Documentation: Player Equipment Management System

1. Introduction

1.1 Purpose

The purpose of this document is to provide a comprehensive overview of the Player Equipment Management System, including its functionality, features, and usage.

1.2 Document Conventions

- Code Formatting: Code snippets are formatted using monospace font.

- **User Input:** User input is represented within colon `: `.

- **System Output:** System output is presented in regular.

1.3 Intended Audience and Reading Suggestions

This document is intended for developers, testers, and stakeholders involved in the development and deployment of the Player Equipment Management System. It is recommended to read through the entire document to gain a complete understanding of the system.

1.4 Product Scope

The Player Equipment Management System is designed to provide players with a user-friendly interface to manage their in-game inventory and equipment. It offers various functionalities such as buying and selling items, equipping gear, leveling up equipment, and interacting with a virtual shop.

1.5 References

- No external references are currently applicable.

2. Overall Description

2.1 Product Perspective

The Player Equipment Management System operates as a standalone application within the context of a larger gaming environment. It interacts with the game's core mechanics to provide players with inventory management capabilities.

2.2 Product Functions

- Inventory management
- Buying, selling and deleting items
- Equipping gear
- Leveling up equipment
- Interacting with a virtual shop
- Displaying details, statistics and shop

2.3 User Classes and Characteristics

Users of the system include players of the game who wish to manage their inventory and equipment efficiently. They may vary in experience level and familiarity with the game's mechanics.

2.4 Operating Environment

The system is designed to run on platforms compatible with C++ programming language and standard input/output streams.

2.5 Design and Implementation Constraints

- The system is implemented in C++.
- It may have dependencies on specific libraries or system functions for screen clearing and user input/output.

2.6 User Documentation

User documentation may include in-game tutorials, help menus, and tooltips to guide players through the system's functionality.

2.7 Assumptions and Dependencies

The system assumes a basic understanding of gaming concepts such as inventory management and equipment upgrading. Dependencies may include the availability of hardware resources and system libraries.

3. External Interface Requirements

3.1 User Interfaces

- Command-line interface for user interaction
- Menu-based navigation system

3.2 Hardware Interfaces

No specific hardware interfaces are required.

3.3 Software Interfaces

- C++ Standard Library for input/output operations
- System functions for screen clearing

3.4 Communications Interfaces

No external communication interfaces are required.

4. System Features

4.1 Inventory Management

- Move items within the inventory
- Buy and sell items
- Delete items from the inventory

4.2 Equipment Management

- Equip weapons and armor
- Level up items
- View detailed information about items

4.3 Other Management

- Extend equipment

- View shop or players statistics

- Sort inventory

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- System response time should be minimal for smooth user experience.

- Memory usage should be optimized to prevent excessive resource consumption.

5.2 Safety Requirements

- The system should handle user input safely to prevent crashes or unexpected behavior.

5.3 Security Requirements

- No specific security requirements are applicable.

5.4 Software Quality Attributes

- Reliability: The system should function consistently without unexpected errors.

- Maintainability: The codebase should be well-structured and easily modifiable for future updates.

- Usability: The user interface should be intuitive and easy to navigate.

5.5 Business Rules

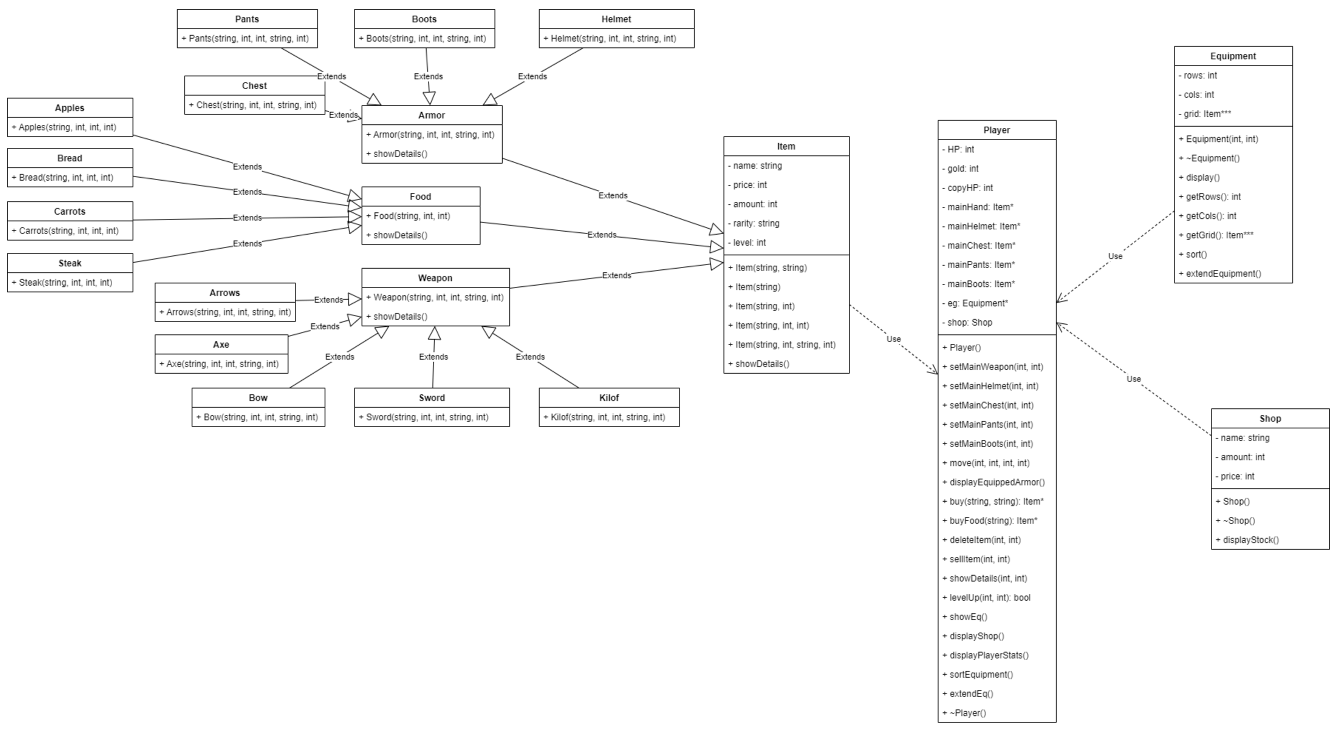
- Transactions should be accurately recorded and reflected in the player's inventory.

- Players should not be able to perform actions that would result in unfair advantages or exploits.

6. Other Requirements

No other specific requirements are currently identified.

Diagram UML


Pants
+ Pants(string, int, int, string, int)
Boots
+ Boots(string, int, int, string, int)
Helmet
+ Helmet(string, int, int, string, int)
Chest
+ Chest(string, int, int, string, int)
Extends
Extends
Extends
Extends
Armor
+ Armor(string, int, int, string, int)
+ showDetails()
Apples
+ Apples(string, int, int, int)
Bread
+ Bread(string, int, int, int)
Carrots
+ Carrots(string, int, int, int)
Steak
+ Steak(string, int, int, int)
Extends
Extends
Extends
Extends
Food
+ Food(string, int, int)
+ showDetails()
Weapon
+ Weapon(string, int, int, string, int)
+ showDetails()
Arrows
+ Arrows(string, int, int, string, int)
Axe
+ Axe(string, int, int, string, int)
Bow
+ Bow(string, int, int, string, int)
Sword
+ Sword(string, int, int, string, int)
Kilof
+ Kilof(string, int, int, string, int)
Extends
Extends
Extends
Extends
Extends
Extends
Extends
Extends
Item
- name: string
- price: int
- amount: int
- rarity: string
- level: int
+ Item(string, string)
+ Item(string)
+ Item(string, int)
+ Item(string, int, int)
+ Item(string, int, string, int)
+ showDetails()
Use
Player
- HP: int
- gold: int
- copyHP: int
- mainHand: Item*
- mainHelmet: Item*
- mainChest: Item*
- mainPants: Item*
- mainBoots: Item*
- eg: Equipment*
- shop: Shop
+ Player()
+ setMainWeapon(int, int)
+ setMainHelmet(int, int)
+ setMainChest(int, int)
+ setMainPants(int, int)
+ setMainBoots(int, int)
+ move(int, int, int, int)
+ displayEquippedArmor()
+ buy(string, string): Item*
+ buyFood(string): Item*
+ deleteItem(int, int)
+ sellItem(int, int)
+ showDetails(int, int)
+ levelUp(int, int): bool
+ showEq()
+ displayShop()
+ displayPlayerStats()
+ sortEquipment()
+ extendEq()
+ ~Player()
Equipment
- rows: int
- cols: int
- grid: Item***
+ Equipment(int, int)
+ ~Equipment()
+ display()
+ getRows(): int
+ getCols(): int
+ getGrid(): Item***
+ sort()
+ extendEquipment()
Use
Use
Shop
- name: string
- amount: int
- price: int
+ Shop()
+ ~Shop()
+ displayStock()