Project Details

Rachel Backert, bac2@umbc.edu, programmer
Located at /afs/umbc.edu/users/s/l/slupoli/pub/cs443/bac1/Project1/proj1.zip

Introduction

This web application provides a game to help Pre-K through first grade students practice reading analog clocks, and data collection to help teachers track students' progress.

Gameplay

Level Design

The time and the clock face become more complex across ten levels. There are five difficulties of time shown: 1) On the hour (12:00, 1:00…); 2) Multiples of thirty minutes (12:00, 12:30…); 3) Multiples of fifteen (12:00, 12:15, 12:30, 12:45…); 4) Multiples of five (12:05, 12:10, 12:15…); and 5) Multiples of one (12:01, 12:02…).

In addition, there are two difficulties of clock face; the "easy" clock face has tick marks and numbering for every minute, while the "hard" face has tick marks for every minute and numbering for every hour. Students must read a time difficulty in both the easy and hard clock faces before moving to the next difficulty. So, the game levels follow this pattern:

1. on the hour, easy face
2. on the hour, hard face
3. multiples of thirty, easy face

etc.

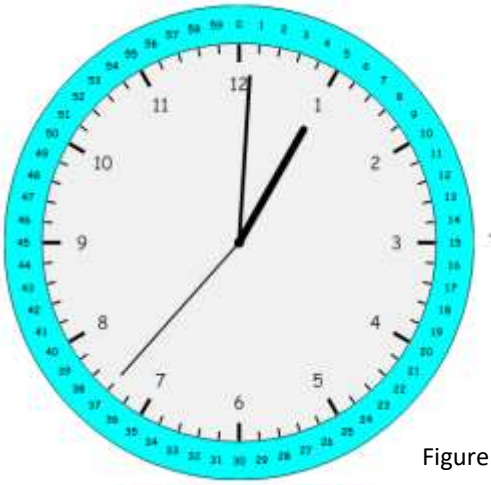

Figure 1

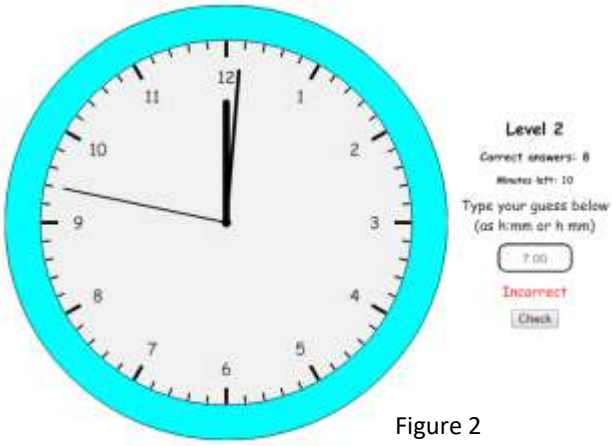

Figure 2

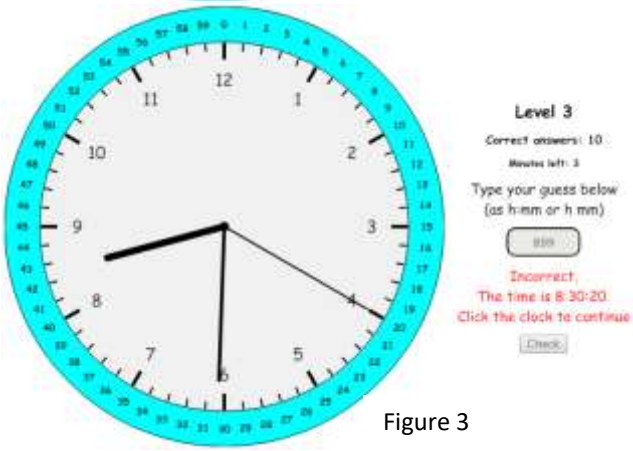

Figure 3

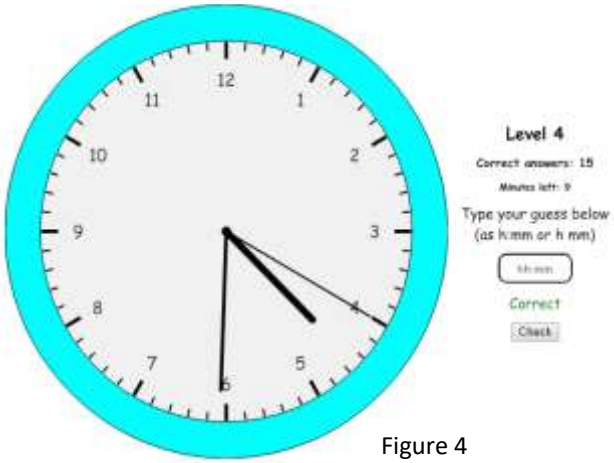

Figure 4

Advancing Levels

Students must read five analog times correctly to advance levels. They get three chances to type the correct answer for each time shown. If their third answer is incorrect, the game displays the correct answer in digital format, and doesn't show a new time until the student clicks to continue (figure 3). So, students have time to determine why they got the question wrong, and why the correct answer is correct.

Ending the Game

The game ends when the student finishes all ten levels, when they click end game, or when they've been playing for ten minutes. In other words, the game ends when the student has demonstrated proficiency in reading analog clocks, or when their time is up.

Data Collection

Students cannot start the game until they type in their name. Once a student finishes or ends the game, the application adds their performance data to the collection. For each analog time, the game stores the correct answer, each of the student's guesses, and the amount of time (in seconds) it took to make each guess. At the end of the day, teachers can click the button at the bottom of the webpage and enter a password (letmein) to see a table of results (figure5) or download a comma-separated value (CSV) file of all students' performance for that day (figure 6). They can open this file as a table in any spreadsheet application, such as Microsoft Excel or Google Sheets. They can then graph or chart the student's progress if desired.

Download CSV

Student	Level	Answer	Guess 1	Time 1 (s)	Guess 2	Time 2 (s)	Guess 3	Time 3 (s)
Rachel	1	4:00	4:00	4.5				
Rachel	1	12:00	4:35:	2.5	12:00	2.4		
Rachel	1	12:00	12:00	4.5				
Rachel	1	7:00	7:56	3.1	8:56	2.0	7:00	1.5
Rachel	1	8:00	8:01	4.5	8:02	2.1	8:03	1.9
Rachel	1	7:00	7:00	2.1				
Rachel	2	6:00	Inva	4.7	6:00	2.3		
Rachel	2	12:00						
Lehcar	1	6:00	6:00	2.5				
Lehcar	1	3:00	3:00	1.7				
Lehcar	1	5:00	5:00	1.9				
Lehcar	1	12:00	12:00	1.7				
Lehcar	1	11:00	11:12	3.3	11:00	1.9		
Lehcar	2	2:00	2:00	3.2				
Lehcar	2	4:00						

Figure 5

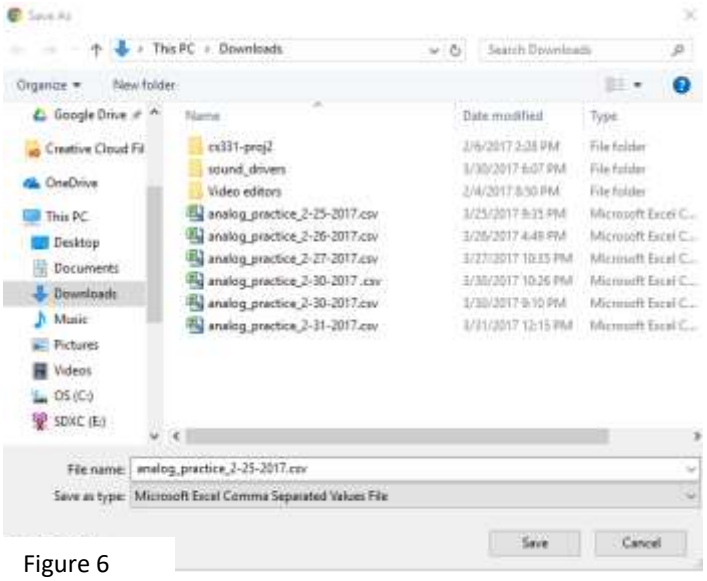

Figure 6

Technical Details

Languages Used

HTML, CSS, and Javascript

Program Overview

The application depends on two objects and a driver, contained respectively in Analog_clock.js, Game.js, and main.js. The clock and game are encapsulated to follow good programming principles.

Analog_clock

This object draws a versatile analog clock using HTML canvas, and thus requires a canvas element. The clock can be drawn with or without hour tick marks, minute tick marks, hour numbering, or minute numbering. In addition, the clock is redrawn as the webpage resizes. It can produce a random time in which the minutes are multiples of zero, thirty, fifteen, five, or one. It stores the time shown as a Javascript Date object, to take advantage of Date functions (toString, getHours(), etc.)

Game

Game's constructor requires the two components essential to the game: an Analog_clock object and the player's name. Only two functions drive gameplay: start_round() and process_answer(); start_round() sets up round of the game, which consists of a question (an analog time to read) and the student's attempts at answering the question (up to three). So, start_round() begins a timer to track how long the student takes to read the clock, randomizes the time, and draws the clock face according to the current Game level. process_answer() determines whether the student is correct, incorrect, incorrect and has no more guesses, ready for the next level, or finished the game. It adjusts the current level and the clock time accordingly.

This object also stores the data about the game: current player, level, Analog_clock, number of correct answers, and an array of Rounds. Round objects represent each of the game's rounds (as defined above) by storing the round's level, correct answer, guesses, and the time it took to make each guess. At the end of the game, the Game formats its Rounds as a HTML table and as a CSV file via export_table() and export_csv ();

main

The main file brings the encapsulated Analog_clock, Game, and the HTML webpage together. It translates clicks, keypresses, and text into triggers and inputs for Game. In reverse, it translates the Game's level, clock, and evaluation of the answer into feedback for the player. In addition, main gets all HTML elements and passes them to Analog_clock and Game.