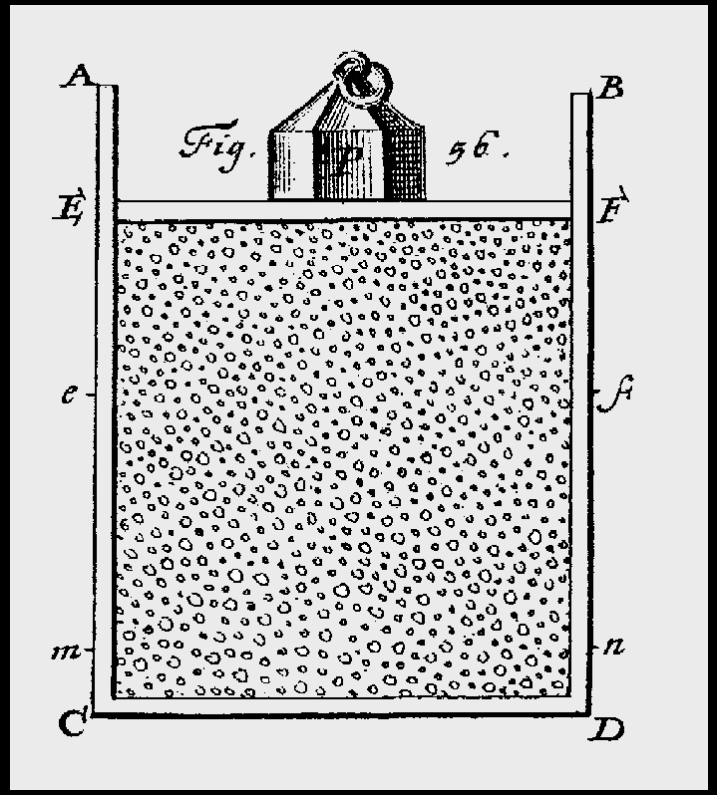
Symulacje komputerowe w fizyce



08-09.11.2022

Ćwiczenia 5: Gaz

Zadanie

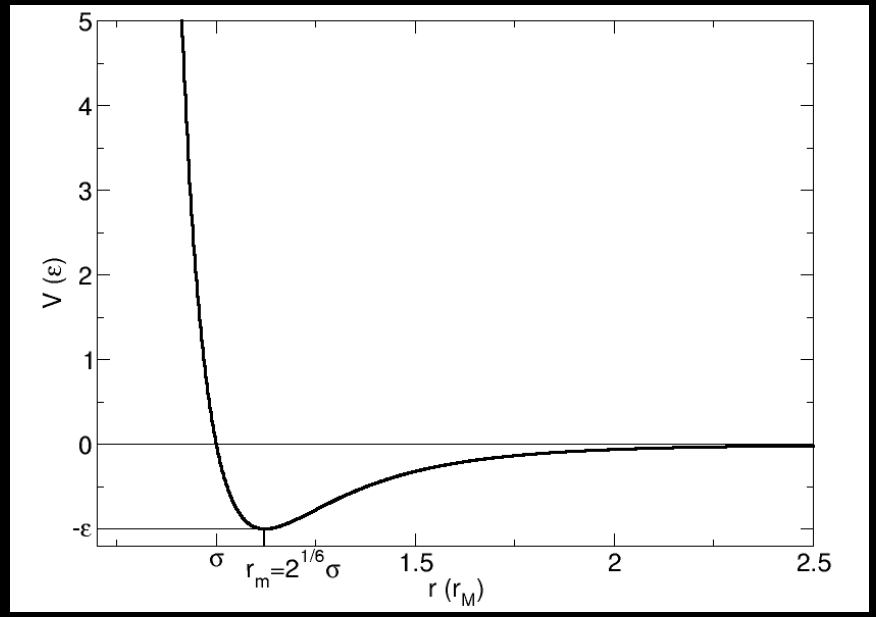
Napisz symulację dynamiki molekularnej dwuwymiarowego gazu szlachetnego (cząstek oddziałujących siłami Lennarda-Jonesa) w periodycznych warunkach brzegowych.

Potencjał Lennarda-Jonesa:

$$V(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

• prosty potencjał przybliżający oddziaływanie między obojętnymi elektrycznie atomami,

• człon r^{-12} opisuje odpychanie chmur elektronowych na bliskich odległościach, podczas gdy człon r^{-6} przyciąganie na dużych odległościach (siły van der Waalsa).



Szczegóły zadania

1. Weź N cząstek (na początek niech $N = 16$), ustaw je na regularnej sieci.

2. Nadaj im prędkości odpowiadające początkowej temperaturze *temp*.

3. Prowadź symulację przez pewien czas t (korzystając z algorytmu żabki) i – co jakiś czas - zapisuj konfigurację.

4. Zrób krótką animację.

*5. Znajdź ewolucję temperatury i ciśnienia w Twoim układzie.

Klasa „cząstka”

Klasa to wygodna konstrukcja pozwalająca trzymać wszystkie atrybuty cząstki w jednym miejscu.

```
class czastka:
    """Klasa opisujaca pojedyncza czastke gazu
    """
    def __init__(self, promien, pos, vel):
        self.promien = promien
        self.r = pos      # położenie
        self.v = vel      # prędkość
```

dwie podłogi! (

pierwszym argumentem każdej metody (funkcji składowej w klasie) jest zawsze odniesienie do przetwarzanego właśnie egzemplarza klasy i zwyczajowo jest nazwane *self*

konstruktor – automatycznie wywoływany przy tworzeniu nowego egzemplarza klasy

- nową cząstkę można utworzyć za pomocą:
  g = czastka(promien, polozenie, predkosc) uwaga – w argumenty nie włączamy *self*
- własności cząstki mogą być uzyskane poprzez:
  g.promien, g.r, g.v etc.

Ogólna struktura programu

Inicjalizacja:

1. Tworzymy listę cząstek.
2. Usuwamy prędkość środka masy.
3. Skalujemy prędkości do temperatury.

Ewolucja:

```
for i in range(iters):
    for pi in particles:

        Obliczanie sił:
         F_sum = 0
         for pj in particles:
             if not pi == pj:
                 F_sum += force(pi, pj)

        Aktualizuj prędkości i położenia
        *Obliczaj wielkości makroskopowe (temperatura, ciśnienie, etc)
```

Przykładowe wartości parametrów:

particle_number = 16
box_size = 8.0
eps = 1.0
sigma = 1.0
promien = sigma / 2
dt = 0.0001
temp = 2.5
$k_B = 1$
m = 1

Inicjalizacja

1

```
particles = []
for i in range(nx):
    for j in range(ny):
        polozenie = np.array([i * dx + 1, j * dx + 1])
        predkosc = np.array([(np.random.random() - 1 / 2), (np.random.random() - 1 / 2)])
        particles.append(czastka(promien, polozenie, predkosc))
```

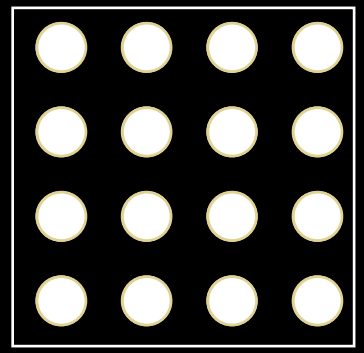
utwórz siatkę n_x x n_y cząstek
(żeby nie nachodziły na siebie)
d_x, d_y = box_size / n_x, n_y

przypadkowe prędkości
początkowe [0,1)

utwórz cząstki i dopisz
je do listy cząstek

2

```
sumv = 0.0
for p in particles:
    sumv += p.v
sumv = sumv / particle_number  # prędkość środka masy
for p in particles:
    p.v -= sumv # teraz środek masy spoczywa
```



3

```
sumv2 = 0.0
for p in particles:
    sumv2 += np.dot(p.v, p.v) / 2.0
sumv2 = sumv2 / particle_number  # średnia energia kinetyczna (masa jednostkowa)
fs = np.sqrt(temp / sumv2)       # czynnik skalujący,  temp - żądana temperatura (a dokładnie kT)
for p in particles:
    p.v = p.v * fs     # skalujemy
```

Siły

potencjał Lennarda-Jonesa:

$$u(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

siła:

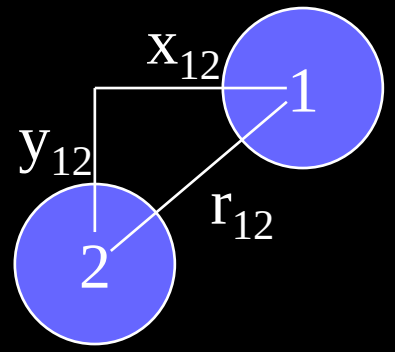
wartość skalarna

$$f(r) = -\frac{du}{dr} = \frac{48\varepsilon}{\sigma}\left[\left(\frac{\sigma}{r}\right)^{13} - \frac{1}{2}\left(\frac{\sigma}{r}\right)^{7}\right]$$

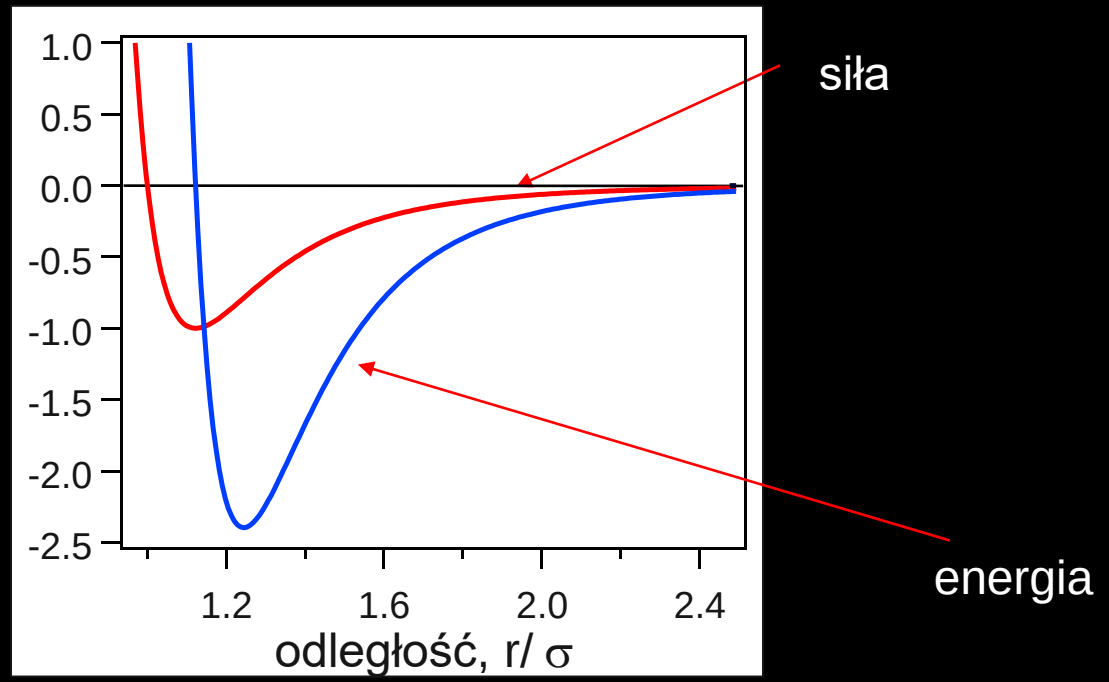
$$\mathbf{F}_{2\to1} = -\frac{f(r_{12})}{r_{12}}\left[x_{12}\mathbf{e}_x + y_{12}\mathbf{e}_y\right]$$

$$\mathbf{F}_{2\to1} = -\frac{48\varepsilon}{\sigma^2}\left[\left(\frac{\sigma}{r_{12}}\right)^{14} - \frac{1}{2}\left(\frac{\sigma}{r_{12}}\right)^{8}\right]\left[x_{12}\mathbf{e}_x + y_{12}\mathbf{e}_y\right]$$

wektor



$$r_{12} = [(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2}$$



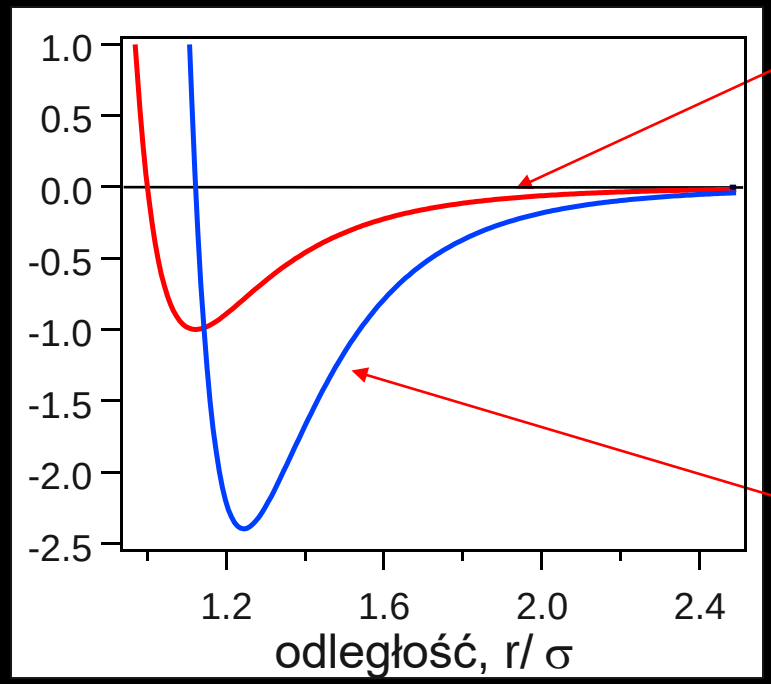
Obcięcie potencjału LJ

zwykle obcinamy w $r_c = 2.5\sigma$

$$u_s(r) = \begin{cases} u(r) - u(r_c) & r \le r_c \\ 0 & r > r_c \end{cases}$$

obcinamy i przesuwamy
(aby potencjał był ciągły)

$$f_s(r) = \begin{cases} f(r) & r \le r_c \\ 0 & r > r_c \end{cases}$$



siła

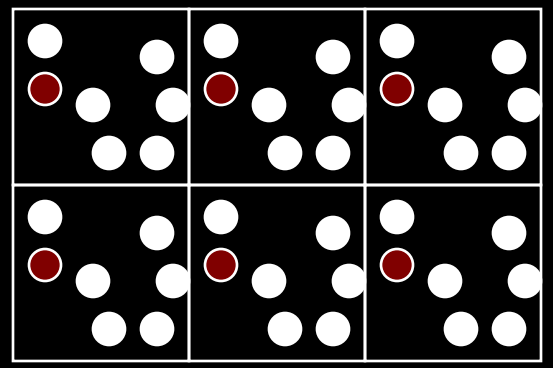
energia

Periodyczne warunki brzegowe

Modelowanie ścianek jest niepraktyczne:

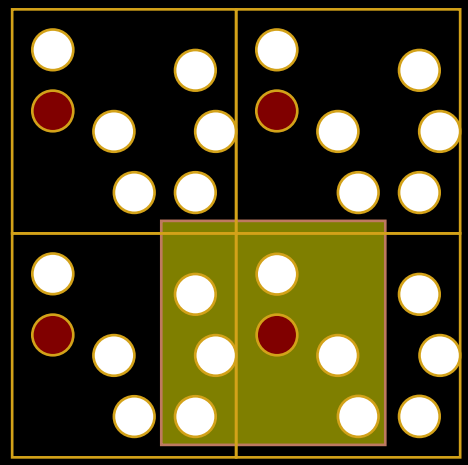
- silne artefakty związane ze skończonymi rozmiarami układu,
- sztuczny wpływ ściany na własności układu.

Zamiast modelować ścianki, lepiej jest otoczyć nasz układ jego kopiami – “periodyczne warunki brzegowe” (PBC).



PBC - implementacja

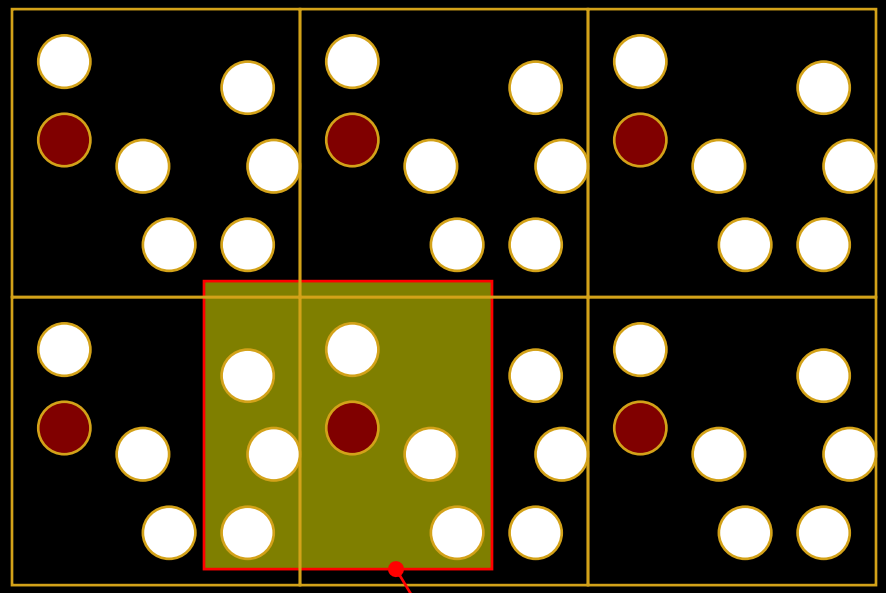
```
if newX > box_size:
    newX -= box_size
if newX < 0:
    newX += box_size

if newY > box_size:
    newY -= box_size
if newY < 0:
    newY += box_size
```



newX, newY – nowe (zaktualizowane) współrzędne x i y cząstki

Najbliższy obraz

Rozważaj tylko najbliższy obraz danej cząstki przy wyznaczaniu sił!

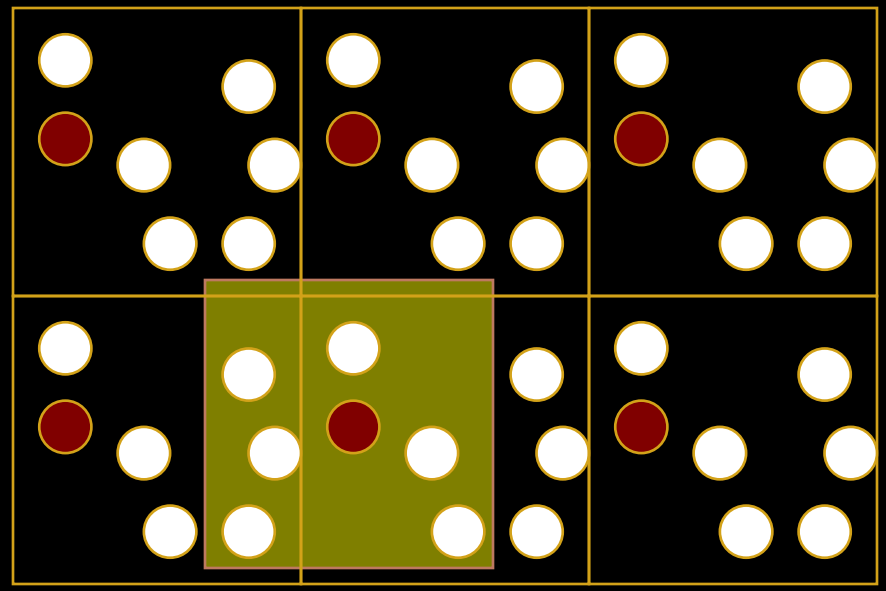


najbliższe obrazy czerwonej cząstki

(można stosować wtedy kiedy obcięcie sił jest mniejsze niż połowa boku pudełka, dla sił długozasięgowych jest trudniej!)

Najbliższy obraz - implementacja

```
def closest_image(x1, x2):
    x12 = x2 - x1
    if x12[0] > box_size / 2:
        x12[0] = x12[0] - box_size
    elif x12[0] < -box_size / 2:
        x12[0] = x12[0] + box_size
    if x12[1] > box_size / 2:
        x12[1] = x12[1] - box_size
    elif x12[1] < -box_size / 2:
        x12[1] = x12[1] + box_size
    return x12
```



Metoda całkowania - Żabka

$$\mathbf{v}(t + \delta t/2) = \mathbf{v}(t - \delta t/2) + \left(\frac{\mathbf{F}(t)}{m}\right)\Delta t$$

$$\mathbf{r}(t + \delta t) = \mathbf{r}(t) + \mathbf{v}(t + \delta t/2)\Delta t$$

Animacja

```
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

…
inicjalizacja
…

for i in range(iterations):
    …
    ewolucja
    …
    if (i % 100 == 0):     #  co 100-na klatka
        plt.clf()              #  wyczyść obrazek
        fig = plt.gcf()        #   zdefiniuj nowy
        for p in particles:    # pętla po cząstkach
            a = plt.gca()
            cir = Circle((p.r[0], p.r[1]), radius = p.promien)  # kółko tam gdzie jest cząstka
            a.add_patch(cir)                # dodaj to kółko do rysunku
            plt.plot()                      # narysuj
        plt.xlim((0, box_size))               # obszar do narysowania
        plt.ylim((0, box_size))
        fig.set_size_inches((6, 6))        # rozmiar rysunku
        plt.title(f'Symulacja gazu Lennarda-Jonesa, krok  {i:06d}')
        plt.savefig(f'img{i:06d}.png')
```

w środku pętli z ewolucją
(nie potrzeba zapisywać trajektorii)

Robienie filmu

convert -delay 0.1 *.png anim.gif (https://imagemagick.org/script/index.php)

opóźnienie między klatkami

nazwa animacji

lub:

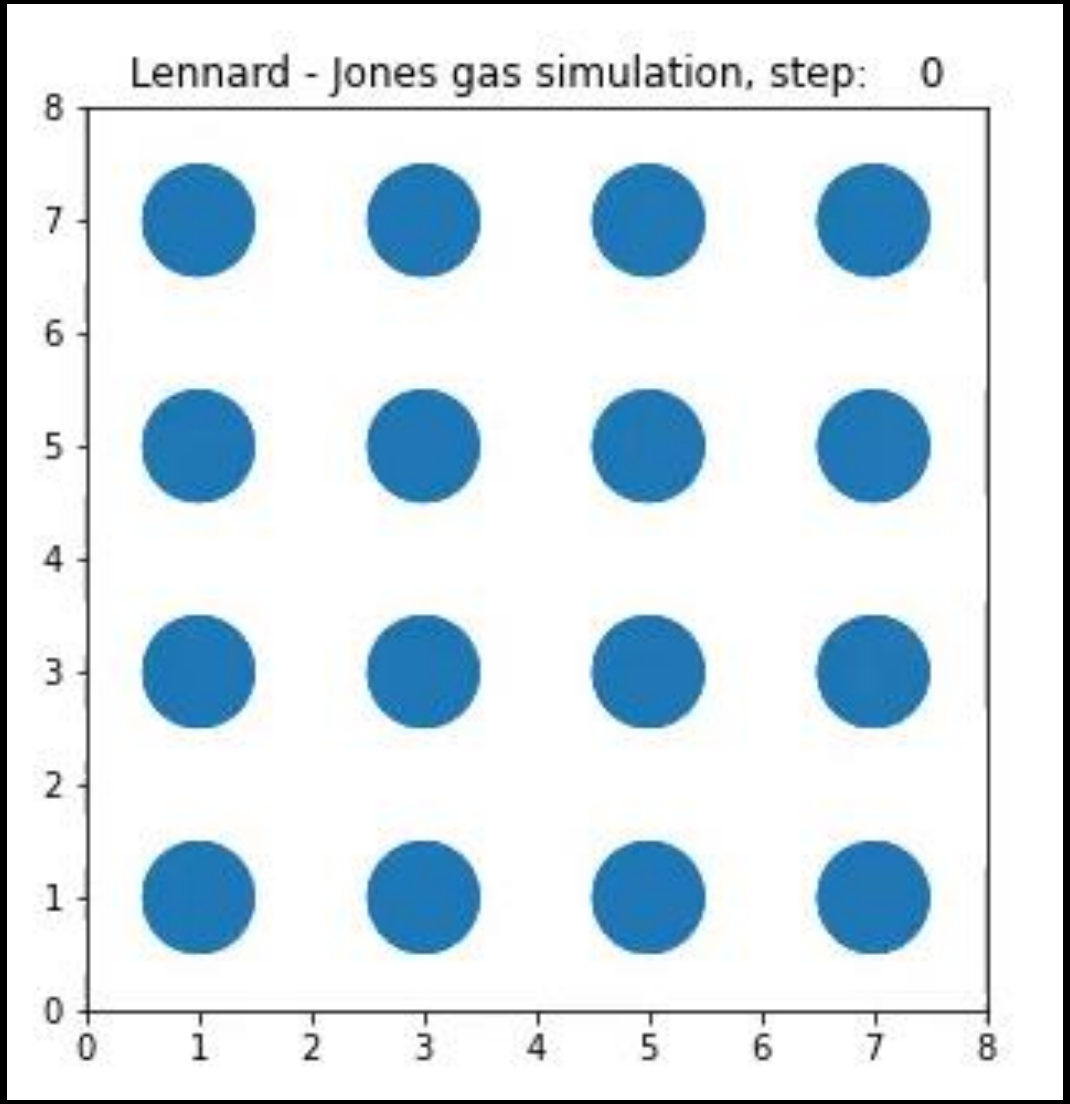
ffmpeg -f image2 -framerate 20 -i "img%06d.png" animation.avi

(https://www.ffmpeg.org/)

używając *ffmpeg* musimy mieć klatki ponumerowane sekwencją co 1 bez przerw (0000, 0001, 0002, 0003, 0004, …) a nie np. co 10: 0000, 0010, 0020, 0030, …)

- Ściągnąć i zainstalować jeden z programów.
- Komendy wpisujemy w terminalu.

Animacja



Istotna uwaga:

- operacja obliczania sił jest rzędu $O(N^2)$,
- bardzo kosztowne dla dużych układów,
- istnieją efektywne metody O(N) aby to przyspieszyć (lista sąsiadów, lista komórek).

Punktowanie zadania:

1. Implementacja „gazu doskonałego” (swobodne cząstki: $r(t + \mathrm{d}t) = r(t) + v\,\mathrm{d}t$) – brak sił, periodyczne warunki brzegowe, rysowanie poszczególnych klatek – 0.5 pkt.

2. Dodanie sił z wykorzystaniem najbliższych obrazów, aktualizowanie prędkości (Żabka) – 0.5 pkt.

*3. Wyznaczanie chwilowej temperatury i ciśnienia w układzie – 0.2 pkt.

$$\mathcal{T} = \frac{2}{2Nk_B} K \qquad \mathcal{P} = \frac{Nk\mathcal{T}}{V} + \frac{1}{2V} \sum_{\text{pary i, j}} \vec{r}_{ij} \cdot \vec{F}_{ij}$$

dwa wymiary

uwaga na znak siły, w symulacji liczymy F_{ji}