

UNIWERSYTET KARDYNAŁA STEFANA WYSZYŃSKIEGO  
W WARSZAWIE

WYDZIAŁ MATEMATYCZNO-PRZYRODNICZY  
SZKOŁA NAUK ŚCISŁYCH

Piotr Majblat

56857

Nauki ścisłe

Specjalność: informatyka stosowana

Wyznaczanie pól powierzchni wielokątów metodą Monte Carlo

Praca licencjacka wykonana  
pod kierunkiem naukowym  
dr. Tomasza Sowińskiego

WARSZAWA 2008



# Spis treści

<b>1. Wstęp</b>	<b>1</b>
<b>2. Metoda Monte Carlo</b>	<b>2</b>
2.1. Informacje ogólne o metodzie Monte Carlo.....	2
2.2. Historia nazwy.....	2
<b>3. Instrukcja dla użytkownika i opis działania programu</b>	<b>3</b>
3.1. Środowisko programu.....	3
3.2. Program po uruchomieniu.....	3
3.3. Rysowanie wielokąta.....	4
3.4. Przycisk „Monte Carlo”.....	5
3.5. Przycisk „Triangulacja”.....	7
3.6. Przycisk „Reset”.....	8
<b>4. Opis poruszonych problemów i metod użytych do ich rozwiązania</b>	<b>8</b>
4.1. Generowanie liczb losowych.....	8
4.2. Przycinanie się odcinków.....	9
4.3. Rysowanie wielokątów prostych.....	11
4.4. Wzór Meistera-Gaussa na pole powierzchni wielokąta prostego.....	11
4.5. Przynależność punktu do wielokąta.....	12
4.6. Triangulacja wielokątów.....	14
<b>5. Bibliografia</b>	<b>16</b>
<b>6. Kod źródłowy programu</b>	<b>17</b>

## 1. Wstęp

Celem niniejszej pracy jest wyjaśnienie działania napisanego przeze mnie programu edukacyjnego demonstrującego wykorzystanie metody Monte Carlo do obliczania pól powierzchni wielokątów.

Napisanie powyższego programu wymagało stworzenia następujących elementów:

1. Interfejs dający użytkownikowi możliwość rysowania wielokątów za pomocą myszki.
2. Opracowanie i zaimplementowanie algorytmu rozstrzygającego czy zadany punkt płaszczyzny należy do wnętrza wielokąta.
3. Zastosowanie metody Monte Carlo do wyznaczenia pola powierzchni.

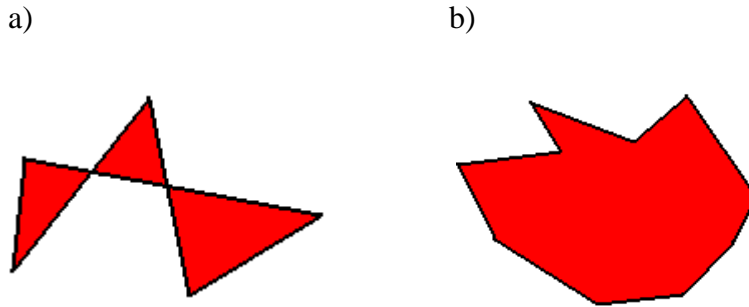
Aby bardziej uściślić problemy, którymi się w tej pracy zajmuję, na wstępie chciałbym zacząć od przedstawienia definicji wielokąta oraz sprecyzować jakiej klasy wielokątami będę się zajmować.

**Wielokąt** to część płaszczyzny ograniczona łamaną zamkniętą, złożoną przynajmniej z trzech odcinków.

Zbiór tak zdefiniowanych wielokątów dzielimy na dwa rozłączne podzbiory:

- wielokąty samoprzecinające się: takie, których przynajmniej jeden bok przecina inny bok tego wielokąta,
- wielokąty proste: takie, które nie są samoprzecinające się.

Działanie mojego programu ograniczyłem do wielokątów prostych, wykluczając możliwość rysowania wielokątów samoprzecinających się.



**Rys. 1.1.** Przykłady wielokątów: a) samoprzecinający się, b) prosty

## **2. Metoda Monte Carlo**

### **2.1. Informacje ogólne o metodzie Monte Carlo**

Metoda Monte Carlo dostarcza przybliżonych rozwiązań różnorodnych problemów matematycznych i fizycznych poprzez wykonywanie próbkowania statystycznego eksperymentów na komputerze [7]. Metoda ta stosuje się zarówno do problemów o probabilistycznym, jak również i nieprobabilistycznym charakterze. Jest ona użyteczna przy rozwiązywaniu problemów zbyt trudnych lub niemożliwych do rozwiązania za pomocą podejścia analitycznego.

### **2.2. Historia nazwy**

W latach 40-tych ubiegłego wieku polski matematyk S. Ulam wraz z J. Von Neumannem, N. Metropolisem i R. Feynmanem w ramach projektu „Manhattan” prowadził pierwsze na dużą skalę symulacje wykorzystujące liczby losowe. Dotyczyły one rozpraszania i absorpcji neutronów. Nadał on nazwę „Monte Carlo” rachunkom opartym o takie właśnie podejście, gdyż kojarzyło mu się ono z jego wujkiem, który grywał w kasynach w Monte Carlo – dzielnicy Monako znanej m.in. właśnie z kasyn. Należy jednak zaznaczyć, że metody Monte Carlo stosowano już znacznie wcześniej – prawdopodobnie pierwsze ich użycie miało miejsce już w 1777 r. przez G. Comte de Buffona.

### **3. Instrukcja dla użytkownika i opis działania programu**

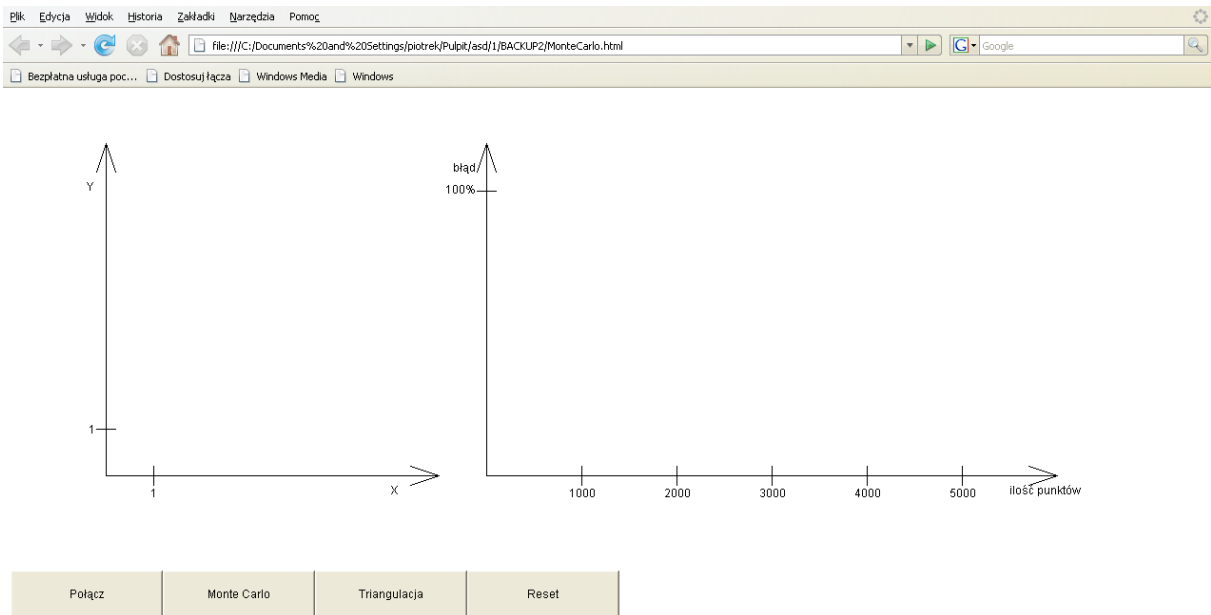
#### **3.1. Środowisko programu**

Program został napisany jako aplet Java. Można go uruchomić przy pomocy dowolnej przeglądarki internetowej po wcześniejszym zainstalowaniu środowiska uruchomieniowego dla programów w Javie czyli tzw. „Java Runtime Environment” (program można pobrać bezpłatnie z Internetu).

#### **3.2. Program po uruchomieniu**

Po uruchomieniu programu widzimy na ekranie dwa układy współrzędnych oraz znajdujące się poniżej nich cztery przyciski.

Układ współrzędnych po lewej stronie stanowi pole do rysowania wielokątów. Na jego osiach pokazana jest skala w przyjętych jednostkach (jedna jednostka = 50 pikseli, czyli jedna jednostka kwadratowa =  $50 \times 50 = 2500$  pikseli).

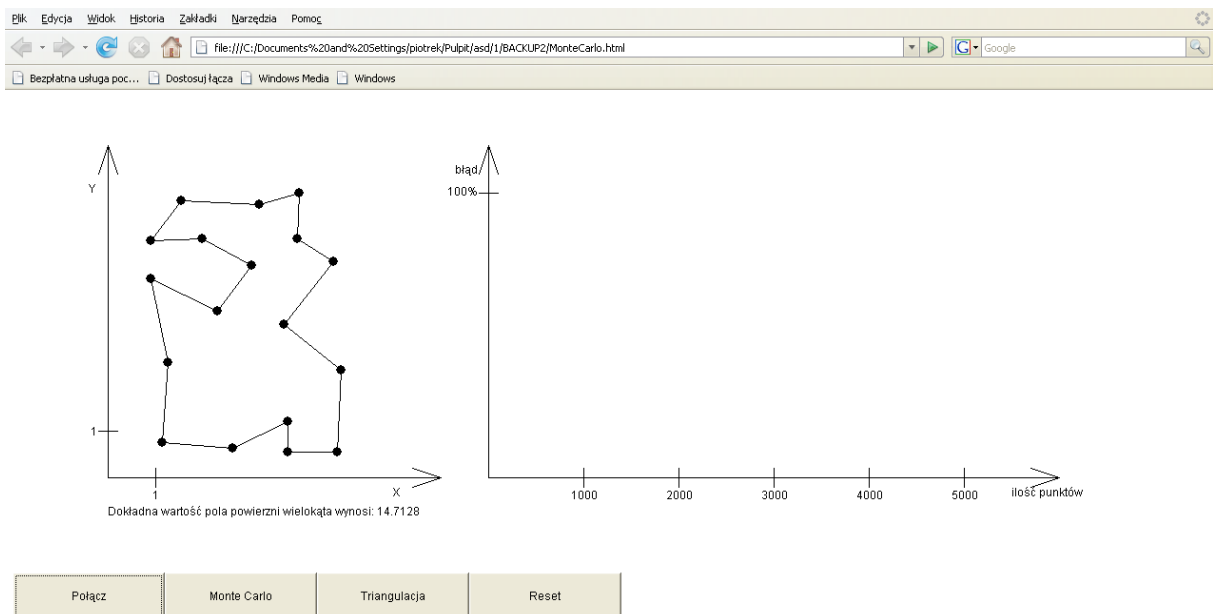


Applet MonteCarlo started

**Rys. 3.1.** Program po uruchomieniu.

### 3.3. Rysowanie wielokąta

Wielokąt rysujemy klikając lewym przyciskiem myszy w wybrane przez nas miejsca. Postawione w ten sposób punkty będą widoczne na ekranie jako małe czarne koła reprezentujące wierzchołki wielokąta. Nasz wielokąt powstaje w wyniku łączenia tych wierzchołków w takiej kolejności w jakiej zostały postawione. Wybór miejsca postawienia wierzchołków z wyjątkiem pierwszego i drugiego nie jest jednak dowolny. Musimy stawiać kolejne wierzchołki w taki sposób, aby nowo powstałe boki nie przecinały już istniejących. Po wprowadzeniu wszystkich wierzchołków klikamy przycisk **Połącz** aby połączyć wierzchołek pierwszy z ostatnim i zakończyć rysowanie. Poniżej układu współrzędnych zostanie wyświetlona wartość pola powierzchni narysowanego wielokąta, obliczona ze wzoru Meistersa-Gaussa [2].



Applet: MonteCarlo started

**Rys. 3.2.** Program po wprowadzeniu wierzchołków wielokąta i kliknięciu w przycisk **Połącz**

### 3.4. Przycisk „Monte Carlo”

Po narysowaniu zadanego przez nas wielokąta wciskamy przycisk **Monte Carlo**. Program wylosuje wtedy 5000 punktów w obszarze układu współrzędnych, który znajduje się po lewej stronie (obszar ten ma  $6 \times 6$  jednostek = 36 jednostek kwadratowych). Każdy z tych punktów zostanie sprawdzony przy pomocy odpowiedniego algorytmu (opisanego w dalszej części pracy), czy leży wewnątrz, czy na zewnątrz narysowanego wielokąta. Wszystkie punkty będą widoczne na ekranie jako małe punkciki o rozmiarach jednego piksela. Te z nich, które zostały sklasyfikowane, jako leżące wewnątrz narysowanego wielokąta, będą miały czerwony kolor, natomiast punkty uznane za leżące na zewnątrz - kolor niebieski. Współrzędne  $x$  i  $y$  punktów są losowane spośród liczb rzeczywistych (a dokładniej - zmiennoprzecinkowych) z zakresu od 0 do 300 (obszar losowania ma  $300 \times 300$  pikseli). Przedstawienie tych punktów wymaga pewnego przybliżenia, ponieważ nie możemy przedstawić liczb rzeczywistych na ekranie monitora. Dlatego też współrzędne punktów wyrażone w pikselach są zaokrąglane do liczb całkowitych. Zaokrąglenie to jest jedynie na potrzeby wizualizacji punktów na ekranie –

same obliczenia są wykonywane na współrzędnych rzeczywistych. Pole powierzchni narysowanego wielokąta jest szacowane według wzoru:

$$P = \frac{n}{N} \times S,$$

gdzie:

n – liczba punktów sklasyfikowanych jako leżące wewnątrz wielokąta,

N – liczba wszystkich wylosowanych punktów,

S – powierzchnia całego obszaru losowania (w naszym przypadku 36 jednostek kwadratowych).

Powyższy wzór wynika z następującej obserwacji: prawdopodobieństwo wylosowania punktu wewnątrz dowolnej figury na płaszczyźnie (w naszym przypadku wielokąta) jest wprost proporcjonalne do stosunku pola powierzchni tej figury do pola powierzchni całego obszaru losowania.

Obliczona przy pomocy tego wzoru przybliżona wartość pola powierzchni wielokąta będzie widoczna na dole ekranu.

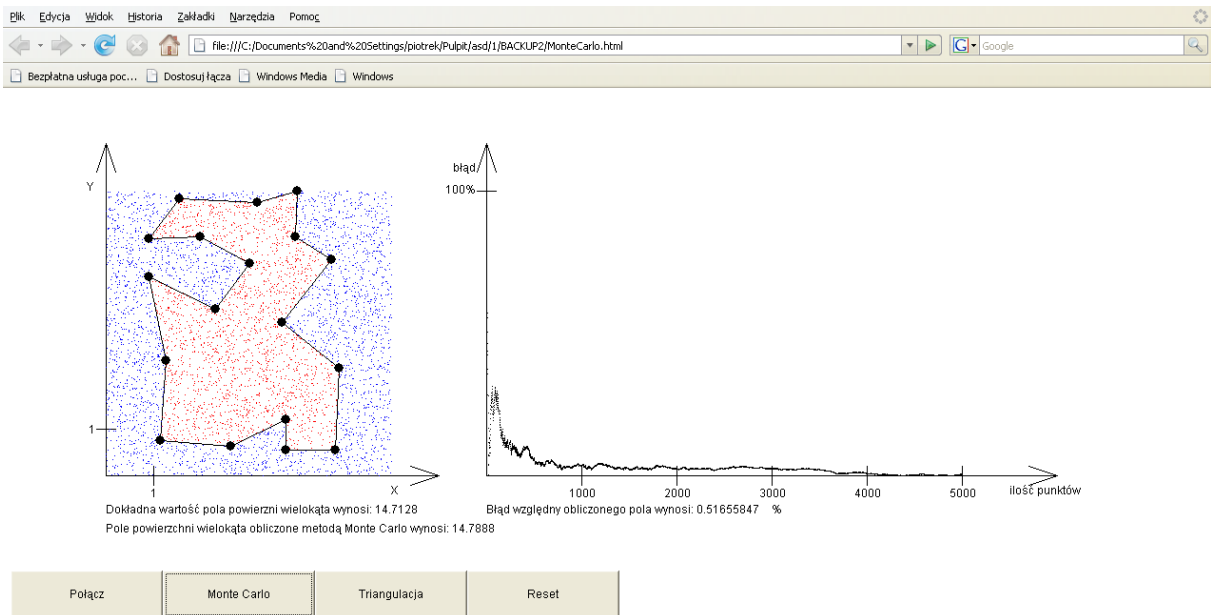
W układzie współrzędnych po prawej stronie natomiast zostanie wyświetlony wykres. Przedstawia on zależność procentowego błędu względnego pola powierzchni obliczonego przy użyciu metody Monte Carlo, od ilości losowanych punktów. Poniżej wykresu będzie widoczna wartość procentowa tego błędu dla 5000 punktów. Wartość tego błędu jest obliczana zgodnie ze wzorem:

$$\delta = \frac{|P_d - P_{mc}|}{P_d},$$

gdzie:

$P_{mc}$  – pole powierzchni wielokąta obliczone metodą Monte Carlo,

$P_d$  – pole powierzchni wielokąta obliczone metodą analityczną(dokładną).



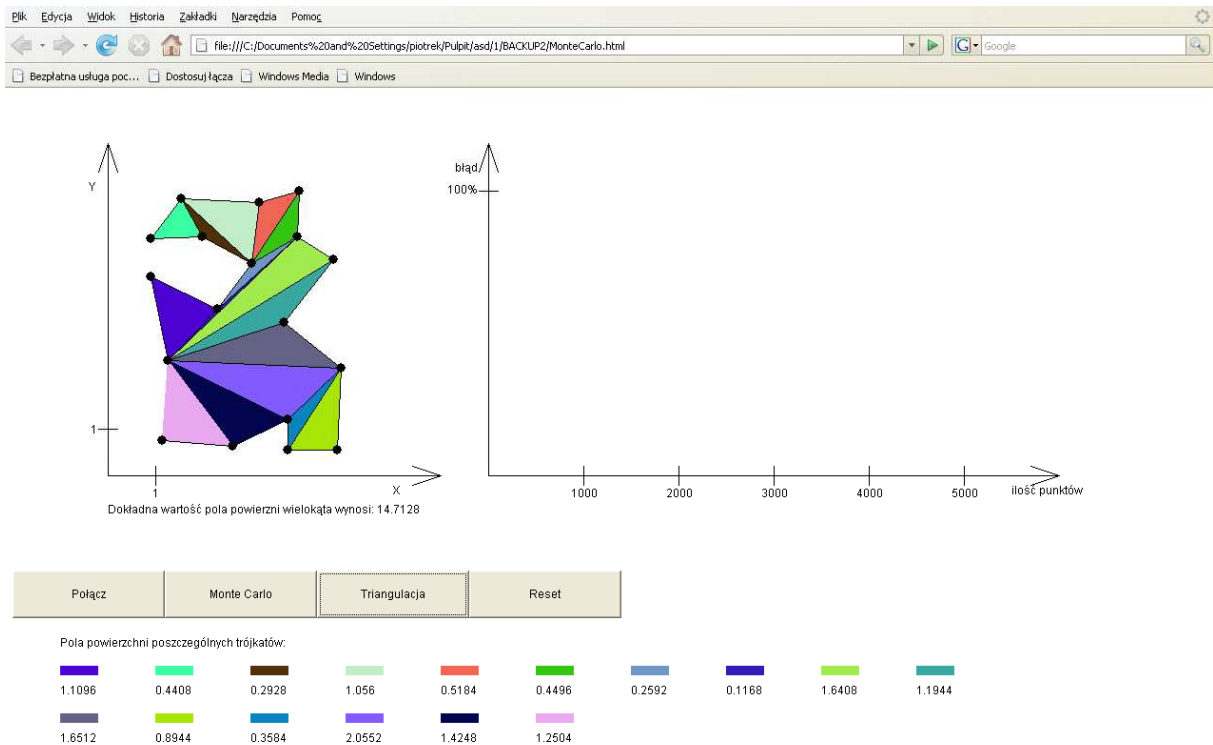
**Rys. 3.3.** Program po wprowadzeniu wierzchołków wielokąta i kliknięciu kolejno w przycisk **Połącz** i przycisk **Monte Carlo**.

### 3.5. Przycisk „Triangulacja”

Po wciśnięciu przycisku **Triangulacja** nasz wielokąt zostanie graficznie podzielony na trójkąty. Każdy trójkąt zostanie wypełniony losowym kolorem (niestety niektóre kolory mogą być bardzo podobne). Ponadto na dole ekranu zostanie wyświetlony zbiór kolorowych prostokątów. Pod każdym z prostokątów będzie widoczna liczba, która wyraża pole tego trójkąta, którego kolor jest identyczny z kolorem prostokąta, pod którym liczba się znajduje. Prostokąty są wyświetlane w takiej kolejności, w jakiej powstawały trójkąty w wyniku triangulacji. Pola powierzchni trójkątów obliczamy według wzoru:

$$P = \frac{1}{2} \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix} = \frac{1}{2} |x_A y_B + x_B y_C + x_C y_A - x_C y_B - y_C x_A - x_B y_A|$$

gdzie:  $A=(x_A;y_A)$  ,  $B=(x_B;y_B)$ ,  $C=(x_C;y_C)$  to współrzędne wierzchołków



Applet: MonteCarlo started

**Rys. 3.4.** Program po wprowadzeniu wierzchołków wielokąta i kliknięciu kolejno w przycisk **Połącz** i przycisk **Triangulacja**.

### 3.6. Przycisk „Reset”

Wciśnięcie przycisku **Reset** powoduje uruchomienie programu od nowa. Możemy wówczas narysować inny interesujący nas wielokąt.

## 4. Opis poruszonych problemów i metod użytych do ich rozwiązania

### 4.1. Generowanie liczb losowych

Współczesne metody generowania liczb losowych można podzielić na dwie grupy. Jedną z nich to wykorzystanie algorytmów matematycznych (albo ich sprzętowej realizacji), czyli metod które są powtarzalne (ten sam ciąg liczb pseudolosowych, można otrzymać wielokrotnie, powtarzając przebieg algorytmu z tymi samymi parametrami). Drugą grupę metod stanowią generatory, w których proces wytwarzania ciągu liczb

losowych polega na zamianie na liczby mierzonych parametrów losowego procesu fizycznego. W tym przypadku zarówno powtórzenie trajektorii procesu, jak i powtórne uzyskanie identycznego ciągu liczb losowych nie jest możliwe. W moim programie do generowania liczb pseudolosowych wykorzystałem gotowy generator, który znajduje się w klasie `Random` pakietu `java.util` standardowo dostępnego w JDK. Powyższy generator wykorzystuje liniowy algorytm kongruentny. Szczegóły algorytmu można znaleźć w [6]. Do otrzymywania liczb zmiennoprzecinkowych typu `float` w Javie przy użyciu powyższego generatora służy metoda `nextFloat`.

#### 4.2. Przycinanie się odcinków

Niech dane będą punkty na płaszczyźnie:  $A=(x_A;y_A)$ ,  $B=(x_B;y_B)$ ,  $C=(x_C;y_C)$

Zdefiniujmy następujący wyznacznik:

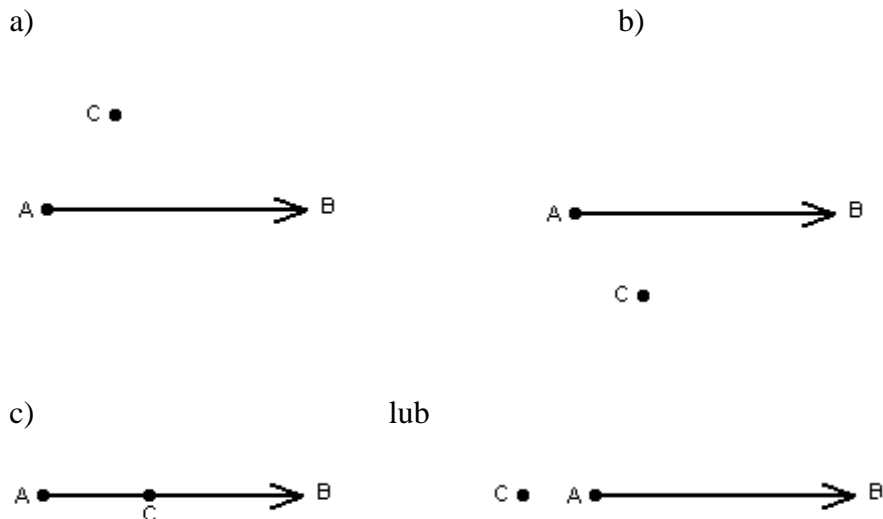
$$\det(A, B, C) = \det \begin{bmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{bmatrix}$$

Ma on następującą interesującą nas własność:

Jeśli  $\det(A,B,C) > 0$ , to punkt C znajduje się po lewej stronie wektora  $\overrightarrow{AB}$ .

Jeśli  $\det(A,B,C) < 0$ , to punkt C znajduje się po prawej stronie wektora  $\overrightarrow{AB}$ .

Jeśli  $\det(A,B,C) = 0$ , to punkty A, B i C są współliniowe.



**Rys. 4.1.** Położenie punktu C względem wektora AB, a wyznacznik  $\det(A, B, C)$ :

a)  $\det(A, B, C) > 0$  ; b)  $\det(A, B, C) < 0$  ;  $\det(A, B, C) = 0$

Możemy wykorzystać tę własność do sprawdzenia, czy dane odcinki  $\overline{AB}$  i  $\overline{CD}$  się przecinają, zauważając że:

Jeśli odcinki  $\overline{AB}$  i  $\overline{CD}$  się przecinają, to możliwe są jedynie następujące przypadki:

- 1) Punkty A i B leżą po przeciwnych stronach odcinka  $\overline{CD}$  oraz punkty C i D leżą po przeciwnych stronach odcinka  $\overline{AB}$ <sup>1</sup>.
- 2) Któryś z końców jednego z odcinków należy do drugiego odcinka.

Przypadek drugi możemy dla prostoty pominąć, ponieważ występuje on bardzo rzadko ze względu na fakt że współrzędne punktów są losowane ze zbioru liczb rzeczywistych.

Przypadek pierwszy występuje tylko wtedy, gdy spełnione są równocześnie następujące warunki:

- znak  $\det(A, B, C)$  jest różny od znaku  $\det(A, B, D)$ ,
- znak  $\det(C, D, A)$  jest różny od znaku  $\det(C, D, B)$ .

<sup>1</sup> Będąc ścisłym matematycznie należałoby powiedzieć, że punkty leżą po przeciwnych stronach prostej przechodzącej przez dany odcinek.

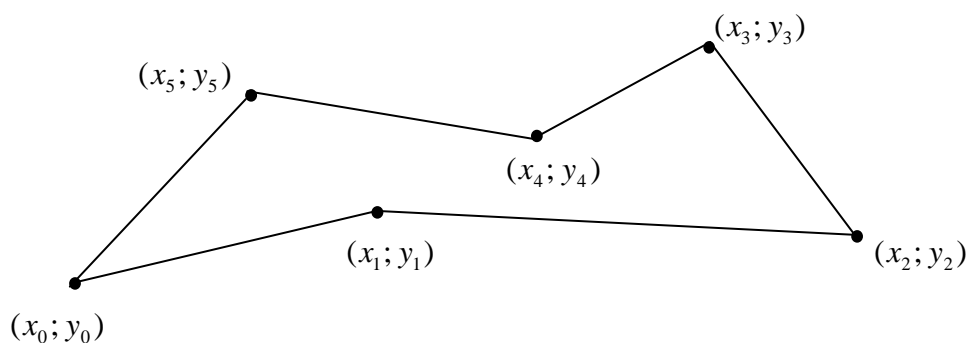
### 4.3. Rysowanie wielokątów prostych

Kliknięcie myszką w obszarze rysowania powoduje zapisanie współrzędnych tego punktu do dwóch tablic – współrzędnych  $x$  i  $y$ : `xwierzch[]` i `ywierzch[]`. Indeks  $d$  numeruje wierzchołki, zaczynając od „0” – pierwszy wierzchołek ma współrzędne  $(xwierzch[0];ywierzch[0])$ , drugi:  $(xwierzch[1];ywierzch[1])$ , itd. Każdy kolejny punkt jest sprawdzany, czy odcinek łączący ten punkt z punktem poprzednim nie przecina któregoś z poprzednich boków wielokąta (funkcja `czyсиеprzecinazpoprzednimi`). Jeśli nie przecina, wówczas dany punkt zostanie zatwierdzony i pojawia się na ekranie, a jego współrzędne zostaną zapisane w tablicy. Jeśli przecina, wówczas punkt nie zostanie zatwierdzony, tzn. „kliknięcie” nie wywołuje żadnego skutku. W ten sposób ograniczamy się tylko do rysowania wielokątów prostych.

### 4.4. Wzór Meistersa-Gaussa na pole powierzchni wielokąta prostego

Mając uporządkowany zgodnie lub przeciwnie do ruchu wskazówek zegara zbiór współrzędnych wierzchołków  $(x_0; y_0), (x_1; y_1), (x_2; y_2) \dots (x_{n-1}; y_{n-1})$   $n$ -kąta prostego, możemy obliczyć pole jego powierzchni korzystając ze wzoru Meistersa-Gaussa [2]:

$$P = \left| \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \right|$$



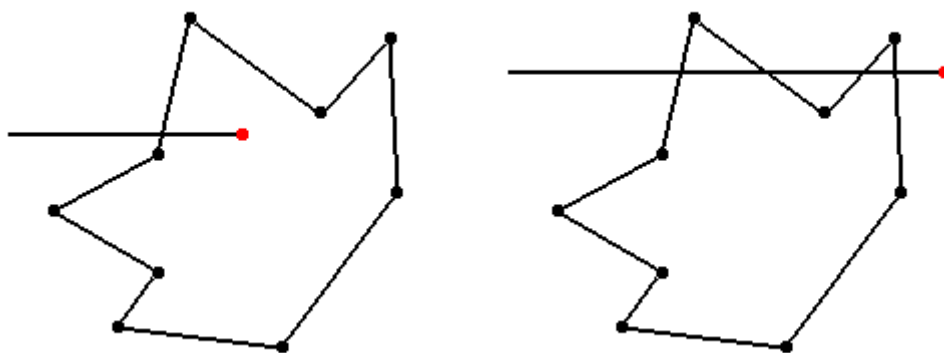
**Rys 4.2.** Numeracja współrzędnych wierzchołków sześciokąta w kierunku odwrotnym do ruchu wskazówek zegara.

#### 4.5. Przynależność punktu do wielokąta

Aby rozstrzygnąć, czy dany punkt leży wewnątrz wielokąta, zastosowałem tzw. algorytm parzystości [1]. Istnieje wiele innych, często szybszych, ale i trudniejszych do implementacji algorytmów sprawdzających przynależność punktu do wielokąta. Opis niektórych z nich można znaleźć w artykule [5]. Algorytm parzystości działa w oparciu o następujące spostrzeżenie:

Jeśli z danego punktu poprowadzimy dowolną półprostą, to punkt ten leży wewnątrz wielokąta, gdy ta półprosta przecina boki wielokąta nieparzystą ilość razy. W przeciwnym razie, tzn. gdy półprosta przecina boki wielokąta parzystą ilość razy, analizowany punkt leży na zewnątrz wielokąta.

W szczególności możemy zatem wziąć półprostą poziomą wychodzącą w lewą stronę od danego punktu. W naszym programie jednak wystarczy wziąć tylko jej kawałek - odcinek na tyle długi, aby kończył się poza obszarem rysowania. Następnie używamy wcześniej już napisanej funkcji `czySiePrzecina` dla każdego boku wielokąta w zestawieniu z tym właśnie odcinkiem. Za każdym razem, gdy przecięcie występuje, zwiększamy o jeden zmienną `ilazPrzecina`. Po zliczeniu wszystkich przecięć funkcja `czyNależy` zwraca prawdę, gdy reszta z dzielenia zmiennej `ilazPrzecina` przez dwa wynosi jeden oraz fałsz w przypadku przeciwnym.

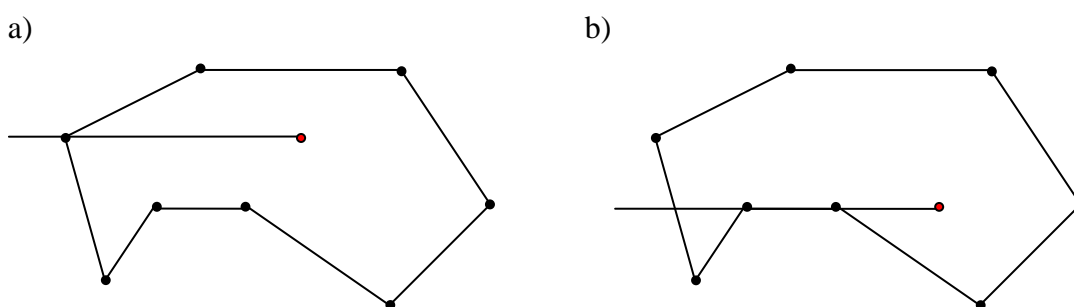


**Rys 4.3.** Położenie punktu względem wielokąta, a liczba przecięć półprostej poprowadzonej z tego punktu z bokami wielokąta: a) nieparzysta liczba przecięć – punkt należy do wielokąta  
b) parzysta liczba przecięć – punkt nie należy do wielokąta

Istnieje kilka szczególnych przypadków, w których powyższy algorytm bez odpowiednich modyfikacji zawodzi:

- 1) Półprosta przechodzi przez jeden lub więcej wierzchołków wielokąta.
- 2) Przynajmniej jeden bok wielokąta leży na półprostej.

W pierwszym przypadku punkt przecięcia byłby policzony podwójnie<sup>2</sup>, ponieważ punkt będący wierzchołkiem należy do obu sąsiednich boków. W rezultacie punkt zostałby nieprawidłowo sklasyfikowany. W drugim przypadku punkt przecięcia byłby liczony potrójnie, co może w niektórych przypadkach zaowocować błędnym zakwalifikowaniem punktu.



**Rys. 4.4.** Przypadki szczególne w algorytmie parzystości: a) półprosta przechodzi przez wierzchołek wielokąta, b) półprosta przechodzi przez bok wielokąta.

Może się zdarzyć również sytuacja, że badany punkt znajduje się na jednym z boków wielokąta. Gdyby zastosowany algorytm na przecinanie się odcinków obejmował przypadek, gdy koniec jednego odcinka leży na drugim odcinku, wówczas punkty leżące na bokach wielokąta byłyby klasyfikowane prawidłowo.

Pominięcie tych wszystkich przypadków szczególnych nie ma znaczenia dla działania programu, dzięki temu, że losowanie punktów odbywa się na zbiorze liczb rzeczywistych. Prawdopodobieństwo wylosowania punktu takiego, że półprosta wychodząca z niego będzie przecinała wierzchołek lub przechodziła przez bok wielokąta wynosi wtedy 0, zatem nie ma sensu rozpatrywanie takich sytuacji<sup>3</sup>.

<sup>2</sup> Ściślej mówiąc byłby sklasyfikowany podwójnie, gdyby nasz algorytm na przecinanie się odcinków działał również w przypadku, gdy jeden z końców jednego odcinka leży na drugim odcinku. Z powodu pominięcia takiego przypadku, w moim programie punkt przecięcia nie byłby policzony wcale, co daje w praktyce ten sam efekt – punkt zostaje błędnie sklasyfikowany.

<sup>3</sup> W praktyce na komputerze nie mamy możliwości losowania liczb ze zbioru liczb rzeczywistych, a jedynie liczb typu zmiennoprzecinkowego, których ilość jest skończona. Ściślej więc mówiąc prawdopodobieństwo to jest bardzo bliskie zera, ale większe od zera. Jest ono jednak tak małe, że możemy je zaniedbać.

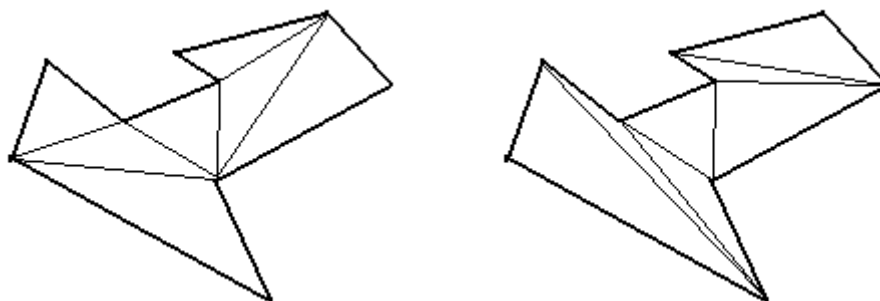
## 4.6. Triangulacja wielokątów

Triangulacja jest podstawowym zagadnieniem w geometrii obliczeniowej, ponieważ pierwszym krokiem przy badaniu skomplikowanych obiektów geometrycznych jest rozbicie ich na obiekty prostsze. Najprostszymi obiektami są trójkąty w przestrzeni dwuwymiarowej i czworościany w przestrzeni trójwymiarowej. W zależności od zastosowań triangulacje można definiować na różne sposoby.

Na potrzeby mojego programu, przyjąłem następującą definicję [1]:

**Triangulacja** jest to podział wielokąta zwykłego na sumę nie nakładających się na siebie trójkątów, których wierzchołkami mogą być tylko wierzchołki danego wielokąta.

Tak zdefiniowany podział  $n$ -kąta prowadzi zawsze do  $n-2$  trójkątów. Nie jest to jednak podział jednoznaczny, bowiem przy tak przyjętej definicji triangulację jednego wielokąta możemy wykonać na wiele różnych sposobów (patrz rys 3.1).



Rys 4.4. Różne triangulacje tego samego wielokąta.

Obecnie znanych jest wiele algorytmów triangulacji wielokątów, różniących się złożonością obliczeniową oraz tym, do jakich klas wielokątów można je stosować. W moim programie zastosowałem algorytm znany pod nazwą „Triangulation by ear clipping”. Postaram się możliwie szczegółowo wyjaśnić jego działanie.

Ucha występujące w nazwie tego algorytmu są to trójkąty, których dwa boki są również bokami danego wielokąta, a ich trzeci bok zawiera się wewnątrz wielokąta. Każdy wielokąt nie będący trójkątem ma przynajmniej dwa „ucha”. Algorytm działa w sposób następujący:

Algorytm znajduje „ucho” i „obcina” je, w rezultacie czego otrzymujemy mniejszy wielokąt mający o jeden wierzchołek mniej. Następnie powtarzamy procedurę dla tego nowego wielokąta, w wyniku czego dostajemy kolejny wielokąt itd. Schemat ten kontynuujemy aż do momentu, gdy zostanie nam już tylko trójkąt. Aby sprawdzić czy dany trójkąt o kolejnych wierzchołkach  $w[i]$ ,  $w[i+1]$ ,  $w[i+2]$  w wielokącie jest „uchem”, sprawdzamy czy można poprowadzić przekątną z wierzchołka  $w[i]$  do  $w[i+2]$ . Dany odcinek łączący dwa wierzchołki wielokąta jest jego przekątną, jeśli spełnia dwa następujące warunki:

1. Nie przecina żadnego z boków wielokąta.
2. Znajduje się wewnątrz tego wielokąta.

Aby sprawdzić warunek 1. , stosujemy algorytm rozstrzygający, czy dane dwa odcinki przecinają się (w programie robi to funkcja `czySiePrzecinaja`). Sprawdzenie to wykonujemy dla odcinka o końcach  $w[i]$ ,  $w[i+2]$  w zestawieniu kolejno z każdym bokiem wielokąta (funkcja `czyPrzecinaKtorysBok`).

Warunek 2. z kolei sprawdzamy, wykorzystując następujące spostrzeżenie: jeśli każdy punkt odcinka, nie będący jego końcem leży wewnątrz danego wielokąta to odcinek ten zawiera się w tym wielokącie. Prawdziwość koniunkcji warunków 1. i 2. wymaga aby sprawdzany odcinek nie przecinał żadnego boku wielokąta (odcinki leżące częściowo wewnątrz i częściowo na zewnątrz nigdy nie spełnią warunku 1. , więc możemy pominąć ich rozpatrywanie). Wystarczy więc sprawdzić przynależność jednego punktu leżącego na odcinku, aby stwierdzić czy odcinek ten leży w całości wewnątrz czy w całości na zewnątrz wielokąta. W szczególności możemy łatwo sprawdzić przynależność punktu środkowego danego odcinka, używając do tego celu wcześniej już wykorzystywaną funkcję `czyNalezy`. Sprawdzamy powyższe warunki w pętli dla kolejnych trójek  $w[i]$ ,  $w[i+1]$ ,  $w[i+2]$  następujących po sobie wierzchołków wielokąta, aż napotkamy na taką, która spełnia obydwie warunki. Jeśli trójkąt o wierzchołkach  $w[i]$ ,  $w[i+1]$ ,  $w[i+2]$  okazuje się być „uchem”, odcinamy go od wielokąta poprzez wyrzucenie wierzchołka  $w[i+1]$  z tablicy wierzchołków wielokąta oraz przesunięcie wszystkich następujących po nim wierzchołków o jedno miejsce w tablicy w tył. Następnie całą procedurę powtarzamy dla nowego wielokąta, aż do momentu, w którym nowo powstały wielokąt jest trójkątem.

## 5. Bibliografia

- [1] Michał Jankowski, „Elementy grafiki komputerowej”  
Wydawnictwa Naukowo-Techniczne, Warszawa (2006)
- [2] Paul Bourke, “Calculating the area and centroid of a polygon”  
<http://local.wasp.uwa.edu.au/~pbourke/geometry/polyarea/>
- [3] Wykorzystane algorytmy „Współliniowość trzech punktów”,  
„Przecinanie się odcinków”, „Przynależność punktu do wielokąta” znajdują się na  
stronie <http://www.algorytm.org> (rozdział: Algorytmy/Geometria obliczeniowa,  
autor: Michał Knasiecki)
- [4] David Eberly, “Triangulation by ear clipping”  
<http://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf>
- [5] Eric Haines, “Point in polygon strategies”  
<http://www.erichaines.com/ptinpoly/>
- [6] Donald Knuth, „Sztuka programowania tom II : Algorytmy seminumeryczne”  
Wydawnictwa Naukowo-Techniczne, Warszawa (2002)
- [7] Sabri Pllana, „History of Monte Carlo method”  
<http://www.geocities.com/CollegePark/Quad/2435/history.html>

## 6. Kod źródłowy programu

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import java.applet.*;

public class MonteCarlo extends Applet implements MouseListener{
public int n=0,N=0,z,s;
public float x,y;
public float POLE,Poletriang=0,blad,usersPoletriang;
public boolean czynalezy=false,polacz=false,
wyswietl=false,pp=false,pokaztriang;
public Button Polacz,MonteCarlo,Triangulacja,Reset;;
public int bladrys,m,d=-1,lw,i;
public int xwierzch[]=new int[100];
public int ywierzch[]=new int[100];
public int xw[]=new int[100];
public int yw[]=new int[100];
public int xt[]=new int[3];
public int yt[]=new int[3];
public int R[]=new int[100];
public int G[]=new int[100];
public int B[]=new int[100];
public float polatrojkatow[]=new float[100];

public void init(){
    setLayout(null);

    ActionListener al=new ActionListener(){
        public void actionPerformed(ActionEvent e){
            if(e.getSource()==Polacz&&polacz==false&&d>=2)
            {
                polacz=true;
                repaint();
            }
            if(e.getSource()==MonteCarlo&&polacz)
            {
                n=0;
                N=0;
                lw=d+1;

                for(i=0;i<=d+2;i++)
                {
                    xw[i]=xwierzch[i];
                    yw[i]=ywierzch[i];
                }
                wyswietl=!wyswietl;
                repaint();
            }
            if(e.getSource()==Reset)
            {
                d=-1;
                polacz=false;
                pokaztriang=false;
                polacz=false;
                pp=false;
                wyswietl=false;
                for(i=0;i<=99;i++)
```

```

        {
            xwierzch[i]=0;
            ywierzch[i]=0;
        }
        repaint();
    }
    if(e.getSource()==Triangulacja&&polacz)
    {
        n=0;
        N=0;
        lw=d+1;
        pokaztriang=true;

        for(i=0;i<=d+2;i++)
        {
            xw[i]=xwierzch[i];
            yw[i]=ywierzch[i];
        }
        repaint();
    }
}
};

```

```

MonteCarlo=new Button("Monte Carlo");
MonteCarlo.setBounds(160,500,160,50);
MonteCarlo.addActionListener(al);
add(MonteCarlo);

```

```

Triangulacja=new Button("Triangulacja");
Triangulacja.setBounds(320,500,160,50);
Triangulacja.addActionListener(al);
add(Triangulacja);

```

```

Reset=new Button("Reset");
Reset.setBounds(480,500,160,50);
Reset.addActionListener(al);
add(Reset);

```

```

Polacz=new Button("Połącz");
Polacz.setBounds(0,500,160,50);
Polacz.addActionListener(al);
add(Polacz);
addMouseListener(this);
}

```

```

boolean czysieprzecinaja(float xA,float xB,float xC,float xD,float
yA,float yB,float yC,float yD)
{
    if
        (wyznacznik(xA,yA,xB,yB,xC,yC)*wyznacznik(xA,yA,xB,yB,xD,yD)<0&&
wyznacznik(xC,yC,xD,yD,xA,yA)*wyznacznik(xC,yC,xD,yD,xB,yB)<0)
        {
            return true;
        }
    else
        {
            return false;
        }
}

```

```

float wyznacznik(float xx,float xy,float yx,float yy,float zx,
float zy)

```

```

    {
        return (xx*yy+yx*zy+zx*xy-zx*yy-xx*zy-yx*xy);
    }
}
boolean czysieprzecinazpoprzednimi(int e)
{
    int i,licznik=0;
    for(i=2;i<e;i++)
    {
        if(czysieprzecinaja(xwierzch[e],xwierzch[e-1],
        xwierzch[e-i],xwierzch[e-i-1],ywierzch[e],
        ywierzch[e-1],ywierzch[e-i],ywierzch[e-i-1]))
        {
            licznik++;
        }
    }
    if(licznik>0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

boolean czynalezy(float x,float y)
{
    int ilerazyprzecina=0;
    xwierzch[d+1]=xwierzch[0];
    ywierzch[d+1]=ywierzch[0];
    for(i=0;i<=d;i++)
    {
        if(czysieprzecinaja(0,x,xwierzch[i],xwierzch[i+1],
        y,y,ywierzch[i],ywierzch[i+1]))
        {
            ilerazyprzecina++;
        }
    }
    if(ilerazyprzecina%2==1)
    {
        return true;
    }
    else
    {
        return false;
    }
}

boolean czyprzecinaktorysbok(int xA, int yA,int xB, int yB)
{
    boolean cp=false;
    int i;
    xwierzch[d+1]=xwierzch[0];
    ywierzch[d+1]=ywierzch[0];
    for(i=0;i<=d;i++)
    {
        if(czysieprzecinaja(xA,xB,xwierzch[i],xwierzch[i+1],
        yA,yB,ywierzch[i],ywierzch[i+1]))
        {
            cp=true;
        }
    }
}

```

```

    }
}
return cp;
}
float Polezewzoru(int d)
{
    int i;
    float Suma=0,Poledokladne=0;
    for(i=0;i<=d;i++)
    {
        Suma=Suma+xwierzch[i]*ywierzch[i+1]-
            xwierzch[i+1]*ywierzch[i];
    }
    Poledokladne=Math.abs(Suma/5000);
    return Poledokladne;
}

Color kolor(int c)
{
    return(new Color(R[c],G[c],B[c]));
}

public void paint (Graphics g){

if(polacz)
{
    if(czyprzecinaktorysbok(xwierzch[0],ywierzch[0],
        xwierzch[d],ywierzch[d])==false)
    {
        g.drawLine(xwierzch[0],ywierzch[0],xwierzch[d],ywierzch[d]);
        g.drawString("Dokładna wartość pola powierzchni wielokąta
            wynosi: "+Polezewzoru(d), 100, 440);
        polacz=true;
    }
    else
    {
        polacz=false;
    }
}

g.drawLine (100,400,450,400);
g.drawLine (450,400,420,410);
g.drawLine (450,400,420,390);
g.drawString("X", 400, 420);
g.drawLine (100,400,100,50);
g.drawLine (100,50,90,80);
g.drawLine (100,50,110,80);
g.drawString("Y",80, 100);

g.drawLine (500,400,1100,400);
g.drawLine (1100,400,1070,410);
g.drawLine (1100,400,1070,390);
g.drawString("ilość punktów", 1050, 420);
g.drawLine (500,400,500,50);
g.drawLine (500,50,490,80);
g.drawLine (500,50,510,80);
g.drawString("błąd",465, 80);

g.drawLine (150,390,150,410);

```

```

g.drawString("1", 147, 423);
g.drawLine (90,350,110,350);
g.drawString("1", 82, 355);
g.drawLine (490,100,510,100);
g.drawString("100%", 457, 103);
g.drawLine (600,390,600,410);
g.drawLine (700,390,700,410);
g.drawLine (800,390,800,410);
g.drawLine (900,390,900,410);
g.drawLine (1000,390,1000,410);
g.drawString("1000", 587, 423);
g.drawString("2000", 687, 423);
g.drawString("3000", 787, 423);
g.drawString("4000", 887, 423);
g.drawString("5000", 987, 423);

Random generator = new Random();

int k,j,i=0;

if(pokaztriang&&polacz)
{
    i=0;
    int c=-1;
    xw[lw]=xw[0];
    yw[lw]=yw[0];
    xw[lw+1]=xw[1];
    yw[lw+1]=yw[1];
    while(lw>3)
    {
        for(i=0;i<lw;i++)
        {
            if(czyprzecinaktorysbok(xw[i],yw[i],xw[i+2],yw[i+2])==false
&&czynalezy((xw[i]+xw[i+2])/2,(yw[i]+yw[i+2])/2))
            {
                for(k=0;k<=2;k++)
                {
                    xt[k]=xw[i+k];
                    yt[k]=yw[i+k];
                }
                c++;
                R[c]=generator.nextInt(256);
                G[c]=generator.nextInt(256);
                B[c]=generator.nextInt(256);

                polatrojkatow[c]=Math.abs(1F*(xw[i]*yw[i+1]+
xw[i+1]*yw[i+2]+xw[i+2]*yw[i]-xw[i+2]*yw[i+1]-
xw[i]*yw[i+2]-xw[i+1]*yw[i])/5000);

                g.setColor(kolor(c));
                g.fillPolygon(xt,yt,3);
                g.setColor(Color.BLACK);
                g.drawPolygon(xt,yt,3);
                for(j=i;j<lw;j++)
                {
                    xw[j+1]=xw[j+2];
                    yw[j+1]=yw[j+2];
                }
                lw--;
                break;
            }
        }
    }
}

```

```

    }
  }
}
if(lw==3)
{
  for(k=0;k<=2;k++)
  {
    xt[k]=xw[i+k];
    yt[k]=yw[i+k];
  }
  R[c+1]=generator.nextInt(256);
  G[c+1]=generator.nextInt(256);
  B[c+1]=generator.nextInt(256);

  polatrojkatow[c+1]=Math.abs((1F*xw[i]*yw[i+1]+
xw[i+1]*yw[i+2]+xw[i+2]*yw[i]-xw[i+2]*yw[i+1]-
xw[i]*yw[i+2]-xw[i+1]*yw[i])/5000);

  g.setColor(kolor(c+1));
  g.fillPolygon(xt,yt,3);
  g.drawPolygon(xt,yt,3);
  g.setColor(Color.BLACK);
}
}

if(wyswietl&&polacz)
{
  for(k=0;k<5000;k++)
  {
    N++;
    x = generator.nextFloat()*300+100;
    y = generator.nextFloat()*300+100;
    z=Math.round(x);
    s=Math.round(y);

    if(czynalezy(x,y)==true)
    {
      g.setColor(Color.RED);
      n++;
    }
    if(czynalezy(x,y)==false)
    {
      g.setColor(Color.BLUE);
    }
    g.drawLine (z,s,z,s);
    g.setColor(Color.BLACK);

    POLE=36F*n/N;
    usersPoletriang=Poletriang/2500;
    blad=Math.abs((POLE-Polezewzoru(d))/Polezewzoru(d));
    bladrys=Math.round(blad*300);
    g.drawLine(Math.round(N/10)+500,400-bladrys,
    Math.round(N/10)+500,400-bladrys);
  }
  g.drawString("Pole powierzchni wielokąta obliczone metodą Monte
Carlo wynosi: "+POLE, 100, 460);
  g.drawString("Błąd względny obliczonego pola wynosi: "+blad*100,
500, 440);
  g.drawString("%", 800, 440);
}

```

```

}
if(pp)
{
    g.fillOval(xwierzch[0]-5, ywierzch[0]-5, 10, 10);
}

for(m=1;m<=d;m++){
    g.drawLine(xwierzch[m],ywierzch[m],xwierzch[m-1],ywierzch[m-1]);
    g.fillOval(xwierzch[m]-5, ywierzch[m]-5, 10, 10);
}

if(pokaztriang)
{
    g.drawString("Pola powierzchni poszczególnych
    trójkątów:",50,580);
    int l=(d-2)%10;
    int b=0;
    for(i=0;i<=l;i++)
    {
        for(j=i*10;j<i*10+10&&b<=(d-2);j++)
        {
            b++;
            g.setColor(kolor(j));
            g.fillRect(50+100*j-i*1000,600+50*i,40,10);
            g.setColor(Color.BLACK);
            g.drawString(""+polatrojkatow[j],50+100*j-
            i*1000,630+50*i);
        }
    }
}

public void mouseClicked(MouseEvent e){}
public void mouseEntered(MouseEvent e){}
public void mouseExited(MouseEvent e){}
public void mousePressed(MouseEvent e){
x=e.getX();
y=e.getY();
if(100<=x&&x<=400&&100<=y&&y<=400&&polacz==false)
{
    pp=true;

    d++;
    xwierzch[d]=e.getX();
    ywierzch[d]=e.getY();

    if(d<=2)
    {
        repaint();
    }

    else if(czysieprzecinazpoprzednimi(d)==false)
    {
        repaint();
    }
    else
    {
        d--;
    }
}
}

```

```
}  
public void mouseReleased(MouseEvent e){  
}
```