

# Design Document for Monsters and Heroes

Jerin Joseph - U11191999

## 1. Class Architecture Overview

The game is structured using **object-oriented design principles** with a strong emphasis on **inheritance, abstraction, and polymorphism**. To support multiple types of characters, items, tiles, and maps, several abstract and base classes were created. These enable code reuse, scalability for future game expansions, and separation of responsibilities.

### Core Abstract Classes

1. **Entity** (*Common Parent for Hero & Monster*)  
Created to unify shared attributes such as name, level, health, and core stats.  
This allows the combat system to reference common attributes regardless of whether the object is a hero or a monster.
2. **Tile**  
Represents a single location in the world map. Different tile types (e.g., MarketTile, CommonTile) inherit from Tile.
3. **Item**  
Abstract parent for all purchasable/usable objects (Potions, Spells, Weapons, Armor).
4. **Spell**  
An abstract subclass of Item to represent magic-based attacks. IceSpell, FireSpell, and LightningSpell extend this.
5. **GameMap**  
Base class for map generation, movement logic, and rendering.

Each subclass specializes in behavior based on the type of entity, item, or tile.

---

## 2. Character System

### Hero Class

The **Hero** class extends *Entity* and contains additional attributes:

- Strength
- Dexterity
- Agility
- Mana
- Money
- Experience
- Inventory (Weapons, Armor, Potions, Spells)

Heroes can:

- Attack
- Cast spells
- Use potions
- Equip weapons and armor
- Gain experience and level up

Different hero types (Warrior, Paladin, Sorcerer) override stat scaling to emphasize specific strengths.

### Monster Class

The **Monster** class extends *Entity* and contains:

- Base damage
- Defense

- Dodge chance

Monster subclasses (Dragon, Exoskeleton, Spirit) modify these stats based on their type.

---

## 3. Inventory & Item System

### Item Hierarchy

All items inherit from **Item**.

This allows uniform handling of price, required level, and name.

#### Weapon

- Damage value
- Required hands (1H or 2H)
- Equip/unequip logic

#### Armor

- Damage reduction value
- Equip/unequip behavior

#### Potion

- Affects multiple attributes
- Stored as a string "Strength\_Dexterity\_Agility" which is parsed during use
- The updated version supports applying effects to **multiple attributes simultaneously**

#### Spells

Abstract Spell → **FireSpell**, **IceSpell**, **LightningSpell**

Each spell has:

- Base damage
  - Mana cost
  - Secondary effect (e.g., reduce defense, reduce dodge, reduce damage)
- 

## 4. Map & Tile System

### Tile Classes

- **CommonTile**: For exploration and random monster encounters
- **MarketTile**: Player can buy or sell items
- **InaccessibleTile**: Movement blocked

Tile classes encapsulate rendering logic, accessibility, and interactions.

### Map Generation

GameMap generates:

- Randomized accessible tiles
- Markets placed at random
- Inaccessible tiles to create obstacles

Movement validation ensures:

- Player cannot move to invalid or blocked tiles
  - Random encounters occur only on CommonTiles
-

## 5. Combat System

The combat system is turn-based and supports:

- Basic attacks
- Spell casting
- Using potions
- Equipping gear mid-battle

Combat flow:

1. Player selects action
2. Enemy responds
3. Stats update dynamically based on spells, potions, or gear changes

Damage calculation considers:

- Weapon damage
- Monster defense
- Hero dexterity scaling (for spells)
- Armor reduction
- Dodge probability

---

## 6. Market System

The MarketTile contains:

- A list of sellable items

- Logic for purchase (checking money + level requirement)
- Logic for selling items from inventory

The hero inventory is updated immediately and rendered back to the player.

---

## 7. Input Handling & Validation

- All player input is validated for correctness
  - Menu choices, movement keys, and combat choices default safely when incorrect input is detected
  - For name input, a default player name is assigned if user enters an empty string
- 

## 8. Scalability

The system is designed to be **easily extendable**:

### Shared Abstract Classes Enable Modular Growth

- Adding new hero types requires only a new subclass of Hero
- Adding new monsters or spells only requires extending Monster or Spell
- New tile types can be added without affecting map logic
- Inventory system easily supports new Item subclasses

### Decoupled Logic Improves Maintainability

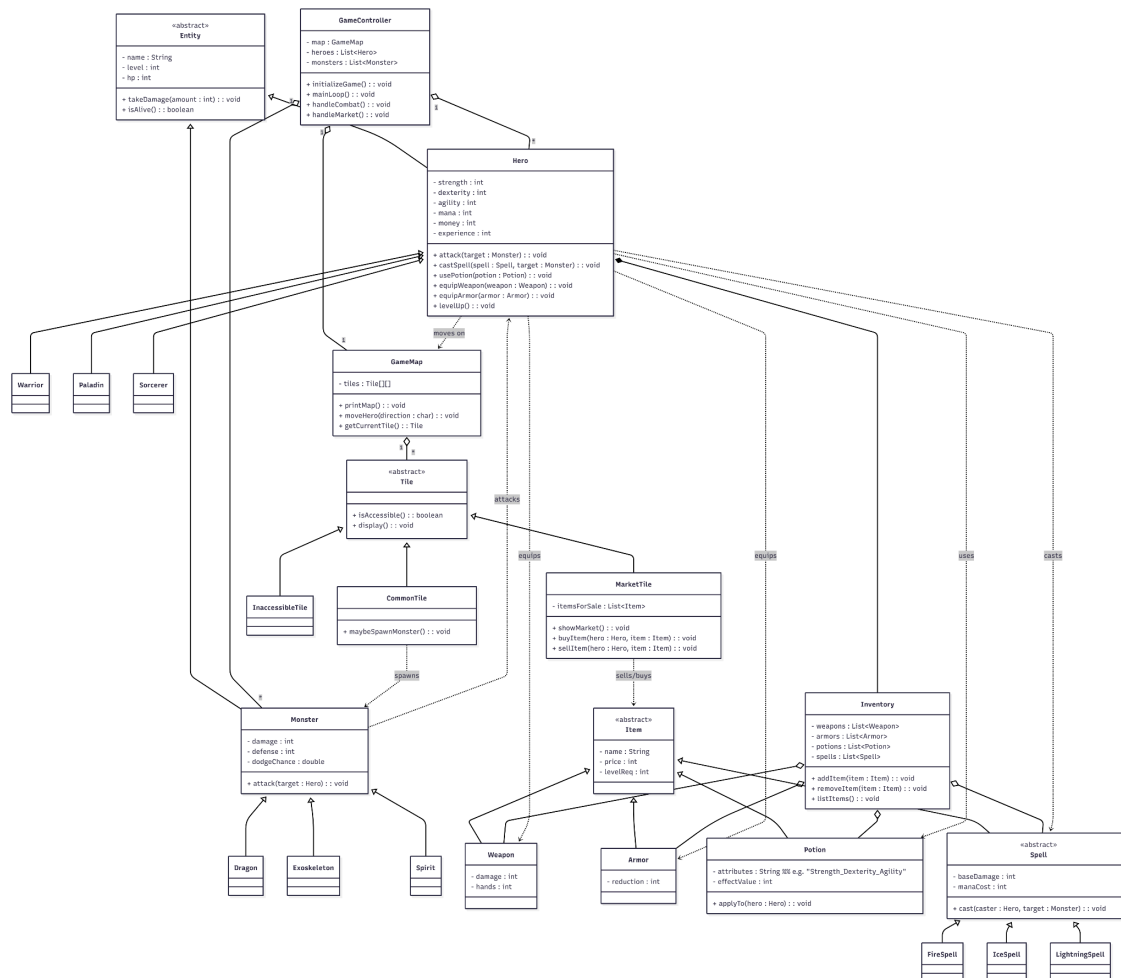
- Combat logic does not depend on hero or monster concrete classes
- Map generation is independent of item or combat classes
- Tile behavior is independent of the entities walking over them

## Why the Design Scales Well

- Clear inheritance structure
- Single Responsibility Principle
- Reusable interfaces and methods
- Minimal hard-coding

The project can support new mechanics (quests, bosses, parties, crafting) without refactoring core code.

## 9. UML Diagram



## Description of Diagram

classDiagram

```
class Entity {  
    <<abstract>>  
    - name : String  
    - level : int  
    - hp : int  
    + takeDamage(amount : int) : void  
    + isAlive() : boolean  
}
```

```
class Hero {  
    - strength : int  
    - dexterity : int  
    - agility : int  
    - mana : int  
    - money : int  
    - experience : int  
    + attack(target : Monster) : void  
    + castSpell(spell : Spell, target : Monster) : void  
    + usePotion(potion : Potion) : void  
    + equipWeapon(weapon : Weapon) : void  
    + equipArmor(armor : Armor) : void  
    + levelUp() : void  
}
```

```
class Monster {  
    - damage : int  
    - defense : int  
    - dodgeChance : double  
    + attack(target : Hero) : void  
}
```

```
Entity <|-- Hero  
Entity <|-- Monster  
class Warrior  
class Paladin  
class Sorcerer
```

```
Hero <|-- Warrior  
Hero <|-- Paladin
```



```
Hero <|-- Sorcerer
class Dragon
class Exoskeleton
class Spirit
```

```
Monster <|-- Dragon
Monster <|-- Exoskeleton
Monster <|-- Spirit
class Inventory {
    - weapons : List~Weapon~
    - armors : List~Armor~
    - potions : List~Potion~
    - spells : List~Spell~
    + addItem(item : Item) : void
    + removeItem(item : Item) : void
    + listItems() : void
}
```

```
class Item {
    <<abstract>>
    - name : String
    - price : int
    - levelReq : int
}
```

```
class Weapon {
    - damage : int
    - hands : int
}
```

```
class Armor {
    - reduction : int
}
```

```
class Potion {
    - attributes : String %% e.g. "Strength_Dexterity_Agility"
    - effectValue : int
    + applyTo(hero : Hero) : void
}
```

```
class Spell {
    <<abstract>>
```

```

    - baseDamage : int
    - manaCost : int
    + cast(caster : Hero, target : Monster) : void
}

```

```

class FireSpell
class IceSpell
class LightningSpell

```

```

Item <|-- Weapon
Item <|-- Armor
Item <|-- Potion
Item <|-- Spell

```

```

Spell <|-- FireSpell
Spell <|-- IceSpell
Spell <|-- LightningSpell
Hero *-- Inventory
Inventory o-- Weapon
Inventory o-- Armor
Inventory o-- Potion
Inventory o-- Spell
class Tile {
    <<abstract>>
    + isAccessible() : boolean
    + display() : void
}

```

```

class CommonTile {
    + maybeSpawnMonster() : void
}

```

```

class MarketTile {
    - itemsForSale : List~Item~
    + showMarket() : void
    + buyItem(hero : Hero, item : Item) : void
    + sellItem(hero : Hero, item : Item) : void
}

```

```

class InaccessibleTile

```

```

Tile <|-- CommonTile

```

Tile <|-- MarketTile

Tile <|-- InaccessibleTile

```
class GameMap {  
    - tiles : Tile[]  
    + printMap() : void  
    + moveHero(direction : char) : void  
    + getCurrentTile() : Tile  
}
```

GameMap "1" o-- "\*" Tile

```
class GameController {  
    - map : GameMap  
    - heroes : List~Hero~  
    - monsters : List~Monster~  
    + initializeGame() : void  
    + mainLoop() : void  
    + handleCombat() : void  
    + handleMarket() : void  
}
```

GameController "1" o-- "1" GameMap

GameController "1" o-- "\*" Hero

GameController "1" o-- "\*" Monster

Hero ..> Weapon : equips

Hero ..> Armor : equips

Hero ..> Potion : uses

Hero ..> Spell : casts

Hero ..> GameMap : moves on

Monster ..> Hero : attacks

CommonTile ..> Monster : spawns

MarketTile ..> Item : sells/buys