# Zastosowanie pakietu Geant4 w fizyce jądrowej

Aleksandra Fijałkowska

Wydział Fizyki, Uniwersytet Warszawski

*aleksandra.fijalkowska@fuw.edu.pl*

5 marca 2020

# Organizacja zajęć

**Miejsce spotkań:** czwartek 12.50 – 15.35 lub piątek 11.15 – 14.00, sala 2.94

**Konsultacje:** pokój 2.42

**Literatura:**

- Geant4 Book For Application Developers – Podstawowy podręcznik do pakietu Geant4
- Geant4 Installation Guide – Instrukcja instalacji
- Physics Reference Manual – Procesy fizyczne i ich modele wykorzystane w kodzie Geant4
- Bruce Eckel, *Thinking in C++*, Helion, 2000
- Czasem warto spojrzeć na kody źródłowe

# Organizacja zajęć cd

**Strona przedmiotu:** https://github.com/olafijalkowska/Geant4_2020

**Zasady zaliczenia:** Zaliczenie nastąpi w oparciu o końcowy projekt oraz obecność na zajęciach. Dopuszczam jedną nieusprawiedliwioną nieobecność. Tematy projektów zostaną zaproponowane w drugiej połowie semestru. Zachęcam do samodzielnego wymyślenia tematu projektu.

# Tematyka wykładów

- Programowanie obiektowe w języku C++ (test)
- Metody Monte Carlo
- Moduły kodu wykorzystującego pakiet Geant4
- Definicja geometrii detektora, kształty i materiały
- Określanie procesów fizycznych
- Określanie warunków początkowych zdarzenia (Event)
- Pojęcia Przebiegu (Run), Zdarzenia (Event) i Kroku (Step)
- Wyciągnie informacji z symulacji
- Manipulacja symulacją przy pomocy skryptów
- Opcjonalnie – Tworzenie geometrii z wykorzystaniem rysunków z CAD
- Nietypowe problemy – z pewnością się z nimi spotkacie

Geant4 oferuje bogatą bibliotekę przykładów, na których nie będziemy się skupiać. Proszę je jednak traktować jako pomoc przy pisaniu projektu.

# Zadanie 1.1

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }
    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};
```

```
A* a = new A();
B* b = new B();
C* c = new C();
cout << c->cax(a) << endl;
cout << c->cax(b) << endl;
```

# Zadanie 1.1

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }
    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};
```

```
A* a = new A();
B* b = new B();
C* c = new C();
cout << c->cax(a) << endl;
cout << c->cax(b) << endl;
```

# Zadanie 1.1

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }
    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};
```

```
A* a = new A();
B* b = new B();
C* c = new C();
cout << c->cax(a) << endl;  1
cout << c->cax(b) << endl;  1
```

# Zadanie 1.2

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }

    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};
```

```
B* b = new B();
C* c = new C();
cout << c->cay(b) << endl;
cout << c->cby(b) << endl;
```

# Zadanie 1.2

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }

    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};
```

```
B* b = new B();
C* c = new C();
cout << c->cay(b) << endl;
cout << c->cby(b) << endl;
```

# Zadanie 1.2

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }

    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};
```

```
B* b = new B();
C* c = new C();
cout << c->cay(b) << endl;    2
cout << c->cby(b) << endl;    22
```

# Zadanie 1.3

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }
    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};
```

```
A* a = new A();
B* b = new B();
C* c = new C();
cout << c->caz(a) << endl;
cout << c->caz(b) << endl;
cout << c->cbz(b) << endl;
```

# Zadanie 1.3

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }
    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};
```

```
A* a = new A();
B* b = new B();
C* c = new C();
cout << c->caz(a) << endl;
cout << c->caz(b) << endl;
cout << c->cbz(b) << endl;
```

# Zadanie 1.3

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }
    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};
```

```
A* a = new A();
B* b = new B();
C* c = new C();
cout << c->caz(a) << endl;   3
cout << c->caz(b) << endl;   33
cout << c->cbz(b) << endl;   33
```

# Zadanie 2

```
class KlasaCpp {
public:
    virtual int foo() = 0;
};
```

Co jest wynikiem wywołania:

```
KlasaCpp* x = new KlasaCpp();
std::cout << x->foo() << std::endl;
```

Metoda

```
virtual int foo() = 0;
```

jest **czysto wirtualna**, czyli taka, która powinna być przesłonięta w klasie pochodnej. Klasa posiadająca metodę czysto wirtualną jest **klasą abstrakcyjną**. Nie można utworzyć instancji klasy abstrakcyjnej!! Klasy abstrakcyjne służą jako interfejsy dla klas pochodnych. Jeżeli klasa pochodna nie przesłoni wszystkich metod czysto wirtualnych, również jest klasą abstrakcyjną.

# Zadanie 3

```
class Counter {
private:
    static int _x;
public:
     Counter(int x) { _x = x;}
     void inc() { _x++; }
     int getState() { return _x; }
};
int Counter::_x = 0;
```

Co jest wynikiem wywołania:

```
Counter* c1 = new Counter(10);
Counter* c2 = new Counter(100);
c1->inc();
c2->inc();
std::cout << c1->getState() << std::endl;
std::cout << c2->getState() << std::endl;
```

# Zadanie 3

```
class Counter {
private:
    static int _x;
public:
     Counter(int x) { _x = x;}
     void inc() { _x++; }
     int getState() { return _x; }
};
int Counter::_x = 0;
```

Co jest wynikiem wywołania:

```
Counter* c1 = new Counter(10);   // _x = 10
Counter* c2 = new Counter(100);  // _x = 100
c1->inc();                       // _x = 101
c2->inc();                       // _x = 102
std::cout << c1->getState() << std::endl;  //102
std::cout << c2->getState() << std::endl;  //102
```

# Zadanie 4

Co zostanie wypisane w konsoli po wpisaniu polecenia:

```
./app paramValue1 paramValue2
```

Jeśli program app jest wynikiem kompilacji kodu:

```
int main(int argc, char *argv[])
{
    for(int i = 0; i!= argc; ++i)
        cout << argv[i] << endl;
    return 0;
}
```

Ile wynosi zmienna argc?

# Zadanie 4

Co zostanie wypisane w konsoli po wpisaniu polecenia:

```
./app paramValue1 paramValue2
```

Jeśli program app jest wynikiem kompilacji kodu:

```
int main(int argc, char *argv[])
{
    for(int i = 0; i!= argc; ++i)
        cout << argv[i] << endl;
    return 0;
}
```

Ile wynosi zmienna argc? **3**
Co kryje argv[0]?

# Zadanie 4

Co zostanie wypisane w konsoli po wpisaniu polecenia:

```
./app paramValue1 paramValue2
```

Jeśli program app jest wynikiem kompilacji kodu:

```
int main(int argc, char *argv[])
{
    for(int i = 0; i!= argc; ++i)
        cout << argv[i] << endl;
    return 0;
}
```

Ile wynosi zmienna argc? **3**
Co kryje argv[0]? **./app**

Co kryje argv[1] i argv[2]?

# Zadanie 4

Co zostanie wypisane w konsoli po wpisaniu polecenia:

```
./app paramValue1 paramValue2
```

Jeśli program app jest wynikiem kompilacji kodu:

```
int main(int argc, char *argv[])
{
    for(int i = 0; i!= argc; ++i)
        cout << argv[i] << endl;
    return 0;
}
```

Ile wynosi zmienna argc? **3**
Co kryje argv[0]? **./app**

Co kryje argv[1] i argv[2]? **paramValue1** i **paramValue2**

# Zadanie 5

Napisz fragment kodu wypisujący na standardowe wyjście dziesięć pierwszych liczb parzystych.

Zadanie na rozgrzewkę. Pewnie powinno mieć numer 1.

# Zadanie 5

Napisz fragment kodu wypisujący na standardowe wyjście dziesięć pierwszych liczb parzystych.

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    int nrOfNumbers=10;
    for(int i = 0; i!= nrOfNumbers; ++i)
        cout << 2*i << endl;
    return 0;
}
```

# Zadanie 6

Napisz szablon funkcji (np. applyForEach) , która przyjmie jako argumenty funkcję $f$ o jednym argumencie i wektor elementów tego samego typu $T$, i zwróci wektor elementów $f(x_i)$ Wskazówka: Deklaracja szablonu funkcji może wyglądać tak:

```
template<typename T>
std::vector<T>  applyForEach(T (*fun)(T), std::vector<T> arrayIn)
```

W wersji bardziej zaawansowanej można zastąpić wektory tablicami.

# Zadanie 6

Napisz szablon funkcji (np. applyForEach) , która przyjmie jako argumenty funkcję $f$ o jednym argumencie i wektor elementów tego samego typu $T$, i zwróci wektor elementów $f(x_i)$ Wskazówka: Deklaracja szablonu funkcji może wyglądać tak:

```
template<typename T>
std::vector<T>  applyForEach(T (*fun)(T), std::vector<T> arrayIn)
```

W wersji bardziej zaawansowanej można zastąpić wektory tablicami.

### Wektor

```
template<typename T>
std::vector<T>  applyForEach(T (*fun)(T), std::vector<T> arrayIn)
{
    vector<T> arrayOut;
    for (int i = 0; i < arrayIn.size(); ++i)
        arrayOut.push_back(fun( arrayIn.at(i) ) );
    return arrayOut;
}
```

### Tablica

```
template<typename T>
T* applyForEach(T (*fun)(T), T* arrayIn, int size)
{
    T *arrayOut = new T[size];
    for(int i = 0; i != size; ++i)
        arrayOut[i] = fun(arrayIn[i]);
    return arrayOut;
}
```

# Zadanie 7

Napisz kod programu losującego dziesięciokrotnie liczbę całkowitą z przedziału od 0 do 10 i wypisujący ją na ekran jeśli jest równa numerowi losowania. Przykład: jeśli program w trzecim losowaniu wylosuje liczbę 3 to zostanie ona wypisana na ekran. Losowania numerujemy nietypowo od 1 (nie ma losowania zerowego). Wskazówka: możesz skorzystać z biblioteki standardowej random.

```
std::random_device rd;
std::default_random_engine engine(rd());//zrobienie silnika
std::uniform_int_distribution<int> dist(min, max);//tworzy rozkład liczb
                                                  //nautralnych od min do max
int randomElement=dist(engine);//losuje liczbę z rozkładu dist
```

# Zadanie 7

Napisz kod programu losującego dziesięciokrotnie liczbę całkowitą z przedziału od 0 do 10 i wypisujący ją na ekran jeśli jest równa numerowi losowania. Przykład: jeśli program w trzecim losowaniu wylosuje liczbę 3 to zostanie ona wypisana na ekran. Losowania numerujemy nietypowo od 1 (nie ma losowania zerowego). Wskazówka: możesz skorzystać z biblioteki standardowej random.

```
std::random_device rd;
std::default_random_engine engine(rd());//zrobienie silnika
std::uniform_int_distribution<int> dist(min, max);//tworzy rozkład liczb
                                                  //nautralnych od min do max
int randomElement=dist(engine);//losuje liczbę z rozkładu dist
```

```
#include <iostream>
#include <random>
using namespace std;
int main(int argc, char *argv[])
{
    std::random_device rd;
    std::default_random_engine engine(rd());
    int min = 0;
    int max=10;
    std::uniform_int_distribution<int> dist(min, max);

    for(int i = min+1; i!= max+1; ++i)
    {
        int randomElement=dist(engine);
        if(randomElement == i)
            cout << randomElement << endl;
    }
    return 0;
}
```

# Pytania testowe

1. Tablice są zbiorami elementów:
   a) dowolnego typu
   b) tego samego typu
2. Dostęp do tablicy odbywa się przez:
   a) iterator
   b) indeks
   c) obie odpowiedzi prawidłowe

   Komentarz: W zasadzie sprawny programista byłby w stanie napisać swój iterator do tablic, pytanie mogłoby być bardziej prezycyjne i określać, że chodzi o iteratory wbudowane w bibliotekę standardową
3. Czym jest tablica dynamiczna?
   Jest to tablica, której rozmiar jest określany w trakcie działania programu, a nie na etapie kompilacji.
4. Dostęp do pola struktury odbywa się przez wykorzystanie operatora:
   a) &
   b) .
   c) *

   Komentarz: Do pól klasy lub struktury uzyskujemy dostęp przez operator ".", jeśli istancja klasy lub strunktu jest stworzona pzrez wartość oraz "$\rightarrow$", jeśli dysponujemy wskaźnikiem do obiektu.

# Pytania testowe

5. Jak poprawnie zwolnić pamięć tak stworzonego obiektu: Foo* f = new Foo[N];
   delete [] f;
   Komentarz: Wywołanie operatora **delete** zwalnia pamięć (wywołuje destruktor) po jednym obiekcie stworzonym jako wskaźnik, czyli z wykorzystanem operatora **new**. W przypadku zwalaniania pamięci po tablicy dynamicznej należy wywołać "tablicową" wersję tego operatora, czyli **delete []**.
6. Czy można stworzyć instancję klasy, której konstruktor jest prywatny? A jeśli tak, to jak stworzyć instancję takiej klasy?
   Można, taką klasą są min. **singletony**, czyli klasy, które w zamierzeniu nie mogę mieć więcej niż jednej instancji w programie. Aby zapobiec możliwosci tworzenia wielu instancji konstruktor czyni się prywatnym. Dostęp do klasy uzyskuje się poprzed publiczną statyczną metodę (np. **getInstance()**), która zwraca wskaźnik do obiektu. Ten sam wskaźnik jest zwracany przy każdym wywołaniu metody **getInstance()**.
   Geant4 w swoich bibliotekach wykorzystuje koncept singletonów i my też będziemy takie klasy tworzyć.

# Wykorzystanie symulacji w fizyce jądrowej

Symulacje pełnią istotną rolę na wielu etapach projektów naukowych

- Projektowanie – Ułatwiają znalezienie optymalnego układu eksperymentalnego
- Tworzenie i obrona projektu – Pomagają oszacować prawdopodobieństwo sukcesu badań, niejednokrotnie uwiarygodniając ambitne ale ryzykowne pomysły
- Przeprowadzenie pomiaru – Pozwalają na szybką ocenę otrzymanych wyników, wykrycie ewentualnych problemów
- Analiza danych – Ułatwiają przeprowadzenie analizy danych i ocenę niepewności

W trakcie tych zajęć skupimy się na symulowaniu procesów fizycznych zachodzących podczas przejścia cząstek przez materię.

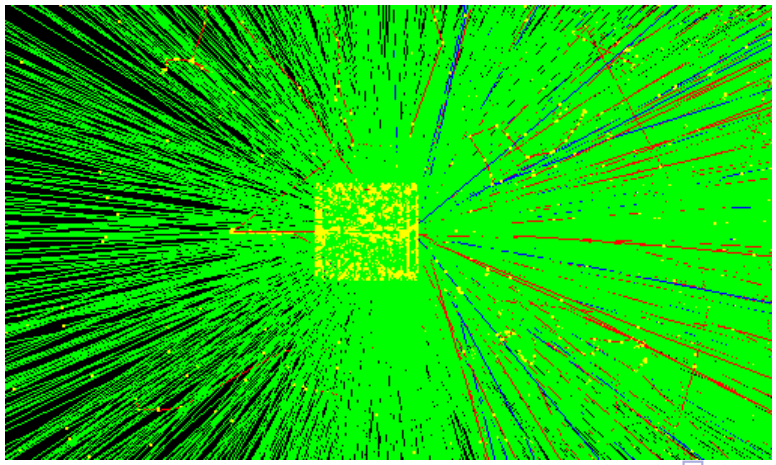
# Dlaczego Geant4?

- Elastyczny – Symulacja różnych materiałów, kształtów, różnego rodzaju promieniowania
- Aktualnie rozwijany – kolejne wersje przynoszą coraz udoskonalenie modeli, przekroje czynne pochodzą z aktualnych baz danych
- Napisany obiektowo w C++ (rozumiem sceptyków, należy jednak docenić, że nie jest napisany w FORTRAN-ie jak Geant3)
- Bardzo szeroko rozpowszechniony – Znajomość pakietu Geant4 jest ceniona w wielu grupach badawczych, bogata dostępność gotowych kodów, przykładów
- Projekt Open Source – Możliwość wprowadzenia swoich wasnych modeli i pomysłów

W trakcie tych zajęć skupimy się na symulowaniu procesów fizycznych zachodzących podczas przejścia cząstek przez materię.

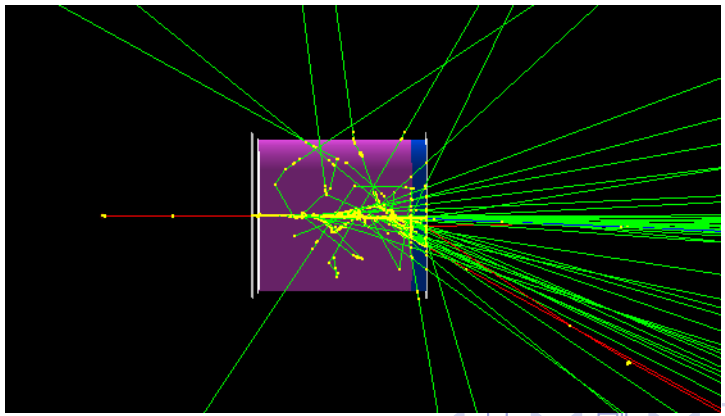
# GEANT – "GEometry ANd Tracking"

- Geant posiada możliwość określania geometrii detektora i śledzenia cząstek w nim propagujących.
- Zanim symulacja zostanie wykonana użytkownik musi podać informacje niezbędne do jej inicjalizacji – zdefiniować geometrię, materiały, określić cząstki początkowe, ich energię i pęd, podać procesy fizyczne i ich przekroje czynne.
- Nadrzędną jednostką symulacji jest Run (seria?)
- Run składa się z szeregu zdarzeń (Event) przeprowadzonych dla określonych warunków początkowych (geometrii i procesów fizycznych)
- Run reprezentowany jest przez klasę `G4Run`
- Użytkownik może określić działania wykonywane na początku i końcu Run-u (np. otwarcie pliku wyjściowego, zapis danych i zamknięcie pliku)



# Zdarzenie - Event

- Każdy Rus składa się z określonej przez użytkownika liczby zdarzeń (Event, klasa `G4Event`)
- Event rozpoczyna wysłanie zdefiniowanych przez użytkownika cząstek pierwotnych
- Na początku zdarzenia wszystkie cząstki pierwotne umieszczane są na stosie a następnie transportowane przez geometrię
- Niektóre procesy, którym ulegają cząstki pierwotne mogą powodować powstanie cząstek wtórnych (np. kreacja pary elektron-pozyton)
- Powstałe cząstki wtórne są odkładane na stos, a następnie jedna po drugiej transportowane przez detektor
- Po przetransportowaniu wszystkich cząstek przez geometrię program wykonuje polecenia określone w klasie `G4UserEventAction` (zapisanie danych do pliku) i kończy zdarzenie Zdarzeniem kieruje klasa G4EventManager.



# Krok - Step

Proces transportowania cząstek pierwotnych i wtórnych odbywa się w krokach (`G4Step`).

- Dla każdego z możliwych procesów dyskretnych **losuje** się odległość oddziaływania (w oparciu o przekroje czynne)
- Najmniejsza z odległości jest wybrana jako krok fizyczny (*physical step length*)
- Program oblicza odległość od granicy bryły, w której krok się odbywa - krok geometryczny (*geometric step length*)
- Zanim proces dyskretny zostanie wykonany program realizuje wszystkie aktywowane procesy ciągłe, wpływają one min. na zmianę energii kinetycznej cząstki, powstanie cząstek wtórnych,
- Jeśli energia kinetyczna cząstki spadnie do zera kończy się śledzenie cząstki, w przeciwnym razie wykonuje się proces dyskretny, mogą powstać cząstki wtórne, zmienić się kinematyka cząstki itp.
- Program wykonuje polecenia określone w klasie G4UserSteppingAction (światło w NaI) i zapamiętuje dane w Trajektorii
- Przed rozpoczęciem nowego kroku program wyznacza nowe wartości średniej drogi swobodnej

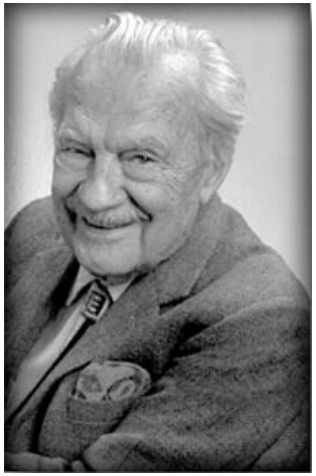
# Losowanie? Monte Carlo!

W naszym przypadku metody Monte Carlo będą wykorzystywane do symulacji procesów, które są statystyczne.
Metody te mają szersze zastosowanie. Pierwotnie były wykorzystywane do rozwiązywania skomplikowanych problemów dla których może istnieć rozwiązanie analityczne.
Podstawowym założeniem metody jest stwierdzenie, że **losowa** próbka wybrana z całej populacji przedstawia zbliżone własności do całej populacji.
Błąd metody maleje ze wzrostem liczności próbki (Prawo Bernoulliego).

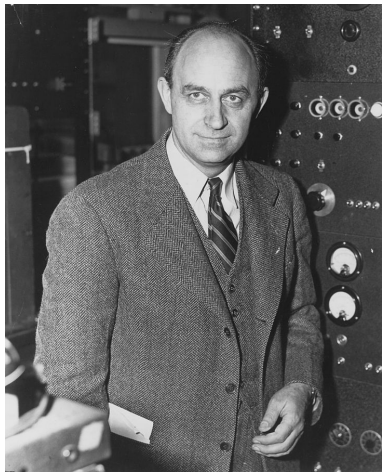
# Losowanie? Monte Carlo!

(a) S. Ulam

(b) J. von Neumann

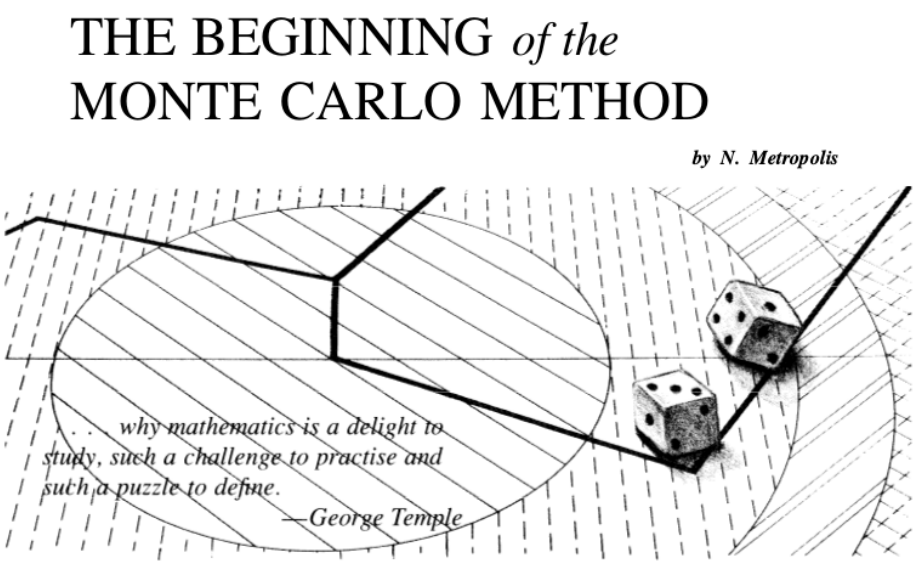


(c) N. Metropolis



(d) E. Fermi

Fotografie pochodzą z zasobów Wikipedii

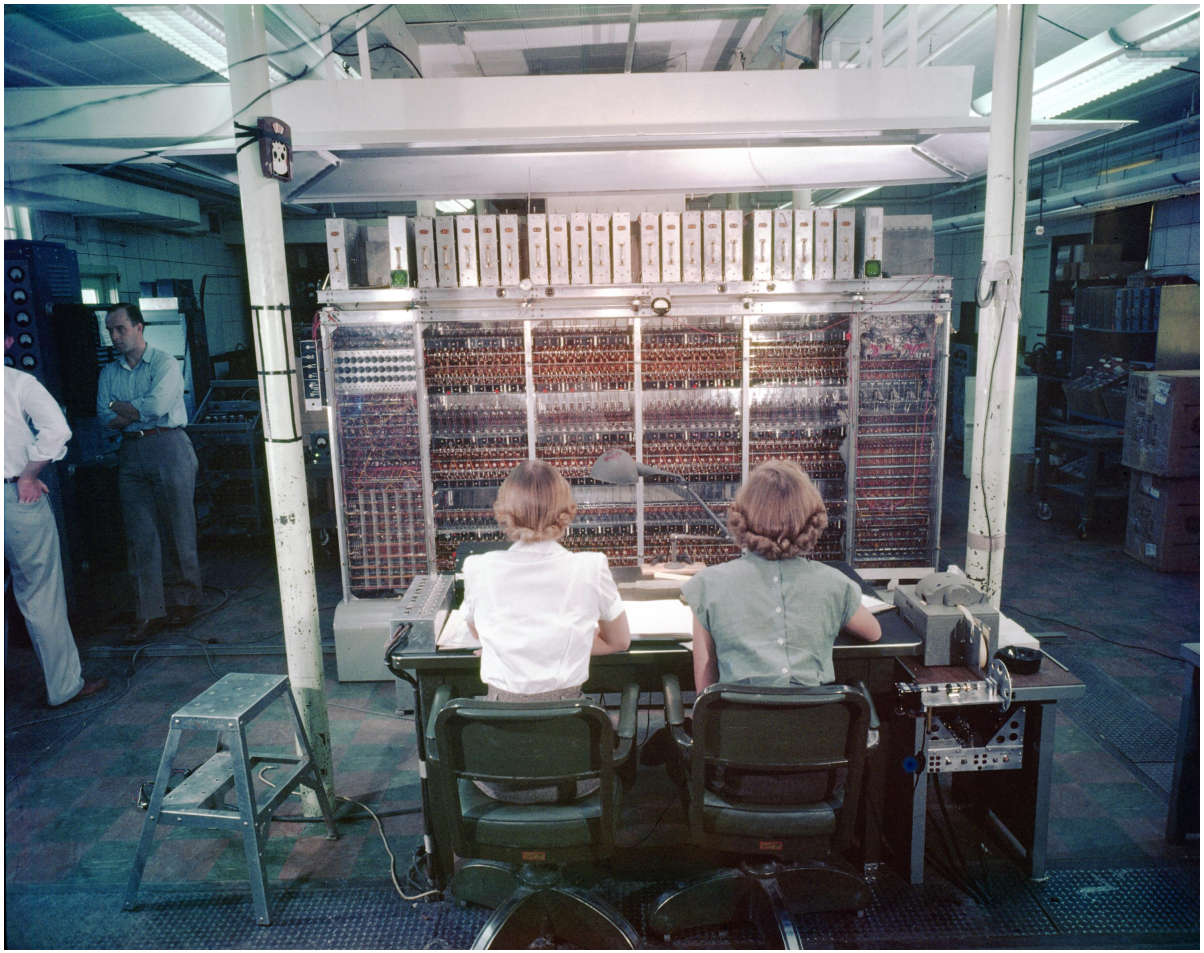

THE BEGINNING *of the*
MONTE CARLO METHOD

*by N. Metropolis*

. . . why mathematics is a delight to study, such a challenge to practise and such a puzzle to define.

—George Temple

# Losowanie? Monte Carlo!



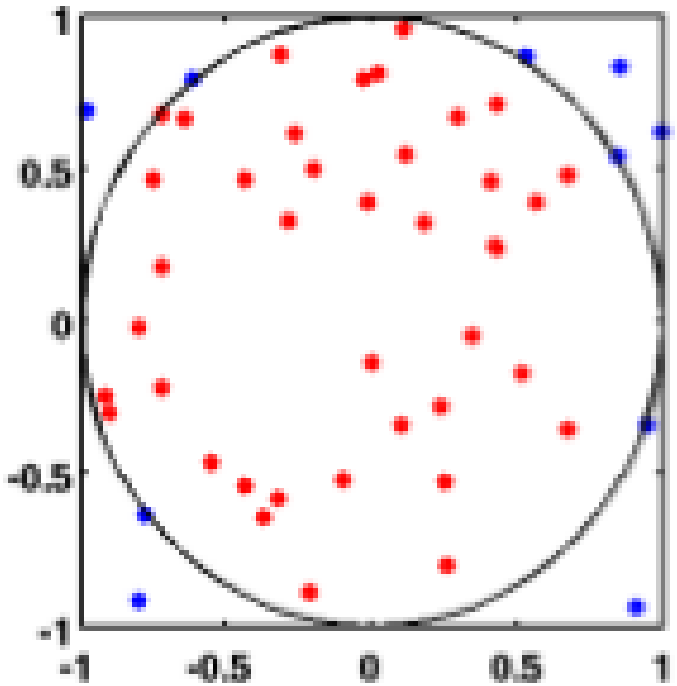
MANIAC 1, Mathematical Analyzer, Numerator, Integrator, and Computer Fotografia pochodzi z zasobów LANL

# Przykłady - objętość N-wymiarowej kuli

Wyznacz objętość $n$-wymiarowej kuli o promieniu $R$ metodą Monte Carlo.

Sugestia:

- Wyslosuj N punktów z n-wymiarowego pudełka o boku 2R
- Wyznacz liczbę punktów (M), dla których odległość od punktu 0 znajduje jest mniejsza od R (te znajdą się wewnątrz kuli)
- Objętość kuli $V = \frac{M}{N} \cdot (2R)^n$, gdzie $(2R)^n$ jest objętością pudełka



# Przykłady - ruletka, szatańska gra

$$1 + 2 + 3 + ... + 34 + 35 + 36 = 666$$

W zależności od systemu na kole znajduje się jedno 0 (system europejski) lub 0 i 00 (amerykański).
Założymy uproszczoną wersję zakładów, można obstawiać jedno pole i w razie wygranej uzyskuje się 35-krotność postawionych pieniędzy.

# Kompilacja

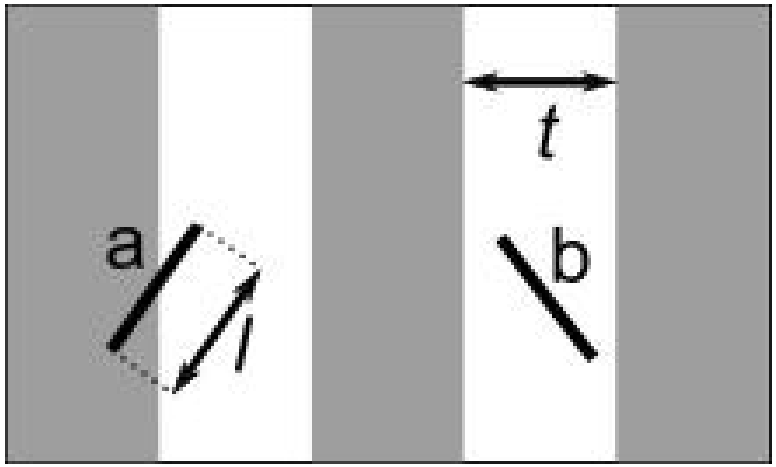
g++ -std=c++11 -o outputName Source.cpp

Przykład CMakeLists.txt:

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
set (CMAKE_CXX_STANDARD 11)
project(projectName)

include_directories(include)

set(CMAKE_BUILD_TYPE release)

# User code
file(GLOB sources src/*.cpp)
file(GLOB headers include/*.h)

add_executable(projectName mainCode.cpp ${sources} ${headers})
```

# Igła Buffona

Oszacuj wartość liczby $\pi$ rzucając igłą Buffona.




Prawdopodobieństwo, że rzucona losowo igła o długości $l$ (losowe położenie środka, oraz kąt $\theta$ względem linii) przetnie jedną z linii oddalonych o siebie o odległość $t$ wynosi: $p = \frac{2}{\pi}\frac{l}{t}$.
Znajdź parawdopodobieństwo $p$ metodą Monte Carlo, a następnie wyznacz wartość liczby $\pi$. Narysuj wykres wartości otrzymanej wartości liczby $\pi$ w funkcji liczby rzutów igłą $N$.

# Pytania?