Final Report on IKN Design Case

Case Base Team

2201020066 – Obi Luter Sihombing

2201020074 – Adon Alviandre Sirait

Informatics Department

School of Engineering and Maritime Technology Universitas Maritim Raja Ali Haji

2024

1. Latar Belakang

IKN adalah ibu kota baru Indonesia. Sebagai ibu kota baru Indonesia yang direncanakan untuk menggantikan Jakarta sebagai pusat pemerintahan Kota ini akan dirancang dari awal untuk mengakomodasi fungsi dan layanan unit pemerintah Serta dirancang untuk mengakomodasi fungsi dan layanan unit pemerintah .pembangunan Ibu Kota Negara (IKN) baru di Indonesia merupakan langkah strategis untuk mengatasi berbagai masalah di ibu kota lama, seperti kemacetan lalu lintas, kepadatan penduduk, dan kurangnya ruang terbuka hijau. IKN diharapkan menjadi solusi jangka panjang untuk menciptakan kota yang ideal bagi masa depan Indonesia. Desain IKN harus mencerminkan efisiensi penggunaan energi dan waktu, serta berkelanjutan untuk menciptakan lingkungan yang nyaman bagi semua penduduk.

Salah satu tantangan utama dalam pembangunan IKN adalah kompleksitas algoritma desain, yang harus mempertimbangkan penempatan bangunan dan infrastruktur secara efisien, desain transportasi yang terintegrasi, serta penyediaan ruang terbuka hijau yang memadai. Selain itu, efisiensi pembangunan menjadi krusial, mencakup optimalisasi penggunaan lahan, penerapan teknologi konstruksi canggih, dan rancangan infrastruktur yang mendukung logistik serta pergerakan manusia. Validasi dan verifikasi desain juga merupakan tantangan penting, dengan kebutuhan untuk melakukan pengujian teliti guna memastikan desain memenuhi semua aturan dan standar yang ditetapkan.

Dalam mewujudkan pembangunan IKN yang di inginkan maka diperlukan perencanaan arsitektur dan infrastruktur yang canggih serta terintegrasi. Perencanaan tata ruang harus membagi wilayah ke dalam zona-zona fungsional seperti pemerintahan, komersial, perumahan, dan ruang terbuka hijau, dengan konektivitas yang baik untuk memastikan mobilitas yang lancar. Desain bangunan pemerintahan, perumahan, dan fasilitas publik harus modern dan efisien, mencerminkan identitas nasional dan memenuhi kebutuhan penduduk. Infrastruktur transportasi yang mencakup jaringan jalan, transportasi publik harus dirancang untuk mengurangi kemacetan dan emisi karbon.

2. Solusi Permasalahan

Dengan memanfaatkan sumber sumberdaya yang ada, saya dan teman saya berhasil membangun map/peta yang bisa membuat jalur tanpa buntu dan letak bangunan tidak tumpeng tindih serta di setiap jalan terdapat rumah. Kami membuatnya sesuai aturan yang diberikan dan menjadikannya sebuah aplikasi Gambaran perencanaan design IKN yang dapat mengenerate map secara acak.

Berikut ini tahapan-tahapan penyelesaian proyek yang kami buat, sebagai berikut :

- **Pembuatan peta / map**

```
#Fungsi untuk membuat map baru, yang mana akan dieksekusi ketika tombol generate map ditekan
def createMap(self):
    self.simpang = [(0,0), (0,self.height), (self.width , 0) ,(self.width, self.height)]
    self.len = 0
    self.lastVertex = (random.randrange(0, self.width, self.road_len), random.choice([0, self.height]))
    self.lastVertex2 = (random.randrange(0, self.width, self.road_len), random.choice([0, self.height]))
    self.map = Image.new("RGBA",(self.width, self.height ), (100,100,100) )
    self.mapDraw = ImageDraw.Draw(self.map)
    self.makeRoads(self.lastVertex, "y")
    #Save map
    self.map.save("map1.png")
    self.mapping()
    #Save map2
    self.map.save("map2.png")
    return self.map
```

Gambar 1. Pembuatan Map menggunakan Brute Force

Metode **createMap** pada kode bertanggung jawab untuk menghasilkan peta yang kompleks dengan jaringan jalan yang terhubung dan berbagai elemen lainnya. Algoritma yang digunakan dimulai dengan inisialisasi variabel-variabel penting seperti skala, dimensi peta, panjang jalan, dan lainnya.

- **Pembuatan Jalan**

```
def makeRoads(self, pos, direction):
    self.len += 1
    nearX = pos[0]
    for sim in self.simpang:
        if (sim[0] > pos[0] - 20 and sim[0] < pos[0]+20):
            nearX = sim[0]
    pos = (nearX , pos[1])
    pos = (pos[0]//20 * 20, pos[1]//20 * 20)
    if self.len > 150 and (pos[0]<=0 or pos[0]>=self.width or pos[1] <= 0 or pos[1] >=self.height ): return
    self.simpang.append(pos)
    step = random.choice([1,1])
    nextvertice = ( pos[0] + self.road_width if direction == "y" else pos[0] + self.road_len  * step, pos[1] + s
    self.simpang.append(nextvertice)
    valid = [pos[0] <= 0 and direction == "y", pos[1] <=0 and direction == "x", nextvertice[1] >= self.height an
    nextvertice = (self.limitX(nextvertice[0]), self.limitY(nextvertice[1]))
    xsort, ysort = sort([pos[0], nextvertice[0]]), sort([pos[1], nextvertice[1]])
    if not sum(valid) :
        self.mapDraw.rectangle(((xsort[0], ysort[0]), (xsort[1] , ysort[1] )), "black")
        if direction == "y" : self.mapDraw.line(((xsort[0] + 10, ysort[0] + 10), ( xsort[0] +10 , ysort[1] - 10)
        else : self.mapDraw.line(((xsort[0] + 20 , ysort[0] + 10), ( xsort[1] -10 , ysort[0]+10 )),"white", 1)
        # self.map.paste(self.jalan[0] if direction == "x" else self.jalan[1], (xsort[0], ysort[0]))
    # if pos[0] >= 0 and pos[0] <= self.width and pos[1] >= 0 and pos[1] <= self.height:
    # self.simpang.append(nextvertice)
    nextX = self.width if nextvertice[0] <= 0 else (nextvertice[0] - (20 if direction == "y" else 0) if nextvert
    nextY = self.height if nextvertice[1] <= 0 else (nextvertice[1] - (20 if direction == 'x' else 0) if nextver
    self.makeRoads((nextX,nextY ),random.choice(["x", 'y']) )
```

Gambar 2. Pembuatan jalur jalan menggunakan metode Rekursif

Pada pembuatan jalan pada map IKN ini menggunakan **metode makeRoads** untuk menciptakan jaringan jalan di kota. Pada awalnya, beberapa variabel seperti **`self.len`** (panjang), **`self.width`** (lebar), dan **`self.height`** (tinggi) diatur. Ini membentuk dasar dari peta yang akan dibuat. Pada setiap langkah, kami memilih posisi untuk membuat persimpangan baru di peta. Posisi ini dipilih secara acak, namun dengan memperhitungkan persyaratan seperti jarak minimum dari tepi peta. Setelah memilih posisi persimpangan baru, koordinatnya ditambahkan ke daftar **`self.simpang`**, yang menyimpan koordinat semua persimpangan di peta.Selanjutnya, kita menentukan arah jalan berikutnya dari persimpangan saat ini. Ini dilakukan secara acak dengan memilih arah horizontal (x) atau vertikal (y).Setelah menentukan arah, kita memperpanjang jalan dari persimpangan saat ini ke arah yang dipilih. Ini dilakukan dengan menambahkan titik akhir dari jalan ke daftar **`self.simpang`** dan menggambar jalan di peta dari persimpangan saat ini ke titik akhir jalan.

- **Pembuatan dan penempatan Bangunan dan lain-lain**

```
def generateBuilding(self, pos):
    x = pos[0][0]
    #titik paling atas area
    def getBangunanX(x):
        return [bangunan for bangunan in self.buildings if bangunan.size[0] + x < pos[1][0]-self.padding]

    def getBangunanY(x,y):
        return [bangunan for bangunan in self.env if bangunan.size[0] + x < pos[1][0]-self.padding and bangunan.size

    while x < pos[1][0]:
        kandidat = getBangunanX(x)
        if len(kandidat):
            build = random.choice(kandidat)
            self.mapDraw.rectangle(((x, pos[0][1]), (x+build.size[0]+20, pos[0][1]+build.size[1])),"green")
            self.map.paste(build, (x, pos[0][1]))
            x += build.size[0] + self.jarak
        else: break
    x = pos[0][0]
    y = pos[0][1] + 50 + self.padding
    while  y < pos[1][1] - 50:
        tertinggi = 50
        while x < pos[1][0] :
            kandidat = getBangunanY(x, y)
            if len(kandidat):
                build = random.choice(kandidat)
                self.mapDraw.rectangle(((x,y), (x+build.size[0]+20, y+build.size[1])),"green")
                self.map.paste(build, (x, random.randint(y, y+(tertinggi - build.size[1]))))
                tertinggi = max(build.size[1], tertinggi)
                x += build.size[0] + self.padding
            else : break
        y += tertinggi + self.padding
    x = pos[0][0]

    #titik paling bawah area
    while x < pos[1][0]:
        kandidat = getBangunanX(x)
        if len(kandidat):
            build = random.choice(kandidat)
            self.mapDraw.rectangle(((x, pos[1][1]-build.size[1]), (x+build.size[0]+20, pos[1][1]-build.size[1]+build
            self.map.paste(build, (x, pos[1][1]-build.size[1]))
            x += build.size[0] + self.jarak
        else: break
```

Gambar 3. penempatan bangunan dan lain-lain menggunakan Algoritma Greedy

Setelah jaringan jalan selesai dibuat, bangunan dan elemen lingkungan ditambahkan ke peta.Metode **`generateBuilding`** di kelas **`map`** digunakan untuk menempatkan bangunanbangunan dan elemen lingkungan lainnya di sepanjang jalan-jalan. Pada pembuatan dan penempatan bangunan dan lain-lain dalam kode tersebut, digunakan serangkaian algoritma yang bertanggung jawab untuk menentukan lokasi, ukuran, serta tata letak bangunan dan elemen lingkungan dalam peta. Pertama-tama, algoritma menentukan lokasi bangunan dengan dua pendekatan: penempatan horizontal dan vertikal. Melalui proses ini, area yang tersedia untuk penempatan bangunan ditentukan, dan dari situ, bangunan yang sesuai dipilih secara acak dari kandidat yang telah disiapkan sebelumnya. Setelah seleksi, validasi dilakukan untuk memastikan bahwa bangunan tersebut muat di area yang telah ditetapkan, dan jika memungkinkan, bangunan tersebut ditempatkan sesuai dengan lokasi yang telah ditentukan.

Selanjutnya, algoritma juga bertanggung jawab untuk menambahkan elemen dekorasi atau lingkungan seperti pepohonan, semak-semak, atau batu-batuan. Proses ini dilakukan untuk menciptakan variasi dalam tata letak bangunan dan lingkungan di dalam peta, sehingga menghasilkan kesan kota yang hidup dan beragam.

- **Pemetaan**

```
def mapping(self):
    print(self.simpang)
    for titik in self.simpang:
        if titik[1] >= self.height : continue
        nearX,nearY = 0,0
        for titikTetangga in self.simpang:
            if titik != titikTetangga and titik[0] > 0 and titik[1] > 0:
                nearX = titikTetangga[0] if  (titikTetangga[0] > nearX and titikTetangga[0] < titik[0] and titik[1]
                nearY = titikTetangga[1] if  (titikTetangga[1] > nearY and titikTetangga[1] < titik[1]) else nearY
        # print("Nemu ", nearX , nearY)
        if titik[0] - nearX >= 30 and titik[1]-nearY >= 30:
            if (nearX, nearY) not in self.simpang: self.simpang.append((nearX, nearY))
            if (nearX, titik[1]) not in self.simpang: self.simpang.append((nearX - self.jarak, titik[1]))
            if (titik[0], nearY) not in self.simpang: self.simpang.append((titik[0], nearY))
            self.mapDraw.rectangle(((nearX+10,nearY+10), (titik[0]-10, titik[1]-10)), "green")
            self.generateBuilding(((nearX+10,nearY+10), (titik[0]-10, titik[1]-10)))
```

Gambar 4. Pemetaan menggunakan metode Iteratif

Algoritma yang digunakan untuk pemetaan (mapping) pada kodingan tersebut adalah sebuah algoritma iteratif yang mengatur penempatan bangunan dan elemen lingkungan di sekitar jalan-jalan yang telah dibuat sebelumnya. Proses pemetaan ini terjadi setelah pembuatan jalan-jalan selesai dilakukan. Penggambaran Peta :

- Setelah penempatan bangunan dan elemen lingkungan selesai dilakukan, peta yang telah dimodelkan secara virtual akan digambar pada objek gambar menggunakan modul `PIL`.
- Setiap bangunan dan elemen lingkungan digambar di posisi yang sesuai pada peta.

- **Generate**

```
#Fungsi yang dipanggil untuk mengupdate map dari tombol generate building
def update_map():
    global new_map, map_label
    new_map  = myMap.createMap()
    print(myMap.simpang)
    cropped_map = new_map.crop((viewport_x, viewport_y, viewport_x + viewport_width, viewport_y + viewport_height))
    resized_map = cropped_map.resize((INITIAL_WIDTH, INITIAL_HEIGHT))
    img_tk = ImageTk.PhotoImage(resized_map)
    map_label.config(image=img_tk)
    map_label.image = img_tk
    update()
```

Gambar 5. Mengenerate Map

Fungsi **update_map()** ini bertanggung jawab untuk memperbarui peta setiap kali tombol **"Generate Map"** ditekan. Prosesnya melibatkan pemanggilan metode **createMap()** dari objek myMap yang merupakan objek dari kelas **map**. Setelah peta baru dibuat, gambar peta diperoleh dengan memanggil metode **crop()** untuk memotong peta sesuai dengan area viewport yang ditampilkan di layar. Selanjutnya, gambar peta tersebut diubah ukurannya agar sesuai dengan ukuran layar GUI, dan kemudian dikonversi ke format yang dapat ditampilkan menggunakan **ImageTk**. Terakhir, gambar peta tersebut ditampilkan di label pada GUI.

- **Tampilan Output pada Program**

Gambar 5. Hasil Output Program

Untuk hasil program seperti gambar di atas, program tersebut menghasilkan peta kota dengan jaringan jalan yang terhubung, termasuk simpang atau persimpangan. Jalan-jalan memiliki bentuk dan arah yang beragam, menciptakan keragaman dalam struktur kota. Pada peta tersebut, tersebar bangunan-bangunan seperti rumah, gedung, dan elemen lingkungan seperti pepohonan, semak-semak, dan batu-batuan.Bangunan-bangunan ditempatkan di sekitar jalan-jalan, sedangkan elemen lingkungan dapat berada di sepanjang jalan atau di area terbuka. Pada program juga terdapat mengenerate map ulang serta Pengguna juga dapat memperbesar atau memperkecil tampilan peta