Unitec Institute of Technology
TE WHARE WĀNANGA O WAIRAKA

School of Computing and Information Technology

Percentage Mark: 40% of final course marks

Mini-Project - documentation

Learning outcomes covered in this assignment.

1. Apply object-oriented design and implementation techniques.
2. Interpret the trade-offs and issues involved in the design, implementation, and application of various data structures with respect to a given problem.
3. Explain the purpose and answer questions about data structures and design patterns that illustrate strengths and weaknesses with respect to resource consumption.
4. Assess the impact of data structures on algorithms.
5. Analyse the scalability of data structures and algorithms in terms of both space and time complexity.

Cover Sheet

First Name	Sifa
Last Name	Zhang
Student ID	1606796
Chosen Data Structure(s) or Algorithm(s)	Queue

By submitting files and/or work to the approved Moodle submission link for this assessment, I declare that all work has been performed by myself unless explicitly declared. Any code not created by me has been cited adequately. I accept that failure to comply with the Unitec Guidelines of Appropriate Student Conduct will result in enforcement of the relevant consequences.

Signed: Sifa Zhang Date: 22/09/2025

1.Data Structure / Algorithm

Introduce and describe your chosen DS/A. This may include a brief history, and its purpose.

A queue is a linear data structure that follows the First-In, First-Out (FIFO) principle—much like a line at a ticket counter. The first element added is the first one to be removed. This behavior makes queues ideal for scenarios where order matters, such as task scheduling, buffering, and breadth-first search in graphs.

- **Brief History**

The concept of queues dates back to early computing systems, where they were used to manage job scheduling in operating systems. As programming languages evolved, queues became a fundamental part of many standard libraries, reflecting their importance in both theoretical and practical applications.

- **Purpose & Use Cases**

Queues are used to:

- Manage asynchronous data (e.g., print queues, IO buffers)

- Implement algorithms like Breadth-First Search (BFS) in graphs

- Control access to shared resources in concurrent systems

- Simulate real-world processes (e.g., customer service lines, traffic systems)

2.Strengths & Weaknesses

Discuss the aspects or characteristics of the DS/A that are good, bad, or somewhere in between. This is a good section to cover time and space complexity issues. You can reflect on how the intent of the DS/A aligns with the performance characteristics as to whether it is an effective tool or not.

- **Strengths**
  - ✓ Simplicity & Predictability
    The FIFO (First-In, First-Out) behavior is intuitive and mirrors many real-world systems
  - ✓ Efficient Operations
    - Enqueue (add to rear): O(1) in linked list or circular buffer implementations
    - Dequeue (remove from front): O(1) in linked list or circular buffer
      These constant-time operations make queues ideal for high-throughput systems.

- ✓ Great for Asynchronous Processing

  Queues are foundational in producer-consumer models, event-driven systems, and multithreading, where tasks are processed in the order they arrive.

- ✓ Breadth-First Search (BFS)
  In graph traversal, queues ensure nodes are explored level by level, making them essential for algorithms like BFS.

- **Limitations**
  - ✓ Limited Access
    You can only access the front or rear—no random access like arrays or lists. This makes queues unsuitable for tasks requiring indexed lookup or sorting.
  - ✓ Space Overhead
    - In array-based queues, resizing or shifting elements can be costly unless implemented as a circular buffer.
    - In linked list queues, each node carries pointer overhead, which adds memory cost.
  - ✓ Blocking Behaviour in Real-Time Systems
    If not managed properly, queues can become bottlenecks—especially in systems where producers outpace consumers (e.g., message queues without throttling).

3.Real-World Example

List and briefly describe at least one use of your DS/A in a real-world use case.

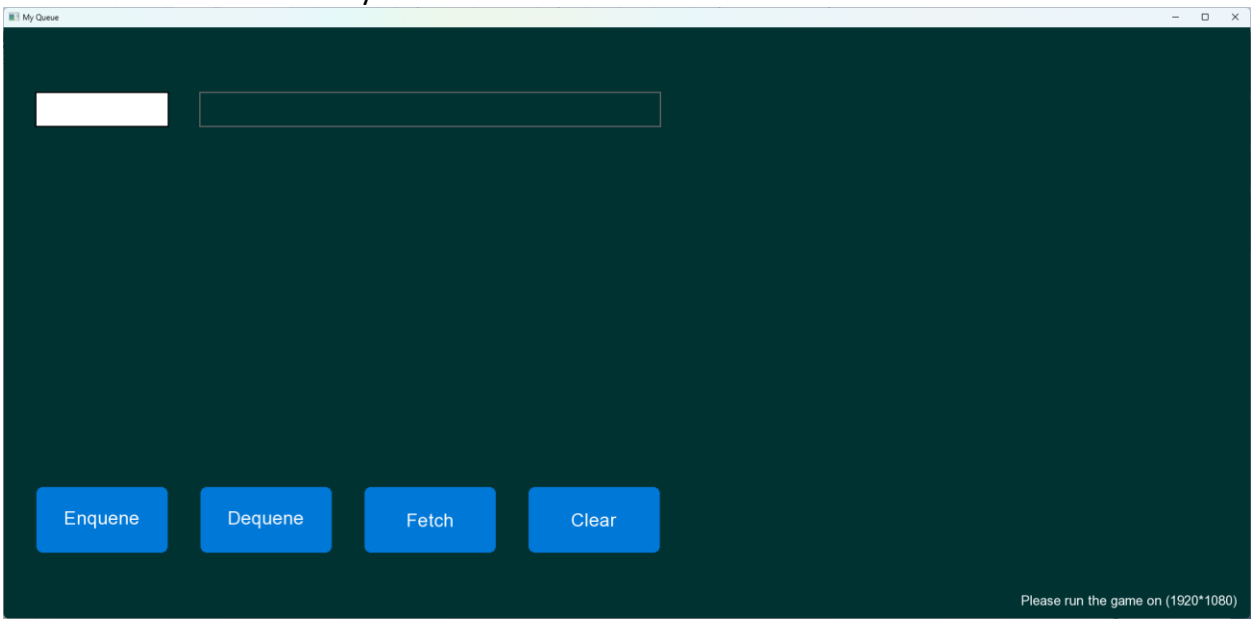
In modern operating systems, a print queue manages multiple print jobs sent to a printer. When several users or applications request printing at the same time, the system places each job into a queue, ensuring that documents are printed in the order they were received—following the First-In, First-Out (FIFO) principle. It works like this:

- Enqueue: Each print job is added to the rear of the queue.
- Dequeue: The printer processes and removes the job from the front of the queue.
- Benefits: Prevents job collisions, ensures fairness, and allows users to cancel or reorder jobs if needed.

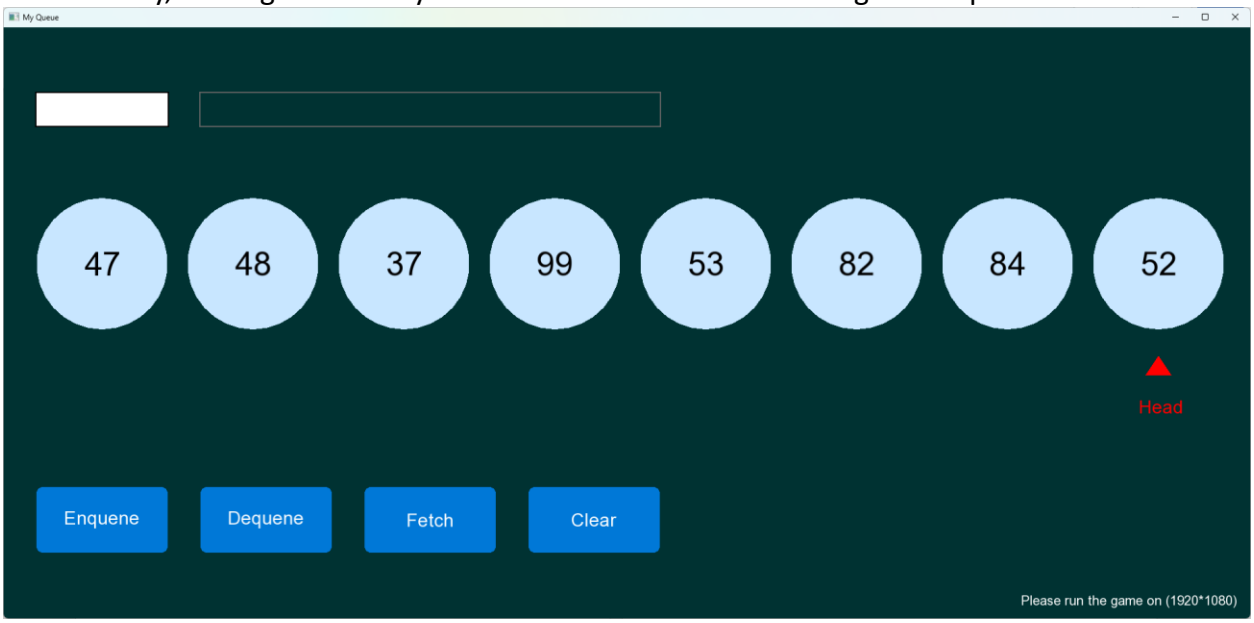
4.Implementation

Discuss how you implemented the DS/A in code, and how you changed it to be suitable for visualisation in your project. This is your chance to show your time and effort, and cover details that you may not have previously. This section and your demo will have significant overlap – use it to your advantage.
Screenshots, code snippets, and other supplementary sources are recommended for this section, but not required.

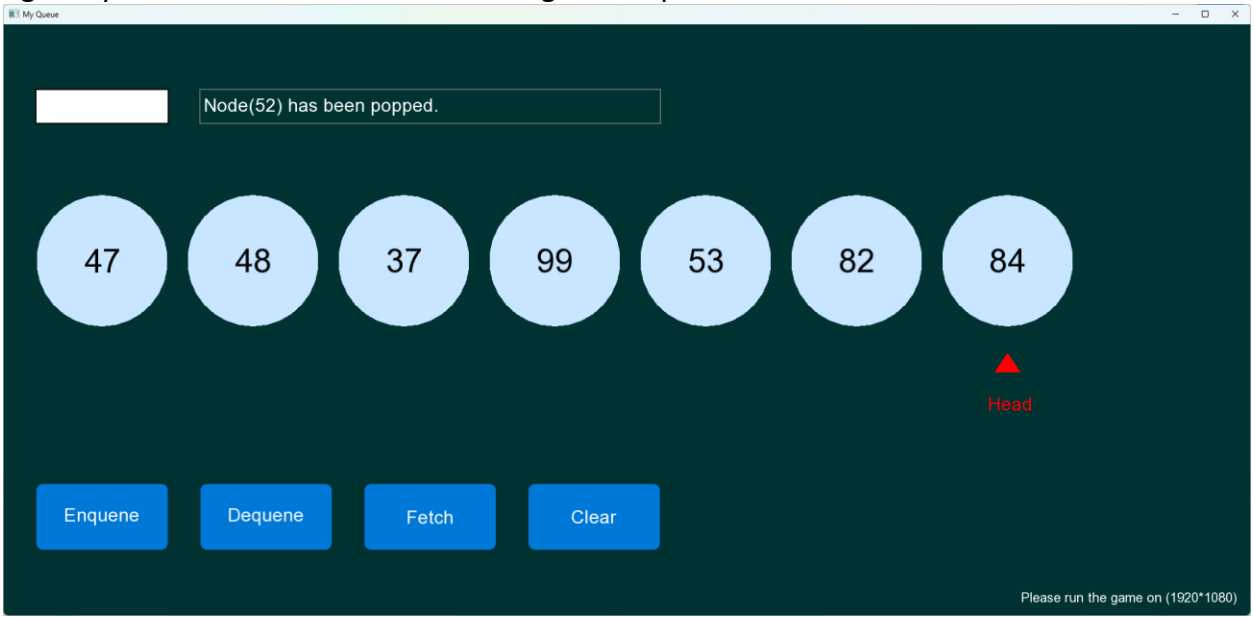
I implemented the visual Queue based on Assignment 3, adding some buttons and shapes. I add four buttons(Enqueue, Dequeue, Clear and Fetch), a Textbox and a Label to interact with users. The system will draw a list of circles from head to tail to show the order of the queue. Also, the system will draw an arrow to indicate the head node.
This is the initiation of the system.



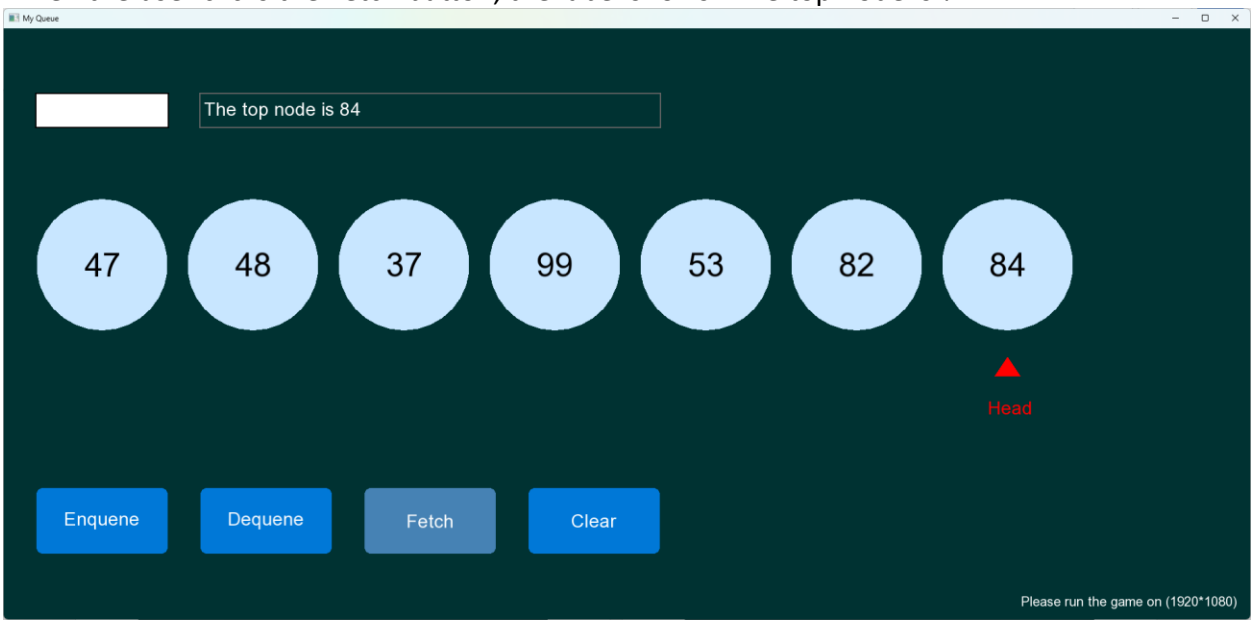
When the user inputs an integer and clicks the Enqueue button, the system will draw circles. An arrow shows the head node. The system will give a random data if the user does not input. Alternatively, clicking the left key of the mouse can work as clicking the Enqueue button.



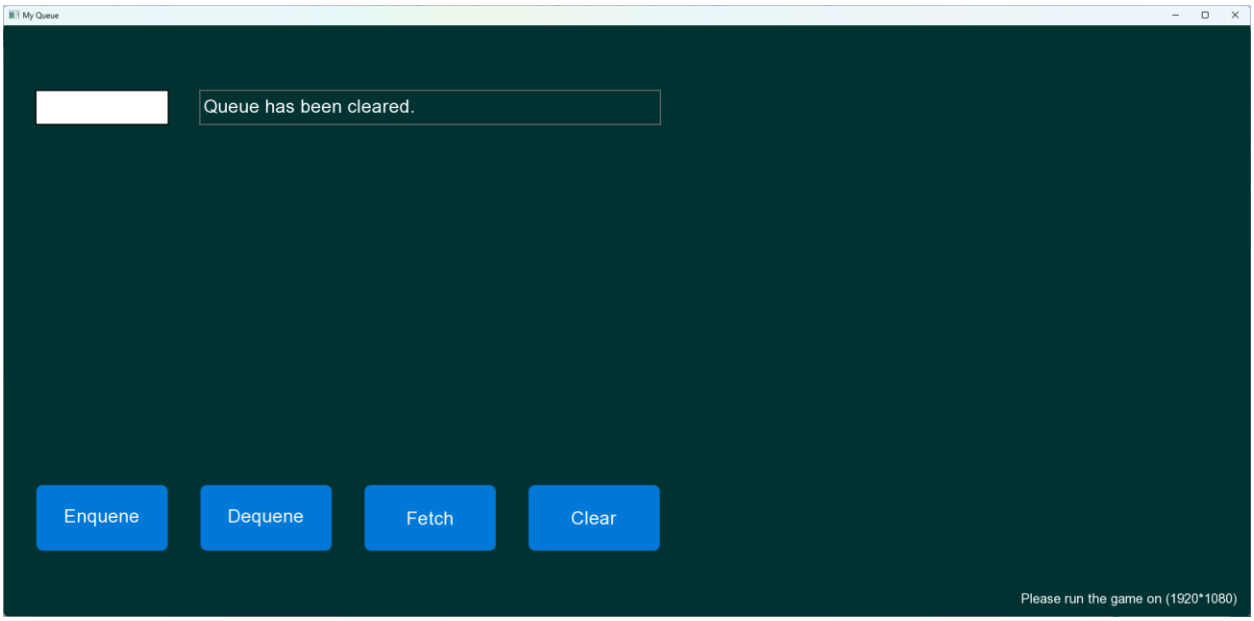
When the user clicks the Dequeue button, the system will update the circles. The top node has been removed, and the label will show Node(?) has been popped. Alternatively, clicking the right key of the mouse can work as clicking the Dequeue button.



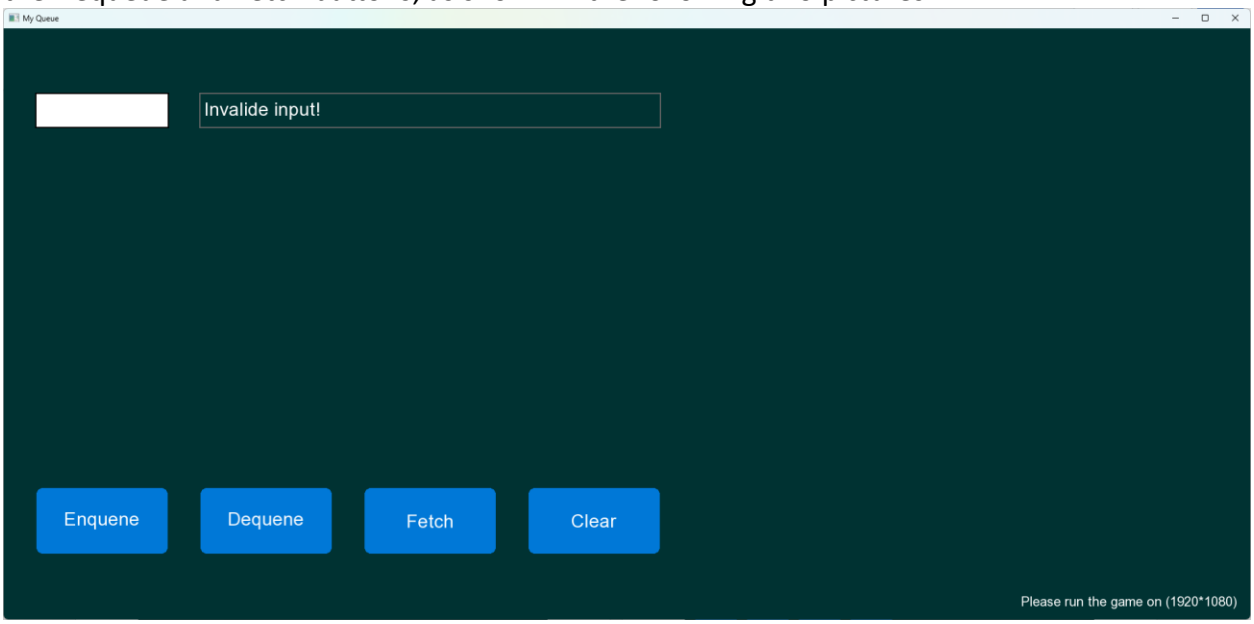
When the user clicks the Fetch button, the label shows “The top node is ?”.

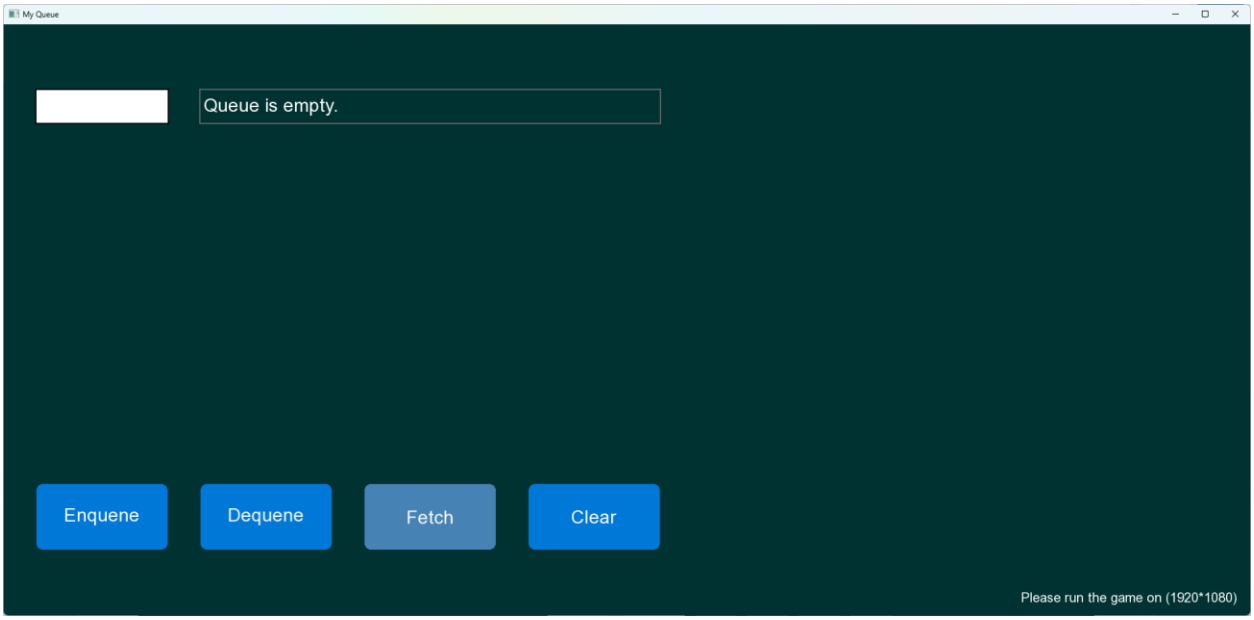


When the user clicks the Clear button, the queue clears and the label shows “Queue has been cleared.”.



The system also checks the validity of the input and the emptiness of the queue when clicking the Dequeue and Fetch buttons, as shown in the following two pictures.





When the user presses the Space key, the background colour of the circles will change to yellow.