

CSE-316 (Operating Systems)

CA-2 (Skill-Based)

**Submitted By:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S no. | Name | Registration No. | Roll No. | Section |
| 1. | Ragini Yadav | 12311227 | 30 | K23GR |
| 2. | Tanvi | 12315499 | 08 | K23GR |
| 3. | Priya Rani | 12304538 | 41 | K23GR |

**Submitted To: Dr. Nahida Nazir**

**LOVELY PROFESSIONAL UNIVERSITY**

**Table of Contents:**

|  |  |  |  |
| --- | --- | --- | --- |
| S.no | Title | Page no. | |
| 1. | Declaration | 3 | |
| 2. | Acknowledgement | 4 | |
| 3. | Abstract | 5 | |
| 4. | Project overview | 6 | |
| 5. | Module wise Breakdown | 7 | |
| 6. | Functionalities | 8 | |
| 7. | Literature Overview | 10 | |
| 8. | Problem Statement | 11 | |
| 9. | Implementation & Development | 12 | |
| 10. | Screenshot of output | 14 | |
| 11. | Result & Analysis | 17 | |
| 12. | Challenges & Limitations | 18 | |
| 13. | Flow Diagram | 19 | |
| 14. | Conclusion and Future Scope | 20 | |
| 15. | References | | 21 |
|  |  | |  |

**1. Declaration**

We hereby declare that the project titled **“Graphical Interface for Garbage Collection Visualization”** submitted for CSE-316 (Operating Systems) - CA-2 (Skill-Based) is our original work. It has not been copied from any source or previously submitted for any other course or assignment.

All resources and references used during the development of this project have been duly acknowledged and cited in the report.

**2. Acknowledgement**

We would like to express our heartfelt gratitude to our supervisor, **Dr. Nahida Nazir**, for her valuable guidance, consistent support, and encouragement throughout the duration of our project, **“Graphical Interface for Garbage Collection Visualization.”** Her expert insights and timely feedback were instrumental in helping us successfully complete this work.

We also extend our thanks to the **Department of Computer Science and Engineering, Lovely Professional University**, for providing us with the necessary resources and a conducive environment to work on this project.

**Ragini Yadav**  
**Tanvi**  
**Priya Rani**

**3. Abstract**

Memory management is a critical function of operating systems, and garbage collection plays a key role in ensuring efficient utilization of memory by automatically reclaiming unused memory blocks. However, understanding garbage collection processes can be difficult without proper visualization. This project, titled **“Graphical Interface for Garbage Collection Visualization,”** aims to provide a real-time simulation that helps users grasp how memory is allocated, used, and cleaned up.

The objective is to build an interactive, web-based tool that visually demonstrates memory allocation and garbage collection. Technologies used include HTML, CSS, JavaScript, and Chart.js for real-time data visualization. The system allows users to allocate memory blocks with random sizes and filenames, visualize memory usage through charts, and trigger garbage collection manually. Deleted memory blocks are displayed in a tabular format for better analysis.

The project successfully simulates the garbage collection cycle, offering an intuitive and educational experience. It enables users to understand how memory fills up, how unused memory is identified, and how it is reclaimed. This tool enhances learning by making abstract concepts more tangible and interactive.

Future improvements may include integrating different garbage collection algorithms and AI-based memory optimization to make the simulation even more comprehensive and insightful.

1. **Project Overview:**

Title**:**

Graphical Interface for Garbage Collection Visualization.

Introduction**:**

Overview of the problem:

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.

Objective:

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

1. Update memory usage continuously as user allocate and deallocate memory.
2. Allocate, garbage-collect and restart memory state.
3. Show recently deleted files.
4. Interactive learning to gain a deeper insight into garbage collection.

**5. Module wise Breakdown:**

User Interface Module:

* Handling layout, buttons and interactive elements.
* Charts for visualization of the processes and shows allocated and deallocated memory blocks.

Memory Allocation Module:

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

Garbage Collection Module:

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

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.

**6. Functionalities**

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.

Garbage Collection Simulation:

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

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.

Reset functionality:

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

User Controlled Garbage Collection:

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

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.

1. **Literature Overview**

Garbage collection (GC) is an essential aspect of automatic memory management in modern programming languages and operating systems. It aims to reclaim memory occupied by objects no longer in use, thus preventing memory leaks and optimizing resource utilization. Several works in literature and real-world tools have focused on improving or visualizing GC processes.

Existing tools like JVM VisualVM and Eclipse Memory Analyzer provide graphical memory analysis but are typically suited for advanced users and specific environments like Java. These tools, while powerful, focus on debugging and profiling rather than teaching the fundamental concepts in an intuitive manner.

In contrast, our project focuses on educational visualization. Unlike traditional GC simulators or debuggers, our interface is interactive, beginner-friendly, and designed for academic learning. It offers simplified memory block simulations and real-time charts using lightweight technologies like JavaScript and Chart.js.

This comparative advantage makes our project more suitable for instructional use, helping students understand core GC concepts visually before engaging with complex real-world tools or underlying implementations.

1. **Problem Statement**

Definition:

Understanding the internal working of garbage collection and memory management in operating systems is often abstract and difficult for students, especially without visual support. Traditional learning methods rely heavily on theoretical explanations, which can limit practical comprehension. There is a lack of simple, interactive tools that simulate and visualize how memory is allocated, used, and reclaimed in real time.

This project addresses the need for an educational tool that **visually simulates garbage collection** processes, helping learners observe the dynamic nature of memory operations and understand core concepts more effectively.

Scope and Limitation

This Project 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. It allows users to interactively allocate memory, trigger garbage collection, and observe memory changes in real-time. However, it does not implement actual operating system-level memory management and is limited to basic garbage collection logic. Advanced techniques like generational GC are not included, and the tool is intended purely for conceptual understanding rather than technical analysis.

1. **Implementation & Development**

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

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>

<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;

}

.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();

});

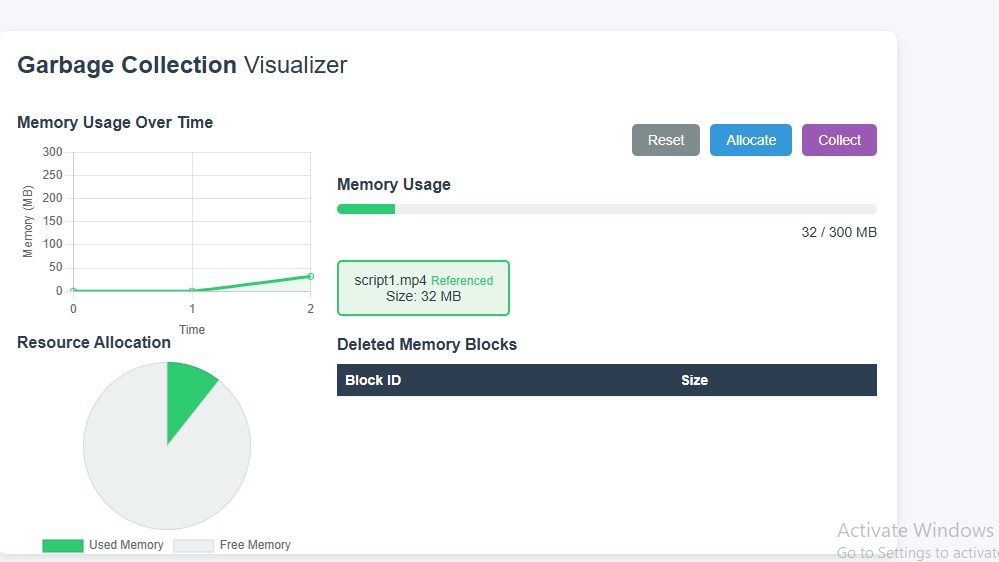
document.querySelectorAll('th').forEach((th, index) => {

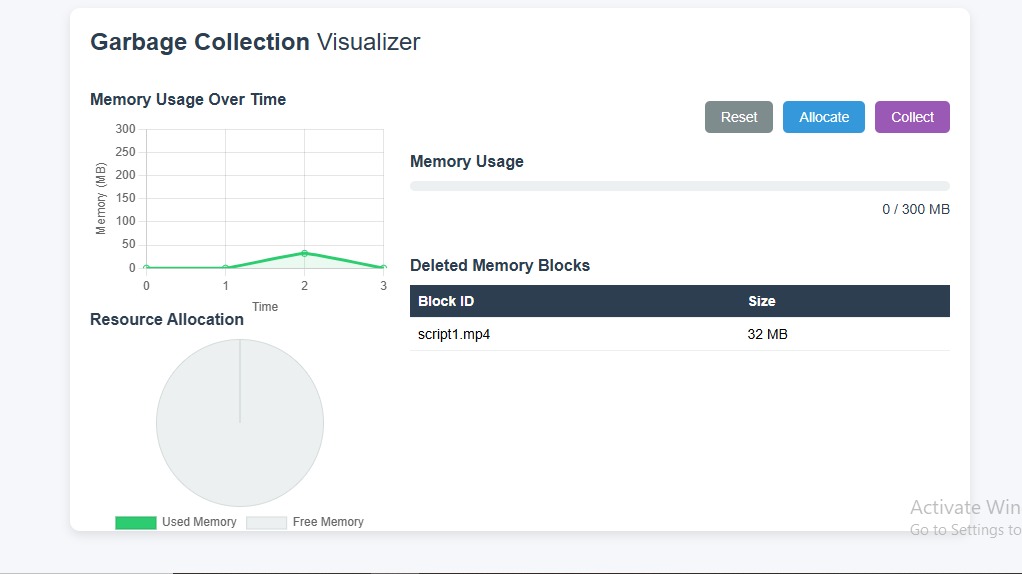
th.addEventListener('click', () => sortTable(index));

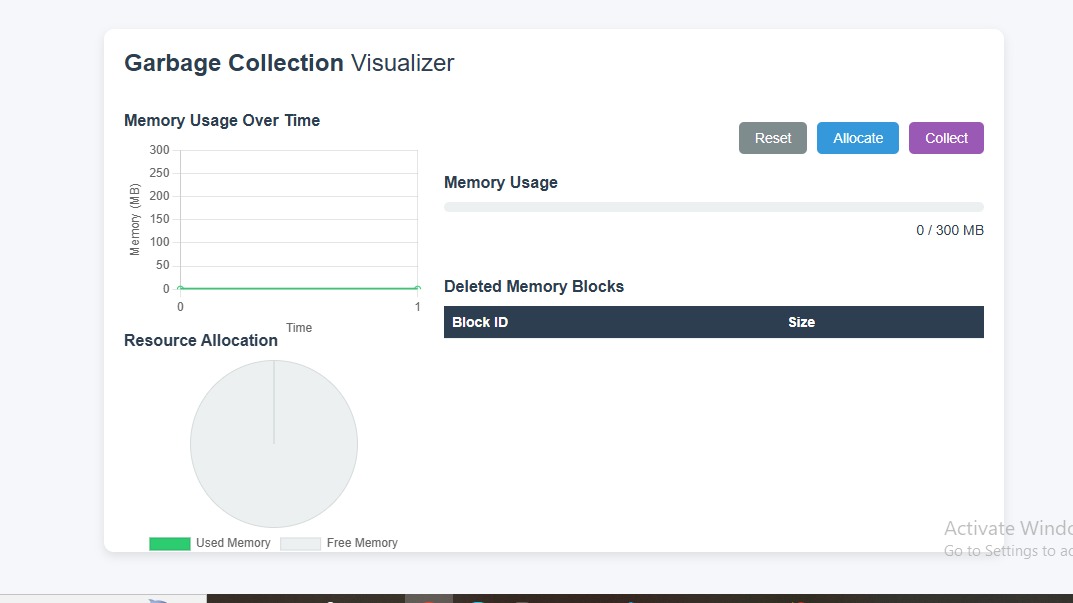
});

updateMemoryUsage();

1. **Screenshot Of Output**

****

****

****

11.**Result & Analysis**

The developed simulation successfully demonstrates dynamic memory allocation and garbage collection through an interactive web interface. Users can allocate memory blocks of varying sizes, observe memory usage in real-time, and manually trigger garbage collection to reclaim unused space. The system uses line and pie charts to reflect memory consumption trends, offering a clear visual understanding of memory distribution and optimization.

**Performance Evaluation:**

The tool performs efficiently within browser environments, handling multiple memory operations without lag. Charts update instantly as memory is allocated or collected, maintaining a smooth and responsive user experience. The simulation remains stable even with continuous allocation and garbage collection cycles.

**Case Study – User-Defined Input Example:**

* A user repeatedly allocates memory blocks of 20–50 MB until near capacity.
* The memory usage bar and line chart increase progressively.
* Upon triggering garbage collection, unreferenced blocks are cleared, reflected instantly in the charts and deleted memory table.
* Memory usage returns to a lower value, and the pie chart adjusts accordingly.

This simple scenario effectively illustrates how memory fills up and how garbage collection reclaims space.

**12. Challenges & Limitations**

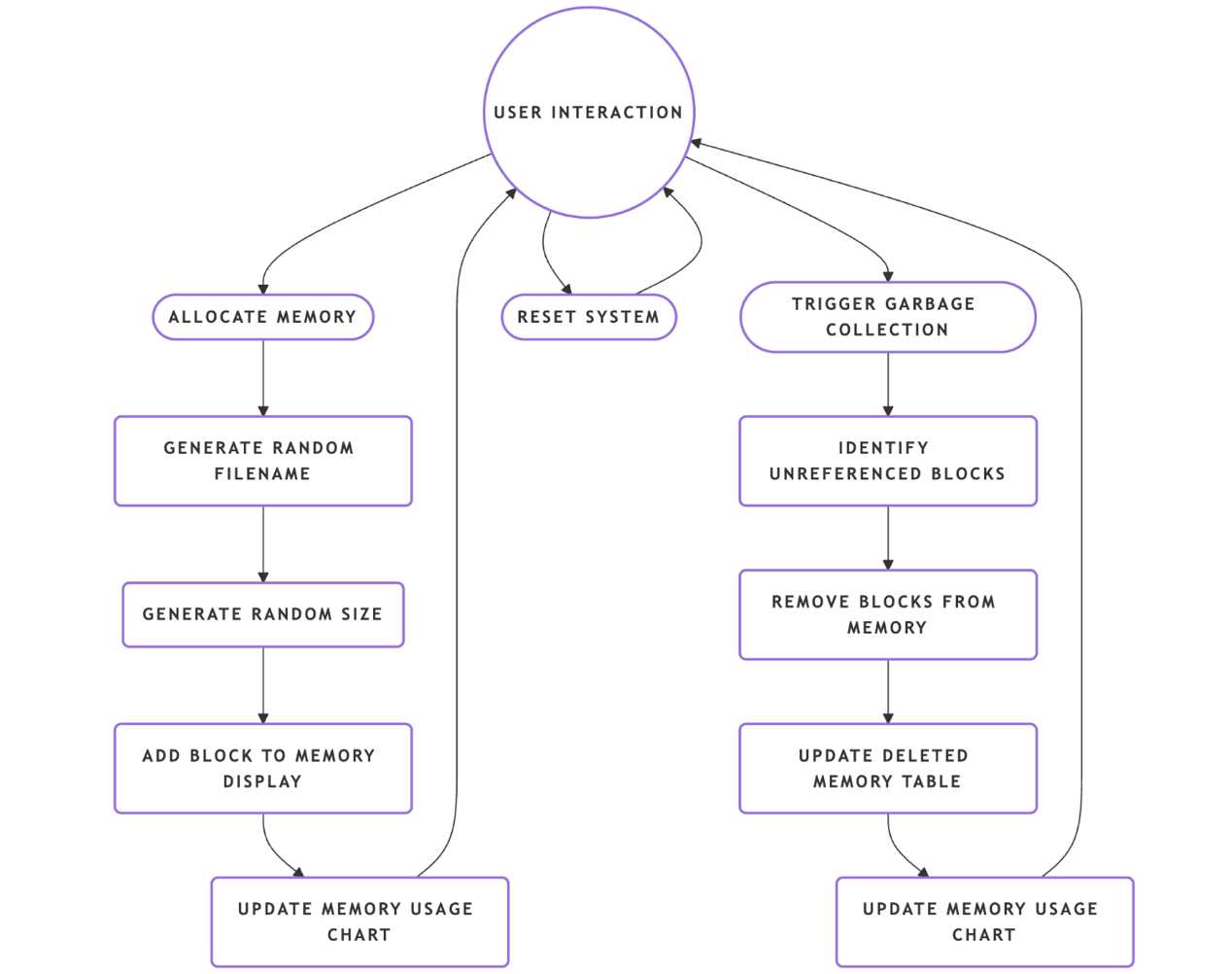
Challenges Faced During Development:

During the development of this project, one major challenge was synchronizing real-time chart updates with memory operations while maintaining performance in the browser. Ensuring that memory blocks updated dynamically without delays required precise DOM manipulation and efficient JavaScript event handling. Another challenge was simulating realistic garbage collection behavior, such as identifying unreferenced memory blocks, without overcomplicating the logic. Designing an interface that was both informative and user-friendly also involved multiple iterations and usability testing.

Limitations and Potential Improvements:

The current system simulates garbage collection based on fixed patterns and user triggers, rather than implementing sophisticated algorithms like mark-and-sweep or generational GC. It does not integrate with actual system memory or programming language runtimes, limiting its realism. Future improvements could include support for different garbage collection strategies, algorithm comparisons, and step-by-step execution of memory events. Additionally, implementing AI-based memory behavior prediction and offering downloadable logs or reports could enhance the learning experience.

**13. Flow Diagram**



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.

**14. 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.

**15. References**

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

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

**16 . GitHub Link**

[**https://github.com/Tanvi67/garbage-collection-visualizer**](https://github.com/Tanvi67/garbage-collection-visualizer)