
GAME

Welcome To Our Presentation

Title: TypeCaster Battle: AI-Adaptive Typing Game

Subtitle: A Dynamic Difficulty Adjustment System using Python & Machine Learning

OUR TEAM

Ahmed Haitham

Wed Ahmed

Ruqayah Alaa

The Problem & The Solution

Objective: Most typing games have static difficulty levels (Easy, Medium, Hard) that don't fit every player.

The Problem: A static "Hard" mode might be too fast for some but too slow for others. Players either get bored or frustrated.

Our Solution: An "Intelligent" game that watches you play. It analyzes your typing speed (WPM) and accuracy in real-time to generate words that are *exactly* challenging enough for you.

Game Concept & Gameplay

1. Genre: RPG-Style Typing Battle.

   Core Loop.

2. Attack: Type the displayed word correctly to damage the enemy.

3 Defend: When the enemy attacks, you must type a "Counter Word" quickly to block damage.

4 The Twist: The enemy doesn't just have more health; it gets "smarter." As you play better, the enemy starts using trickier words (e.g., words with 'Z' or 'X', or capitalizing letters like PyThOn).

System Architecture (How it Works)

1- The "Director" (Rule-Based AI):

Watches for specific physical mistakes.

-Example:If you keep missing keys on the left side of the keyboard (Q, W, E, A, S), it intentionally gives you more "Left-Hand Dominant" words to force you to practice.

2-The "Predictor" (Machine Learning):

Uses Logistic Regression (Scikit-Learn) to predict if you will fail the *next* word.

If it predicts you will succeed easily (>80% probability), it raises the difficulty immediately.

The "Brain" of the Game (Libraries Used)

1-Pygame: Handles the graphics, game loop, and input detection.

2- Scikit-Learn: The core AI library. We use it to train a Logistic Regression model on the player's past performance.

3- NumPy: Used for efficient numerical calculations (handling arrays of game data for the AI).

4- JSON: Used for "Memory." The game saves every session, so the AI has more data to learn from every time you play.

How We Measure Difficulty

The game doesn't just guess difficulty; it calculates it mathematically using our WordLibrary Class:

Word Length: Longer words = Higher score.

Rare Letters: Words with Z, X, Q, J add difficulty points.

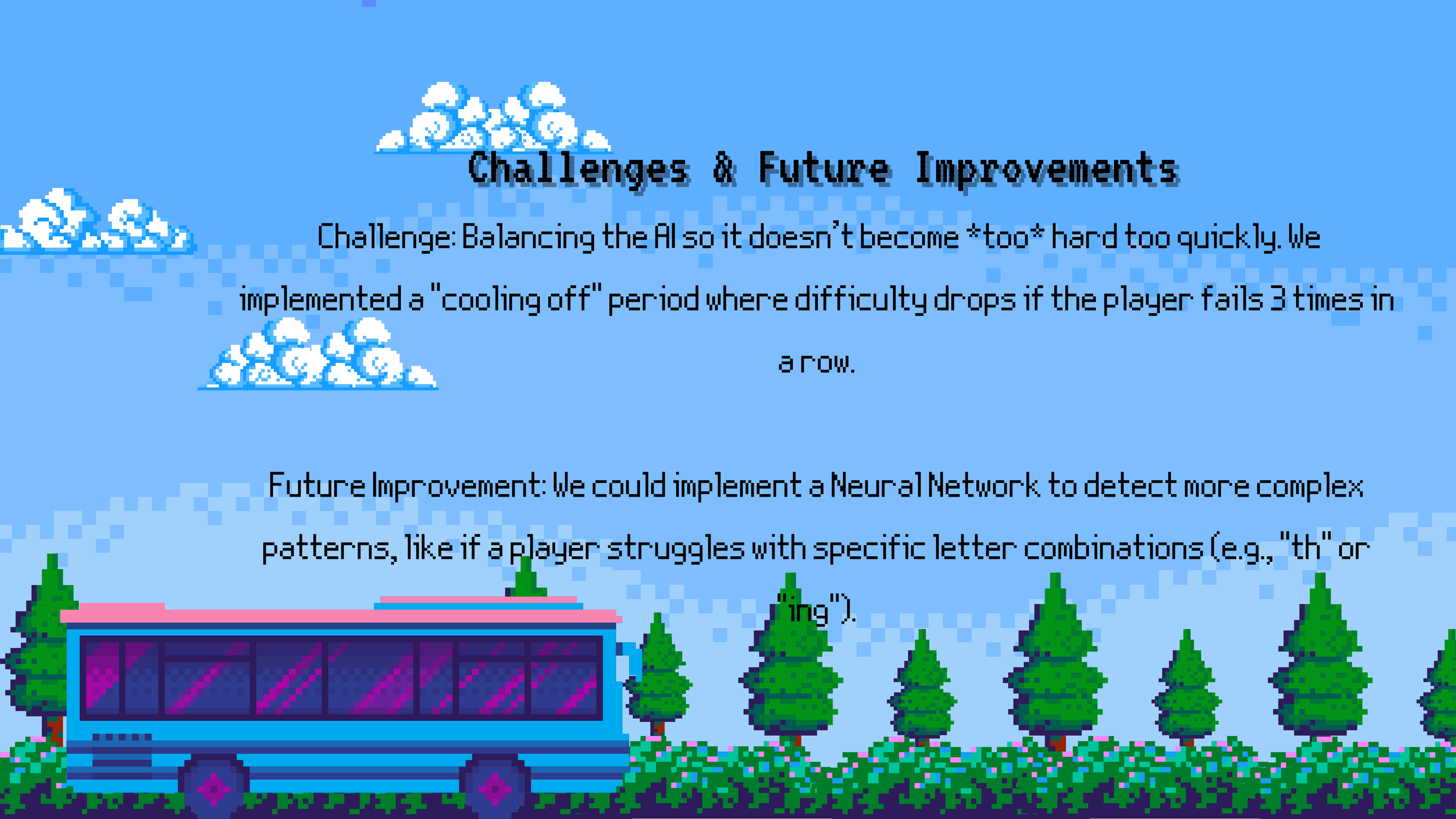
Keyboard Row Jumps: Words that force you to jump between the top and bottom rows (e.g., "Muzzy") are scored higher than "Home Row" words (e.g., "Salsa").

Result: A difficulty score from 1-10 is assigned to every word dynamically.

Challenges & Future Improvements

Challenge: Balancing the AI so it doesn't become *too* hard too quickly. We implemented a "cooling off" period where difficulty drops if the player fails 3 times in a row.

Future Improvement: We could implement a Neural Network to detect more complex patterns, like if a player struggles with specific letter combinations (e.g., "th" or "ing").




THANK
YOU