DSA

University of management and technology |

Project

name

[Year]

**Game Arcade Token & Session Management System**

**Final Project Report**

**1. Introduction**

The **Game Arcade Token & Session Management System** is a C++ application designed to manage gaming machines, player token requests, and session history in a game arcade. The system uses fundamental data structures (Tree, Linked List, Queue, and Stack) to efficiently organize and process arcade operations.

**2. System Requirements**

**Functional Requirements**

1. **Zone & Machine Management**
   * The arcade has multiple zones (VR, Racing, Puzzle, Shooting).
   * Each zone contains gaming machines with:
     + Machine ID
     + Game Type
     + Status (Available, In Use, Reserved, Out of Order)
     + Token Type (1-token, 2-token, or 3-token games)
2. **Token Request Handling**
   * Players can request tokens by specifying:
     + Game Type
     + Session Duration (minutes)
   * Priority-based queue:
     + **Regular players** → FIFO queue
     + **Premium/VIP players** → High-priority queue
3. **Session History & Rollback**
   * Completed sessions are stored in a **stack** (LIFO).
   * If a session is canceled (e.g., machine crash), the last session is rolled back.

**Non-Functional Requirements**

* **Efficiency**: Uses optimized data structures for fast operations.
* **User-Friendly**: Menu-driven interface for easy interaction.
* **Memory Safety**: Proper cleanup of dynamically allocated memory.

**3. Data Structures Used**

| **Data Structure** | **Purpose** | **Implementation** |
| --- | --- | --- |
| **Tree (Binary Search Tree)** | Organizes arcade zones (VR, Racing, etc.) | ZoneTree class |
| **Linked List** | Manages machines in each zone | MachineList class |
| **Queue (Priority Queue)** | Handles player token requests | RequestQueue class |
| **Stack** | Records session history (LIFO) | SessionStack class |

**4. Key Features**

**1. Machine & Zone Management**

* **Initialization**:
  + 4 zones (VR, Racing, Puzzle, Shooting).
  + 8 machines per zone.
* **Machine Status Tracking**:
  + Available, In Use, Reserved, Out of Order.

**2. Token Request Processing**

* **Priority Handling**:
  + VIP requests jump to the front of the queue.
* **FIFO Order**:
  + Regular requests follow a first-come-first-serve basis.

**3. Session History & Rollback**

* **Stack-Based History**:
  + Each session is pushed onto the stack.
  + Canceling a session pops the last entry.

**4. Statistics & Reporting**

* **Machine Status**:
  + Shows available vs. in-use machines.
* **Request Queue**:
  + Displays pending requests.
* **Session History**:
  + Shows completed/canceled sessions.

**5. Code Implementation Highlights**

**1. Tree-Based Zone Management (**ZoneTree**)**

A black rectangular object with white text

AI-generated content may be incorrect.

* **Operations**:
  + addZone() → Adds a new zone (e.g., "VR").
  + getZone() → Retrieves a zone by name.

**2. Linked List for Machines (**MachineList**)**

A black rectangular object with white text

AI-generated content may be incorrect.

* **Key Methods**:
  + findAvailableMachine() → Assigns a machine to a player.
  + updateMachineStatus() → Changes machine state.

**3. Priority Queue for Requests (**RequestQueue**)**

A black rectangular object with white text

AI-generated content may be incorrect.

* **VIP Requests**:
  + Added to the front of the queue.

**4. Stack for Session History (**SessionStack**)**

A black rectangular object with white text

AI-generated content may be incorrect.

**Rollback**:

* pop() removes the last session (e.g., machine crash).

**6. Testing & Validation**

**Test Cases**

| **Test Case** | **Input** | **Expected Output** |
| --- | --- | --- |
| **Initialize Arcade** | 1 | 4 zones, 8 machines each |
| **Add Token Request** | 2 → (Player: "Alice", Game: "VR", Priority: Yes) | Request added to front of queue |
| **Process Requests** | 3 | VIP requests processed first |
| **Cancel Session** | 4 | Last session removed, machine status reset |
| **View Statistics** | 6 | Shows available/in-use machines |

**Edge Cases Handled**

✔ **Invalid Inputs** (e.g., negative duration, wrong game type)  
✔ **Empty Queue/Stack** (graceful handling)  
✔ **Memory Leaks** (destructors clean up dynamically allocated memory)

**7. Conclusion**

The **Game Arcade Management System** successfully implements:  
✅ **Zone & Machine Organization** (Tree + Linked List)  
✅ **Priority-Based Token Requests** (Queue)  
✅ **Session History & Rollback** (Stack)  
✅ **User-Friendly Menu Interface**

**Future Improvements**

* **Database Integration**: Store player profiles and machine data persistently.
* **GUI**: Implement a graphical interface for better usability.
* **Multiplayer Support**: Allow shared machine sessions.