

Project 6 - MultiSet Design

Data Structure and Algorithm

December 1, 2025

Abstract

1 Introduction

Based on what I understand from description of project 6, I had two options: an RPG game or a tower defense game. An RPG game is about a hero's journey, while Tower Defense is about strategic defense. I decided to go with the "Tower Defense" game. I like this kind of game more. This design will propose a class named "DefenseTower", a multiset data structure for core logic of Tower Defense game system. The player will manages defense strategies and structures to stop the incoming enemies. As mentioned in the project pdf, multiset (DefenseTower) will support multiple instances of same item e.g. a player likes to have 5 cannons and 2 towers to attack. As mentioned in Table 1, the comparison between Hash Table and AVL Tree bring me to the decision of choosing Hash Table (HashMap)(jstring, unsigned int;) as shown in Figure 1.

Comparison Criteria	Hash Table	AVL Tree
Search Time Complexity	$O(1)$	$O(\log n)$
Insertion Time Complexity	$O(1)$	$O(\log n)$
Deletion Time Complexity	$O(1)$	$O(\log n)$
Memory Overhead	High	Low
Range Searches	Requires special implementation	Efficient
Re-balancing	Not necessary	Required
Recursion	Not Inherently RS	RS
Implementation	Mostly relies on Libraries	Easily Customizable
Suitability for Small Data Sets	Less suitable due to memory overhead	More suitable

Table 1: Comparison of HahsTable vs AVL-Tree[Geea]

Hash Table for game design is perfect because it will check the inventory (game loop Figure 2) thousands of times in a simple game. A Hash Table is the only structure fast enough to do this instantly ($O(1)$), so the game doesn't lag/stutter.

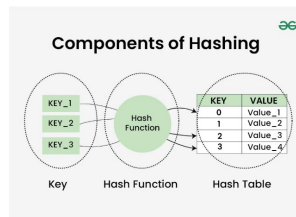


Figure 1: Simple game loop[VCF16]

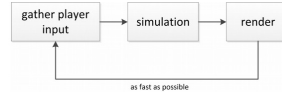


Figure 2: Simple game loop[VCF16]

2 Design Philosophy

There are three primary qualities in designing the DefenseTower: **Efficiency**, **Simplicity**, and **Extensibility**. But before talking about them, let us define the latency, and compaction in the operation system.

Latency is a measurement of delay in a system. Network latency is the amount of time it takes for data to travel from one point to another across a network. The higher the latency, the slower the response times[IBM]. Compaction is a technique to collect all the free memory present in the form of fragments into one large chunk of free memory, which can be used to run other processes. It does that by moving all the processes towards one end of the memory and all the available free space towards the other end of the memory so that it becomes contiguous[Gee24].

2.1 Efficiency:

I believe in a game, latency (response time) is more important than memory compactness. In a tower defense game, operations like checking if a tower is available (contains) or removing a tower (remove) occur frequently.

2.2 Simplicity:

Just imagine: you want to drive a car home. As a driver, is it more important to you how the cylinder works inside the engine or how the oil circulates inside it? The answer is NO. The same thing is happening here, The client code which can be the Game UI or Level Manager, should not need to manage the underlying hashing logic. The interface of the game should only focus on concepts like adding or removing the tower, not low-level data manipulation.

2.3 Extensibility:

It is said that "No single platform can provide everything out of the box to meet business needs. To drive product and quality excellence, the extensibility of the chosen software platform is critical". Therefore, an extensible software platform should be flexible, configurable, customizable, upgradeable, accessible and collaborative[Raz24]. The game is designed to support future expansions, which dynamically add new features like towers without recompiling the core structure.

3 Core Operation:

3.1 `unordered_map< string, int >`

: Instead of using add or insert, I decided to use `unordered_map`. `Unordered_map` is an unordered container that stores data in the form of a unique key-value pair with hashing. This provide average constant-time complexity $O(1)$ for search, insert, and delete operations[Geeb]. If the item in the game already exists as a key in the map, it will update the value of it. If that item in the game is not present, it will create a new key-value.

3.2 `remove< item >`

If the key is in the table, removing it will "erase" the pair of keys of value. This might just be marking a bucket as empty after removal. This provides average constant-time complexity $O(1)$.

3.3 `count< item >`

It will look up a specific item, such as arrows, gold, towers, etc. The time complexity will be $O(1)$.

3.4 `contains< conststring&key >`

It will present a true or false. If true, it means in our game inventory of the game have that item like tower or arch. If false, it is not available. So overall, It will check the availability of that item. The same as before, the time complexity will be $O(1)$.

4 Set Operations:

4.1 `merge_loot(other_items): – (Union_with())`

In this game, we have a permanent inventory. In the game context, we will have some temporary containers, such as rewards from an attack or a cleared wave. This `merge_loot` operation behaves like a Sum Union for multisets. Let me bring an example: if player holds Archer: 3 and obtains a loot crate with Archer:8, Cannon:1, the result will be: Archer:11, Cannon:1. It means this operation combines existing inventory with new inventory and creates a new one.

The time complexity is the number of items in the loot source. Number of items in the hash table means the number of unique types or keys. For clarification, let me bring an example: a loot chest contain 200 arrows and 3000 gold coins. It seems it will check 200 items, but the hash table sees only 2.

As a conclusion for this part, since the loot table is typically small and it is key-wise, not count-wise, combined hash tables will be extremely fast.

References

- [Geea] GeeksforGeeks. Advantages of bst over hash table. <https://www.geeksforgeeks.org/dsa/advantages-of-bst-over-hash-table/>. Accessed: 2025-11-30.
- [Geeb] GeeksforGeeks. Unordered map in c++ stl. https://www.geeksforgeeks.org/cpp/unordered_map-in-cpp-stl/. Last updated 23 July 2025, accessed on 01 December 2025.
- [Gee24] GeeksforGeeks. Compaction in operating system. <https://www.geeksforgeeks.org/operating-systems/compaction-in-operating-system/>, 2024. Accessed: 2025-11-30.
- [IBM] IBM. What is latency?
- [Raz24] Zara Raza. What is extensibility?, March 29 2024. Accessed: 2025-11-30.
- [VCF16] Luis Valente, Aura Conci, and Bruno Feijo. Game loop model properties and characteristics on multi-core cpu and gpu games. In *Proceedings of SBGames*, 2016.