Rune System - Complete Implementation Guide

Overview

Runes are **magical consumable items** with powerful effects. Unlike accessories (which are equipped), runes are single-use items that are consumed when activated.

Rune Types

1. Rune of Return

Effect: Teleports player to last save point **Rarity:** Rare **Cooldown:** 10 seconds **Source:** Craft, Quest **Craftable:** Yes **Tradable:** No (special item) **Dismantlable:** Yes

Crafting Materials:

- 1x Wooden Tablet
- 1x Clay
- 1x Essence
- 1x Verdant Shard

2. Rune of Spawn

Effect: Teleports player to world spawn point **Rarity:** Uncommon **Cooldown:** 5 seconds **Source:** Craft, Grand Trade **Craftable:** Yes **Tradable:** Yes **Dismantlable:** Yes

Crafting Materials:

- 1x Wooden Tablet
- 1x Clay
- 1x Essence

Class Architecture

```
Rune (Core Class)
 ├── RuneType enum (RETURN, SPAWN, ...)
 ├── RuneSource enum (CRAFT, QUEST, DROP, GRAND_TRADE)
 ├── RuneRarity enum (COMMON, UNCOMMON, RARE, EPIC, LEGENDARY)
 ├── Properties (craftable, tradable, dismantlable)
 ├── Crafting materials (Map<String, Integer>)
```

```
└── use() method (applies effect)

RuneManager (Factory)
├── createRune(RuneType) → Rune
├── craftRune(RuneType, inventory) → Rune
├── canCraftRune(RuneType, inventory) → boolean
└── runeToItem(Rune) → Item
```

Usage Examples

Example 1: Create and Use a Rune

```java
// Create rune
Rune runeOfReturn = RuneManager.createRuneOfReturn();

// Use rune
Entity player = gameState.getPlayer();
boolean success = runeOfReturn.use(player, gameState);

if (success) {
    System.out.println("Teleported!");
    // Rune is consumed (single use)
}
```

Example 2: Add Rune to Inventory

```java
// Create rune as Item
Item runeItem = RuneManager.createRuneItem(Rune.RuneType.RETURN);

// Add to inventory
uiManager.addItemToInventory(runeItem);

// Item appears in:
// - "Misc" tab (all items)
// - (Consumables don't appear in other tabs)
```

Example 3: Craft a Rune

```java
```

```java
// Player's inventory
Map<String, Integer> inventory = new HashMap<>();
inventory.put("Wooden Tablet", 2);
inventory.put("Clay", 3);
inventory.put("Essence", 1);
inventory.put("Verdant Shard", 1);

// Check if can craft
boolean canCraft = RuneManager.canCraftRune(
    Rune.RuneType.RETURN,
    inventory
);

if (canCraft) {
    // Craft rune (consumes materials)
    Rune rune = RuneManager.craftRune(
        Rune.RuneType.RETURN,
        inventory
    );

    if (rune != null) {
        System.out.println("Crafted: " + rune.getName());

        // Convert to item and add to inventory
        Item runeItem = RuneManager.runeToItem(rune);
        uiManager.addItemToInventory(runeItem);
    }
}
```

Example 4: Check Crafting Requirements

java

```java
// Get crafting materials needed
Map<String, Integer> materials =
    RuneManager.getCraftingMaterials(Rune.RuneType.RETURN);

System.out.println("Required materials:");
for (Map.Entry<String, Integer> entry : materials.entrySet()) {
    System.out.println("  " + entry.getValue() + "x " + entry.getKey());
}

// Output:
// Required materials:
//   1x Wooden Tablet
//   1x Clay
//   1x Essence
//   1x Verdant Shard
```

Example 5: Dismantle Rune

```java
Rune rune = RuneManager.createRuneOfReturn();

if (rune.isDismantlable()) {
    List<String> rewards = rune.getDismantleRewards();

    System.out.println("Dismantle rewards:");
    for (String reward : rewards) {
        System.out.println("  " + reward);
    }

    // Output:
    // Dismantle rewards (50% of materials):
    //   1x Wooden Tablet (from 1)
    //   1x Clay (from 1)
    //   1x Essence (from 1)
    //   1x Verdant Shard (from 1)
}
```

Example 6: Get Detailed Info

```java
```

```
String info = RuneManager.getRuneInfo(Rune.RuneType.RETURN);
System.out.println(info);

// Output:
// === Rune of Return ===
// Type: RETURN
// Rarity: RARE
//
// A mystical rune that teleports you to your last save point...
//
// Cooldown: 10.0s
// Single use (consumed)
//
// --- Properties ---
// Craftable: Yes
// Tradable: No
// Dismantlable: Yes
// Source: CRAFT
//
// --- Crafting Cost ---
// 1x Wooden Tablet, 1x Clay, 1x Essence, 1x Verdant Shard
```

Integration with Game Systems

1. Inventory System Integration

```java
// Runes appear as consumable items
Item runeItem = ItemManager.createRuneOfReturn();

// Shows in inventory
uiManager.addItemToInventory(runeItem);

// Filter: Consumable items show in "Misc" tab only
// (Can add separate "Consumable" or "Rune" tab if needed)
```

2. Quest Rewards

```java
```

```
// In quest completion
public void giveQuestReward(Entity player) {
    // Create rune with QUEST source
    Rune rune = RuneManager.createRune(
        Rune.RuneType.RETURN,
        Rune.RuneSource.QUEST
    );

    // Add to inventory
    Item runeItem = RuneManager.runeToItem(rune);
    uiManager.addItemToInventory(runeItem);

    System.out.println("Received: " + rune.getName() + " (Quest Reward)");
}
```

3. Monster Drops

```java
// In monster death handler
public void onMonsterDeath(Entity monster) {
    // Random chance to drop rune
    if (Math.random() < 0.05) { // 5% chance
        Item runeItem = ItemManager.createRuneOfSpawn();

        // Drop at monster location
        Position pos = monster.getComponent(Position.class);
        createItemDrop(runeItem, pos.x, pos.y);
    }
}
```

4. Crafting UI

```java
```

```java
// Display crafting recipe
public void showCraftingRecipe(Rune.RuneType type) {
    Rune rune = RuneManager.createRune(type);

    System.out.println("=== Craft " + rune.getName() + " ===");
    System.out.println(rune.getDescription());
    System.out.println();
    System.out.println("Materials Required:");

    Map<String, Integer> materials = rune.getCraftingMaterials();
    for (Map.Entry<String, Integer> entry : materials.entrySet()) {
        int playerHas = getPlayerMaterialCount(entry.getKey());
        String status = playerHas >= entry.getValue() ? "✓" : "✗";

        System.out.println(String.format("  %s %dx %s (have: %d)",
            status, entry.getValue(), entry.getKey(), playerHas));
    }
}
```

5. Use Rune from Inventory

```
java
```

```java
// In UIInventorySlot.onClick() or hotkey
public void useRuneFromInventory(int slotIndex) {
    Item item = inventoryGrid.getItemAtSlot(slotIndex);

    if (item != null && item.getType() == Item.ItemType.CONSUMABLE) {
        // Check if it's a rune (by name for now)
        if (item.getName().contains("Rune of")) {
            // Determine rune type from name
            Rune.RuneType type = getRuneTypeFromName(item.getName());

            // Create and use rune
            Rune rune = RuneManager.createRune(type);
            Entity player = gameState.getPlayer();

            boolean success = rune.use(player, gameState);

            if (success) {
                // Remove from inventory (consumed)
                inventoryGrid.removeItemFromSlot(slotIndex);
                System.out.println("Used " + rune.getName());
            }
        }
    }
}

private Rune.RuneType getRuneTypeFromName(String name) {
    if (name.contains("Return")) return Rune.RuneType.RETURN;
    if (name.contains("Spawn")) return Rune.RuneType.SPAWN;
    return null;
}
```

Extending the System

Add New Rune Type

```
java
```

```java
// 1. Add to RuneType enum
public enum RuneType {
    RETURN,
    SPAWN,
    HASTE  // NEW
}

// 2. Initialize in Rune class
private void initializeRuneProperties() {
    switch (type) {
        case RETURN:
            initializeReturnRune();
            break;
        case SPAWN:
            initializeSpawnRune();
            break;
        case HASTE:  // NEW
            initializeHasteRune();
            break;
    }
}

private void initializeHasteRune() {
    this.cooldown = 30f;
    this.maxStack = 3;  // Can stack 3
    this.consumeOnUse = true;

    if (craftable) {
        craftingMaterials.put("Swift Essence", 1);
        craftingMaterials.put("Wind Crystal", 2);
    }
}

// 3. Implement effect
public boolean use(Entity caster, GameState gameState) {
    // ...
    case HASTE:
        success = useHasteRune(caster, gameState);
        break;
}

private boolean useHasteRune(Entity caster, GameState gameState) {
    Movement movement = caster.getComponent(Movement.class);
    if (movement == null) return false;

    // Apply haste buff
    movement.setHaste(true);

    System.out.println("Haste activated! (3x speed for 30s)");
    return true;
}

// 4. Register in RuneManager
registerRune(
    Rune.RuneType.HASTE,
    "Rune of Haste",
    "Grants incredible speed for 30 seconds.",
    Rune.RuneRarity.RARE,
    true, true, true,
    Rune.RuneSource.CRAFT
);
```

Material System

Crafting Materials Table

Material	Rarity	Source	Used For
Wooden Tablet	Common	Craft/Drop	All runes
Clay	Common	Drop/Gather	All runes
Essence	Uncommon	Drop	All runes
Verdant Shard	Uncommon	Boss drop	Rune of Return

Add Materials to Game

```java
```

```
// In monster loot table
public void generateLoot(Entity monster) {
    // Common drops
    if (Math.random() < 0.4) {
        addItemToInventory(ItemManager.createClay());
    }

    // Uncommon drops
    if (Math.random() < 0.15) {
        addItemToInventory(ItemManager.createEssence());
    }

    // Boss drops
    if (isBoss(monster) && Math.random() < 0.8) {
        addItemToInventory(ItemManager.createVerdantShard());
    }
}
```

Future Enhancements

Stackable Runes

```java
// Modify Rune class to support stacking
public class RuneStack {
    private Rune runeType;
    private int quantity;
    private int maxStack;

    public boolean addRune() {
        if (quantity < maxStack) {
            quantity++;
            return true;
        }
        return false;
    }
}
```

Rune Cooldown System

```java
```

```
// Track rune cooldowns per player
public class RuneCooldownManager {
    private Map<Rune.RuneType, Float> cooldowns = new HashMap<>();

    public boolean canUseRune(Rune.RuneType type) {
        Float cooldown = cooldowns.get(type);
        return cooldown == null || cooldown <= 0;
    }

    public void startCooldown(Rune rune) {
        cooldowns.put(rune.getType(), rune.getCooldown());
    }

    public void update(float delta) {
        for (Rune.RuneType type : cooldowns.keySet()) {
            float remaining = cooldowns.get(type) - delta;
            cooldowns.put(type, Math.max(0, remaining));
        }
    }
}
```

Visual Effects

```java
// Add to Rune.use()
private void spawnTeleportEffect(Entity caster, GameState gameState) {
    Position pos = caster.getComponent(Position.class);

    // Spawn particles
    for (int i = 0; i < 20; i++) {
        spawnParticle(pos.x, pos.y, "teleport_sparkle");
    }

    // Play sound
    AudioManager.play("teleport.wav");

    // Screen flash
    screenFlashEffect(Color.WHITE, 0.5f);
}
```

Summary

The Rune system provides: ✅ **Magical consumable items** with powerful effects ✅ **Crafting system** with material requirements ✅ **Flexible properties** (craftable, tradable, dismantlable) ✅ **Multiple sources** (craft, quest, drop, trade) ✅ **Easy to extend** with new rune types ✅ **Full inventory integration** ✅ **Dismantle for material recovery**