Type Type Revenge

Anabelle Chang
Princeton University
anabelle@

Donovan Coronado
Princeton University
dtc2@

Christine Kwon
Princeton University
cmkwon@

Christy Lee
Princeton University
christyl@

Abstract

Type Type Revenge is a 3D web game built with Three.js to provide an interactive environment for users of various typing levels to practice their typing skills. In this game, random letters fall down the screen in time with music, and players must type the corresponding keyboard key as a letter reaches its target at the bottom of the screen. There are two song options and three levels of difficulty offered, and point values increase as the player reaches longer streaks of successfully typed letters.

1 Introduction

1.1 Motivation and Goal

The goal of our project is to create a 3D game that helps users practice their typing skills by pressing down keyboard keys to visual cues. There already exist many variations on typing practice games, as well as several versions of rhythm games which require the player to press keys in time with music. The former serves an educational purpose while the latter is a popular form of gaming entertainment, so Type Type Revenge aims to combine the practical and enjoyable aspects of both into one game.

1.2 Related Work

"Tap Tap Revenge" is an iPhone game initially released in 2008 by the mobile app developer "Tapulous" [5]. It quickly became the most downloaded free game amongst iPhone and iPod Touch users [4]. In this game, the user taps on-screen buttons that move in time to music when they reach a line at the bottom of the screen. Points are gained when the buttons are tapped at the correct time. However, as of January 2014, the game has been removed from the App Store. "Tap Tap Revenge" inspired both the idea and the name for Type Type Revenge.

The Japanese game "Dance Dance Revolution" (DDR), introduced in 1998, is a music video game with a similar concept. Players press arrow targets laid out on a dance platform with their feet in response to visual cues matching a song rhythm in order to gain points [2]. However, this product has a greater focus on dance and exercise.

As computer use became more prevalent in education, elementary schools began to adopt typing practice into its curriculum as well. By the time the authors of this paper were in elementary school, it was common practice for typing to be taught through games targeted at young children. Today, there is an abundance of typing games offered online for free for all ages.

1.3 Approach

Our game makes the following contributions:

- We created our 3D game for use on a Web browser using the Three.js framework rather than creating a mobile device game. The browser-based nature of the game allows users to practice typing since they will be using a keyboard to play the game. Furthermore, a web game is easier for more users to access since it does not require them to own a game console or a smartphone.
- We provide a free service for users to practice their typing skills. Doing so allows our game to be accessible to a wider audience.

Before the game begins, the user is greeted with instructions on how to play the game, and is prompted to choose the difficulty level of the game and a song selection. When the game begins, a series of glowing letters begin falling from the top of the screen. The sequence and colors of the letters is determined by the melody of the music. The user can press the corresponding letter key on the keyboard when a falling letter on the screen matches up with its target at the bottom in order to earn points. As the user accumulates a higher streak, each correctly pressed letter earns more points. The letters continue to fall until the song terminates. Once the song is over, we display a score summary, and give the user the option to restart the game again.

2 Methodology

2.1 Music Overview

This game uses two different file formats for songs. MP3 versions of songs are played for the user to hear, while MIDI file versions of the same songs are utilized to obtain the sequence of notes (which determine which letters fall and where). The MP3 files were downloaded from various online sources, while the MIDI files were either generated manually or using AnthemScore based on the MP3 files. Manual generation involved using GarageBand for Mac to record a simple version of the melody in time with the full arrangement of the song. AnthemScore is an automatic music transcription software trained on a neural network which was used to convert MP3 song files directly to MIDI versions. This dual use allowed for the full sound of a MP3 song file and the easy note access of MIDI messages.

2.1.1 Note Parsing

Note parsing of MIDI files was done separately using Java and the Javax.sound.midi package. The parsing script initialized a MIDI sequencer, which was then used to parse MIDI messages in real time. MIDI messages come with a variety of different commands, including note on, note off, key change, and control change messages, among several others. For the purposes of this project, we only used messages with note on commands, which signified the start of a note. For each of these messages, we wrote the time that the note played (measured in milliseconds elapsed from the start of the song), the pitch of the note (a value between 0 and 127 which corresponds directly to a musical note), and the velocity of the note (a value between 0 and 127 which conveys the amount of force a note was played with, which is perceived similarly to volume) to a text file.

We used a separate Python script to convert the information in the text file into a JSON file for easy parsing. This was done in Python rather than in the initial Java file due to some local issues with Maven project dependencies and for the sake of time. The resulting JSON format is as follows:

```
{
  "notes": [
    {
```

```
      "time": "1027",
      "note": "62",
      "velocity": "70"
    },
    {
      "time": "1034",
      "note": "66",
      "velocity": "87"
    },
    ...
  ]
}
```

2.1.2 Note Playing

After the user selects a song of their choice, the notes of the song then determine what letters fall, where in the screen they fall, and their assigned colors. As mentioned in the earlier subsection, MIDI note values range from 0 to 127. These values map directly to a musical pitch, which can be found by calculating (note value - 21) % 12. This will result in a value between 0 and 11, where 0 maps to A, 1 maps to A♯, 2 maps to B, etc. up the chromatic scale. This is the value used to determine the aforementioned letter qualities.

To understand how a note maps to a letter, it is first necessary to explain the layout of the game screen. The QWERTY keyboard is split into three sections based on the positions of the keys on the keyboard, and each of these sections corresponds with a third of the game screen. That is, because the letter 'q' is on the left end of the keyboard, it would fall somewhere in the leftmost third section of the screen. Similarly, because the letter 'k' is on the right end of the keyboard, it would fall somewhere in the rightmost third section of the screen. The first four notes (A, A♯, B, C) are mapped to the leftmost section, the next four notes (C♯, D, D♯, E) are mapped to the middle section, and the last four notes (F, F♯, G, and G♯) are mapped to the rightmost section. When a note is played, we thus find which third it corresponds to, randomly select a letter in that third, and randomly select a position within that third of the screen for the letter to fall. This was calculated as such to include a correlation between notes and letters, but to also ensure a more even distribution of which letters fall and where. With the random component, each of the 26 letters of the alphabet have a more equal chance of representation no matter what key the song is in, and all falling letters are visually distributed more evenly across the screen.

The twelve different musical pitches were each mapped directly to twelve different colors. This was based off of the note to color mapping approach used in [3] (wherein increasing frequencies of notes are mapped to increasing frequencies of colors), and can be seen in Figure 1.

2.2 Components of the Game

We use the following 3D objects in our game:

2.2.1 Letter

The Letter class defines the multi-colored lowercase letters that fall from the top of the screen during the game. We use the Three.js FontLoader class to load a typeface (Helvetiker Bold) that is stored in a JSON format. Then, we convert the loaded font to a 3-dimensional text mesh composed of the edge geometry and material. As a Letter is added at a random location on top of the screen to the scene, it will immediately begin falling down the screen. This was achieved by a simple use of TWEEN.js. Upon completion of the fall, the letter and its corresponding target will immediately be disposed and removed from the scene.

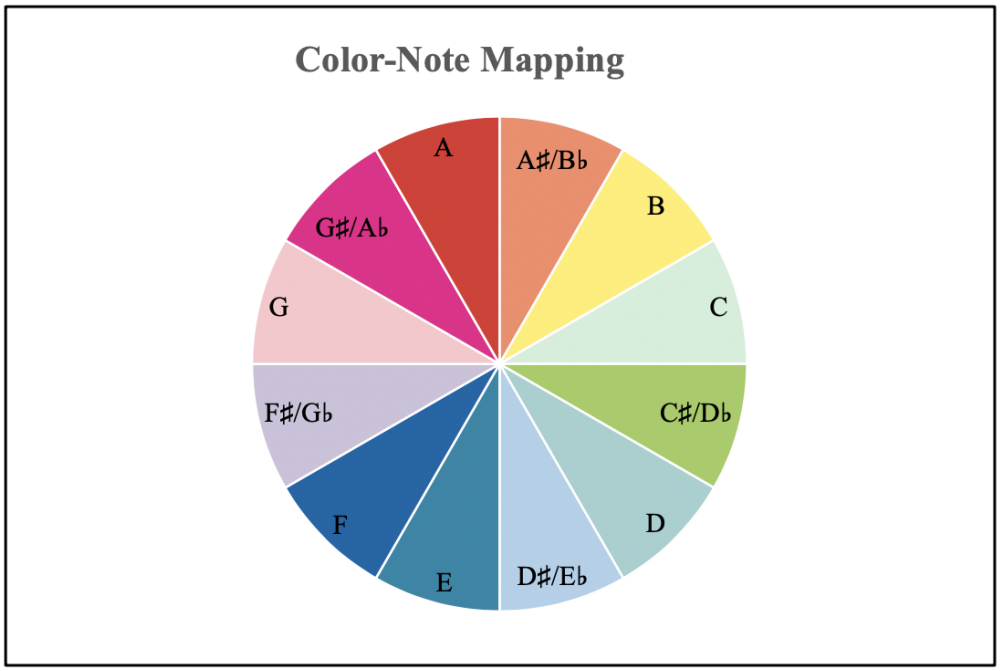

Figure 1: MIDI Note to Color Mapping (from [3])

2.2.2 Target

For every falling Letter object, we create a corresponding Target object. This object is placed near the bottom of the screen, and is a glowing white color. If the player presses the correct key just as the falling Letter passes through the Target object, he or she gains a point.

The Target class has additional effects for when the player presses the correct key as the Letter passes through the Target object. When this occurs, the edged mesh will change into a solid mesh filled with the corresponding color of the Letter object. This was done by created a solid mesh with its visibility initialized ‘false‘ in the constructor of the Target class. The visibility property will then only be set to ‘true‘ in the correct scenario.

Another effect we wanted to include was an indication to the user that he or she has missed hitting a Letter. To accomplish this, we checked if the falling Letter was past the the Target’s position, then set the Target’s edged mesh color to red if the eventListener was not triggered for a keydown.

2.2.3 Cube

The spinning cube that is present at the top of the game is a mesh object created in Maya, and displays the title of the game on one side and the initials of the creators on the other. The object is loaded and rendered in the Cube.js class, where the material is also specified. Since we did not intend for the Cube object to reflect the Bloom post-processing effect, we rendered the object in a separate scene.

2.2.4 Error Bar

The error bar is a rectangular mesh part of the game’s scene, and it serves to indicate to the player when he or she has pressed the incorrect key. To create this function, a set was used to keep track of all the current letters in the scene, and as each letter falls out of the scene (and subsequently gets disposed), it also gets removed from the set. Whenever the user presses a keyboard key, we used an eventListener to obtain the key and check if the letter typed is part of

the scene. If not, the error bar lights up by setting its visiblity to ‘true‘ for 300 milliseconds, using the setInterval() function, before it is turned off. If the letter is part of the scene, there are additional requirements to check:

- First, the corresponding falling letter must be on top of its target to be considered correct. If the falling letter is out of range of its target, then the error bar will light up.
- Second, if the user has spammed his/her keyboard with the correct key, meaning that he has typed the letter once again after already correctly pressed the key when the letter and target have aligned, the error bar will light up.

2.2.5 Rain Effect

We generate line segments of various lengths at random x coordinates and make them fall from the top of the screen. Once they reach the bottom, they are disposed. The rain effect dynamically changes with the user's streak – as he or she increases the number of letters he or she correctly types in a row, the rain will change color.

2.2.6 Percentage

Upon completion of the game, a 3D percentage of how many letters you hit on time will pop up on the screen. This percentage was created using a TextGeometry, and its color will vary with the percentage it displays. For example, a 100% score will be shown in green, and a 50% score will display an orange-ish yellow color, while a 0% score will be red. The Percentage object is then disposed upon starting a new game.

2.3 The Gameplay Elements

2.3.1 User Interface

The Landing Page
We decided to incorporate a landing page because we wanted to describe our game's premise to the player. It features 4 traditional "keyboard" keys spelling "play" on the lower half of the screen. When the player presses "p", each letter will impress into the key itself, simulating a keyboard key being pressed, before taking the player to the main page of the game.

Key Presses
We added eventListeners to detect if the user is pressing a key on the keyboard, and if so, we look at the keyCode in order to determine which key is currently being pressed. If the keyboard event matches up with a letter that is intersecting with its target at the time, the letter lights up and the user gains a point. However, if the user presses the incorrect key, the error bar lights up and their streak is reset.

Mute Toggling
While music was a fundamental part of our game, we recognized that some users may not want to listen to the song we were playing, may not have headphones available, or they may not be in an environment that permits music playing. To mitigate this problem, we decided to incorporate a mute functionality to the game– when the game starts, the player has the option to click on the volume icon below the scoreboard to toggle the music volume on and off.

2.3.2 Letter Placement

We were intentional with the position of the letters on the screen, which corresponded with their positions on the keyboard. This was because our game was fundamentally a typing game, and we hope that doing this would help our players learn to associate letters and their respective positions on the keyboard. More detail on how letter positions were determined can be found in Section 2.1.2.

We created a PositionFinder class in order to keep track of the positions of the letters being added to the scene, and to prevent potential overlap of the letters. The PositionFinder was initialized to have a Boolean array the size of the width of the screen, and to add a letter, would calculate an index at random and set the value at that index to be 'true'. Once the Letter was disposed of the scene, the index of its position in the PositionFinder would be cleared.

We noticed that even though the PositionFinder didn't allow for letters to share a position, if there were two wider letters next to each other (like 'w'), then they would still overlap. Therefore, we added functionality that checked one unit to the left and right of the potential position to make sure that two letters had space between them.

2.3.3 Score Tracker

We created a HighScore class to keep track of the player's running streak and total score. The streak value maintained how many letters the player had correctly pressed in a row, while the score increased with each correct letter. Scores were calculated using streak multipliers, in which a streak below 5 would add 1 point for each correct letter, a streak between 5 and 10 would add 2 points for each correct letter, etc. for a maximum of 5 point letters (for streaks above 20 letters). The score and streak values are displayed in a box in the top right corner of the screen for the player's information.

The HighScore class was also used to display encouraging messages for the player at different milestones. For example, when the player reaches a streak of 10, a text will pop up in the background to read "perfect!" for 1.5 seconds before disappearing again. At the end of the game, this class provides the final game score.

2.3.4 Difficulty Levels

We decided to have three difficulty levels: easy, medium, and hard. The default difficulty is easy, and the user can select any difficulty in the menu. The difficulty level controls how many notes from the midi file we process and convert to letters on the screen. The higher the difficulty, the more notes we display on the screen.

2.3.5 Game Modes

We added support for the user to select between two different songs: Grenade by Bruno Mars and the Wii Theme Song. We selected these songs because they have relatively few instruments and therefore a more noticeable tonal pattern. We parsed the notes from the midi files using our note parser in section 2.1.1. Then we used the notes from the resulting JSON file to determine when and where on the screen the Letter objects would fall. When a user selects a song, we load the mp3 of the corresponding song into our global sound object. Then we set a variable in the scene that determines which set of notes to read from. We also scale the difficulty based on the song since there are a different number of notes in the two songs. The combination of song choice and difficulties gives the user 6 different game modes that they can choose from.

2.4 Post-processing Effects

We use the Three.js UnrealBloomPass to add an outer glow effect to the edges of all of our 3D objects.

3 Results

Our baseline measure of success was the completion of a basic version of our goal: a 3D game that helps users practice their typing skills by pressing down keyboard keys to visual cues. We were ultimately successful in this goal, and in many of our stretch goals as well. We were able to parse real time song information, select letters based on notes, and have them fall in time with the song. We were also able to add in game functionalities such as reactions to correct and incorrect key presses, score and streak keeping, reactions to score throughout the game and at the end through

messages, changes in background color, and summary messages. Moreover, we were able to use Three.js as well as basic graphics coding to shape the appearance of every aspect to our intended aesthetic.

Experiments included testing multiple songs, multiple levels of difficulty, various levels of correctness, and various manipulations of the webpage (i.e. restarting games in the middle or the end of a current game, muting or unmuting the song, and other interactions with the page).

We also solicited feedback from our peers and adjusted the game accordingly. One of the top comments was the difficulty of our game when we initially only had one option to choose from. As a result, we were encouraged to add different difficulty levels, which we were able to create. Another comment was providing more feedback to the user upon game completion. This inspired us to add a percentage score, in addition to the final score and highest streak, to the final post-game render.

4 Discussion

Holistically, we felt that the approach we took with our project was very promising – we accomplished almost all of the functionalities we set our mind to when initially planning this game. However, we did run into issues with a select few functionalities we tried to implement, one of them being able to toggle between playing and pausing the game. This was due to the nature of the TWEEN library, which did not allow us to stop a TWEEN in its intermediate state (that is, while it is moving). Despite multiple approaches, including an attempt to use CreateJS/TweenJS, it did not end up working out. We think this problem could be resolved, perhaps in the near future, with a different library, or by moving the Letter objects manually in the update loop using the timeStamp variable.

Another issue we had was with the asynchronous nature of JavaScript. When switching songs, we would load the new song into our sound variable with an audioLoader object, but it took about one second to do so. Since JavaScript is asynchronous, sometimes the line to play the song would run before the audioLoader finished reading in the mp3 file. First, we tried to remedy this issue by making the restart function async and adding an await call when the audioLoader read in the mp3. When that didn't work, we decided to issue an await call and sleep for 1.2 seconds before playing the song and loading the new game. JavaScript doesn't have a built in sleep function, but we used a Promise object to achieve the same functionality.

5 Conclusion

We believe we quite effectively attained our goal that we had set from the start. Our project's final iteration aligns with the digital identity we initially envisioned, and it seemed to be well-liked by our peers through the feedback form. We were even able to meet some of our stretch goals such as different difficulty levels and song options. We believe our game is enjoyable, polished, visually pleasing, and fully functional as well.

Potential next steps would be to implement additional support for our application, such as a Typing Practice mode, or adding options for including numbers and symbols in the game. Another option would be to create a mode where the letters spelled out lyrics in the song.

Additionally, we would want to revisit creating a play/pause button, as well as dealing with the asynchronous nature of JavaScript. We were able to get around the asynchronous nature by inserting an "await sleep()" command between our song loading and playing, but this isn't the most ideal way to solve that problem. Right now, we sleep for a set amount of time, but it might take different CPUs more or less time to load the song. Ideally, we can just await the function call that loads in the song and then play it after it is done.

Creating this game was quite a fruitful and enriching experience. All members of our team are happy to see how a seed of an idea ultimately resulted in a fully-fledged app that has many of the functionalities we first set out to create. To be

able to build this app almost from scratch also gave us an opportunity to understand and explore ThreeJS's libraries, which was extremely rewarding. Because our only prior experience was using ThreeJS for class assignments to create filters, effects, and whatnot, it is fulfilling to finally be able to go over our prior assignments' code and actually be able to understand how the scenes are being built.

6 Contributions

Much of the graphics responsibilities were evenly shared among the members of our group. In order to efficiently complete our project, we had a shared to-do list with a number of features we desired to implement for the game, and each team member could pick-and-choose to complete the the tasks on the list. This structure was preferred over delegating specific parts of the project to each member to ensure that no team member would fall behind in completing a list of set responsibilities, especially since many of us had very busy reading weeks this semester. We maintained active communication to ensure everyone contributed equally. Listed below are some of the main contributions each team member made.

6.1 Anabelle Chang

- Created Letter class and automatic falling animation
- Added Target effects (Target glowing in color and changing edge color when incorrect)
- Error bar functionality and logic
- Dynamically changing rain with colors
- Mute toggling functionality during the game
- Added disposal functions of Letter/Target with ResourceTracker from ThreeJSFundamentals, as well as disposal of objects in the scene upon game completion
- Added support for an editable index.html
- Created landing page and its subsequent animations, effects, and disposal
- Added display for scoring and effects with varying Percentage colors, messages, highest streak
- Created PositionFinder class and foundation for the HighScore class
- Attempted pause functionality with TWEEN.js and CreateJS/TweenJS

6.2 Donovan Coronado

- Letter mapping from position on keyboard to position on screen
- Read json of song notes and created corresponding letters to the screen
- Prevent overlap of falling letters
- Added support for multiple songs
- Added support for multiple difficulties
- Included ability to switch songs in the middle of games
- Linked gameplay options frontend to the backend
- Included code to load and play mp3 songs corresponding to song choices

6.3 Christine Kwon

- Create and integrate Target class
- Check Letter-Target intersection + subsequent disposal
- Add keyboard event listeners
- Check correctness and timing of keyboard press
- Implement Note-color mapping
- Add delay functionality for Letter generation
- Style instructions page and score board
- Create spinning title Cube
- Create Cube scene and integrate with main scene
- Adjust scene lights
- Map camera coordinates to screen coordinates
- Create rain effect

6.4 Christy Lee

- Create Java program to parse NOTE_ON messages from a MIDI song file
- Create Python script to convert data into JSON format
- Create instruction/game selection screen
- Create game over screen
- Link the various screens together smoothly
- Add streak component, weighted scoring, and mid-game messages
- Add functionality for starting a new game in the middle of a current one
- Attempted pause functionality
- Adjust various aesthetic features

References

[1] "Three.js cleanup." [Online]. Available: https://threejsfundamentals.org/threejs/lessons/threejs-cleanup.html

[2] A. D. Kloos, N. E. Fritz, and S. K. Kostyk, "Video game play (dance dance revolution) as a potential exercise therapy in huntington's disease: a controlled clinical trial," in *Clinical Rehabilitation 27 (11)*, 2013, pp. 972–982.

[3] C. Lee, "A framework for music visualization using fractals."

[4] D. Moren, "Report: Tap tap revenge is iphone's most popular app." [Online]. Available: https://www.macworld.com/article/1139906/appstore_data.html

[5] N. True, "Tap tap revenge 2: A history." [Online]. Available: https://web.archive.org/web/20101221163917/http://tapulous.com/blog/2009/03/tap-tap-revenge-2-a-history/