sprawdza ona czy w danej iteracji powinno przyjść nowe zgłoszenie czy nie, porównując obecny czas do wcześniej obliczonego czasu przyjścia nowego zgłoszenia. Czas przyjścia nowego zgłoszenia jest ustalany w chwili przyjścia „starego" zgłoszenia, tak więc podczas przyjścia zgłoszenia do systemu poznajemy czas przyjścia kolejnego zgłoszenia. Jeżeli w systemie jest jakiś wolny kanał obsługi to nowe zgłoszenie bezpośrednio do niego przechodzi. Jeżeli natomiast wszystkie kanały obsługi są zajęte to zgłoszenie wchodzi do kolejki w odpowiednie miejsce, w zależności od swojego priorytetu. Zgłoszenia o niskim priorytecie idą na koniec kolejki, natomiast zgłoszenia o priorytecie wysokim ustawiają się zaraz za ostatnim zgłoszeniem w kolejce o wysokim priorytecie. Może oczywiście się okazać, że kolejka jest ograniczona i więcej zgłoszeń już się w niej nie zmieści. W takim przypadku priorytet nowego zgłoszenia również odgrywa znaczącą rolę; zgłoszenie o priorytecie niskim jest odrzucane, natomiast zgłoszenie z wysokim priorytetem wchodzi do kolejki na zasadach opisanych wyżej, wyrzucając z kolejki ostatnie zgłoszenie bez priorytetu. Nowe zgłoszenie priorytetowe również może zostać odrzucone w przypadku, gdy w kolejce nie ma już miejsca, a wszystkie znajdujące się w niej zgłoszenia mają priorytet. Procedura ta również odpowiada za kończenie obsługi zgłoszeń przez kanały je obsługujące oraz decydowanie, czy niecierpliwe zgłoszenie powinno już opuścić kolejkę czy jeszcze nie.

```
void Scientific::update(){
 /*
 DODAWANIE ZGLOSZEN DO KOLEJKI
 */
 clock_t uptime = clock() / (CLOCKS_PER_SEC / 1000);
 static clock_t new_job_time = uptime
  + long(Simulation::r_wykladniczy(average_new_job_time)*1000);
 static int new_id = 1;

 if(!this->closed_queue_system){
  Job* j = NULL;
  if(uptime>new_job_time ){
   int priority = 0;
   if(prob(vip_probability)){
    priority = 1;
   }

   if(queue.size() < unsigned int(max_queue_size) || priority == 1 ||
     (max_queue_size<=0 && servers.size() < unsigned int(max_servers_size) ) ){
    bool impatient = 0;
    int get_impatient_in = 0;
    if(prob(impatient_probability)){
     impatient = 1;
     if(impatient_time_distribution==1){
      get_impatient_in = Simulation::r_wykladniczy(average_impatient_time);
     }if(impatient_time_distribution==2){
      get_impatient_in = (rand()%int(max_impatient_time*100-min_impatient_time*100)
       + int(min_impatient_time*100))/100.0;
     }else{
      get_impatient_in = average_impatient_time;
     }
    }
    int services = get_services_number(services_probabilities,5) + 1;

    j = new Job;

    j->system_join_time = clock();  //zapisujemy czas wejscia do systemu
    j->queue_join_time = clock();  //zapisujemy czas wejscia do kolejki
```

```
    clock_t resign_time = uptime + (get_impatient_in*1000);
    j->set(new_id++, priority, services, impatient, resign_time, get_impatient_in, 7
,0);

    /*
    DODAJEMY DO KOLEJKI ZGLOSZENIE: BEZ PRIORYTETU
    NA KONIEC, Z PRIORYTETEM NA POCZATEK ZARAZ ZA
    OSTATNIM INNYM ZGŁOSZENIEM Z PRIORYTETEM
    */
    update_average_queue_size();
    if(max_queue_size>0){//JEZELI ISTNIEJE KOLEJKA W SYSTEMIE,
     if(j->priority == 0){
      queue.push_back(j);
     }else{
      bool job_added = false;
      for(list<Job*>::iterator it=queue.begin() ; it != queue.end(); ++it){
       if( (*it)->priority == 0 ){
        queue.insert(it,j);
        job_added = true;
        break;
       }
      }
      if(job_added==false && queue.size() < unsigned int(max_queue_size) ){
       queue.push_back(j);
      }
     }
    }else{//JEZELI W SYSTEMIE NIE MA KOLEJKI, TO ZGLOSZENIA PROBOJA ODRAZU ISC DO
KANALOW
     if(servers.size() < unsigned int(max_servers_size)){

      if(service_time_distribution==1){//rozklad wykladniczy
       j->service_end_time = uptime
        + long(Simulation::r_wykladniczy(average_service_time)*1000);
      }else if(service_time_distribution==2){//rozklad jednostajny
       j->service_end_time = uptime
        + ((rand()%int(max_service_time*100-min_service_time*100)
        + int(min_service_time*100))/100.0)*1000;
      }else{//wartosc nielosowa
       j->service_end_time = uptime + average_service_time*1000;
      }
      j->action = 3;
      if(servers.empty()){
       servers.push_back(j);
      }else{
       bool job_added = false;
       for(list<Job*>::iterator it=servers.begin() ; it != servers.end(); ++it){
        if( (*it)->service_end_time < j->service_end_time){
         servers.insert(it,j);
         job_added = true;
         break;
        }
       }
       if(job_added==false){
        servers.push_back(j);
       }
      }
      update_average_business();
      busy_servers++;

     }else{
      queue.push_back(j);
     }

    }
    /*
    JEŻELI PO DODANIU ZGŁOSZENIA DO KOLEJKI
    OKAZAŁO SIĘ, ŻE KOLEJKA JEST ZA
    DŁUGA TO USUWAMY OSTATNI JEJ ELEMENT
    */
    if(queue.size() > unsigned int(max_queue_size) ){
```

```
     if(queue.back()->impatient){
      for(list<Job*>::iterator it=impatients.begin() ; it != impatients.end(); ++it){
       if( (*it)->id == queue.back()->id ){
        impatients.erase(it);
        break;
       }
      }
     }
     delete queue.back();
     queue.pop_back();
     jobs_rejected++;
    }

   }else{
    jobs_rejected++;
   }

   if(new_job_time_distribution==1){
    new_job_time = uptime + long(Simulation::r_wykladniczy(average_new_job_time)*1000);
   }else if(new_job_time_distribution==2){
    new_job_time = uptime + ((rand()%int(max_new_job_time*100-min_new_job_time*100)
     + int(min_new_job_time*100))/100.0)*1000;
   }else{
    new_job_time = uptime + average_new_job_time*1000;
   }
  }
  /*
  DODAWANIE NOWYCH NIECIERPLIWYCH ZGLOSZEN
  DO LISTY NIECIERPLIWYCH ZGLOSZEN
  */
  if(j!=NULL && j->impatient){
   if(impatients.empty()){
    impatients.push_back(j);
   }else{
    bool job_added = false;
    for(list<Job*>::iterator it=impatients.begin() ; it != impatients.end(); ++it){
     if( (*it)->resign_time < j->resign_time){
      impatients.insert(it,j);
      job_added = true;
      break;
     }
    }
    if(job_added==false){
     impatients.push_back(j);
    }
   }
  }

 }else{//ZAMKNIETY SYSTEM KOLEJKOWY - ZGLOSZENIE JEST W OBRABIARCE
  while(!destroyers.empty() && destroyers.back()->service_end_time < uptime){
   destroyers.back()->action = 7;

   //ZGŁOSZENIE BĘDĄC W OBRABIARCE JEST POZA SYSTEMEM
   destroyers.back()->system_join_time = clock();  //zapisujemy czas wejscia do systemu
   destroyers.back()->queue_join_time = clock();  //zapisujemy czas wejscia do kolejki

   update_average_queue_size();
   queue.push_back(destroyers.back());

   destroyers.pop_back();
   jobs_destroyed++;
  }
 }

 /*
 USUWANIE NIECIERPLIWYCH ZGŁOSZEŃ
 Z KOLEJKI
 */
 while(!impatients.empty() && impatients.back()->resign_time < uptime){
  if(impatients.back()->action != 3){
```

```
  update_average_queue_size();
  for(list<Job*>::iterator it=queue.begin() ; it != queue.end(); ++it){
   if( (*it)->id == impatients.back()->id ){
    queue.erase(it);
    break;
   }
  }
  delete impatients.back();
  jobs_resigned++;
 }
 impatients.pop_back();
}
/////////////////////////////////

/*
PRZECHODZENIE Z KOLEJKI
DO KANAŁÓW
*/
while(!queue.empty() && servers.size() < unsigned int(max_servers_size) ){
 if(service_time_distribution==1){//rozklad wykladniczy
  queue.front()->service_end_time = uptime
   + long(Simulation::r_wykladniczy(average_service_time)*1000);
 }else if(service_time_distribution==2){//rozklad jednostajny
  queue.front()->service_end_time = uptime
   + ((rand()%int(max_service_time*100-min_service_time*100)
   + int(min_service_time*100))/100.0)*1000;
 }else{//wartosc nielosowa
  queue.front()->service_end_time = uptime + average_service_time*1000;
 }
 queue.front()->action = 3;
 if(servers.empty()){
  servers.push_back(queue.front());
 }else{
  bool job_added = false;
  for(list<Job*>::iterator it=servers.begin() ; it != servers.end(); ++it){
   if( (*it)->service_end_time < queue.front()->service_end_time){
    servers.insert(it,queue.front());
    job_added = true;
    break;
   }
  }
  if(job_added==false){
   servers.push_back(queue.front());
  }
 }
 update_average_business();
 busy_servers++;

 //DO WYKRESÓW I SYMULACYJNEJ ANALIZY
 sys.sum_of_waiting_in_queue_times+=clock() - queue.front()->queue_join_time;
 sys.jobs_that_left_queue++;
 ////////////////////////////////////

 update_average_queue_size();
 queue.pop_front();
}

/*
USUWAMY KLIENTÓW Z KANAŁÓW
*/
while(!servers.empty() && servers.back()->service_end_time < uptime){
 if(!this->closed_queue_system){
  servers.back()->services--;
  if(servers.back()->services > 0){
   if(servers.back()->impatient){
    servers.back()->resign_time = uptime + (servers.back()->get_impatient_in*1000);
   }
   /*
   DODAJEMY DO KOLEJKI ZGLOSZENIE: BEZ PRIORYTETU
   NA KONIEC, Z PRIORYTETEM NA POCZATEK ZARAZ ZA
```

```
     OSTATNIM INNYM ZGŁOSZENIEM Z PRIORYTETEM
     */
     servers.back()->queue_join_time = clock(); //zapisujemy czas wejscia do kolejki
     update_average_queue_size();
     if(servers.back()->priority == 0){
      queue.push_back(servers.back());
     }else{
      bool job_added = false;
      for(list<Job*>::iterator it=queue.begin() ; it != queue.end(); ++it){
       if( (*it)->priority == 0 ){
        queue.insert(it,servers.back());
        job_added = true;
        break;
       }
      }
      if(job_added==false){
       queue.push_back(servers.back());
      }
     }

     /*
      JEŻELI PO DODANIU ZGŁOSZENIA DO KOLEJKI OKAZAŁO SIĘ,
      ŻE KOLEJKA JEST ZA DŁUGA TO USUWAMY OSTATNI JEJ ELEMENT
     */
     if(queue.size() > unsigned int(max_queue_size) ){
      if(queue.back()->impatient){
       for(list<Job*>::iterator it=impatients.begin() ; it != impatients.end(); ++it){
        if( (*it)->id == queue.back()->id ){
         queue.erase(it);
         break;
        }
       }
      }
      /*
      JEŻELI USUWANE ZGŁOSZENIE BYŁO NIECIERPLIWE TO USUWAMY JE TEŻ
      Z LISTY NIECIERPLIWYCH ZGŁOSZEŃ
      */
      if(queue.back()->impatient){
       for(list<Job*>::iterator it=impatients.begin() ; it != impatients.end(); ++it){
        if(queue.back()->id == (*it)->id){
         impatients.erase(it);
         break;
        }
       }
      }

      delete queue.back();
      queue.pop_back();
     }

    }else{
     /*
     JEŻELI USUWANE ZGŁOSZENIE BYŁO NIECIERPLIWE TO USUWAMY JE TEŻ
     Z LISTY NIECIERPLIWYCH ZGŁOSZEŃ
     */
     if(servers.back()->impatient){
      for(list<Job*>::iterator it=impatients.begin() ; it != impatients.end(); ++it){
       if(servers.back()->id == (*it)->id){
        impatients.erase(it);
        break;
       }
      }
     }

     //DO WYKRESÓW I SYMULACYJNEJ ANALIZY
     sys.sum_of_waiting_in_system_times+=clock() - servers.back()->system_join_time;
     sys.jobs_that_left_system++;
     ////////////////////////////////////
     delete servers.back();
    }
```

```
    servers.pop_back();
    jobs_serviced++;

    update_average_business();
    busy_servers--;
   }else{
    /*
    ZAMKNIETY SYSTEM KOLEJKOWY:
    ZGLOSZENIA Z KANALOW WRACAJA
    WPROST DO OBRABIAREK
    */

    //DO WYKRESÓW I SYMULACYJNEJ ANALIZY
    servers.back()->action = 8;
    sys.sum_of_waiting_in_system_times+=clock() - servers.back()->system_join_time;
    sys.jobs_that_left_system++;
    ////////////////////////////////////

    clock_t job_destroy_time;
    if(new_job_time_distribution==1){
     job_destroy_time = uptime
      + long(Simulation::r_wykladniczy(average_new_job_time)*1000);
    }else if(new_job_time_distribution==2){
     job_destroy_time = uptime
      + ((rand()%int(max_new_job_time*100-min_new_job_time*100)
      + int(min_new_job_time*100))/100.0)*1000;
    }else{
     job_destroy_time = uptime + average_new_job_time*1000;
    }
    servers.back()->service_end_time = job_destroy_time;

    if(destroyers.empty()){
     destroyers.push_back(servers.back());
    }else{
     bool job_added = false;
     for(list<Job*>::iterator it=destroyers.begin() ; it != destroyers.end(); ++it){
      if( (*it)->service_end_time < servers.back()->service_end_time){//HMM?
       destroyers.insert(it,servers.back());
       job_added = true;
       break;
      }
     }
     if(job_added == false){
      destroyers.push_back(servers.back());
     }
    }
    servers.pop_back();
    jobs_repaired++;

    update_average_business();
    busy_servers--;
   }
  }
 }
```

- Procedura *render()* klasy **Simulation_Box** odpowiada za rysowanie na ekranie symulacji systemu kolejkowego w trybie edukacyjnym. Najpierw rysowana jest obwódka prostokąta, służąca za opakowanie symulacji, umiejscowiona w dolnej prawej ćwiartce ekranu. Następnie w zależności od tego jak długa jest kolejka rysowane są odpowiednio szara ściana oraz kolejka wydrążona poziomo przez jej środek. Wolne kanały rysowane są w kolorze zielonym, natomiast zajęte w kolorze czerwonym. Dla systemu kolejkowego zamkniętego, gdzie kanały są uznawane za konserwatorów, rysowane są również obrabiarki na lewo od końca kolejki. Każde zgłoszenie rysowane jest jako kwadrat o wielkości 7x7 pikseli, a jego kolor zależy od kilku czynników. Początkowo zgłoszenia bez priorytetu mają kolor czarny, natomiast z priorytetem kolor niebieski. Czekając w kolejce niecierpliwe z nich stają się z czasem coraz bardziej czerwone, co odzwierciedla ich poziom zniecierpliwienia, jeżeli ich kolor stanie się intensywnie czerwony to opuszczają system. Czerwone są również zgłoszenia opuszczające system z powodu odrzucenia tzn. próby wejścia do systemu podczas, gdy kolejka jest zapełniona, co powoduje odrzucenie. Zgłoszenia opuszczające system po pomyślnym obsłużeniu mają kolor zielony, natomiast te, które po obsłużeniu wracają z powrotem do kolejki, by zostać ponownie obsłużone – mają kolor żółty. Procedura na samym końcu wywołuje inną procedurę: *render_info()*, której zadaniem jest narysowanie informacji o symulacji znajdujących się w dolnej lewej ćwiartce ekranu.

```
void Simulation_Box::render(){
 //PROSTOKĄT Z SYMULACJĄ
 glBegin(GL_LINE_LOOP);
 glColor3f(0.f, 0.f, 0.f );
 glVertex2f(position.x-1, position.y);
 glVertex2f(position.x, position.y+position.h);
 glVertex2f(position.x+position.w, position.y+position.h);
 glVertex2f(position.x+position.w, position.y);
 glEnd();

 //SZARE ŚCIANY NAD I POD KOLEJKĄ
 glColor3f(0.9f,0.9f,0.9f);
 glBegin(GL_QUADS);
 glVertex2f(position.x+queue_x, position.y);
 glVertex2f(position.x+queue_end_x+10, position.y);
 glVertex2f(position.x+queue_end_x+10, position.y+position.h-1);
 glVertex2f(position.x+queue_x, position.y+position.h-1);
 glEnd();
 glColor3f(0.0f,0.0f,0.0f);
 glBegin(GL_LINES);
 glVertex2f(position.x+queue_x+1, position.y);
 glVertex2f(position.x+queue_x+1, position.y+position.h-1);
 glVertex2f(position.x+queue_end_x+10, position.y);
 glVertex2f(position.x+queue_end_x+10, position.y+position.h-1);
 glEnd();

 //KOLEJKA JAKO BIAŁA DROGA PRZEZ ŚCIANY
 glColor3f(1.f,1.f,1.f);
 glBegin(GL_QUADS);
 glVertex2f(position.x+queue_x, position.y+(position.h/2)-25);
 glVertex2f(position.x+queue_end_x+10, position.y+(position.h/2)-25);
 glVertex2f(position.x+queue_end_x+10, position.y+(position.h/2)+25);
 glVertex2f(position.x+queue_x, position.y+(position.h/2)+25);
 glEnd();
 glColor3f(0.f,0.f,0.f);
 glBegin(GL_LINES);
 glVertex2f(position.x+queue_x, position.y+(position.h/2)-25);
 glVertex2f(position.x+queue_end_x+10, position.y+(position.h/2)-25);
 glVertex2f(position.x+queue_x, position.y+(position.h/2)+25);
 glVertex2f(position.x+queue_end_x+10, position.y+(position.h/2)+25);
 glEnd();
 if(available_servers){
  glColor3f(0.f, 0.8f, 0.f );
 }else{
```

```
 glColor3f(0.9f, 0.f, 0.f );
}
glBegin(GL_LINES);
glVertex2f(position.x+queue_end_x, position.y+(position.h/2)-25);
glVertex2f(position.x+queue_end_x, position.y+(position.h/2)+24);
glEnd();

for(unsigned int i=0;i<servers.size();i++){
 Rect* r = &servers[i].pos;
 if(servers[i].servicing!=NULL){
  glColor3f(0.9f, 0.f, 0.f );
 }else{
  glColor3f(0.f, 0.8f, 0.f );
 }
 glBegin(GL_QUADS);
 glVertex2f(position.x-(r->w/2) + r->x, position.y-(r->h/2) + r->y);
 glVertex2f(position.x-(r->w/2) + r->x, position.y+(r->h/2) + r->y);
 glVertex2f(position.x+(r->w/2) + r->x, position.y+(r->h/2) + r->y);
 glVertex2f(position.x+(r->w/2) + r->x, position.y-(r->h/2) + r->y);
 glEnd();
 glColor3f(0.f, 0.f, 0.f );
 glBegin(GL_LINE_LOOP);
 glVertex2f(position.x-(r->w/2) + r->x-1, position.y-(r->h/2) + r->y);
 glVertex2f(position.x-(r->w/2) + r->x, position.y+(r->h/2) + r->y);
 glVertex2f(position.x+(r->w/2) + r->x, position.y+(r->h/2) + r->y);
 glVertex2f(position.x+(r->w/2) + r->x, position.y-(r->h/2) + r->y);
 glEnd();

 Label l;
 stringstream id_string;
 id_string << servers[i].id;
 l.set(position.x+r->x,position.y+r->y,
      id_string.str(),GLUT_BITMAP_TIMES_ROMAN_10,CENTER);
 l.render();
}
if(this->closed_queue_system){
 for(unsigned int i=0;i<destroyers.size();i++){
  Rect* r = &destroyers[i].pos;
  if(destroyers[i].servicing!=NULL){
   glColor3f(0.9f, 0.f, 0.f );
  }else{
   glColor3f(0.f, 0.8f, 0.f );
  }
  glBegin(GL_QUADS);
  glVertex2f(position.x-(r->w/2) + r->x, position.y-(r->h/2) + r->y);
  glVertex2f(position.x-(r->w/2) + r->x, position.y+(r->h/2) + r->y);
  glVertex2f(position.x+(r->w/2) + r->x, position.y+(r->h/2) + r->y);
  glVertex2f(position.x+(r->w/2) + r->x, position.y-(r->h/2) + r->y);
  glEnd();
  glColor3f(0.f, 0.f, 0.f );
  glBegin(GL_LINE_LOOP);
  glVertex2f(position.x-(r->w/2) + r->x-1, position.y-(r->h/2) + r->y);
  glVertex2f(position.x-(r->w/2) + r->x, position.y+(r->h/2) + r->y);
  glVertex2f(position.x+(r->w/2) + r->x, position.y+(r->h/2) + r->y);
  glVertex2f(position.x+(r->w/2) + r->x, position.y-(r->h/2) + r->y);
  glEnd();

  Label l;
  stringstream id_string;
  id_string << destroyers[i].id;
  l.set(position.x+r->x,position.y+r->y,
      id_string.str(),GLUT_BITMAP_TIMES_ROMAN_10,CENTER);
  l.render();
 }
}

for (list<Job>::iterator it=jobs.begin(); it != jobs.end(); ++it){
 float red = 0.f;
 if(it->impatient && it->action==7){
  /*
```

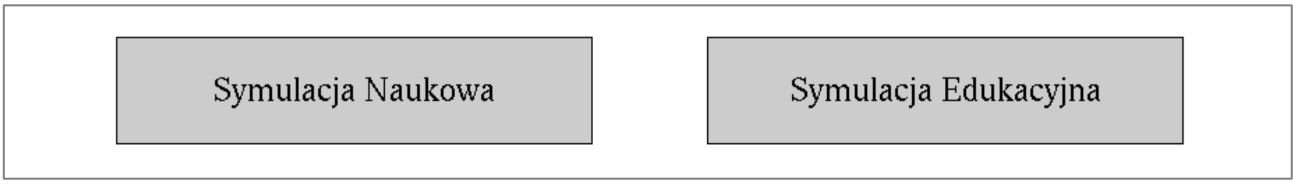
```
    CZYM BARDZIEJ ZNIECIERPLIWIONE ZGŁOSZENIE
    TYM BARDZIEJ CZERWONY KOLOR ZGŁOSZENIA
    */
    red = impatient_level(it);
   }
   /*
   W ZAMKNIĘTYM SYSTEMIE KOLEJKOWYM
   INNE ZASADY KOLOROWANIA
   */
   if(this->closed_queue_system){
    glColor3f(0.f,0.f,0.f);
   }else{
    if(instant_resign){
     glColor3f(red, red, red);
    }else{
     glColor3f(red, 0.f, 0.f );
    }
    if(it->action==6){
     glColor3f(0.9f, 0.f, 0.f );
    }else if(it->action==4){
     glColor3f(0.f, 0.8f, 0.f );
    }else if(it->action==5 || it->already_serviced){
     glColor3f(0.8f, 0.8f, 0.f );
    }else if(it->priority==1){
     if(instant_resign){
      glColor3f(red, red, 1.f);
     }else{
      glColor3f(red, 0.0f, 1.f - red );
     }
    }
   }
   glBegin(GL_QUADS);
   glVertex2f(position.x-3 + it->pos.x, position.y-3 + it->pos.y);
   glVertex2f(position.x-3 + it->pos.x, position.y+3 + it->pos.y);
   glVertex2f(position.x+3 + it->pos.x, position.y+3 + it->pos.y);
   glVertex2f(position.x+3 + it->pos.x, position.y-3 + it->pos.y);
   glEnd();

   if(it->impatient && instant_resign){
    if(it->priority==0){
     glColor3f(0.f, 0.f, 0.f );
    }else if(it->priority==1){
     glColor3f(0.f, 0.f, 1.f );
    }
    glBegin(GL_LINE_LOOP);
    glVertex2f(position.x-2 + it->pos.x-1, position.y-2 + it->pos.y);
    glVertex2f(position.x-2 + it->pos.x, position.y+3 + it->pos.y);
    glVertex2f(position.x+3 + it->pos.x, position.y+3 + it->pos.y);
    glVertex2f(position.x+3 + it->pos.x, position.y-2 + it->pos.y);
    glEnd();
   }
   if(it->choice){
    Label l;
    stringstream id_string;
    id_string << it->choice->id;
    l.set(position.x+it->pos.x,position.y+it->pos.y-5,
        id_string.str(),GLUT_BITMAP_TIMES_ROMAN_10,CENTER);
    l.render();
   }
  }
  render_info();
 }
```

4. Instrukcja obsługi

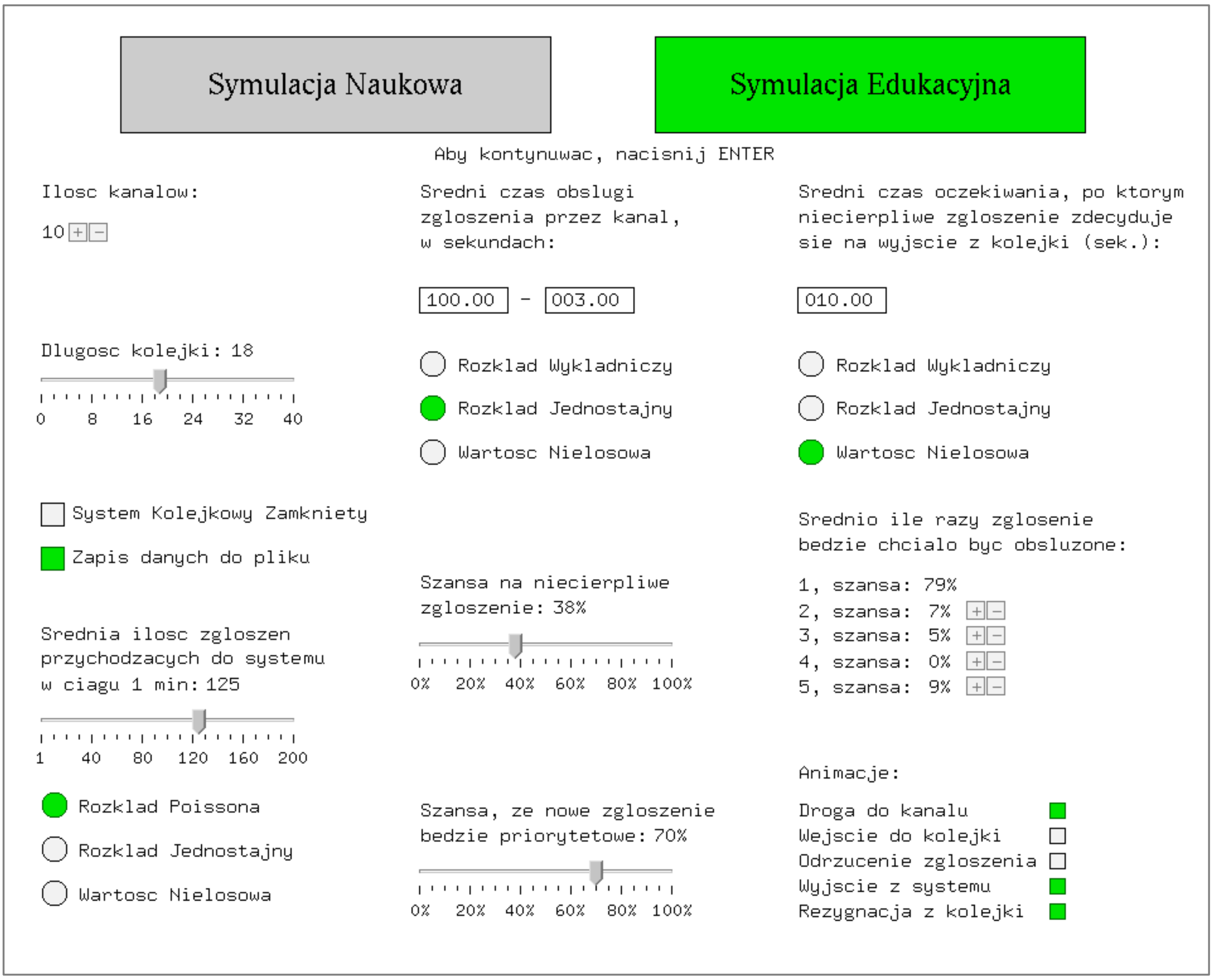
4.1 Opis menu interfejsu

Na początku wybieramy tryb pracy: naukowy lub edukacyjny. Wybrana opcja podświetli się na kolor zielony.



Rys. 4.1 Wybór trybu pracy

Następnie w menu wyboru parametrów określamy model systemu kolejkowego, dla jakiego chcemy przeprowadzić symulację, oraz jego charakterystykę.

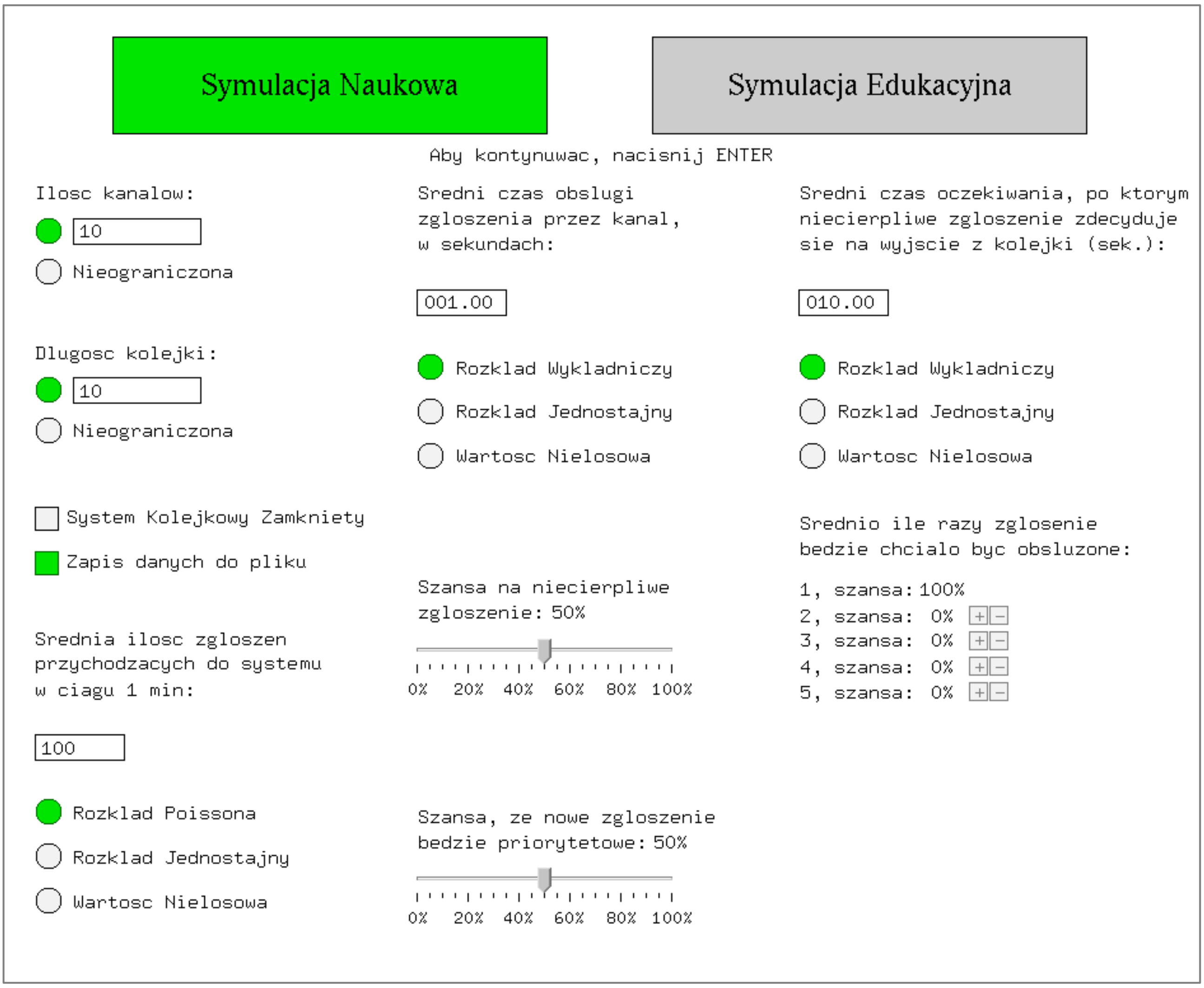


Rys. 4.2 Menu wyboru parametrów dla trybu edukacyjnego

W trybie edukacyjnym możemy regulować:
- ilość kanałów,
- ilość miejsc w kolejce,
- czy system jest zamknięty

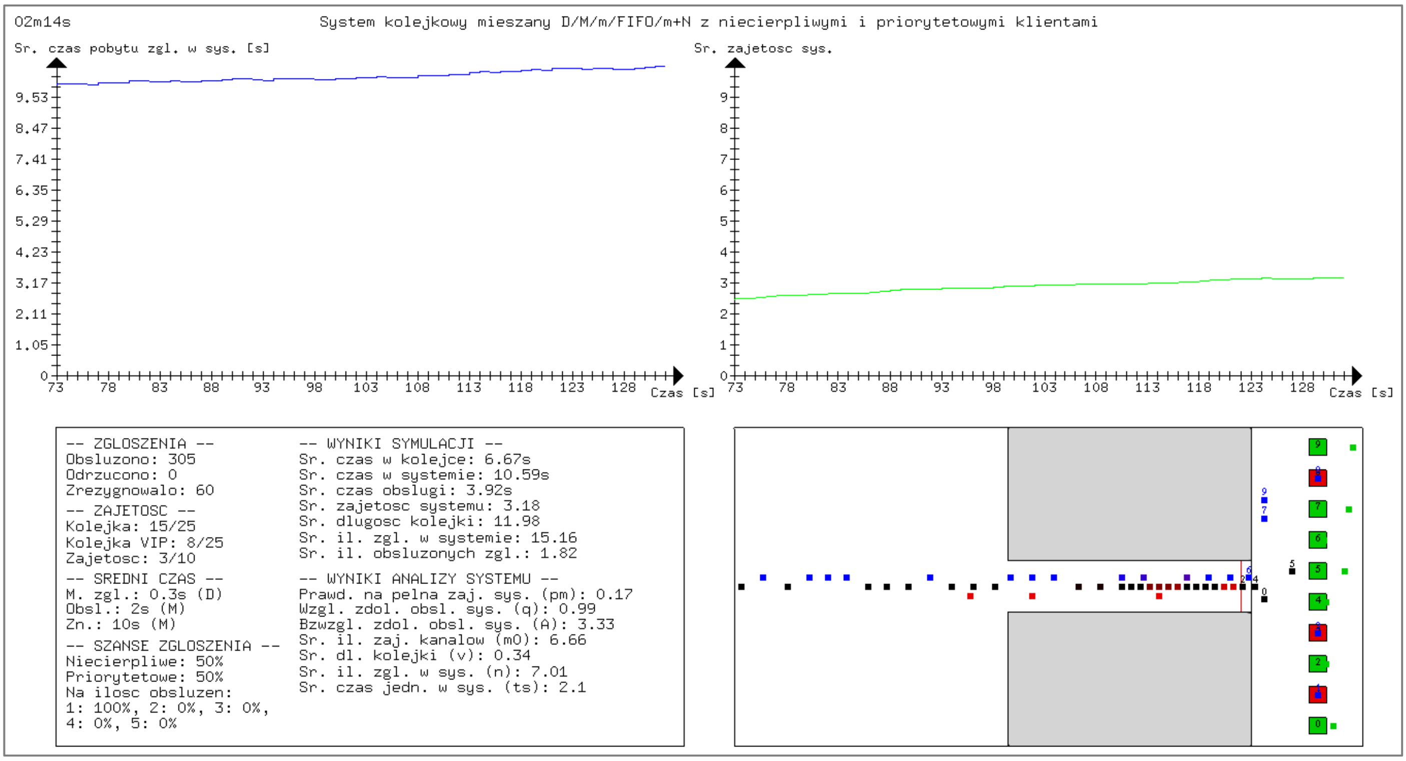
- ilość zgłoszeń w ciągu minuty
- czas obsługi zgłoszeń w sekundach
- prawdopodobieństwo, że zgłoszenie będzie niecierpliwe,
- prawdopodobieństwo, że zgłoszenie będzie z priorytetem,
- czas, po którym niecierpliwe zgłoszenie odchodzi z systemu
- prawdopodobieństwo, że zgłoszenie będzie chciało być obsłużone więcej niż 1 raz (określenie prawdopodobieństw w procentach osobno dla każdej z 5 opcji, ich suma zawsze musi wynieść 100%)
- włączanie/wyłączanie wyświetlania poszczególnych części animacji

Menu wyboru parametrów dla trybu naukowego wygląda generalnie tak samo z wyjątkiem braku części poświęconej animacjom oraz nie posiada górnych limitów długości kolejki i ilości kanałów:



Rys. 4.3 Menu wyboru parametrów dla trybu naukowego

4.2 Informacje wyświetlane w części edukacyjnej



Rys. 4.4 Informacje wyświetlane w części edukacyjnej

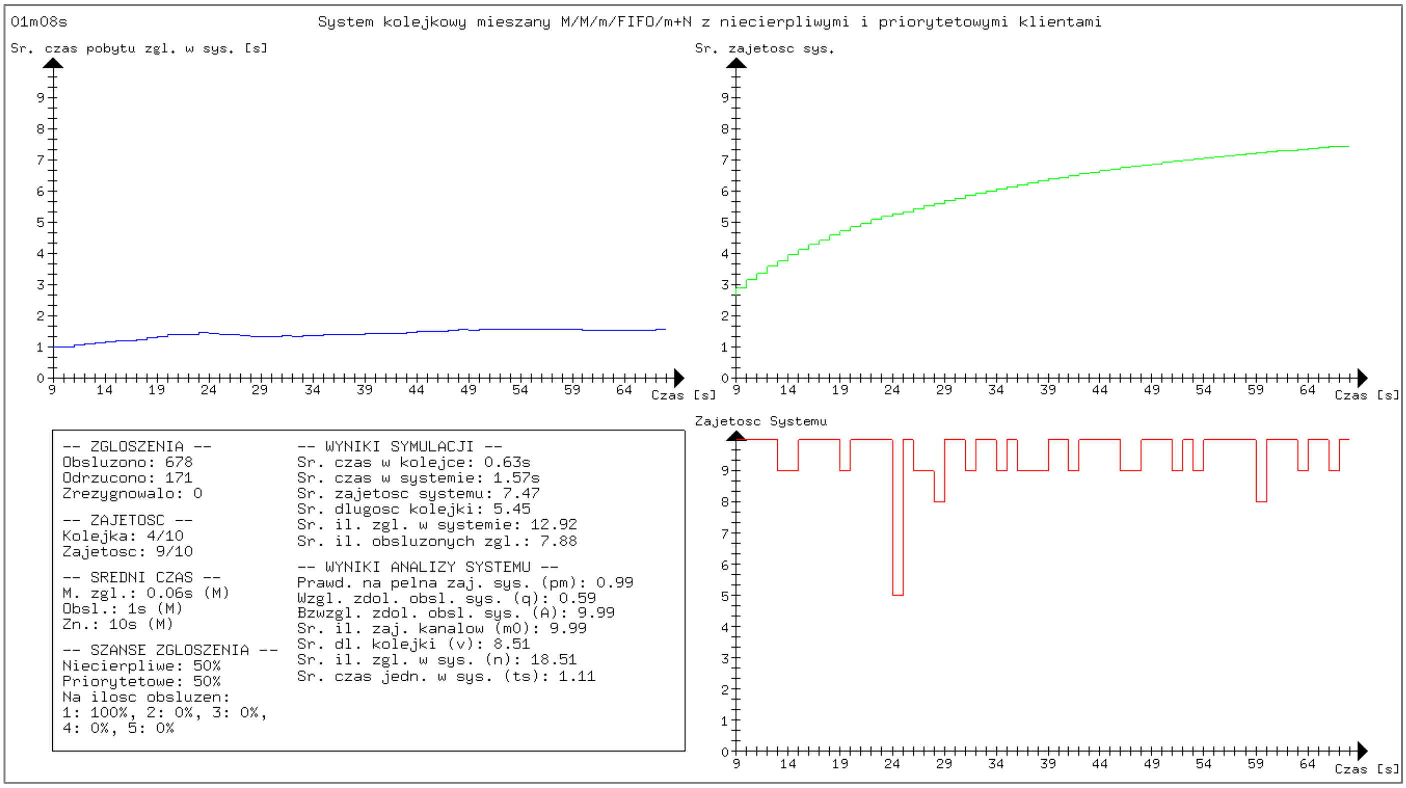
Po rozpoczęciu symulacji ukazuje nam się główny ekran programu. Opatrzony jest tytułem z nazwą wybranego modelu s.k. i jego oznaczeniem kodowym. Widoczny obszar podzielony został na 4 równe części, zawierające:

- graficzne przedstawienie obecnego stanu systemu,
- pole z tekstowymi informacjami o:
    - parametrach systemu ustalonych przed rozpoczęciem symulacji,
    - wynikach symulacji,
    - wynikach analizy systemu
- wykres średniego czasu pobytu zgłoszenia w systemie
- wykres średniej zajętości systemu

W ramce z animacją widzimy:

- ponumerowane kanały obsługi z prawej strony w kolorze zielonym (wolne) i czerwonym (zajęte),
- zgłoszenia nadchodzące z lewej strony w kolorze niebieskim (z priorytetem) i czarnym (zwykłe); w przypadku niecierpliwych zgłoszeń zmnieniają one stopniowo swój kolor na czerwony i w momencie jego osiągnięcia wychodzą kolejki, wracając skąd przybyły; obsłużone zgłoszenia wychodzą z kanałów w kolorze zielonym lub żółtym w przypadku, gdy chcą być obłużone jeszcze raz.

4.3 Informacje wyświetlane w części naukowej



Rys. 4.5 Informacje wyświetlane w części naukowej

Widok symulacji trybu naukowego jest podobny do edukacyjnego, z jedną różnicą: zamiast animacji jest wykres zajętości systemu.

Wynikiem analizy systemu jest obliczone:

- prawdopodobieństwo na pełną zajętość systemu,
- względna zdolność obsługi systemu,
- bezwzględna zdolność obsługi systemu,
- średnia ilość zajętych kanałów,
- średnia długość kolejki,
- średnia ilość zgłoszeń w systemie,
- średni czas przebywania w systemie

5. Zawartość CD

- katalog z plikiem wykonywalnym .exe
- katalog z projektem MS VC++ (kod źródłowy)
- dokumentacja w formacie .pdf (ten dokument)

```
D:\
│   dokumentacja.pdf
│
├───Release
│       DevIL.dll
│       freeglut.dll
│       glew32.dll
│       ILU.dll
│       ILUT.dll
│       logo.png
│       Microsoft.VC90.CRT.manifest
│       msvcm90.dll
│       msvcp90.dll
│       msvcr90.dll
│       Symulator Kolejki.exe
│       Symulator Kolejki.pdb
│       ziarno.txt
│
└───SK kod źródłowy
    │   c_Button.cpp
    │   c_Button.h
    │   c_Chart.cpp
    │   c_Chart.h
    │   c_Chart_Line.cpp
    │   c_Chart_Line.h
    │   c_Job.cpp
    │   c_Job.h
    │   c_Menu.cpp
    │   c_Menu.h
    │   c_Queue.cpp
    │   c_Queue.h
    │   c_Scientific.cpp
    │   c_Scientific.h
    │   c_Server.cpp
    │   c_Server.h
    │   c_Simulation_Box.cpp
    │   c_Simulation_Box.h
    │   c_Slider.cpp
    │   c_Slider.h
    │   c_System.cpp
    │   c_System.h
    │   h_Includes.h
    │   logo.png
    │   LTexture.cpp
    │   LTexture.h
    │   main.cpp
    │   n_Simulation.cpp
    │   n_Simulation.h
    │   Symulator Kolejki.vcproj
    │   Symulator Kolejki.vcproj.user
    │   Symulator Kolejki.vcxproj
    │   Symulator Kolejki.vcxproj.filters
    │   Symulator Kolejki.vcxproj.user
    │   s_Color.h
    │   s_Label.cpp
    │   s_Label.h
    │   s_Point.h
    │   s_Rect.h
    │   s_Scale.h
    │   ziarno.txt
    │
    ├───DevIL-SDK-x86-1.7.8
    │   ├───include
    │   │   └───IL
    │   │           config.h
    │   │           config.h.win
    │   │           devil_internal_exports.h
    │   │           il.h
    │   │           ilu.h
    │   │           ilut.h
    │   │           ilut_config.h
    │   │           ilu_region.h
    │   │           il_wrap.h
    │   │
    │   └───lib
    │           DevIL.dll
    │           DevIL.lib
    │           ILU.dll
    │           ILU.lib
    │           ILUT.dll
    │           ILUT.lib
    │
    ├───freeglut
    │   │   LFRect.cpp
    │   │   LFRect.h
    │   │   LOpenGL.cpp
    │   │   LOpenGL.h
    │   │   LTexture.cpp
    │   │   LTexture.h
    │   │   LUtil.cpp
    │   │   LUtil.h
    │   │   main.cpp
    │   │
    │   ├───bin
    │   │   │   freeglut.dll
    │   │   │
    │   │   └───x64
    │   │           freeglut.dll
    │   │
    │   ├───include
    │   │   └───GL
    │   │           freeglut.h
    │   │           freeglut_ext.h
    │   │           freeglut_std.h
    │   │           glut.h
    │   │
    │   └───lib
    │       │   freeglut.lib
    │       │
    │       └───x64
    │               freeglut.lib
    │
    ├───glew-1.9.0
    │   ├───bin
    │   │       glew32.dll
    │   │       glew32mx.dll
    │   │       glewinfo.exe
    │   │       visualinfo.exe
    │   │
    │   ├───include
    │   │   └───GL
    │   │           glew.h
    │   │           glxew.h
    │   │           wglew.h
    │   │
    │   └───lib
    │           glew32.lib
    │           glew32mx.lib
    │           glew32mxs.lib
    │           glew32s.lib
    │
    └───ttmath
            ttmath.h
            ttmathbig.h
            ttmathdec.h
            ttmathint.h
            ttmathmisc.h
            ttmathobjects.h
            ttmathparser.h
            ttmaththreads.h
            ttmathtypes.h
            ttmathuint.h
            ttmathuint_noasm.h
            ttmathuint_x86.h
            ttmathuint_x86_64.h
```

Rys. 5.1. Drzewo katalogów i plików

6. Bibliografia

[1] B. Filipowicz, "Modele stochastyczne w badaniach operacyjnych", WNT, 1996

[2] B. Stroustrup, “Język C++”

[3] Kurs OpenGL, http://www.glprogramming.com/red/

7. Oświadczenie

Oświadczamy, że niniejszy projekt oraz dokumentację wykonaliśmy samodzielnie i wyrażamy zgodę na wykorzystanie ich do celów dydaktycznych.

Krzysztof Jura	Michał Gawroński
....	....