LOVELY PROFESSIONAL UNIVERSITY

CSE-316 (Operating Systems)

CA-2 (Skill-Based)

Submitted By:

S no.	Name	Registration No.	Roll No.	Section
1.	Jyoti Kushwaha	12325612	43	KE056
2.	Ananya Gupta	12310219	28	KE056

Submitted To: Amandeep kaur

LOVELY PROFESSIONAL UNIVERSITY

Table of Contents:

1. Project Overview:

1.1 Title:

Graphical Interface for Garbage Collection Visualization.

1.2 Introduction:

Most common software relies on many functions, with memory management being an important part, Garbage collection automates this process further by identifying and retaining the blocks of memory that are not in use thus preventing memory leaks and increasing efficiency. The concept of garbage collection would be difficult to understand without proper visuals. This project works towards addressing this issue by providing a visual and interactive experience of memory allocation. Through this project user will have a good understanding of how garbage collection works in operating system by simulating how memory is allocated, used and collected over time.

1.3 Objective:

1.Allow users to visualize memory allocation and garbage collection.

2. Update memory usage continuously as user allocate and deallocate memory.

3. Allocate, garbage-collect and restart memory state.

4. Show recently deleted files.

5. Interactive learning to gain a deeper insight into garbage collection.

6. Module wise Breakdown:

6.1 User Interface Module:

- Handling layout, buttons and interactive elements.

- Charts for visualization of the processes and shows allocated and deallocated memory blocks.

6.2 Memory Allocation Module:

- Generated blocks of memory with random filename and size dynamically.
- New type of chart which updates memory based on inputs.

6.3 Garbage Collection Module:

- Traverses through memory blocks, finds unreferenced memory blocks and removes them simulating garbage collection.
- Updates deleted memory table.

6.4 Module for Chart and Data Visualization Module:

- Uses chart.js for visualizing memory usage and allocation.
- Graphs Memory Usage Schedule in a line graph and Memory Distribution in a pie chart.

3. Functionalities

3.1 Memory Allocation:

- Random filename and size memory blocks generator (for understanding how memory is filled in different scenarios) Users can analyse trends in allocation over time.

3.2 Garbage Collection Simulation:

- Releases blocks as required based on conditions and takes care of memory optimization. It mimics a real world automatic memory management.

3.3 Dynamic Chart Updates:

- Shows real time memory consumption status, therefore you can see how memory is allocated and garbage collected. No better way to visualize immediate memory usage than with charts.

3.4 Reset functionality:

- Free up all of the memory it has allocated, and return the charts back to initial states.

3.5: User Controlled Garbage Collection:

- Users can initiate garbage collection themselves which enables them to see the impact of memory deallocation immediately.

4. Technologies Used

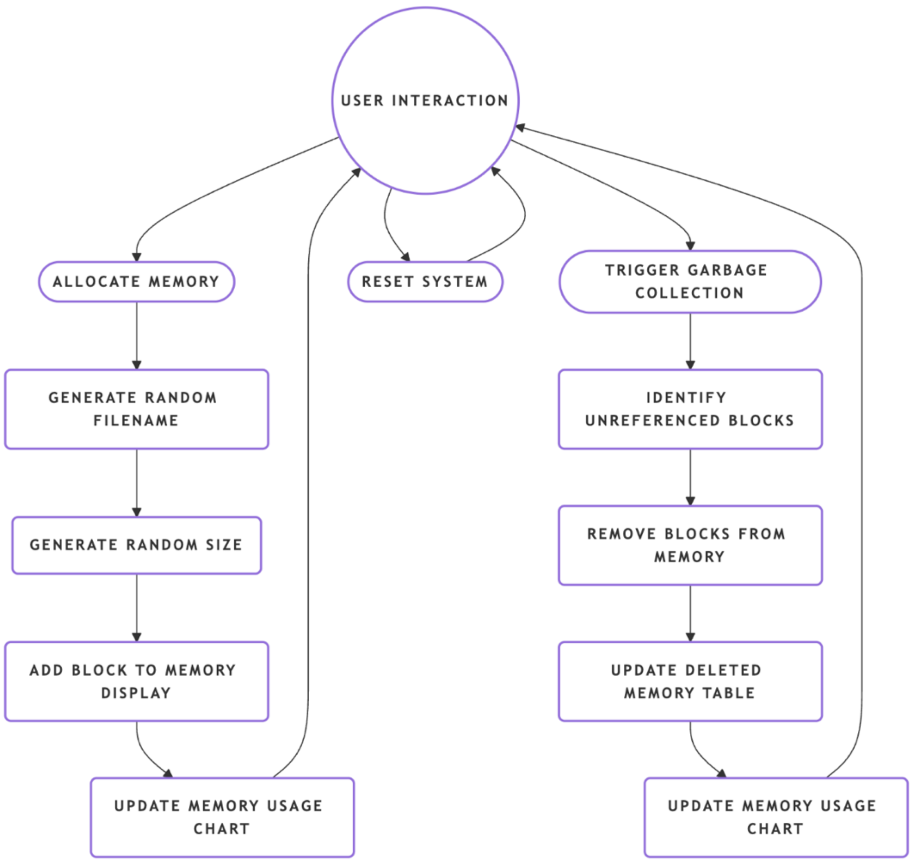
- HTML, CSS, JavaScript for front end development and interactive user interface elements.

- Chart.js for real time data visualization and graph rendering.

- DOM Manipulation for implementing dynamic memory updates.

- CSS Grid and Flexbox for responsive and structured design.

5. Flow Diagram


USER INTERACTION
ALLOCATE MEMORY
RESET SYSTEM
TRIGGER GARBAGE COLLECTION
GENERATE RANDOM FILENAME
GENERATE RANDOM SIZE
ADD BLOCK TO MEMORY DISPLAY
UPDATE MEMORY USAGE CHART
IDENTIFY UNREFERENCED BLOCKS
REMOVE BLOCKS FROM MEMORY
UPDATE DELETED MEMORY TABLE
UPDATE MEMORY USAGE CHART

1. The user interacts with the system, allocates memory, triggers garbage collection.

2. Memory allocation updates the UI and charts instantly.

3. Garbage collection takes deleted files and updates table.

4. Dynamic updates helps in tracking memory usage in real time.

6. Revision Tracking on GitHub

- Repository Name: garbage-collection-Visualizer

- GitHub Link: https://github.com/Kushwahajyoti/Garbage-Collection-Visualizer

7.Conclusion and Future Scope

Implementation this project represents the fundamentals of garbage collection and management of memory using graph and chart via graphical interface. User can see the memory allocation and de allocation, garbage collection and real time memory utilization. Dynamic charts enhance the overall learning experience.

This project is a good way to learn about how memory management and garbage collection work. Future scope may include:

Twin GC Algorithms: Mark-Sweep, Generational and G1 GC.

Custom allocation patterns: Provide users a way to define allocation behaviours.

AI Integration: Use AI to synchronize garbage collection with the usage patterns of the application.

The solution may be in a cloud-based implementation; it may provide for access and analysis of discrete data from remote locations.

8. References

1. Operating Systems Concepts – Silberschatz, Galvin, Gagne.

2. Modern Operating Systems – Andrew S Tanenbaum(Memory management chapters relevant to GC).

9. Appendix

A. <u>AI-Generated Project Elaboration/Breakdown Report:</u>

Here's a detailed breakdown of your project:

---

1. Project Overview

Goals

The project aims to create a Graphical User Interface (GUI) that visually demonstrates how garbage collection (GC) works in an operating system. It should simulate memory allocation, garbage collection cycles, and object lifecycle management, helping users understand different GC algorithms.

Expected Outcomes

- **A real-time, interactive GUI that displays memory blocks, allocated objects, and reclaimed memory.**
- **Support for multiple garbage collection techniques (e.g., reference counting, mark-and-sweep, generational GC).**
- **User controls to pause, restart, or step through the garbage collection process.**

Scope

- Focuses on visualization rather than actual memory management.
- Covers major GC algorithms used in OS and programming languages like Java and Python.
- Not implementing OS-level garbage collection but simulating it for educational purposes.

---

2. Module-Wise Breakdown

Module 1: GUI Module

Purpose: Provides an interactive interface for users to observe garbage collection in action.

Module 2: Simulation Engine

Purpose: Handles memory allocation, object lifecycle simulation, and garbage collection algorithms.

Module 3: Data Visualization & Analysis

Purpose: Displays memory usage trends, GC events, and statistics dynamically.

---

3. Functionalities

1. GUI Module Functionalities

- Memory Visualization: Represent heap memory as a grid/blocks.
- Controls: Play, Pause, Step-by-Step Execution.
- Algorithm Selection: Dropdown to choose different GC algorithms.

2. Simulation Engine Functionalities

- **Heap Allocation: Simulate memory blocks being allocated and freed.**
- **GC Algorithm Implementation: Simulate reference counting, mark-and-sweep, etc.**
- **Object Lifecycle Handling: Assign references, remove references, trigger GC.**

3. Data Visualization & Analysis Functionalities

- **Live Memory Usage Graph: Shows heap usage over time.**
- **GC Event Logs: Display actions like object creation, reference updates, and collection.**
- **Performance Metrics: Display execution time, memory reclaimed, objects collected.**

---

4. Technology Recommendations

Programming Language

- **Java (Recommended for native GC simulation) or Python (easier visualization).**

Libraries & Tools

- **GUI: Java Swing/JavaFX (Java), Tkinter/PyQt (Python).**
- **Graphics & Visualization: Java 2D API, JavaFX Canvas, Matplotlib (Python).**
- **Backend Logic: Custom algorithms in Java/Python.**

---

5. Execution Plan

Step 1: GUI Design (2 Weeks)

- **Create UI layout with memory visualization, controls, and logs.**

- **Implement basic interaction (e.g., buttons for Play, Pause).**

Step 2: Implement Simulation Engine (4 Weeks)

- **Implement basic object allocation/deallocation logic.**
- **Implement and integrate different GC algorithms.**

Step 3: Data Visualization (3 Weeks)

- **Implement memory usage graphs and GC event logs.**

Step 4: Testing & Optimization (2 Weeks)

- **Optimize performance, refine UI/UX, fix bugs.**

Step 5: Final Deployment & Documentation (1 Week)

- **Package project, create user guide, and finalize submission.**

B. Problem Statement: GUI for garbage collection in OS

C. Solution/Code:

HTML :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Garbage Collection Simulation</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <link rel="stylesheet" href="osfinal.css"> <!-- Updated from styles.css -->
</head>
```

```html
<body>
  <div class="container">
    <h1>Garbage Collection <span>Visualizer</span></h1>
    <div class="charts-container">
      <div class="chart-container">
        <h3>Memory Usage Over Time</h3>
        <canvas id="memoryChart"></canvas>
      </div>
      <div class="chart-container">
        <h3>Resource Allocation</h3>
        <canvas id="allocationChart"></canvas>
      </div>
    </div>
    <div>
      <div class="controls">
        <button id="resetBtn" class="btn reset">Reset</button>
        <button id="allocateBtn" class="btn allocate">Allocate</button>
        <button id="collectBtn" class="btn collect">Collect</button>
      </div>
      <div class="memory-usage">
        <h3>Memory Usage</h3>
        <div class="progress-bar">
          <div id="progress" class="progress"></div>
        </div>
        <span id="usage">139 / 300 MB</span>
      </div>
      <div class="memory-blocks">
        <div class="block">
          <p>document4.txt <span class="status">Referenced</span></p>
          <p>Size: 30 MB</p>
        </div>
        <div class="block">
          <p>image5.jpg <span class="status">Referenced</span></p>
          <p>Size: 20 MB</p>
        </div>
        <div class="block">
          <p>video7.mp4 <span class="status">Referenced</span></p>
          <p>Size: 36 MB</p>
        </div>
        <div class="block">
```

```
                <p>script11.js <span class="status">Referenced</span></p>
                <p>Size: 53 MB</p>
            </div>
        </div>
        <div class="deleted-memory">
            <h3>Deleted Memory Blocks</h3>
            <table>
                <thead>
                    <tr>
                        <th>Block ID</th>
                        <th>Size</th>
                    </tr>
                </thead>
                <tbody id="deletedTableBody"></tbody>
            </table>
        </div>
      </div>
  </div>
  <script src="osfinal.js"></script> <!-- Updated from script.js -->
</body>
</html>
```

CSS :

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: Arial, sans-serif;
}

body {
  background-color: #f5f7fa;
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

.container {
  background-color: #fff;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  width: 900px;
  display: grid;
  grid-template-columns: 1fr 2fr;
  gap: 20px;
}

.charts-container {
  display: flex;
  flex-direction: column;
  gap: 20px;
}

h1 {
  font-size: 24px;
  color: #2c3e50;
  margin-bottom: 15px;
  grid-column: span 2;
}

h1 span {
  font-weight: normal;
}

.controls {
  display: flex;
  justify-content: flex-end;
  margin: 10px 0;
```

```css
}

.btn {
   padding: 8px 16px;
   border: none;
   border-radius: 5px;
   color: #fff;
   font-size: 14px;
   cursor: pointer;
   margin-left: 10px;
}

.reset {
   background-color: #7f8c8d;
}

.allocate {
   background-color: #3498db;
}

.collect {
   background-color: #9b59b6;
}

.memory-usage {
   margin: 20px 0;
}

.memory-usage h3,
.chart-container h3,
.deleted-memory h3 {
   font-size: 16px;
   color: #2c3e50;
   margin-bottom: 10px;
}

.progress-bar {
   width: 100%;
   height: 10px;
   background-color: #ecf0f1;
```

```
    border-radius: 5px;
    overflow: hidden;
    margin: 10px 0;
}

.progress {
    width: 46%;
    height: 100%;
    background-color: #2ecc71;
    transition: width 0.3s ease;
}

.memory-usage span {
    display: block;
    text-align: right;
    font-size: 14px;
    color: #2c3e50;
}

.memory-blocks {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 10px;
    margin-top: 20px;
}

.block {
    background-color: #e8f5e9;
    border: 2px solid #2ecc71;
    border-radius: 5px;
    padding: 10px;
    text-align: center;
    transition: opacity 0.3s ease;
}

.block p {
    font-size: 14px;
    color: #2c3e50;
}
```

```
.status {
   color: #2ecc71;
   font-size: 12px;
}

.chart-container {
   height: 200px;
}

.deleted-memory {
   margin-top: 20px;
   max-height: 200px;
   overflow-y: auto;
}

table {
   width: 100%;
   border-collapse: collapse;
   font-size: 14px;
}

th,
td {
   padding: 8px;
   text-align: left;
   border-bottom: 1px solid #ecf0f1;
}

th {
   background-color: #2c3e50;
   color: #fff;
   cursor: pointer;
   user-select: none;
}

th:hover {
   background-color: #34495e;
}

tr:nth-child(even) {
    background-color: #f8f9fa;
}

tr:hover {
    background-color: #ecf0f1;
}
```

JAVASCRIPT :

```
const resetBtn = document.getElementById('resetBtn');
const allocateBtn = document.getElementById('allocateBtn');
const collectBtn = document.getElementById('collectBtn');
const progress = document.getElementById('progress');
const usageText = document.getElementById('usage');
const memoryBlocks = document.querySelector('.memory-blocks');
const deletedTableBody = document.getElementById('deletedTableBody');

let totalMemory = 300;
let usedMemory = 139;
let blockCount = 11;
let timeStep = 0;
let deletedBlocks = [];

// Lists of filenames and extensions
const filenames = [
    'document', 'image', 'video', 'audio', 'script',
    'data', 'config', 'archive', 'project', 'backup'
];
const extensions = ['.txt', '.jpg', '.mp4', '.js'];

// Function to generate a random filename with extension
function generateFilename() {
    const randomFilename = filenames[Math.floor(Math.random() *
filenames.length)];
    const randomExtension = extensions[Math.floor(Math.random() *
extensions.length)];
    return `${randomFilename}${blockCount}${randomExtension}`;
```

```
}

const memoryCtx =
document.getElementById('memoryChart').getContext('2d');
const memoryChart = new Chart(memoryCtx, {
   type: 'line',
   data: {
      labels: [0],
      datasets: [{
         label: 'Memory Usage (MB)',
         data: [usedMemory],
         borderColor: '#2ecc71',
         backgroundColor: 'rgba(46, 204, 113, 0.1)',
         fill: true,
         tension: 0.1
      }]
   },
   options: {
      responsive: true,
      maintainAspectRatio: false,
      scales: {
         x: { title: { display: true, text: 'Time' } },
         y: { min: 0, max: totalMemory, title: { display: true, text: 'Memory (MB)'
} }
      },
      plugins: {
         legend: { display: false },
         tooltip: { enabled: false }
      }
   }
});

const allocationCtx =
document.getElementById('allocationChart').getContext('2d');
const allocationChart = new Chart(allocationCtx, {
   type: 'pie',
   data: {
      labels: ['Used Memory', 'Free Memory'],
      datasets: [{
         data: [usedMemory, totalMemory - usedMemory],
            backgroundColor: ['#2ecc71', '#ecf0f1'],
            borderColor: ['#27ae60', '#d5dbdb'],
            borderWidth: 1,
            hoverOffset: 0,
            hoverBackgroundColor: ['#2ecc71', '#ecf0f1'],
            hoverBorderColor: ['#27ae60', '#d5dbdb']
        }]
    },
    options: {
        responsive: true,
        maintainAspectRatio: false,
        plugins: {
            legend: { position: 'bottom' },
            tooltip: { enabled: false }
        }
    }
});

function updateMemoryUsage() {
    const percentage = (usedMemory / totalMemory) * 100;
    progress.style.width = `${percentage}%`;
    usageText.textContent = `${usedMemory} / ${totalMemory} MB`;

    timeStep++;
    memoryChart.data.labels.push(timeStep);
    memoryChart.data.datasets[0].data.push(usedMemory);

    if (memoryChart.data.labels.length > 20) {
        memoryChart.data.labels.shift();
        memoryChart.data.datasets[0].data.shift();
    }

    allocationChart.data.datasets[0].data = [usedMemory, totalMemory -
usedMemory];

    memoryChart.update();
    allocationChart.update();
}

function updateDeletedTable() {
    deletedTableBody.innerHTML = '';
    deletedBlocks.forEach(block => {
        const row = document.createElement('tr');
        row.innerHTML = `
            <td>${block.id}</td>
            <td>${block.size} MB</td>
        `;
        deletedTableBody.appendChild(row);
    });
}

function sortTable(column) {
    deletedBlocks.sort((a, b) => {
        if (column === 0) return a.id.localeCompare(b.id);
        if (column === 1) return a.size - b.size;
    });
    updateDeletedTable();
}

resetBtn.addEventListener('click', () => {
    usedMemory = 0;
    memoryBlocks.innerHTML = '';
    deletedBlocks = [];
    blockCount = 0;
    timeStep = 0;
    memoryChart.data.labels = [0];
    memoryChart.data.datasets[0].data = [0];
    updateMemoryUsage();
    updateDeletedTable();
});

allocateBtn.addEventListener('click', () => {
    const size = Math.floor(Math.random() * 50) + 10;
    if (usedMemory + size > totalMemory) {
        alert('Not enough memory to allocate!');
        return;
    }

    blockCount++;
    usedMemory += size;
    const filename = generateFilename();
    const block = document.createElement('div');
    block.classList.add('block');
    block.innerHTML = `
      <p>${filename} <span class="status">Referenced</span></p>
      <p>Size: ${size} MB</p>
    `;
    memoryBlocks.appendChild(block);
    updateMemoryUsage();
});

collectBtn.addEventListener('click', () => {
    const blocks = memoryBlocks.querySelectorAll('.block');
    let removedMemory = 0;

    blocks.forEach((block, index) => {
      if (index % 2 === 0) {
        const blockText = block.querySelector('p:first-child').textContent;
        const blockId = blockText.split(' ')[0]; // Extract the filename (before
"Referenced")
        const sizeText = block.querySelector('p:last-child').textContent;
        const size = parseInt(sizeText.match(/\d+/)[0]);
        removedMemory += size;

        block.style.opacity = '0';
        setTimeout(() => block.remove(), 300);

        deletedBlocks.unshift({
          id: blockId,
          size: size
        });
      }
    });

    usedMemory -= removedMemory;
    updateMemoryUsage();
    updateDeletedTable();
});
```

```
document.querySelectorAll('th').forEach((th, index) => {
   th.addEventListener('click', () => sortTable(index));
});

updateMemoryUsage();
```