Summary for task 5

The Queue Data Structure

A **queue** is a linear data structure that follows the **First In, First Out (FIFO)** principle, meaning elements are added at the **rear** and removed from the **front**. It works like a real-world queue (e.g., people standing in line).

Key Operations of a Queue

1. **Enqueue (Insertion)** → Adds an element to the rear of the queue.
2. **Dequeue (Removal)** → Removes an element from the front of the queue.
3. **IsEmpty** → Checks if the queue is empty.
4. **Peek (Optional)** → Returns the front element without removing it.

Queues are commonly used in **task scheduling, breadth-first search (BFS), simulations, and game loops**.

How We Used a Queue in Our Game

In our **Unity game**, we implemented a **custom queue** using a **linked list** instead of using Unity's built-in `Queue<T>`. Here's how the queue was used in the game:

1. **Initializing the Queue:**
    - At the start of the game, all characters are added to the queue using **Enqueue()**.
2. **Game Loop (Turn-Based System):**
    - In each round, the first character in the queue is removed using **Dequeue()** and takes its turn.
    - The character takes damage, and if it **survives**, it is added back to the queue using **Enqueue()**.
    - This cycle continues until no characters are left in the queue.
3. **Printing the Queue:**
    - At the start of each round, we **print the queue** to see which characters are still alive and taking turns.
4. **Game Over Condition:**

- The game **ends** when the queue becomes **empty**, meaning all characters have died.

This **queue-based system** ensures a fair turn-based gameplay, where each character acts in order, and only those who survive continue in the game.