

**DOKUZ EYLUL UNIVERSITY**

**ENGINEERING FACULTY**

**DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING**

EED 1010

EED 1010 ALGORITHMS & PROGRAMMING FINAL REPORT PROJECT

BIG CITY

Group-29

by

Umut Efe AL & 2024502014

Oğuz Efe GÜNEŞ & 2024502036

Nurettin YILDIZ & 2023502188

Aleyna DOĞRUER & 2023502228

**Lecturer**

Ögr. Gör. Dr. Özlem ÖZTÜRK

IZMIR

Date Submitted : 29.05.2025

Contents

[1. CHAPTER: INTRODUCTION 3](#_Toc198824990)

[1.1 Project Objectives 3](#_Toc198824991)

[1.2 Progress Description 5](#_Toc198824992)

[2.CHAPTER: TASK SUMMARY 6](#_Toc198824993)

[2.1 Completed Task 6](#_Toc198824994)

[2.2 Additional Improvements 7](#_Toc198824995)

[3.CHAPTER: PROGRESS OVERVIEW 8](#_Toc198824996)

[3.1 Functions 8](#_Toc198824997)

[1. Passenger & Queue Management 8](#_Toc198824998)

[2. Bus Stop System 9](#_Toc198824999)

[3. Bus System 9](#_Toc198825000)

[4. Luggage System 10](#_Toc198825001)

[5. User Interaction 10](#_Toc198825002)

[6. Statistics 10](#_Toc198825003)

[3.2 Solution Strategies of Algorithms 10](#_Toc198825004)

[4.CHAPTER: PROBLEMS ENCOUNTERED 13](#_Toc198825005)

[5.CHAPTER: FUTURE WORK 15](#_Toc198825006)

[6.CHAPTER: CONCLUSION 16](#_Toc198825007)

[References 17](#_Toc198825008)

[Appendices 17](#_Toc198825009)

# 1. CHAPTER: INTRODUCTION

## 1.1Project Objectives

The "Big City" project aims to develop an application that simulates the intricate public transportation system of a large urban environment. The simulation focuses on the interactions between bus lines, buses, and passengers.

Simulation objectives listed below:

**Passengers**

* Each passenger has a starting point and a destination.
* Passengers may carry up to 2 pieces of luggage.
* Every passenger selects one or more bus lines to use.
* When a bus on a selected line arrives at the passenger’s location, the passenger boards it.
* Upon reaching their destination stop, the passenger exits the simulation.

**Passenger Queue System**

* A queue system holds up to 15 passengers waiting to enter the simulation.
* Each passenger in the queue has a randomly assigned waiting time.
* Once a passenger’s waiting time expires, they enter the simulation.
* As soon as a passenger leaves the queue and enters the simulation, a new passenger is generated with:
  + A random ID
  + A randomly chosen bus line
  + Random starting and destination bus stops
  + A random amount of luggage (0–2)
* The newly created passenger is added to the queue, maintaining a continuous flow of new participants.

**Buses**

* Each bus has a maximum luggage capacity of 8.
* Luggage is managed using a stack structure—only the topmost luggage item can be accessed at any time.
* Buses pick up and drop off passengers and their luggage as they move along their designated lines.

**Simulation System**

* The simulation runs on a discrete-time system, where each unit of simulation time corresponds to either 0.5 or 1 second in real time.
* User Controls:
  + Press 'R' to run the simulation.
  + Press 'Space' to pause the simulation.

**User Interface**

* The UI allows users to inspect individual buses in real-time.
* Hovering the mouse cursor over a bus displays detailed information on the right side of the screen, including:
  + A list of passengers currently on the bus
  + Details about each passenger’s luggage

## 1.2 Progress Description

**Week 1: Research and Planning**

* Discussed potential algorithms and data structures to be used in the project.
* Conducted in depth research on relevant topics such as queue management, stack structures, and mouse detection.

**Weeks 2–3: System Design**

* Finalized the main structure of the program.
* Drawing the map of the simulation.
* Bus and bus stop system.

**Weeks 3: Core Implementation**

* Mouse event detection and bus selection.
* Passenger and queue management system.
* Luggage handling system based on stack data structure.
* Adding statistic to the simulation.

**Week 4: Finalization and Presentation**

* Completed all coding and integration tasks.
* Fixed existing issues and bugs.
* Prepared and practiced for the project presentation.
* Wrote and finalized the project report, prepared presentation file and poster of the project.

# 2.CHAPTER: TASK SUMMARY

## 2.1 Completed Task

|  |  |  |
| --- | --- | --- |
| Who Did It? | Work | Description |
| Aleyna DOĞRUER | **Luggage System** | The luggage system tracks each passenger’s luggage, manages the transfer of luggage onto and off buses |
| Nurettin YILDIZ | **Passenger & Queue Management** | This system manages passengers using a queue for those waiting to be assigned to stops, and linked lists at each stop for those waiting for a bus. |
| Oğuz Efe GÜNEŞ | **Bus System** | The bus system manages the creation, movement, and operation of buses, including passenger and luggage handling at each stop. |
| Umut Efe AL | **User Interaction**  **&**  **Luggage System** | User interaction is managed through mouse and keyboard input for selecting buses and controlling the simulation.  The luggage system tracks, loads, and unloads passenger luggage in sync with passenger movement |

**Table 1**: Task distribution

## 2.2 Additional Improvements

To enhance the visual clarity and understanding of the simulation, we have implemented the following additional improvements:

* **Improved Visuality:** Beyond the core simulation logic, the user experience has been improved by adding features that use ANSI escape codes. These codes allow for updates and color changes at specific locations on the console screen, leading to dynamic displays.

For instance, bus locations, passenger counts, and queue states are updated in real-time at fixed positions, with different colors distinguishing buses, stops, and key statistics. This method makes the simulation more interactive and provides clear, immediate feedback as it progresses.



**Figure 1:** Example of ANSI Code Printing a Bus at a Specific Location

* **Thread Management:** Thread functions are used to separate user interaction (mouse/keyboard) and dynamic display updates from the main simulation logic, providing a responsive and visually appealing experience.

A screen shot of a computer screen

AI-generated content may be incorrect.

**Figure 2:** Example of Using Threads to Simultaneously Check for Mouse Input

* **Intelligent Pathfinding for Bus Navigation:** In the current simulation, each bus dynamically determines its next target stop and moves step-by-step toward that stop’s coordinates on the map.

This is accomplished by continuously checking the bus's current location and subtracting it from the next bus stop's location. The bus moves toward the stop until the result of this subtraction is zero. Once it reaches the stop, a new destination is chosen, and the process repeats.

A screenshot of a computer screen

AI-generated content may be incorrect.

**Figure 3:** Code that calculates which direction to move in the X (rows) and Y (columns) axes

# 3.CHAPTER: PROGRESS OVERVIEW

## 3.1 Functions

## 

The functions that we used for creating algorithms in this project given below:

### 1. Passenger & Queue Management

* **Queue createQueue()**  
  Creates and initializes an empty queue for passengers.
* **Passenger createPassenger()**  
  Allocates and initializes a new passenger with random attributes (line, start/end, luggage, etc.).
* **void SelectBusStops(Passenger \*passenger)**  
  Randomly selects start and end stops for a passenger and sets their direction.
* **void enqueue(Queue \*q, Passenger \*p)**  
  Adds a passenger to the end of the queue.
* **Passenger dequeue(Queue \*q)**  
  Removes and returns the passenger at the front of the queue, and enqueues a new random passenger.
* **void printQueue(Queue \*q)**  
  Displays the current queue of waiting passengers at a fixed screen location using ANSI codes.

### 2. Bus Stop System

* **void createBusStops(BusStop arr[BUS\_STOP\_SIZE])**  
  Initializes all bus stops with names, locations, and resets passenger lists/counts.
* **void connectPassengerToStop(BusStop arr[BUS\_STOP\_SIZE]** *Passenger passenger)*  
  Adds a passenger to the correct bus stop’s linked list and updates the passenger count.
* **void printPassenger(BusStop \*stop)**  
  Prints information about all passengers waiting at a specific stop.
* **Passenger\* getPassenger(BusStop \*stop, char line[BUS\_LINE\_SIZE], int dir)**  
  Finds and removes a suitable passenger (matching line and direction) from a stop’s list.
* **void printPassengerCounts(BusStop arr[BUS\_STOP\_SIZE])**  
  Prints the number of waiting passengers at each stop at their map locations using ANSI codes.

### 3. Bus System

* **void CreateBusses(Bus b[BUS\_LINE\_COUNT \* 2])**  
  Initializes all buses with names, routes, directions, locations, and resets all state variables.
* **void moveBusses(Bus b[BUS\_LINE\_COUNT \* 2], BusStop stops[BUS\_STOP\_SIZE])**  
  Controls bus movement, handles transitions between stops, and triggers passenger and luggage operations. Updates bus positions visually on the map.
* **void GetInTheBus(BusStop \*stop, Bus \*bus)**  
  Transfers eligible passengers from a stop to a bus, queues their luggage for loading, and updates the bus’s passenger list.
* **void GetOutTheBus(BusStop \*stop, Bus \*bus)**  
  Removes passengers from the bus who have reached their destination, queues their luggage for unloading, and updates the bus’s passenger list.

### 4. Luggage System

* **void loadLuggages(Bus \*bus)**  
  Loads luggage from the temporary loading queue onto the bus’s main luggage stack, updating indices and flags.
* **void UnloadLuggages(Bus \*bus)**  
  Unloads luggage from the bus for passengers who have reached their destination, managing temporary waiting and reloading as needed.

### 5. User Interaction

* **int getMousePosition()**  
  Waits for and returns the current mouse click position in the console.
* **unsigned \_\_stdcall SelectViaMouse(void \*arg)**  
  Thread function that continuously checks for mouse clicks to select and highlight a bus on the map.
* **unsigned \_\_stdcall DisplayBusInfo(void \*arg)**  
  Thread function that continuously displays detailed information about the selected bus, its passengers, and luggage.
* **unsigned \_\_stdcall CheckForKeyboard(void \*arg)**  
  Thread function that monitors keyboard input to pause/resume the simulation (Spacebar/'R').

### 6. Statistics

* **void printStatistic(Bus b[BUS\_LINE\_COUNT \* 2], BusStop s[BUS\_STOP\_SIZE])**  
  Calculates and displays overall statistics such as waiting passengers, travelling passengers, and bus fullness at a fixed screen location using ANSI codes.
* **int countDigits(int num)**  
  Utility function to count the number of digits in an integer (used for formatting statistics).

## 3.2 Solution Strategies of Algorithms

**1. Passenger & Queue Management**

* **Dynamic Creation:**  
  Use a queue to manage the flow of new passengers. Dynamically create passengers with random attributes and enqueue them, ensuring a steady supply for the simulation.
* **Efficient Assignment:**  
  When dequeuing, immediately assign the passenger to the correct bus stop based on their starting point. Maintain linked lists at each stop for efficient passenger management.
* **Capacity Handling:**  
  Always check bus and stop capacities before boarding or enqueuing to prevent overflows and maintain simulation integrity.
* **Visual Feedback:**  
  Use ANSI codes to display the queue and passenger information in real time, helping users track system state.

**2. Bus Stop System**

* **Initialization:**  
  Initialize all bus stops with unique names, locations, and empty passenger lists at the start of the simulation.
* **Passenger Linking:**  
  Use linked lists for passengers at each stop, allowing efficient addition, removal, and traversal.
* **Real-Time Updates:**  
  Regularly update and display the number of waiting passengers at each stop using ANSI codes, providing immediate visual feedback.
* **Synchronization:**  
  Ensure thread-safe operations if bus stops are accessed by multiple threads (e.g., during boarding and statistics updates).

**3. Bus System**

* **Structured Initialization:**  
  Create buses with unique names, routes, directions, and initial positions. Initialize all state variables, including passenger and luggage arrays.
* **Movement Logic:**  
  Implement robust logic for moving buses along their routes, handling direction changes at endpoints, and updating positions visually.
* **Boarding/Unboarding:**  
  At each stop, trigger boarding (GetInTheBus) and unboarding (GetOutTheBus) functions, ensuring only eligible passengers are transferred.
* **State Management:**  
  Use flags and counters to manage bus states (e.g., loading, unloading, moving) and prevent race conditions.

**4. Luggage System**

* **Ownership Tracking:**  
  Assign luggage to passengers upon creation and ensure ownership is maintained as luggage moves between stops and buses.
* **Stack & Queue Operations:**  
  Use stack-like arrays for bus luggage and temporary arrays for loading/unloading. Carefully manage indices and counts to avoid overflows.
* **Loading/Unloading Control:**  
  Use flags and delays to control the pace of luggage operations, simulating realistic loading/unloading times.
* **Visual Representation:**  
  Display luggage status for each bus using ANSI codes, making it easy for users to track luggage flow.

# 4.CHAPTER: PROBLEMS ENCOUNTERED

1. ANSI Code Does Not Work

***Problem:*** Initially, the ANSI codes used to print at specific locations did not work properly in some terminals, such as the one built into Visual Studio Code. This is because ANSI codes are specifically designed for the Windows terminal.

Additionally, when the code does run in the Visual Studio Code terminal, it may overwrite existing text on the screen that was not printed by the program.

***Solution:*** The solution is to compile the C program into an .exe file using the GCC compiler, and then run the executable in the Windows Command Prompt (CMD). This ensures that ANSI codes function correctly and display output as intended.

1. Busses Using Wrong Path

***Problem:*** Initially, the buses followed incorrect paths because their location system did not align properly with the CMD terminal display. There was a visible displacement of the buses on CMD. This occurred because the first two rows and columns in CMD are reserved for row and column labels (numbers and letters), which shifted the visual map.

***Solution:*** The solution was to add a base offset to the buses' X and Y coordinates to align them correctly with the map on CMD. In our case, the offset value was 2, since the map was visually shifted by 2 units both vertically and horizontally.

1. Mouse Can Select Multiple Busses at The Same Time

***Problem:*** If the mouse clicked on the busses that on the same location, then it selected that busses at the same time and make them all green (Our busses turn into green when selected) resulting in incorrect or misleading information being shown in the bus information menu.

***Solution:*** The issue was resolved by modifying the SelectViaMouse function. Conditional logic (if / else statements) was added to ensure that only the first bus found at a given location is selected, preventing multiple buses at the same spot from being selected simultaneously.

1. Luggage Cannot Be Removed from the Bus

***Problem:*** When the luggage system was first implemented, it did not support properly removing luggage from the stack. Although removing an item seemed straightforward, the system required removing luggage from the top of the stack, since stacks follow a Last-In, First-Out (LIFO) structure.

For example, imagine we are trying to remove a luggage item with ID 95, and the current stack looks like this:  
head → [100, 91, 95, 96]  
In this case, we cannot directly access luggage 95. Instead, we must first remove 100 and 91, then take 95. After removing 95, the previously removed items (100 and 91) need to be pushed back onto the stack in the correct order.

However, the function responsible for this process had a bug: it could only remove luggage from the top of the stack. If the desired luggage was in the middle or at the bottom, the function failed to retrieve it, leaving the luggage on the bus — even after the passenger had already left.

***Solution:*** The solution involved adding logic to handle cases where the desired luggage was not at the top of the stack. If the target luggage was deeper in the stack, the program would:

**1.**Pop luggage items from the top of the stack one by one until the target luggage was found.

**2.**Temporarily store the removed luggage in a special list.

**3.**Once the target luggage was reached and removed, the items in the special list were pushed back onto the stack in the correct order.

This approach allowed the system to safely retrieve any luggage, regardless of its position in the stack, and then restore the remaining luggage correctly. It effectively resolved the issue of luggage staying on the bus after the passenger had left.

***A screen shot of a computer code

AI-generated content may be incorrect.***

**Figure 4:** Part of the Code Responsible for Solving the Luggage Removal Issue

# 5.CHAPTER: FUTURE WORK

We have already completed all the tasks and objects specified in the project file.

However, in the future, we could add traffic interactions—like making buses wait when another bus is blocking the road or approaching an intersection.

We could also give passengers emotions that change over time; for example, they might become angry if the bus is late or too crowded, and then become happy when the ride improves.

Additionally, we could implement a point-based system where buses are rated by their performance. If a bus consistently receives low ratings, the driver could be fired or replaced in the simulation.

# 6.CHAPTER: CONCLUSION

In this project, we successfully implemented all the required components outlined in the project file, including the core functionalities of the bus system. Our simulation now demonstrates the basic operations of buses, stops, and passenger interactions effectively.

Looking ahead, we see potential for future improvements such as realistic traffic behavior, dynamic passenger emotions based on ride conditions, and a performance-based point system for drivers. These additions would make the simulation more interactive, lifelike, and engaging.

Overall, this project provided valuable experience in both designing systems and thinking critically about how to enhance user experience in simulations.

# References

|  |  |
| --- | --- |
| [1] | "MOUSE\_EVENT\_RECORD structure," 30 10 2020. [Online]. Available: https://learn.microsoft.com/en-us/windows/console/mouse-event-record-str. [Accessed 8 5 2025]. |
| [2] | "Multithreading with C and Win32," 8 3 2021. [Online]. Available: https://learn.microsoft.com/en-us/cpp/parallel/multithreading-with-c-and-win32?view=msvc-170. [Accessed 8 05 2025]. |
| [3] | "Console Structures," 18 05 2021. [Online]. Available: https://learn.microsoft.com/en-us/windows/console/console-structures. [Accessed 8 5 2025]. |
| [4] | "The entire table of ANSI color codes working in C!," [Online]. Available: https://gist.github.com/RabaDabaDoba/145049536f815903c79944599c6f952a. [Accessed 2 05 2025]. |
| [5] | "Print character at coordinates in C," 01 2021. [Online]. Available: https://stackoverflow.com/questions/65294104/print-character-at-coordinates-in-c. [Accessed 2 05 2025]. |
| [6] | "Queue in C," [Online]. Available: https://www.geeksforgeeks.org/queue-in-c/. [Accessed 10 05 2025]. |

# Appendices

Demo video is uploaded to on the SAKAI together with this report