
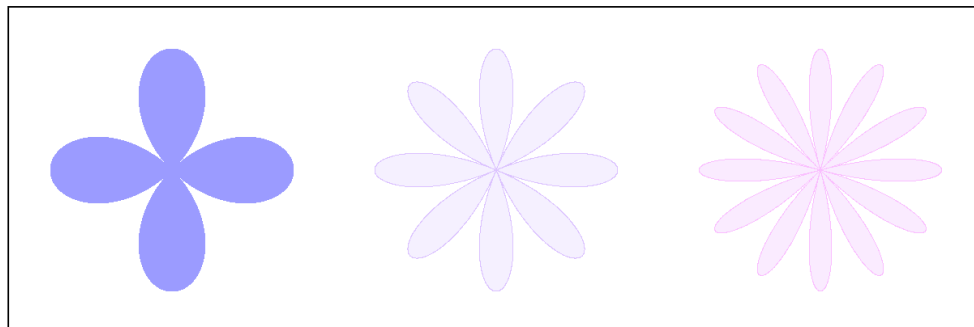


| | | | |
|--|---|------------------|---------|
|  <div>UNIVERSITAS KRISTEN MARANATHA</div> | NAMA PERGURUAN TINGGI : UNIVERSITAS KRISTEN MARANATHA NAMA FAKULTAS : Teknologi dan Rekayasa Cerdas NAMA JURUSAN / PRODI : S1 Teknik Informatika | | |
| LEMBAR KERJA MAHASISWA KE-4 | | | |
| MATA KULIAH | Grafika Komputer (Praktikum) | | |
| KODE | IN253 A | SKS | 3+1 SKS |
| DOSEN PENGAMPU | Sulaeman Santoso, S.Kom., M.T. Asisten Dosen : Yehezkiel David S. Nathan Joshua | Pertemuan | 4 |
| PETUNJUK PELAKSANAAN TUGAS : | | | |

Aturan Main :

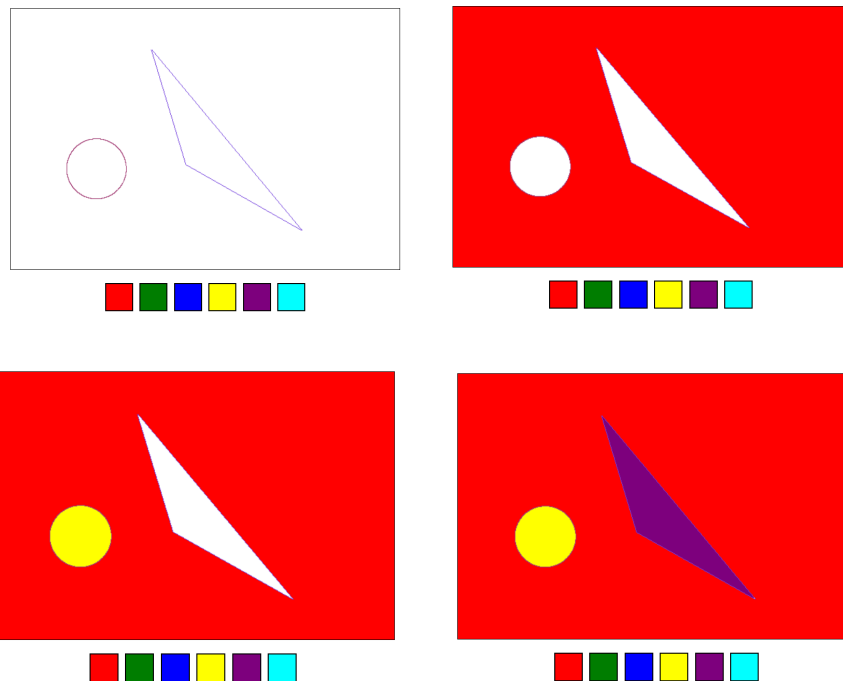
- Sertakan nama dan NRP pada setiap file yang anda buat
- Anda boleh bertanya pada rekan anda namun tidak boleh menyalin jawaban dalam bentuk apapun → Pelanggar akan diberikan sanksi
- Anda tidak diperkenankan menggunakan bantuan *Generative AI* apapun (ChatGPT, Claude, Gemini, dll).
- Setiap nomor, dibuat SATU FOLDER. Penamaan folder sesuai dengan penamaan nomor.

1. Buatlah seperti gambar berikut (T04A_NRP)



2. Paint Sederhana (T04B_NRP)

Anda diminta untuk menggambar objek acak (lingkaran dan poligon) pada canvas, dengan warna objek yang dihasilkan secara acak. Aplikasi harus mengimplementasikan algoritma flood fill berbasis stack, yang memungkinkan pengguna untuk mengisi warna area berdasarkan klik pada koordinat x dan y. Selain itu, sediakan palet warna sederhana yang memungkinkan pengguna memilih warna sebelum mengklik objek untuk menggantinya.



3. Pathfinding with A* (T04C_NRP)

Anda diminta membuat sebuah aplikasi visualisasi algoritma A* untuk mencari jalur terpendek dalam sebuah maze atau labirin. Aplikasi ini harus dapat menghasilkan maze secara acak menggunakan bentuk-bentuk poligon (persegi dan lingkaran) serta memungkinkan pengguna untuk menetapkan titik awal, tujuan, dan satu titik antara melalui klik pada canvas. Aplikasi harus menampilkan jalur terpendek yang ditemukan oleh algoritma A* dengan visualisasi yang jelas di atas maze. Selain itu, terdapat tombol untuk mereset canvas dan memulai ulang proses.

Generate Maze Find Path Reset Canvas

```
// Algoritma Pencarian Jalur A*
const heuristic = (a, b) => {
  return Math.abs(a.x - b.x) + Math.abs(a.y - b.y); // Menghitung jarak Manhattan
};

const aStar = (grid, start, goal) => {
  const rows = grid.length; // Jumlah baris
  const cols = grid[0].length; // Jumlah kolom

  let openSet = []; // Daftar titik yang akan diperiksa
  openSet.push(start); // Menambahkan titik awal ke dalam set

  let cameFrom = new Map(); // Peta untuk melacak jalur kembali
  let gScore = Array.from({ length: rows }, () => Array(cols).fill(Infinity)); // Skor jarak dari awal
```

```

let fScore = Array.from({ length: rows }, () => Array(cols).fill(Infinity)); // Skor total

gScore[start.x][start.y] = 0; // Skor jarak awal
fScore[start.x][start.y] = heuristic(start, goal); // Skor total awal

while (openSet.length > 0) {
  let current = openSet.sort((a, b) => fScore[a.x][a.y] - fScore[b.x][b.y]).shift(); //
  Mengambil titik dengan skor terendah

  if (current.x === goal.x && current.y === goal.y) {
    return reconstructPath(cameFrom, current); // Mengembalikan jalur jika mencapai
    tujuan
  }

  const neighbors = getNeighbors(grid, current); // Mendapatkan tetangga dari titik saat
  ini

  for (let neighbor of neighbors) {
    let tentativeGScore = gScore[current.x][current.y] + 1; // Menghitung skor jarak
    sementara

    if (tentativeGScore < gScore[neighbor.x][neighbor.y]) {
      // Jika jalur lebih baik
      cameFrom.set(`${neighbor.x},${neighbor.y}`, current); // Melacak dari mana titik ini
      berasal
      gScore[neighbor.x][neighbor.y] = tentativeGScore; // Memperbarui gScore
      fScore[neighbor.x][neighbor.y] = gScore[neighbor.x][neighbor.y] + heuristic(neighbor,
      goal); // Memperbarui fScore

      if (!openSet.some((el) => el.x === neighbor.x && el.y === neighbor.y)) {
        openSet.push(neighbor); // Menambahkan tetangga ke dalam openSet
      }
    }
  }
}

return 'No path found'; // Jika tidak ada jalur ditemukan
};

// Fungsi untuk membangun kembali jalur
const reconstructPath = (cameFrom, current) => {
  let path = [current]; // Memulai jalur dari titik saat ini
  while (cameFrom.has(`${current.x},${current.y}`)) {
    // Selama masih ada titik sebelumnya
    current = cameFrom.get(`${current.x},${current.y}`); // Menelusuri kembali
    path.push(current); // Menambahkan titik ke jalur
  }
  return path.reverse(); // Membalikkan jalur untuk mendapatkan urutan yang benar
};

```

```
// Mendapatkan tetangga dari titik saat ini
const getNeighbors = (grid, current) => {
  const deltas = [
    { x: -1, y: 0 }, // Atas
    { x: 1, y: 0 }, // Bawah
    { x: 0, y: -1 }, // Kiri
    { x: 0, y: 1 }, // Kanan
    { x: -1, y: -1 }, // Atas Kiri
    { x: 1, y: 1 }, // Bawah Kanan
    { x: -1, y: 1 }, // Atas Kanan
    { x: 1, y: -1 }, // Bawah Kiri
  ];

  let neighbors = []; // Array untuk menyimpan tetangga
  for (let delta of deltas) {
    let newX = current.x + delta.x; // Koordinat x tetangga
    let newY = current.y + delta.y; // Koordinat y tetangga

    // Memeriksa apakah tetangga berada dalam batas dan bukan dinding
    if (newX >= 0 && newX < grid.length && newY >= 0 && newY < grid[0].length &&
      grid[newX][newY] === 0) {
      neighbors.push({ x: newX, y: newY }); // Menambahkan tetangga yang valid
    }
  }
  return neighbors; // Mengembalikan daftar tetangga
};
```