Technical Documentation

Burmese Group

Authors

Allison Lenhoff
Brian Lawser
Michael Dickenson
Nikki Cayas
Yonas Tadesse

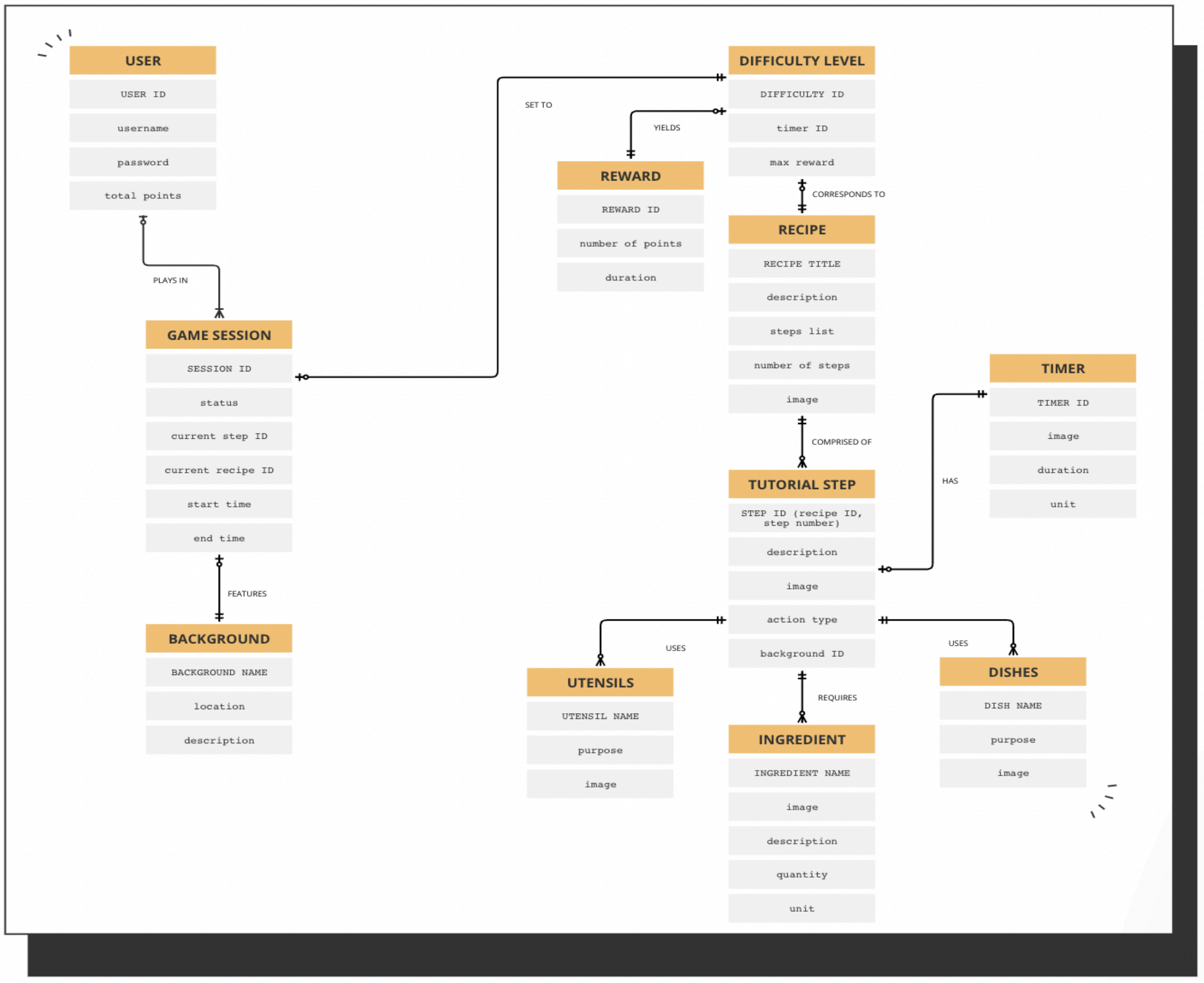
Date Prepared

5/1/2024

Table of Contents

Diagrams

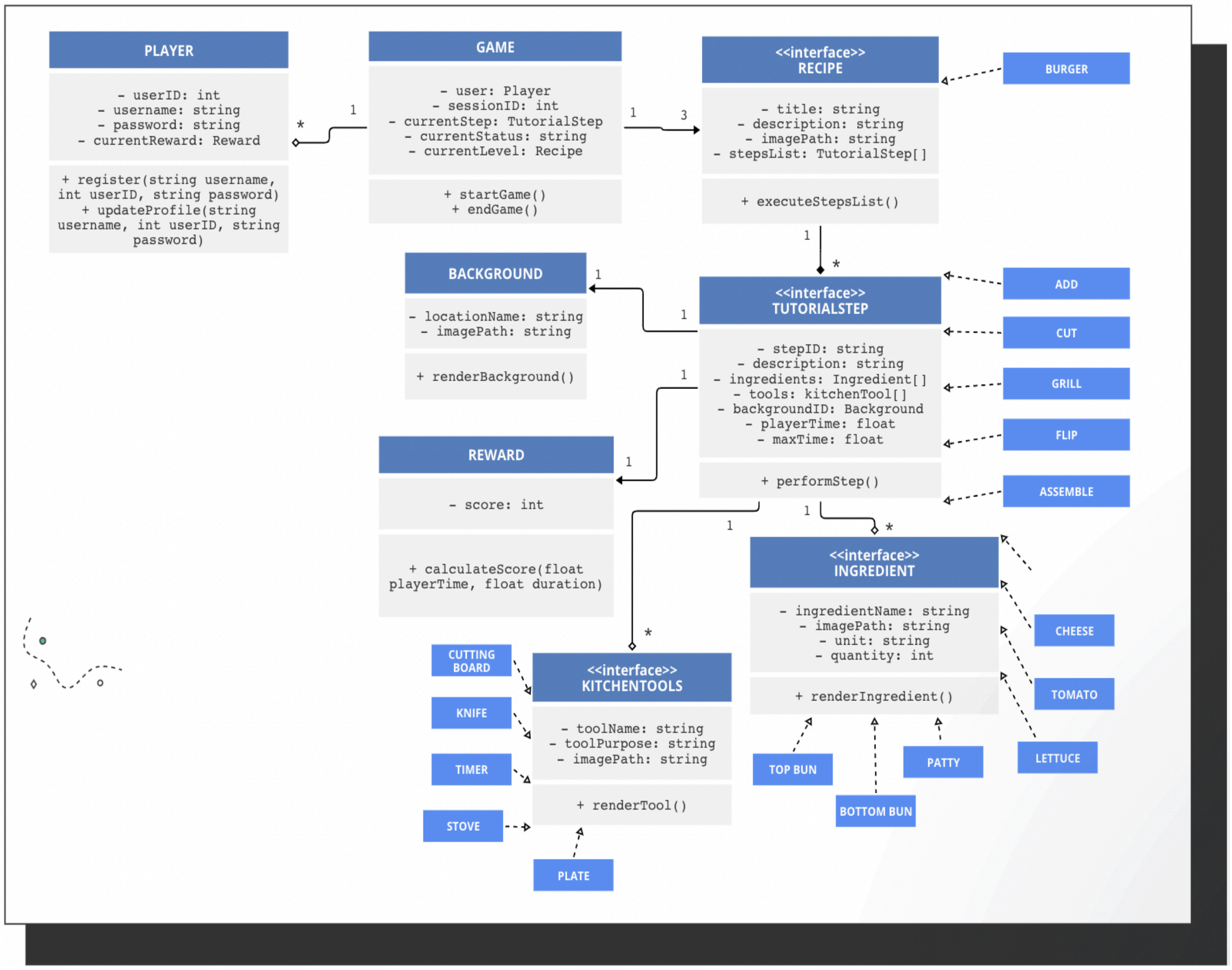
ER Diagram



In our project, we rely on the Entity-Relationship (ER) diagram as a foundational tool. We use this diagram to understand the intricate relationships between different entities within our system. For example, we recognize that a User can engage in multiple GameSessions, each of which is associated with specific DifficultyLevels and Backgrounds. This comprehension guides us as we define the database schema, including tables and their attributes. By identifying primary keys and foreign keys, such as user_id in the User table, we ensure data integrity and establish clear connections between entities. This understanding of data relationships allows us to tailor game mechanics to the chosen difficulty level. Additionally, the ER diagram serves as a visual representation of our project's data model, facilitating communication amongst our frontend and backend developers.

The ER Diagram provides a shared understanding of our data requirements and guides us toward a successful implementation of our Cooking Mama-inspired project.

Class Diagram



In our project, we used the class diagram as a blueprint for organizing the various components of our system. By examining the relationships and interactions between classes, we gain a strong understanding of how different elements of the game are structured and interact with each other. For example, we realized that classes such as Player, Recipe, and Step are central to our game mechanics. The diagram illustrates how these classes interact with each other, with Player initiating Game instances, which in turn involve specific Recipe instances. This understanding guides us as we implement the functionalities of each class, ensuring that they align with the overall game design. Additionally, the class diagram facilitates communication and understanding between the frontend and backend team. It provides a visual representation of our system's structure, helping us stay organized and intentional with what we programmed during the development process and during each separate sprint.

Ultimately, the class diagram plays a crucial role in guiding the implementation of our Cooking Mama-inspired project.

Frontend Development

Our frontend development process for the Godot project encompassed distinct roles and responsibilities. We made sure to leverage Godot's features. We created pixel art sprites for the game's visuals, each crafted using Pixel Studio. Using Godot's node system, we integrated these sprites into the game environment, ensuring smooth rendering and interaction.

We implemented the cooking mechanics, using Godot's node-based architecture to create intuitive controls and interfaces. By using nodes like Area2D for collision detection and KinematicBody2D for player interaction, we created a drag-and-drop scheme for putting burgers on the grill. Additionally, we used Godot's signal system to connect input events to game actions, allowing responsiveness and fluid transitions.

We also used Godot's scripting capabilities to devise the logic for vegetable cutting and prep. By implementing custom scripts and Godot's animation features, we created dynamic knife movements and texture changes to simulate the cutting process.

Godot's node-based scene system also allowed us to implement the assembling functionality. Using nodes like RigidBody2D and CollisionShape2D, we designed mechanisms for stacking ingredients and detecting collisions. Using Godot's physics behaviors for replicated realistic interactions between falling ingredients and the burger assembly area. By connecting input signals to player actions (left and right arrow keys) using Godot's signal system, we simulated movement. Lastly, we utilized Godot's resource and file system to handle sprite assets.

Overall, by leveraging these features, we created a cohesive frontend experience for our Godot project.

Our Database

For the game we developed, we designed a SQLite database to efficiently store and manage user information and game progress. The database consists of a central "users" table that holds essential user details such as the username, total points, and current game state. Additionally, we created separate tables to track level progress, owned items, and music tracks. These tables are connected to the "users" table through foreign key relationships, allowing for seamless retrieval and updating of relevant data for each user as they interact with the game.

To set up the database, we implemented a create_database() function that is executed when the application starts, provided that the "users.db" file does not already exist. This function establishes a connection to the database and creates the required tables using SQL queries. While writing the queries for each table was a meticulous process, it ensures that the database is properly structured from the beginning. Below is an outline of the database structure.

users table:

- ➔ Primary table that stores user information
- ➔ Fields:
    - ◆ username (primary key): unique identifier for each user
    - ◆ total_points: total points earned by the user
    - ◆ active_music: currently active music track for the user
    - ◆ active_knife: currently active knife for the user
    - ◆ active_plate: currently active plate for the user
    - ◆ current_level: user's current level in the game
    - ◆ current_stage: user's current stage within the level

level_one, level_two, level_three tables:

- ➔ Store user progress for each level
- ➔ Fields:
    - ◆ username (foreign key): references the username in the users table
    - ◆ score: user's score for the specific level
    - ◆ completed: indicates if the user has completed the level (0 or 1)

track_one, track_two, track_three tables:

- ➔ Store music track ownership status for each user
- ➔ Fields:
    - ◆ username (foreign key): references the username in the users table
    - ◆ is_enabled: indicates if the user owns the specific music track (0 or 1)

green_knife, purple_knife, pink_knife tables:

- ➔ Store knife ownership status for each user
- ➔ Fields:
    - ◆ username (foreign key): references the username in the users table
    - ◆ is_owned: indicates if the user owns the specific knife (0 or 1)

purple_plate, blue_plate, green_plate tables:

- ➔ Store plate ownership status for each user
- ➔ Fields:
    - ◆ username (foreign key): references the username in the users table
    - ◆ is_owned: indicates if the user owns the specific plate (0 or 1)

API Calls

API Endpoints: The web server exposes a set of API endpoints defined by me that allow for communication between the game client and the server, enabling various actions related to user management, game progress tracking, and in-game purchases. I utilized the Flask framework to define these endpoints, which simplifies the process of handling HTTP requests and defining routes.

The API endpoints cover a range of functionalities, including:

1. Account Management:
    - `/create_account`: Allows users to create a new account by providing a username. The endpoint checks for username uniqueness and inserts the new user into the "users" table.
    - `/login`: Validates user credentials and grants access to the game if the provided username exists in the database.
2. Game Progression:
    - `/get_level_points`: Retrieves the user's score and completion status for a specific level.
    - `/set_level_points`: Updates the user's score and completion status for a specific level.
    - `/get_current_level_stage`: Fetches the user's current level and stage progress.
    - `/set_current_level_stage`: Updates the user's current level and stage progress.
3. Purchases and Customization:
    - `/purchase_music`, `/purchase_knife`, `/purchase_plate`: Allows users to purchase music tracks, knives, and plates, respectively. The endpoints verify if the user already owns the item and updates the corresponding table/entries to reflect the purchase.
    - `/get_active_track`, `/set_active_track`: Retrieves and sets the user's currently active music track.
    - `/get_active_knife`, `/set_active_knife`: Retrieves and sets the user's currently active knife.
    - `/get_active_plate`, `/set_active_plate`: Retrieves and sets the user's currently active plate.
    - `/get_ownership_details`: Retrieves the user's ownership status for all knives, plates, and music tracks.
4. Leaderboard and Points:
    - `/get_points`: Retrieves the user's total points and increments it by one.
    - `/get_total_points`, `/set_total_points`: Retrieves and updates the user's total points.

    - `/get_top_user_scores`: Fetches the top 5 user scores and ranks them on a leaderboard.
    - `/get_top_scores`: Retrieves the top 5 scores and formats them for external leaderboard integration.

These API endpoints were designed for easy integration into our development environment, making it easy to interact with our database whenever we needed to retrieve or store any information.