**ГБОУ "Президентский ФМЛ № 239"**

**Моделирование поведения неклейких гранулированных частиц при их выливании**

Годовой проект по информатике

Работу выполнил

Ученик 10-2 класса

Коврижных Дмитрий

Санкт – Петербург

2018

**1. Постановка задачи**

Программа запрашивает данные о неклейких гранулированных частицах у пользователя. На их основе она должна рассчитать движение и взаимодействие частиц и смоделировать его в отдельном окне.

**2. Уточнение исходных и выходных данных и ограничений на них**

**2.1. Исходные данные**

Программа запрашивает у пользователя данные в специальном окне, а именно о среднем размере частицы (от 5 до 30, иначе изображение будет трудно воспринимать, так как частицы будет либо плохо видно, либо они будут занимать слишком много места), о коэффициенте потерь при соударении (от 0.41 до 1, так как меньшее значение слишком сильно замедляет частицы и они фактически просто стоят на месте, а коэффициент больше 1 физически не может быть), о времени шага (от 0.05 до 0.43, при меньшем значении компьютер начинает тормозить, а при большем частицы начинают двигаться рывками), и количестве частиц (до 20, так как при большем значении становится трудно различать частицы).

**2.2. Выходные данные**

На экран выводится окно размером 1000 на 1000 пикселей, где происходит визуализация моделирования движения и взаимодействия частиц.

**3. Математическая модель**

При расчёте используются формулы: скорости и координат при равноускоренном движении для каждой оси, столкновения с частицей и столкновения со стенкой:

𝒙 = 𝒙𝟎 + 𝒗𝒙 , где 𝒙 и 𝒙𝟎 – конечная и начальная координаты по оси 𝒙 соответственно в конце и начале шага, 𝒗𝒙 – скорость вдоль той же оси;

𝒗𝒙 = 𝒗𝟎𝒙 + 𝒂𝒙 , где 𝒗𝒙 и 𝒗𝟎𝒙 – конечная и начальная скорости по оси 𝒙 соответственно в конце и начале шага, 𝒂𝒙 – ускорение вдоль той же оси;

∆𝒗𝒙 = 𝒗𝒙 ′ × 𝐬𝐢𝐧 𝜶 + 𝒗𝒚 ′ × 𝐜𝐨𝐬𝜶 , где ∆𝒗𝒙 – изменение скорости по оси 𝒙, 𝒗𝒙 ′ и 𝒗𝒚 ′ -- скорости второй частицы по осям 𝒙 и 𝒚 соответственно, 𝜶 – угол между осью 𝒙 и отрезком, соединяющим центры сталкивающихся частиц; 𝒗𝒙 = −𝒗𝟎𝒙, где обозначения те же, что и в формуле для расчёта скорости.

**r = r0 + U0\*t + (at^2)/2 –** общая формула для расчета координат вдоль каждой оси

**4. Структура данных**

Структура данных представляет собой набор переменных и массив данных о частицах.

Набор переменных содержит в себе технические величины, физические величины, величины, введённые пользователем, номера строк массива в виде переменных для их более удобного восприятия, т. е данные, которые не будут изменяться в ходе работы программы.

Массив содержит в себе 4 столбца в каждом из которых хранится одна из величин, характеризующих положение частицы (координата вдоль оси ox, координата вдоль оси oy, скорость вдоль оси oy, скорость вдоль оси ox). Размер столбца массива равен количеству частиц, которое задаётся пользователем. Все характеристики частиц хранятся в массиве, по которому каждый шаг выполняется прохождение, в процессе которого положения и скорости частиц обновляются в соответствии с математической моделью и на экране изображаются круги на основании координат частиц.

**5. Метод решения**

Сначала программа создаёт данные о заданном кол-во частиц вдоль вертикальной средней линии экрана на заданном расстоянии с небольшими горизонтальными отклонениями внутри массива, выше верхней границы. Затем идёт расчет перемещения каждой частицы на один шаг в соответсвии с уравнением равноускоренного движения для координат и записывается в массив. После идёт расчет соударений частицв соответсвии с законом для нецетральных ударов, и их отрисовка.

Далее программа делает новый шаг, повторяя все приведённые выше расчеты, за исключением того, что перед отрисовкой следующих, программа удаляет предыдущие. Таким образом, мы получаем анимированную визуализацию движения и взаимодействия частиц.

**6. Листинг программы**

**from** tkinter **import** \* *# импротируемые модули***from** random **import** \*  
g = 10 *# ускорение свободного падения*height=1000 *# регулировка размеров окна*y = 0 *# переменные для удобного перемещения по массиву*x = 1  
uy = 2  
ux = 3  
boo = **"false"** *# строковой индикатор***while** (boo != **'да'**): *# ввод данных о частицах* print(**"Введите число частиц:"**)  
 number = int(input())  
 print(**"Введите время шага:"**)  
 t = float(input())  
 print(**"Введите коэффициент потерь:"**)  
 kpoteri = float(input())  
 print(**"Введите размер частицы:"**)  
 r = int(input())  
 print(**"Введите 'да', если всё верно, 'нет' в противном случае:"**)  
 boo = input()  
  
  
Particles = [0,0] \* 4 *# создание массива***for** i **in** range(4):  
 Particles[i] = [0] \* number  
  
**for** z **in** range(0,number): *# заполнение массива начальными данными* Particles[y][z] = -70\*z + randint(-5,+5)  
 Particles[x][z]+= height/2+ randint(-10,+10)  
 Particles[ux][z]=0  
 Particles[uy][z]=0  
  
**def** time\_move(n):  
 **if** n < 800:  
 i = 0  
 j = 0  
 **for** i **in** range(number):  
  
 Particles[uy][i] += round(g \* t) *# перемещение частиц по действием силы тяжести* Particles[y][i] += round(Particles[uy][i] \* t)  
 Particles[x][i] += round(Particles[ux][i] \* t)  
 **if** Particles[y][i] >=height-r :  
 **if** Particles[uy][i] >0:  
 Particles[uy][i] = round(-kpoteri \* Particles[uy][i])  
  
  
 **if** Particles[x][i] >=height-r : *# проверка на соударение с боковой границей* **if** Particles[ux][i] >0:  
 Particles[ux][i] =round(-kpoteri\*Particles[ux][i])  
 **if** Particles[x][i] <=0 :  
 **if** Particles[ux][i] <0:  
 Particles[ux][i] =round(-kpoteri\*Particles[ux][i])  
  
  
 **for** i **in** range(number): *# проверка соударения между частицами и его расчет* j=i+1  
 **for** j **in** range(i+1,number):  
  
 **if** (j <= number):  
 length=round(pow(((Particles[y][i] - Particles[y][j]) \*\* 2 + (Particles[x][i] - Particles[x][j]) \*\* 2), 0.5))  
 **if** (length <= 2 \* r)**and**(length!=0):  
 a = round(  
 Particles[uy][i] \* abs(((Particles[y][i] - Particles[y][j])) / length) + Particles[ux][i] \* abs((  
 (Particles[x][i] - Particles[x][j])) / length))  
 b = round(Particles[uy][i] \* abs((Particles[x][i] - Particles[x][j])) / length)  
 Particles[uy][i] = round(  
 Particles[uy][j] \* abs(((Particles[y][i] - Particles[y][j])) / length)+ Particles[ux][j] \* abs((  
 (Particles[x][i] - Particles[x][j])) / length))  
 Particles[uy][j] = a  
 **if** (Particles[x][i] > Particles[x][j]):  
 Particles[ux][i] += round(  
 Particles[uy][j] \* abs(((Particles[y][i] - Particles[y][j])) / length) + Particles[ux][  
 j] \* abs(((Particles[x][i] - Particles[x][j])) / length))  
 Particles[ux][j] += -b  
 **if** (Particles[x][i] < Particles[x][j]):  
 Particles[ux][i] += -round(  
 Particles[uy][j] \* abs(((Particles[y][i] - Particles[y][j])) / length)+ Particles[ux][  
 j] \* abs(((Particles[x][i] - Particles[x][j])) / length))  
 Particles[ux][j] += b  
 **break** *#time.sleep(0.1) вдруг понадобиться тормозить программу* canvas.delete(ALL) *# отрисовка частиц на новых позициях* **for** ln **in** range(0, number):  
 canvas.create\_oval(Particles[x][ln] - r, Particles[y][ln] - r, Particles[x][ln] + r, Particles[y][ln] + r,  
 fill=**'black'**, width=0)  
  
 canvas.after(50, **lambda**: time\_move(n + 1)) *# повторение рекурсии  
#*frame = Tk() *# создание холста*canvas = Canvas(frame, width=height, height=height, background=**"white"**)  
canvas.grid()  
canvas.after(50, **lambda**: time\_move(0))  
*#*frame.mainloop()

**7. Пример работы**

**7.1 Входные данные:**

Время шага 0,06 сек

Число частиц 4

Размер частиц 10

Коэффициент потерь 0,8

**7.2 Выходные данные:**

<https://github.com/HCL-271/Informatic-project/blob/master/Result_of_work.mp4>

**8. Анализ правильности решения**

Программа наглядно демострирует взаимодействие и движение частиц. Получаемый результат соответствует действительности и законам физики. Хотя программа может обработать и большое количество частиц( порядка 100 – 200), это не требуется, так как при этом теряется возможность различать частицы по отдельности. Решение не является самым оптимальным, но оно успешно справляется с данной ему задачей, а значит в целом оно правильное.