Report article

An Android implementation of the Hodgkin-Huxley membrane model for student practice

Haruo Toda

Department of Orthoptics and Visual Sciences, Niigata University of Health and Welfare, Niigata, Japan

Keywords: Hodgkin-Huxley model, excitatory membrane simulator, Android, Java language, Open-source
Received: 19 January 2019 / Accepted: 19 March 2019

Abstract

Computational models of physiological phenomena are not only interesting by themselves but are also potentially able to be a partial substitute for student practice with real animals. In this paper, the author introduce a student-oriented simulation system for a popular model of the sodium spike that covers the principles of membrane excitation, essential for an understanding of neuronal function. The approach here employs the popular Android operating system for student practice to build a Java-based simulation of the well-known Hodgkin-Huxley theoretical model that has long provided insights in this field. Despite totally written in Java, the strip-chart display and user-triggered stimulation capabilities have been readily executed even on low-cost tablets and relatively primitive personal computers. By employing user-defined membrane parameters, together with its ability to program numerous types of membrane stimulation, this modeling system may be quite productive in helping students of general physiology and medicine to more quickly and accurately obtain a strong understanding of the sodium spike and its related phenomena.

Introduction

Creating a quantitative model of a biological phenomenon has long been one of the most important and attractive topics in physiology. A classic example is a mathematical model of the excitatory membrane, proposed in the 1950's by Hodgkin and Huxley. They derived the necessary equations after careful consideration of the physical chemistry involved (the chemical kinetics and an equivalent circuit of membranes), and determined the required parameters through repeated experiments using the giant axons of squids. Despite the relatively simple formulae involved, their model explains many important properties of sodium spikes, such as the all-or-nothing response, resting periods, and anodal break potentials [1]. Hence, a strong understanding of this model would facilitate the speed and depth of how well students can grasp the nature of excitatory membranes.

The Hodgkin–Huxley algorithm described in [1] can be efficiently calculated by personal computers (PCs), and the author has implemented in on numerous platforms including MS-DOS, OS/2, and today's major operating systems (OSs) for PCs (Windows, Mac OS X, and Linux). These results were achieved by using Turbo Pascal and its

Corresponding author: Haruo Toda
Department of Orthoptics and Visual Sciences, Niigata University of Health and Welfare, 1398 Shimami-cho, Kita-ku, Niigata 950-3198, Japan
TEL/FAX: +81-25-257-4753, E-mail: toda@nuhw.ac.jp

descendants (Speed Pascal, Virtual Pascal, and Lazarus), and this model readily enables students to practice working with how the algorithm works. However, due to widespread use of Android OS-based mobile devices are widely available today. Being easy-to-use and relatively of low-cost and readily accessible platform, to further facilitate student access to quickly learning the nuances of sodium spikes and membrane excitation. Although Android devices are relatively easy to program, a Hodgkin-Huxley model for this platform surprisingly cannot be found in the most popular search engines (with a keyword "Hodgkin and Huxley" in Google Play, or with a combination of keywords "Android" + "Hodgkin" + "Huxley" + "apk" in Google (http://www.google.co.jp/), where "apk" stands for the application package format of Android OS, searched on July 19, 2018).

In this report, the author explains how an Object Pascal-based membrane simulator was ported over to the Java language for Android OS-based mobile devices, and the implications of this result.

Materials and Methods

1. Numerical Analysis of the Hodgkin-Huxley model

The main element of the Hodgkin-Huxley model is given as their equation 26 in [1].

$$I = C_M\left(\frac{dV}{dt}\right) + \bar{g}_K n^4(V - V_K) + \bar{g}_{Na} m^3 h(V - V_{Na}) + \bar{g}_l(V - V_l) \quad (1)$$

where, V is the membrane potential; I is the total membrane current; C_M is the membrane capacitance; $\bar{g}_K$ and $\bar{g}_{Na}$ are the maximum conductance of the potassium and sodium ions, respectively; $\bar{g}_l$ is the leakage ion conductance; V_K, V_{Na}, and V_l are the equilibrium potentials for potassium, sodium, and leakage ions, respectively (the resting potential is zero, depolarization is denoted as negative); n, m, and h are the 'gate' parameters that determine the open states of the potassium and sodium channels. The dn/dt, dm/dt, and dh/dt are the functions of V (formulae not shown). In this model, if I is zero, V can be solved as a function of time. This assumption is fulfilled when the whole membrane excites at the same time [2]. Because the differentials of n, m, and h are also functions of V, the main equation is a second-order non-linear differential equation of t, which can be only solved by a numerical analysis. In this simulator, Hartree's method was employed to obtain a numerical solution of the model in a manner similar to that used by Hodgkin and Huxley [1]. The time step for the calculation varies between 0.01 and 0.001 ms according to the rates of parameter changes and the membrane potential. The core calculations are encapsulated in an object or a class (membrane object) and its outputs are logged approximately every 0.05 ms (the interval of the data logging depends on the time per width of the plot area in pixels). The user can save the logged data to a text file and recall them on the screen by dragging items to the plot area. To visualize the log data, GNU R (https://www.r-project.org) is used.

2. Multithread programming in Android

Before making the Android version, PC and Macintosh applications were updated using Lazarus (version 1.8.4; http://www.lazarus-ide.org/) to have a strip-chart style main window (a window that automatically scrolls leftward and new plots continuously appear at the rightmost of the window). Lazarus is an open-source rapid application developing environment that is highly compatible with Borland Delphi and its predecessor, Turbo Pascal. As a next step, the Lazarus source codes were ported to Java source codes for Android using Android Studio for Macintosh (versions 3.1.3 and 3.2). The Android application was tested on two tablets (MediaPad T3 7 and MediaPad 7 Youth; HUAWEI, ROC) and one Android stick PC (CX919; Andoer, ROC) running Android 6.0, 4.1.2, and 4.2, respectively. The CPUs utilized here are Cortex-A7, 1.3 GHz quad-core, Cortex-A9, 1.6 GHz dual-core, and Cortex-A9, 1.6

GHz quad-core, respectively. For the Lazarus and Android programming, two Macintosh computers were used (Mac Pro; Apple, CA, USA), both running OS X El Capitan.

Of course the simulator must be able to accept user inputs (*e.g.* pressing a mouse button or touching to the screen) and continuously refresh the screen(s) as the calculation for simulation are performed. In the Lazarus environment, there is a convenient method to achieve this, called "Application.ProcessMessages", which executes the current entries in the message queue attached to the application (Figure 1A). The program can call this method at an approximately regular interval to allow user inputs and screen refreshing (Figure 1B). If the interval is short enough (in this case, after 10 to 100 iterations of the main calculation loop), the user feels as if the application concurrently calculates the membrane model, processes the user inputs, and refreshes the waveforms on the screen, without multithreading. In Android as well, such pseudo-concurrent processing can be achieved by posting a strip of the program codes, called "Runnable", to the message queue (Figure 1C). However, in this environment the performance of calculations was not fast enough and multithread programming was required to achieve an acceptable speed at the expense of difficulties in debugging and possible memory leaks. Fortunately, the Android OS has the AsyncTask class, which is especially designed for such a purpose.

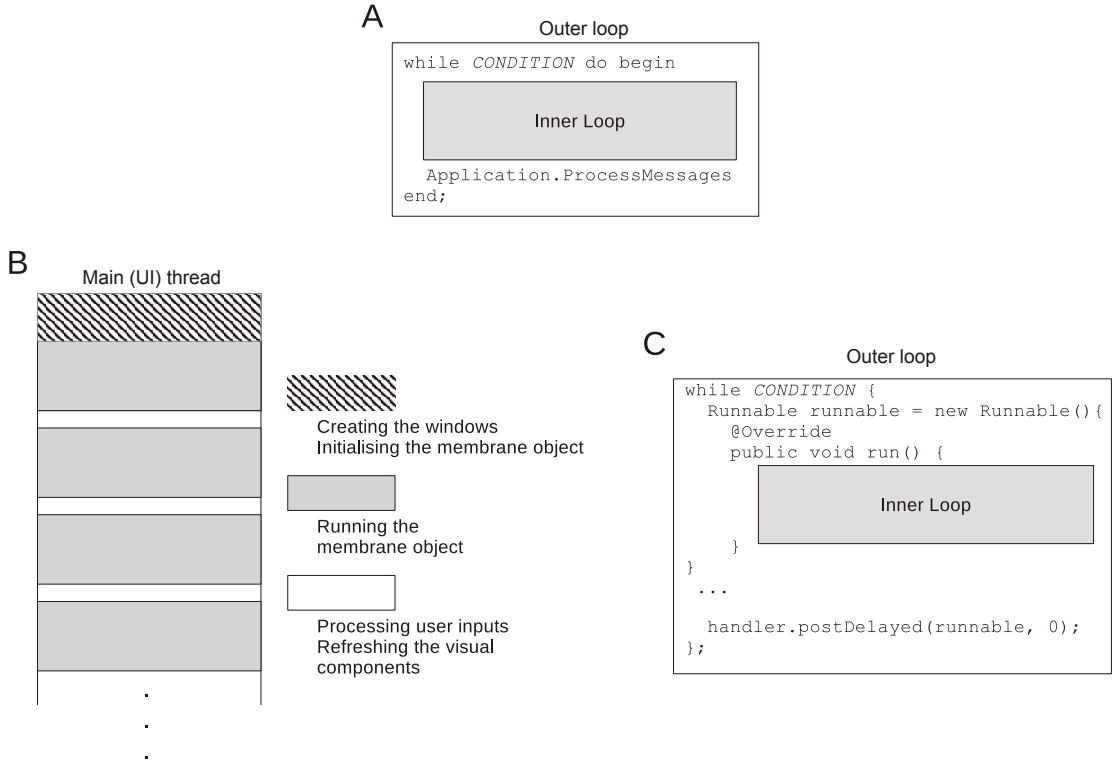

Figure 1. Schematic of (pseudo) concurrent processing in GUI applications.
A: usage of the Application.ProcessMessages method in Lazarus. B: schematic schedule of calculation of the membrane model and user interface processing. C: usage of the Hander. postDelayed method in Android. Calling this method posts the codes in the inner loop to the message queue for further execution.

3. Availability of the source code

The complete set of the source codes for the simulator is freely accessible via Google Drive (https://drive.google.com/file/d/1EGE8z6NOBieR6ho5OhsyuaOalQ-GySlh/view?usp=sharing) under the GPL license.

Results

1. The threshold and the refractory periods

Although the membrane simulator is written entirely in Java, and low-cost tablets or stick-PCs were used to build and test it, sufficient performance was achieved, so that it can serve as an effective learning system. Figure 1 shows the appearance of the simulator's dashboard. The strip-like time chart is displayed at the left side on the panel (where many colored lines are drawn against a black background). When user presses the "START" button, the worker thread is invoked to solve the membrane model repeatedly until the

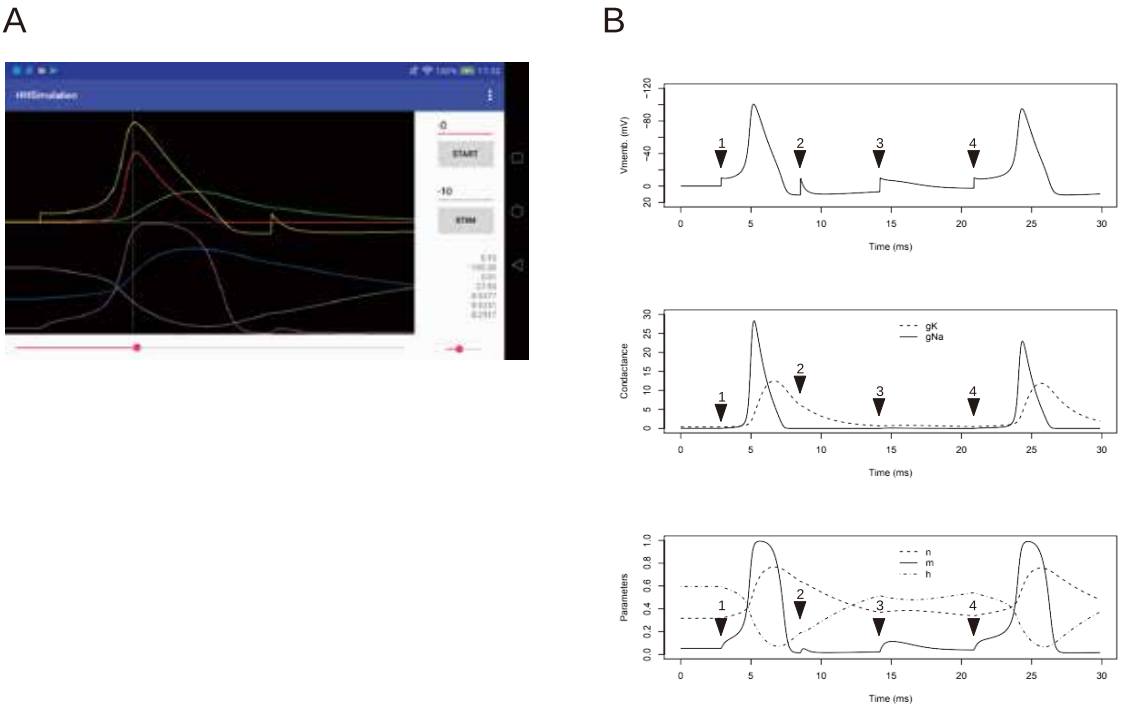

Figure 2. A calculated action potential.

A: Appearance of the membrane simulator on an Android 6.0 tablet. The large black rectangle works as a multi-channel strip-chart. The membrane potential, sodium conductance, and potassium conductance are displayed on the upper half in yellow, red, and green, respectively. On the lower half of the strip-chart, 'n', 'm', and 'h' parameters are displayed in blue, magenta, and grey, respectively. The vertical grey line can be moved by user with a slider to read out the membrane parameters that are displayed below the "STIM" button (time, the membrane potential, sodium conductance, potassium conductance, 'n', 'm', and 'h' parameters, from top to bottom). B: Externally plotted time-courses of (top) the membrane potential, (mid) the sodium and potassium conductance, and (bottom) 'n', 'm', and 'h' parameters. The time-courses are based on the data file derived in the same run as A. Downward arrowheads (numbered 1 to 4) indicate the timings of depolarizing voltage stimulation (10 mV).

"STOP" button (displayed in the same position as the "START" button) is pressed. After the end of the calculations, the user can drag the strip-chart to bring back the logged data which was earlier extinguished from the panel. Moreover, the user is able to move the read-out cursor (the vertical grey line) to inspect the values of each parameter at any desired point (displayed on the right).

The user can also enter an initial membrane potential in the upper right text box. According to Hodgkin and Huxley [1], the resting potential under the standard conditions is zero, and a negative value represents depolarization. Even in two older devices running Android 4, the simulator allowed user-triggered stimulation even while it was updating the screen continuously. Pressing the "STIM" button immediately changes the membrane potential to the user-defined value (in Figure 2A, -10 mV). This function helps students understand principles of threshold and refractory peri-

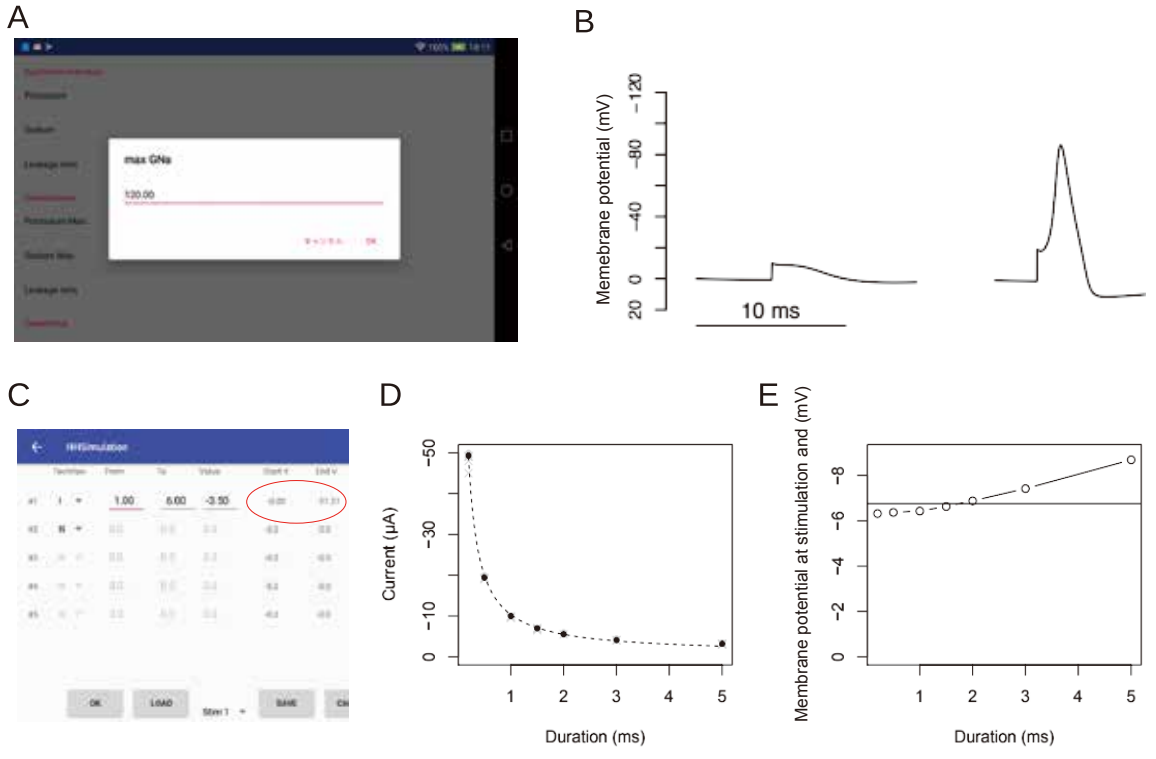

Figure 3. Setting panels of the simulator.
A: Membrane constants setting panel of the simulator on an Android 6.0 tablet. In this example, $\bar{g}_{Na}$ is to be set. B: Examples of simulated responses with a lowered (from 120 to 60 mmho/cm^2) $\bar{g}_{Na}$: (left) local potential in the response to a 10-mV depolarization, (right) reduced action potential following a 20-mV depolarization. C: A part of the stimulation setting panel of the simulator on an Android 6.0 tablet. Each row corresponds to a single stimulus. The input fields are, from left to right, the selector for current or voltage stimulation, the stimulation start time, the stimulation end time, and the stimulation intensity. The following two values (red eclipse) indicate the membrane potentials at the start and the end of stimulation, respectively. D: A time-strength relationship to constant-current stimuli. E: The voltage threshold vs. stimulation duration.

ods. For example, in Figure 2B, we can see that depolarization to -10 mV at first elicited an action potential (arrowhead 1); however, within the hyperpolarization phase, depolarization to the same voltage did not elicit an action potential (arrowheads 2 and 3), thereby simulating the refractory period. At this period, the potassium conductance is high (dashed line on the middle panel) and the 'h' parameter is low (dotted line on the bottom panel), representing decreased membrane resistance and inactivation of the sodium channel, two major causes for threshold increase in the refractory period [3].

2. Setting the membrane parameters and programmed stimulation

The user can modify all the membrane parameters appearing in equation (1): C_M, $\bar{g}_K$, $\bar{g}_{Na}$, $\bar{g}_l$, V_K, V_{Na}, and V_l. For example, the primary effect of the sodium channel blockers can be simulated by lowering $\bar{g}_{Na}$ (Figure 3A and B), while the effect of tetraethylammonium on the voltage-dependent potassium channels can be simulated by lowering g_K.. Likewise, the effects of intra-or extracellular ionic compositions can be simulated by changing the equivalent potential for the corresponding ions (V_K, V_{Na}, and V_l).

The simulator has a stimulation setting panel (Figure 3C), which allows for a wide array of programmed stimulations. The user can set the type (voltage or current), timing, duration, and intensity of the stimulation up to five times. The membrane potentials before and right after each stimulus are displayed on the panel (shown as the red eclipse in Figure 3C). The user can simulate the time-strength relationship in constant current stimulation via this function. For example, in Figure 3D, the minimal current to evoke an action potential is an approximately hyperbolic function of the duration of the stimulation current with a small positive offset corresponding to the rheobase [2]. The 'h' parameter is decreased during a prolonged injection of depolarizing current, which results in higher threshold membrane potentials (Figure 3E). In simulating the anodal break potentials (Figure 4), the 'h' parameter increases (dotted line on the bottom panel) during the hyperpolarizing current injection, and when the stimulation breaks, the 'neuron' fires without any depolarizing stimulation.

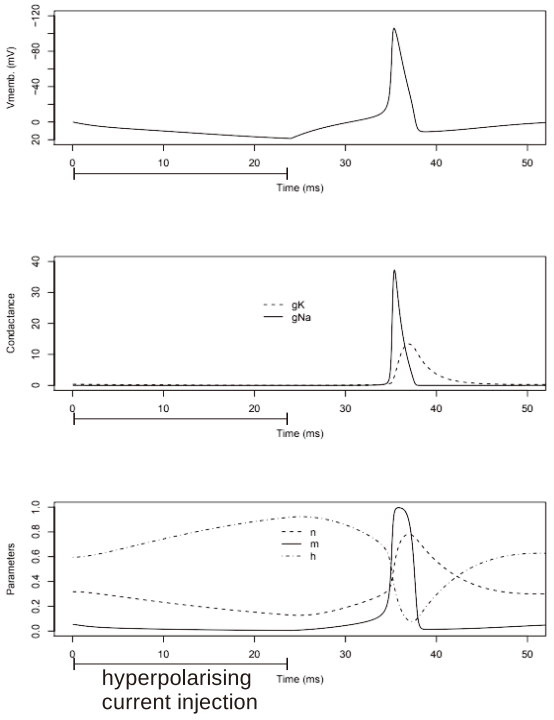

Figure 4. An anodal break potential. Externally plotted time-courses of (top) the membrane potential, (mid) the sodium and potassium conductance, and (bottom) 'n', 'm', and 'h' parameters. The time-courses are based on the data file derived in the same run as A. Downward arrowheads (numbered 1 to 4) indicate the timings of depolarizing voltage stimulation (10 mV).

Discussion

To help extend student access to a productive membrane simulator for the Hodgkin-Huxley model, it earlier PC version was ported from an Object Pascal-based GUI program to a Java-based

Android application. This tool can enable students to simulate the primary properties of a sodium spike (all-or-nothing nature, refractory periods, and anodal break), the effects of the intra- and extracellular ionic concentrations by manipulating equivalent potentials, and also the effects of sodium- and potassium-channel blockers by reducing gNa^+ and gK^+. Though totally written in Java, the strip-chart display and user-triggered stimulations can be concurrently executed even on a dual-core device running Android 4. Such performance may be due to recent improvements in the Java Virtual Machine, the ARM processors, and also to simplicity of the calculations described in the original article [1].

A thorough knowledge of the excitatory membrane's behavior is essential for understanding neuronal functions. However, in vivo (*i.e.*, using real animals) student practice with intracellular recording is difficult outside the large medical school faculties. Because these Android devices are easy to use and available at affordable prices, this Java-based membrane simulator could be readily used by virtually every students as a reasonable substitute for in vivo practice with intracellular recording.

However, the current version of the simulator has some issues that need addressing as follows; 1. Based on the original article [1], depolarization from the normal resting potential is displayed in the negative. The instructor could use this feature to explain that the potential read-out depends on the reference electrode location and selection of the terminal (positive or negative) that is to be connected to the signal electrode, 2. The standalone usage of the simulators is secure, but somewhat degraded. Running the system on HTTP clients of mobile devices (*e.g.*, Safari for iOS, Chrome for Android OS) may be preferable for student practice. Although Java is mainly used to create server-side programs today, it may not be a suitable language to write a Web-based simulator because such a servlet is difficult to maintain for teaching organizations without network specialists. Fortunately, HTML5 newly supports the canvas elements for dynamic two-dimensional graphics and the Web Workers API for multithread execution [3]. These new features can be programmed in JavaScript, one of the most frequently used object oriented programming languages [4]. The author is planning to port the simulator to the HTML5 environment and test it under practical conditions (student practice or demonstration in a class) in the near future.

Conflicts of Interest

There are no conflicts of interest to declare.

References

1. Hodgkin AL, Huxley AF. A quantitative description of membrane current and its application to conduction and excitation in nerve. Journal of Physiology. 1952; 117: 500-544.
2. Brinley Jr. FJ. Excitation and conduction in nerve fibers.In: Mountcastle VB, editor. Medical Physiology. 14th ed. St. Louis: The C.V. Mosby Company; 1980; 46-81.
3. Mozilla Developer Connection. "HTMLCanvasElement". Available from: https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvas Element (accessed November 15, 2018)
4. Mozilla Developer Connection. "Using Web Workers". Available from: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers (accessed November 15, 2018)