Proposed Entity-Relation Diagram for MayAztec project

Profesor: Esteban Castillo Juárez

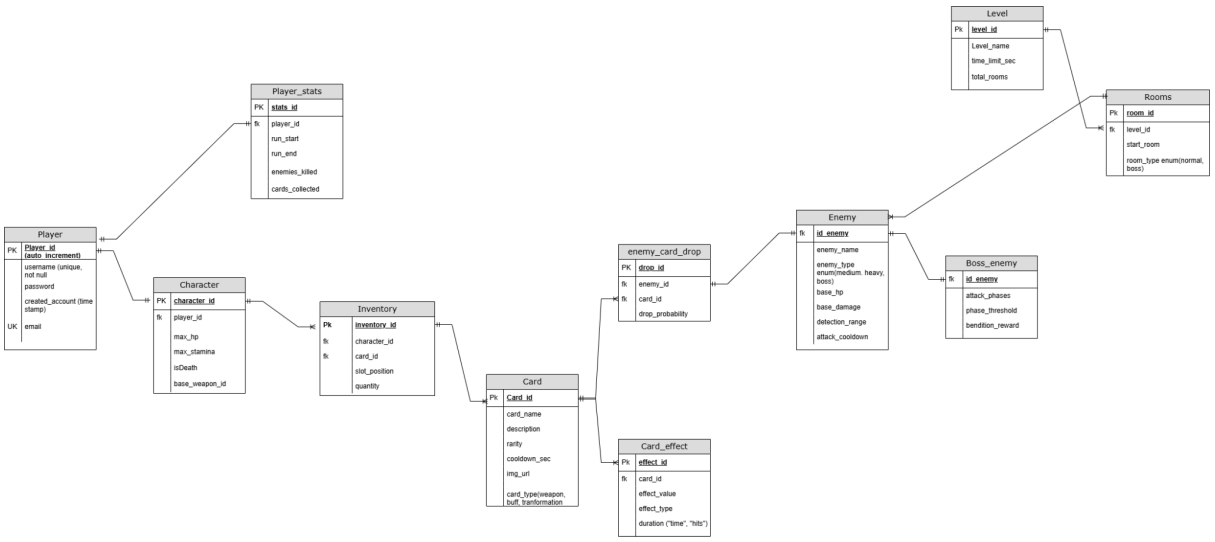
Materia: Construcción de Software para la toma de decisiones

Nombres:

- Mauricio Emilio Monroy González - A01029647
- Héctor Lugo Gabino - A01029811
- Nicolás Quintana - A01785655

Fecha: 04 de abrol del 2025

Justification



Notes changes:

Players - this table was deleted, because it made no sense to store players in another table, since they are already stored in the player table.

Chest - The chests table, we decided that it is not necessary to have it stored in the database, because we can manage it in a simpler way from the front-end. And having it in the database may be something not so necessary, because they are only likely to appear and have certain types of cards.

Added Run tables and card usages.

Remove enemy_card_drop.

- Justification: Drop probabilities are game rules, not critical data requiring persistence (e.g., do not affect player history).
- Simplifies the database schema and reduces the number of tables.
- All this will be handled from the front end of the game, since there are probabilities that will remain fixed after local testing, we consider that it is better to handle it directly in the game attributes.

Simplify Enemy table.

Due to the decision to handle probabilities in the front end of the game, to simplify the testing and development process. These enemy related attributes should be removed.

Remove cardEffects.

Because we manage effects, usage times, and probabilities. We consider pertinent to eliminate the cardEffects table, since these will be implicit in the creation of the objects of each card in the front end of the game, simplifying its implementation without having to make a request to the database to instantiate them in the game.

Benefits of the Changes

- Performance: By reducing the number of database queries during the game, we improve the user experience.
- Simplicity: A simpler schema facilitates maintenance and reduces the probability of errors.
- Flexibility: Separation of game rules (frontend) and persistent data (backend) allows faster adjustments to the roll without changes to the database structure.
- Data Analysis: New RUN and CardUsages tables provide a solid foundation for statistical analysis, enabling improvements based on real game data.

In our proposed ER model for our game MayAztec, we proposed 11 Tables with no intermediate tables.

Main entities and their attributes:

Player

- This table manages user account information.
- Attributes:
    - player_id (int auto_increment) PK
    - username (varchar)
    - password (varchar)
    - date_create (time_stamp)
    - email (varchar) UK
- Relations:
    - One-to-one with ChatacterStatus:

        - The separation between Player and CharacterStatus allows the account management system to be decoupled from the game state. This makes it easier to restore character statues.
    - One-to-one with RUN:
        - This relationship allows multiple games to be registered for the same player, implementing the “runs” system, characteristic of roguelites.
        - **Constraints**: the foreign key player_id in Player_Runstats ensures that each statistic belongs to a specific player.

RUN

- This table will serve as a support, to save the complete information of the level, defeated enemies, combat time, if I finished the game or not, how long it took to complete it.
- Attributes:
    - run_id (PK) auto_increment
    - level_id(fk)
    - player_id(fk)
    - run_star (time_stamp)
    - run_finised (time_stamp)
    - timefighting (int)
    - isGameFinished (bool)
- Relations:
    - One to many with player
        - It will allow the player to generate statistics through the run game. This will allow us to create specific views of the runs.
    - One to many with level
        - A room can have many levels. In our case 2 levels. This allows us to track the entire game of player u and not just one level.

Character

- Purpose**:** Store in-game character attributes (Hp, stamina, base weapon) associated with a player.
- isDeath will be used to know whether or not I passed a level or a room. Thanks to combining the time stamp with this attribute we will be able to make summary screens, because it is a bool that indicates the state of the character.

- Attributes:
    - character_id (PK) auto_increment
    - player_id (FK) fk (Player)
    - max_hp (tiny int)
    - current_hp (tiny int)
    - max_stamina (tiny int)
    - current_stamina (tiny int)
    - base_weapon_id (int) foreigkey to card
    - isDeath
- Relations:
    - One-to-one with Player
        - Even if the player has many runs. The base character remains the same and a player will always have the same character.
    - One-to-one with Inventory
        - This is key as this limits the character to just one inventory, not allowing more than the 6 cards in their inventory

Inventory:

- Purpose: Store the card that the character has in his “deck” or available slots.
- The limited quantity allows the gameplay to be strategic for the player, without abusing the cards. And have a centralized space for this field

- Attributes:
    - run_id (int) PK
    - character_id (int) FK to CharacterStatus
    - card_id (int) FK to Card
    - current_size (tinyint)
- Relations:
    - One-to-one with CharacterStatus
    - One Mandatory-to-Many Mandatory with Card:
        - An inventory can contain different cards and this inventory is unique for each run.

Card

- Purpose: To model each lottery card. The games revolve around the cards, these are the power ups of the adventure.
- Attributes:
    - Card_id (int) PK
    - card_name (varchar)
    - description (varchar)
    - rarity (enum(“low_tier, high_tier))
    - cooldown_sec (smallint)
    - img_url
    - card_type (enum(“weapon, transformation, buff))
    - effect_id (int) FK to Card_effect
- Relations:
    - Many Mandatory-to-One Mandatory with Inventory
        - Allows many cards (5 plus base card weapon) to be attributed to only one inventory

Annotations:

- img_url: This field stores the path to the image that represents the chart graphically in the game.

CardUsages

- **Purpose:** This card gives us the possibility to generate more accurate statistics, which otherwise could not be generated. Since the inventory only serves as a connection between the player and the cards, but does not count specifically which card was used. The purpose of this table is that when we want to know which are the most used cards by the players, it would give us an idea of how they use them, also if a query is made with the drop rate we can generate inferences. This table is a record to generate statistics.
- Attributes:
    - cardUsages_id
    - character_id
    - inventory_id
    - run_id
    - card_id
- Relations:

    - Many to one with Character:
        - A player can have many cards of different types used, that is why cardinality is assigned to a single player and a single run.
    - Many to one with inventory:
        - Many uses come from a single inventory, since an inventory stores the different cards and as you use them they are deleted, and recorded in the uses.

Enemy

- To store the data of each enemy. Centralizes the common information of all enemies.
- The important attributes of the enemy as its attackcooldown is to not attack you repeatedly, detection range is to switch between pursuit and dispersion modes a very simple implementation of a finite state machine.
- Its attributes like base_hp, damage, are fundamental to instantiate the enemies with these base values for each one.

- Attributes:
    - id_enemy (int) PK
    - enemy_name (varchar)
    - enemy_type (enum(“medium”, “heavy”, “boss”))
    - base_hp (tinyint)
    - base_damage (tinyint)
    - detection_range (smallint)
    - attack_cooldown (float)
- Relations:
    - One-to-one with enemy_card_drop:
        - This relationship implements the probabilistic drop system. Each enemy can drop several cards with different probabilities.
        - **Constraints:** Foreign key ensure referential integrity.
    - One-to-one with Boss_enemy:
        - This is a specialization relationship where bosses are a special type of enemy with additional attributes. It allows reusing common attributes while adding specific characteristics.

        - **Constraints**: the **primary key** of Boss_enemy is also a **foreign key** that references Enemy, ensuring that only some enemies are bosses.
    - Many Mandatory-to-One Mandatory with Rooms

Annotations:

- detection_range: Defines the distance at which the enemy can detect the player, crucial for AI systems and enemy behavior.
- attack_cooldown: Represents the time the enemy must wait between attacks, balancing the difficulty of the game.

Level

- Purpose: define game levels. Is a way to persist the configuration of each level required.
    - Timestamps are useful to know how much time was spent in each level.
    - The count of how many cards you drop can help us to know the difference of cards used vs. cards left unused or uncollected.
- Attributes
    - level_id (int) PK
    - name (varchar)
    - time_limit_sec (float)
    - total_rooms (tinyint)
    - startLevel (timeStamp)
    - finishLevel (timestamp)
    - cardsdrops (tinyInt)
- Relations:
    - One-to-one with Player_Runstats:
        - In a level a set of statistics is generated to the player, and that is unique for that run.
    - One Mandatory-to-Many Mandatory with Rooms:
        - One level contains many rooms

Annotations:

- time_limit_sec: This field sets a time limit to complete the level, adding pressure to the player and an additional element of challenge.

- total_rooms: Defines the number of rooms in a level, allowing to generate levels of different complexity.
- room_type: Classifies the rooms (combat, boss), allowing to create varied and progressive experiences.

Rooms

- Purpose: store rooms whitin the level, each room its own room-type. A level can have multiple rooms. This facilitates the generation of dungeons and the assignment of enemies to each room.
- The rooms, for example, are fundamental to know which cards to use in the boss room. It helps us to know specific things about the rooms of the level.

- Attributes
  - room_id (int) PK
  - level_id (int) FK to Level
  - room_type (enum(“normal”, “boss”))
- Relations:
  - Many Mandatory-to-One Mandatory with Level
  - One Mandatory-to-Many Mandatory with Enemy
  - One-to-one with Chest

Boss Enemy

- Purpose: Separates to empathize with specialization from a normal enemy to one with boss attributes.
- Attributes
  - id_enemy (int) FK
  - attack_phases (tinyint)
  - phase_threshold (int)
  - bendition_reward (int) FK to Card_effec
- Relations
  - One-to-one with Enemy
    - It is a derivative of a normal enemy, containing unique attributes, phases of attacks.

1. First Normal Form (1NF): Atomic Attributes

Compliance:

- All attributes store indivisible values. For example:
    - card_type in Card uses an ENUM(“weapon”, “transformation”, “buff”), which defines unique and specific values.
    - enemy_type in Enemy classifies enemies as “medium”, “heavy”, or “boss”, without mixing multiple values in one field.

2. Second Normal Form (2NF): No Partial Dependencies

Compliance:

- All tables have simple primary keys (e.g. player_id, run_id), and non-key attributes are fully dependent on them.
- Example in Card_effect:
    - effect_id is the PK, and attributes like effect_value or duration depend only on it, not on card_id (which is a FK to link effects to cards).