Software Design Specification

for

Type-Based

Version 1.0 approved

Prepared by:
Garrett Willis
Chan Rain
Kevin Tieu
Tye Maynard

Procrasti-Nation

October 2, 2024

Table of Contents

1. Introduction

1.1 Goals, Objectives, Statement of scope

1.2 Software Context

1.3 Major Constraints

2. Data Design

2.1 Internal Information

2.2 Global Information

2.3 Temporary Information

2.4 Database Description

2.4.1 User account

2.4.2 Word typer

2.4.3 Math typer

3. Interface Design

3.1 Main Menu

3.2 Main Gameplay Scene

3.3 Game Over/Statistics

4. Appendix

4.1 Test Plan tables

1. Introduction

1.1 Goals, Objectives, and Statement of Scope

This game will be built using a combination of html, css, and javascript on the frontend and Python(and maybe integrated C++ for the performance-heavy non-graphical parts) on the backend to create a web browser based game. The user will choose to play the typing or math version of the game, with various game settings. Then, depending on what is chosen the game will show the user words that need to be typed or math questions that need to be entered correctly before a set point. The question will float across the screen and when it reaches a designated point the user will hit the spacebar or enter to time the answer in time with the music(they have to type it in time to get the points).

1.2 Software Context

Type-Based is a typing and math based rhythm game that is designed to help the player improve on both of these skills. The simplistic design and interface allow for ease of access to even the more tech illiterate users. This allows the user base to be as wide as possible to allow for the greatest effect.

1.3 Major constraints

The major constraints will come from the fact that this will be played with in a web browser and any constraints that come with that. Also the backend will also have constraints that will have to be taken into account.

2. Data Design

2.1 Internal Data

- User information
- Password hash table
- Cookies of the user storing their login information
- Song structure for all songs across all game settings(too complicated to put in a database and requires some random generation of song structure depending on game type and settings – if it's too complicated to put in a database, then should I even list it here?)

2.2 Global Data

- Score history for all users
- Song play statistics
- Songs available

2.3 Temporary Data(not stored in a database)

- Within each song, a running tally of the user's score must be kept. This temporary score is released after the final score is recorded in the database
- The game settings are temporarily stored when running a song, to know how to score everything after the song ends. Harder game settings = higher score. Undecided if this will be done as a score multiplier of the final score or a live increase in the number of tallied points per correct word/math
- A score multiplier that builds up after a certain number of correct answers in a row, that increases the live score tally amount by the multiplier, and gets reset on an incorrect answer

2.4 Database Description

- Format used: Table_name(PRIMARY_KEY(s), attribute_1, attribute_2, …) - The primary key(s) are capitalized, normal attributes are lowercase
- To indicate a foreign key, on each line after the initial table structure, say foreign key (table_key) references Foreign_table_name (foreign_table_key)
- When referencing a date in the score tables, it is not only the calendar date but also the time, it is the expanded date. This is done so that multiple different scores for the same song on the same day can be uniquely identified

2.4.1 User Information

- Player(PLAYER_ID, username, date_joined)
    - foreign key (player_id) references Word_type_scores (player_id)
    - foreign key (player_id) references Math_type_scores (player_id)
    - foreign key (username) references Password_hash (username)
    - This table represents the core identity of the user, used to access score information by querying the score tables in specific ways(described more after those tables are defined)

- Password_hash(USERNAME, hash_key)
    - Used to securely log the user into their account. The hash_key is calculated from the password. If the hash key from the entered password is the same as the hash_key in the table for that username, the user becomes successfully logged in

2.4.2 Word Typer

- Word_type_scores(PLAYER_ID, SONG_ID, DATE_PLAYED, difficulty_level, score)
    - This table stores the score history for every user across every game of word_typer.
    - Query by player_id to get all the scores for that user
    - Querying by song allows the average score for a song across all users to be calculated. Add a player id and it is the average score for that user for that song

- Querying by player_id and by date allows for the user to see different timeframes of their score history
- Querying by player_id and song_id allows the user to see the score history for a specific song
- Querying by song_id and difficulty_level allows us to see the scores of that song at that preset difficulty_level
- All the other game settings besides difficulty level are represented in changes to the score(special game settings apart from only the difficulty level will lead to different scoring rates – harder settings higher scores). Undecided if difficulty_level is necessary to store or if all differences in game settings can be reduced to differences in scores(regardless, storing difficulty_level provides useful queries)

- Word_type_global_scores(SONG_ID, highscore_1, highscore_2, highscore_3, highscore_4, highscore_5, average_score)
    - For purpose of compiling the most relevant song score information that the user can compare themselves with
    - Information can be found through the overall score tables, but it is collected here for quicker access(without having to do the calculations as often)
    - Highscores are recalculated if a user's game score is larger than highscore_5(which is set by default to 0)
    - Average score is recalculated either after a fixed amount of time, or after each game, undecided
    - Might not be implemented, but should help performance if implemented

2.4.3 Math Typer

- Math_type_scores(PLAYER_ID, SONG_ID, DATE_PLAYED, difficulty_level score)
    - This table stores the score history for every user across every game of math_typer
    - Query by player_id to get all the scores for that user
    - Querying by song_id allows the average score for a song to be calculated
    - Querying by player_id and by date allows for the user to see different timeframes of their score history
    - Querying by player_id and song_id allows the user to see the score history for a specific song
    - Querying by song_id and difficulty_level allows us to see the scores of that song at that preset difficulty_level

    - All the other game settings besides difficulty level are represented in changes to the score(special game settings apart from only the difficulty level will lead to different scoring rates – harder settings higher scores). Undecided if difficulty_level is necessary to store or if all differences in game settings can be reduced to differences in scores(regardless, storing difficulty_level provides useful queries)

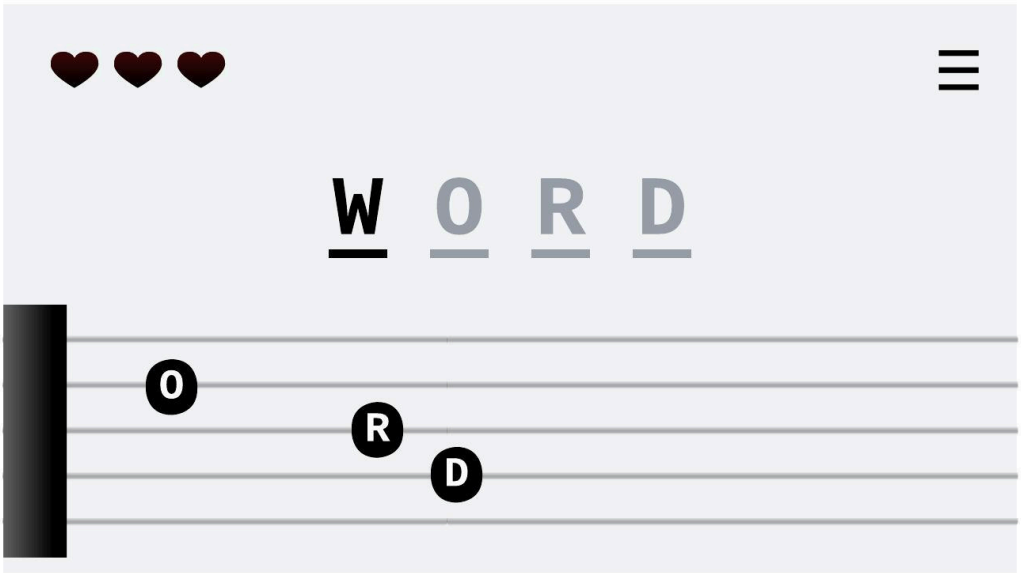
- Word_type_global_scores(SONG_ID, highscore_1, highscore_2, highscore_3, highscore_4, highscore_5, average_score)
    - For purpose of compiling the most relevant song score information that the user can compare themselves with
    - Information can be found through the overall score tables, but it is collected here for quicker access(without having to do the calculations as often)
    - Highscores are recalculated if a user's game score is larger than highscore_5(which is set by default to 0)
    - Average score is recalculated either after a fixed amount of time, or after each game
    - Might not be implemented, but should help performance if implemented

3. Interface Design

3.1 Main Menu

- The user can edit any game settings/modes from the main menu, and view information about the current version of the game they are playing.

3.2 Gameplay Scene



-The user is shown a word they must type before the corresponding letter-notes reach the black bar on the bottom left of the screen. They are also shown their attempts remaining (hearts), and have access to a hamburger menu dropdown where they can configure game settings available during gameplay.

3.3 Game Over Scene

-The game over screen displays statistics and other values generated during gameplay, and gives the user options to continue.

4. Appendix

4.1 Test Plan Tables

Function Area	Requirement ID	Method/Function	Test data/Input	Expected output	Actual Output
User Authentication	FA-001	loginUser	Correct username and password	User is logged in and directed to the gameplay scene.	
User Authentication	FA-002	loginUser	Incorrect username and/or password	Error message indicating login failure.	
Game Settings	FA-003	Update Game Settings	Change difficulty level to "Hard"	Settings updated successfully, reflected in gameplay.	
Gameplay	FA-004	Display Word	Game running with word "example"	Display the word "example" for the user to type.	
Scoring	FA-005	Calculate Score	User types "example" correctly	Score incremented by the correct amount.	
Scoring	FA-006	Apply Score Multiplier	Correct answers with multiplier active	Score should reflect the multiplier effect.	
Database	FA-007	saveScore	Score of 100, player ID 1	Score saved in the database for the user.	
Database	FA-008	getUserScoreHistory	Player ID 1	Retrieve list of scores for player ID 1.	
UI Elements	FA-009	renderHearts	Game with 3 attempts remaining	Display 3 hearts on the screen.	

Game Over	**FA-0010**	**displayGameOver**	**Final score of 150**	**Display "Game Over" with final score of 150.**	
Temporary Data	**FA-011**	**calculateTempScore**	**Correct inputs during gameplay**	**Temporary score reflects the ongoing game correctly.**	
Song Management	**FA-012**	**loadSong**	**Song ID 1**	**Song plays without issues.**	
Performance	**FA-013**	**loadTest**	**100 simultaneous users**	**The game performs without lag or errors.**	
User Data	**FA-014**	**persistUserData**	**User logs in and plays a game**	**User data remains consistent after logout and login.**	