# Zastosowanie pakietu Geant4 w fizyce jądrowej

Aleksandra Fijałkowska

Wydział Fizyki, Uniwersytet Warszawski

*aleksandra.fijalkowska@fuw.edu.pl*

14 października 2021

# 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.
Wykonaj obliczenia „stopy zwrotu" gry w ruletkę w wersji amerykańskiej i europejskiej.

# 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})
```

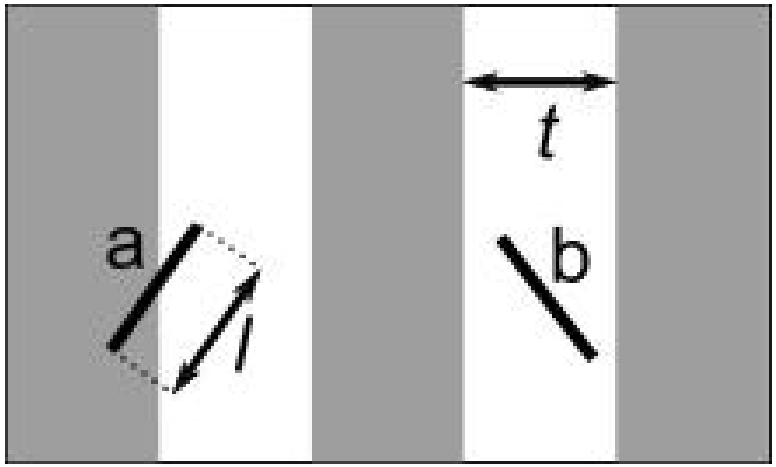
# Kompilacja

Przykład Makefile:

```
DESTDIR=.
CINCLUDEDIRS = -Iinclude
CPPFLAGS  += -O3
CPPFLAGS  += -std=c++0x
CPPFLAGS += $(CINCLUDEDIRS)
SRC = src/

ruletka: szatanskaGra.o Ruletka.o
g++ szatanskaGra.o Ruletka.o -o ruletka

szatanskaGra.o: szatanskaGra.cpp
g++ $(CPPFLAGS) -c szatanskaGra.cpp

Ruletka.o: $(SRC)Ruletka.cpp
g++ $(CPPFLAGS) -c $(SRC)Ruletka.cpp

clean:
rm -f *.o *~
```

# 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 $N$.

# 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 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 6

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.

# GIT

Adres „naszego” repozytorium:
`https://github.com/olafijalkowska/Geant4_2021`

Kilka komend GIT-owych
`https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html`
Github ma obecnie dwustopniowy system autoryzacji, oprócz hasła należy utworzyć klucz ssh.
Instrukcja jak to zrobić jest opisana na stronie Githuba.

# NeutronX

Geant - trzeba wpisać:
. geant4 (wersja Geant4.10.04)
Root - jest kilka wersji:
'root' daje 5.34/30
'root5' jest 5.34/36
'root6' - 6.14/04

# Sterowanie symulacją

Polecenia umieszczone w skrypcie vis.mac są tymi samymi poleceniami, które możemy podać w konsoli geant'a.
Dodatkowo, te same polecenia możemy wpisywać bezpośrednio w kodzie programu, jako argumenty metody `ApplyCommand` klasy `G4UImanager`:

```
UImanager→ApplyCommand("/control/execute commend");
```

Oczywiście wpisywanie komend sterujących symulacją w samym kodzie byłoby szalenie niewygodne. Przy każdej zmianie ustawień musielibyśmy kompilować kod. Sama idea wprowadzenia komend sterujących symulacją ma na celu możliwie duże uproszczenie kodu kompilowanego, przez co zapewnienei elastyczności końcowego programu.

# Sterowanie symulacją

Sterowanie wykonaniem symulacji w środowisku GEANT4 można więc zrealizować na trzy sposoby:

1. Za pomocą kodu w programie main. Każda zmiana w sposobie realizacji symulacji wymaga zmiany kodu i jego ponownej kompilacji.
   Opcja niewygodna, wymagająca kompilacji kodu po każdej zmianie.
2. Za pomocą komend wpisywanych w oknie UI (jak na przykład wysłanie cząstek w załączonym filmie).
   Przydatna opcja na etapie projaktowania symulacji, robienia małych testów itd. Jednakże doprowadzenie programu do realizowania pożądanej symulacji z ładnym wyświetlaniem wymaga wpisania kilkunastu komend, które przy tym rozwiązaniu należy mieć w pamięci. Warto posiłkować się więc dodatkowym skryptem.
3. Za pomocą skryptów. Uzgodniliśmy, że będziemy używać dwóch typów skryptów:
   - 3.1 skrypt "domyślny" - "vis.mac", który jest wczytywany za każdym razem, gdy użytkownik nie poda argumentu wejściowego. Skrypt będzie zawierał podstawowe komendy dotyczące wyświetlania.
   - 3.2 skrypt dostosowany do konkretnej symulacji, którego nazwa będzie podawana podczas uruchomienia programu. To rozwiązanie jest stosowane najczęściej, gdy nie chcemy wyświetlać przebiegu symulacji, zależy nam na jak największej wydajności i szybkości działania programu.

# Realizacja trzeciej strategii

1. Stworzenie instancji User Interface Manager:

   ```
   G4UImanager* UImanager = G4UImanager::GetUIpointer();
   ```

2. Stworzenie i zainicjowanie instancji klasy obsługującej wizualizację:

   ```
   G4VisManager* visManager = new G4VisExecutive;
   visManager->Initialize();
   ```

3. Stworzenie instancji klasy obsługującej interfejs zgodny ze zdefiniowaną zmienną środowiskową (np G4UI_USE_QT):

   ```
   G4UIExecutive* ui = new G4UIExecutive(argc, argv);
   ```

4. Sprawdzenie liczby podanych argumentów wejściowych, w przypadku braku argumentu uruchomienie skryptu "vis.mac", zaś w przeciwnym sczytanie nazwy skryptu i uruchomienie go:

   ```
   if(argc == 1)
       UImanager->ApplyCommand("/control/execute ../vis.mac");
   else
   {
       G4String filename = argv[1];
       UImanager->ApplyCommand("/control/execute " + filename);
   }
   ```

5. Rozpoczęcie sesji ui:

   ```
   ui->SessionStart();
   ```

# Przykładowy plik vis.mac

```
/vis/open OGL 600x600-0+0
/vis/drawVolume
/vis/scene/add/trajectories smooth
/vis/scene/endOfEventAction accumulate
/vis/modeling/trajectories/create/drawByCharge
/vis/modeling/trajectories/drawByCharge-0/default/setDrawStepPts true
/vis/modeling/trajectories/drawByCharge-0/default/setStepPtsSize 2
/vis/scene/add/axes 0 0 0 2.0 m
```

użyj openGL do wyświetlania
narysuj bryły zdefiniowane w kodzie
rysuj tory cząstek
wyświetlaj wyniki wielu zdarzeń (Event) na jednym obrazku
nadaj torom cząstek różne kolory w zależności od ich ładunku
rysuj punkty interakcji (końce kroków) w postaci punktów
określa rozmiar punktu oznaczającego koniec kroku
dodaj osie w punkcie (0,0,0) i długości 2 m

# Podstawowe polecenia sterujące symulacją

```
/control/   UI control commands.
/units/   Available units.
/analysis/   ...Title not available...
/process/   Process Table control commands.
/particle/   Particle control commands.
/geometry/   Geometry control commands.
/tracking/   TrackingManager and SteppingManager control commands.
/event/   EventManager control commands.
/cuts/   Commands for G4VUserPhysicsList.
/run/   Run control commands.
/random/   Random number status control commands.
/gun/   Particle Gun control commands.
/material/   Commands for materials
/vis/   Visualization commands.
/gui/   UI interactors commands.
```

# Najpopularniejsze przykłady

```
/run/beamOn <nrOfEvents>
#wysyła nrOfEvents cząstek pierwotnych

/gun/particle <particleName>
#Określa rodzaj cząstki pierwotnej (gamma, neutron, e-, e+ itd.)

/gun/energy <Energy> <Unit>
#Ustala energię cząstki pierwotnej

/gun/position <X> <Y> <Z> <Unit>
#Określa pozycję, z której są wysyłane cząstki

/gun/direction <ex> <ey> <ez>
#Określa kierunek cząstek pierwotnych

/vis/viewer/zoom <multiplier>
#Dziś już zapomniana komenda do zwiększania obrazu wizualizacji.
#Teraz wystarczy użyć kółka myszki.
```

Przykład użycia:

```
/run/beamOn 100
/gun/particle neutron
/gun/energy 0.5 MeV
/gun/direction 1 0.5 2
#wektor direction nie musi być znormalizowany
```