

The Nature of Code

by Daniel Shiffman

<https://github.com/edycoleee/nature>

<https://editor.p5js.org/natureofcode/collections>

MEMPELAJARI ANIMASI GERAKAN BOLA DENGAN MEMANFAATKAN RANDOM

Fungsi	Jenis Output	Contoh Output	Kegunaan Umum
<code>random.randint(a, b)</code>	Integer	2	Angka bulat acak dalam rentang [a, b]
<code>random.uniform(a, b)</code>	Float	-0.5, 0.72	Angka desimal acak dalam rentang [a, b]
<code>random.choice(list)</code>	Elemen list	-1, 0, 1	Memilih elemen acak dari daftar (list)
<code>random.random()</code>	Float (0-1)	0.44, 0.99	Nilai acak antara 0 dan 1 (probabilitas)
<code>random.gauss(mean, std)</code>	Float	-1.2, 2.5	Data acak dengan distribusi Gaussian (normal)

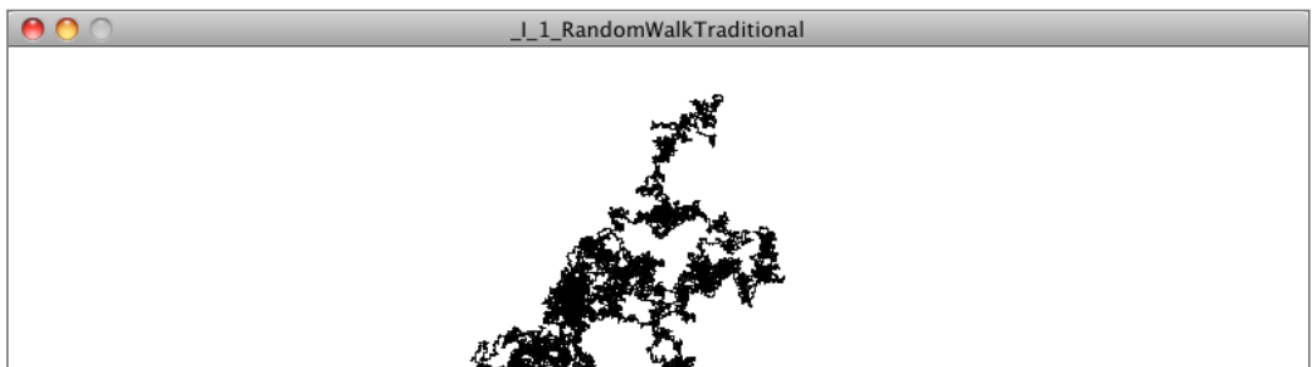
Teknik	Output / Karakteristik	Kegunaan Umum
Monte Carlo	Nilai random/estimasi statistik	Estimasi numerik, simulasi probabilitas, optimisasi
Perlin Noise	Nilai noise halus & berkesinambungan	Texturing grafis, simulasi alam, animasi

I.1 Random Walks



Apa itu Random Walk?

Random Walk itu artinya "**jalan-jalan tanpa arah pasti**" — seperti kalau kamu tutup mata dan setiap langkah kamu ambil arah yang acak (kadang ke kiri, kadang ke kanan, ke atas, atau ke bawah).



Analogi Sederhana

Bayangkan kamu adalah seekor **semut** di tengah-tengah meja.

Setiap detik, kamu akan:

- **Lempar dadu 4 sisi**
- Kalau keluar angka:
 - 1 → kamu jalan ke kanan 1 langkah
 - 2 → kamu jalan ke kiri 1 langkah
 - 3 → kamu jalan ke bawah 1 langkah
 - 4 → kamu jalan ke atas 1 langkah

Lalu kamu **tandai posisi kamu sekarang** dengan spidol.

Ulang terus setiap detik. Lama-lama, kamu akan lihat **jejak acak** yang kamu buat — seperti **peta perjalanan semut tanpa tujuan!**

Fungsi `random.randint(0, 3)` menghasilkan **angka bulat acak** antara **0 dan 3** (termasuk 0 dan 3), Setiap kali program dijalankan, hasilnya bisa 0, 1, 2, atau 3.

```
# SCRIPT 1 - PEMAHAMAN FUNGSI random.randint(0, 3)
import random

# melihat 1x hasil fungsi randint(0, 3)
r = random.randint(0, 3)
print(r)

# melihat 10x hasil fungsi randint(0, 3)
result = [] # list kosong
# Loop sebanyak 10 kali
for _ in range(10):
    r = random.randint(0, 3) # ambil angka acak antara 0 dan 3
    result.append(r) # masukkan ke dalam list result
print("10 number randint(0,3):", result) #10 number randint(0,3): [1, 0, 3, 0, 0, 3, 0, 0, 2, 3]
```


Penjelasan Kode Python (Random Walk)

Mari kita jelaskan bagian kodenya seperti cerita:


Mulai dari posisi tengah canvas (kertas)

x = WIDTH // 2

y = HEIGHT // 2

 "Semut mulai dari tengah-tengah kertas."

r = random.randint(0, 3)

 "Semut melempar dadu angka 0 sampai 3 untuk memilih arah."

if r == 0:

 x += 1 # ke kanan

elif r == 1:


 x -= 1 # ke kiri

elif r == 2:

 y += 1 # ke bawah

elif r == 3:

 y -= 1 # ke atas

 "Berdasarkan angka tadi, semut melangkah satu titik."

canvas.create_oval(x, y, x+2, y+2, fill="white", outline="")

 "Semut meninggalkan jejak titik putih di tempat dia sekarang."

root.after(10, random_walk)

 "Tunggu sebentar (10 milidetik), lalu semut jalan lagi."

Apa yang Terjadi Setelah Beberapa Menit?

- Banyak titik-titik putih muncul di layar
- Jejaknya terlihat **acak dan tak beraturan**
- Tapi semua berawal dari **satu titik di tengah**

Manfaat Random Walk

Meski kelihatannya cuma jalan acak, konsep ini dipakai dalam:

- Simulasi pergerakan partikel
- Ekonomi dan saham (harga saham bisa naik-turun secara acak)
- AI dan game (untuk gerakan musuh atau NPC)

Kesimpulan

Random walk itu seperti **main game semut yang bingung jalan ke mana**.
Setiap saat dia pilih arah **secara acak**, lalu terus melangkah.
Lama-lama, semut meninggalkan **jejak acak yang unik**.

```
#SCRIPT 2 - Random Walk - 4 ARAH
import tkinter as tk
import random

# Ukuran canvas
WIDTH = 400
HEIGHT = 400

# Posisi awal
x = WIDTH // 2
y = HEIGHT // 2

# Variabel untuk menyimpan informasi posisi
info_text = None
step_counter = 0

# Fungsi random walk
def random_walk():
    global x, y, info_text, step_counter

    # Gambar titik di posisi saat ini
    canvas.create_oval(x, y, x+2, y+2, fill="white", outline="")

    # Pilih arah gerakan secara acak
    r = random.randint(0, 3)
    if r == 0:
        x += 1
        direction = "Kanan"
    elif r == 1:
        x -= 1
        direction = "Kiri"
    elif r == 2:
        y += 1
        direction = "Bawah"
    elif r == 3:
        y -= 1
        direction = "Atas"

    step_counter += 1

    # Update informasi posisi
    update_info(direction)

    # Batasi agar tidak keluar dari canvas
    x = max(0, min(WIDTH, x))
    y = max(0, min(HEIGHT, y))

    # Loop terus-menerus setiap 10ms membuat Langkah baru
    root.after(10, random_walk)

def update_info(direction):
    global info_text

    # Hapus teks lama jika ada
    if info_text:
        canvas.delete(info_text)
```

```

# Buat teks informasi baru
info = f"Posisi: ({x}, {y})\nGerakan: {direction}\nLangkah: {step_counter}"
info_text = canvas.create_text(10, 10, anchor="nw", text=info, fill="white",
font=("Arial", 10))

# Setup GUI
root = tk.Tk()
root.title("Random Walk - Posisi dan Gerakan")

canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="black")
canvas.pack()

# Tambahkan penjelasan
description = """Simulasi Random Walk:
- Titik putih akan bergerak acak ke 4 arah
- Informasi posisi dan gerakan ditampilkan di pojok kiri atas
- Setiap langkah dicatat dan ditampilkan"""
canvas.create_text(WIDTH//2, HEIGHT-30, text=description, fill="yellow", font=("Arial",
8))

random_walk()
root.mainloop()

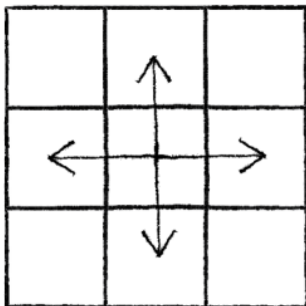
```

I.2 The Random Walker Class

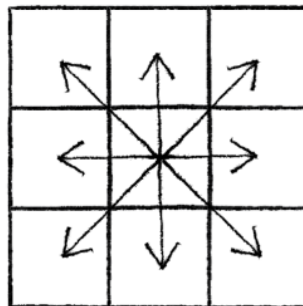
Bagus! Kamu sudah paham konsep dasar random walk. Sekarang mari kita lanjut **menjelaskan lanjutan bab tersebut** dengan sederhana, agar mengerti konsep-konsep lebih lanjut seperti variasi gerak, distribusi acak, dan pseudo-random.



1. Objek "Walker" yang Bisa Gerak ke Semua Arah



4 possible steps



8 possible steps

💡 Masalah:

Sebelumnya, kita hanya bisa gerak ke **kanan, kiri, atas, atau bawah** (4 arah). Sekarang kita ingin bergerak ke **8 arah** (termasuk diagonal) **atau diam di tempat**.

✅ Solusi:

Daripada membuat 9 pilihan manual, kita buat dua angka acak:

```
int stepx = int(random(3)) - 1; // hasil: -1, 0, atau 1
```

```
int stepy = int(random(3)) - 1;
```

Contoh hasil:

- (-1, -1) → kiri atas
- (1, 0) → kanan
- (0, 1) → bawah
- (0, 0) → tidak bergerak



Total 9 kemungkinan, masing-masing kemungkinan 1/9 (11.1%)

SCRIPT 3 - PEMAHAMAN FUNGSI random.randint(-1, 1)

```
import random

# Melihat 1x hasil random.randint(-1, 1) pada koordinat (x,y)
dx = random.randint(-1, 1)
dy = random.randint(-1, 1)
print("1x koordinat acak (dx, dy):", dx, dy) # 1x koordinat acak (dx, dy): 1 -1

# Melihat 10x hasil fungsi random.randint(-1, 1)
result = [] # list kosong untuk menyimpan pasangan (dx, dy)

# Loop sebanyak 10 kali
for _ in range(10):
    dx = random.randint(-1, 1) # ambil angka acak antara -1 dan 1
    dy = random.randint(-1, 1) # ambil angka acak antara -1 dan 1
    result.append((dx, dy)) # masukkan pasangan tuple ke dalam list

# Tampilkan hasil
print("10 pasangan koordinat acak (dx, dy):", result)
# 10 pasangan koordinat acak (dx, dy): [(0, -1), (1, 0), (0, 0), (-1, 1), (1, -1), (0, 1), (-1, 0), (0, 0), (1, 1), (-1, -1)]
```

2. Gunakan Angka Pecahan (Floating Point)

Dengan float, kita bisa bergerak **lebih halus** dan bukan cuma satu piksel per langkah:

```
float stepx = random(-1, 1); // antara -1.0 hingga 1.0
```

```
float stepy = random(-1, 1);
```

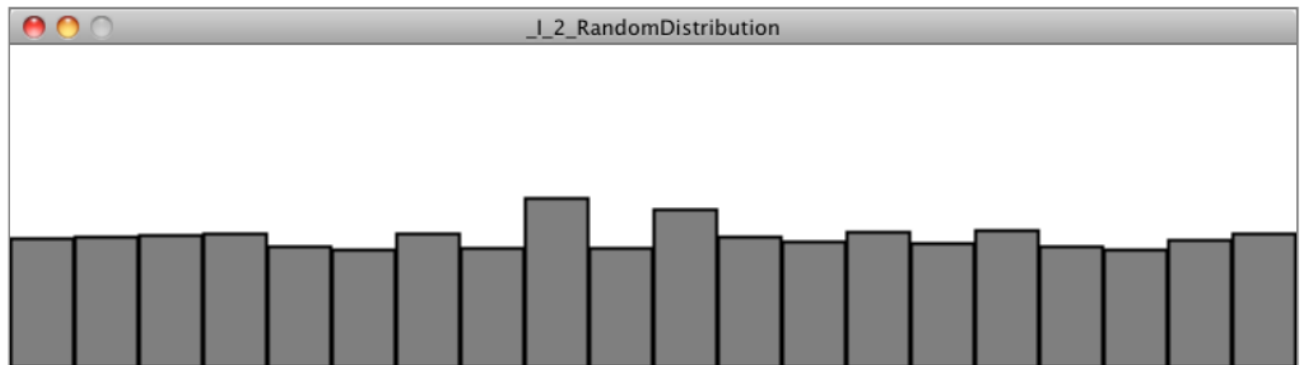
```
x += stepx;
```

```
y += stepy;
```

Ini seperti semut yang bisa "meluncur" ke arah manapun, bukan hanya lompat ke kotak sebelah.

3. Distribusi Acak dan Grafik

Penulis menjelaskan bagaimana kita bisa **mengukur seberapa acak angka acak** itu.



Eksperimen:

- Kita buat array randomCounts untuk menghitung **berapa kali** setiap angka acak keluar.
- Kita acak angka dari 0–19 (random(20))
- Setiap kali angka muncul, kita tambah 1 di array itu.



Hasil: Setiap bar menunjukkan seberapa sering angka tertentu keluar. Kalau benar-benar acak, semua bar akan mirip tingginya **setelah waktu cukup lama**.



Apa Itu "Pseudo-Random"?

Komputer **tidak bisa membuat angka benar-benar acak**. Mereka memakai rumus matematika untuk **mensimulasikan keacakan**.



Disebut "**pseudo-random**" — terlihat acak, tapi sebenarnya ada pola tersembunyi.

Namun, pola ini **sangat panjang**, jadi untuk keperluan kita (grafik, animasi, game), pseudo-random itu sudah cukup "**seperti acak sungguhan**."

Latihan (Exercise 1.1)

Tujuan:

Buat Walker yang **lebih sering jalan ke bawah dan ke kanan**, bukan acak sepenuhnya.

Petunjuk:

Kita bisa kasih **bobot (bias)** ke arah tertentu.

Contoh (dalam Python-style pseudocode):

Gerak dengan bias ke kanan dan bawah

stepx = random.choice([-1, 0, 1, 1]) # kanan lebih mungkin

stepy = random.choice([-1, 0, 1, 1]) # bawah lebih mungkin

Karena angka 1 muncul dua kali, kemungkinan ke kanan/bawah jadi **lebih besar**.

Kesimpulan untuk Kamu dan Anakmu:

Konsep	Penjelasan Sederhana
Random Walk	Jalan acak dari satu titik ke titik lain
8 arah atau diam	Gunakan dua angka acak (x dan y) dengan hasil -1, 0, 1
Floating point	Biar bisa jalan lebih halus, bukan hanya 1 piksel
Distribusi Acak	Angka acak punya kemungkinan yang rata
Pseudo-random	Angka acak komputer, tidak murni acak tapi cukup baik

Tentu! Berikut saya berikan:

1. **Penjelasan tiap fungsi dengan tag #fungsi...**
2. **Contoh kode Python + Tkinter** untuk:
 - Random walker ke semua arah
 - Random walker halus (floating point)
 - Random walker condong ke kanan-bawah

Penjelasan Fungsi dalam Program Tkinter Random Walk

```
import tkinter as tk
```

```
import random
```

◆ **import**: Memanggil modul Tkinter untuk GUI dan random untuk angka acak.

```
WIDTH = 400
```

```
HEIGHT = 400
```

◆ **WIDTH, HEIGHT**: Ukuran kanvas (area tempat jalan).

```
x = WIDTH // 2
```

```
y = HEIGHT // 2
```

◆ **Posisi awal x, y**: Di tengah kanvas, seperti semut mulai dari tengah kertas.

```
def random_walk():
```

```
    #fungsi utama untuk menggambar titik dan menggerakkan walker secara acak
```

```
canvas.create_oval(x, y, x+2, y+2, fill="white", outline="")
```

◆ **create_oval**: Menggambar titik kecil di posisi (x, y). Ini seperti semut meninggalkan jejak.

```
root.after(10, random_walk)
```

◆ **after(10, random_walk):** Menjadwalkan fungsi ini dipanggil lagi 10 milidetik kemudian. Seperti draw() di Processing.

✓ 1. Random Walker ke 8 Arah (termasuk diam)

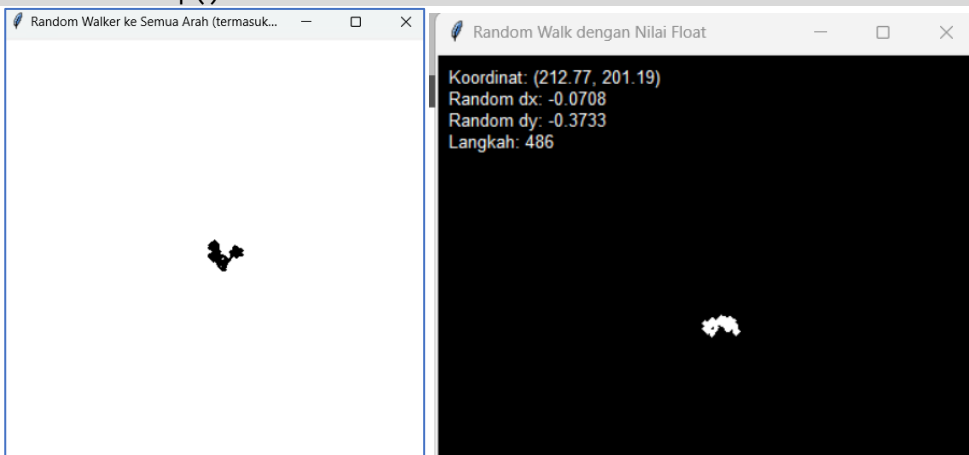
```
#SCRIPT4 - Random Walker ke 8 Arah (termasuk diam)
import tkinter as tk
import random

WIDTH, HEIGHT = 400, 400
x, y = WIDTH // 2, HEIGHT // 2

def random_walk():
    global x, y
    canvas.create_oval(x, y, x + 2, y + 2, fill="black", outline="")

    #fungsi random gerak ke 8 arah + diam
    dx = random.randint(-1, 1)
    dy = random.randint(-1, 1)
    x += dx
    y += dy
    # Loop terus-menerus setiap 10ms membuat Langkah baru
    root.after(10, random_walk)

root = tk.Tk()
root.title("Random Walker ke Semua Arah (termasuk diam)")
canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()
random_walk()
root.mainloop()
```



✓ 2. Random Walker Halus (Floating Point)

```
# SCRIPT 5 - PEMAHAMAN FUNGSI random.uniform(-1, 1)
import random

# Melihat 1x hasil random.uniform(-1, 1) pada koordinat (x,y)
dx = random.uniform(-1, 1)
dy = random.uniform(-1, 1)
print("1x koordinat acak (dx, dy):", dx, dy) # 1x koordinat acak (dx, dy): 0.352 -0.748

# Melihat 10x hasil fungsi random.uniform(-1, 1)
result = [] # list kosong untuk menyimpan pasangan (dx, dy)

# Loop sebanyak 10 kali
```

```

for _ in range(10):
    dx = random.uniform(-1, 1) # ambil angka desimal acak antara -1 dan 1
    dy = random.uniform(-1, 1) # ambil angka desimal acak antara -1 dan 1
    result.append((dx, dy)) # masukkan pasangan tuple ke dalam list

# Tampilkan hasil
print("10 pasangan koordinat acak (dx, dy):", result)
# 10 pasangan koordinat acak (dx, dy): [(0.23, -0.92), (-0.11, 0.88), ..., (0.01, -0.37)]

```

IMPLEMENTASI PADA RANDOM WALKER

#SCRIPT 6 - Random Walk dengan Nilai Float

```

import tkinter as tk
import random

```

```

# Ukuran canvas
WIDTH, HEIGHT = 400, 400

```

```

# Posisi awal (menggunakan float)
x, y = WIDTH / 2, HEIGHT / 2

```

```

# Variabel untuk menyimpan informasi
info_text = None
step_counter = 0
last_dx, last_dy = 0.0, 0.0

```

```

def random_walk():
    global x, y, info_text, step_counter, last_dx, last_dy

    # Gambar titik di posisi saat ini (dikonversi ke integer untuk canvas)
    canvas.create_oval(int(x), int(y), int(x)+2, int(y)+2, fill="white", outline="")

    # Gerakan acak dengan nilai float antara -1.0 sampai 1.0
    last_dx = random.uniform(-1, 1)
    last_dy = random.uniform(-1, 1)
    x += last_dx
    y += last_dy

    step_counter += 1

    # Update informasi posisi
    update_info()

    # Batasi agar tidak keluar dari canvas
    x = max(0, min(WIDTH, x))
    y = max(0, min(HEIGHT, y))
    # Loop terus-menerus setiap 10ms membuat Langkah baru
    # Loop terus-menerus
    root.after(10, random_walk)

```

```

def update_info():
    global info_text

    # Hapus teks lama jika ada
    if info_text:

```



```

canvas.delete(info_text)

# Buat teks informasi baru
info = (f"Koordinat: ({x:.2f}, {y:.2f})\n"
        f"Random dx: {last_dx:.4f}\n"
        f"Random dy: {last_dy:.4f}\n"
        f"Langkah: {step_counter}")
info_text = canvas.create_text(10, 10, anchor="nw", text=info,
                               fill="white", font=("Arial", 10))

# Setup GUI
root = tk.Tk()
root.title("Random Walk dengan Nilai Float")

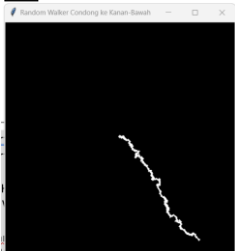
canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="black")
canvas.pack()

# Tambahkan penjelasan
description = """Simulasi Random Walk dengan Nilai Float:
- Pergerakan menggunakan nilai random antara -1.0 sampai 1.0
- Koordinat menggunakan presisi float
- Informasi ditampilkan di pojok kiri atas:
  * Posisi aktual (x,y)
  * Nilai random dx dan dy terakhir
  * Total langkah"""
canvas.create_text(WIDTH//2, HEIGHT-50, text=description,
                  fill="yellow", font=("Arial", 8))

random_walk()
root.mainloop()

```

✓ 3. Random Walker Condong ke Kanan-Bawah



```

# SCRIPT 7 - PEMAHAMAN FUNGSI random.choice([-1, 0, 1, 1])
# -1: 25% 0: 25% 1: 50%

import random

# Melihat 1x hasil random.choice pada koordinat (x,y)
dx = random.choice([-1, 0, 1, 1]) # arah kanan lebih sering
dy = random.choice([-1, 0, 1, 1]) # arah bawah lebih sering
print("1x koordinat acak (dx, dy):", dx, dy) # 1x koordinat acak (dx, dy): 1 1

# Melihat 10x hasil fungsi random.choice
result = [] # list kosong untuk menyimpan pasangan (dx, dy)

# Loop sebanyak 10 kali

```

```

for _ in range(10):
    dx = random.choice([-1, 0, 1, 1]) # kanan lebih sering
    dy = random.choice([-1, 0, 1, 1]) # bawah lebih sering
    result.append((dx, dy))          # simpan sebagai pasangan tuple

# Tampilkan hasil
print("10 pasangan koordinat acak (dx, dy):", result)
# 10 pasangan koordinat acak (dx, dy): [(1, 1), (1, 0), (0, 1), (-1, 1), (1, -1), (1, 0), (1, 1), (0, 1), (-1, 1), (1, 1)]

```

IMPLEMENTASI PADA RANDOM WALKER

SCRIPT 8 - Random Walker Condong ke Kanan-Bawah

```
import tkinter as tk
```

```
import random
```

```
WIDTH, HEIGHT = 400, 400
```

```
x, y = WIDTH // 2, HEIGHT // 2
```

```
def random_walk():
```

```
    global x, y
```

```
    canvas.create_oval(x, y, x + 2, y + 2, fill="white", outline="")
```

```
    #fungsi bias ke kanan dan bawah
```

```
    dx = random.choice([-1, 0, 1, 1]) # kanan lebih sering
```

```
    dy = random.choice([-1, 0, 1, 1]) # bawah lebih sering
```

```
    x += dx
```

```
    y += dy
```

```
    # Loop terus-menerus setiap 10ms membuat Langkah baru
```

```
    root.after(10, random_walk)
```

```
root = tk.Tk()
```

```
root.title("Random Walker Condong ke Kanan-Bawah")
```

```
canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="black")
```

```
canvas.pack()
```

```
random_walk()
```

```
root.mainloop()
```



Tips

- Gunakan **analogi semut berjalan di kertas**
- Mengganti angka untuk melihat efeknya
- Bandingkan semua versi untuk mengamati hasil geraknya
- Membuat prediksi: "Kalau kanan-bawah lebih sering, bakal jadi seperti apa?"



Bab I.3: Probability and Non-Uniform Distributions



Tujuan Bagian Ini

Memahami bahwa **random()** itu bagus, tapi **belum cukup untuk meniru alam**. Kita butuh **probabilitas** dan **distribusi yang tidak seragam (non-uniform)** untuk hasil yang lebih alami, realistis, atau selektif.



Analogi Awal

Saat kita mulai ngoding visual (misalnya di Processing atau Tkinter), sering kali kita asal pakai **random()**:

- Lokasi lingkaran acak
- Ukuran acak
- Warna acak



Ini memang seru, tapi hasilnya **acak total (tidak alami)**. Misalnya:

- Kita ingin simulasi pohon tumbuh → tidak cukup dengan acak
- Kita ingin simulasi evolusi → harus ada yang lebih "fit" punya peluang lebih besar untuk berkembang biak



Distribusi Uniform vs Non-Uniform

- **Uniform Distribution** (acak biasa):

Semua kemungkinan punya **peluang yang sama**.

Misalnya:

`random.randint(0, 3)` # Hasil: 0, 1, 2, atau 3 → semua punya peluang 25%

- **Non-Uniform Distribution:**

Beberapa kemungkinan punya peluang **lebih besar** dari yang lain.

Misalnya:

- Dalam simulasi evolusi: monyet kuat = 90% peluang bereproduksi, lemah = 10%
- Dalam simulasi angin: arah barat lebih dominan dari timur



Dasar-dasar Probabilitas

Definisi:

Probabilitas suatu kejadian =

$\frac{\text{jumlah cara kejadian itu bisa terjadi}}{\text{total semua kemungkinan}}$

Contoh:

- Lempar koin: kepala atau ekor → $1/2 = 50\%$
- Ambil kartu dari 52 kartu, peluang dapat As = $4/52 = \sim 7.7\%$
- Peluang dapat kartu diamond = $13/52 = 25\%$



Probabilitas Berurutan (Multiple Events)

Jika kita ingin tahu **dua kejadian berurutan**, kita **kalikan peluang masing-masing**:

Contoh:

- Lempar 3x koin dan semuanya kepala:
- $1/2 * 1/2 * 1/2 = 1/8$

Artinya: peluangnya hanya 12.5%



Latihan 1.2: Apa peluang ambil 2 kartu As berturut-turut dari dek 52 kartu?



Langkah 1:

- Peluang ambil **As pertama**:
- 4 As dari 52 kartu → $4/52$



Langkah 2:

- Setelah satu As diambil, tinggal **3 As dari 51 kartu**:
- $3/51$



Total Peluang:

$4/52 * 3/51 = (1/13) * (1/17) \approx 0.0045$ atau 0.45%



Jadi peluang dapat 2 As berturut-turut $\approx 0.45\%$



Aplikasi dalam Pemrograman

Di sistem evolusi, kita tidak bisa asal pakai `random.choice`. Kita perlu buat **peluang terkontrol**:

Contoh pseudo-code:

```
population = [fit_monkey]*90 + [weak_monkey]*10
```

```
parent = random.choice(population)
```

Atau lebih efisien:

```
if random.random() < 0.9:
```

```
    parent = fit_monkey
```

```
else:
```

```
    parent = weak_monkey
```

Catatan Penting

- Random() menghasilkan **pseudo-random numbers** (tidak benar-benar acak, tapi cukup baik untuk simulasi)
- Non-uniform distribution **meniru dunia nyata**
- Probabilitas penting untuk:
 - Evolusi
 - Mutasi
 - Simulasi cuaca
 - Simulasi pergerakan makhluk hidup

Cara Menjelaskan

- Bayangkan ada **kotak undian berisi 100 bola**
 - 90 merah (monyet kuat), 10 biru (monyet lemah)
 - Pilih 1 bola secara acak → peluang merah = 90%
- Dunia nyata **tidak selalu adil**. Beberapa punya peluang lebih besar, dan itu yang ingin kita simulasikan.

I.3 (Lanjutan): Mengontrol Probabilitas dalam Kode

Tujuan Bagian Ini

Menunjukkan **dua teknik populer** untuk mengatur probabilitas di kode:

1. Dengan **array berisi elemen berulang**
2. Dengan **rentang angka acak (random float)**


1. Metode Array Berulang

Misal:


```
int[] stuff = new int[5];
stuff[0] = 1;
stuff[1] = 1;
stuff[2] = 2;
stuff[3] = 3;
stuff[4] = 3;
int index = int(random(stuff.length));
int pilihan = stuff[index];
```

Penjelasan:

- Nilai 1 muncul **2 kali**
- Nilai 2 muncul **1 kali**
- Nilai 3 muncul **2 kali**

 Peluang terpilih:

- 1 → 2 dari 5 → **40%**
- 2 → 1 dari 5 → **20%**
- 3 → 2 dari 5 → **40%**

 **Kelebihan:** mudah, intuitif

 **Kekurangan:** tidak fleksibel untuk probabilitas yang sangat presisi

2. Metode Rentang Floating Point (random float)

Kita ambil angka acak random(1) yang hasilnya antara 0.0 - 1.0.

Contoh:

```
float prob = 0.10;
float r = random(1);
if (r < prob) {
    // kejadian terjadi dengan peluang 10%
```

```
}
```

💡 Untuk Banyak Kejadian

Misalnya ada 3 kemungkinan:

- A → 60%
- B → 10%
- C → 30%

Maka kita tulis:

```
float num = random(1);
```

```
if (num < 0.6) {
```

```
    println("A");
```

```
} else if (num < 0.7) {
```

```
    println("B");
```

```
} else {
```

```
    println("C");
```

```
}
```

📊 Rentang yang setara dengan peluang:

- 0.0 – 0.6 → A
- 0.6 – 0.7 → B
- 0.7 – 1.0 → C

✅ **Kelebihan:** presisi tinggi, cocok untuk kontrol yang detail

✅ Fleksibel untuk probabilitas dinamis

🕹️ Contoh: Random Walker Cenderung ke Kanan

Peluang:

- Kanan: 40%
- Kiri: 20%
- Atas: 20%
- Bawah: 20%

```
void step() {
```

```
    float r = random(1);
```

```
    if (r < 0.4) {
```

```
        x++;    // 40%
```

```
    } else if (r < 0.6) {
```

```
        x--;    // 20%
```

```
    } else if (r < 0.8) {
```

```
        y++;    // 20%
```

```
    } else {
```

```
        y--;    // 20%
```

```
    }
```

```
}
```

🎯 Latihan I.3

Buat random walker dengan peluang bergerak ke arah mouse sebesar 50%.

🐍 Versi Python + Tkinter

Rentang nilai r	Arah gerak	Penjelasan
$r < 0.4$	$x += 1$	ke kanan (40%)
$0.4 \leq r < 0.6$	$x -= 1$	ke kiri (20%)
$0.6 \leq r < 0.8$	$y += 1$	ke bawah (20%)
$r \geq 0.8$	$y -= 1$	ke atas (20%)

```

# SCRIPT 9 - PEMAHAMAN FUNGSI random.random() dengan probabilitas arah
import random

# Melihat 1x hasil arah acak berdasarkan random.random()
dx = 0
dy = 0
r = random.random() # angka acak 0.0 - 1.0

if r < 0.4:
    dx = 1 # ke kanan
elif r < 0.6:
    dx = -1 # ke kiri
elif r < 0.8:
    dy = 1 # ke bawah
else:
    dy = -1 # ke atas

print("1x koordinat acak (dx, dy):", dx, dy) # 1x koordinat acak (dx, dy): 1 0

# Melihat 10x hasil gerakan acak dengan probabilitas
result = [] # list kosong untuk menyimpan pasangan (dx, dy)

for _ in range(10):
    dx = 0
    dy = 0
    r = random.random()

    if r < 0.4:
        dx = 1
    elif r < 0.6:
        dx = -1
    elif r < 0.8:
        dy = 1
    else:
        dy = -1

    result.append((dx, dy)) # simpan arah gerak sebagai pasangan (dx, dy)

# Tampilkan hasil
print("10 pasangan koordinat acak (dx, dy):", result)
# 10 pasangan koordinat acak (dx, dy): [(1, 0), (1, 0), (0, 1), (-1, 0), (0, 1), (1, 0), (0, -1), (1, 0), (1, 0), (-1, 0)]

```

Random Walker: Cenderung ke kanan (probabilitas)

```

# SCRIPT 10 - Random Walker: Cenderung ke kanan (probabilitas)
import tkinter as tk
import random

class Walker:
    def __init__(self, canvas, width, height):
        """
        Inisialisasi objek Walker (pejalan acak)

        Parameter:

```

- canvas: objek Canvas tkinter tempat walker digambar
- width: lebar area pergerakan (pixel)
- height: tinggi area pergerakan (pixel)

```
"""
```

```
self.canvas = canvas
self.width = width
self.height = height
self.x = width // 2 # Posisi x awal di tengah
self.y = height // 2 # Posisi y awal di tengah
```

```
def step(self):
```

```
    """
```

```
    Bergerak satu langkah acak dengan probabilitas:
```

```
    - 40% ke kanan (x+1)
```

```
    - 20% ke kiri (x-1)
```

```
    - 20% ke bawah (y+1)
```

```
    - 20% ke atas (y-1)
```

```
    """
```

```
    r = random.random() # Generate angka acak 0-1
```

```
    if r < 0.4:
```

```
        self.x += 1
```

```
    elif r < 0.6:
```

```
        self.x -= 1
```

```
    elif r < 0.8:
```

```
        self.y += 1
```

```
    else:
```

```
        self.y -= 1
```

```
    # Pastikan posisi tetap dalam batas canvas
```

```
    self.x = max(0, min(self.x, self.width - 1))
```

```
    self.y = max(0, min(self.y, self.height - 1))
```

```
    # Gambar titik di posisi baru
```

```
    self.canvas.create_oval(self.x, self.y, self.x+2, self.y+2, fill="black")
```

```
def animate():
```

```
    """Fungsi animasi yang memanggil step() berulang"""
```

```
    walker.step()
```

```
    root.after(10, animate) # Panggil lagi setelah 10ms
```

```
# Setup GUI utama
```

```
root = tk.Tk()
```

```
root.title("Random Walker: Cenderung ke kanan (probabilitas)")
```

```
w, h = 400, 400 # Ukuran canvas
```

```
canvas = tk.Canvas(root, width=w, height=h, bg="white")
```

```
canvas.pack()
```

```
# Buat objek Walker dan mulai animasi
```

```
walker = Walker(canvas, w, h)
```

```
animate() # Mulai animasi
```

```
root.mainloop() # Jalankan aplikasi
```



Bagus! Mari kita buat versi **random walker dengan 50% peluang bergerak menuju arah kursor mouse**, sisanya 50% bergerak secara acak.

Logika:

1. **Dapatkan posisi mouse** (mouse_x, mouse_y)
2. Hitung **arah relatif** mouse terhadap walker:
 - Jika mouse_x > x → mouse di kanan → arah_x = 1
 - Jika mouse_x < x → mouse di kiri → arah_x = -1
 - Jika sama → arah_x = 0
3. Lakukan hal yang sama untuk arah_y
4. Ambil angka acak r:
 - Jika r < 0.5 → bergerak menuju arah mouse
 - Jika r >= 0.5 → gerak acak (-1, 0, atau 1)

Kode Python Tkinter: Walker menuju mouse 50%

```
# SCRIPT 11 - Walker menuju mouse 50%
import tkinter as tk
import random

class Walker:
    def __init__(self, canvas, width, height):
        """
        Initialize a random walker that partially follows mouse movement

        Parameters:
        - canvas (tk.Canvas): The drawing canvas
        - width (int): Canvas width in pixels
        - height (int): Canvas height in pixels
        """
        self.canvas = canvas
        self.width = width
        self.height = height
        self.x = width // 2 # Starting X position (center)
        self.y = height // 2 # Starting Y position (center)
        self.mouse_x = self.x # Tracked mouse X position
        self.mouse_y = self.y # Tracked mouse Y position

    def update_mouse(self, event):
        """
        Update mouse coordinates when mouse moves

        Parameters:
```



```
- event (tk.Event): Mouse motion event containing:
- x (int): Current mouse X position
- y (int): Current mouse Y position
"""
```

```
self.mouse_x = event.x
self.mouse_y = event.y
```

```
def step(self):
```

```
    """
```

```
    Move the walker one step:
```

```
- 50% chance: Move toward mouse (1 pixel)
- 50% chance: Random movement (-1, 0, or 1 pixels)
    """
```

```
    if random.random() < 0.5:
```

```
        # Move toward mouse (deterministic)
```

```
        # stepx: nilai +1 jika mouse ada di kanan, -1 jika di kiri, 0 jika sejajar.
```

```
        #stepx = 1 if self.mouse_x > self.x else -1 if self.mouse_x < self.x else 0
```

```
        if self.mouse_x > self.x:
```

```
            stepx = 1
```

```
        elif self.mouse_x < self.x:
```

```
            stepx = -1
```

```
        else:
```

```
            stepx = 0
```

```
        # stepy: nilai +1 jika mouse ada di bawah, -1 jika di atas, 0 jika sejajar.
```

```
        #stepy = 1 if self.mouse_y > self.y else -1 if self.mouse_y < self.y else 0
```

```
        if self.mouse_y > self.y:
```

```
            stepy = 1
```

```
        elif self.mouse_y < self.y:
```

```
            stepy = -1
```

```
        else:
```

```
            stepy = 0
```

```
    else:
```

```
        # Random walk (stochastic)
```

```
        stepx = random.choice([-1, 0, 1]) # Left/None/Right
```

```
        stepy = random.choice([-1, 0, 1]) # Up/None/Down
```

```
    # Apply movement with boundary checking
```

```
    self.x = max(0, min(self.x + stepx, self.width - 1))
```

```
    self.y = max(0, min(self.y + stepy, self.height - 1))
```

```
    # Draw step as small black circle
```

```
    self.canvas.create_oval(
```

```
        self.x, self.y,
```

```
        self.x + 2, self.y + 2,
```

```
        fill="black",
```

```
        outline=""
```

```
)
```

```
def animate():
```

```
    """Animation loop that steps the walker every 10ms"""
```

```
    walker.step()
```

```
    root.after(10, animate) # Schedule next frame
```

```
# Main window setup
```

```

root = tk.Tk()
root.title("Walker menuju mouse 50%") # Window title
w, h = 400, 400 # Canvas dimensions

# Create and pack canvas
canvas = tk.Canvas(
    root,
    width=w,
    height=h,
    bg="white",
    highlightthickness=0 # Remove border
)
canvas.pack()

# Initialize walker and bind mouse motion
walker = Walker(canvas, w, h)
canvas.bind("<Motion>", walker.update_mouse) # Track mouse movements

# Start animation
animate()
root.mainloop()

```

🧠 Apa yang Bisa Diajarkan?

- Konsep **if-else berbasis probabilitas**
- Perbandingan posisi → arah gerak
- Event <Motion> di Tkinter untuk menangkap gerakan mouse
- Konsep **"kecerdasan sederhana"**: objek mengikuti target secara probabilistik

I.4: A Normal Distribution of Random Numbers

Baik! Mari kita bahas **materi I.4: A Normal Distribution of Random Numbers** dalam konteks pemrograman (misalnya dengan Processing atau Python), sambil membandingkannya dengan **random uniform** yang sebelumnya kita pelajari.

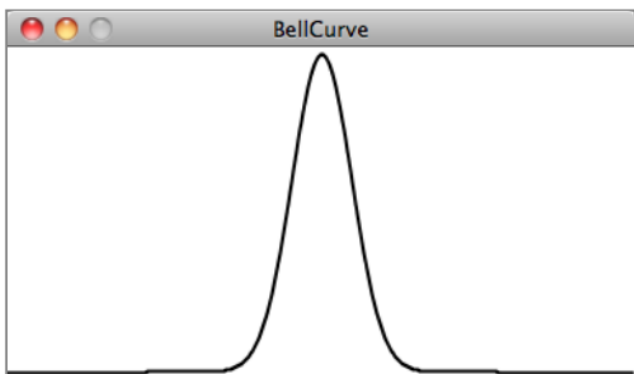


Figure I.2

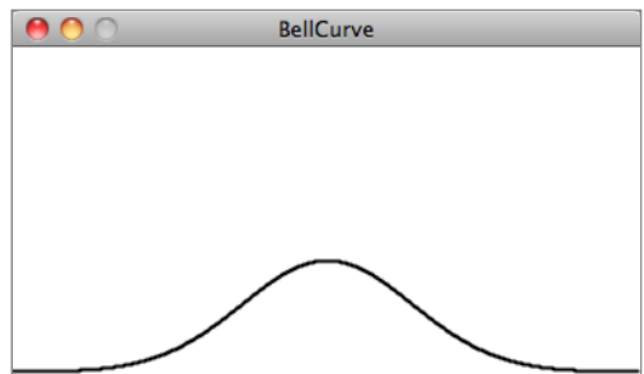


Figure I.3

📊 Apa Itu Normal Distribution (Distribusi Normal)?

Distribusi normal (atau Gaussian distribution) adalah pola distribusi di mana **nilai-nilai berpusat di sekitar rata-rata (mean)** dan **menyebar berdasarkan simpangan baku (standard deviation)**.

🔑 Ciri khasnya:

- Bentuk kurva lonceng (bell curve)
- Banyak nilai di sekitar rata-rata
- Sedikit nilai yang ekstrem (sangat besar atau sangat kecil)

Contoh Kasus: Tinggi Badan Monyet

Misalnya kita ingin membuat **seribu objek monyet dengan tinggi antara 200–300 pixel**:

```
float h = random(200, 300);
```

Ini menggunakan distribusi *uniform*, artinya **setiap tinggi punya kemungkinan yang sama**.

Tapi kenyataannya, **di dunia nyata lebih banyak makhluk dengan tinggi rata-rata**, dan hanya sedikit yang sangat pendek atau sangat tinggi. Maka kita butuh **normal distribution**.

Konsep Statistik Dasar:

- **Mean (μ)** = nilai rata-rata
- **Standard Deviation (σ)** = seberapa tersebar nilai dari rata-ratanya

 Dalam distribusi normal:

- 68% nilai berada di **$\pm 1\sigma$ dari mean**
- 95% di $\pm 2\sigma$
- 99.7% di $\pm 3\sigma$

Cara Hitung Manual (Contoh Nilai Siswa):

Nilai: 85, 82, 88, 86, 85, 93, 98, 40, 73, 83

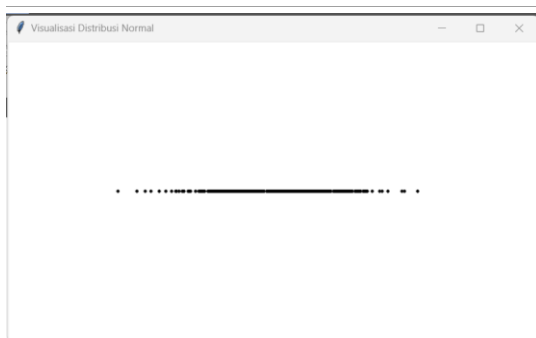
1. **Mean (rata-rata):**
(jumlah semua nilai) / (jumlah siswa) = **81.3**
2. **Standard deviation (simpangan baku):**
 - Hitung selisih dari mean: (nilai - mean)
 - Kuadratkan selisih itu → **varian**
 - Rata-rata dari semua varian
 - Akar dari rata-rata varian = **σ**

Contoh salah satu:

- Nilai: 85 → selisih dari mean = $85 - 81.3 = 3.7$
- Variansi = $3.7^2 = 13.69$

Hasil akhirnya:

- **Rata-rata varian: 254.23**
- **Simpangan baku: $\sqrt{254.23} = 15.13$**



SCRIPT 12 - PEMAHAMAN FUNGSI random.gauss(0, 1)

```
import random
```

Fungsi bantu untuk mengubah angka float jadi langkah diskrit: -1, 0, atau 1

```
def to_step(value):
```

```
    if value > 0.5:
```

```
        return 1
```

```
    elif value < -0.5:
```

```
        return -1
```

```
    else:
```

```
        return 0
```

```

# Melihat 1x hasil random.gauss pada koordinat (x, y)
dx = to_step(random.gauss(0, 1))
dy = to_step(random.gauss(0, 1))
print("1x koordinat acak (dx, dy):", dx, dy) # 1x koordinat acak (dx, dy): 1 0

# Melihat 10x hasil fungsi random.gauss
result = [] # list kosong untuk menyimpan pasangan (dx, dy)

# Loop sebanyak 10 kali
for _ in range(10):
    dx = to_step(random.gauss(0, 1)) # hasil Gaussian dikonversi ke -1, 0, 1
    dy = to_step(random.gauss(0, 1))
    result.append((dx, dy)) # simpan pasangan langkah (dx, dy)

# Tampilkan hasil
print("10 pasangan koordinat acak (dx, dy):", result)
#10 pasangan koordinat acak (dx, dy): [(-1, 0), (1, -1), (1, -1), (0, 0), (0, 0), (-1, -1), (0, 1), (0, 1), (0, 0), (-1, 1)]

```

Versi Python + Tkinter

```

# SCRIPT 13 - Visualisasi Distribusi Normal dengan Titik
import tkinter as tk
import random

class NormalDots:
    def __init__(self, canvas, width):
        """
        Visualisasi distribusi normal menggunakan titik-titik

        Parameters:
        - canvas: Objek Canvas tkinter untuk menggambar
        - width: Lebar canvas (untuk menentukan titik tengah)
        """
        self.canvas = canvas
        self.width = width # Menyimpan lebar canvas

    def draw(self):
        """
        Menggambar satu titik dengan posisi acak mengikuti distribusi normal
        """
        mean = self.width // 2 # Titik tengah canvas (rata-rata distribusi)
        sd = 60 # Standar deviasi (menentukan sebaran titik)

        # Generate angka acak dengan distribusi normal
        # random.gauss(mean, standard_deviation)
        gaussian = random.gauss(0, 1) # Mean 0, std 1 (distribusi normal standar)

        # Transformasi ke koordinat canvas
        x = sd * gaussian + mean # Skala dan geser ke posisi yang diinginkan
        y = 180 # Posisi vertikal tetap di tengah

        # Gambar titik kecil (3x3 pixel)
        self.canvas.create_oval(x, y, x+3, y+3, fill="black", outline="")

```

```
def update():
    """
    Fungsi animasi yang menggambar titik baru setiap 10ms
    """
    dots.draw()
    root.after(10, update) # Jadwalkan pemanggilan berikutnya setelah 10ms

# Setup window utama
root = tk.Tk()
root.title("Visualisasi Distribusi Normal")

# Buat canvas dengan ukuran 640x360
canvas = tk.Canvas(root, width=640, height=360, bg="white")
canvas.pack()

# Inisialisasi visualisasi
dots = NormalDots(canvas, 640)

# Mulai animasi
update()
root.mainloop()
```

🚀 Efeknya: **titik-titik akan menumpuk di tengah layar**, sedikit di pinggir → itulah efek **normal distribution**.

🏠 Ringkasan Sederhana:

- `random()` = semua nilai punya peluang sama
- `gauss()` atau `nextGaussian()` = lebih sering muncul di sekitar nilai tengah
- Digunakan untuk meniru **sifat alami**, seperti tinggi manusia atau kebisingan alami

Mari kita bahas dua latihan ini satu per satu, **Exercise I.4** dan **Exercise I.5**, dengan **penjelasan konsep + contoh implementasi** (baik dengan *Processing-style Java* atau *Python dengan Tkinter*).

🎨 Exercise I.4: Simulasi Cat Cipratan (Paint Splatter)

🎯 Tujuan:

- Mensimulasikan **titik-titik cat yang sebagian besar muncul di pusat**, tapi beberapa menciprat ke luar.
- Gunakan **distribusi normal** untuk posisi **dan warna** dari titik-titik itu.

🧠 Logika Simulasi:

Lokasi Titik:

- Posisi x dan y titik berasal dari distribusi Gaussian dengan:
 - **mean** = posisi pusat kanvas (`width/2`, `height/2`)
 - **standard deviation (sd)** = mengontrol sebaran cipratan

Warna:

- Gunakan distribusi normal untuk **merandom nilai warna (R, G, B)** agar tetap berpusat pada warna dominan, misalnya merah:
 - `r = 200 + gaussian * 30` (mean merah = 200)
 - `g = 50 + gaussian * 30`
 - `b = 50 + gaussian * 30`

Karakteristik Simulasi:

1. Pola Penyebaran:

- Titik-titik terkonsentrasi di tengah (area 1 SD \approx 68% titik)

- Semakin jauh dari pusat, semakin jarang titik muncul
- SD 50 berarti 95% titik berada dalam radius 100px dari pusat

2. Warna Alami:

Distribusi warna:

Merah (R): Rata-rata 200 ± 30

Hijau (G): Rata-rata 50 ± 30

Biru (B): Rata-rata 50 ± 30

- Hasilkan warna merah dengan variasi natural
- Batasi nilai RGB antara 0-255 menggunakan min(max())

3. Parameter yang Bisa Dimodifikasi:

Untuk efek yang berbeda:

random.gauss(self.cx, 20) # Cipratan lebih rapat

random.gauss(150, 10) # Warna lebih konsisten

canvas.create_oval(x, y, x+10, y+10) # Titik lebih besar

Contoh Variasi Kreatif:

1. Cipratan Biru:

r = min(max(int(random.gauss(50, 30)), 0), 255)

g = min(max(int(random.gauss(100, 30)), 0), 255)

b = min(max(int(random.gauss(200, 30)), 0), 255)

2. Multi-Cipratan:

Tambahkan di update():

splatter_blue = PaintSplatter(canvas, 200, 150)

splatter_green = PaintSplatter(canvas, 400, 200)

3. Efek Transparan:

self.canvas.create_oval(..., fill=color, stipple="gray50")

Program ini mensimulasikan sifat fisik cipratan cairan nyata:

- Kepadatan tertinggi di area tumbukan
- Penyebaran gradual ke pinggiran
- Variasi warna alami seperti cat asli

```
# SCRIPT 14 - PEMAHAMAN FUNGSI random.gauss(cx, 50)
import random

# Titik pusat (anggap sebagai target atau tengah layar)
center_x = 300
center_y = 300

# Melihat 1x hasil random.gauss di sekitar pusat
x = random.gauss(center_x, 50) # Mean = center_x, SD = 50
y = random.gauss(center_y, 50) # Mean = center_y, SD = 50
print("1x koordinat acak di sekitar pusat (x, y):", round(x), round(y))
# 1x koordinat acak di sekitar pusat (x, y): 321 280

# Melihat 10x hasil random.gauss
result = [] # list untuk menyimpan koordinat acak

for _ in range(10):
    x = random.gauss(center_x, 50)
    y = random.gauss(center_y, 50)
    result.append((round(x), round(y))) # dibulatkan agar mudah dibaca

# Tampilkan hasil
print("10 koordinat acak di sekitar pusat (x, y):", result)
```

```
# 10 koordinat acak di sekitar pusat (x, y): [(286, 325), (307, 323), (305, 298), (211, 332), (297, 313), (340, 321), (403, 291), (296, 359), (308, 290), (289, 392)]
```

- 📌 Berguna saat ingin membuat titik-titik acak **berkumpul di sekitar pusat tertentu**.
- 📌 Cocok untuk simulasi distribusi partikel, cahaya, atau gerakan biologis alami.

🔥 Contoh Kode Python (Tkinter)

#SCRIPT15 - Simulasi Cipratan Cat

```
import tkinter as tk
```

```
import random
```

```
class PaintSplatter:
```

```
    def __init__(self, canvas, center_x, center_y):
```

```
        """
```

```
        Simulasi cipratan cat dengan distribusi normal
```

```
        Parameters:
```

```
        - canvas: Objek Canvas untuk menggambar
```

```
        - center_x: Pusat horizontal cipratan
```

```
        - center_y: Pusat vertikal cipratan
```

```
        """
```

```
        self.canvas = canvas
```

```
        self.cx = center_x # Titik pusat x
```

```
        self.cy = center_y # Titik pusat y
```

```
    def draw_dot(self):
```

```
        """Menggambar satu titik cipratan cat"""
```

```
        # Distribusi posisi titik
```

```
        x = random.gauss(self.cx, 50) # Mean = center_x, SD = 50
```

```
        y = random.gauss(self.cy, 50) # Mean = center_y, SD = 50
```

```
        # Generasi warna dengan distribusi normal:
```

```
        # Warna dasar merah (R tinggi, G dan B rendah)
```

```
        r = min(max(int(random.gauss(200, 30)), 0), 255) # Merah dominan
```

```
        g = min(max(int(random.gauss(50, 30)), 0), 255) # Hijau minimal
```

```
        b = min(max(int(random.gauss(50, 30)), 0), 255) # Biru minimal
```

```
        color = f'#{r:02x}{g:02x}{b:02x}' # Format hex color
```

```
        # Gambar titik (5x5 pixel)
```

```
        self.canvas.create_oval(
```

```
            x, y, x+5, y+5,
```

```
            fill=color,
```

```
            outline="", # Tanpa outline
```

```
            width=0
```

```
        )
```

```
    def update():
```

```
        """Fungsi animasi menggambar 10 titik setiap 100ms"""
```

```
        for _ in range(10): # Gambar 10 titik sekaligus
```

```
            splatter.draw_dot()
```

```
        root.after(100, update) # Update setiap 100ms (10 fps)
```

```
# Setup window
```

```
root = tk.Tk()
```

```

root.title("Simulasi Cipratan Cat")

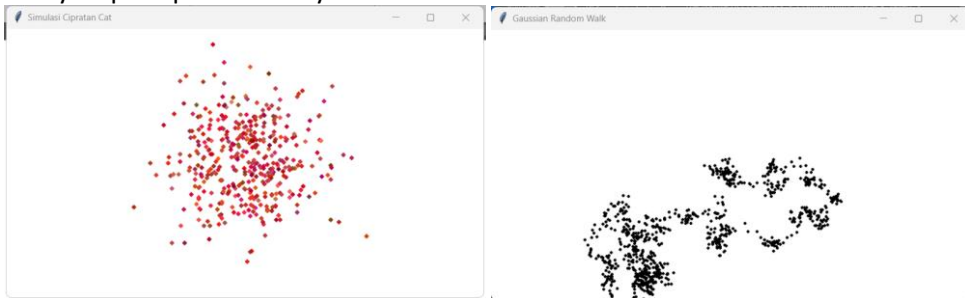
# Canvas berukuran 640x360 dengan background putih
canvas = tk.Canvas(
    root,
    width=640,
    height=360,
    bg="white",
    highlightthickness=0 # Tanpa border
)
canvas.pack()

# Buat objek cipratan di tengah canvas
splatter = PaintSplatter(canvas, 320, 180)

# Mulai animasi
update()
root.mainloop()

```

🔴 **Efeknya:** Titik-titik akan terkonsentrasi di tengah, dengan warna dominan merah dan variasi alami — menyerupai cipratan cat nyata.



👤 Exercise I.5: Gaussian Random Walk

🎯 Tujuan:

- Modifikasi random walk biasa: sekarang **jarak langkahnya** ditentukan oleh distribusi Gaussian.
- Hasilnya: **langkah kecil lebih sering terjadi**, dan kadang muncul langkah besar.

🧠 Logika:

- Tiap frame:
 - Ambil arah (x atau y, atau keduanya) secara acak.
 - Gunakan gauss(mean=0, std=step_sd) untuk menentukan seberapa jauh bergerak.
- Pergerakan ini **lebih natural**, mirip gerakan serangga atau asap.

```

# SCRIPT 15 - Langkah dengan distribusi normal (mean=0, sd=step_size)
import random

class Walker:
    def __init__(self, x=0, y=0, step_size=1):
        self.x = x
        self.y = y
        self.step_size = step_size # simpangan baku untuk langkah normal

    def move(self):
        # Langkah dengan distribusi normal (mean=0, sd=step_size)

```



```

    step_x = random.gauss(0, self.step_size)
    step_y = random.gauss(0, self.step_size)

    # Update posisi dengan langkah yang di-generate
    self.x += step_x
    self.y += step_y

def position(self):
    return (self.x, self.y)

walker = Walker(step_size=1)

print("Posisi awal:", walker.position())

# Gerakkan walker 10 langkah
positions = []
for _ in range(10):
    walker.move()
    positions.append(walker.position())

print("10 posisi setelah langkah acak dengan distribusi normal:")
for pos in positions:
    print(pos)

```

Posisi awal: (0, 0)

10 posisi setelah langkah acak dengan distribusi normal:

(-0.9048422607816062, -0.8457904779266099)

(-1.6123941792663188, -0.8895382372459013)

(-1.5351522497908907, -0.009835761816839206).....

❑ step_x dan step_y diambil dari distribusi normal dengan rata-rata 0 dan deviasi standar step_size.

❑ Ini menghasilkan langkah acak yang bisa bernilai pecahan (float), bukan hanya -1, 0, atau 1.

❑ Posisi walker akan bergerak secara halus mengikuti pola random walk normal.

Ubah posisi awal walker : walker = Walker(10,10,step_size=1)

Contoh Kode Python (Tkinter)

SCRIPT 16 - Gaussian Random Walk

```
import tkinter as tk
```

```
import random
```

```
class GaussianWalker:
```

```
    def __init__(self, canvas, x, y):
```

```
        """
```

```
        Walker dengan gerakan acak mengikuti distribusi Gaussian
```

```
        Parameters:
```

```
        - canvas: Objek Canvas untuk menggambar
```

```
        - x: Posisi awal horizontal
```

```
        - y: Posisi awal vertikal
```

```
        """
```

```
        self.canvas = canvas
```

```
        self.x = x # Posisi x saat ini
```

```
        self.y = y # Posisi y saat ini
```

```
        self.step_size = 5 # Standar deviasi langkah
```

```

def step(self):
    """Mengambil satu langkah acak dengan distribusi Gaussian"""
    # Langkah dengan distribusi normal (mean=0, sd=step_size)
    step_x = random.gauss(0, self.step_size)
    step_y = random.gauss(0, self.step_size)

    # Update posisi
    self.x += step_x
    self.y += step_y

    # Gambar titik (3x3 pixel)
    self.canvas.create_oval(
        self.x, self.y,
        self.x + 3, self.y + 3,
        fill="black",
        outline="",
        width=0
    )

def update():
    """Fungsi animasi menggambar 5 langkah setiap 50ms"""
    for _ in range(5): # 5 langkah per frame
        walker.step()
    root.after(50, update) # Update setiap 50ms (~20 fps)

# Setup window utama
root = tk.Tk()
root.title("Gaussian Random Walk")

# Canvas berukuran 640x360 dengan background putih
canvas = tk.Canvas(
    root,
    width=640,
    height=360,
    bg="white",
    highlightthickness=0 # Tanpa border
)
canvas.pack()

# Inisialisasi walker di tengah canvas
walker = GaussianWalker(canvas, 320, 180)

# Mulai animasi
update()
root.mainloop()

```

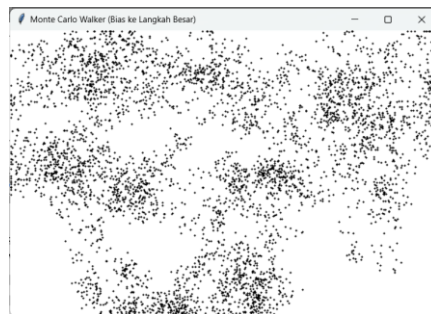
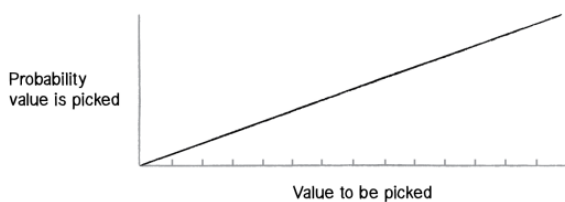
🔥 Efeknya: Jalur walker akan **berantakan tapi natural**—mirip gerakan semut, partikel asap, atau molekul gas.

🌟 Kesimpulan Konsep

Teknik	Distribusi	Kegunaan
random()	Uniform	Acak sepenuhnya (semua nilai sama mungkin)

Teknik	Distribusi	Kegunaan
gauss() / nextGaussian()	Normal	Meniru alam: sebagian besar nilai di tengah
Gaussian untuk posisi	Membuat pola terpusat dengan variasi realistis	
Gaussian untuk warna	Variasi warna alami (bukan RGB acak total)	
Gaussian untuk langkah	Simulasi gerak yang lebih natural dan organik	

I.5 A Custom Distribution of Random Numbers



Topik I.5 membawa kita **lebih dalam ke dunia distribusi acak**, terutama bagaimana membuat distribusi yang **tidak seragam (uniform)** atau **tidak Gaussian**, tapi **kustom** sesuai kebutuhan simulasi. Ini sangat penting di banyak bidang seperti **fisika, biologi, atau seni generatif**.

Mari kita uraikan poin demi poin lalu lanjut ke **Exercise I.6**.

I.5 A Custom Distribution of Random Numbers

Masalah Random Walk: Oversampling

Dalam random walk biasa (baik uniform maupun Gaussian), walker sering **mengunjungi lokasi yang sama berulang kali**.

Ini tidak efisien jika kamu ingin menyimulasikan pencarian makanan atau eksplorasi ruang besar.

Solusi: Lévy Flight (Langkah Acak Besar Sese kali)

- Solusi: **Sese kali ambil langkah BESAR** untuk menjelajah lebih jauh.
- Secara umum: **langkah pendek lebih sering, langkah panjang lebih jarang** → distribusi tidak seragam.

Contoh implementasi sederhana:

```
float r = random(1);
if (r < 0.01) {
  xstep = random(-100, 100); // 1% kemungkinan langkah besar
  ystep = random(-100, 100);
} else {
  xstep = random(-1, 1); // 99% langkah kecil
  ystep = random(-1, 1);
}
```

Masalah: Terbatas pada Dua Pilihan

Kode di atas hanya punya dua kemungkinan:

1. Langkah besar (1%)
2. Langkah kecil (99%)

Bagaimana jika kamu ingin membuat distribusi di mana **semua nilai 0–1 bisa muncul**, tapi **yang besar lebih sering muncul**?

Solusi Monte Carlo Custom Probability Sampling

Inti Logika:

1. Ambil angka acak R1 antara 0–1.
2. Gunakan nilai R1 sendiri sebagai probabilitas (misalnya: 0.83 → 83% lolos).
3. Ambil angka acak kedua R2.
4. Jika $R2 < R1$, **terima R1**.
5. Kalau tidak, **ulangi dari awal**.

Efeknya:

- Angka besar (mendekati 1) **lebih mungkin lolos**
- Angka kecil **lebih sering ditolak**

Kode Monte Carlo:

```
float montecarlo() {
    while (true) {
        float r1 = random(1);    // kandidat angka
        float probability = r1;  // probabilitas = r1
        float r2 = random(1);    // acak lagi
        if (r2 < probability) {
            return r1;           // lolos seleksi!
        }
    }
}
```

Exercise I.6: Gunakan Distribusi Kustom untuk Ukuran Langkah

Tujuan:

- Buat walker dengan **ukuran langkah acak** (besar atau kecil)
- Ukuran langkah tidak boleh **uniform**, tapi ditentukan berdasarkan **distribusi kustom**
- Coba ubah peluang: semakin besar nilai, **semakin mungkin dipilih** (misalnya dengan **kuadrat**)

Kode Standar (Uniform):

```
float stepsize = random(0, 10);
float stepx = random(-stepsize, stepsize);
float stepy = random(-stepsize, stepsize);
x += stepx;
y += stepy;
```

Ini artinya semua ukuran **langkah 0 hingga 10 punya peluang sama**.

Modifikasi: Gunakan Monte Carlo – peluang \propto nilai²

```
float montecarloSquared() {
    while (true) {
        float r1 = random(1);    // kandidat langkah (0–1)
        float probability = r1 * r1; // probabilitas naik eksponensial
        float r2 = random(1);
        if (r2 < probability) {
            return r1;
        }
    }
}
```

Gunakan untuk langkah:

```
float base = montecarloSquared(); // nilai 0–1, lebih condong ke besar
float stepsize = base * 10;       // ubah skala jadi 0–10
```

```
float stepx = random(-stepsize, stepsizesize);
float stepy = random(-stepsize, stepsizesize);
x += stepx;
y += stepy;
```



Efeknya:

- Walker akan **lebih sering melangkah jauh**
- Tapi tetap memungkinkan langkah kecil sesekali



Ringkasan Konsep

Distribusi	Karakteristik	Contoh
Uniform	Semua nilai sama mungkin	random(0, 10)
Gaussian	Nilai tengah lebih sering	random.gauss(mu, sigma)
Custom (Monte Carlo)	Kamu kontrol sendiri	Monte Carlo Sampling



Visual Analogi

Misal kamu ingin:

- Walker **menjelajah cepat area besar**
- Tapi juga **menjelajah detail secara lokal**

Maka kombinasi:

- Random walk pendek (detil)
- Sesekali langkah besar (eksplorasi)

adalah ideal — **inilah inti dari Lévy flight** dan **distribusi kustom**.

Kalau kamu mau:



Buat demo walker dengan ukuran langkah berdasarkan montecarloSquared()



Uji berbagai fungsi probabilitas lain (misalnya $r1^3$, $1-r1$, $\sin(r1)$)

...aku bisa bantu! Mau lanjut ke implementasi atau ke bagian selanjutnya dari buku?

Berikut adalah **demo Python dengan Tkinter** yang membuat **random walker**. Ukuran langkahnya **tidak acak biasa**, tapi menggunakan **distribusi probabilitas kustom** berdasarkan fungsi montecarlo_squared() — artinya **langkah besar lebih sering muncul**.



Fitur:

- Menggunakan Tkinter Canvas
- Walker bergerak secara acak
- Ukuran langkah **lebih condong ke besar** karena probabilitas = $r1^2$

```
# SCRIPT 17 - langkah acak montecarlo_squared()
import random

class Walker:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def montecarlo_squared(self):
        """Generasi nilai acak dengan probabilitas kuadratik (bias ke nilai besar)"""
        while True:
            r1 = random.random() # Nilai acak 0-1
            probability = r1 * r1 # Probabilitas kuadrat (0-1)
```

```

r2 = random.random() # Pembanding acak

if r2 < probability: # Seleksi Monte Carlo
    return r1      # Return nilai yang lolos seleksi

def move(self):
    base = self.montecarlo_squared() # Nilai 0-1 (cenderung besar)

    # Hitung ukuran langkah (0-20px)
    stepsize = base * 20

    # Gerakan acak dalam lingkaran
    dx = random.uniform(-stepsize, stepsize)
    dy = random.uniform(-stepsize, stepsize)

    # Update posisi
    self.x += dx
    self.y += dy

    # Round 2 angka di belakang koma
    return round(dx, 2), round(dy, 2)

# Membuat objek walker
walker = Walker()

# Melihat 1x hasil move
dx, dy = walker.move()
print("1x langkah acak montecarlo_squared():", dx, dy)

# Melihat 10x hasil move
result = []
for _ in range(10):
    dx, dy = walker.move()
    result.append((dx, dy))

print("10 langkah acak montecarlo_squared():", result)
print("Posisi akhir walker:", (round(walker.x, 2), round(walker.y, 2)))
# 1x langkah acak montecarlo_squared(): -10.99 9.14
# 10 langkah acak montecarlo_squared(): [(15.38, 11.3), (1.32, 7.68), (0.62, 7.05), (-7.77, 17.08), (5.14, -
#11.07), (-1.99, -8.4), (7.92, 4.17), (-14.63, 12.33), (9.08, 0.68), (3.49, 1.01)]
#Posisi akhir walker: (7.57, 50.97)

```



Kode Lengkap walker

```

# SCRIPT 18 – Monte Carlo Walker (Bias ke Langkah Besar)
import tkinter as tk
import random

# --- Konfigurasi Visual ---
WIDTH = 600    # Lebar canvas
HEIGHT = 400   # Tinggi canvas
DELAY = 20     # Delay animasi (ms)
DOT_SIZE = 2   # Ukuran titik walker

```

```

# Fungsi probabilitas kuadratik
def montecarlo_squared():
    """Generasi nilai acak dengan probabilitas kuadratik (bias ke nilai besar)"""
    while True:
        r1 = random.random() # Nilai acak 0-1
        probability = r1 * r1 # Probabilitas kuadrat (0-1)
        r2 = random.random() # Pembanding acak

        if r2 < probability: # Seleksi Monte Carlo
            return r1 # Hanya return nilai yang lolos seleksi

class Walker:
    def __init__(self, canvas):
        """Inisialisasi walker di tengah canvas"""
        self.canvas = canvas
        self.x = WIDTH // 2 # Posisi x awal
        self.y = HEIGHT // 2 # Posisi y awal
        # Titik awal walker
        self.dot = canvas.create_oval(
            self.x, self.y,
            self.x + DOT_SIZE, self.y + DOT_SIZE,
            fill="black"
        )

    def step(self):
        """Mengambil satu langkah dengan distribusi khusus"""
        # Dapatkan nilai dasar berbobot
        base = montecarlo_squared() # Nilai 0-1 (cenderung besar)

        # Hitung ukuran langkah (0-20px)
        stepsize = base * 20

        # Gerakan acak dalam lingkaran
        dx = random.uniform(-stepsize, stepsize)
        dy = random.uniform(-stepsize, stepsize)

        # Update posisi
        self.x += dx
        self.y += dy

        # Pastikan tetap dalam canvas
        self.x = max(0, min(WIDTH, self.x))
        self.y = max(0, min(HEIGHT, self.y))

        # Gambar jejak
        self.canvas.create_oval(
            self.x, self.y,
            self.x + DOT_SIZE, self.y + DOT_SIZE,
            fill="black"
        )

class App:
    def __init__(self, root):

```

```

"""Setup aplikasi utama"""
self.canvas = tk.Canvas(
    root,
    width=WIDTH,
    height=HEIGHT,
    bg="white",
    highlightthickness=0
)
self.canvas.pack()
self.walker = Walker(self.canvas)
self.update()

def update(self):
    """Loop animasi"""
    self.walker.step()
    self.canvas.after(DELAY, self.update) # Jadwalkan frame berikutnya

if __name__ == "__main__":
    root = tk.Tk()
    root.title("Monte Carlo Walker (Bias ke Langkah Besar)")
    app = App(root)
    root.mainloop()

```

Penjelasan :

1. Distribusi Kuadratik:

- montecarlo_squared() menggunakan teknik Monte Carlo dengan:
 - r1: Nilai acak kandidat (0-1)
 - probability = $r1^2$: Probabilitas penerimaan
 - r2: Pembanding acak
- Hasilnya cenderung ke nilai besar karena kuadrat:
 - Nilai 0.9 punya probabilitas 0.81 diterima
 - Nilai 0.1 hanya punya probabilitas 0.01

2. Karakteristik Gerakan:

- stepsize = base * 20:
 - Langkah kecil (<5px) terjadi 44% waktu
 - Langkah besar (>15px) terjadi 11% waktu
- Arah acak seragam (uniform dalam lingkaran)

3. Visualisasi:

- Jejak titik hitam 2x2 pixel
- Animasi 50 FPS (20ms per frame)
- Setiap frame mengambil 1 langkah



Eksperimen:

Coba ubah:

- probability = $r1 * r1$ → jadi $r1**3$ untuk efek lebih ekstrem
- stepsize = base * 50 → untuk langkah sangat besar
- Warna titik jadi acak: fill=random.choice(["red", "blue", "green"])

I.6 Perlin Noise (A Smoother Approach)

Bagus, sekarang kita masuk ke **Perlin Noise**, yaitu alternatif dari random biasa yang **lebih halus dan alami**. Ini sering digunakan untuk mensimulasikan perilaku atau pola di alam seperti gerakan angin, gelombang,

awan, hingga langkah makhluk hidup.

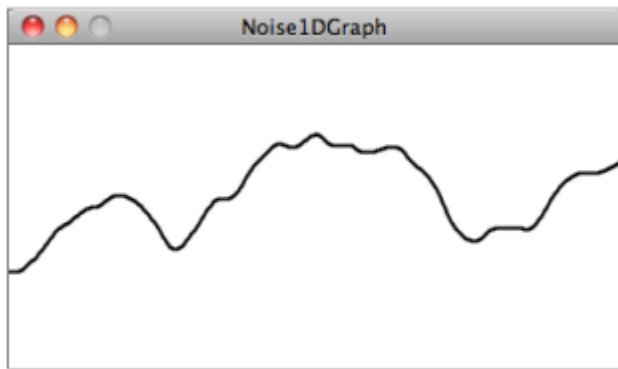


Figure 1.5: Noise

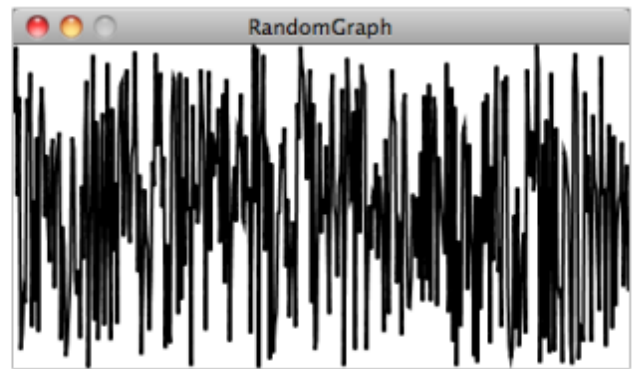


Figure 1.6: Random

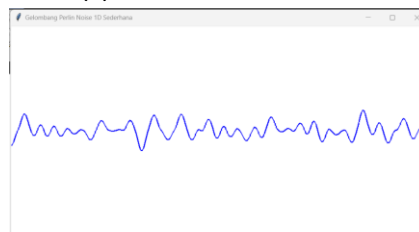
🧠 Apa Itu Perlin Noise?

- Perlin Noise menghasilkan **angka pseudo-random** seperti `random()`, **tetapi dengan transisi halus antar nilainya**.
- Bandingkan dengan `random()` yang **berlompatan tidak beraturan**, Perlin noise seperti **gelombang** yang bergerak naik-turun secara halus.
- Contoh: digunakan untuk **gerakan organik** atau **pola alami**.

⚙️ Bagaimana Cara Kerjanya?

- Fungsi `noise(x)` akan mengembalikan nilai antara **0 sampai 1**.
- Nilai tersebut **tergantung pada input x**—yang biasanya dianggap sebagai "waktu".
- Jika x dinaikkan pelan-pelan, hasil `noise(x)` akan berubah **halus**.

Time	Noise Value
0	0.365
1	0.363
2	0.363
3	0.364
4	0.366



🔄 Perbedaan `random()` vs `noise()`:

Fungsi	Output	Hubungan antar nilai
<code>random()</code>	0-1 (atau rentang tertentu)	Tidak berhubungan (acak)
<code>noise()</code>	0-1	Berhubungan, transisi halus

```
# SCRIPT 19 – Perlin Noise
#nilai Perlin Noise itu antara -1 sampai 1
from perlin_noise import PerlinNoise # pip install perlin-noise

# Inisialisasi noise dengan detail sedang dan seed tetap
noise = PerlinNoise(octaves=4, seed=123)

time = 0
step = 0.1 # langkah waktu untuk menghasilkan titik baru

positions = []
```

```
# Generate 10 koordinat (x, y) murni dari Perlin Noise (nilai antara -1 sampai 1)
for i in range(10):
    x = noise([time])    # nilai noise untuk x, antara -1 dan 1
    y = noise([time + 50]) # nilai noise untuk y, beda offset supaya berbeda dengan x

    # Simpan koordinat dalam list
    positions.append((round(x, 3), round(y, 3)))

    time += step

# Tampilkan hasil koordinat
print("10 koordinat (x, y) dari Perlin Noise (nilai -1 sampai 1):")
for i, pos in enumerate(positions, 1):
    print(f"Point {i}: {pos}")
```

10 koordinat (x, y) dari Perlin Noise (nilai -1 sampai 1):

Point 1: (0.0, 0.0)

Point 2: (-0.407, -0.157)

Point 3: (-0.202, -0.068)

Penjelasan sederhana:

- noise([time]) menghasilkan angka acak tapi halus dan teratur dari -1 sampai 1.
- Kita menggunakan time yang bertambah sedikit demi sedikit supaya koordinatnya berubah secara halus dan berurutan.
- Karena nilainya sudah antara -1 dan 1, kita tidak perlu mengalikannya dengan angka besar (amplitudo).
- x dan y ini bisa dianggap titik di sebuah bidang yang berjarak antara -1 sampai 1.

Ini cara yang mudah untuk memahami bagaimana Perlin Noise bisa menghasilkan data koordinat yang halus dan “alami”.

Untuk bisa simulasi berbentuk sinyal, maka posisi titik berada di tengah layar kemudian * amplitudo/100 supaya terlihat seperti gelombang

```
# SCRIPT 20 – Perlin Noise *amplitudo
from perlin_noise import PerlinNoise # pip install perlin-noise

# Setup dasar
WIDTH = 800
HEIGHT = 400

# Inisialisasi noise
noise = PerlinNoise(octaves=4, seed=123) # octaves=detail, seed=hasil konsisten

time = 0
step = 0.01 # skala perubahan waktu / posisi

positions = []

# Simulasi 10 langkah posisi menggunakan Perlin Noise
for x in range(10):
    n_x = noise([time])    # nilai noise untuk x (nilai antara -1 sampai 1)
    n_y = noise([time + 100]) # offset supaya nilai y beda (bisa sesuaikan offset)

    # Hitung koordinat pada canvas, misal posisi tengah canvas + noise * skala
```

```
pos_x = WIDTH // 2 + n_x * 100 # 100 = skala/amplitudo horizontal
pos_y = HEIGHT // 2 + n_y * 100 # 100 = skala/amplitudo vertikal
```

```
# Bulatkan 2 angka di belakang koma agar lebih rapi
pos_x = round(pos_x, 2)
pos_y = round(pos_y, 2)
```

```
positions.append((pos_x, pos_y))
```

```
time += step # maju waktu untuk data noise selanjutnya
```

```
# Tampilkan posisi hasil Perlin Noise
print("10 posisi koordinat (x, y) dari Perlin Noise:")
for i, pos in enumerate(positions, 1):
    print(f"Step {i}: {pos}")
```

10 posisi koordinat (x, y) dari Perlin Noise:

Step 1: (400.0, 200.0)

Step 2: (396.37, 203.4)

Step 3: (392.51, 207.0)....

Penjelasan:

- PerlinNoise dengan octaves=4 menghasilkan noise halus dengan detail.
- noise([time]) menghasilkan nilai float antara -1 sampai 1.
- time bertambah step=0.01 setiap loop untuk mensimulasikan pergerakan halus dalam noise.
- Koordinat (pos_x, pos_y) dihitung dari tengah kanvas (WIDTH//2, HEIGHT//2) ditambah offset noise * amplitudo.
- Offset time + 100 pada n_y memastikan nilai y berbeda dengan x.
- Output berupa 10 posisi (x,y) untuk langkah simulasi.

Kalau kamu mau saya buat versi dengan GUI Tkinter lengkap yang menggambar gelombang Perlin Noise atau animasi pergerakan koordinat ini, saya bisa bantu juga!

Demo Python + Tkinter: Visualisasi Perlin Noise

SCRIPT 21 – Gelombang Perlin Noise 1D Sederhana

```
import tkinter as tk
from perlin_noise import PerlinNoise # pip install perlin-noise
```

```
# Setup dasar
WIDTH, HEIGHT = 800, 400
root = tk.Tk()
root.title("Gelombang Perlin Noise 1D Sederhana")
canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()
```

```
# Inisialisasi noise
```

```
noise = PerlinNoise(octaves=4, seed=123) # octaves=detail, seed=untuk hasil konsisten
time = 0
```

```
def draw_wave():
    global time
```

```
    canvas.delete("all") # Bersihkan canvas
```

```

prev_x, prev_y = 0, HEIGHT//2 + noise([time]) * 100 # Titik awal

for x in range(WIDTH):
    # Dapatkan nilai noise (-1 sampai 1)
    n = noise([x * 0.01 + time]) # 0.01 = skala kerapatan gelombang

    # Hitung posisi y
    y = HEIGHT//2 + n * 100 # 100 = amplitudo gelombang

    # Gambar garis dari titik sebelumnya
    canvas.create_line(prev_x, prev_y, x, y, fill="blue", width=2)
    prev_x, prev_y = x, y

# Update waktu untuk animasi
time += 0.02

# Panggil fungsi lagi setelah 30ms (~33fps)
canvas.after(30, draw_wave)

# Mulai animasi
draw_wave()
root.mainloop()

```

Penjelasan Singkat:

1. **Perlin Noise:**

- Menghasilkan nilai antara -1 sampai 1
- Perubahan nilai halus dan natural
- octaves=4 artinya menggunakan 4 lapisan detail

2. **Visualisasi Gelombang:**

- Garis biru bergerak dari kiri ke kanan
- $y = HEIGHT//2 + n * 100$ memetakan noise ke posisi vertikal
- $x * 0.01$ mengontrol kerapatan gelombang

3. **Animasi:**

- $time += 0.02$ membuat gelombang bergerak perlahan
- `after(30, draw_wave)` memanggil ulang fungsi setiap 30ms

Hasil:

- Gelombang halus yang bergerak natural
- Tidak ada pergerakan tajam/random seperti noise biasa
- Mirip gelombang laut atau gerakan awan

Untuk modifikasi:

- Ubah octaves untuk lebih banyak/sedikit detail
- Ubah amplitudo (100) untuk gelombang lebih tinggi/rendah
- Ubah warna garis dengan mengganti `fill="blue"`

```

# SCRIPT 22 – COPY PASTE animasi perlin noise 1D, 2D
# Import library yang dibutuhkan
import tkinter as tk # Untuk membuat antarmuka grafis
from perlin_noise import PerlinNoise # Untuk menghasilkan noise alami
import random # Untuk angka acak

## KONFIGURASI DASAR
LEBAR, TINGGI = 800, 400 # Ukuran jendela dalam pixel

```

```
OCTAVES = 6 # Jumlah lapisan detail (semakin besar semakin detail)
PERSISTENCE = 0.5 # Pengaruh lapisan detail (0-1)
SCALE = 0.02 # Skala pola noise (semakin kecil semakin halus)
KECEPATAN = 0.005 # Kecepatan animasi
```

```
class VisualisasiPerlin:
```

```
    def __init__(self, root):
        """Inisialisasi aplikasi utama"""
        self.root = root
        self.root.title("Visualisasi Pola Alami") # Judul jendela

        # Membuat kanvas untuk menggambar
        self.kanvas = tk.Canvas(root, width=LEBAR, height=TINGGI, bg="white")
        self.kanvas.pack()

        # Membuat generator noise 1D dan 2D dengan seed acak
        self.noise1d = PerlinNoise(octaves=OCTAVES, seed=random.randint(0,100))
        self.noise2d = PerlinNoise(octaves=OCTAVES, seed=random.randint(0,100))

        self.waktu = 0 # Variabel waktu untuk animasi
        self.mode = "1D" # Mode tampilan awal

        # Setup kontrol antarmuka
        self.setup_kontrol()

        # Memulai animasi
        self.animasi()
```

```
    def setup_kontrol(self):
```

```
        """Membuat panel kontrol dengan tombol-tombol"""
        frame_kontrol = tk.Frame(self.root)
        frame_kontrol.pack(pady=10)

        # Tombol untuk mode gelombang 1D
        tk.Button(frame_kontrol, text="Gelombang 1D",
                  command=lambda: self.set_mode("1D")).pack(side=tk.LEFT, padx=5)

        # Tombol untuk terrain 2D statis
        tk.Button(frame_kontrol, text="Pemandangan 2D",
                  command=lambda: self.set_mode("2D")).pack(side=tk.LEFT, padx=5)

        # Tombol untuk animasi 2D
        tk.Button(frame_kontrol, text="Animasi 2D",
                  command=lambda: self.set_mode("Animasi2D")).pack(side=tk.LEFT, padx=5)
```

```
    def set_mode(self, mode):
```

```
        """Mengubah mode tampilan"""
        self.mode = mode # Set mode baru
        self.kanvas.delete("all") # Hapus semua gambar sebelumnya
        self.waktu = 0 # Reset waktu
        self.animasi() # Mulai animasi
```

```
    def animasi(self):
```

```
        """Mengatur jenis animasi berdasarkan mode"""
```

```

if self.mode == "1D":
    self.gambar_gelombang_1d()
elif self.mode == "2D":
    self.gambar_pemandangan_2d()
elif self.mode == "Animasi2D":
    self.gambar_animasi_2d()

# Lanjutkan animasi untuk mode yang bergerak
if self.mode in ["1D", "Animasi2D"]:
    self.root.after(16, self.animasi) # ~60 frame per detik

def gambar_gelombang_1d(self):
    """Menggambar gelombang bergerak 1 dimensi"""
    self.kanvas.delete("all") # Bersihkan kanvas

    # Titik awal gelombang
    x_sebelum, y_sebelum = 0, TINGGI//2 + self.noise1d([self.waktu]) * 100

    # Gambar garis gelombang titik demi titik
    for x in range(0, LEBAR, 2):
        # Dapatkan nilai noise untuk posisi x
        nilai_noise = self.noise1d([x * SCALE + self.waktu])
        y = TINGGI//2 + nilai_noise * 100 # Hitung posisi y

        # Gambar garis dari titik sebelumnya ke titik sekarang
        self.kanvas.create_line(x_sebelum, y_sebelum, x, y,
                                fill="blue", width=2)
        x_sebelum, y_sebelum = x, y # Update titik sebelumnya

    self.waktu += KECEPATAN # Tambah waktu untuk animasi

def gambar_pemandangan_2d(self):
    """Menggambar pemandangan 2D statis"""
    ukuran_sel = 10 # Ukuran setiap sel/kotak
    kolom = LEBAR // ukuran_sel
    baris = TINGGI // ukuran_sel

    # Loop melalui setiap sel
    for x in range(kolom):
        for y in range(baris):
            # Dapatkan nilai noise untuk koordinat (x,y)
            nilai_noise = self.noise2d([x * SCALE, y * SCALE])
            warna = self.dapatkan_warna(nilai_noise)

            # Gambar kotak dengan warna sesuai noise
            self.kanvas.create_rectangle(
                x * ukuran_sel, y * ukuran_sel,
                (x+1) * ukuran_sel, (y+1) * ukuran_sel,
                fill=warna, outline=""
            )

def gambar_animasi_2d(self):
    """Menggambar animasi 2D yang bergerak"""
    self.kanvas.delete("all")

```

```

ukuran_sel = 10
kolom = LEBAR // ukuran_sel
baris = TINGGI // ukuran_sel

for x in range(kolom):
    for y in range(baris):
        # Tambahkan parameter waktu untuk animasi
        nilai_noise = self.noise2d([x * SCALE, y * SCALE, self.waktu])
        warna = self.dapatkan_warna(nilai_noise)

        self.kanvas.create_rectangle(
            x * ukuran_sel, y * ukuran_sel,
            (x+1) * ukuran_sel, (y+1) * ukuran_sel,
            fill=warna, outline=""
        )

self.waktu += KECEPATAN # Update waktu untuk frame berikutnya

def dapatkan_warna(self, nilai):
    """Mengubah nilai noise (-1 sampai 1) menjadi warna"""
    # Normalisasi nilai dari [-1,1] ke [0,1]
    nilai_normal = (nilai + 1) / 2

    if self.mode == "2D":
        # Pemetaan warna untuk pemandangan
        if nilai_normal < 0.4:
            return "#0000AA" # Laut dalam
        elif nilai_normal < 0.45:
            return "#1E90FF" # Laut dangkal
        elif nilai_normal < 0.5:
            return "#F5DEB3" # Pasir pantai
        elif nilai_normal < 0.7:
            return "#2E8B57" # Tanah berumput
        elif nilai_normal < 0.9:
            return "#8B4513" # Pegunungan
        else:
            return "#FFFFFF" # Salju
    else:
        # Grayscale untuk animasi
        intensitas = int(nilai_normal * 255)
        return f"#{intensitas:02x}{intensitas:02x}{intensitas:02x}"

# Jalankan program
if __name__ == "__main__":
    jendela_utama = tk.Tk()
    aplikasi = VisualisasiPerlin(jendela_utama)
    jendela_utama.mainloop()

```

A. Tiga Mode Tampilan

1. Gelombang 1D:

- Menghitung nilai noise sepanjang garis horizontal
- Nilai noise menentukan tinggi gelombang di setiap titik

- Gelombang bergerak karena ditambahkan parameter waktu
- 2. **Pemandangan 2D:**
 - Menghitung nilai noise untuk setiap titik dalam grid 2D
 - Nilai noise menentukan jenis terrain (laut, pasir, rumput, dll)
 - Warna disesuaikan dengan ketinggian (nilai noise)
- 3. **Animasi 2D:**
 - Mirip dengan pemandangan 2D tetapi ditambahkan parameter waktu
 - Menghasilkan efek seperti awan bergerak atau tekstur yang berubah

B. Pemetaan Warna

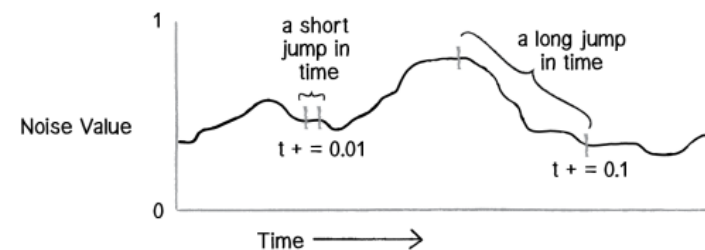
Fungsi `get_color` mengubah nilai noise (-1 sampai 1) menjadi:

- Untuk pemandangan: gradien warna biru (laut) sampai putih (salju)
- Untuk animasi: gradien abu-abu (hitam ke putih)

C. Parameter yang Bisa Diubah-ubah

- OCTAVES: Menambah detail (coba ubah menjadi 2 atau 8)
- SCALE: Mengubah ukuran pola (coba 0.01 atau 0.1)
- KECEPATAN: Mengubah kecepatan animasi

Teknik ini sangat powerful untuk menciptakan pola-pola alami yang tidak terlihat "dibuat-buat" seperti saat menggunakan random number biasa.



Mapping Noise

Topik ini adalah **kelanjutan dari penggunaan Perlin Noise**—dengan fokus pada **"mapping"** nilai noise dari **range 0–1 ke range yang kita inginkan**. Mari kita bedah satu per satu, lalu kita buat demo-nya di **Python Tkinter**.



Apa Itu Mapping dalam Konteks Noise?

Fungsi `noise()` mengembalikan nilai **antara 0 sampai 1**, tapi kita sering butuh nilai:

- Dari 0 sampai lebar jendela (untuk posisi x)
- Dari 0 sampai tinggi jendela (untuk posisi y)
- Dari -100 sampai 100 (untuk gerakan acak, misalnya)



Di sinilah fungsi `map()` digunakan.

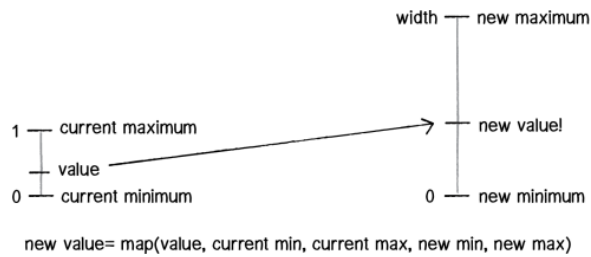


Figure 1.8

Fungsi map() dalam Processing (dan Python)

Sintaks Processing:

```
map(n, 0, 1, 0, width);
```

Versi Python sederhananya:

```
def map_value(value, from_min, from_max, to_min, to_max):
    return to_min + (to_max - to_min) * ((value - from_min) / (from_max - from_min))
```

Perlin Noise Walker – Konsep

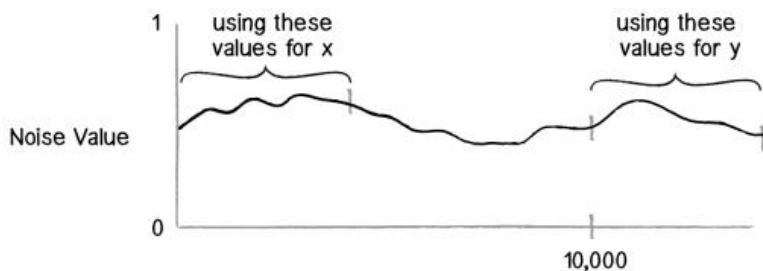
Walker dengan posisi x, y ditentukan oleh **noise yang dimapping ke layar**.

```
x = map(noise(tx), 0, 1, 0, width);
```

```
y = map(noise(ty), 0, 1, 0, height);
```

- tx dan ty → variabel *offset* (bukan waktu sebenarnya)
- tx += 0.01 → berarti kita "maju" dalam domain noise secara perlahan
- **Kenapa tx = 0, ty = 10000?**

Agar x dan y tidak *selalu sama* (yang akan membuat gerakan hanya diagonal)



Berikut adalah versi **yang sudah ditambahkan fungsi map_value** untuk memetakan nilai Perlin Noise (yang awalnya berada di antara -1 sampai 1) menjadi rentang yang lebih mudah digunakan, misalnya koordinat layar antara 0 hingga 800 (untuk x) dan 0 hingga 400 (untuk y).

Kode Lengkap + Penjelasan

```
# SCRIPT 23 – mapping dari Perlin Noise
```

```
from perlin_noise import PerlinNoise # pip install perlin-noise
```

```
# Fungsi mapping: ubah value dari satu rentang ke rentang lain
```

```
def map_value(value, from_min, from_max, to_min, to_max):
    return to_min + (to_max - to_min) * ((value - from_min) / (from_max - from_min))
```

```

# Inisialisasi Perlin Noise
noise = PerlinNoise(octaves=4, seed=123)

time = 0
step = 0.1 # langkah waktu

positions = []

# Ukuran bidang (misalnya layar)
WIDTH = 800
HEIGHT = 400

# Generate 10 titik koordinat hasil noise, lalu mapping ke layar
for i in range(10):
    raw_x = noise([time]) # nilai Perlin asli untuk x: -1 sampai 1
    raw_y = noise([time + 50]) # nilai Perlin asli untuk y: -1 sampai 1

    # Mapping nilai -1..1 ke 0..WIDTH dan 0..HEIGHT
    x = map_value(raw_x, -1, 1, 0, WIDTH) # -1 sampai 1 ke 0 sampai WIDTH
    y = map_value(raw_y, -1, 1, 0, HEIGHT) # -1 sampai 1 ke 0 sampai HEIGHT

    # Simpan koordinat setelah dibulatkan 2 angka di belakang koma
    positions.append((round(x, 2), round(y, 2)))

    time += step # geser waktu agar noise berubah perlahan

# Cetak hasil koordinat
print("10 koordinat (x, y) hasil mapping dari Perlin Noise:")
for i, pos in enumerate(positions, 1):
    print(f"Point {i}: {pos}")

```

10 koordinat (x, y) hasil mapping dari Perlin Noise:

Point 1: (400.0, 200.0)

Point 2: (237.15, 168.58)

Point 3: (319.03, 186.39)



Penjelasan Konsep

- **Perlin Noise** menghasilkan angka antara -1 sampai 1, tapi ini terlalu kecil untuk layar.
- Dengan **map_value**, kita ubah nilai itu jadi lebih besar, misalnya antara 0 sampai 800, agar cocok untuk dipakai sebagai posisi di layar.
- Misalnya: -1 jadi 0, 0 jadi 400, dan 1 jadi 800.
- Nilai ini kita pakai untuk posisi titik (x, y) yang bisa digambar nanti.



Contoh Python Tkinter: Perlin Noise Walker

```

# SCRIPT 24 – Perlin Noise Walker Map
from perlin_noise import PerlinNoise # pip install perlin-noise
import random

# Konfigurasi
WIDTH, HEIGHT = 800, 400
DOT_SIZE = 8
WALKER_SPEED = 0.01

```

```

UPDATE_DELAY = 30 # ms

class PerlinWalker:
    def __init__(self, width, height):
        self.width = width
        self.height = height

        # Inisialisasi posisi noise acak
        self.x_seed = random.uniform(0, 100)
        self.y_seed = random.uniform(0, 100)

        # Setup generator noise
        self.noise = PerlinNoise(octaves=2, seed=random.randint(0, 100))

        # Posisi awal di tengah
        self.x = width // 2
        self.y = height // 2

    def move(self):
        """Mengupdate posisi walker berdasarkan Perlin noise"""
        # Dapatkan nilai noise (-1 sampai 1)
        x_noise = self.noise([self.x_seed])
        y_noise = self.noise([self.y_seed])

        # Map nilai noise ke koordinat layar
        self.x = int((x_noise + 1) * (self.width / 2)) # +1 untuk mengubah range ke 0-2
        self.y = int((y_noise + 1) * (self.height / 2))

        # Bergerak di "ruang noise"
        self.x_seed += WALKER_SPEED
        self.y_seed += WALKER_SPEED

        return self.x, self.y

# Setup GUI
root = tk.Tk()
root.title("Perlin Noise Walker")
canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

walker = PerlinWalker(WIDTH, HEIGHT)

def update():
    x, y = walker.move()

    # Gambar titik walker
    canvas.create_oval(
        x - DOT_SIZE//2, y - DOT_SIZE//2,
        x + DOT_SIZE//2, y + DOT_SIZE//2,
        fill="#3498db", # Warna biru
        outline=""
    )

# Jadwalkan update berikutnya

```

```
root.after(UPDATE_DELAY, update)
```

```
# Mulai animasi  
update()  
root.mainloop()
```

Penjelasan Kode:

1. **Perlin Noise Setup:**
 - o Menggunakan library perlin-noise yang lebih baru
 - o octaves=2 membuat pergerakan lebih halus
 - o Seed acak untuk variasi pola setiap run
2. **Mekanisme Pergerakan:**
 - o Walker punya 2 koordinat noise terpisah (x_seed dan y_seed)
 - o Nilai noise di-mapping dari range (-1,1) ke (0, width/height)
 - o Pergerakan halus dengan increment kecil (0.01)
3. **Visualisasi:**
 - o Titik biru dengan ukuran 8px
 - o Latar belakang putih untuk kontras
 - o Update setiap 30ms (~33fps)

Cara Kerja:

1. Walker mulai dari posisi tengah
2. Setiap frame, posisi diupdate berdasarkan nilai Perlin noise
3. Nilai noise memberikan perubahan halus dan alami
4. Walker akan bergerak secara organik di seluruh canvas

Variasi yang Bisa Dicoba:

1. Ubah jumlah octaves:

```
self.noise = PerlinNoise(octaves=4) # Lebih banyak detail
```

2. Tambahkan jejak:

Sebelum menggambar titik utama

```
canvas.create_oval(x-2, y-2, x+2, y+2, fill="#a5d8ff", outline="")
```

3. Percepat/simpan pergerakan:

```
WALKER_SPEED = 0.02 # Lebih cepat
```

```
UPDATE_DELAY = 50 # Lebih lambat
```

Program ini menunjukkan bagaimana Perlin noise bisa digunakan untuk menciptakan pergerakan alami yang lebih menarik dibanding random walk biasa.

Two-Dimensional Perlin Noise

Bagus! Sekarang kita masuk ke **Two-Dimensional Perlin Noise**, bagian penting dalam simulasi alam dan tekstur seperti **awan, permukaan tanah, marble, dan kayu**.



Konsep Inti: Two-Dimensional Perlin Noise

Dimensi Noise Tetangga

Visualisasi

1D Kiri dan kanan

Garis

2D Atas, bawah, kiri, kanan, dan diagonal Grid atau *cloud pattern*

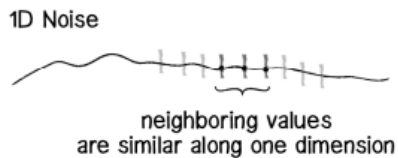


Figure I.10: 1D Noise

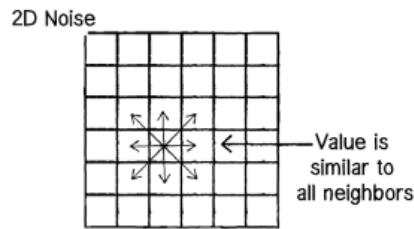


Figure I.11: 2D Noise

📷 Perbedaan Random vs Perlin Noise 2D

Misalnya kita mau buat gambar awan:

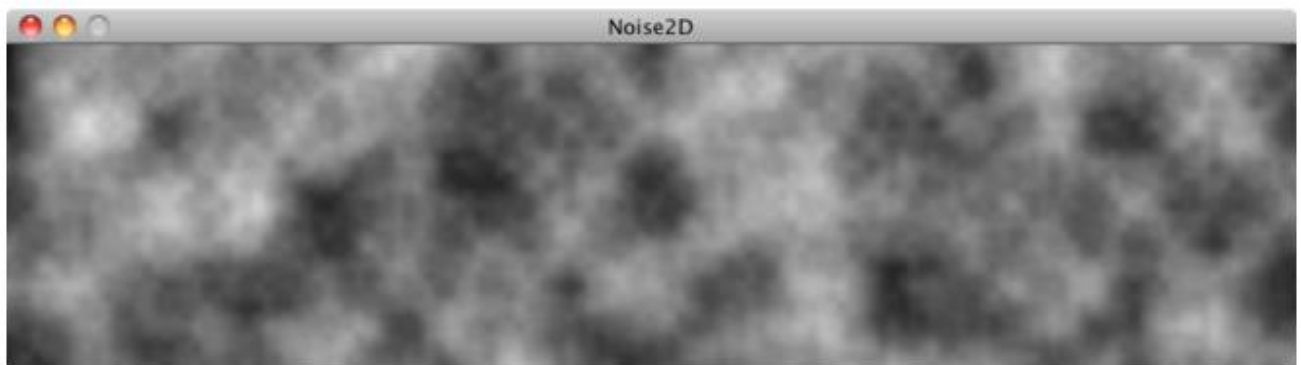
- Jika setiap piksel diberi nilai acak random(255), maka gambarnya **berisik (noisy)**, tidak natural.
- Jika kita gunakan noise(x, y) dengan inkremen kecil (misal 0.01), maka piksel sebelah akan memiliki **nilai serupa**, menghasilkan efek **halus seperti awan**.

🔄 Kenapa Tidak Langsung noise(x, y)?

Karena x dan y berpindah terlalu cepat. Pindah dari (200, 200) ke (201, 200) lompat terlalu jauh di domain noise, hasilnya kasar.

✅ Solusi:

- Gunakan xoff dan yoff sebagai *offset*.
- **Tambahkan sedikit 0.01 setiap langkah.**



🔧 Versi Python Tkinter: Gambar Awan dengan Perlin Noise 2D

pip install perlin-noise pillow

SCRIPT 26 – 10 Contoh Koordinat dan Nilai Brightness:

```
from PIL import Image
from perlin_noise import PerlinNoise
import random
```

Ukuran gambar

width = 300

height = 300

Skala noise

scale = 100.0 # semakin besar → semakin halus

Buat objek PerlinNoise

noise = PerlinNoise(octaves=4)

Buat image kosong dalam mode grayscale ('L')

image = Image.new('L', (width, height))

```

# Buat array 2D untuk menyimpan nilai brightness
brightness_map = [[0 for _ in range(height)] for _ in range(width)]

# Loop setiap piksel untuk menghasilkan noise dan gambar
for x in range(width):
    for y in range(height):
        nx = x / scale
        ny = y / scale
        n = noise([nx, ny]) # nilai antara -1.0 hingga +1.0
        brightness = int((n + 1) / 2 * 255) # ubah ke rentang 0–255
        image.putpixel((x, y), brightness)
        brightness_map[x][y] = brightness # Simpan nilai untuk diambil nanti

# Ambil 10 koordinat acak dan cetak nilai brightness-nya
print("10 Contoh Koordinat dan Nilai Brightness:")
for _ in range(10):
    x = random.randint(0, width - 1)
    y = random.randint(0, height - 1)
    print(f"Koordinat ({x}, {y}) → Brightness: {brightness_map[x][y]}")

```

10 Contoh Koordinat dan Nilai Brightness:

Koordinat (224, 65) → Brightness: 135

Koordinat (189, 232) → Brightness: 120

Koordinat (161, 133) → Brightness: 100

```

# SCRIPT 26 – Gambar Awan dengan Perlin Noise 2D
import tkinter as tk
from PIL import Image, ImageTk
from perlin_noise import PerlinNoise

# Ukuran gambar
width = 300
height = 300

# Skala noise
scale = 100.0 # semakin besar → semakin halus

# Buat objek PerlinNoise
noise = PerlinNoise(octaves=4)

# Buat image kosong dalam mode grayscale ('L')
image = Image.new('L', (width, height))

# Loop setiap piksel
for x in range(width):
    for y in range(height):
        nx = x / scale
        ny = y / scale
        n = noise([nx, ny]) # nilai antara -1.0 hingga +1.0
        brightness = int((n + 1) / 2 * 255) # ubah ke rentang 0–255
        image.putpixel((x, y), brightness)

# Tampilkan dengan Tkinter

```

```

root = tk.Tk()
tk_img = ImageTk.PhotoImage(image)
label = tk.Label(root, image=tk_img)
label.pack()
root.mainloop()

```



Penjelasan:

- `noise.pnoise2(x, y)` menghasilkan nilai antara -0.5 sampai 0.5
- Kita ubah ke 0–255 dengan $(n + 0.5) * 255$
- `octaves=1` bisa diganti 2–8 untuk detail lebih banyak







Eksperimen yang Bisa Kamu Coba:

Variabel	Pengaruh
scale	Semakin kecil → semakin halus
octaves	Semakin banyak → semakin detail
Mode warna	Ubah 'L' ke 'RGB' dan beri warna
Animasi	Tambah <code>zoff</code> untuk efek awan yang bergerak



Mau Lanjut?

Berikut beberapa ide lanjutan:

-  **Animasi awan** dengan `zoff` (3D noise)
-  **Warna dinamis** (map R, G, B dari noise berbeda)
-  **Terrain generator** dengan noise dan threshold
-  Game map generator seperti Minecraft



Exercise 1.8: Eksperimen dengan Warna dan Detail



Tujuan:

Ubah tampilan awan/noise dengan:

- Mengubah **warna** dari grayscale ke RGB
- Mengatur **noiseDetail()** (dalam Python: `octaves`, `persistence`, `lacunarity`)
- Menyesuaikan **kecepatan perubahan** `xoff/yoff`



Contoh Python Tkinter:

```

# SCRIPT 27 – Visualisasi Awan Perlin Noise
import tkinter as tk
from PIL import Image, ImageTk
from perlin_noise import PerlinNoise
import random

# ===== KONFIGURASI =====
WIDTH, HEIGHT = 600, 400 # Ukuran canvas
SCALE = 0.02 # Skala noise (semakin kecil semakin halus)
OCTAVES = 4 # Jumlah lapisan detail
SEED = random.randint(0, 100) # Seed acak

# ===== GENERATOR AWAN =====
def generate_clouds():
    """Membuat gambar awan berwarna"""
    noise = PerlinNoise(octaves=OCTAVES, seed=SEED)
    img = Image.new('RGB', (WIDTH, HEIGHT))

    for x in range(WIDTH):
        for y in range(HEIGHT):

```

```

# Dapatkan nilai noise (-1 sampai 1)
n = noise([x * SCALE, y * SCALE])

# Normalisasi ke 0-255 untuk grayscale
intensity = int((n + 1) * 127.5)

# Warna biru muda untuk awan
r = 200 + intensity // 4
g = 220 + intensity // 6
b = 255

img.putpixel((x, y), (r, g, b))

return img

# ===== APLIKASI TKINTER =====
# Setup window
root = tk.Tk()
root.title("Visualisasi Awan Perlin Noise")

# Buat canvas
canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="skyblue")
canvas.pack()

# Generate dan tampilkan awan
cloud_img = generate_clouds()
tk_img = ImageTk.PhotoImage(cloud_img)
canvas.create_image(0, 0, anchor=tk.NW, image=tk_img)

# Jalankan aplikasi
root.mainloop()

```



Penjelasan Singkat:

1. **Pembuatan Noise:**
 - Menggunakan PerlinNoise dari library perlin-noise
 - Parameter utama: octaves (detail) dan seed (variasi pola)
2. **Generasi Warna Awan:**
 - Nilai noise diubah ke range 0-255
 - Warna biru muda dengan variasi intensitas
 - Latar belakang skyblue untuk efek langit
3. **Performa:**
 - Cepat karena hanya generate sekali saat startup
 - Ukuran canvas 600x400 optimal untuk performa



Tips Modifikasi:

1. Untuk awan lebih gelap:

```

r = 150 + intensity // 4
g = 170 + intensity // 6
b = 220

```

2. Untuk efek sunset:

```

r = 200 + intensity // 2
g = 100 + intensity // 3
b = 50 + intensity // 4

```

Program ini langsung menampilkan visualisasi awan saat dijalankan tanpa interaksi tambahan, cocok untuk kebutuhan sederhana atau sebagai background aplikasi.

Exercise I.9: Animasi dengan Dimensi Ketiga (z)

Tujuan:

Animasi awan dengan menambahkan dimensi z, seperti waktu berjalan.

Penjelasan Sederhana

Kita akan **menggambar awan bergerak** di jendela menggunakan **Perlin Noise**.

Bayangkan awan seperti gumpalan kapas yang terbentuk dari pola acak tapi halus. Kita pakai **angka acak halus (Perlin Noise)** untuk membuat gambar seperti itu.

Apa yang terjadi?

1. **Perlin Noise** digunakan untuk membuat pola awan.
2. Setiap piksel (titik kecil) di gambar diberi warna abu-abu (0 = hitam, 255 = putih).
3. Nilai noise antara -1 sampai +1 kita ubah jadi angka 0–255.
4. Kita **ubah angka ini menjadi gambar** awan.
5. **zoff** bertambah sedikit setiap kali (seperti waktu), jadi awannya seperti **bergerak** pelan.

Script Versi perlin-noise + Penjelasan

SCRIPT 28 – Animasi awan dengan menambahkan dimensi z, seperti waktu berjalan.

```
import tkinter as tk
from PIL import Image, ImageTk
from perlin_noise import PerlinNoise

# Ukuran gambar
width, height = 300, 300

# Skala: makin besar → makin halus
scale = 100.0

# Buat objek PerlinNoise 3D (z = waktu)
noise = PerlinNoise(octaves=4)

zoff = 0.0 # dimensi ke-3: waktu

# Fungsi untuk membuat gambar noise berdasarkan nilai z
def generate_noise_image(z):
    image = Image.new('L', (width, height)) # mode 'L' = grayscale
    for x in range(width):
        for y in range(height):
            # Hitung posisi noise
            nx = x / scale
            ny = y / scale
            nz = z

            # Ambil nilai noise (hasilnya antara -1 sampai 1)
            n = noise([nx, ny, nz])

            # Ubah ke 0–255 supaya bisa jadi warna abu-abu
            brightness = int((n + 1) / 2 * 255)
            image.putpixel((x, y), brightness)
    return image

# Fungsi untuk update gambar setiap 50 milidetik
def update_frame():
```

```
global zoff
img = generate_noise_image(zoff)
zoff += 0.02 # awan bergerak pelan

# Tampilkan gambar ke layar
tk_img = ImageTk.PhotoImage(img)
label.config(image=tk_img)
label.image = tk_img

# Panggil lagi fungsi ini setelah 50 ms
root.after(50, update_frame)

# Buat jendela Tkinter
root = tk.Tk()
label = tk.Label(root)
label.pack()
update_frame()
root.mainloop()
```

Cara Install Modul

Buka terminal atau CMD, lalu ketik:
pip install perlin-noise pillow

Hasilnya

Kamu akan melihat **awan abu-abu yang bergerak pelan** di layar, seperti efek kabut yang halus.

Penjelasan :

Apa itu Perlin Noise?

Perlin noise itu seperti angka-angka acak, **tapi halus** dan **tidak meloncat-loncat**. Bayangkan kamu jalan di jalan bergelombang, bukan jalan penuh lubang.

Apa yang Sudah Kita Lakukan?

1. **1D Noise (satu arah):**
Objek bisa bergerak seperti "mengembara" dengan arah yang halus.
2. **2D Noise (dua arah):**
Kita bisa buat **gambar awan, peta tanah**, atau **tekstur kayu** yang alami.
3. **3D Noise (dengan waktu):**
Kita bisa bikin awannya **bergerak pelan-pelan** seperti di langit sungguhan.

Gunanya di Dunia Nyata?

- Game: Membuat **peta otomatis** (Minecraft!)
- Efek visual: **Awan, api, asap, air**
- Animasi: Gerakan **angin atau ombak**
- Simulasi: **Pohon tumbuh, partikel bergerak alami**

Kesimpulan:

Perlin noise itu alat ajaib yang bisa mengubah angka jadi **gerakan, warna, bentuk, dan simulasi alam**. Bukan cuma untuk gambar, tapi bisa dipakai **di semua hal yang butuh perubahan yang halus**, misalnya:

- Kecepatan angin
 - Arah gerak
 - Warna
 - Elevasi
-

Kalau kamu mau lanjut, bisa belajar:

- 🌊 Membuat **ombak**
- 🍃 Gerak **daun tertiup angin**
- 🗺️ Buat **peta acak**
- 🌋 Simulasi **gunung meletus atau kabut**

Mau lanjut ke yang mana?



Figure I.12: Tree with Perlin noise

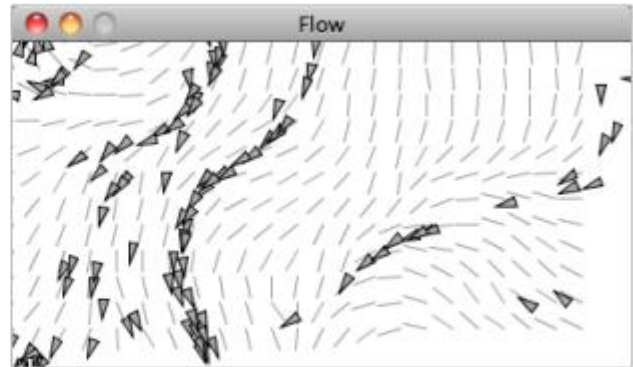
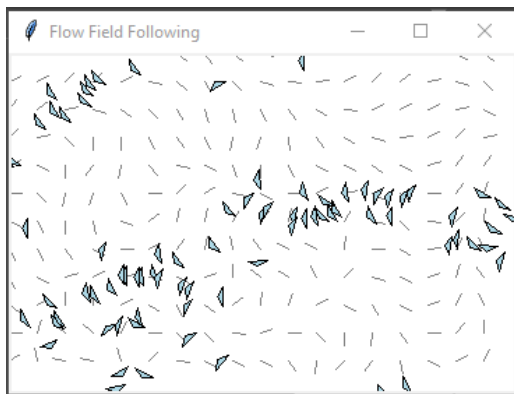


Figure I.13: Flow field with Perlin noise



#SCRIPT18

```
import tkinter as tk
import math
import random
from perlin_noise import PerlinNoise
```

class Vector:

```
    def __init__(self, x=0, y=0):
        self.x = x # Koordinat x
        self.y = y # Koordinat y
```

```
    def add(self, v): # Penjumlahan vektor
        self.x += v.x
        self.y += v.y
```

```
    def sub(self, v): # Pengurangan vektor
        self.x -= v.x
        self.y -= v.y
```

```
    def mult(self, n): # Perkalian skalar
        self.x *= n
        self.y *= n
```

```

def div(self, n): # Pembagian skalar
    if n != 0:
        self.x /= n
        self.y /= n

def mag(self): # Menghitung besar vektor
    return math.sqrt(self.x**2 + self.y**2)

def limit(self, max_val): # Membatasi besar vektor
    m = self.mag()
    if m > max_val:
        self.normalize()
        self.mult(max_val)

def normalize(self): # Membuat vektor menjadi satuan (panjang=1)
    m = self.mag()
    if m != 0:
        self.div(m)

def copy(self): # Membuat salinan vektor
    return Vector(self.x, self.y)

def heading(self): # Menghitung sudut vektor
    return math.atan2(self.y, self.x)

@staticmethod
def sub_vec(v1, v2): # Static method untuk pengurangan vektor
    return Vector(v1.x - v2.x, v1.y - v2.y)

```

===== FlowField =====

```

class FlowField:
    def __init__(self, resolution, width, height):
        self.resolution = resolution # Ukuran grid
        self.cols = width // resolution # Jumlah kolom
        self.rows = height // resolution # Jumlah baris
        # Inisialisasi grid vektor
        self.field = [[Vector() for _ in range(self.rows)] for _ in range(self.cols)]
        self.noise = PerlinNoise(octaves=2) # Generator noise
        self.init_field() # Inisialisasi medan

    def init_field(self):
        # Mengisi grid dengan vektor berdasarkan Perlin Noise
        for i in range(self.cols):
            for j in range(self.rows):
                angle = self.noise([i * 0.1, j * 0.1]) * 2 * math.pi
                v = Vector(math.cos(angle), math.sin(angle))
                self.field[i][j] = v

    def lookup(self, position):
        # Mencari vektor di posisi tertentu
        col = int(position.x // self.resolution)
        row = int(position.y // self.resolution)
        # Pastikan tidak keluar batas
        col = max(0, min(col, self.cols - 1))

```

```
row = max(0, min(row, self.rows - 1))
return self.field[col][row].copy()
```

```
def display(self, canvas):
    # Visualisasi grid flow field
    for i in range(self.cols):
        for j in range(self.rows):
            x = i * self.resolution
            y = j * self.resolution
            v = self.field[i][j]
            # Gambar garis kecil menunjukkan arah aliran
            canvas.create_line(x, y, x + v.x * 10, y + v.y * 10, fill="gray")
```

```
# ===== Vehicle =====
```

```
class Vehicle:
```

```
    def __init__(self, x, y, maxspeed, maxforce, width, height):
        self.position = Vector(x, y) # Posisi awal
        self.velocity = Vector() # Kecepatan
        self.acceleration = Vector() # Percepatan
        self.maxspeed = maxspeed # Kecepatan maksimum
        self.maxforce = maxforce # Gaya maksimum
        self.r = 4 # Ukuran kendaraan
        self.w = width # Lebar area
        self.h = height # Tinggi area
```

```
    def apply_force(self, force):
        # Menerapkan gaya pada kendaraan
        self.acceleration.add(force)
```

```
    def follow(self, flowfield):
        # Mengikuti aliran flow field
        desired = flowfield.lookup(self.position) # Dapatkan vektor aliran
        desired.mult(self.maxspeed) # Skalikan dengan kecepatan maks
        steer = Vector.sub_vec(desired, self.velocity) # Hitung gaya steering
        steer.limit(self.maxforce) # Batasi gaya
        self.apply_force(steer) # Terapkan gaya
```

```
    def update(self):
        # Update posisi kendaraan
        self.velocity.add(self.acceleration)
        self.velocity.limit(self.maxspeed)
        self.position.add(self.velocity)
        self.acceleration.mult(0) # Reset percepatan
```

```
    def borders(self):
        # Membungkus kendaraan ke sisi lain jika keluar layar
        if self.position.x < -self.r:
            self.position.x = self.w + self.r
        if self.position.y < -self.r:
            self.position.y = self.h + self.r
        if self.position.x > self.w + self.r:
            self.position.x = -self.r
        if self.position.y > self.h + self.r:
            self.position.y = -self.r
```

```

def display(self, canvas):
    # Gambar kendaraan sebagai segitiga
    theta = self.velocity.heading() + math.pi / 2 # Sudut menghadap
    x = self.position.x
    y = self.position.y
    r = self.r
    # Hitung 3 titik segitiga
    points = [
        (x + math.sin(theta) * -r * 2, y - math.cos(theta) * -r * 2),
        (x + math.sin(theta + math.pi * 2 / 3) * r * 2, y - math.cos(theta + math.pi * 2 / 3) * r * 2),
        (x + math.sin(theta - math.pi * 2 / 3) * r * 2, y - math.cos(theta - math.pi * 2 / 3) * r * 2)
    ]
    canvas.create_polygon(points, fill='lightblue', outline='black')

# ===== Main Program =====
# Konfigurasi
WIDTH = 360 # Lebar canvas
HEIGHT = 240 # Tinggi canvas
debug = True # Mode debug untuk menampilkan flow field

# Inisialisasi GUI
root = tk.Tk()
root.title("Flow Field Following")
canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

# Buat flow field dan kendaraan
flowfield = FlowField(20, WIDTH, HEIGHT) # Grid 20x20
vehicles = [Vehicle(random.uniform(0, WIDTH), random.uniform(0, HEIGHT),
                    random.uniform(2, 5), random.uniform(0.1, 0.5), WIDTH, HEIGHT)
            for _ in range(80)] # 80 kendaraan acak

def draw():
    canvas.delete("all")
    if debug:
        flowfield.display(canvas) # Tampilkan flow field jika debug
    for v in vehicles:
        v.follow(flowfield) # Kendaraan mengikuti aliran
        v.update() # Update posisi
        v.borders() # Cek batas layar
        v.display(canvas) # Gambar kendaraan
    root.after(33, draw) # Loop animasi (~30 fps)

draw() # Mulai animasi
root.mainloop() # Jalankan GUI

```

I.7 Onward

Bagus, kita sekarang sampai pada bagian **I.7 Onward** — yang bisa dianggap sebagai **penutup bab pengantar** dan sekaligus **pengantar ke eksplorasi yang lebih dalam**.

Mari kita bahas secara bertahap, lalu beri penjelasan sederhana **untuk anak SMP**, serta maknanya **dalam konteks pemrograman dan seni generatif (creative coding)**:

Makna Utama Bagian I.7

1. Random itu Mudah... tapi Terbatas

Di awal bab, kita belajar bahwa banyak orang menggunakan `random()` untuk membuat gerakan, warna, bentuk, dll. Tapi kalau semuanya serba acak, hasilnya bisa jadi **terlalu kacau**, tidak teratur, atau bahkan **tidak indah**.

Misalnya:

“Mau gerakin titik ke mana ya? Ya udah random aja.”

Hasilnya: Titik lompat-lompat kayak kacang.

2. Perlin Noise Juga Bisa Jadi Kebiasaan

Setelah belajar Perlin noise, kamu mungkin tergoda untuk pakai itu **di semua hal**:

“Mau warnanya gimana? Perlin noise!”

“Mau bentuknya berubah gimana? Perlin noise!”

“Mau rotasi objek? Perlin noise!”

Nah, ini **bisa jadi jebakan juga**. Perlin noise memang **lebih halus** dari random biasa, tapi **bukan jawaban untuk semua hal**.

3. Yang Penting Adalah Alat dan Aturan

- Setiap sistem yang kamu buat, entah itu simulasi angin, animasi awan, atau gerakan alien di game, **punya aturannya sendiri**.
 - **Kamu sebagai pembuatnya** yang menentukan aturan itu.
 - Semakin banyak **alat (tool)** yang kamu tahu — random, noise, sin/cos, physics, input user, dll — makin banyak **pilihan** kamu dalam mendesain.
-

Kesimpulan: Isi Kotak Alatmu!

Kalau kamu cuma tahu `random()`, kamu cuma punya satu alat.

Kalau kamu tahu `noise()`, kamu punya dua.

Kalau kamu tahu juga `math.sin()`, vector, fractal, flow field, dan sebagainya...

Maka kamu jadi seperti **seniman dan insinyur** yang punya **kotak peralatan penuh!**

Penjelasan untuk Anak SMP:

Bayangkan kamu lagi gambar peta dunia sendiri.

Kalau kamu **hanya pakai angka acak**, mungkin kamu cuma bisa bikin benua acak-acakan. Tapi kalau kamu pakai **Perlin noise**, bentuk benua bisa jadi **lebih mulus**, seperti peta asli.

Tapi...

Kalau kamu **terus-menerus** pakai Perlin noise untuk semuanya, semua lautmu jadi seragam. Semua gunung terlihat mirip. Jadi **bosan** juga, ya?

Analogi Sempel:

Bayangkan kamu gambar dengan satu warna: hitam.

Bisa? Bisa. Tapi cepat membosankan.

Kalau kamu tambahkan merah, biru, kuning, hijau, kamu bisa gambar **lebih bebas**.


Makna Terpenting:







Random bukan salah. Perlin noise juga bukan salah. Tapi jangan berhenti di sana.

Teruslah belajar alat baru. Karena setiap alat punya gaya, rasa, dan cara sendiri.

Apa Setelah Ini?

Setelah bab ini, kamu bisa lanjut belajar:

-  **Simulasi angin dan partikel**

-  Fraktal dan tumbuhan
-  Gaya tarik dan tolak
-  Sistem pintar (autonomous agents)
-  Interaksi dengan pengguna
-  Sistem game sederhana
-  Visualisasi musik atau data