JOHN ELMER BLANQUISCO BOBIER JR & VINCENT MACAUYAM

Scenario: Galactic Cargo Management

The program places you in a futuristic setting where you manage cargo shipments across the galaxy.

Story

In the far future, humans have built colonies on planets across the galaxy. These colonies depend on shipments of important supplies like food, medicine, and tools to survive and grow. You are the cargo manager on the spaceship *SS Nexus*, one of the fastest and most advanced freighters in the galaxy.

Your job is to:

1. **Insert Shipments:** Add new shipments into the priority system without disrupting the order.
2. **Dispatch Shipments:** Remove and deliver the most urgent shipments efficiently.
3. **Maintain Order:** Use the Heapify process to keep the system working correctly.
4. **Monitor Changes:** Watch how your actions impact the priority structure and make adjustments as needed.

The backstory immerses you as a "cargo manager" on a spaceship, adding a narrative flair. Print statements like *“Inserted shipment with priority X into the heap”* create engaging feedback.

Game Logic

The program uses a Max-Heap to prioritize shipments based on their urgency.

- **Heap Operations:** Demonstrates insertion, removal (dispatch), and heapify operations in a Max-Heap.
- **Heapify Functions:** Maintains heap properties using `heapifyUp` and `heapifyDown`.
- **Practical Use Case:** Illustrates managing priorities in real-world scenarios using heaps.
- **Visual Representation:** Displays the heap as a vector using the `displayHeap` function to show its current state.

Task Operations

The program provides three primary tasks:

1. **Insert Shipments:** Add a shipment with a given priority into the Max-Heap without breaking the heap structure.
2. **Dispatch Shipments:** Remove the highest-priority shipment (root) from the heap while maintaining the Max-Heap property.
3. **Monitor Heap State:** View the current state of the heap after each operation to visualize the changes.

SFML SPRITE -for the interface design of our Game
Below are the explanations:

What is SFML?

SFML (Simple and Fast Multimedia Library) is a library used to create graphics, windows, and handle input/output for game or application development. It lets us display images, shapes, text, and more.

```
1    #include <SFML/Graphics.hpp>
```

In your code, the SFML sprite design is used for **background images** and **text displays**. Here's an explanation of each related function and how it connects to the graphics.

Function: `loadBackground`

This function is responsible for loading an image file (like `.jpg` or `.png`) to use as a **background** for different parts of your application.

```
// Function to load and scale a background image
sf::Sprite loadBackground(const string& filePath, sf::Texture& texture, float scaleX, float scaleY) {
    if (!texture.loadFromFile(filePath)) {
        cout << "Error: Could not load " << filePath << endl; // Error loading background image
    }
    sf::Sprite sprite;
    sprite.setTexture(texture); // Set the texture for the sprite (background)
    sprite.setScale(scaleX, scaleY); // Scale the sprite to fit the window size
    return sprite;
}
```

Key Points:

- **`sf::Texture`**: Holds the image data loaded from a file.
- **`sf::Sprite`**: A graphical object that displays the image on the screen.
- **`setScale(scaleX, scaleY)`**: Changes the size of the image (e.g., to fit the window).

Example of `loadBackground` Usage

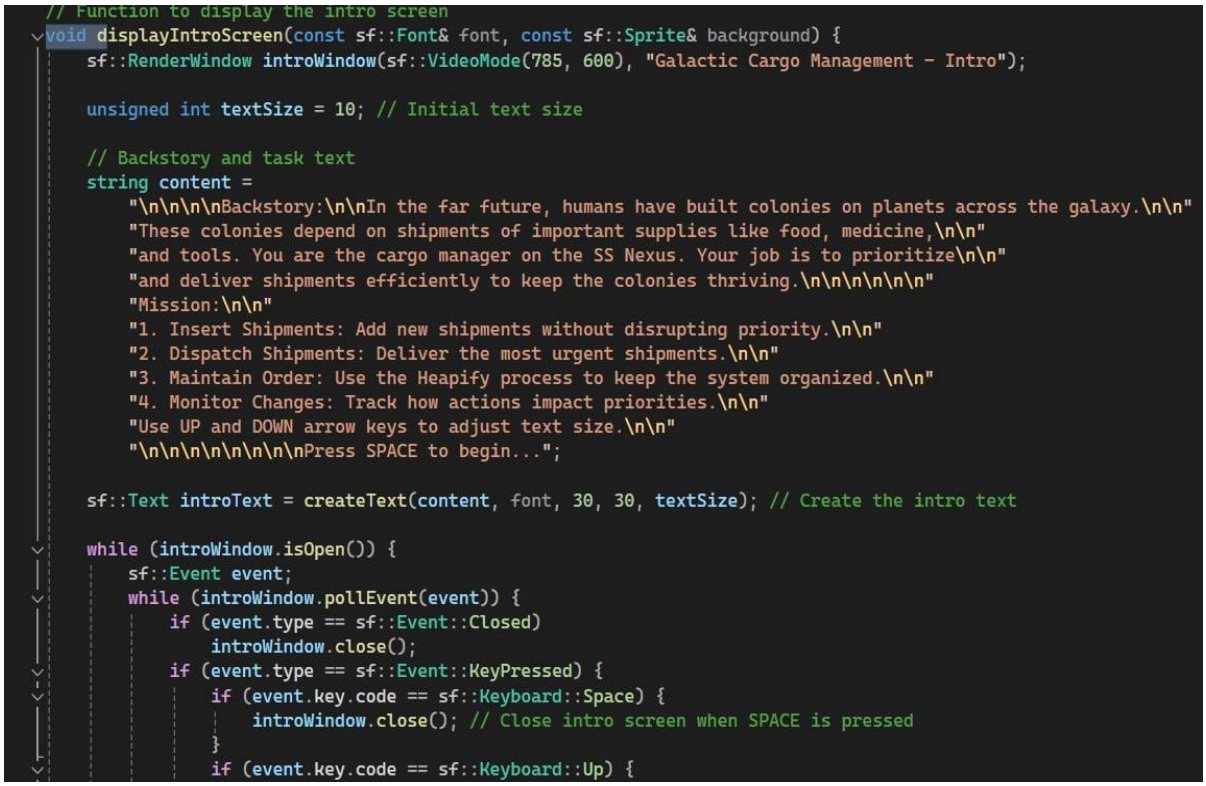
```
// Load background images for various screens
sf::Texture introTexture, menuTexture, insertTexture, dispatchTexture, viewTexture;
sf::Sprite introBackground = loadBackground("intro.jpg", introTexture, 0.5f, 0.6f);
sf::Sprite menuBackground = loadBackground("startup.jpg", menuTexture, 0.5f, 0.6f); // Adjust the scale to fit your window size
sf::Sprite insertBackground = loadBackground("insert.png", insertTexture, 1.7f, 1.5f);
sf::Sprite dispatchBackground = loadBackground("dispatch.png", dispatchTexture, 1.6f, 1.6f);
sf::Sprite viewBackground = loadBackground("view.png", viewTexture, 1.9f, 1.9f);
```

- **`"intro.jpg"`**: The file path for the image.
- **`introTexture`**: Stores the image data.
- **`introBackground`**: A sprite that will display the scaled image (half size on X-axis, 60% on Y-axis).

Later, `introBackground` is drawn using `window.draw(introBackground)`.

Function: `displayIntroScreen`

This function shows the **intro screen** with a background image and some text. Here's how it works:



```
// Function to display the intro screen
void displayIntroScreen(const sf::Font& font, const sf::Sprite& background) {
    sf::RenderWindow introWindow(sf::VideoMode(785, 600), "Galactic Cargo Management - Intro");

    unsigned int textSize = 10; // Initial text size

    // Backstory and task text
    string content =
        "\n\n\n\nBackstory:\n\nIn the far future, humans have built colonies on planets across the galaxy.\n\n"
        "These colonies depend on shipments of important supplies like food, medicine,\n\n"
        "and tools. You are the cargo manager on the SS Nexus. Your job is to prioritize\n\n"
        "and deliver shipments efficiently to keep the colonies thriving.\n\n\n\n\n\n"
        "Mission:\n\n"
        "1. Insert Shipments: Add new shipments without disrupting priority.\n\n"
        "2. Dispatch Shipments: Deliver the most urgent shipments.\n\n"
        "3. Maintain Order: Use the Heapify process to keep the system organized.\n\n"
        "4. Monitor Changes: Track how actions impact priorities.\n\n"
        "Use UP and DOWN arrow keys to adjust text size.\n\n"
        "\n\n\n\n\n\n\n\nPress SPACE to begin...";

    sf::Text introText = createText(content, font, 30, 30, textSize); // Create the intro text

    while (introWindow.isOpen()) {
        sf::Event event;
        while (introWindow.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                introWindow.close();
            if (event.type == sf::Event::KeyPressed) {
                if (event.key.code == sf::Keyboard::Space) {
                    introWindow.close(); // Close intro screen when SPACE is pressed
                }
                if (event.key.code == sf::Keyboard::Up) {
```

Key Points:

- **`sf::RenderWindow`**: Creates a window for displaying the intro screen.
- **`background`**: The sprite passed to this function, which will be drawn on the window.
- **`introWindow.draw(...)`**: Displays the background and text.

Function: `createText`

This function simplifies how you create text to display in our app.

```
// Function to create text for displaying in the window
sf::Text createText(const string& content, const sf::Font& font, float x, float y, unsigned int size) {
    sf::Text text;
    text.setFont(font); // Set the font for the text
    text.setString(content); // Set the text string
    text.setCharacterSize(size); // Set the font size
    text.setFillColor(sf::Color::Green); // Set the text color to green
    text.setPosition(x, y); // Set the position of the text in the window
    return text;
}
```

Key Points:

- `sf::Font`: Holds the font style for the text.
- `setPosition(x, y)`: Positions the text on the window.
- `setCharacterSize(size)`: Adjusts the text size.

How Backgrounds and Sprites are Used in Main Menu

In the main menu, the **background sprite** and text are displayed as follows:

sf::Sprite menuBackground = loadBackground("startup.jpg", menuTexture, 0.5f, 0.6f);

```
sf::Text menuText = createText(
    "1. Insert Shipment\n\n"
    "2. Dispatch Shipment\n\n"
    "3. View Heap ",
    font, 350, 150);
sf::String userInput;
```

1. `loadBackground`: Loads the `startup.jpg` image, scales it, and stores it in `menuBackground`.
2. `createText`: Creates text for the menu options, positioning it at `(350, 150)`.

Inside the event loop, the window is cleared and the sprite and text are drawn:

```
window.clear();
window.draw(menuBackground); // Use the menu background here
window.draw(menuText); // Display menu options
window.draw(inputText); // Display user input
window.display();
```

Summary of Key SFML Concepts

1. **Textures**: Load image files using `sf::Texture`.
2. **Sprites**: Display those images on the screen using `sf::Sprite`.
3. **Windows**: Create a window using `sf::RenderWindow` to draw sprites and text.
4. **Drawing Order**: Use `window.draw(...)` to render graphics, starting with backgrounds first, then other elements like text.

Program Logic:

- The program uses a loop where users can perform actions like inserting, dispatching, or viewing the heap.
- A `vector` represents the heap, while predefined `shipments` provide initial values for demonstration.

Interactive Features:

- Users select from available shipments, ensuring that only valid priorities are added.
- The program ends gracefully when all shipments are managed, giving closure to the scenario.

Instructions for the User

The program includes user-friendly commands to ensure an interactive and enjoyable experience:

Commands:

1. **Insert Shipments:** Select a shipment by its priority value to add it to the heap.

2. **Dispatch Shipments:** Remove and deliver the highest-priority shipment.
3. **View the Heap:** Display the current state of the heap to observe changes.
4. **Exit:** Close the program when all shipments have been managed.

Guided Inputs:

- The program prompts you to select a shipment's priority value from the available list.
- Validates input to ensure only appropriate values are used.

Examples:

1. **Insertion:** Adds a shipment to the heap while showing the updated heap.
2. **Dispatch:** Removes the highest-priority shipment and displays the remaining heap.

<u>This is the Steps / Instructions we follow in Creating our Game:</u>

Step 1: Form a Partnership

Work with a partner to complete this activity.

Discuss and agree on a unique theme or scenario for your challenge.

Step 2: Set the Scene

Imagine a creative scenario or theme for your assessment.

"Galactic Cargo Management"

- In the far future, humans have built colonies on planets across the galaxy. These colonies depend on shipments of important supplies like food, medicine, and tools to survive and grow. To keep everything running smoothly, a system is needed to organize and prioritize deliveries based on how urgent they are.
- participants are on a spaceship named SS Nexus, one of the fastest and most advanced freighters in the galaxy. participants will take on the role of a "Galactic Resource Manager" for a futuristic space colony. The colony's survival depends on efficiently managing shipments of rare resources from different planets. Each resource has a priority based on its value. The challenge involves using Heap Data Structures to prioritize, manage, and allocate resources dynamically as new shipments arrive or existing resources are consumed.

Step 3: Define Your Learning Goals

Together, decide what the participants should learn or demonstrate by completing your tasks.

- Mastering insertion and deletion in Max-Heaps or Min-Heaps:
- Adding shipments into the priority system (Max-Heap or Min-Heap) corresponds to the insertion process.
- Dispatching the most urgent shipment aligns with deleting the root (highest-priority element) in a heap.

Implementing the Heapify operation on an array:

- The Heapify process is used to fix the heap whenever shipments are added, removed, or the order is disrupted.
- This ensures the priority system always follows the heap property.
- Converting between heap types or visualizing heaps:
- Visualizing the heap structure through the display helps participants understand the hierarchy and how operations affect it.
- You could optionally add a feature to convert between Max-Heap and Min-Heap (e.g., changing how priorities are interpreted).

Step 4: Create the Tasks

Include at least three tasks related to heaps.

Insert elements into a heap: Users add shipments (with priorities) into the heap, and the system displays the updated heap after each insertion.

Implement a function to delete the root and restore the heap property: The function removes the root (highest-priority shipment) and restores the heap property by rebuilding the heap.

Write a program to Heapify a random array into a Max-Heap: This is covered by the buildHeap function in your code, which uses heapify to transform an unsorted array into a valid Max-Heap.