# A black and red text Description automatically generated

**PROJECT AND TEAM INFORMATION**

## Project Title

|  |
| --- |
| DYNAMIC MEOMORY ALLOCATOR(HEAP MANAGEMENT SYSTEM) |

## Student/Team Information

|  |  |
| --- | --- |
| Team Name:  Team # | Memory Minds  DAA-IV-T032 |
| Team member 1 (Team Lead) | Gupta,Riya-23022643  [23022643@geu.ac.in](mailto:23022643@geu.ac.in) |
| Team member 2  Team member 3 | Dwivedi,Shubh-230112175  [230112175@geu.ac.in](mailto:230112175@geu.ac.in)    Kumari,Bhumi-23022707  [23022707@geu.ac.in](mailto:23022707@geu.ac.in) |

**PROJECT PROGRESS DESCRIPTION**

## Project Abstract

|  |
| --- |
| **SmartAlloc** is a dynamic memory management simulator in Python, inspired by operating system-level memory allocation techniques. It supports **First-Fit, Best-Fit, Next-Fit, Buddy Allocation, Paging (via page-size simulation), and Smart Allocation** (which auto-selects the optimal strategy based on performance).  Users interact via a **GUI built using Tkinter**, enabling allocation, deallocation, fragmentation analysis, compaction, and performance tracking. A **real-time memory map** visualizes blocks, color-coded by strategy. Additionally, performance is analyzed through **efficiency metrics and graphs** (using Matplotlib), making this a powerful educational and simulation tool. |

## Updated Project Approach and Architecture

|  |
| --- |
| The system architecture is built on a **doubly linked list** structure where each node represents a memory block with metadata such as size, start address, status (free or allocated), and allocation strategy. The simulator supports First-Fit, Best-Fit, Next-Fit, Buddy System, and a Smart Allocation mode that benchmarks all strategies and chooses the most efficient one. Memory operations are modular, ensuring clean separation of logic for each strategy. To simulate real OS behavior, the system also includes features like memory deallocation, fragmentation analysis, and compaction.  A **Tkinter-based GUI** was developed to enhance usability, allowing users to input allocation sizes, deallocate memory, view fragmentation stats, and trigger compaction visually. The memory is displayed as color-coded blocks in a canvas for intuitive understanding. Additionally, the system uses performance tracking metrics such as average steps and allocation time, which are visualized through Matplotlib graphs. This design combines **educational clarity with functional depth**, making it a robust simulation tool for learning and evaluating memory allocation strategies. |

## 

## 

## Tasks Completed

|  |  |
| --- | --- |
| Task Completed | Team Member |
| Implemented First-Fit, Best-Fit, Next-Fit strategies  Developed Smart Allocation logic  Developed Buddy Allocation system  Paging-based simulation (64-byte pages)  Deallocation and merging adjacent blocks  Memory compaction module  Fragmentation and efficiency reporting  GUI for memory operations and visualization  Efficiency comparison graph (matplotlib)  Final test case coverage & demo-ready interface | Riya Gupta  Riya Gupta  Shubh Dwivedi  Shubh Dwivedi  Bhumi Kumari  Bhumi Kumari  All Members  Riya Gupta  Shubh Dwivedi  All members |

## 

## Challenges/Roadblocks

|  |
| --- |
| One of the primary challenges was maintaining the **integrity of memory blocks** during allocation and deallocation. Improper pointer updates led to segmentation faults and memory corruption. This required thorough debugging and validation of the linked list structure.  Implementing the **Buddy Allocation** system introduced complexity due to its recursive nature. Ensuring that all buddy blocks remained power-of-two aligned while properly splitting and merging them took significant testing and logical refinement.  Designing the **Smart Allocation strategy** was another hurdle. It involved benchmarking each strategy on the fly, using cloned memory states, and selecting the most efficient one. Balancing correctness with execution time was crucial to its effectiveness.  Handling **user inputs** in the GUI posed reliability issues. Users could input invalid sizes or addresses, leading to crashes. This was addressed by adding robust validation and informative error messages to guide user interaction.  Integrating **memory compaction** without disrupting ongoing allocations required careful management of memory block addresses and re-linking blocks. Ensuring the stability of the updated memory map after compaction was a delicate process.  Analyzing **fragmentation** accurately—especially distinguishing internal from external fragmentation—was difficult after compaction. The solution involved iterating through memory carefully and classifying each block based on status and alignment.  Finally, creating a **real-time memory visualization and performance graph** added complexity. The GUI needed to dynamically represent memory states, and plotting performance metrics required synchronizing data collection and rendering without affecting core operations. |

## Future Scope

|  |
| --- |
| The SmartAlloc simulator can be extended to support **real-time memory usage logging and analysis**, enabling users to export session data and review allocation patterns over time. Additional strategies like **slab allocation, segregated fits, or garbage collection simulation** could be integrated for advanced system-level comparisons. Incorporating multi-threaded allocation and synchronization models would make the tool suitable for concurrent systems education and research.  Furthermore, the GUI can be enhanced with **interactive memory manipulation**, undo-redo functionality, and more detailed visual analytics like heatmaps for usage frequency. Integration with web technologies (e.g., converting to a browser-based simulator using frameworks like Flask or React) could increase accessibility and allow for **collaborative simulation environments**, especially for educational institutions. |

## 

## Project Outcome/Deliverables

|  |
| --- |
| The project successfully delivers a fully functional **dynamic memory allocation simulator** with support for multiple allocation strategies including First-Fit, Best-Fit, Next-Fit, Buddy Allocation, and Smart Allocation. Users can allocate and deallocate memory, compact memory, and analyze fragmentation using an interactive **Tkinter GUI**. The system also includes detailed performance tracking through metrics like allocation steps and time, offering a comprehensive view of each strategy’s efficiency.  Additional deliverables include a **real-time memory visualization panel**, robust input validation, and a **graphical comparison of strategy performance** using Matplotlib. The simulator is designed as both a learning tool and a testing platform for operating system concepts. With all core functionalities implemented and tested, the project meets its academic and functional goals, providing a user-friendly and insightful simulation environment. |

# Progress Overview

|  |
| --- |
| The project has been fully completed with all planned core and extended features successfully implemented. All six memory allocation strategies—First-Fit, Best-Fit, Next-Fit, Buddy Allocation, Paging (simulated via page size), and Smart Allocation—have been developed, tested, and integrated into a unified system. Deallocation, compaction, fragmentation analysis, and efficiency tracking are also fully functional and accessible through a user-friendly Tkinter-based GUI.  In addition to the core functionalities, enhancements such as **real-time memory visualization**, **performance comparison graphs**, and **robust input handling** have been added to improve usability and presentation. The system has passed all test cases, including edge cases and stress scenarios, making it stable and reliable. The project is now fully ready for demonstration and academic submission, having achieved all proposed objectives and deliverables. |

# Codebase Information

|  |
| --- |
| The codebase is primarily developed in **Python**, with two main files: one for the **core memory allocation logic and GUI (riya.py)**, and the other for **graphical performance comparison (graph.py)**. The core logic uses a class-based structure with modular functions for each allocation strategy, deallocation, compaction, fragmentation analysis, and performance tracking. The MemoryAllocator class encapsulates all memory management functionality, while the MemoryGUI class handles the Tkinter-based interface, ensuring a clean separation of logic and presentation.  Each strategy is implemented with performance benchmarking using time.time() and step counters to track efficiency. Memory is visualized on a canvas, with blocks color-coded by allocation strategy. Additionally, a **Matplotlib-based graph module** presents a visual comparison of average steps and time per strategy. The code is **well-commented, readable, and structured for extensibility**, making it suitable for academic use and further development. Version control and collaboration were managed using GitHub, where the complete codebase is hosted.  GitHub Link-  https://github.com/shubh9125/Dynamic\_Memory\_Allocator\_-Heap\_Management\_System- |

## 

## Testing and Validation Status

|  |  |  |
| --- | --- | --- |
| Test Type | Status (Pass/Fail) | Notes |
| First-Fit, Best-Fit,Next-fit  Buddy Allocation test  Paging Allocation  Deallocation and merging  Compaction test  Fragmentation analysis  Smart alloaction decision  GUI Operation and Input Handling  Performance graph (Matplotlib) | Pass  Pass  Pass  Pass  Pass  Pass  Pass  Pass  Pass | All strategies allocate correctly with minimal fragmentation.  Properly performs recursive splitting and follows power-of-two logic.  Pages are allocated correctly; fragmentation is within expected limits.  Adjacent free blocks are correctly merged, ensuring memory reuse.  Used blocks are shifted left, and a new free block is inserted properly.  Accurately distinguishes between internal and external fragmentation.  Benchmarks strategies and selects the most efficient based on timing.  Responsive GUI with validated inputs and real-time updates.  Displays accurate average time and steps per strategy in a combined chart. |

# Deliverables Progress

|  |
| --- |
| All core deliverables have been completed, including six memory allocation strategies (First-Fit, Best-Fit, Next-Fit, Buddy System, Paging, and Smart Allocation), deallocation with block merging, compaction, fragmentation analysis, and efficiency tracking. The Smart Allocation strategy accurately selects the most optimal method based on benchmarking, and all features are accessible via a user-friendly Tkinter GUI.  Additional enhancements such as **real-time memory visualization**, **robust input validation**, and a **performance graph using Matplotlib** have also been successfully implemented. These features improve usability and clarity, especially for educational demonstrations. The system has been thoroughly tested and is fully functional, making it ready for final submission and practical use. |