Keyrannosaurus Text

Keyboard Kardashians

Kaelin Hoang & Khanhly Nguyen

Abstract

Keyrannosaurus Text is a typing game with the objective of saving dinosaurs from meteors that are falling to earth. By typing a word correctly, a meteor will be destroyed, thus saving the dinosaurs from destruction for a few more seconds. Each meteor destroyed increases the user's score by the number of letters within the word. After four meteors have hit the ground, killing the dinosaur friends in the process, the player loses the game.

The exciting game tests the player's typing ability in a fun and interactive way. The words the user needs to destroy differs in length by selecting a random word from easy to difficult word lengths at any point in time.

URL

http://ec2-54-172-38-139.compute-1.amazonaws.com

Deployed both website and the database.

Introduction

Keyrannosaurus Text is a game to improve typing skills of the user. The web page opens to a bright and exciting looking game which features a simple set of instructions on the onset of the game. Before starting the game, the user can enter their username in order to see their scores after the end of the game. Below the instructions and username textbox is the play button; clicking the button initiates the start of the game.

Once the game begins then word-meteors begin to fall from the sky. In order to destroy the meteors before they hit the ground, the user must correctly type the word corresponding to the meteor. By pressing the Enter key, the user can destroy the meteor and save their dinosaur friends at the bottom of the screen. The user is given four chances to save the Earth before the game is over. After the fourth meteor hits the ground, the game is over and the user's score is displayed.

Keyrannosaurus Text is a typing game that utilized PIXI.js and the MVC Architecture in ASP NET Core. The game integrates Professor Germain's clever Connect 4 implementation along with many different aspects of additional functionality. This involved a main JavaScript file (Game_Controller.js) which was the central controller for the entire game. Everything started in this file and eventually came back around to end in this file. The game's functionality also involved useful classes such as Button and Meteor to keep track of the movements and images of each individual item.

The game was able to listen for keyboard events by using our own keyboard listener to see when the user presses Enter in order to check the correctness of the word they just typed. Additionally, we used PIXI's app.ticker in order to time the creation and dropping

mechanisms of the meteors. Meteors are created infinitely so long as the user does not lose the game.

Features Table

Feature Name	Scope	Programmer	Time Spent	File/Function	LoC
Infinite Meteor Creation	Front End	Kaelin Hoang	9	Game_Controller.js (drop_in_column)	55
PIXI Textbox	Front End	Ly Nguyen	9	Game_Controller.js (load functions)	70
Random Word from Database	DB	Kaelin Hoang	2	WordsController.cs	31
Enter Key Listener	Back End	Ly Nguyen	3	Game_Controller.js (keyboard listener)	65
High Score Storage	DB	Kaelin Hoang	3	HighScoresController.cs	56
Pop-up Start Screen	Front End	Ly Nguyen	5	Game_Controller.js (load functions)	67
Pop-up Game Over Screen	Front End	Ly Nguyen	2	Game_Controller.js	23
Individual ticker for each Meteor	Back End	Kaelin Hoang	3	Meteor.js	49
Textbox Auto-Focus	Front End	Ly Nguyen	3	Game_Controller.js	14
Dinosaur Lives Removal	Front End	Kaelin Hoang	7	Game_Controller.js	36

Individual Contribution

Team Member	Time Spent on Proj	Lines of Code Committed
Khanhly Nguyen	27	45,363
Kaelin Hoang	27	43,139

Summary

Our game uses a database to store words and users. This database is queried multiple times throughout the game to generate words that then get displayed on the meteors. The second table within our database stores the names of previous players along with their high scores. These high scores were not able to be displayed on the game over popup because function calls were being made asynchronously, so users and their high scores were being added to the database multiple times within a small time frame. When we tried working around the asynchronous issue, we ran into a TICKER problem where all animation would cease to continue after the ticker was stopped along with any ajax calls.

Since we built our game primarily using PIXI.js, we had many workarounds to get the game to work. One of the main issues we ran into was PIXI's lack of a text box object. Since our game is a typing game, this posed as a huge issue, but we were fortunate enough to find an open-source library that provided this functionality in PIXI. PIXI also did not have a built in keyboard listener, so we had to program this ourselves. When the player pressed enter after successfully typing in a word, there was no built-in functionality to detect that enter was pressed. We had to create a listener for this, but we then ran into the issue of having the player click on the text box after pressing enter. We then tried "focusing" the text box each time enter was pressed in order to have the text box be "selected." This raised another issue because after calling a manual focus after each enter, the enter listener that we created would be forgotten. We added the scope within the enter listener, so each time enter was released, we hard-coded the

scope back to the enter listener. From this, we ran into issues with the delete key where characters would no longer be deleted, so this led to us reimplementing the backspace event.

Another big contribution that isn't seen visually is our dynamically created meteors. Since each meteor needed an associated word, we needed to keep track of each meteor currently on the screen and what word is associated to each meteor. Once the meteor is deleted, we needed a way to delete the ticker associated to the meteor along with removing it from the screen. When dynamically creating each meteor, we assigned each with its own ticker along with creating a meteor object to have a word attribute. This allowed us to keep track of which meteor was deleted. All meteors were kept in a list and we would perform a search and splice to find the meteor that was closest to the bottom of the screen and remove it from the list. If the player didn't remove a meteor closest to the bottom of the screen, we would maintain a splice functionality. Searching and splicing was also required even if the user did not successfully type in the word. If the meteor reached the bottom of the screen, we would search for it and remove it from our list along with decrementing the user's allotted life.

Overall, I believe our team did good considering how many obstacles we had to overcome as a two women team. Although we ran into a lot of roadblocks throughout this project, it was a great learning opportunity and we can both agree that we learned a lot about JavaScript and web software development.