Data Bases Project

Submission number 1

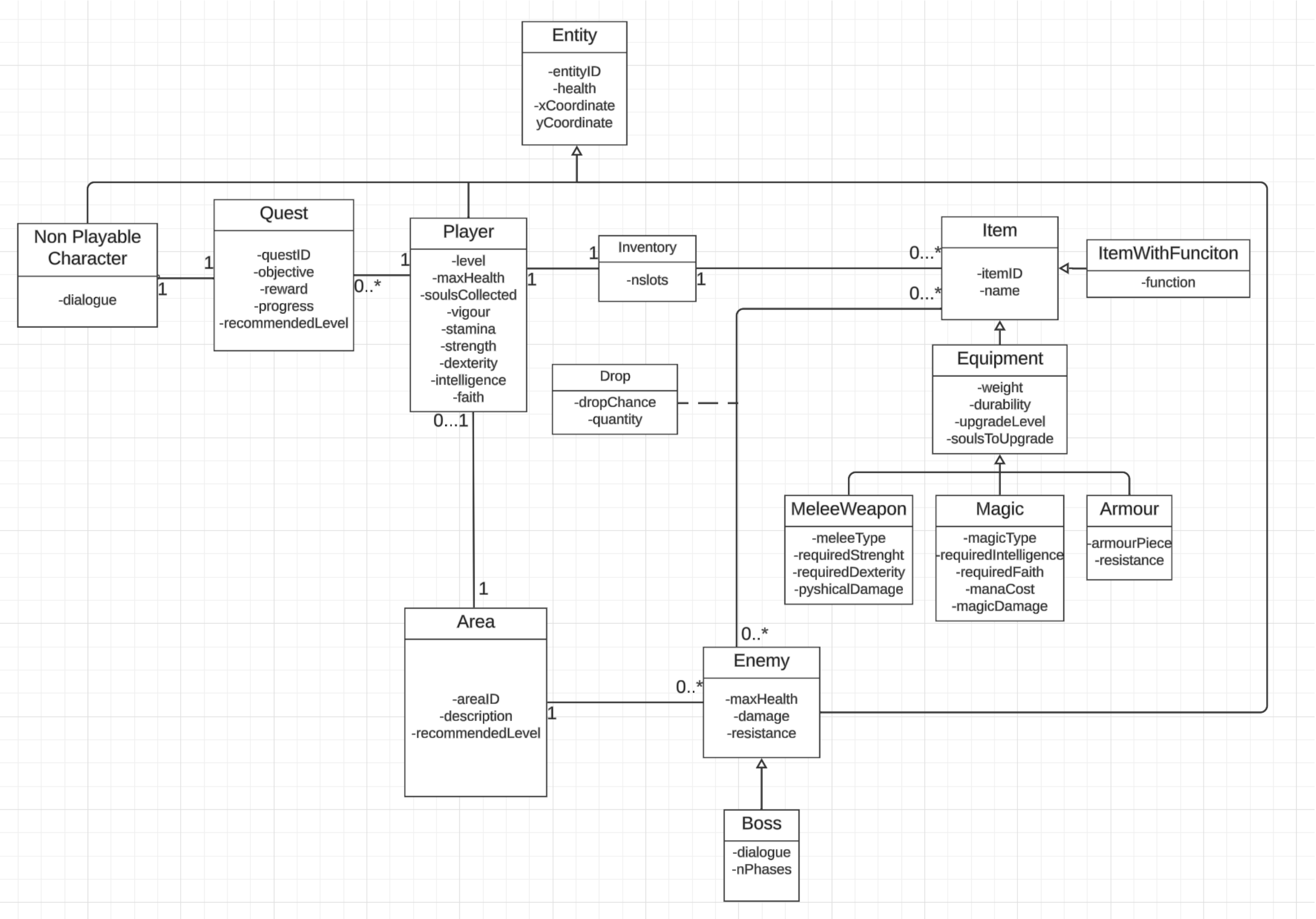
Group 14_05:
Diogo Sousa Campeão, up202307177
Hugo Miguel Gomes Silva, up202307383
Tomás Costa Barros, up202303664

TASK 4.1.1:

The project is designed to track essential aspects of a SoulsLike game. All information is divided into four main categories: Entities, Areas, Items or Events.

For our Entities, we need to know that every Entity in the game has its ID, a current health, a location on x, and a location on y. The entity we control in the game is the Player. For the player, it should store information such as level, its max health, the number of souls it collected (currency of the game), and base attributes such as vigour, stamina, strength, dexterity, intelligence and faith.
Next, the Non-Playable Character (or NPC). This Entity is responsible for giving Quests and giving tips or crucial information for the game progression to the player. Each NPC has a dialogue when you interact with them and should give a Quest.
The Quest should have an ID, an objective, a reward, the current progress of the Quest and the recommended level for the quest completion. The Enemy is also an Entity, and it can appear on various situations of the game, being hostile and attacking the player, when sighted. It should store its max health, its resistance and its damage output. Bosses are a specific type of enemy, much dangerous and only required to defeat once, dictating crucial progress in the game. The bosses should have a dialogue when you enter the boss arena and record the number of phases of the boss fight, determining the amount of different attack patterns and aggressiveness.
The Player can walk into Areas in the game, each area having an ID, a brief description of the Area, and a recommended level to have a normal experience in that Area. Every area has a set of different Enemies.
Player and Enemy both possess items. Every Item should have an ID and its name. Players have an Inventory, which carries a determined number of Items. Every item in the game should be either an Equipment or a general Item with a function. Each piece of Equipment should track weight, durability, the current level and the required souls to upgrade it, and it should be one of three types: Melee, Magic or Armour. Melee should have a type (sword, axe, etc), the requirements for using it, strength and dexterity, and its physical damage output. Magic should also have a type (enchantment, damage magic, etc), required stats, intelligence and faith, the mana cost for the spell and its magical damage output. Armour should track the piece of armour (helmet, plate, etc) and the resistance it gives the entity. An Item with function should only track its functionality (crafting, keys, etc). Enemies at death can drop Items, each having a different percentage of dropping (it can be zero.)

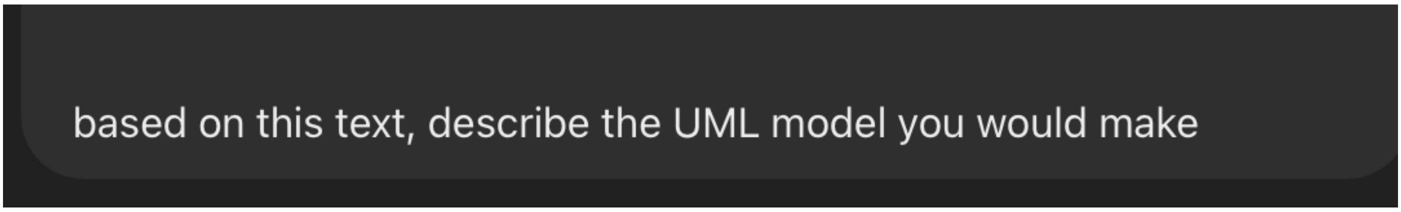
TASK 4.1.2.



TASK 4.1.3.

AI Used – ChatGPT

Prompt 1: We gave the text from 4.1.1. and asked to make the UML model (only the description).



based on this text, describe the UML model you would make

(The rest of the prompt is the text from the task 4.1.1.)

We did this to verify if our text would make a valid Diagram and if the AI could interpret it the way we want. Since the response was fairly satisfactory, we did not need to give specific instructions on how to construct the UML Diagram.

Response 1:

To model this SoulsLike game in UML, we can use the following classes and relationships based on your description. The UML design is organized into four main categories: Entities, Areas, Items, and Events (Quests).

1. Entities

The core entity of the game is the Entity superclass, which is inherited by specific entities like Player, NPC, Enemy, and Boss.

- **Entity (abstract)**
    - Attributes:
        - entityID: int
        - currentHealth: int
        - xCoordinate: int
        - yCoordinate: int
    - Relationships:
        - Aggregates Items
- **Player (inherits from Entity)**
    - Attributes:
        - level: int
        - maxHealth: int
        - soulsCollected: int
        - vigour: int
        - stamina: int
        - strength: int
        - dexterity: int
        - intelligence: int
        - faith: int
    - Relationships:
        - Aggregates Inventory
        - Can have zero or more Quests (association)
- **NPC (inherits from Entity)**
    - Attributes:
        - dialogue: String
    - Relationships:
        - Offers exactly one Quest (composition)
- **Enemy (inherits from Entity)**
    - Attributes:
        - maxHealth: int
        - resistance: int
        - damageOutput: int
    - Relationships:
        - Aggregates Items (as potential drops with drop rates)

- **Boss (inherits from Enemy)**

    - Attributes:
        - dialogue: String
        - nPhases: int (number of phases during battle)

2. Areas

Each Area is an independent section of the game world and contains various enemies. Players traverse these areas as they progress in the game.

- **Area**
    - Attributes:
        - areaID: int
        - description: String
        - recommendedLevel: int
    - Relationships:
        - Contains many Enemies (aggregation)
        - Can be visited by Player (association)

3. Items

Items are divided into Equipment (which includes specific subclasses for Melee, Magic, and Armour) and general Itemswith a function. These items are carried in a player's Inventory and can be dropped by Enemies.

- **Item (abstract)**
    - Attributes:
        - itemID: int
        - name: String
    - Relationships:
        - Can be contained in Inventory (association)
        - Can be dropped by Enemies (association with drop rate)
- **Equipment (inherits from Item, abstract)**
    - Attributes:
        - weight: float
        - durability: int
        - level: int
        - soulsRequiredToUpgrade: int
- **Melee (inherits from Equipment)**
    - Attributes:
        - meleeType: String (sword, axe, etc.)
        - requiredStrength: int
        - requiredDexterity: int
        - physicalDamage: int
- **Magic (inherits from Equipment)**
    - Attributes:
        - magicType: String (enchantment, damage, etc.)
        - requiredIntelligence: int
        - requiredFaith: int
        - manaCost: int
        - magicalDamage: int
- **Armour (inherits from Equipment)**

    - Attributes:
        - armourType: String (helmet, plate, etc.)
        - resistance: int

- **FunctionalItem (inherits from Item)**
    - Attributes:
        - functionality: String (crafting, key, etc.)
- **Inventory**
    - Attributes:
        - slots: int
    - Relationships:
        - Holds many Items (composition with Player)

4. Events (Quests)

Quests are provided by NPCs and guide the player throughout the game. A quest has multiple stages, rewards, and a recommended level.

- **Quest**
    - Attributes:
        - questID: int
        - objective: String
        - reward: String
        - progress: int
        - recommendedLevel: int
    - Relationships:
        - Linked to Player and NPC

Key Relationships:

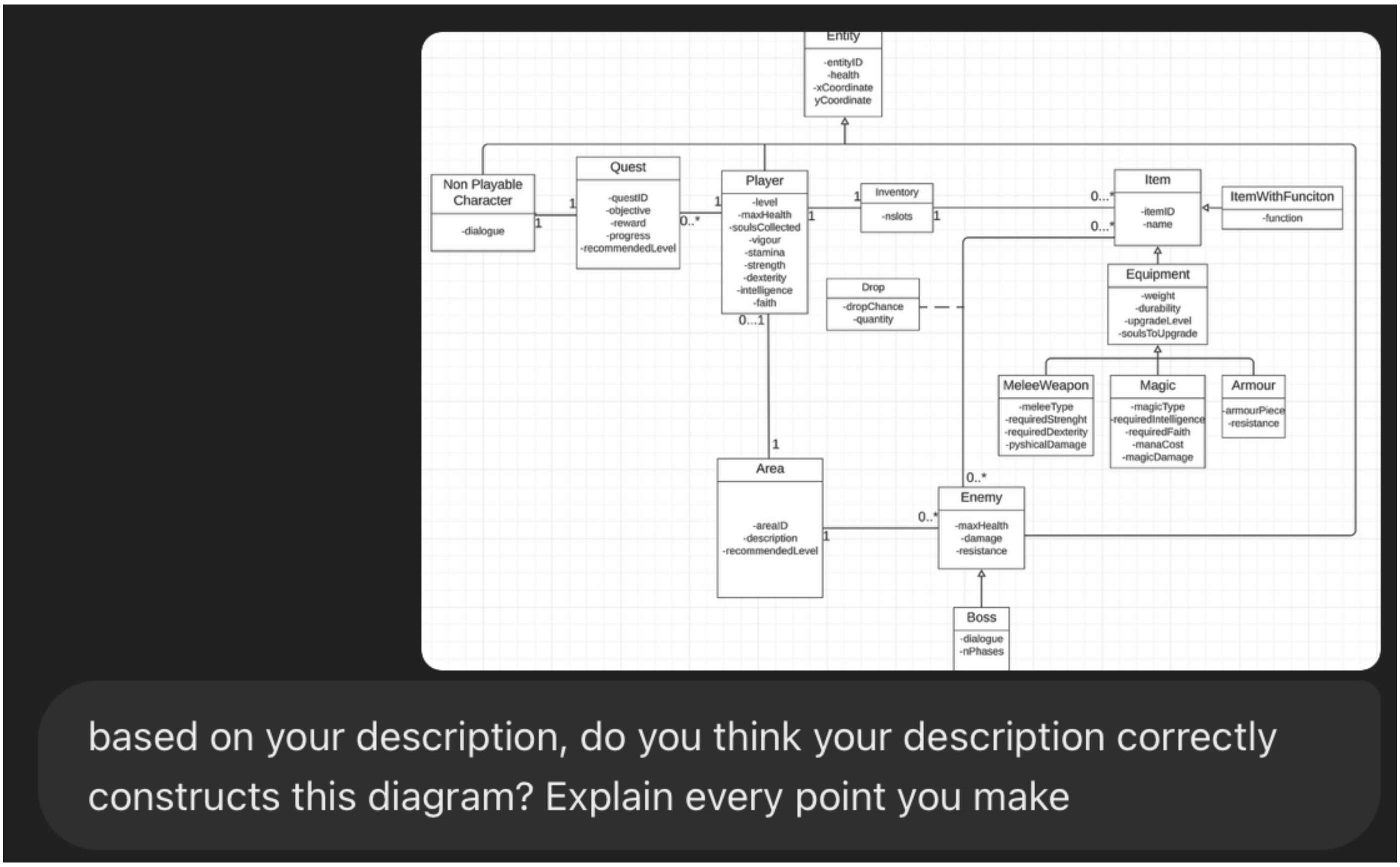
- **Inheritance**:
    - Entity → Player, NPC, Enemy
    - Enemy → Boss
    - Item → Equipment, FunctionalItem
    - Equipment → Melee, Magic, Armour
- **Associations**:
    - Player has many Items through Inventory.
    - Area contains many Enemies.
    - Enemy has a potential drop list of Items with drop rates.
- **Compositions**:
    - NPC has exactly one Quest.
    - Player has Inventory which contains Items.

UML Model Breakdown:

- **Classes**: Entity, Player, NPC, Enemy, Boss, Area, Item, Equipment, Melee, Magic, Armour, FunctionalItem, Inventory, Quest
- **Relationships**: Inheritance, Composition, and Aggregation are used extensively, as described above.

Prompt 2:

Next, we provided the Diagram we did in Task 4.1.2. to see if the AI could reproduce the same diagram we did. Then we asked if, based on the UML description the AI gave us, to compare the diagram it generated to what we did, and provide comments on it.



based on your description, do you think your description correctly constructs this diagram? Explain every point you make

Response 2:

Let's go over the UML diagram to evaluate if it aligns with the design described in the earlier explanation.

Entities

1. **Entity:**
    - Diagram: entityID, health, xCoordinate, yCoordinate.
    - Description: Matches the provided description, as every entity in the game has an ID, health, and location (x, y coordinates).

- **Player:**
    - Diagram: level, maxHealth, soulsCollected, vigour, stamina, strength, dexterity, intelligence, faith.
    - Description: This correctly reflects the attributes from the description: Player has stats like level, max health, and base attributes (vigour, stamina, etc.). The relationship between the Player and Quest is a 0..*relationship, which is correct since a player can have multiple quests.

The relationship with Inventory is 1..1, which indicates a player always has an inventory, which is consistent.

- Non Playable Character (NPC):
    - Diagram: dialogue.
    - Description: NPC has dialogue and is linked to exactly one Quest (1..1), which is correct. This matches the earlier description.
- Enemy:
    - Diagram: maxHealth, damage, resistance.
    - Description: Matches well. Enemies have attributes like max health, damage, and resistance as described. The relationship with Area (0..*) means that multiple enemies can exist in an area, which aligns with the description. Also, the Drop relationship (dashed line) indicates that enemies can drop items, with a drop chance, which is correctly modeled.
- Boss:
    - Diagram: dialogue, nPhases.
    - Description: Correct. The Boss inherits from Enemy, and the addition of dialogue and nPhases captures the boss's specific characteristics, including phases in the battle.

- **Areas**

- Area:
    - Diagram: areaID, description, recommendedLevel.
    - Description: Matches the earlier description of the Area. Each area has an ID, a description, and a recommended level, and the relationship with Enemy (0..*) is correct, as an area can have multiple enemies.

- **Items**

- Item:
    - Diagram: itemID, name.
    - Description: Matches the description. Every item has an ID and a name.
- Equipment:
    - Diagram: weight, durability, upgradeLevel, soulsToUpgrade.
    - Description: This matches the description. The attributes such as weight, durability, upgrade level, and required souls to upgrade are consistent with the earlier explanation.
- MeleeWeapon:
    - Diagram: meleeType, requiredStrength, requiredDexterity, physicalDamage.
    - Description: This is correct. Melee weapons have types, and the attributes like required strength, dexterity, and physical damage are represented.
- Magic:
    - Diagram: magicType, requiredIntelligence, requiredFaith, manaCost, magicDamage.
    - Description: This matches the description of the Magic equipment class, which tracks attributes related to magic type, required intelligence, faith, mana cost, and damage.
- Armour:
    - Diagram: armourPiece, resistance.

    - Description: This is correct, as armour pieces provide resistance and their types are represented as armourPiece.

- ItemWithFunction:
    - Diagram: function.
    - Description: Correct, as per the description. Items with specific functionality (crafting, keys, etc.) only track their function.

- Inventory:
    - Diagram: nslots.
    - Description: Correct, as the inventory tracks the number of slots and is associated with Item (0..*) indicating it can hold multiple items.

- **Events (Quests)**

- Quest:
    - Diagram: questID, objective, reward, progress, recommendedLevel.
    - Description: Correct. A quest has attributes such as ID, objective, reward, progress, and a recommended level, and it is correctly linked to both Player (0..*) and NPC (1..1).

- **Other Relationships**

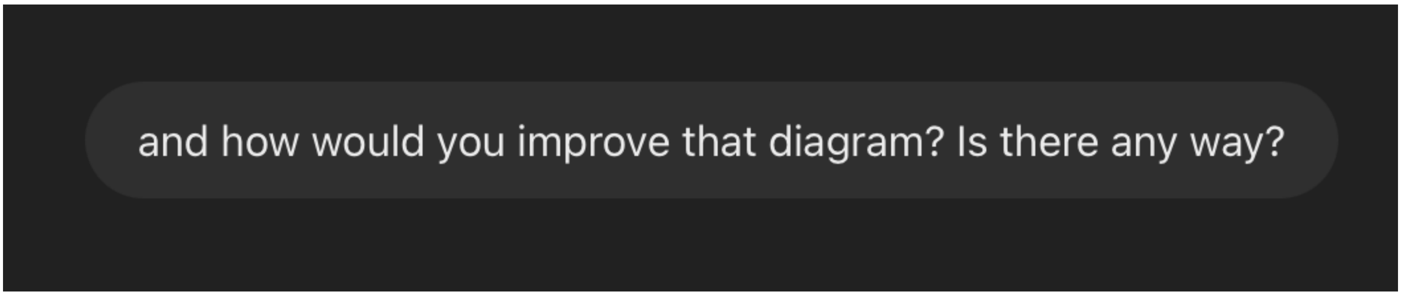
- Drop:
    - Diagram: dropChance, quantity.
    - Description: Correct. The Drop class models the relationship where enemies can drop items with a specific drop chance and quantity.

Conclusion

The diagram is well-constructed and matches the description provided earlier. The classes, attributes, and relationships are properly represented, capturing the core aspects of Entities, Areas, Items, and Quests. The use of inheritance and associations is appropriate, and the multiplicities (e.g., 0..* for many items or enemies) are correct. There are no major discrepancies between the diagram and the textual description.

Prompt 3:

Since the description and comparison of the model was between our expectations, we then asked the AI to, based on the general theme of the Project, if it could improve or change anything in the model. We were looking for more accurate classes and more functionalities in the model to make it more complete.



and how would you improve that diagram? Is there any way?

Response 3:

The current diagram is well-structured and clearly models the core aspects of the SoulsLike game. However, there are a few improvements and refinements that could enhance clarity and modularity, while making the diagram more flexible for future extensions. Here are some suggestions for improvement:

1. Add Abstract Classes for Specific Item Types

To better represent the hierarchy and reduce redundancy for equipment types, you could add an **abstract class** for Weapon(inheriting from Equipment) and have MeleeWeapon and Magic inherit from it.

- **Weapon (abstract)**:
    - Attributes:
        - requiredStrength: int
        - requiredDexterity: int
        - damage: int
    - This would centralize the attributes that both melee and magic share (like damage, strength/dexterity), and allow for future weapon types, making the system more extendable.

New Inheritance Structure:

- Item → Equipment → Weapon → (MeleeWeapon, Magic)

2. Refine Equipment Inheritance

In the current model, Magic and MeleeWeapon both inherit directly from Equipment, but Magic isn't really "equipment" in the traditional sense, and it might make more sense to treat it as a type of spellcasting tool. The distinction between magic as an item and equipment could be made clearer by introducing a **Spell** class that tracks attributes like mana cost and magic damage.

- **Spell** (New Class):
    - Attributes:
        - spellType: String
        - requiredIntelligence: int
        - requiredFaith: int
        - manaCost: int
        - magicDamage: int
    - **Equipment** and **Spell** would both inherit from Item instead of having Magic as an Equipment.

3. Introduce an AttackPattern for Boss Phases

Since bosses have phases, adding an AttackPattern class to handle the different strategies and behaviors for each phase would be a meaningful addition. This class could store different actions or movements for each phase.

- **AttackPattern**:
    - Attributes:
        - patternName: String
        - phaseNumber: int
        - behavior: String

    - Relationship: Boss → (has many) AttackPattern

This can help in modeling different patterns for bosses and can be extended for complex behaviors.

4. Normalize NPC-Quest Relationship

Currently, the NPC is linked to exactly one Quest. While that makes sense for simplicity, some NPCs might offer multiple quests over time, especially in complex games. You could change the relationship to **0..*** or **1..*** between NPC and Quest, to allow NPCs to offer multiple quests or none.

- **New Relationship**: NPC → (has many) Quest

5. Refine Drop Mechanism

In the current diagram, the Drop class manages items that can be dropped by enemies, but there might be an improvement in how the percentage and rarity of items are handled:

- Add a **DropRarity** attribute or class to differentiate between common, rare, and unique drops.
- **DropRarity**:
    - Attributes:
        - rarityType: String (common, rare, epic, etc.)
        - rarityPercentage: float

This could be linked to Drop to add more detail for game balancing.

6. Introduce the Resistance System for Armour

Armour pieces in the current design have a single resistance attribute, but in many games, different armours provide varying resistances against types of damage (e.g., physical, magical, elemental).

- **Armour** could be expanded with attributes like:
    - physicalResistance: int
    - magicResistance: int
    - elementalResistance: int

This adds more flexibility to the types of challenges the player faces and the gear they need.

7. Add Interfaces for Combat and Interactions

To make the model more modular and adaptable for future updates, adding interfaces for combat and interactions between entities could help. For instance:

- **ICombatant (interface)**:
    - Methods:
        - attack()
        - defend()
    - Can be implemented by Player, Enemy, Boss

This abstraction allows all combat-related entities to follow the same rules for attacking and defending without hardcoding their behaviors.

8. Add a Map or World Class

Since areas are a significant part of the game world, introducing a Map or World class that aggregates multiple Areaswould help scale the design for larger worlds. This could be a useful layer between Player and Area.

- **Map/World**:
  - Attributes:
    - mapID: int
    - name: String
  - Relationship: Map → (has many) Area

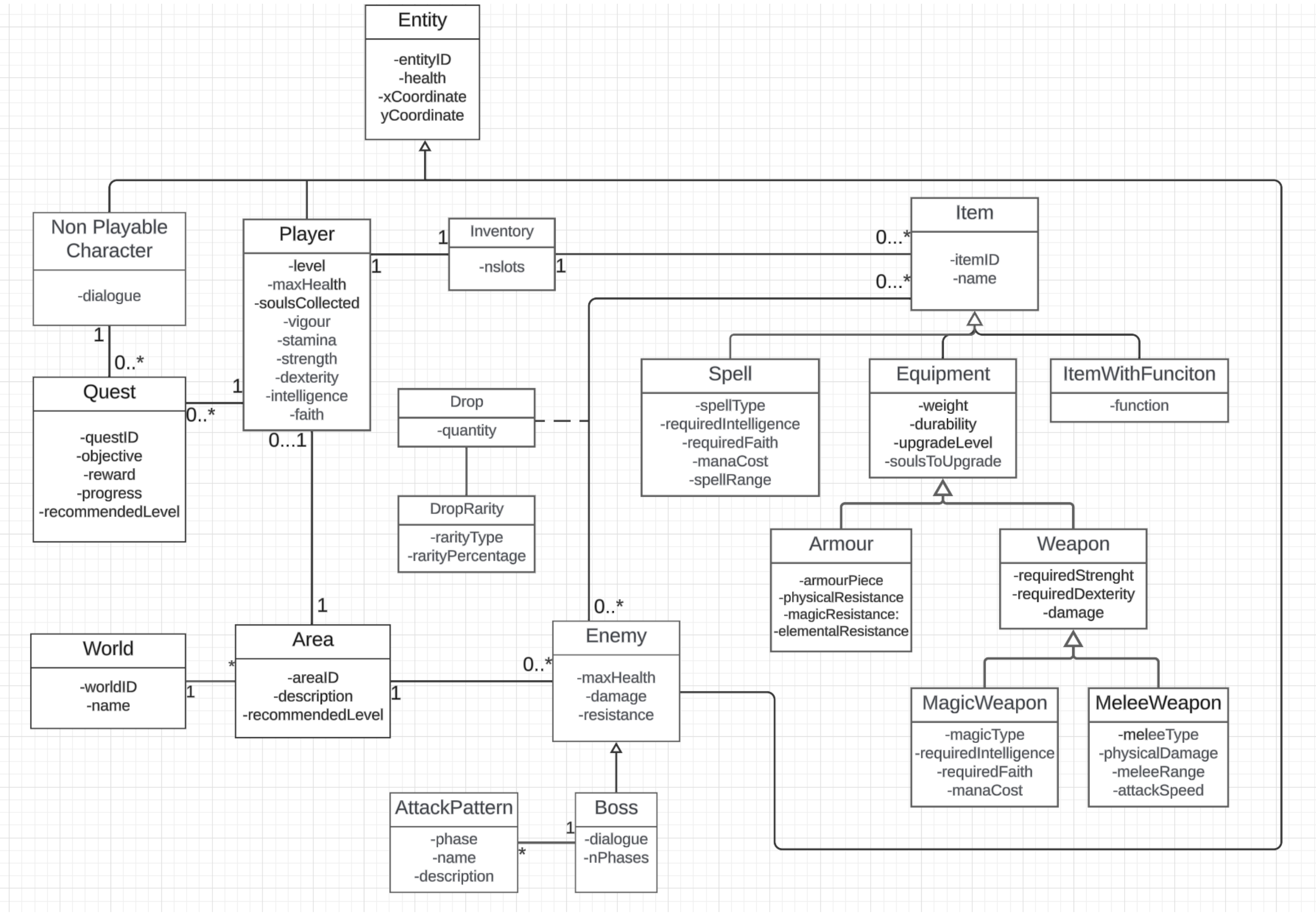
This addition allows for larger, interconnected areas and can model the transition between different zones or regions.

Improved UML Overview:

1. **Item → Equipment**, **Spell**, **ItemWithFunction**
2. **Weapon** as an abstract class between Equipment and MeleeWeapon/Magic
3. Refined **NPC–Quest** relationship (1..*)
4. Add **AttackPattern** for Boss phases
5. Add **DropRarity** to fine-tune item drops
6. Add more detailed resistances in **Armour**
7. Introduce **ICombatant** interface for combat consistency
8. Introduce a **Map** or **World** class for area aggregation

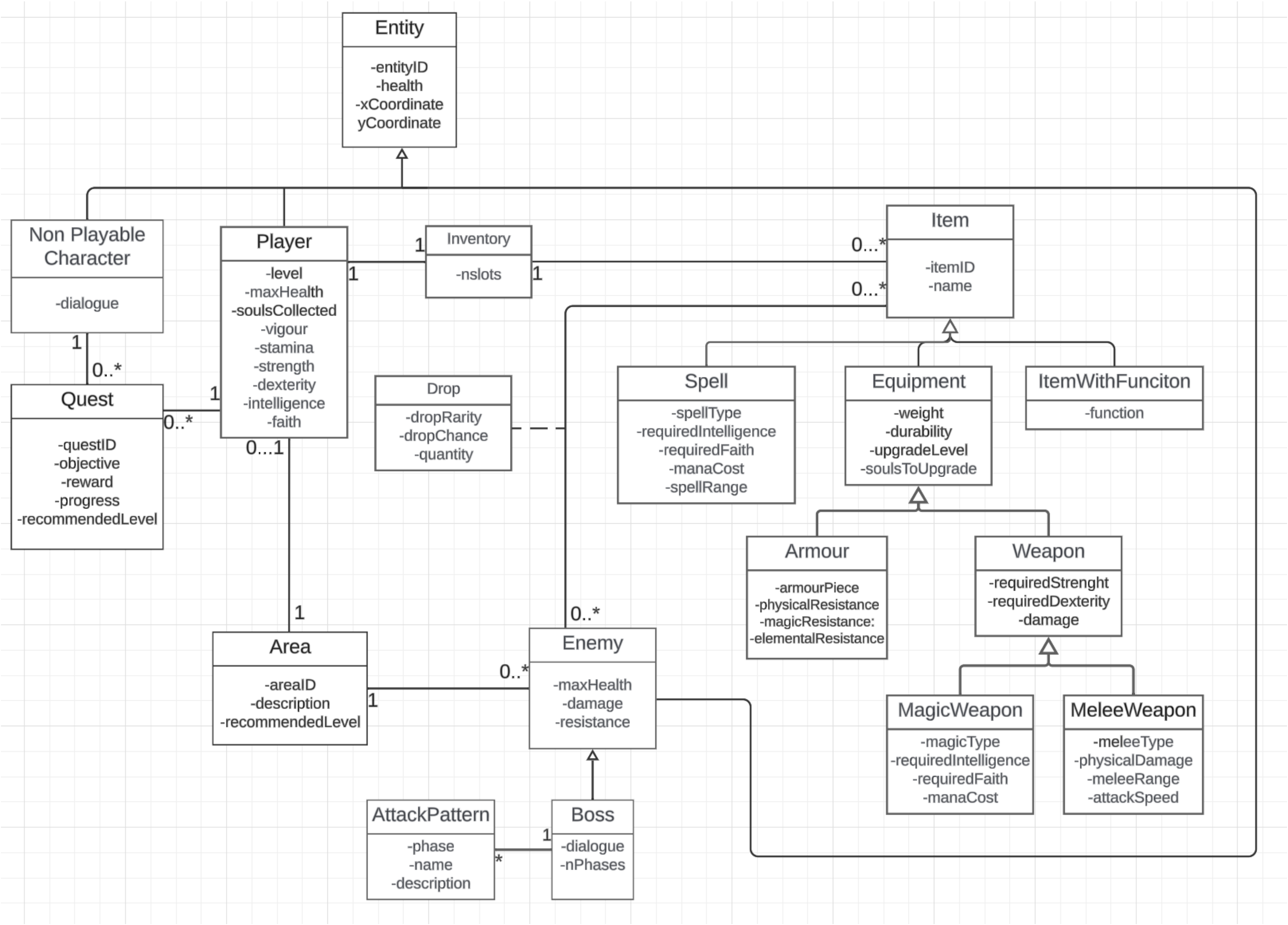
These changes will make the model more flexible, modular, and scalable while capturing more intricate game mechanics.

Conclusion:



We couldn’t add point 7 since we didn’t implement interfaces or methods on our diagram. Other than that, this diagram is the closest we could reproduce from the AI description.

TASK 4.1.4.



We ignored points 7 and 8, since we didn't implement interfaces at all, and we are making the Database such as there is only one map to be recorded, so a single class for a single object wouldn't be necessary.

We went along fully with points 1, 2, 3, 4 and 6, implementing completely the AI suggestions. We added 3 new Classes: Weapon, Spell and AttackPattern.
We also changed the name of Magic to MagicWeapon, since that with the addition of Spell, it would be redundant to have 2 Classes with identical names. MagicWeapon, for example, would be like a "Fire Sword", that is technically a Weapon but it has fire magic, and Spell would be for a castable spell, such as a "Fire Ball".

We went along partially with point 5, not necessarily adding a new class, but adding a new attribute for the Drop association Class: dropRarity.