

SoulsLike Database

Final report of the Course Unit “Bases de Dados”

Diogo Sousa Campeão, up202307177
Hugo Miguel Gomes Silva, up202307383
Tomás Costa Barros, up202303664

Group 14_05



Universidade do Porto

Faculdade de Engenharia

FEUP

Licenciatura em Engenharia Informática e Computação
Lecturer: Lázaro Gabriel Barros da Costa

December 2024

Index

Task 4 - Conceptual Modelling	3
4.1.1. Domain definition	3
4.1.2. Initial Conceptual Modelling	4
4.1.3. Generative AI Integration	4
4.1.4. Final Conceptual Modelling	5
Task 5 – Relational Modelling, Database Creation and Data Population	7
5.1.1 Conceptual Model Refining	7
5.1.10. Final SQLite Database Creation and Data Loading	14
5.1.2. Initial Relational Schema	8
5.1.3 Generative AI Integration	9
5.1.4 Final Relational Schema	10
5.1.5 Initial Analysis of Functional Dependencies and Normal Forms	11
5.1.6. Generative AI Integration for Functional Dependencies and Normal Forms Analysis	12
5.1.7. Final Analysis of Functional Dependencies and Normal Forms	12
5.1.8.Initial SQLite Database Creation	12
5.1.9. Generative AI Integration on SQL Creation and Population	13
Task 6 – Final considerations	15
6.1. Generative AI Integration	15
6.2. Participations	15

Task 4 - Conceptual Modelling

4.1.1. Domain definition

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.

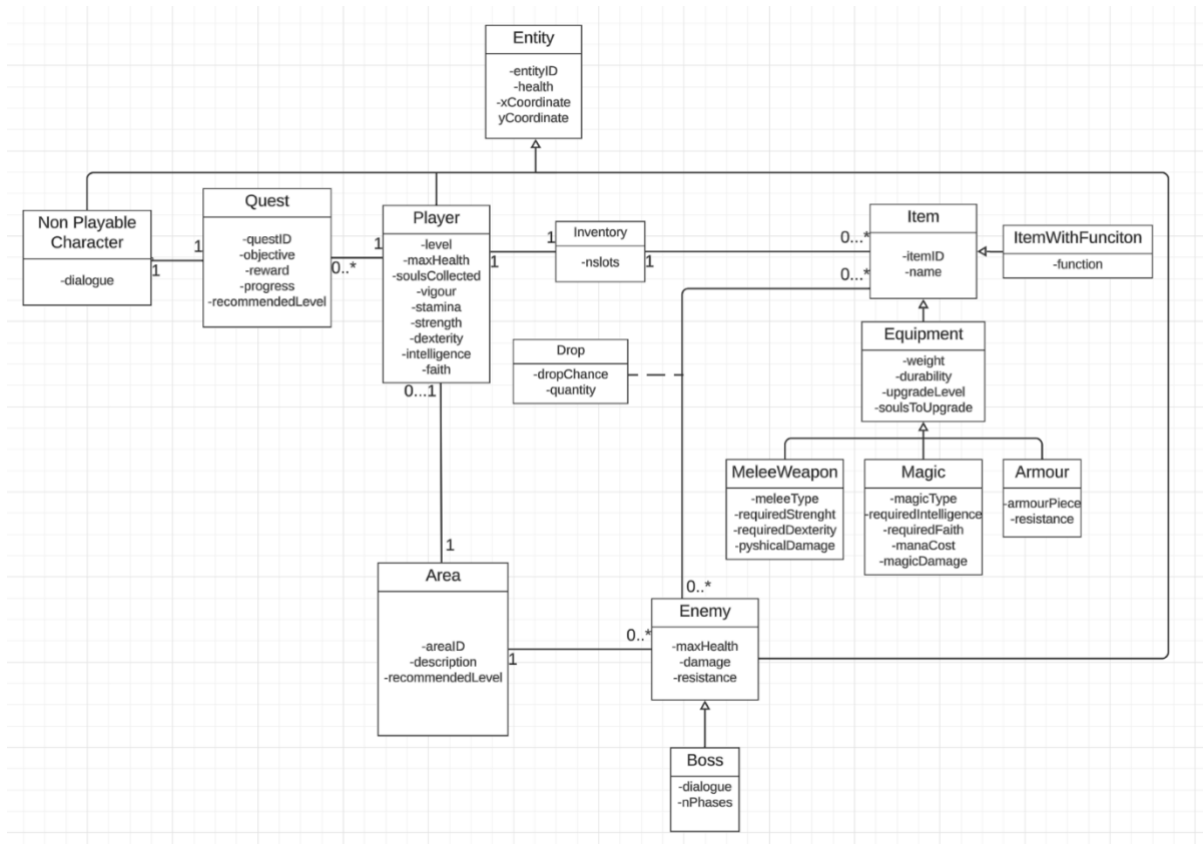
Next, the Non-Playable Character (or NPC). This Entity is responsible for giving Quests and giving tips or crucial information for the game progression to the player. Each NPC has a dialogue when you interact with them and should give a Quest.

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.)

4.1.2. Initial Conceptual Modelling



4.1.3. Generative AI Integration AI Used – ChatGPT 4.0

In our interaction with ChatGPT, firstly, we wanted to see and correct our UML Model by giving AI the Domain we defined and asking if it could reproduce the same as we did. Then, seeing that the description generated by ChatGPT was good enough, we sent AI the UML Model we made and asked if it was fitting AI's description. Again, since AI made no major changes or comments, we then asked:

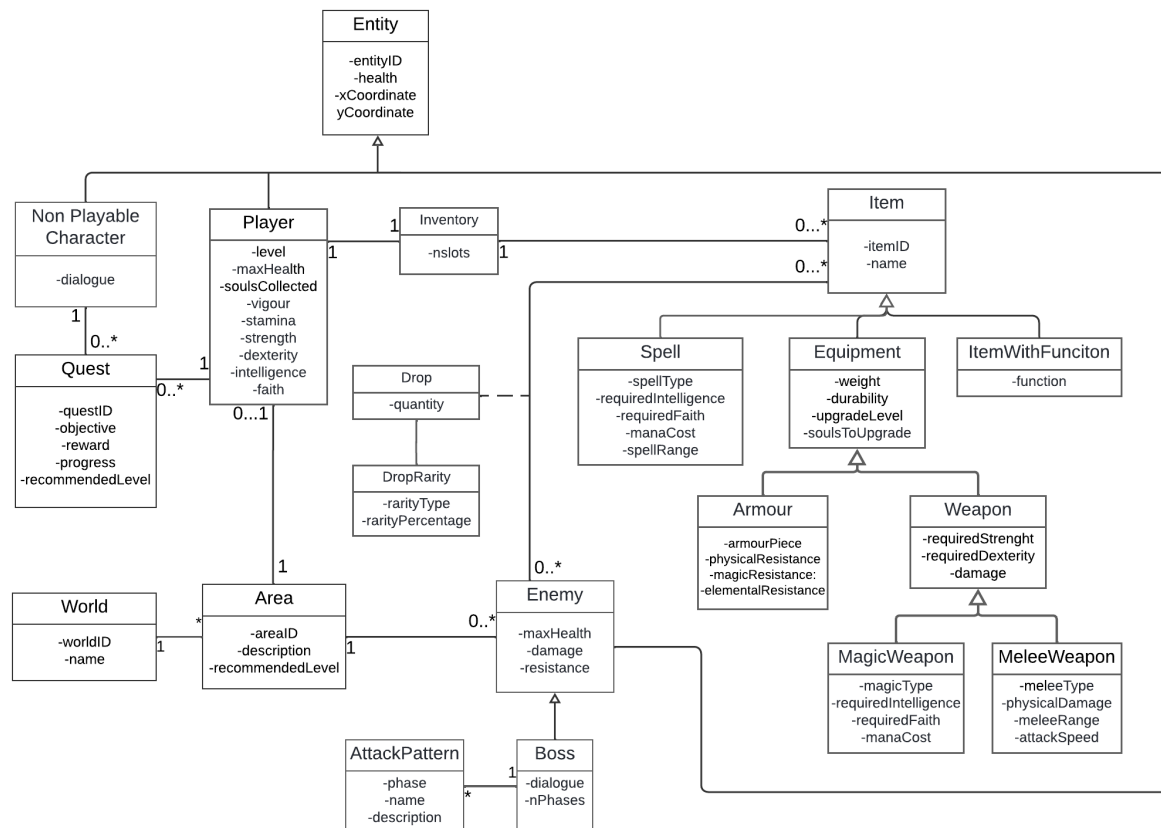
How can you improve the model?

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

We then made the UML Model that the chat entirely did, without any critical analysis (we did it in Task 4.1.4), **except** for point 7, since point 7 integrate Interfaces and our UML didn't implement any.

In the end, this was what we were left with:



4.1.4. Final Conceptual Modelling

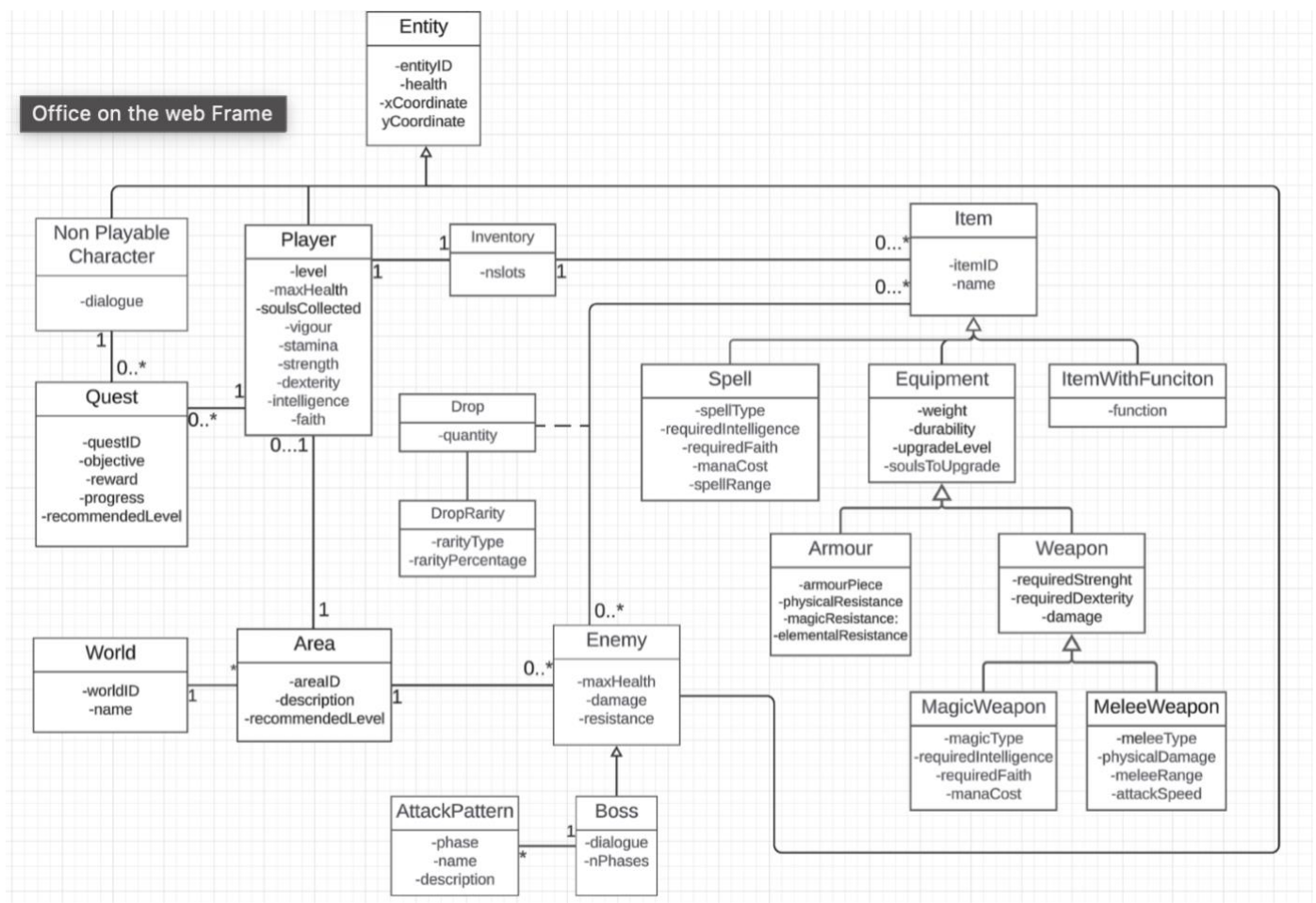
Firstly, we ignored points 7 and 8, since we didn't implement interfaces at all, and we are making the Database such as there is only one map to be recorded, so a single class for a single object wouldn't be necessary.

Then, we went along fully with points 1, 2, 3, 4 and 6, and partially with point 5. Here's the breakdown:

- 1. Adding Abstract Classes for specific Item types.** We thought that this would make the hierarchy easier to represent, since all types of weapons in the game have a specific behaviour, type of damage and can have additional attributes needed to be used (e.g. Magic weapons require not only Strength and Dexterity but also Intelligence and Faith). Prior to that, we added an Abstract Class Weapon, and Classes MagicWeapon and MeleeWeapon.
- 2. Refining Equipment Inheritance.** Spells and MagicWeapons aren't really the same, since a Spell is a sorcery that you cast, and a MagicWeapon is more like a spellcasting tool or a weapon that has magical attributes. Because of this, we created a new Class Spell that inherits from Equipment to better represent the types of equipment you can use.

3. **Introducing AttackPattern for Boss Phases.** For a database, it isn't crucial to know exactly what each Phase does specifically, but it is indeed important to record the name of them, and a brief description of the Pattern. For this, we added Class AttackPattern.
4. **Normalize NPC-Quest Relationship.** In the game, an NPC can have multiple quests or give a specific kind of quest if you are in certain part of the game. So, we changed the multiplicity of the NPC-Quest Relationship to 1 to Many.
5. **Refine Drop Mechanism (Partial).** AI advised to add a new Class DropRarity to handle drop rates and quantities, but we didn't see the need to do it, since the drop system in SoulsLike games is much more complex than a simple rarity drop and wouldn't correctly represent it. To simplify things, we just added an attribute dropRarity in the Drop Class.
6. **Introduce the Resistance System for Armour.** There are different types of resistance, hence there are many different types of armour for different situations and/or enemies (e.g. you would use an armour with high fire resistance against a fire boss). For this, we divided the resistance attribute in 3 different attributes: physicalResistance, magicResistance and elementalResistance.

After those modifications, this was our UML:



5.1.1. Conceptual Model Refining

- Added constraints for almost every Class
- Aligned the attributes to the left
- Restructured the diagram to make it easier to read and interpret
- Changed the class “Non Playable Character” to “NonPlayableCharacter”
- Changed the relation from Enemy and Player to Area, to Entity to Area

[illegible]

5.1.2. Initial Relational Schema

Entity (entityID, health, xCoordinate, yCoordinate, areaID->Area)

Player (entityID->Entity, level, maxHealth, soulsCollected, vigour, stamina, strength, dexterity, intelligence, faith)

NonPlayableCharacter (entityID->Entity, dialogue)

Enemy (entityID->Entity, maxHealth, damage, resistance)

Boss (enemyID->Enemy, dialogue, nPhases)

AttackPattern (bossID->Boss, phase, name, description)

Quest (questID, objective, reward, progress, recommendedLevel, playerID->Player, npcID->NonPlayableCharacter)

Area (areaID, description, recommendedLevel)

Inventory (playerID->Player, nSlots)

Item (itemID, name, inventoryID -> Inventory)

Drop (itemID -> Item, enemyID -> Enemy, dropRarity, dropChance, quantity)

ItemWithFunction (itemID->Item, function)

Spell (itemID -> Item, spellType, requiredIntelligence, requiredFaith, manaCost, spellRange)

Equipment (itemID->Item, weight, durability, upgradeLevel, soulsToUpgrade)

Armour (equipmentID -> Equipment, armourPiece, physicalResistance, magicResistance, elementalResistance)

Weapon (equipmentID->Equipment, requiredStrength, requiredDexterity, damage)

MagicWeapon (weaponID->Weapon, magicType, requiredIntelligence, requiredFaith, manaCost)

MeleeWeapon (weaponID->Weapon, meleeType, physicalDamage, meleeRange, attackSpeed)

5.1.3. Generative AI Integration

In our interaction with ChatGPT, we did basically the same thing we did in the point 4.1.3., we sent AI the UML model and asked to make the Relational Schema. ChatGPT did the Schema, but it had a different thing than our Schema: When it had a hierarchy like this: Item -> Equipment -> Weapon, for Weapon, when referencing to the parent Class, it should go like Weapon (equipmentID -> Equipment), and it generated like this:

Weapon (itemID -> Equipment). While this is not entirely incorrect, we thought that this is not the most efficient way to represent hierarchy between classes.

After notifying ChatGPT about this, it sent another Schema with the following notation: Weapon (weaponID -> Equipment). Again, not completely wrong, but this is not how we visualized the Schema.

So, we made one last prompt on the ChatGPT: We sent the UML model, the initial Relational Schema and asked if it could correct and improve in any way the Schema. Here are the suggestions it made:

1. Add Composite Primary Keys Where Necessary:

- Tables like `AttackPattern` currently list `phase` as part of the primary key but do not explicitly define a composite key (`bossID`, `phase`). Specifying this ensures that each attack pattern for a boss is unique.

Updated `AttackPattern`:

- (`bossID`, `phase`) should be the composite primary key.

2. Normalize Equipment Subtypes:

- Currently, `MagicWeapon` and `MeleeWeapon` inherit from `Weapon`, and both have specific attributes (`magicType`, `meleeType`, etc.). This structure is fine, but if there are additional subtypes in the future, it might be beneficial to create a more generic `WeaponType` table with a type discriminator.

Proposed Option (Only if extensibility is desired):

- Add `weaponType` to `Weapon` as an enum (e.g., `Magic`, `Melee`, etc.), reducing the need for separate subtype tables unless they have very specific behaviors.

3. Handle Complex Relationships Between `Quest`, `Player`, and `NPC`:

- Each quest currently belongs to a single player and is tied to a single NPC. If the game design intends for quests to be shared across multiple players or NPCs, consider:
 - Adding a `QuestParticipation` table (`playerID`, `questID`, `progress`, etc.).
 - This could allow for cooperative quests or interactions with multiple NPCs.

Optional Improvement (If quests are shared):

- Add a `QuestParticipation` table to handle multi-player or multi-NPC associations.

4. Refine `Drop` Constraints:

- The `Drop` table defines items that enemies can drop. To avoid having enemies drop invalid or unassociated items:
 - Add a `validForEnemy` flag or a range of possible item levels tied to the enemy's level.

5. Strengthen `Inventory` Relationship:

- An inventory might need a total `capacity` attribute (based on `nSlots` and item weights) to limit overloading.

Updated `Inventory`:

- Add a calculated `currentWeight` or `maxWeight` attribute.

6. Generalize `ItemWithFunction`:

- The `function` column in `ItemWithFunction` is likely intended for dynamic behavior (e.g., healing, teleporting, etc.). To future-proof this:
 - Consider linking this to a `Functionality` table with predefined effects and their parameters.

Proposed Option:

- Add a `Functionality` table (`functionID`, `description`, `effectType`, `effectValue`), and replace the `function` column in `ItemWithFunction` with a `functionID`.

7. Data Validation and Constraints:

- Add constraints to ensure data integrity:
 - Enforce `dropChance <= 1` in `Drop`.
 - Ensure `progress <= 1` in `Quest`.
 - Validate that `requiredStrength`, `requiredDexterity`, etc., are non-negative in `Weapon`.

Final Thought

These adjustments are optional and depend on the game's intended functionality and scalability. If the current schema aligns with your requirements, it's already well-designed and functional. Let me know if you'd like me to implement any of these suggestions!

5.1.4. Final Relational Schema

After analysing the suggestions AI made, we concluded that none of the suggestions would fit the Schema we made. Since almost every suggestion made by the AI requires changing the UML or adding new logic to the Schema, which was not the objective with the AI Integration in our project. Here's why we didn't apply them:

Point 1: A composite primary key for AttackPattern was already applied. Another composite keys aren't used or necessary because only an ID for each class is sufficient in this context

Point 2: Weapon types have specific behaviours and forms of usage. Adding an WeaponType class would only make the Schema more complicated.

Point 3: Each Quest belongs to 1 NPC and is given to 1 Player. Multiplayer is not designed for this project.

Point 4 and 7: Constrains are later implemented in SQL.

Point 5: In the original game, max weight is only for the Equipment that is currently equipped. We didn't implement any kind of way to know what weapons or armour is currently equipped.

Point 6: To simplify the Database, ItemWithFunction is simply representational. It just represents its functionality.

Per se, we didn't use any of the AI suggestions and went along with the Relational Schema proposed in point 5.1.2.

5.1.5 Initial Analysis of Functional Dependencies and Normal Forms

Analysing our Relational Schema:

Functional Dependencies:

Entity: entityID -> health, xCoordinate, yCoordinate, areaID

Player: entityID -> level, maxHealth, soulsCollected, vigour, stamina, strength, dexterity, intelligence, faith

NonPlayableCharacter: entityID -> dialogue

Enemy: entityID -> maxHealth, damage, resistance

Boss: enemyID -> dialogue, nPhases

AttackPattern: bossID, phase -> name, description

Quest: questID -> objective, reward, progress, recommendedLevel, playerID, npcID

Area: areaID -> description, recommendedLevel

Inventory: playerID -> nSlots

Item: itemID → name, inventory

Drop: itemID, enemyID -> dropRarity, dropChance, quantity

ItemWithFunction: itemID -> function

Spell: itemID -> spellType, requiredIntelligence, requiredFaith, manaCost, spellRange

Equipment: itemID -> weight, durability, upgradeLevel, soulsToUpgrade

Armour: equipmentID -> armourPiece, physicalResistance, magicResistance, elementalResistance

Weapon: equipmentID -> requiredStrength, requiredDexterity, damage

MagicWeapon: weaponID -> magicType, requiredIntelligence, requiredFaith, manaCost

MeleeWeapon: weaponID -> meleeType, physicalDamage, meleeRange, attackSpeed

After analysing, we concluded that:

- All tables have a primary key.
- All attributes that are not a primary key fully depend on the PK, **so there are no partial dependencies.**
- No non-PK attribute depends on another non-PK attribute, **so there are no transitive dependencies.**

This Schema is in BCNF and 3NF and no decomposition is needed.

5.1.6 Generative AI Integration

Interaction with ChatGPT was short, we just asked AI to check if the Relational Schema was following BCNF and 3NF principles.

To analyze the schema for functional dependencies (FDs) and normalization, we will perform the following steps:

1. Identify functional dependencies (FDs) in each relation.
2. Check for 3rd Normal Form (3NF) and Boyce-Codd Normal Form (BCNF) violations.
3. Justify compliance or decompose relations if violations exist.

Step 1: Functional Dependencies (FDs)

Entity

- FDs:
 - `entityID → health, xCoordinate, yCoordinate, areaID`

This is in BCNF and 3NF as `entityID` is the primary key, and all attributes are fully dependent on it.

Player

- FDs:
 - `entityID → level, maxHealth, soulsCollected, vigour, stamina, strength, dexterity, intelligence, faith`

This is in BCNF and 3NF because `entityID` is the primary key, and all attributes are fully dependent on it.

Step 2: Check for BCNF and 3NF Violations

BCNF and 3NF Analysis

1. All tables comply with BCNF and 3NF because:
 - Each table has a primary key.
 - All non-key attributes are fully functionally dependent on the whole primary key (no partial dependency).
 - There are no transitive dependencies (no non-key attribute depends on another non-key attribute).

Step 3: Justification for No Violations

- **No partial dependencies:** In every table, all attributes depend fully on the primary key.
- **No transitive dependencies:** No attribute depends on another attribute that is not a primary key.
- **BCNF and 3NF compliance:** Each relation is already in BCNF because all attributes are fully dependent on candidate keys.

Conclusion

The schema is **already in BCNF and 3NF**, and no decomposition is required. The design adheres to normalization principles, ensuring no redundancy or update anomalies.

5.1.7 Final Analysis of Functional Dependencies and Normal Forms


Since ChatGPT made no alterations on the Schema and our input was working fine, no modifications were made on the Relational Schema from point 5.1.2.

5.1.8. Initial SQLite Database Creation and Data Loading

(Tasks 5.1.8 and 5.1.11)

5.1.9. Generative AI Integration (Task 5.1.9 and 5.1.12)

ChatGPT created a new Database creation and Populate files, not utilizing some of the suggestions mentioned above, but after testing the Database, it had errors derived from some of the new features. After reporting to the AI, it sent a new file with almost every new feature removed and a few structural improvements. Though, we analysed the code and saw that we were checking if the table exists twice for database reutilization purposes, and we could automatically increment the ID in parent classes. After defining create2.sql, we asked AI to implement populate2.sql, based on the data from populate1.sql and create2.sql structure.

 Since the schema cannot change, any suggestions for improvement must work within the existing structure. Here are recommendations that focus on implementation details, optimization, and best practices while adhering to the constraints:

1. Query Optimization

- Indexes:
 - Add indexes on frequently queried fields, especially foreign keys like `areaID`, `playerID`, and `enemyID`.
 - Consider indexing `dropChance` and `dropRarity` in the `Drop` table to speed up loot generation.

3. Data Usability

- Default Values:
 - Define default values for certain fields (e.g., `upgradeLevel = 0` in `Equipment`, `progress = 0` in `Quest`) to reduce the need for explicit initialization during insertion.
- Nullable Columns:
 - Reassess whether some columns should allow `NULL` values:
 - For example, does every `Enemy` need an associated `Drop`? If not, nullable foreign keys could simplify queries.

5. Enforcement of Schema Rules

- Relationships:
 - Ensure constraints like cascading deletes are properly implemented. For instance:
 - Deleting a `Player` should cascade to delete their `Inventory` and associated `Item` rows.
 - Deleting an `Enemy` should remove associated `Drop` records.

6. Scalability and Maintainability

- Entity Generalization:
 - While the schema cannot change, maintaining consistency in derived tables (`Player`, `Enemy`, `Boss`, `NonPlayableCharacter`) by programmatically tying them to their `Entity` parent attributes will reduce errors.
- Inventory Handling:
 - Validate `inventorySlots` dynamically during gameplay to avoid situations where a player's inventory exceeds its capacity.

2. Data Validation

- Consistency Checks:
 - Ensure data integrity with constraints such as:
 - `CHECK` constraints for attribute ranges (e.g., `manaCost > 0`, `weight >= 0`).
 - Enforce relationships where needed (e.g., `recommendedLevel` in `Quest` and `Area` should be within realistic ranges).
- Unique Constraints:
 - Validate uniqueness where implied but not explicitly enforced (e.g., `name` in `Item`).

4. Gameplay Enhancements

- Dialogue Flexibility:
 - Store the `dialogue` field in a structured JSON format (if the SQL dialect supports it) for easier dynamic querying and formatting in-game.
- Drop Table Expansion:
 - Use `dropChance` and `dropRarity` more effectively by standardizing rarity tiers. For instance, create a rarity system (e.g., "common", "rare", "legendary") and map it to drop rates programmatically.

7. Code Quality

- Stored Procedures and Triggers:
 - Implement stored procedures for common operations:
 - Leveling up a player (increment `level` and distribute attribute points).
 - Upgrading an item (increment `upgradeLevel`, deduct `soulsToUpgrade`).
 - Use triggers to enforce rules (e.g., decrement durability when using `Equipment`).

8. Documentation and Metadata

- Commenting:
 - Document relationships and intended use cases for each table and column to assist developers.
- Game Design Annotations:
 - Use metadata tables to describe game logic (e.g., `attribute scaling`, `area difficulty`) for transparency and debugging.

↓

Conclusion:

The schema is robust and well-designed for its purpose, with clear relationships and logical groupings. The suggested improvements aim to optimize data integrity, gameplay dynamics, and usability while respecting the constraints that the schema cannot change. Would you like help implementing any of these improvements?

5.1.10. Final SQLITE Database Creation and Data Loading (Task 5.1.10 and 5.1.13)

Here is our correction of the code on create2.sql:

- We removed IF NOT EXISTS from the table creation, since DROP TABLE IF EXISTS ensures that the database can be re-written safely and adding that condition in the table creation is redundant and unnecessary.
- Added AUTOINCREMENT on parent classes like Entity, Item and Quest to simplify populate2.sql.
- Improved some grammar mistakes, like Monster_Drop_FK to Drop_Item_FK.

```
DROP TABLE IF EXISTS MeleeWeapon;
DROP TABLE IF EXISTS MagicWeapon;
DROP TABLE IF EXISTS Weapon;
DROP TABLE IF EXISTS Armour;
DROP TABLE IF EXISTS Equipment;
DROP TABLE IF EXISTS Spell;
DROP TABLE IF EXISTS ItemWithFunction;
DROP TABLE IF EXISTS DropItem;
DROP TABLE IF EXISTS Item;
DROP TABLE IF EXISTS Inventory;
DROP TABLE IF EXISTS Quest;
DROP TABLE IF EXISTS AttackPattern;
DROP TABLE IF EXISTS Boss;
DROP TABLE IF EXISTS Enemy;
DROP TABLE IF EXISTS NonPlayableCharacter;
DROP TABLE IF EXISTS Player;
DROP TABLE IF EXISTS Entity;
DROP TABLE IF EXISTS Area;

CREATE TABLE Area (
    areaID INTEGER PRIMARY KEY AUTOINCREMENT,
    description TEXT,
    recommendedLevel INTEGER NOT NULL,
    CONSTRAINT positiveRecommendedLevel CHECK (recommendedLevel >= 1)
);
```

Here is our correction of the code on populate.sql:

- We altered Entity, Item and Quest tables and removed the ID from them, since create automatically implements them.

```
INSERT INTO Area (description, recommendedLevel) VALUES
('Trading Haul', 1),
('Cave', 5),
('Arena', 15),
('Hidden Gem Cave', 30),
('Light Temple', 20),
('Alley', 10),
('Shadow Realm', 50),
('Jungle', 20),
('Main Lobby', 1),
('Spider Nest', 70),
('Tilted Towers', 69),
('Los Santos', 80),
('Route A', 30);
```

Task 6 – Final considerations

6.1. – Generative AI Integration

We used AI tool ChatGPT (Version GPT-4.0) mainly to correct our work, give new suggestions or improve in any way our proposals.

In earlier stages of the project, ChatGPT proved to be useful in generating new ideas, adding new features and correcting errors or redundant data.

For example, in point 4.1.3, AI proved to be very useful on giving new ideas to improve the UML and add new relations. Those suggestions helped us very much on designing the model and later the Database.

However, when we had our model defined and asked for any improvements on the Relational Model, AI proved not to be very helpful, since it continuously had errors, did the opposite we asked it to do, or overall didn't improve the work.

For example, when we used ChatGPT in point 5.1.3., since converting an UML to a Relational Schema is just following a set of rules, ChatGPT would only be useful if we had any error or redundant information on the Schema, which we didn't have, and when asked for improvements, it changed the UML and added new tables, even after being asked to not do it.

Overall, we think that using generative AI for giving new ideas or approaches and correcting errors is very useful and can be a great way to improve your work, but it can be unnecessary using it for tasks that require a closed set of rules or methods.

6.2. Participations

Diogo Sousa Campeão was responsible for the UML creation, the Domain definition, the final SQL Database and Populate (after AI integration) and the whole documentation of the project. Tomás Costa Barros helped on the Domain definition and was responsible for the Relational model creation and Analysis of FD's and NF's. Hugo Miguel Gomes Silva made the whole implementation in SQL and helped in the UML. The project was well divided, and everybody did their work.