Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Mencari Pasangan Titik Terdekat 3D dengan Algoritma *Divide and Conquer*

Disusun oleh:

Brian Kheng	(13521049)
Dewana Gustavus Haraka Otang	(13521173)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2023

DAFTAR ISI

BAB I

DESKRIPSI MASALAH

Mencari sepasang titik terdekat dengan Algoritma *Divide and Conquer* sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Anda diminta mengembangkan algoritma mencari sepasang titik terdekat pada bidang 3D. Misalkan terdapat *n* buah titik pada ruang 3D. Setiap titik *P* di dalam ruang dinyatakan dengan koordinat P = (x, y, z). Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titk P_1 = (x_1, y_1, z_1) dan P_2 = (x_2, y_2, z_2) dihitung dengan rumus Euclidean berikut:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

SPESIFIKASI TUGAS

- Buatlah program dalam dalam Bahasa C/C++/Java/Python/Golang/Ruby/Perl (pilih salah satu) untuk mencari sepasang titik yang jaraknya terdekat satu sama lain dengan menerapkan algoritma *divide and conquer* untuk penyelesaiannya, dan perbandingannya dengan Algoritma *Brute Force*.
- **<u>Masukan program</u>**:
    - *n*
    - Titik-titik (dibangkitkan secara acak) dalam koordinat (x, y, z)
- **<u>Luaran program</u>**
    - Sepasang titik yang jaraknya terdekat dan nilai jaraknya
    - Banyaknya operasi perhitungan rumus Euclidean
    - Waktu riil dalam detik (spesifikasikan komputer yang digunakan)

- Bonus 1 (Nilai = 7,5) Penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya

- Bonus 2 (nilai = 7,5) Generalisasi program anda sehingga dapat mencari sepasang titik terdekat untuk sekumpulan vektor di R_n , setiap vektor dinyatakan dalam bentuk x = (x_1, x_2, …, x_n)

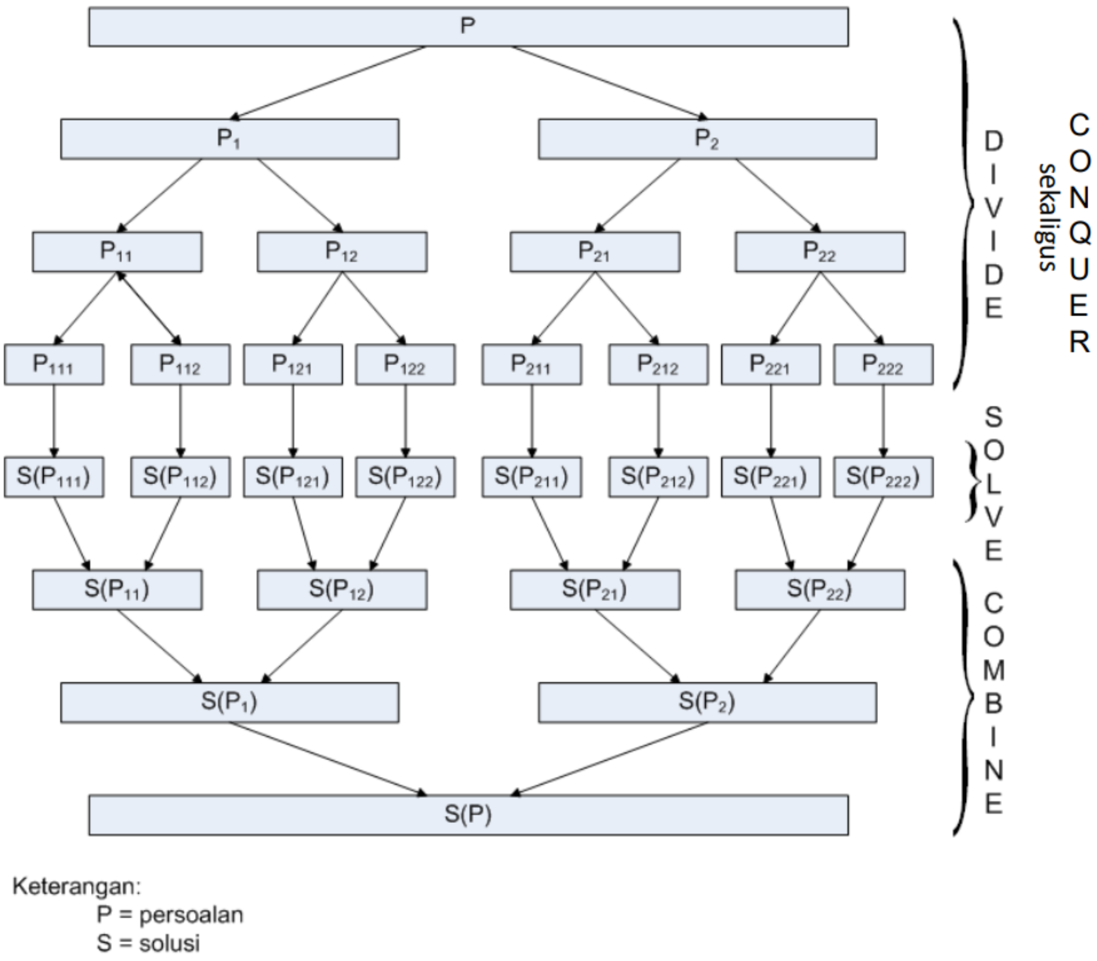
BAB II

TEORI SINGKAT

Teknik *Divide and Conquer* merupakan salah satu metode yang digunakan dalam memecahkan masalah yang kompleks. Metode ini terdiri dari tiga langkah yaitu *divide*, *conquer*, dan *combine*. Langkah pertama, *divide*, yaitu membagi masalah besar menjadi beberapa upa-persoalan yang memiliki kemiripan dengan masalah semula, namun berukuran lebih kecil. Upa-persoalan yang dihasilkan idealnya memiliki ukuran yang hampir sama dengan masalah awal.

Setelah masalah dibagi menjadi upa-persoalan, langkah kedua adalah *conquer* atau menyelesaikan setiap upa-persoalan secara langsung jika sudah berukuran kecil, atau secara rekursif jika masih memiliki ukuran yang besar. Langkah *conquer* bertujuan untuk menyelesaikan setiap upa-persoalan yang sudah dibagi sebelumnya agar lebih mudah dipecahkan.

Langkah terakhir dalam metode *Divide and Conquer* adalah *combine*. Setelah setiap upa-persoalan diselesaikan, langkah *combine* dilakukan untuk menggabungkan solusi dari masing-masing upa-persoalan sehingga membentuk solusi dari masalah awal. Langkah *combine* memungkinkan penyelesaian masalah besar dengan memecahkan beberapa upa-persoalan yang lebih kecil terlebih dahulu, sehingga memudahkan penyelesaiannya secara keseluruhan.



Gambar 2.1. Algoritma *Divide and Conquer* (sumber:

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf)

BAB III

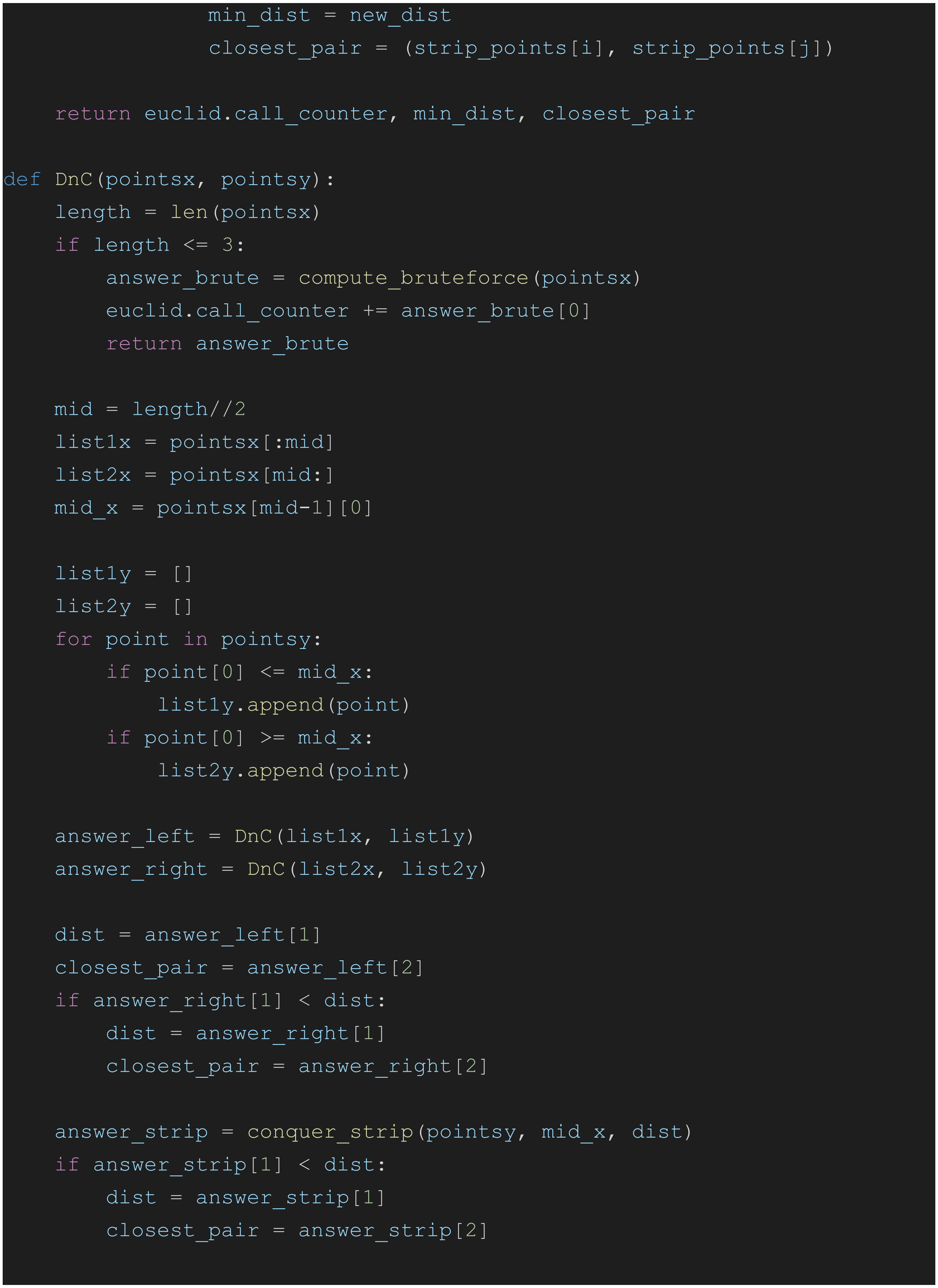
ALGORITMA DIVIDE AND CONQUER

1. Untuk menjalankan algoritma DnC, mula-mula mengambil *list of points* sebagai *input*. Kemudian akan dilakukan pengecekan apakah list tersebut mengandung seminimalnya 2 *points*.
2. Dilakukan *sorting list of points* menggunakan algoritma *merge sort* sebanyak 2 kali yakni, *sorting list of points* berdasarkan sumbu-x dan *sorting list of points* berdasarkan sumbu-y. Lalu, memanggil fungsi rekursif DnC dengan dua parameter tersebut.
3. Fungsi DnC menerima dua input yakni list_points_x dan list_points_y. Fungsi tersebut terus menerus membagi list_points_x menjadi dua bagian hingga mencapai basis dimana jumlah list_points_x ≤ 3. Jika fungsi telah mencapai basis, maka akan dilakukan tahap *SOLVE* dengan algoritma *brute-force*.
4. Jika belum mencapai basis, maka akan dilakukan tahap *DIVIDE* dimana list_points_x akan dibagi menjadi dua bagian sama besar anggap saja list_points_x_1 dan list_points_x_2, kemudian akan dibagi pula list_points_y_1 dan list_points_y_2 dari list_points_y berdasarkan titik tengah list_points_x dan dilakukan pemanggilan fungsi DnC untuk bagian 1 dan bagian 2.
5. Kemudian, pada tahap *COMBINE*, akan dilakukan komparasi untuk diambil jarak terpendek dari bagian 1 dan bagian 2. Akan dilakukan juga pengecekan jarak terpendek untuk lajur yang berada di garis pembagi dengan besar 2 * jarak_terpendek_sekarang secara manual (algoritma *brute-force*) dan mengganti jarak_terpendek_sekarang jika ditemukan.

SOURCE PROGRAM DALAM BAHASA PYTHON

DnC.py

```
from Euclid import Euclid
from Sorting import merge_sort
from Bruteforce import compute_bruteforce
from Randomizer import *

euclid = Euclid()

def compute_DnC(points):
    assert len(points) > 1, "list should contain at least 2 point"
    euclid.call_counter = 0
    pointsx = merge_sort(points, 0)
    pointsy = merge_sort(points, 1)
    return DnC(pointsx, pointsy)

def conquer_strip(pointsy, mid_x, min_dist):
    strip_points = [point for point in pointsy if abs(point[0] - mid_x)
<= min_dist]
    dimension = len(pointsy[0])

    closest_pair = (0, 0)
    strip_length = len(strip_points)
    for i in range(strip_length):
        for j in range(i+1, strip_length):
            if abs(strip_points[i][1] - strip_points[j][1]) >= min_dist:
                break
            skip = False
            for d in range(2, dimension):
                if abs(strip_points[i][d] - strip_points[j][d]) >=
min_dist:
                    skip = True
                    break
            if skip:
                continue
            new_dist = euclid.distance(strip_points[i], strip_points[j])
            if new_dist < min_dist:
```

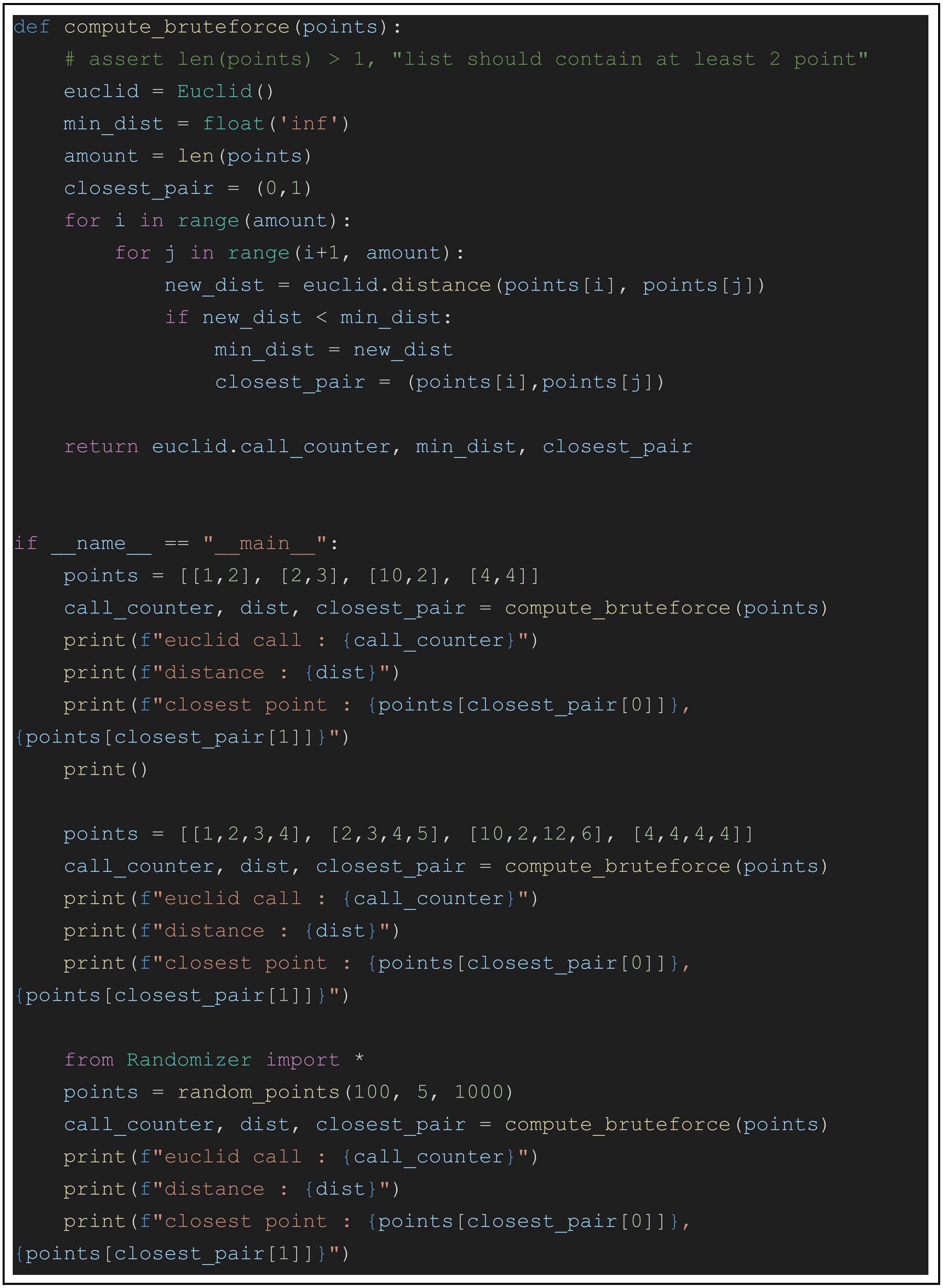


```
                min_dist = new_dist
                closest_pair = (strip_points[i], strip_points[j])

    return euclid.call_counter, min_dist, closest_pair

def DnC(pointsx, pointsy):
    length = len(pointsx)
    if length <= 3:
        answer_brute = compute_bruteforce(pointsx)
        euclid.call_counter += answer_brute[0]
        return answer_brute

    mid = length//2
    list1x = pointsx[:mid]
    list2x = pointsx[mid:]
    mid_x = pointsx[mid-1][0]

    list1y = []
    list2y = []
    for point in pointsy:
        if point[0] <= mid_x:
            list1y.append(point)
        if point[0] >= mid_x:
            list2y.append(point)

    answer_left = DnC(list1x, list1y)
    answer_right = DnC(list2x, list2y)

    dist = answer_left[1]
    closest_pair = answer_left[2]
    if answer_right[1] < dist:
        dist = answer_right[1]
        closest_pair = answer_right[2]

    answer_strip = conquer_strip(pointsy, mid_x, dist)
    if answer_strip[1] < dist:
        dist = answer_strip[1]
        closest_pair = answer_strip[2]
```

```python
    return euclid.call_counter, dist, closest_pair

def loop_driver():
    for i in range(1000):
        points = random_points(1000, 5, 100000000)
        call_counter, dist, closest_pair = compute_bruteforce(points)
        call_counter2, dist2, closest_pair2 = compute_DnC(points)
        if dist != dist2:
            print("wrong")
            print(dist)
            print(dist2)
            print()

def dnc2_driver():
    for i in range(10):
        points = random_points(200, 2, 1000)
        call_counter, dist, closest_pair = compute_bruteforce(points)
        call_counter2, dist2, closest_pair2 = compute_DnC(points)
        call_counter3, dist3, closest_pair3 = compute_DnC2(points)
        print(f"brute euclid call : {call_counter}")
        print(f"DnC euclid call : {call_counter2}")
        print(f"DnC2 euclid call : {call_counter3}")
        print()
        print(f"distance brute : {dist}")
        print(f"distance DnC : {dist2}")
        print(f"distance DnC2 : {dist3}")
        print()


if __name__ == "__main__":
    dnc2_driver()

    # loop_driver()
```

Bruteforce.py

```python
from Euclid import Euclid
```



```python
def compute_bruteforce(points):
    # assert len(points) > 1, "list should contain at least 2 point"
    euclid = Euclid()
    min_dist = float('inf')
    amount = len(points)
    closest_pair = (0,1)
    for i in range(amount):
        for j in range(i+1, amount):
            new_dist = euclid.distance(points[i], points[j])
            if new_dist < min_dist:
                min_dist = new_dist
                closest_pair = (points[i],points[j])

    return euclid.call_counter, min_dist, closest_pair


if __name__ == "__main__":
    points = [[1,2], [2,3], [10,2], [4,4]]
    call_counter, dist, closest_pair = compute_bruteforce(points)
    print(f"euclid call : {call_counter}")
    print(f"distance : {dist}")
    print(f"closest point : {points[closest_pair[0]]},
{points[closest_pair[1]]}")
    print()

    points = [[1,2,3,4], [2,3,4,5], [10,2,12,6], [4,4,4,4]]
    call_counter, dist, closest_pair = compute_bruteforce(points)
    print(f"euclid call : {call_counter}")
    print(f"distance : {dist}")
    print(f"closest point : {points[closest_pair[0]]},
{points[closest_pair[1]]}")

    from Randomizer import *
    points = random_points(100, 5, 1000)
    call_counter, dist, closest_pair = compute_bruteforce(points)
    print(f"euclid call : {call_counter}")
    print(f"distance : {dist}")
    print(f"closest point : {points[closest_pair[0]]},
{points[closest_pair[1]]}")
```

Euclid.py

```python
from math import sqrt

class Euclid:
    def __init__(self):
        self.call_counter = 0

    def distance(self, p1, p2):
        assert len(p1) == len(p2), "Both point should have equal length"
        self.call_counter += 1
        dist2 = 0
        for x1, x2 in zip(p1, p2):
            dx = abs(x1 - x2)
            dist2 += dx*dx
        return sqrt(dist2)
```

Randomizer.py

```python
import random

def random_points(n, m, limit):
    # ? make random so 2 point can't locate in same place
    points = [[random.uniform(0,limit) for __ in range(m)] for _ in range(n)]
    return points

def xyz_random_points(n, limit):
    x = [random.uniform(0,limit) for _ in range(n)]
    y = [random.uniform(0,limit) for _ in range(n)]
    z = [random.uniform(0,limit) for _ in range(n)]
    return (x, y, z)

if __name__ == "__main__":
    print(random_points(3,2,10))
    print(random_points(10,4,20))
```

Sorting.py

```
def merge_sort(input_list, key_index):
    length = len(input_list)
    assert length, "List must contain at least 1 element"
    list = [x for x in input_list]
    if length == 1:
        return list
    mid = length//2
    list1 = list[:mid]
    list2 = list[mid:]
    list1 = merge_sort(list1, key_index)
    list2 = merge_sort(list2, key_index)
    i = 0
    j = 0
    while i < len(list1) or j < len(list2):
        if i == len(list1):
            list[i+j] = list2[j]
            j += 1
        elif j == len(list2):
            list[i+j] = list1[i]
            i += 1
        else:
            if list1[i][key_index] < list2[j][key_index]:
                list[i+j] = list1[i]
                i += 1
            else:
                list[i+j] = list2[j]
                j += 1
    return list

if __name__ == "__main__":
    import random
    list = [[random.randint(1,20), random.randint(0,3)] for i in
range(10)]
    print(list)
    list = merge_sort(list, 0)
    print(list)
    list = [[random.randint(1,20), random.randint(0,3)] for i in
range(10)]
    print(list)
    list = merge_sort(list, 1)
    print(list)
```

Util.py

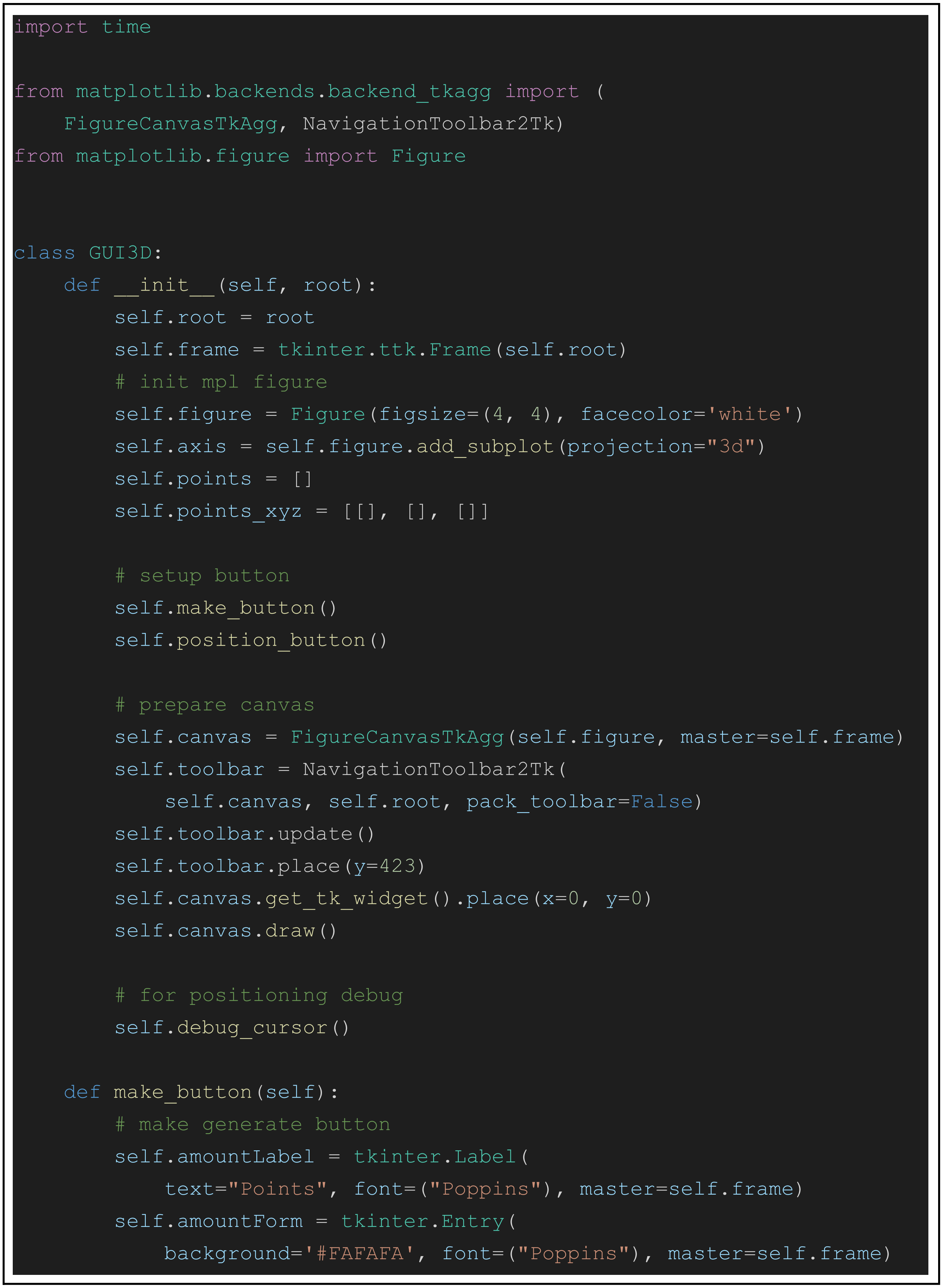
```
def point_to_string(point):
    string = "("
    for xyz in point:
        string += f"{xyz:0.2f}, "
    string = string[:-2] + ")"
    return string

def validate_n(n):
    try:
        n = int(n)
        if n < 2:
            return "points must contain at least 2 point"
        else:
            return n
    except:
        return "please input an integer"

def validate_limit(limit):
    try:
        limit = float(limit)
        return limit
    except:
        return "please input correct decimal value"

def validate_d(d):
    try:
        d = int(d)
        if d < 2:
            return "please input at least 2 dimension"
        else:
            return d
    except:
        return "please input an integer"
```

Main.py

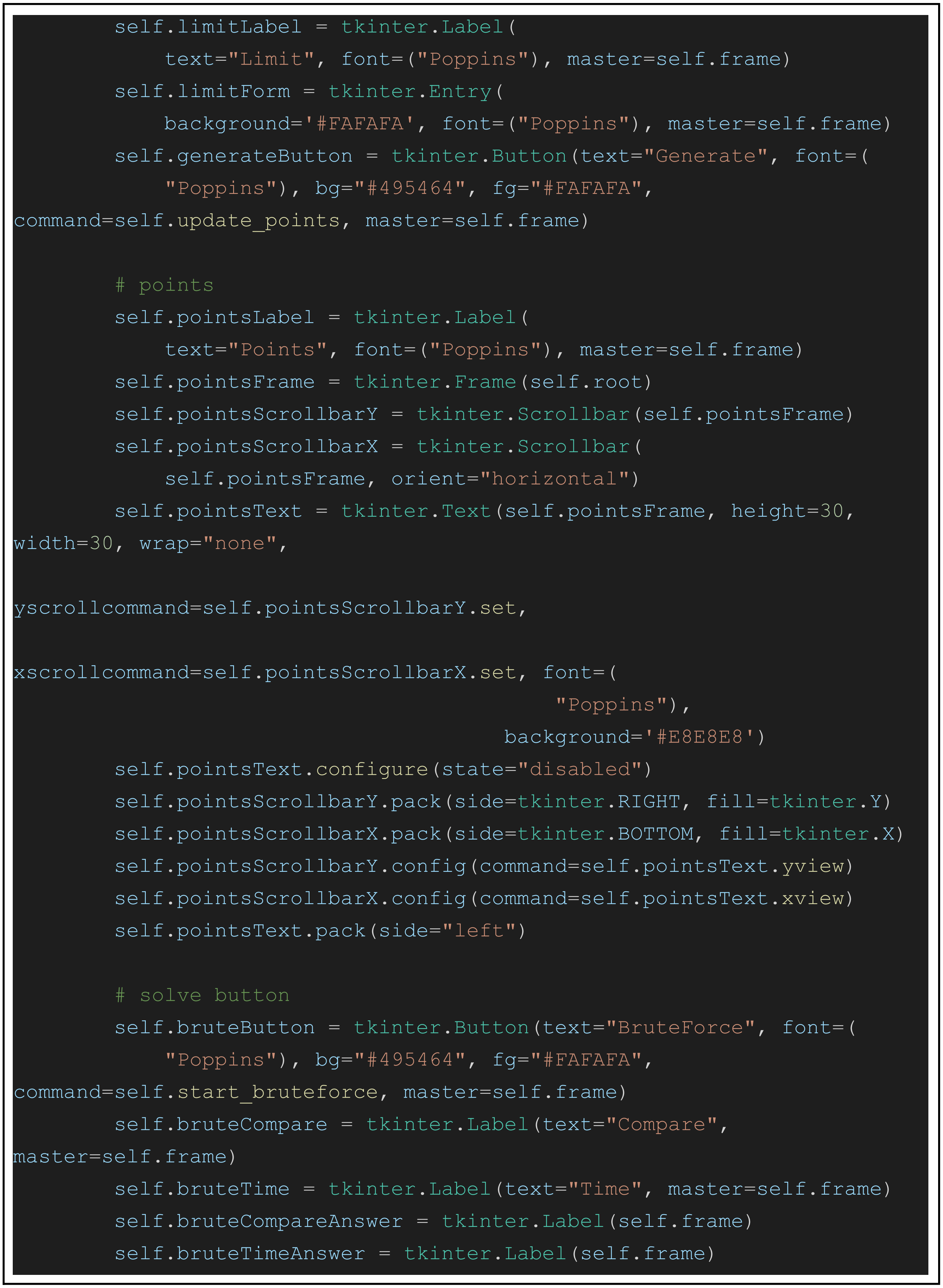
```
from GUI3D import GUI3D
from GUInD import GUInD
import tkinter
import tkinter.ttk

root = tkinter.Tk()
root.title("Closest Pair Visualizer")
tabControl = tkinter.ttk.Notebook(root)

gui1 = GUI3D(tabControl)
tab1 = gui1.frame

gui2 = GUInD(tabControl)
tab2 = gui2.frame

root.geometry("690x750")
root.resizable(False, False)

tabControl.add(tab1, text='3D')
tabControl.add(tab2, text='n-D')
tabControl.pack(expand=1, fill="both")

root.mainloop()
```

GUI3D.py

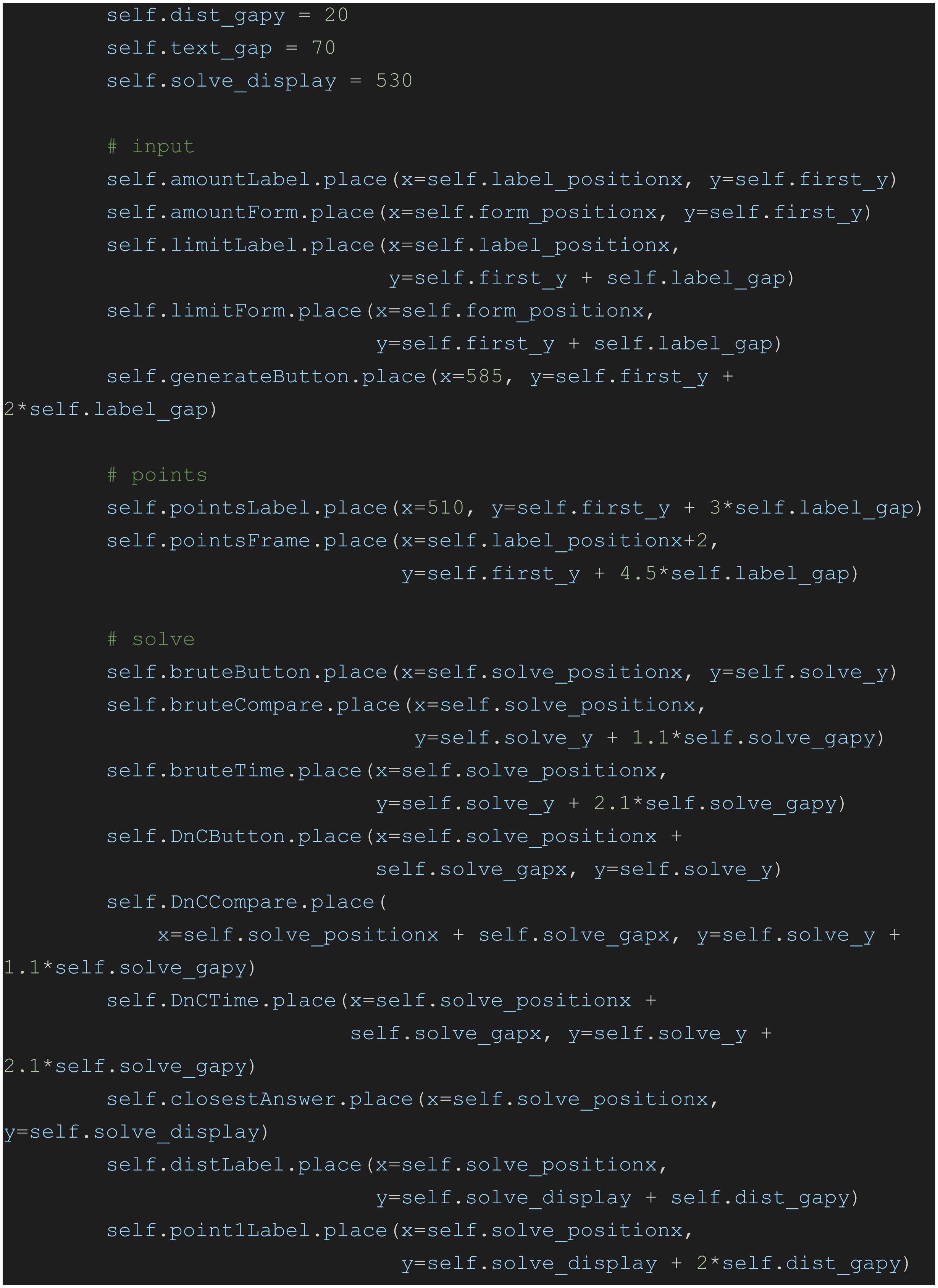
```
from Bruteforce import compute_bruteforce
from DnC import compute_DnC
from Randomizer import xyz_random_points
from Util import *
import tkinter
import tkinter.ttk
```



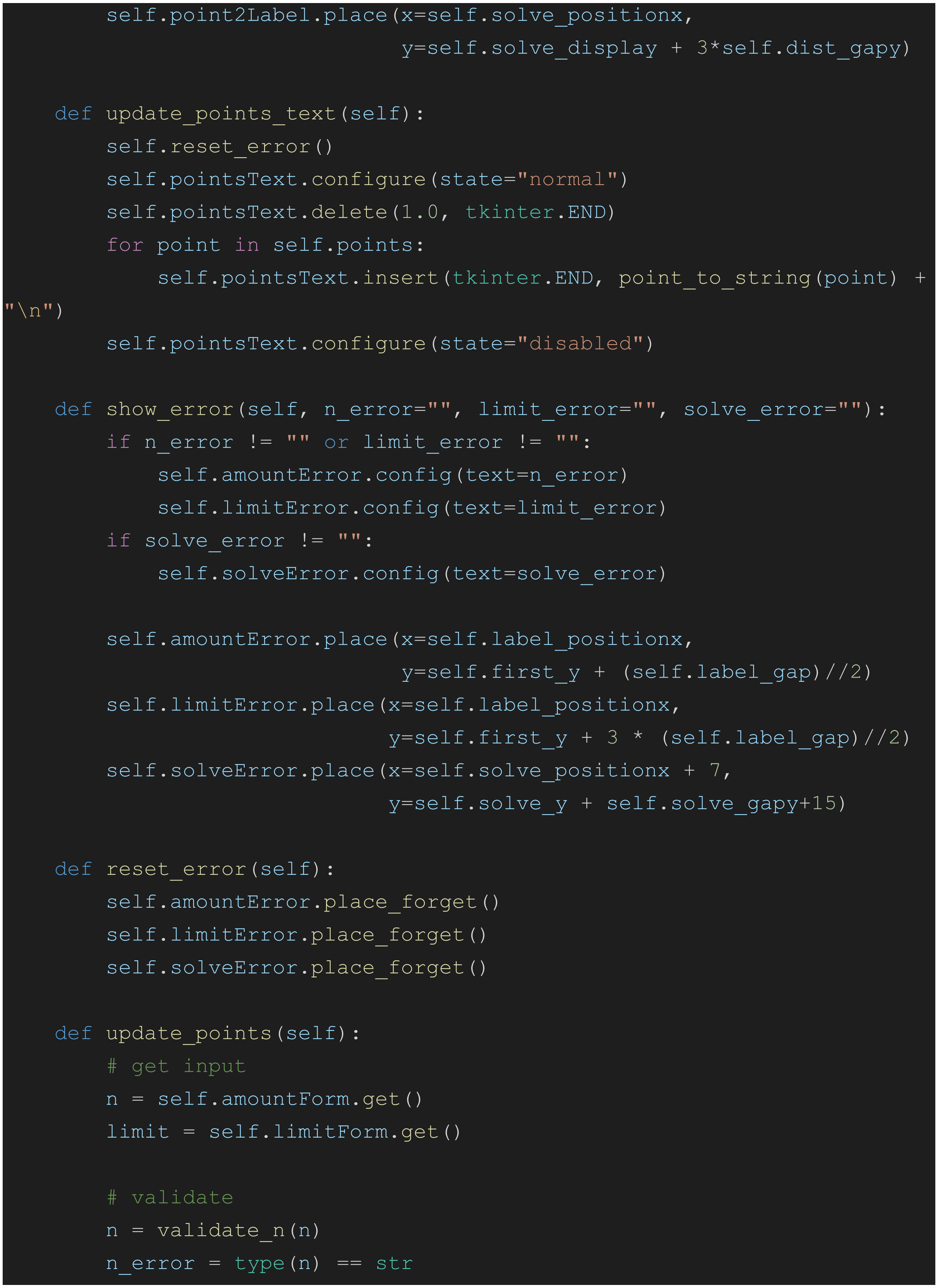
```
import time

from matplotlib.backends.backend_tkagg import (
    FigureCanvasTkAgg, NavigationToolbar2Tk)
from matplotlib.figure import Figure


class GUI3D:
    def __init__(self, root):
        self.root = root
        self.frame = tkinter.ttk.Frame(self.root)
        # init mpl figure
        self.figure = Figure(figsize=(4, 4), facecolor='white')
        self.axis = self.figure.add_subplot(projection="3d")
        self.points = []
        self.points_xyz = [[], [], []]

        # setup button
        self.make_button()
        self.position_button()

        # prepare canvas
        self.canvas = FigureCanvasTkAgg(self.figure, master=self.frame)
        self.toolbar = NavigationToolbar2Tk(
            self.canvas, self.root, pack_toolbar=False)
        self.toolbar.update()
        self.toolbar.place(y=423)
        self.canvas.get_tk_widget().place(x=0, y=0)
        self.canvas.draw()

        # for positioning debug
        self.debug_cursor()

    def make_button(self):
        # make generate button
        self.amountLabel = tkinter.Label(
            text="Points", font=("Poppins"), master=self.frame)
        self.amountForm = tkinter.Entry(
            background='#FAFAFA', font=("Poppins"), master=self.frame)
```



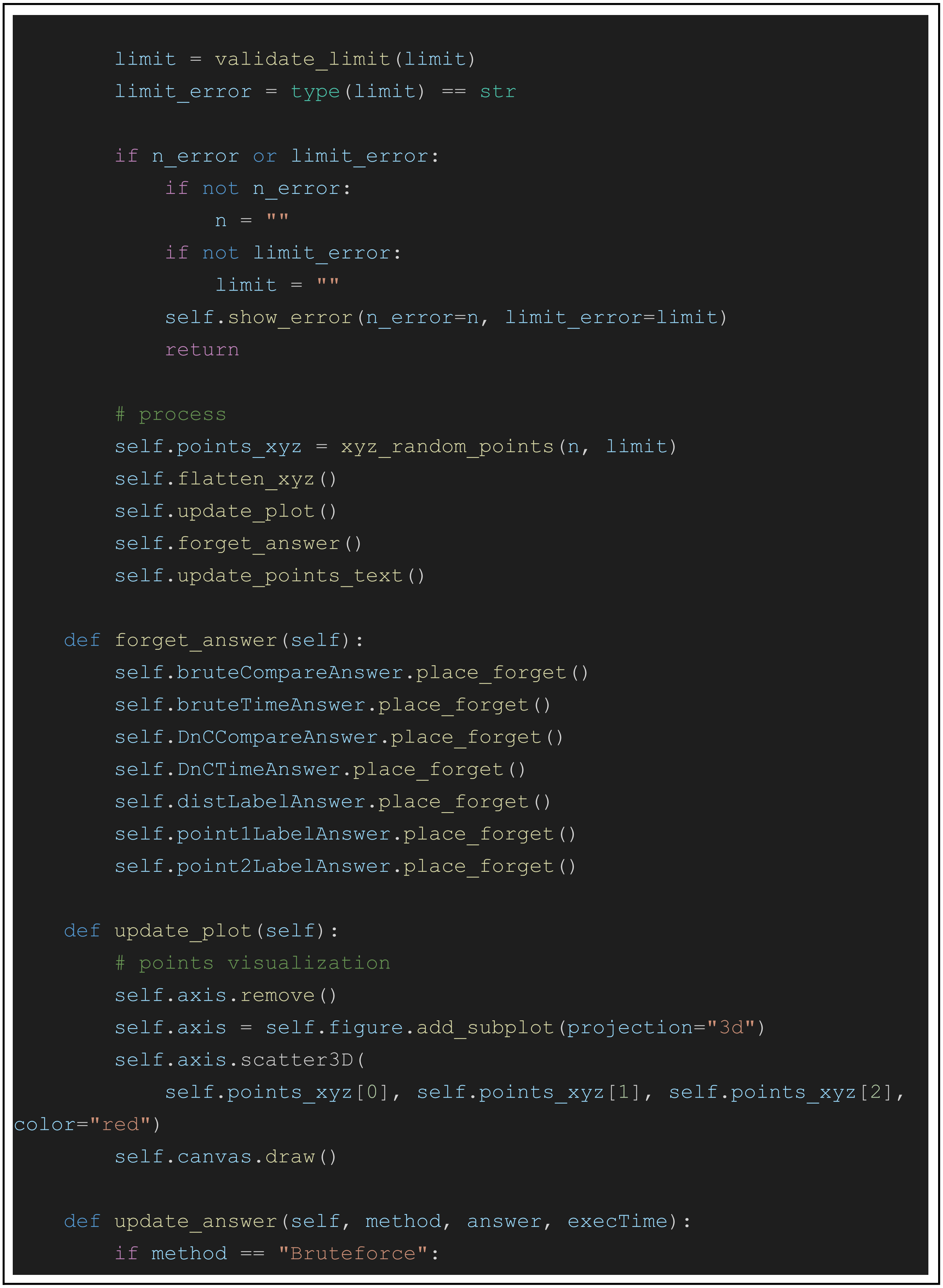
```
        self.limitLabel = tkinter.Label(
            text="Limit", font=("Poppins"), master=self.frame)
        self.limitForm = tkinter.Entry(
            background='#FAFAFA', font=("Poppins"), master=self.frame)
        self.generateButton = tkinter.Button(text="Generate", font=(
            "Poppins"), bg="#495464", fg="#FAFAFA",
command=self.update_points, master=self.frame)

        # points
        self.pointsLabel = tkinter.Label(
            text="Points", font=("Poppins"), master=self.frame)
        self.pointsFrame = tkinter.Frame(self.root)
        self.pointsScrollbarY = tkinter.Scrollbar(self.pointsFrame)
        self.pointsScrollbarX = tkinter.Scrollbar(
            self.pointsFrame, orient="horizontal")
        self.pointsText = tkinter.Text(self.pointsFrame, height=30,
width=30, wrap="none",

yscrollcommand=self.pointsScrollbarY.set,

xscrollcommand=self.pointsScrollbarX.set, font=(
                                           "Poppins"),
                                       background='#E8E8E8')
        self.pointsText.configure(state="disabled")
        self.pointsScrollbarY.pack(side=tkinter.RIGHT, fill=tkinter.Y)
        self.pointsScrollbarX.pack(side=tkinter.BOTTOM, fill=tkinter.X)
        self.pointsScrollbarY.config(command=self.pointsText.yview)
        self.pointsScrollbarX.config(command=self.pointsText.xview)
        self.pointsText.pack(side="left")

        # solve button
        self.bruteButton = tkinter.Button(text="BruteForce", font=(
            "Poppins"), bg="#495464", fg="#FAFAFA",
command=self.start_bruteforce, master=self.frame)
        self.bruteCompare = tkinter.Label(text="Compare",
master=self.frame)
        self.bruteTime = tkinter.Label(text="Time", master=self.frame)
        self.bruteCompareAnswer = tkinter.Label(self.frame)
        self.bruteTimeAnswer = tkinter.Label(self.frame)
        self.DnCButton = tkinter.Button(text="DnC", font=(
            "Poppins"), bg="#495464", fg="#FAFAFA",
command=self.start_DnC, master=self.frame)
        self.DnCCompare = tkinter.Label(text="Compare",
master=self.frame)
        self.DnCTime = tkinter.Label(text="Time", master=self.frame)
        self.DnCCompareAnswer = tkinter.Label(self.frame)
        self.DnCTimeAnswer = tkinter.Label(self.frame)
        self.closestAnswer = tkinter.Label(
            text="Closest Pair", master=self.frame)
        self.distLabel = tkinter.Label(text="Distance",
master=self.frame)
        self.distLabelAnswer = tkinter.Label(self.frame)
        self.point1Label = tkinter.Label(text="Point 1",
master=self.frame)
        self.point1LabelAnswer = tkinter.Label(self.frame)
        self.point2Label = tkinter.Label(text="Point 2",
master=self.frame)
        self.point2LabelAnswer = tkinter.Label(self.frame)

        # error
        self.amountError = tkinter.Label(
            font=("Arial", 8), fg='red', master=self.frame)
        self.limitError = tkinter.Label(
            font=("Arial", 8), fg='red', master=self.frame)
        self.solveError = tkinter.Label(
            font=("Arial", 10), fg='red', master=self.frame)

    def position_button(self):
        # constant
        self.label_positionx = 400
        self.first_y = 0
        self.label_gap = 40
        self.form_positionx = 480

        self.solve_positionx = 15
        self.solve_y = 440
        self.solve_gapx = 130
        self.solve_gapy = 30
```



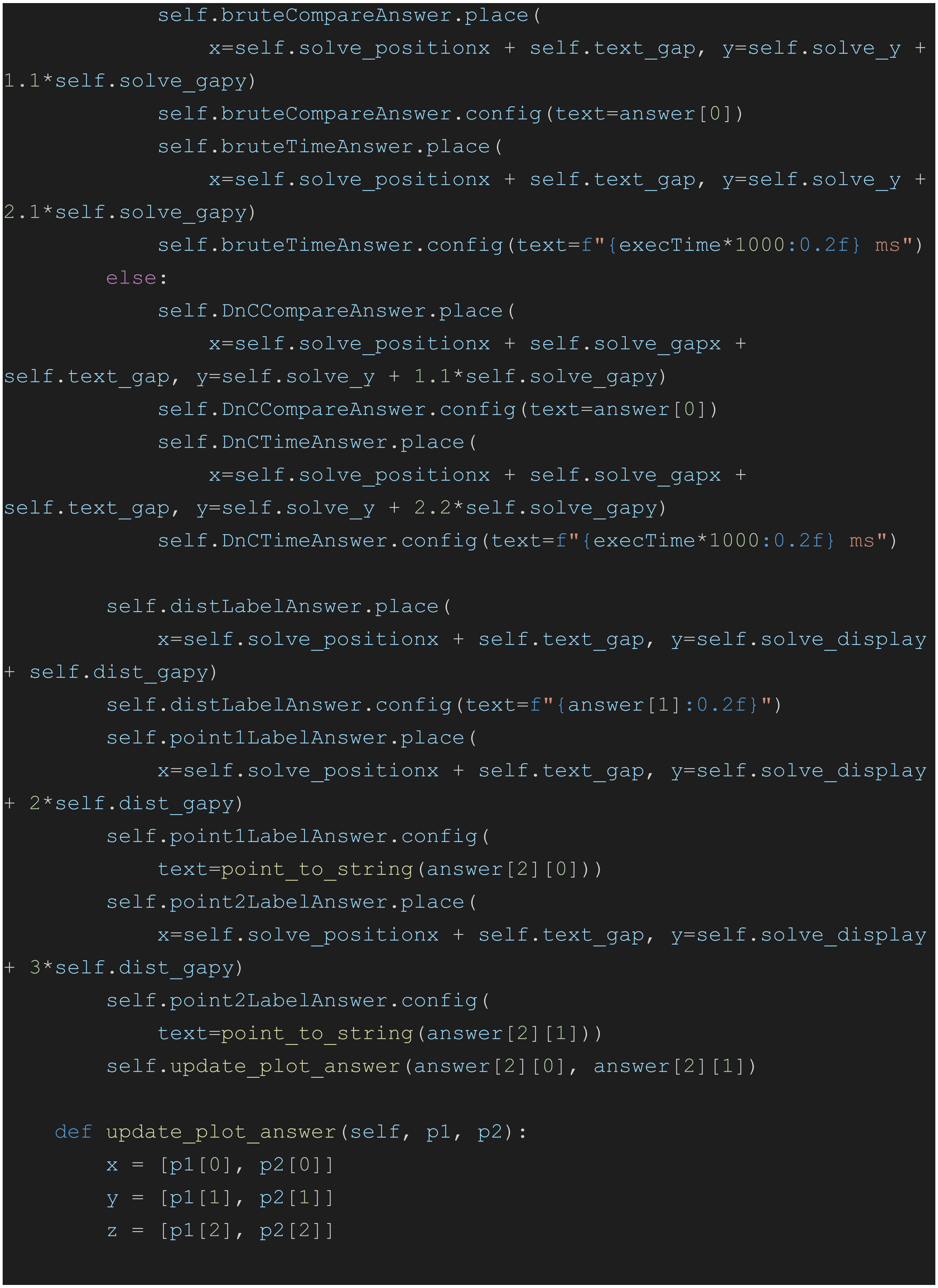
```
        self.dist_gapy = 20
        self.text_gap = 70
        self.solve_display = 530

        # input
        self.amountLabel.place(x=self.label_positionx, y=self.first_y)
        self.amountForm.place(x=self.form_positionx, y=self.first_y)
        self.limitLabel.place(x=self.label_positionx,
                              y=self.first_y + self.label_gap)
        self.limitForm.place(x=self.form_positionx,
                             y=self.first_y + self.label_gap)
        self.generateButton.place(x=585, y=self.first_y +
2*self.label_gap)

        # points
        self.pointsLabel.place(x=510, y=self.first_y + 3*self.label_gap)
        self.pointsFrame.place(x=self.label_positionx+2,
                               y=self.first_y + 4.5*self.label_gap)

        # solve
        self.bruteButton.place(x=self.solve_positionx, y=self.solve_y)
        self.bruteCompare.place(x=self.solve_positionx,
                                y=self.solve_y + 1.1*self.solve_gapy)
        self.bruteTime.place(x=self.solve_positionx,
                             y=self.solve_y + 2.1*self.solve_gapy)
        self.DnCButton.place(x=self.solve_positionx +
                             self.solve_gapx, y=self.solve_y)
        self.DnCCompare.place(
            x=self.solve_positionx + self.solve_gapx, y=self.solve_y +
1.1*self.solve_gapy)
        self.DnCTime.place(x=self.solve_positionx +
                           self.solve_gapx, y=self.solve_y +
2.1*self.solve_gapy)
        self.closestAnswer.place(x=self.solve_positionx,
y=self.solve_display)
        self.distLabel.place(x=self.solve_positionx,
                             y=self.solve_display + self.dist_gapy)
        self.point1Label.place(x=self.solve_positionx,
                               y=self.solve_display + 2*self.dist_gapy)
```



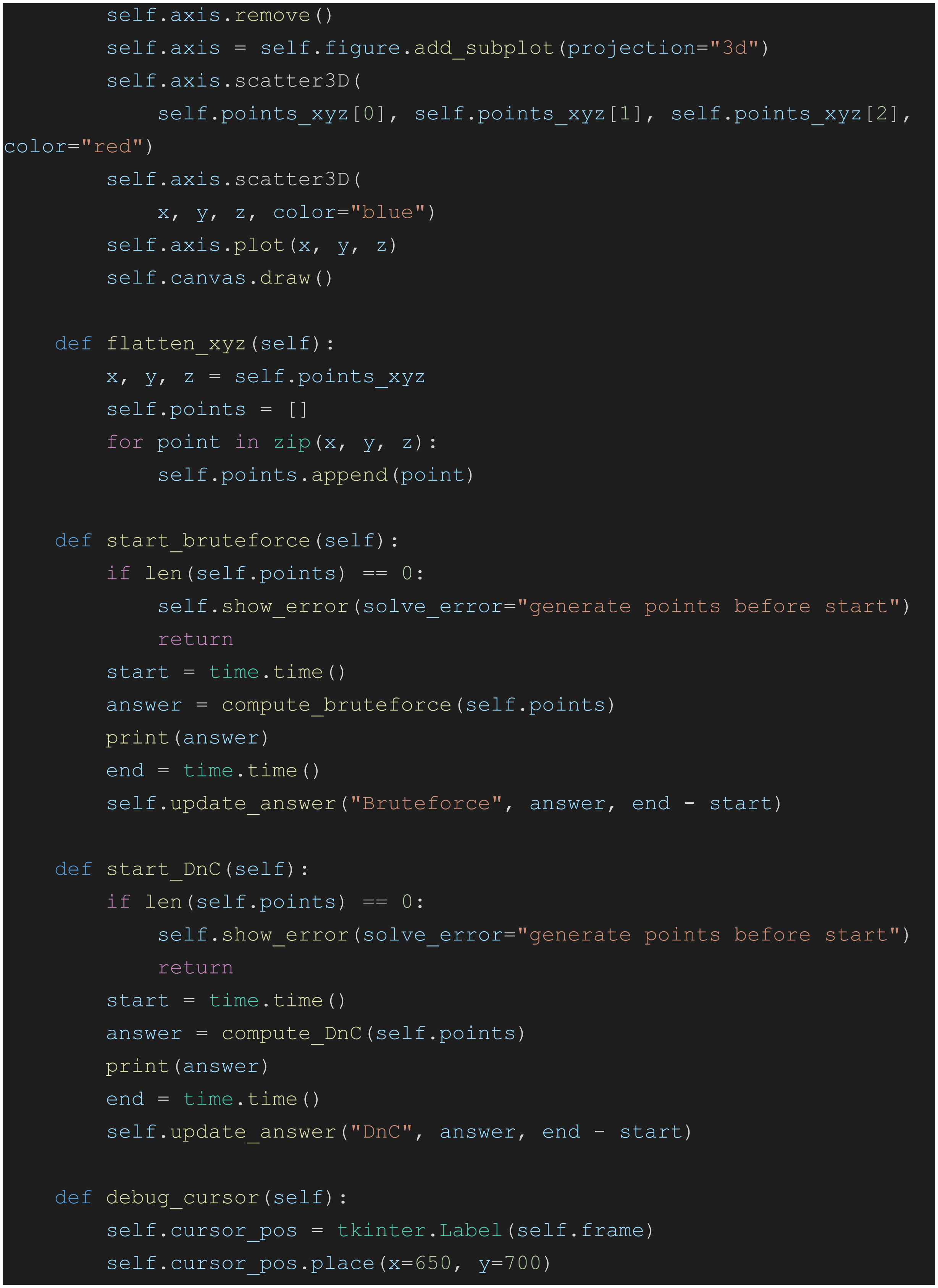
```
        self.point2Label.place(x=self.solve_positionx,
                               y=self.solve_display + 3*self.dist_gapy)

    def update_points_text(self):
        self.reset_error()
        self.pointsText.configure(state="normal")
        self.pointsText.delete(1.0, tkinter.END)
        for point in self.points:
            self.pointsText.insert(tkinter.END, point_to_string(point) +
"\n")
        self.pointsText.configure(state="disabled")

    def show_error(self, n_error="", limit_error="", solve_error=""):
        if n_error != "" or limit_error != "":
            self.amountError.config(text=n_error)
            self.limitError.config(text=limit_error)
        if solve_error != "":
            self.solveError.config(text=solve_error)

        self.amountError.place(x=self.label_positionx,
                               y=self.first_y + (self.label_gap)//2)
        self.limitError.place(x=self.label_positionx,
                              y=self.first_y + 3 * (self.label_gap)//2)
        self.solveError.place(x=self.solve_positionx + 7,
                              y=self.solve_y + self.solve_gapy+15)

    def reset_error(self):
        self.amountError.place_forget()
        self.limitError.place_forget()
        self.solveError.place_forget()

    def update_points(self):
        # get input
        n = self.amountForm.get()
        limit = self.limitForm.get()

        # validate
        n = validate_n(n)
        n_error = type(n) == str
```



```
        limit = validate_limit(limit)
        limit_error = type(limit) == str

        if n_error or limit_error:
            if not n_error:
                n = ""
            if not limit_error:
                limit = ""
            self.show_error(n_error=n, limit_error=limit)
            return

        # process
        self.points_xyz = xyz_random_points(n, limit)
        self.flatten_xyz()
        self.update_plot()
        self.forget_answer()
        self.update_points_text()

    def forget_answer(self):
        self.bruteCompareAnswer.place_forget()
        self.bruteTimeAnswer.place_forget()
        self.DnCCompareAnswer.place_forget()
        self.DnCTimeAnswer.place_forget()
        self.distLabelAnswer.place_forget()
        self.point1LabelAnswer.place_forget()
        self.point2LabelAnswer.place_forget()

    def update_plot(self):
        # points visualization
        self.axis.remove()
        self.axis = self.figure.add_subplot(projection="3d")
        self.axis.scatter3D(
            self.points_xyz[0], self.points_xyz[1], self.points_xyz[2],
color="red")
        self.canvas.draw()

    def update_answer(self, method, answer, execTime):
        if method == "Bruteforce":
```



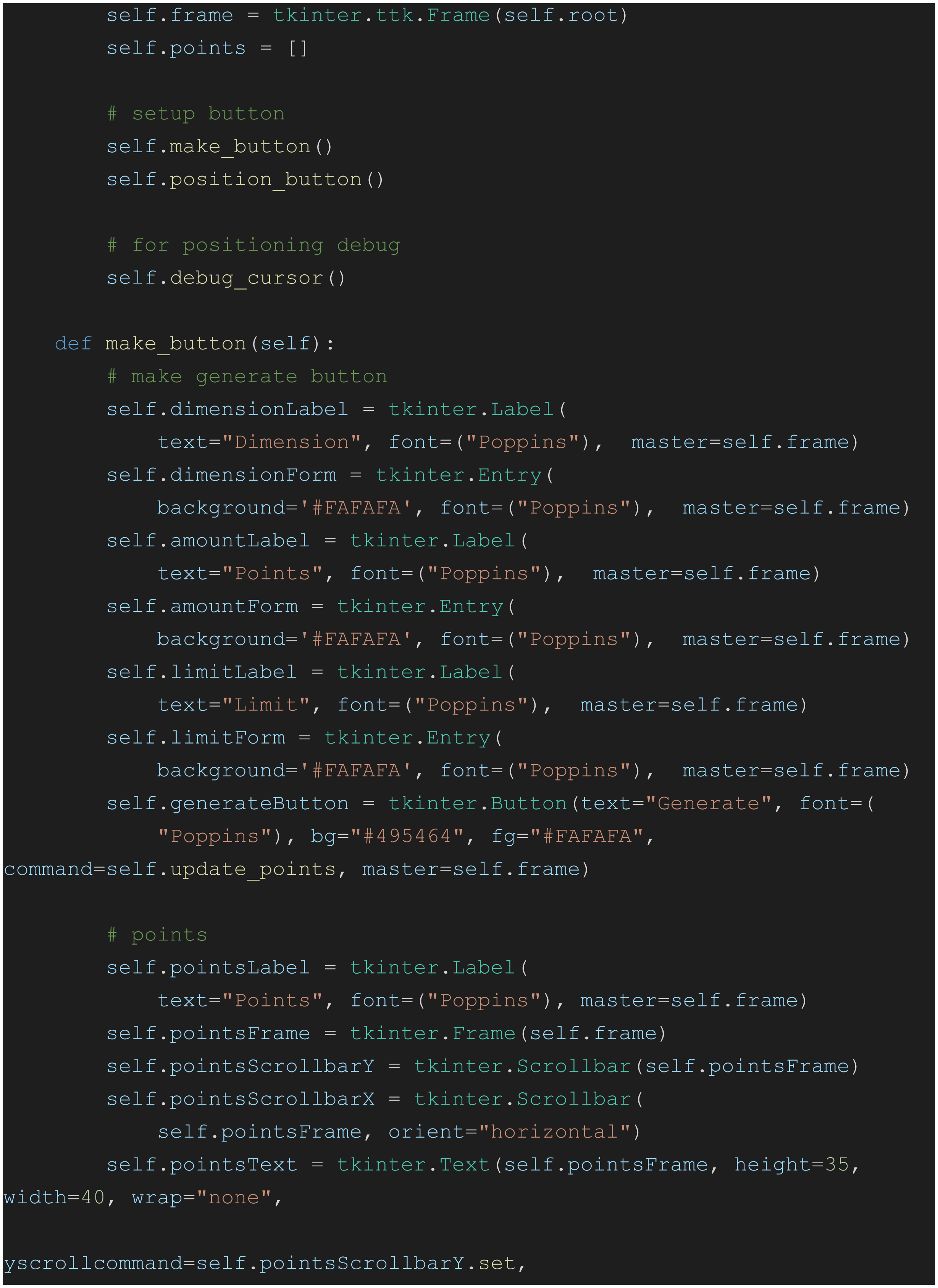
```
            self.bruteCompareAnswer.place(
                x=self.solve_positionx + self.text_gap, y=self.solve_y +
1.1*self.solve_gapy)
            self.bruteCompareAnswer.config(text=answer[0])
            self.bruteTimeAnswer.place(
                x=self.solve_positionx + self.text_gap, y=self.solve_y +
2.1*self.solve_gapy)
            self.bruteTimeAnswer.config(text=f"{execTime*1000:0.2f} ms")
        else:
            self.DnCCompareAnswer.place(
                x=self.solve_positionx + self.solve_gapx +
self.text_gap, y=self.solve_y + 1.1*self.solve_gapy)
            self.DnCCompareAnswer.config(text=answer[0])
            self.DnCTimeAnswer.place(
                x=self.solve_positionx + self.solve_gapx +
self.text_gap, y=self.solve_y + 2.2*self.solve_gapy)
            self.DnCTimeAnswer.config(text=f"{execTime*1000:0.2f} ms")

        self.distLabelAnswer.place(
            x=self.solve_positionx + self.text_gap, y=self.solve_display
+ self.dist_gapy)
        self.distLabelAnswer.config(text=f"{answer[1]:0.2f}")
        self.point1LabelAnswer.place(
            x=self.solve_positionx + self.text_gap, y=self.solve_display
+ 2*self.dist_gapy)
        self.point1LabelAnswer.config(
            text=point_to_string(answer[2][0]))
        self.point2LabelAnswer.place(
            x=self.solve_positionx + self.text_gap, y=self.solve_display
+ 3*self.dist_gapy)
        self.point2LabelAnswer.config(
            text=point_to_string(answer[2][1]))
        self.update_plot_answer(answer[2][0], answer[2][1])

    def update_plot_answer(self, p1, p2):
        x = [p1[0], p2[0]]
        y = [p1[1], p2[1]]
        z = [p1[2], p2[2]]
```



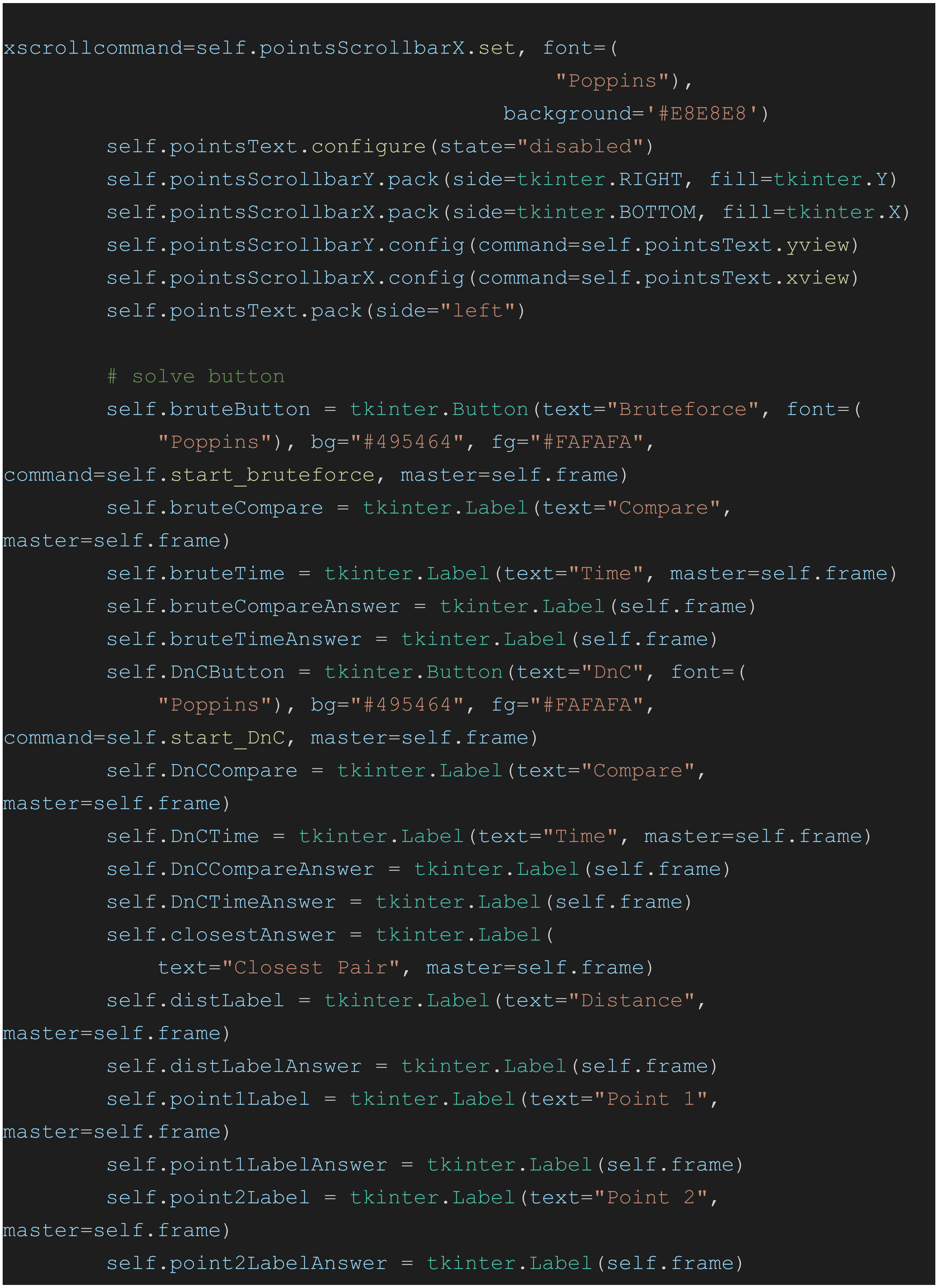
```
        self.axis.remove()
        self.axis = self.figure.add_subplot(projection="3d")
        self.axis.scatter3D(
            self.points_xyz[0], self.points_xyz[1], self.points_xyz[2],
color="red")
        self.axis.scatter3D(
            x, y, z, color="blue")
        self.axis.plot(x, y, z)
        self.canvas.draw()

    def flatten_xyz(self):
        x, y, z = self.points_xyz
        self.points = []
        for point in zip(x, y, z):
            self.points.append(point)

    def start_bruteforce(self):
        if len(self.points) == 0:
            self.show_error(solve_error="generate points before start")
            return
        start = time.time()
        answer = compute_bruteforce(self.points)
        print(answer)
        end = time.time()
        self.update_answer("Bruteforce", answer, end - start)

    def start_DnC(self):
        if len(self.points) == 0:
            self.show_error(solve_error="generate points before start")
            return
        start = time.time()
        answer = compute_DnC(self.points)
        print(answer)
        end = time.time()
        self.update_answer("DnC", answer, end - start)

    def debug_cursor(self):
        self.cursor_pos = tkinter.Label(self.frame)
        self.cursor_pos.place(x=650, y=700)
        self.frame.bind("<Motion>", lambda event:
self.cursor_pos.configure(
            text=f"{event.x}, {event.y}"))

    def start(self):
        # run tkinter
        self.debug_cursor()
        tkinter.mainloop()


def make_root():
    # init tkinter
    frame = tkinter.Tk()
    frame.wm_title("Closest Pair Visualizer")
    frame.geometry("690x720")
    frame.resizable(False, False)
    return frame


if __name__ == "__main__":
    root = make_root()
    program = GUI3D(root)
    program.start()
```

GUInD.py

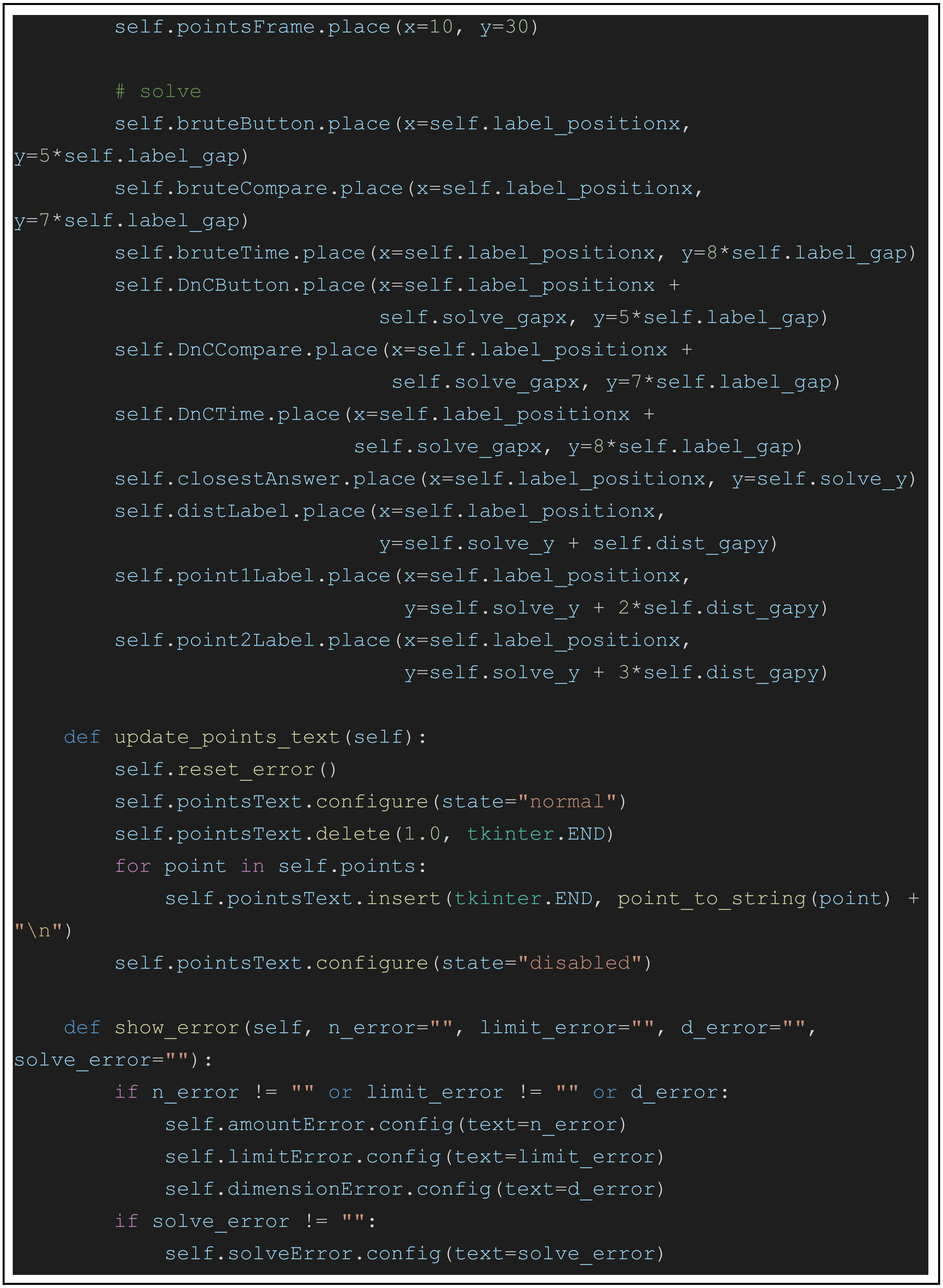
```
from Bruteforce import compute_bruteforce
from DnC import compute_DnC
from Randomizer import random_points
from Util import *
import tkinter
import tkinter.ttk
import time


class GUInD:
    def __init__(self, root):
        self.root = root
```



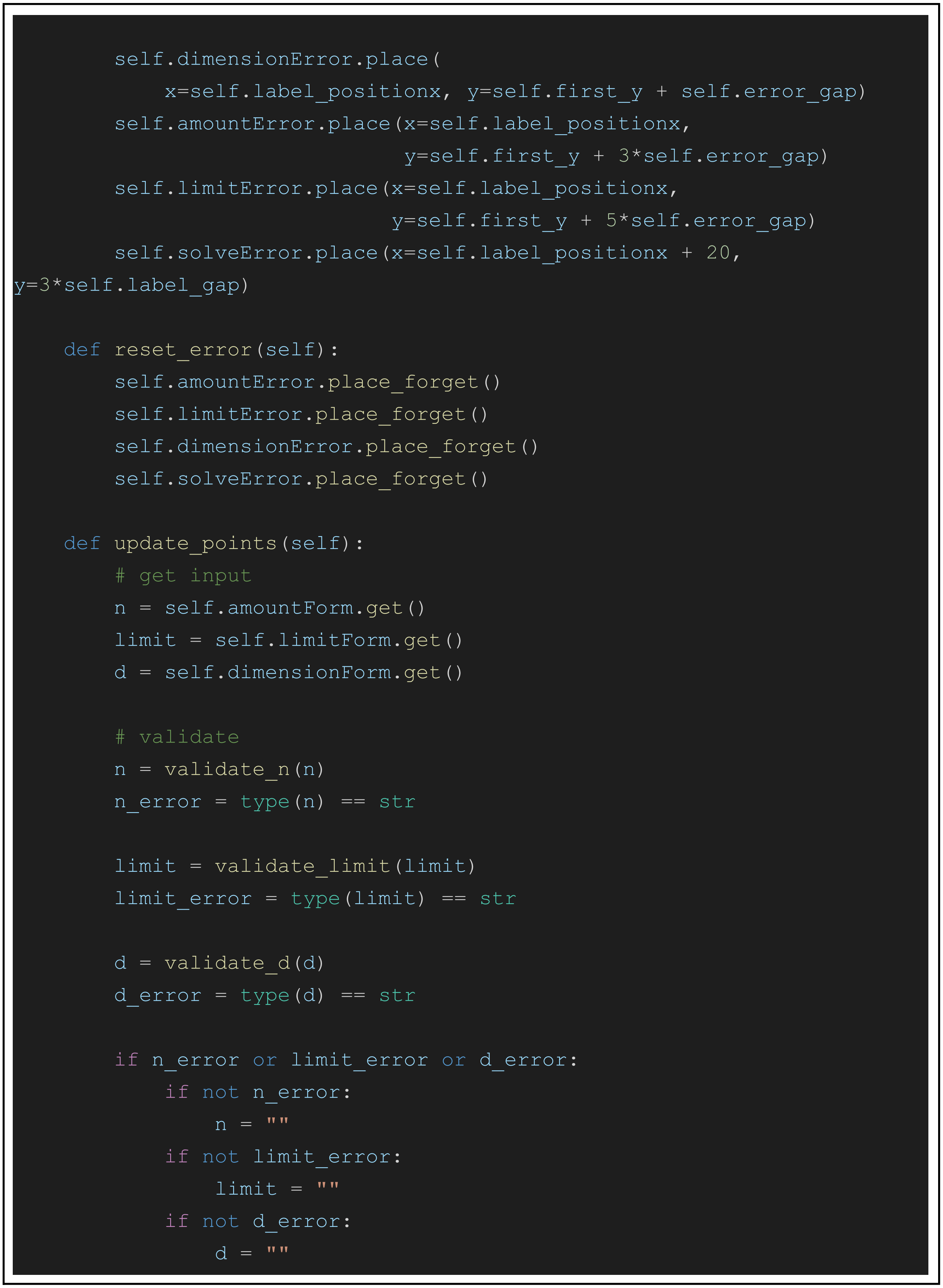
```
        self.frame = tkinter.ttk.Frame(self.root)
        self.points = []

        # setup button
        self.make_button()
        self.position_button()

        # for positioning debug
        self.debug_cursor()

    def make_button(self):
        # make generate button
        self.dimensionLabel = tkinter.Label(
            text="Dimension", font=("Poppins"),  master=self.frame)
        self.dimensionForm = tkinter.Entry(
            background='#FAFAFA', font=("Poppins"),  master=self.frame)
        self.amountLabel = tkinter.Label(
            text="Points", font=("Poppins"),  master=self.frame)
        self.amountForm = tkinter.Entry(
            background='#FAFAFA', font=("Poppins"),  master=self.frame)
        self.limitLabel = tkinter.Label(
            text="Limit", font=("Poppins"),  master=self.frame)
        self.limitForm = tkinter.Entry(
            background='#FAFAFA', font=("Poppins"),  master=self.frame)
        self.generateButton = tkinter.Button(text="Generate", font=(
            "Poppins"), bg="#495464", fg="#FAFAFA",
command=self.update_points, master=self.frame)

        # points
        self.pointsLabel = tkinter.Label(
            text="Points", font=("Poppins"), master=self.frame)
        self.pointsFrame = tkinter.Frame(self.frame)
        self.pointsScrollbarY = tkinter.Scrollbar(self.pointsFrame)
        self.pointsScrollbarX = tkinter.Scrollbar(
            self.pointsFrame, orient="horizontal")
        self.pointsText = tkinter.Text(self.pointsFrame, height=35,
width=40, wrap="none",

yscrollcommand=self.pointsScrollbarY.set,
```



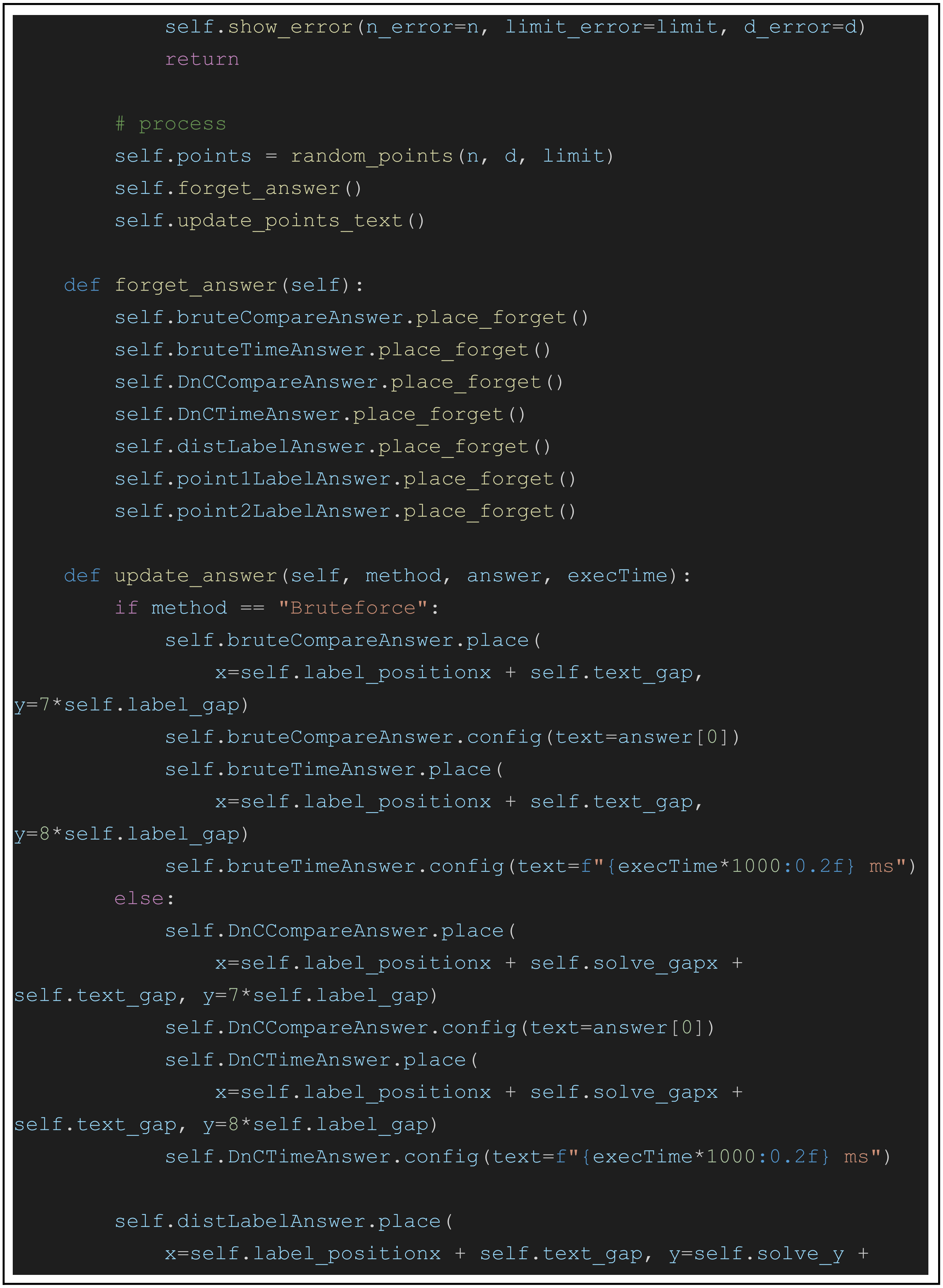
```
xscrollcommand=self.pointsScrollbarX.set, font=(
                                           "Poppins"),
                                       background='#E8E8E8')
        self.pointsText.configure(state="disabled")
        self.pointsScrollbarY.pack(side=tkinter.RIGHT, fill=tkinter.Y)
        self.pointsScrollbarX.pack(side=tkinter.BOTTOM, fill=tkinter.X)
        self.pointsScrollbarY.config(command=self.pointsText.yview)
        self.pointsScrollbarX.config(command=self.pointsText.xview)
        self.pointsText.pack(side="left")

        # solve button
        self.bruteButton = tkinter.Button(text="Bruteforce", font=(
            "Poppins"), bg="#495464", fg="#FAFAFA",
command=self.start_bruteforce, master=self.frame)
        self.bruteCompare = tkinter.Label(text="Compare",
master=self.frame)
        self.bruteTime = tkinter.Label(text="Time", master=self.frame)
        self.bruteCompareAnswer = tkinter.Label(self.frame)
        self.bruteTimeAnswer = tkinter.Label(self.frame)
        self.DnCButton = tkinter.Button(text="DnC", font=(
            "Poppins"), bg="#495464", fg="#FAFAFA",
command=self.start_DnC, master=self.frame)
        self.DnCCompare = tkinter.Label(text="Compare",
master=self.frame)
        self.DnCTime = tkinter.Label(text="Time", master=self.frame)
        self.DnCCompareAnswer = tkinter.Label(self.frame)
        self.DnCTimeAnswer = tkinter.Label(self.frame)
        self.closestAnswer = tkinter.Label(
            text="Closest Pair", master=self.frame)
        self.distLabel = tkinter.Label(text="Distance",
master=self.frame)
        self.distLabelAnswer = tkinter.Label(self.frame)
        self.point1Label = tkinter.Label(text="Point 1",
master=self.frame)
        self.point1LabelAnswer = tkinter.Label(self.frame)
        self.point2Label = tkinter.Label(text="Point 2",
master=self.frame)
        self.point2LabelAnswer = tkinter.Label(self.frame)
        # error
        self.amountError = tkinter.Label(
            font=("Arial", 8), fg='red', master=self.frame)
        self.dimensionError = tkinter.Label(
            font=("Arial", 8), fg='red', master=self.frame)
        self.limitError = tkinter.Label(
            font=("Arial", 8), fg='red', master=self.frame)
        self.solveError = tkinter.Label(
            font=("Arial", 10), fg='red', master=self.frame)

    def position_button(self):
        # constant
        self.label_positionx = 400
        self.first_y = 0
        self.label_gap = 40
        self.form_positionx = 480

        self.solve_positionx = 15
        self.solve_y = 400
        self.solve_gapx = 130
        self.solve_gapy = 30
        self.dist_positionx = 215
        self.dist_gapy = 20
        self.text_gap = 70
        self.error_gap = self.label_gap//2

        # input
        self.dimensionLabel.place(x=self.label_positionx, y=0)
        self.dimensionForm.place(x=self.form_positionx, y=0)
        self.amountLabel.place(x=self.label_positionx, y=self.label_gap)
        self.amountForm.place(x=self.form_positionx, y=self.label_gap)
        self.limitLabel.place(x=self.label_positionx,
y=2*self.label_gap)
        self.limitForm.place(x=self.form_positionx, y=2*self.label_gap)
        self.generateButton.place(x=585, y=3*self.label_gap)

        # points
        self.pointsLabel.place(x=160, y=0)
```



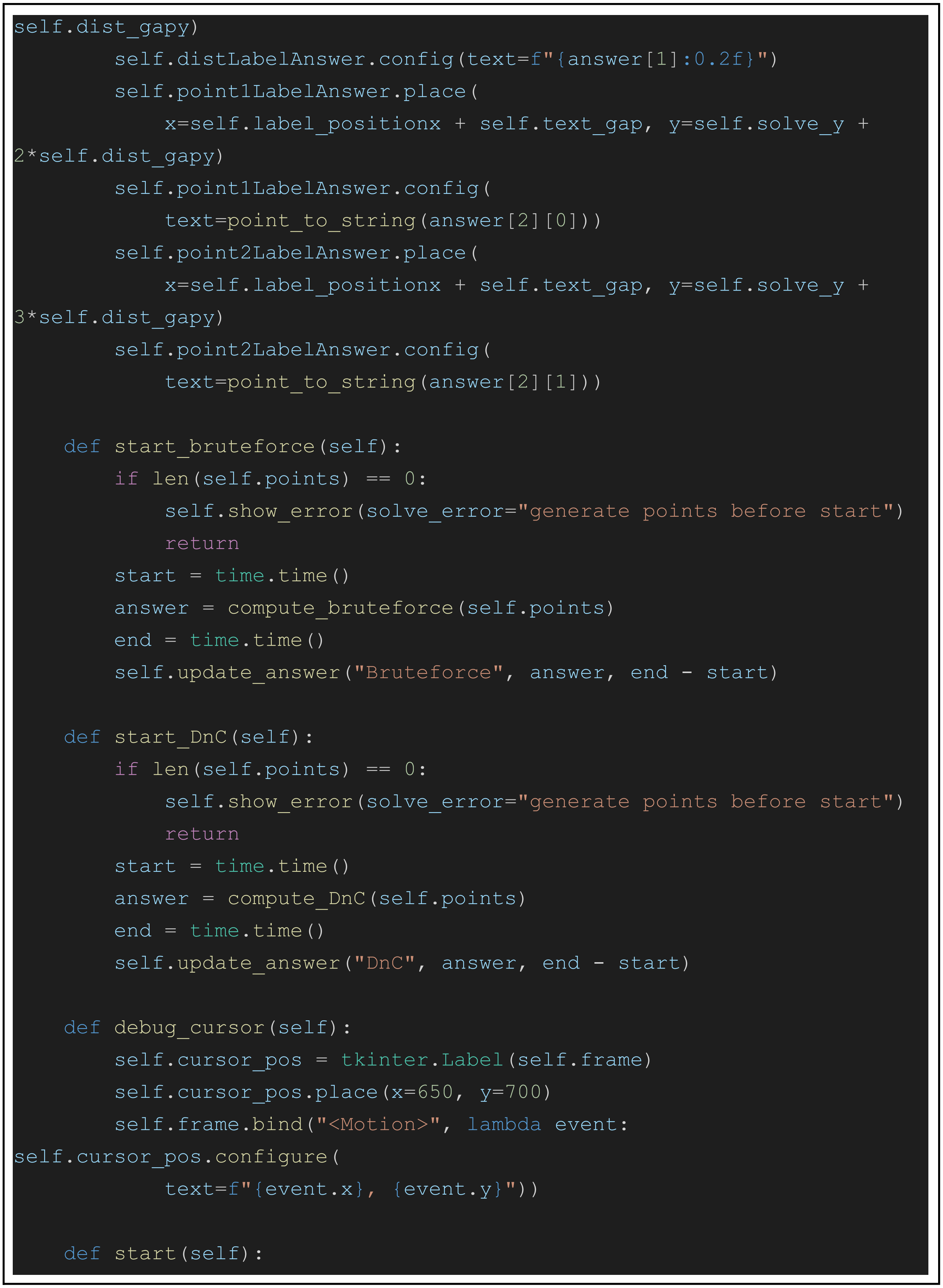
```
        self.pointsFrame.place(x=10, y=30)

        # solve
        self.bruteButton.place(x=self.label_positionx,
y=5*self.label_gap)
        self.bruteCompare.place(x=self.label_positionx,
y=7*self.label_gap)
        self.bruteTime.place(x=self.label_positionx, y=8*self.label_gap)
        self.DnCButton.place(x=self.label_positionx +
                             self.solve_gapx, y=5*self.label_gap)
        self.DnCCompare.place(x=self.label_positionx +
                              self.solve_gapx, y=7*self.label_gap)
        self.DnCTime.place(x=self.label_positionx +
                           self.solve_gapx, y=8*self.label_gap)
        self.closestAnswer.place(x=self.label_positionx, y=self.solve_y)
        self.distLabel.place(x=self.label_positionx,
                             y=self.solve_y + self.dist_gapy)
        self.point1Label.place(x=self.label_positionx,
                               y=self.solve_y + 2*self.dist_gapy)
        self.point2Label.place(x=self.label_positionx,
                               y=self.solve_y + 3*self.dist_gapy)

    def update_points_text(self):
        self.reset_error()
        self.pointsText.configure(state="normal")
        self.pointsText.delete(1.0, tkinter.END)
        for point in self.points:
            self.pointsText.insert(tkinter.END, point_to_string(point) +
"\n")
        self.pointsText.configure(state="disabled")

    def show_error(self, n_error="", limit_error="", d_error="",
solve_error=""):
        if n_error != "" or limit_error != "" or d_error:
            self.amountError.config(text=n_error)
            self.limitError.config(text=limit_error)
            self.dimensionError.config(text=d_error)
        if solve_error != "":
            self.solveError.config(text=solve_error)
```



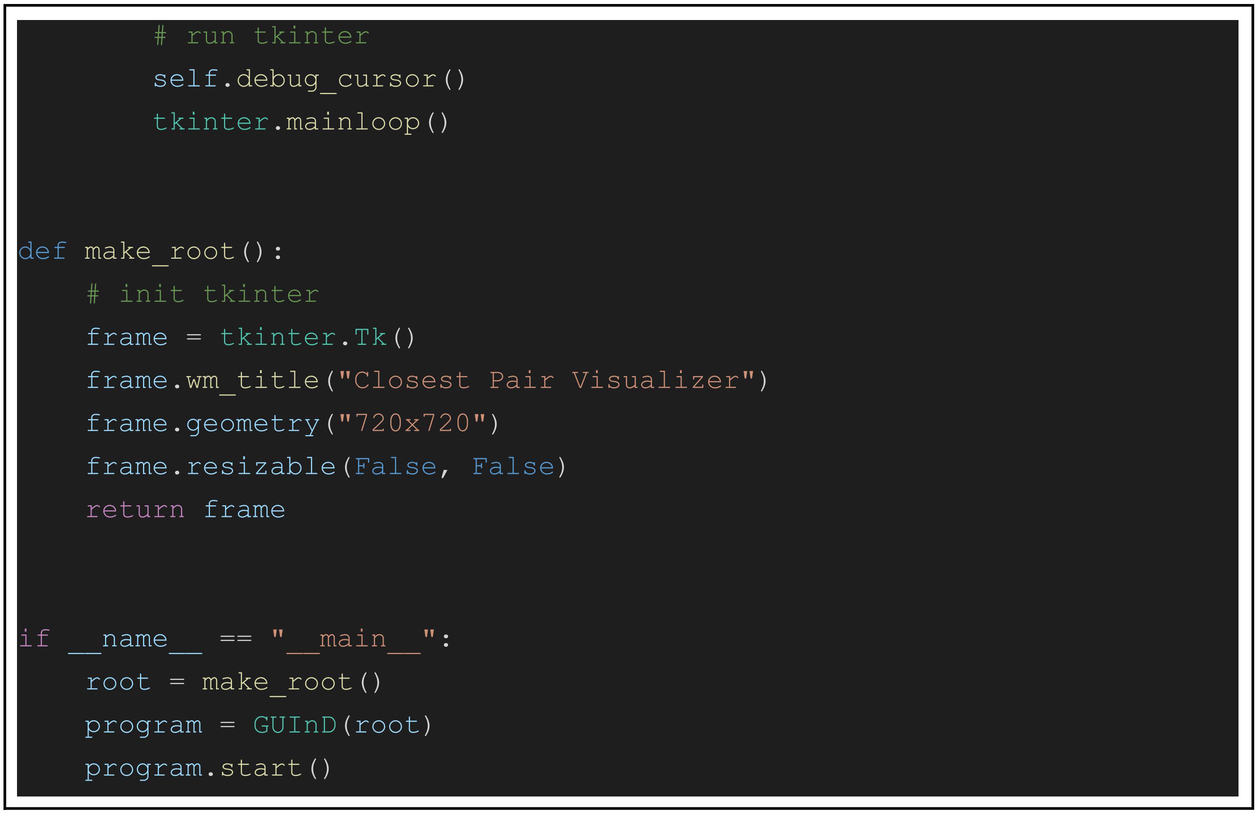
```
        self.dimensionError.place(
            x=self.label_positionx, y=self.first_y + self.error_gap)
        self.amountError.place(x=self.label_positionx,
                               y=self.first_y + 3*self.error_gap)
        self.limitError.place(x=self.label_positionx,
                              y=self.first_y + 5*self.error_gap)
        self.solveError.place(x=self.label_positionx + 20,
y=3*self.label_gap)

    def reset_error(self):
        self.amountError.place_forget()
        self.limitError.place_forget()
        self.dimensionError.place_forget()
        self.solveError.place_forget()

    def update_points(self):
        # get input
        n = self.amountForm.get()
        limit = self.limitForm.get()
        d = self.dimensionForm.get()

        # validate
        n = validate_n(n)
        n_error = type(n) == str

        limit = validate_limit(limit)
        limit_error = type(limit) == str

        d = validate_d(d)
        d_error = type(d) == str

        if n_error or limit_error or d_error:
            if not n_error:
                n = ""
            if not limit_error:
                limit = ""
            if not d_error:
                d = ""
```



```
            self.show_error(n_error=n, limit_error=limit, d_error=d)
            return

        # process
        self.points = random_points(n, d, limit)
        self.forget_answer()
        self.update_points_text()

    def forget_answer(self):
        self.bruteCompareAnswer.place_forget()
        self.bruteTimeAnswer.place_forget()
        self.DnCCompareAnswer.place_forget()
        self.DnCTimeAnswer.place_forget()
        self.distLabelAnswer.place_forget()
        self.point1LabelAnswer.place_forget()
        self.point2LabelAnswer.place_forget()

    def update_answer(self, method, answer, execTime):
        if method == "Bruteforce":
            self.bruteCompareAnswer.place(
                x=self.label_positionx + self.text_gap,
y=7*self.label_gap)
            self.bruteCompareAnswer.config(text=answer[0])
            self.bruteTimeAnswer.place(
                x=self.label_positionx + self.text_gap,
y=8*self.label_gap)
            self.bruteTimeAnswer.config(text=f"{execTime*1000:0.2f} ms")
        else:
            self.DnCCompareAnswer.place(
                x=self.label_positionx + self.solve_gapx +
self.text_gap, y=7*self.label_gap)
            self.DnCCompareAnswer.config(text=answer[0])
            self.DnCTimeAnswer.place(
                x=self.label_positionx + self.solve_gapx +
self.text_gap, y=8*self.label_gap)
            self.DnCTimeAnswer.config(text=f"{execTime*1000:0.2f} ms")

        self.distLabelAnswer.place(
            x=self.label_positionx + self.text_gap, y=self.solve_y +
```



```
self.dist_gapy)
        self.distLabelAnswer.config(text=f"{answer[1]:0.2f}")
        self.point1LabelAnswer.place(
            x=self.label_positionx + self.text_gap, y=self.solve_y +
2*self.dist_gapy)
        self.point1LabelAnswer.config(
            text=point_to_string(answer[2][0]))
        self.point2LabelAnswer.place(
            x=self.label_positionx + self.text_gap, y=self.solve_y +
3*self.dist_gapy)
        self.point2LabelAnswer.config(
            text=point_to_string(answer[2][1]))

    def start_bruteforce(self):
        if len(self.points) == 0:
            self.show_error(solve_error="generate points before start")
            return
        start = time.time()
        answer = compute_bruteforce(self.points)
        end = time.time()
        self.update_answer("Bruteforce", answer, end - start)

    def start_DnC(self):
        if len(self.points) == 0:
            self.show_error(solve_error="generate points before start")
            return
        start = time.time()
        answer = compute_DnC(self.points)
        end = time.time()
        self.update_answer("DnC", answer, end - start)

    def debug_cursor(self):
        self.cursor_pos = tkinter.Label(self.frame)
        self.cursor_pos.place(x=650, y=700)
        self.frame.bind("<Motion>", lambda event:
self.cursor_pos.configure(
            text=f"{event.x}, {event.y}"))

    def start(self):
```



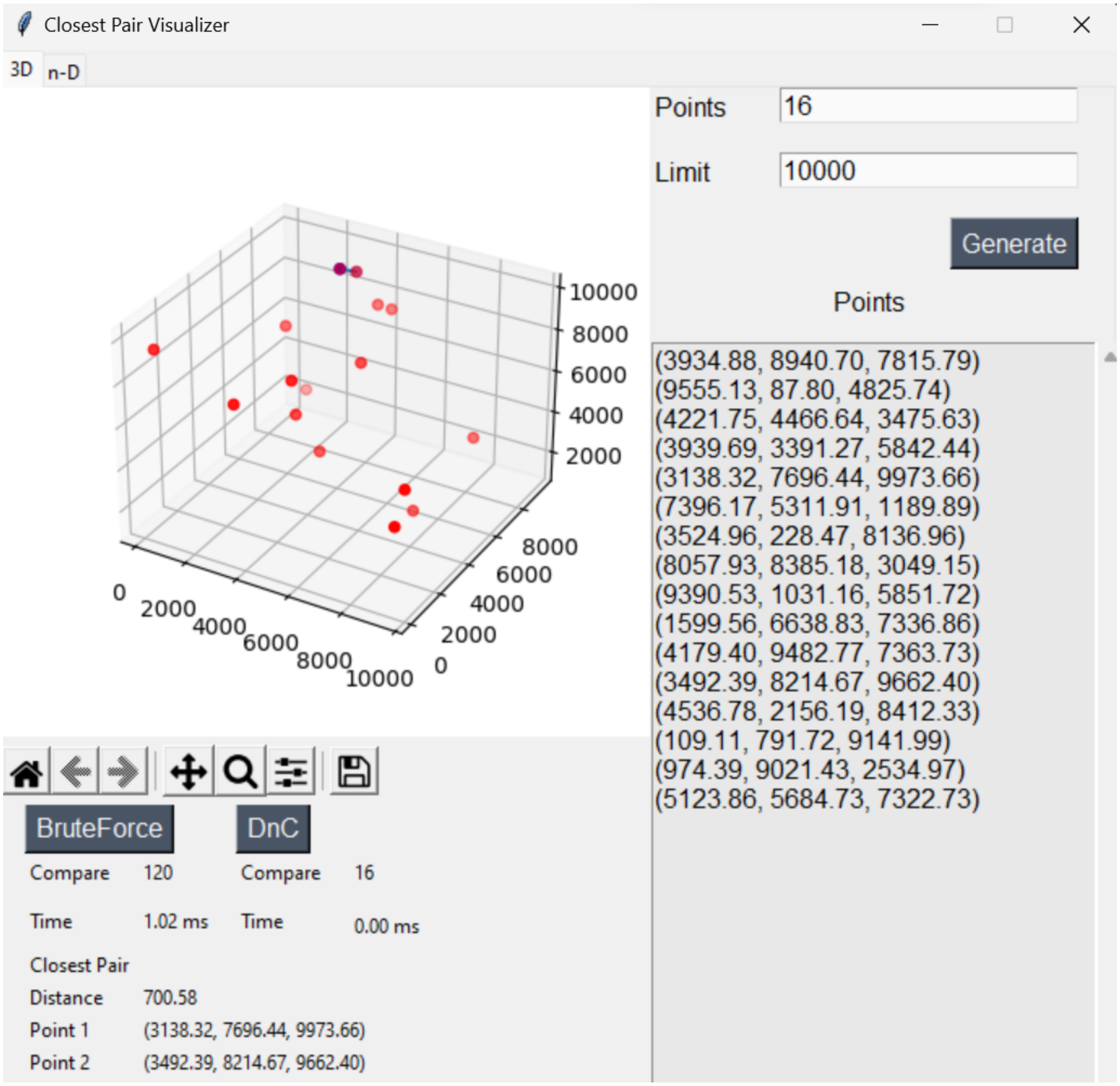
```python
        # run tkinter
        self.debug_cursor()
        tkinter.mainloop()


def make_root():
    # init tkinter
    frame = tkinter.Tk()
    frame.wm_title("Closest Pair Visualizer")
    frame.geometry("720x720")
    frame.resizable(False, False)
    return frame


if __name__ == "__main__":
    root = make_root()
    program = GUInD(root)
    program.start()
```

BAB IV

TESTING PROGRAM

Pada testing program berikut, semua testing dilakukan pada: **Laptop Lenovo Ideapad C340 i5-10210U/MX230**.

1. Test Input: **n = 16**



Points 16

Limit 10000

Points

(3934.88, 8940.70, 7815.79)
(9555.13, 87.80, 4825.74)
(4221.75, 4466.64, 3475.63)
(3939.69, 3391.27, 5842.44)
(3138.32, 7696.44, 9973.66)
(7396.17, 5311.91, 1189.89)
(3524.96, 228.47, 8136.96)
(8057.93, 8385.18, 3049.15)
(9390.53, 1031.16, 5851.72)
(1599.56, 6638.83, 7336.86)
(4179.40, 9482.77, 7363.73)
(3492.39, 8214.67, 9662.40)
(4536.78, 2156.19, 8412.33)
(109.11, 791.72, 9141.99)
(974.39, 9021.43, 2534.97)
(5123.86, 5684.73, 7322.73)

BruteForce		DnC	
Compare	120	Compare	16
Time	1.02 ms	Time	0.00 ms

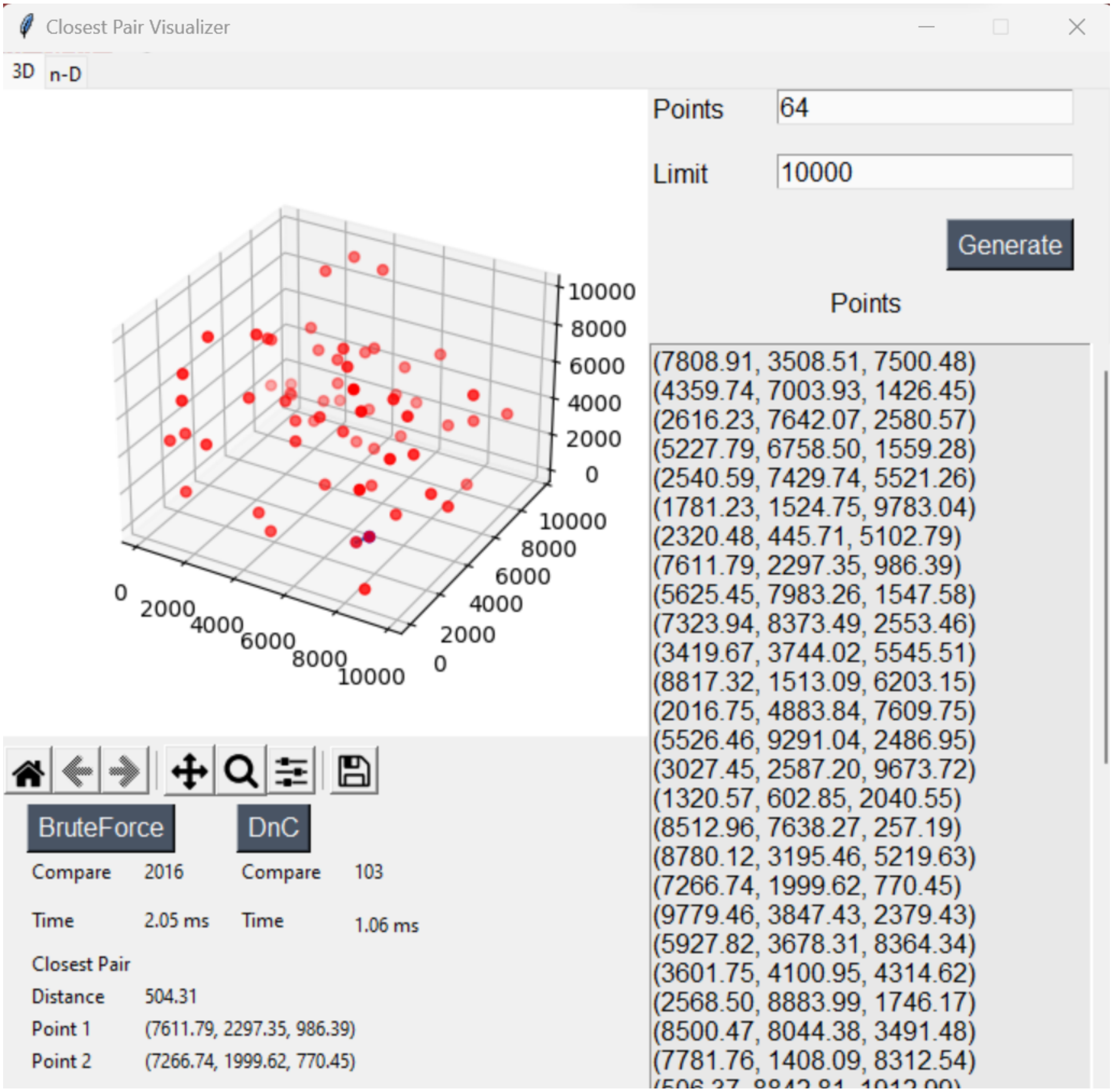
Closest Pair

Distance 700.58

Point 1 (3138.32, 7696.44, 9973.66)

Point 2 (3492.39, 8214.67, 9662.40)

2. Test Input: **n = 64**



Points 64

Limit 10000

Points

(7808.91, 3508.51, 7500.48)
(4359.74, 7003.93, 1426.45)
(2616.23, 7642.07, 2580.57)
(5227.79, 6758.50, 1559.28)
(2540.59, 7429.74, 5521.26)
(1781.23, 1524.75, 9783.04)
(2320.48, 445.71, 5102.79)
(7611.79, 2297.35, 986.39)
(5625.45, 7983.26, 1547.58)
(7323.94, 8373.49, 2553.46)
(3419.67, 3744.02, 5545.51)
(8817.32, 1513.09, 6203.15)
(2016.75, 4883.84, 7609.75)
(5526.46, 9291.04, 2486.95)
(3027.45, 2587.20, 9673.72)
(1320.57, 602.85, 2040.55)
(8512.96, 7638.27, 257.19)
(8780.12, 3195.46, 5219.63)
(7266.74, 1999.62, 770.45)
(9779.46, 3847.43, 2379.43)
(5927.82, 3678.31, 8364.34)
(3601.75, 4100.95, 4314.62)
(2568.50, 8883.99, 1746.17)
(8500.47, 8044.38, 3491.48)
(7781.76, 1408.09, 8312.54)
(506.37, 8842.81, 1012.00)

BruteForce: Compare 2016; Time 2.05 ms

DnC: Compare 103; Time 1.06 ms

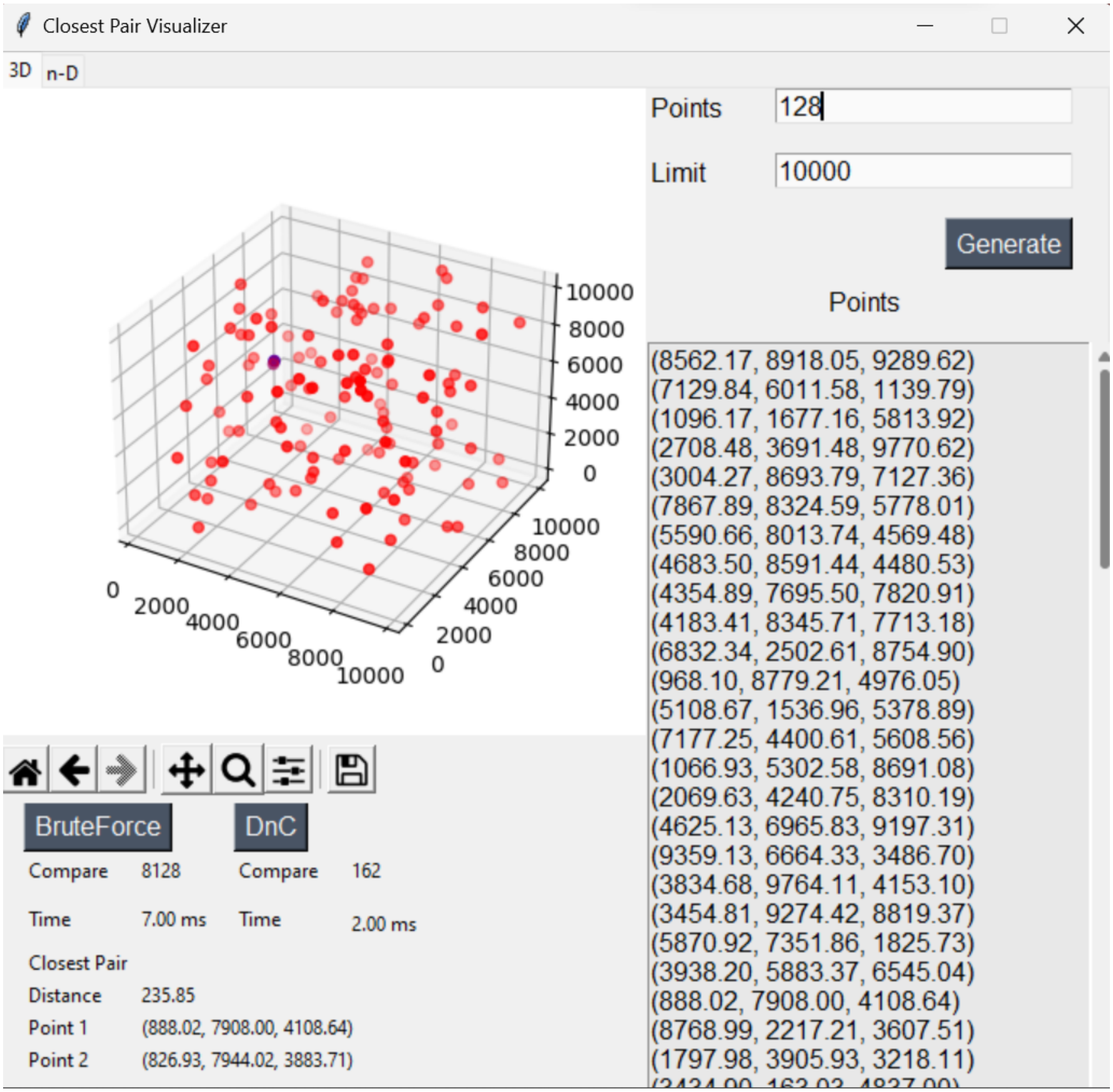
Closest Pair

Distance 504.31

Point 1 (7611.79, 2297.35, 986.39)

Point 2 (7266.74, 1999.62, 770.45)

3. Test Input: **n = 128**



Points 128

Limit 10000

Points

```
(8562.17, 8918.05, 9289.62)
(7129.84, 6011.58, 1139.79)
(1096.17, 1677.16, 5813.92)
(2708.48, 3691.48, 9770.62)
(3004.27, 8693.79, 7127.36)
(7867.89, 8324.59, 5778.01)
(5590.66, 8013.74, 4569.48)
(4683.50, 8591.44, 4480.53)
(4354.89, 7695.50, 7820.91)
(4183.41, 8345.71, 7713.18)
(6832.34, 2502.61, 8754.90)
(968.10, 8779.21, 4976.05)
(5108.67, 1536.96, 5378.89)
(7177.25, 4400.61, 5608.56)
(1066.93, 5302.58, 8691.08)
(2069.63, 4240.75, 8310.19)
(4625.13, 6965.83, 9197.31)
(9359.13, 6664.33, 3486.70)
(3834.68, 9764.11, 4153.10)
(3454.81, 9274.42, 8819.37)
(5870.92, 7351.86, 1825.73)
(3938.20, 5883.37, 6545.04)
(888.02, 7908.00, 4108.64)
(8768.99, 2217.21, 3607.51)
(1797.98, 3905.93, 3218.11)
(3434.90, 163.03, 4837.00)
```

BruteForce		DnC	
Compare	8128	Compare	162
Time	7.00 ms	Time	2.00 ms

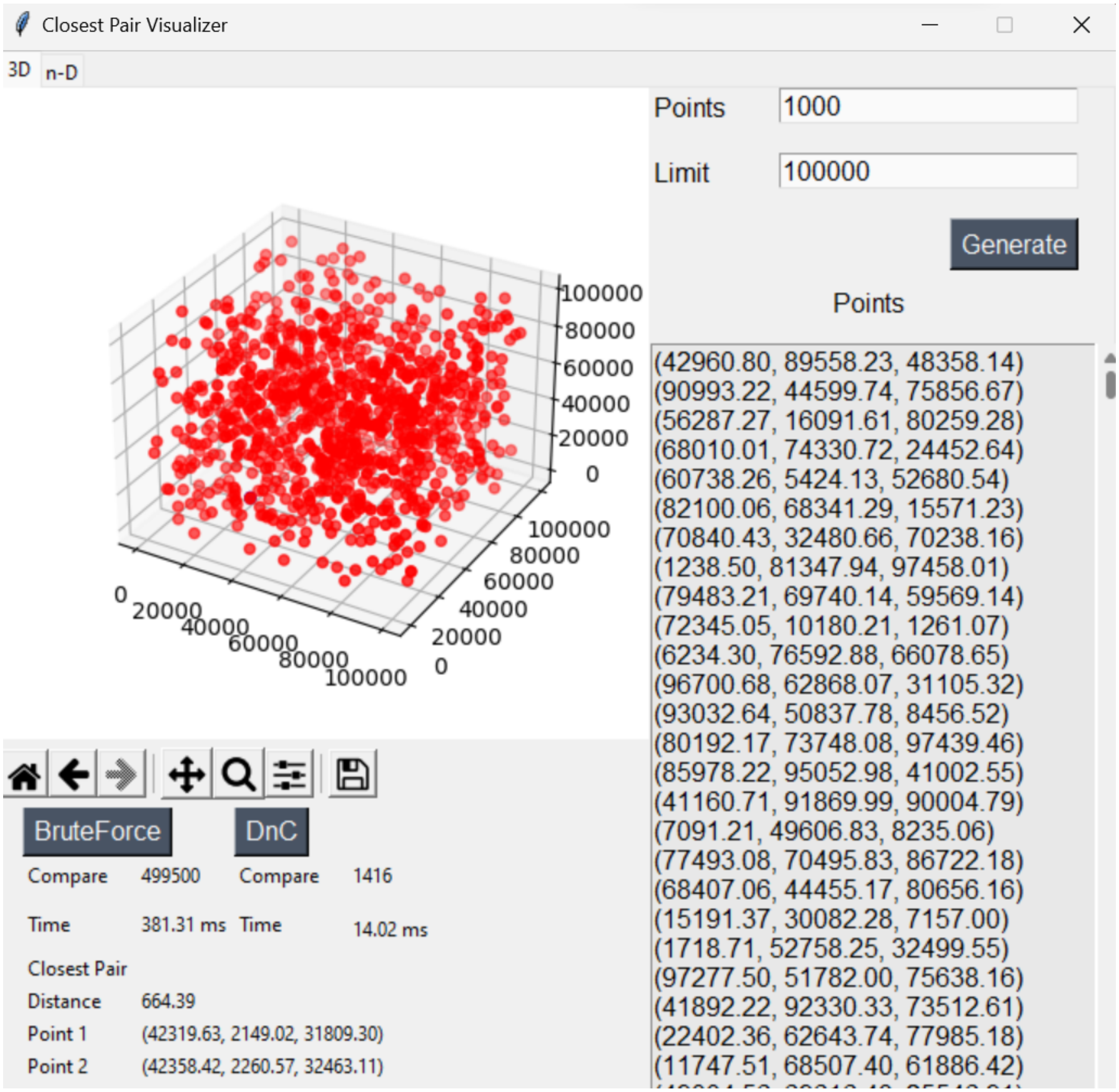
Closest Pair

Distance 235.85

Point 1 (888.02, 7908.00, 4108.64)

Point 2 (826.93, 7944.02, 3883.71)

4. Test Input: **n = 1000**



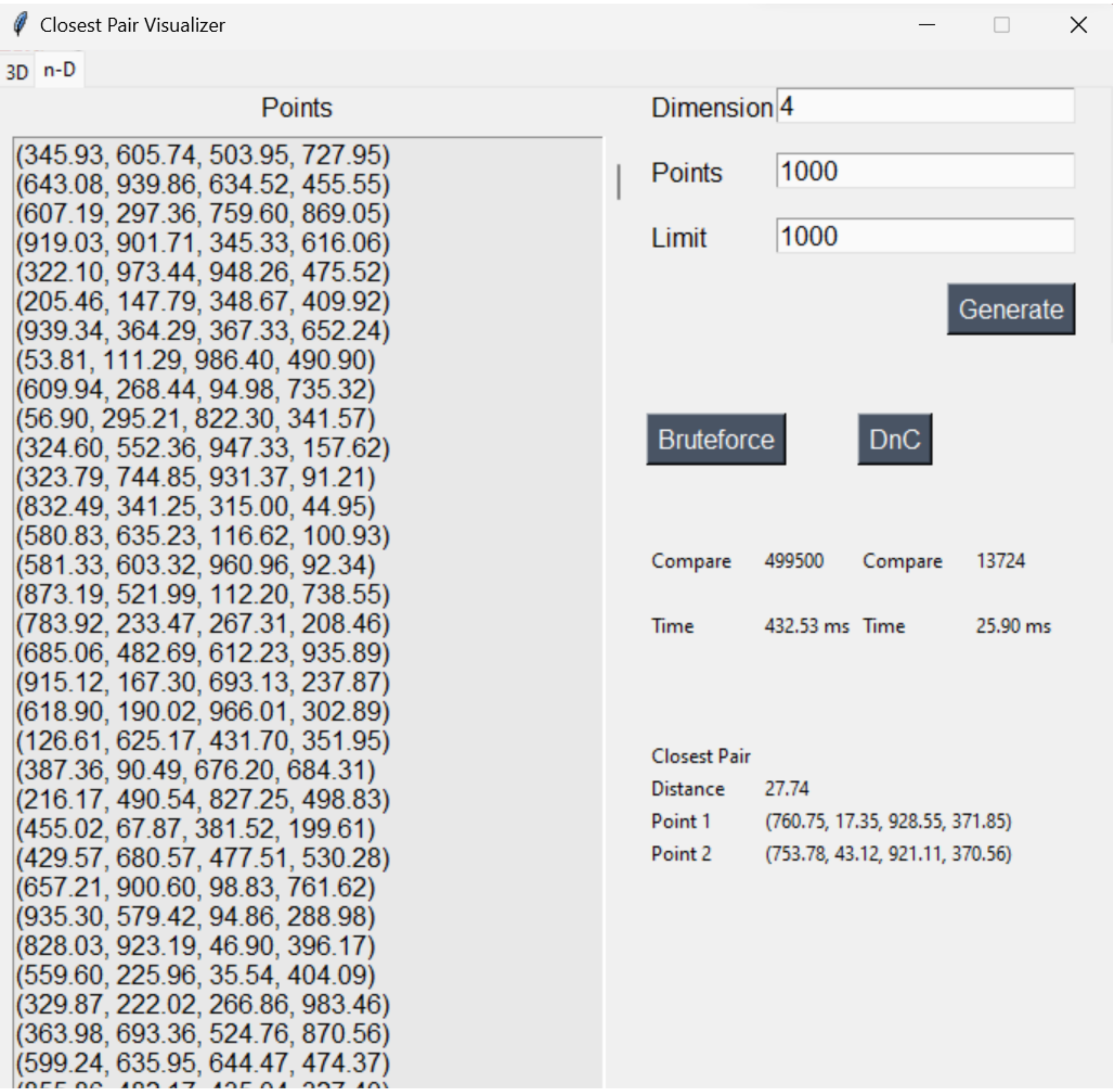
```
Points    1000
Limit     100000
```

Points

```
(42960.80, 89558.23, 48358.14)
(90993.22, 44599.74, 75856.67)
(56287.27, 16091.61, 80259.28)
(68010.01, 74330.72, 24452.64)
(60738.26, 5424.13, 52680.54)
(82100.06, 68341.29, 15571.23)
(70840.43, 32480.66, 70238.16)
(1238.50, 81347.94, 97458.01)
(79483.21, 69740.14, 59569.14)
(72345.05, 10180.21, 1261.07)
(6234.30, 76592.88, 66078.65)
(96700.68, 62868.07, 31105.32)
(93032.64, 50837.78, 8456.52)
(80192.17, 73748.08, 97439.46)
(85978.22, 95052.98, 41002.55)
(41160.71, 91869.99, 90004.79)
(7091.21, 49606.83, 8235.06)
(77493.08, 70495.83, 86722.18)
(68407.06, 44455.17, 80656.16)
(15191.37, 30082.28, 7157.00)
(1718.71, 52758.25, 32499.55)
(97277.50, 51782.00, 75638.16)
(41892.22, 92330.33, 73512.61)
(22402.36, 62643.74, 77985.18)
(11747.51, 68507.40, 61886.42)
```

```
Compare     499500      Compare     1416

Time        381.31 ms   Time        14.02 ms

Closest Pair
Distance    664.39
Point 1     (42319.63, 2149.02, 31809.30)
Point 2     (42358.42, 2260.57, 32463.11)
```

5. Test Input: **4-D, n = 1000**



Points

```
(345.93, 605.74, 503.95, 727.95)
(643.08, 939.86, 634.52, 455.55)
(607.19, 297.36, 759.60, 869.05)
(919.03, 901.71, 345.33, 616.06)
(322.10, 973.44, 948.26, 475.52)
(205.46, 147.79, 348.67, 409.92)
(939.34, 364.29, 367.33, 652.24)
(53.81, 111.29, 986.40, 490.90)
(609.94, 268.44, 94.98, 735.32)
(56.90, 295.21, 822.30, 341.57)
(324.60, 552.36, 947.33, 157.62)
(323.79, 744.85, 931.37, 91.21)
(832.49, 341.25, 315.00, 44.95)
(580.83, 635.23, 116.62, 100.93)
(581.33, 603.32, 960.96, 92.34)
(873.19, 521.99, 112.20, 738.55)
(783.92, 233.47, 267.31, 208.46)
(685.06, 482.69, 612.23, 935.89)
(915.12, 167.30, 693.13, 237.87)
(618.90, 190.02, 966.01, 302.89)
(126.61, 625.17, 431.70, 351.95)
(387.36, 90.49, 676.20, 684.31)
(216.17, 490.54, 827.25, 498.83)
(455.02, 67.87, 381.52, 199.61)
(429.57, 680.57, 477.51, 530.28)
(657.21, 900.60, 98.83, 761.62)
(935.30, 579.42, 94.86, 288.98)
(828.03, 923.19, 46.90, 396.17)
(559.60, 225.96, 35.54, 404.09)
(329.87, 222.02, 266.86, 983.46)
(363.98, 693.36, 524.76, 870.56)
(599.24, 635.95, 644.47, 474.37)
```

Dimension 4

Points 1000

Limit 1000

```
Compare   499500      Compare   13724

Time      432.53 ms   Time      25.90 ms


Closest Pair
Distance  27.74
Point 1   (760.75, 17.35, 928.55, 371.85)
Point 2   (753.78, 43.12, 921.11, 370.56)
```

BAB V

KESIMPULAN

Algoritma DnC dapat digunakan untuk pencarian *closest points* di bidang 3D maupun n-D. Algoritma DnC merupakan algoritma yang lebih efisien dalam pencarian *closest points* dibandingkan dengan algoritma *brute-force*. Pada kasus n = 1000, hanya dibutuhkan waktu sebesar 16.25 ms jika menggunakan algoritma DnC. Sedangkan, dengan algoritma *brute-force* dibutuhkan waktu sebesar 371.00 ms.

SARAN

Dapat dilakukan eksplorasi untuk menemukan algoritma yang lebih efisien dibandingkan algoritma DnC untuk pencarian *closest points*.

DAFTAR PUSTAKA

SUMBER PUSTAKA

1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf (diakses pada tanggal 26 Februari 2023 pukul 19.24 WIB)
2. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf (diakses pada tanggal 26 Februari 2023 pukul 20.08 WIB)
3. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian3.pdf (diakses pada tanggal 26 Februari 2023 pukul 20.19 WIB)
4. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Divide-and-Conquer-(2022)-Bagian4.pdf (diakses pada tanggal 26 Februari 2023 pukul 21.07 WIB)

LAMPIRAN

1. Github : https://github.com/DewanaGustavus/Tucil2_13521049_13521173
2.

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil *running*	✓	
3. Program dapat menerima masukan dan menuliskan luaran	✓	
4. Luaran program sudah benar (solusi *closest pair* benar)	✓	
5. Bonus 1 dikerjakan	✓	
6. Bonus 2 dikerjakan	✓	