Mole Whack

The Concept

Develop a web page based game that can be made into an app for a cell phone. For this exercise the game will be similar to the arcade game Wack-A-Mole which we will call Mole Wack.

The Design

HTML —HyperText Markup Language

Provides the basic structure of the webpage, like paragraphs, titles, headers, buttons, etc. The main elements for this game are:

- a running score
- a mole to be whacked
- a start button
- an end of game message.

CSS—Cascading Style Sheets

Defines the layout and appearance of the webpage. This places elements on the screen positioned relative to other elements or to the screen itself. The appearance of the elements is also controlled here, like the choice of fonts, how big (or small) things are, what color things are, how much white space is around an object, what do element borders look like, etc.

- a running score — blue, fairly big, upper right corner
- a mole to be whacked — brown square for now. This can be replaced with an image.
- a start button — lower right corner. Hide during game.
- an end of game message — center of screen. Hide during game.

Javascript

Defines the logic and function of the webpage.

- Game opens to a fairly blank screen
- Game is started with the Start button. Score is set to zero.
- Game ends after a fixed amount of time, say 10 seconds.
- During play, the mole changes position and stays for a brief period before moving to another location.
- Clicking on a mole increases the score.

Implementation

Use a single page instead of more serious three page design that separates HTML, CSS and Javascript. This is to make it easier to edit (for this exercise) and easier to share.

HTML basics

Basically HTML is a text file. Elements are tagged so that they can take on individual attributes and treatment. A tagged element consists of an open tag and a close tag. Tags are a name surrounded by a left and right angle bracket. Tags usually come in pairs: an open tag to start the element and a close tag to end the element. Close tags use the same name as the open tag except a slash is added before the name. Usually an open tag is followed by a close tag (except for unitary tags like <br/>). Open tags may include attributes that further describe the tagged object, like an identifier (id) or class.

Paragraphs use the <p> tag, so a paragraph could be coded as:

```
<p>
This is the text of a paragraph. This is just some more words to fill out
the paragraph.
</p>
```

White space doesn't mean anything to HTML. White space are basically the following characters: space, tab, carriage return or line breaks. So whether things are all mushed up in a single line or spread out over multiple lines will not change the appearance of the presented text. Desired text formatting must be set up by tagging the it, so it can be treated differently than other text.

HTML has some special tags.

- `<html>….</html>` surrounds all of the text of the HTML document.
- `<head>…</head>` surrounds HTML commands that do not appear on the finished page.
- `<body>…</body>` surrounds the HTML that will form the contents of the finished page.
- `<style> </style>` tag is used to include some CSS code with its own syntax. `/*Beware of CSS comments */`
- `<script> </script>` tag is used to include some Javascript code with its own syntax `// Beware of Javascript comments`
- `<!– comment in HTML –>` comments to explain what and why something is done
- `<!DOCTYPE html>` tells the browser that the document is in "modern" HTML.
- `<meta charset="utf-8">` tells the browser that the characters are encoded as Roman characters. This can go beyond what is normal English with accented characters, foreign characters and special symbols like curly quotation marks.

HTML Skeleton

The following HTML code has the "required" elements of the most basic HTML page. All HTML pages start with the following:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title></title>
</head>
<body>
```

```
</body>
</html>
```

The Basic HTML Code

The following HTML code provides the elements required for the Mole Wack in its most basic form:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Mole Wack</title>
</head>
<body>
  <p>0</p>
  <button>Start</button>
  <button>Mole</button>
  <p>Game Over</p>
</body>
</html>
```

The Enhanced HTML code

This HTML enhances the basic code by:

- naming the display elements so that they can be singled out in the CSS and Javascript code
- adding a format class to the paragraphs.
- adding place holders for the CSS and Javascript code.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Mole Wack</title>
  <style>
    /* CSS code goes here */
  </style>
</head>
<body>
  <p id="score" class="noselect">0</p>
  <button id="start">Start</button>
  <button id="mole">Mole</button <meta charset="utf-8"> >
  <p id="gameOver" class="noselect">Game Over</p>
</body>
<script>
  // Javascript goes here
</script>
</html>
```

Cascading Style Sheets—CSS

These define the appearance of the elements used for the mole whack game.

Basic Syntax

CSS is a declarative language. This means that things are only declared. It does not have conditionals, do loops, or variables. White space doesn't mean anything, so things can be put all into the same line or spread out over many lines. To make it easier to read, CSS is usually split over many lines and white space introduced to show indenting. There are tools to remove white space to make the code faster to load for the final product.

Basic syntax is:

```
Selector { attribute : value; }
```

Where the selector is the name of a tag, like p for the paragraph tag, or body for the body tag. Tag IDs can be selected with `#id`. Tag styles can be called out as `.style.` Selectors can be stacked to be more specific or general, but that is beyond the scope of this exercise.

Each selector may specify a set of CSS attributes, like width, height, font-family, font-size, color, etc. Each attribute is followed by a colon and the value appropriate for that selector. Each attribute is separated by a semi-colon.

If the browser cannot understand they syntax, it ignores it silently. To catch these errors takes careful testing and checking or the use of a "lint" progam that reports all syntax errors.

A lot of the characteristics of elements are inherited. That means that they take on the characteristics of the element that contains them. All elements of a page are contained within the body tag, so the body tag is used to set the general format for all elements on the page, especially the font characteristics.

Initial Cut of CSS

To program the CSS for Mole wack we start out with definitions for the body and the displayed elements of the game, trying to encode the characterists initially stated for those elements.

```
body { /* define global style and background */
  font-family: Helvetica neue, sans-serif;
  background-color:green;
}
#score { /* big, bold, blue, upper right */
  font-weight:bold;
  font-size:400%;
  color:blue;
  position:absolute;
  top:.1em;
  right:1em;
}
#start { /* normal button, lower right */
  position:absolute;
  bottom:1em;
  right:1em;
}
#mole { /* brown square with no positioning */
  height:75px;
```

```
    width:75px;
    position:absolute;
    background-color:brown;
  }
  #gameOver { /* big centered text */
    position:absolute;
    margin:auto;
    width:95%;
    text-align:center;
    top:50%;
    font-size:200%;
  }
```

Enhanced Javascript

The initial cut was pretty good, but it did have some problems. The score has an outline. The characters in the score, the mole has an undesired border and the Game Over label are selectable.

```
  body { /* define global style and background */
    font-family: Helvetica neue, sans-serif;
    background-color:green;
  }

  #score { /* big, bold, blue, upper right */
    font-weight:bold;
    font-size:400%;
    color:blue;
    text-align:right;
    position:absolute;
    top:.1em;
    right:1em;
  }

  #start { /* normal button, lower right */
    font-weight:bold;
    font-size:100%;
    position:absolute;
    bottom:1em;
    right:1em;
  }

  #mole { /* brown square with no positioning */
    font-weight:bold;
    font-size:150%;
    height:75px;
    width:75px;
    position:absolute;
    background-color:brown;
    border:none; /* get rid of border */
    display:none; /* don't display at first */
  }

  #gameOver { /* big centered text */
    position:absolute;
```

```
      margin:auto;
      width:100%;
      text-align:center;
      top:50%;
      font-size:200%;
      display:none; /* don't display at first */
    }

    .noselect { /* make it so text elements can't be selected */
      -webkit-touch-callout: none; /* iOS Safari */
      -webkit-user-select: none;   /* Chrome/Safari/Opera */
      -khtml-user-select: none;    /* Konqueror */
      -moz-user-select: none;      /* Firefox */
      -ms-user-select: none;       /* IE/Edge */
      user-select: none;           /* non-prefixed version, currently
                                      not supported by any browser */
    }
```

Javascript

Overall code design

- Initialize variables and wait for the start button
  - show start button
  - hide mole and Game over
- Mole
  - when clicked, increment score
- Start button
  - starts game loop
  - show mole
  - hide start button and game over
- Game loop
  - Provide timing for moving mole and determine end of game
  - at end of game:
    - show Game over and start button
    - hide mole
  - Move mole

Javascript Language Characteristics

Javascript is a procedural language that allows a browser to detect user actions, request more information from a server, modify the appearance of information on the window. It allows for conditional execution. It is an "untyped" language. This means that a variable can be a number in one instant and changed to a string in another. This makes the language very easy to learn and write, but it can make it difficult to debug as the program gets complex.

It has some object-oriented characteristics that:

- define data structures that are accessed hierarchically with dot notation. E.g., person.hairColor is the hairColor attribute of the person object.
- define methods that act on those structures. These are also specified using the dot notation, e.g., person.Address() could be a function that returns the address of a person object. Methods are specialized functions that only work with a specified object.

- All tagged elements in HTML are objects that reside in the Document Object Model—DOM for short. This model allows access to characteristics of individual attributes, e.g., document.body.style.font-size can be used to access the font size of the <body> element that would apply to all inherited objects on the page.

Javascript Mini Reference

Javascript is a language that has a lot of capabilities. These capabilities can be found in Javascript reference manuals or through on-line searches. The following is a mini reference that can be used for the understanding of the Mole Wack web page

- `// blah blah` to end of line. a comment
- `/* blah blah */.` a comment which may be within a line or use multiple lines.
- `;` lines may end in a semi-colon. Some programmers prefer no semi-colons.
- `{ blah blah }` a block of code that is executed as a unit, for example a then clause, an else clause of the definition of a function.
- `function foo (params) { blah blah }` defines a function (subroutine) *foo* which has zero or more parameters *params* and instrutions *blah blah*.
- `var symbol;` define *symbol* as a variable. Where it is placed determines its scope or area of influence. Inside a function, it may only be used locally within the function, a so-called local variable. Placed at the outer level, it may be accessed by all functions, a so-called global variable.
- `a + b` if a and b are strings, say a="a" and b="b", the strings are concatenated together with the result "ab".
- `a + b` is a and b are integers, say a=1 and b=2, the variables are added together with the result 3.
- `document.getElementById( "identity")` find the element with Id *"identity"* in the document or DOM. This returns the entire object. This object can be saved to a variable to prevent multiple look ups. The following examples are equivalent:
- `var foo = document.getElementById( "identity")foo.attribute = value`
- `document.getElementById( "identity").attribute = value`

  where *attribute* and *value* are arbitrary
- `element.onclick() = foo;` set up the listener for click events on that element to use the function "foo".
- `element.innerHTML` access the HTML text that is between the begin and end tags of the element. This can be used to read (e.g. `foo = document.getElementById( "score").innerHTML` or to write the inner HTML `(document.getElementById( "score").innerHTML = foo.`
- `element.style.display=string` set the display attribute of the style for the element in the DOM named "identity" to a string, where *string* may be:
- `"none"` for no display
- `"block"` for display with line break
- `"inline"` for display inline without line breaks

- `element.style.top` is used to access the top attribute of the style of an element. This can be used to position an element with absolute positioning relative to the browser window. e.g.,
- `element.style.top ="0px";` says to align the top of the element with the top of the window.
- `element.style.top ="5px";` says to align the top of the element 5 pixels below the top of the window.
- `element.style.bottom` is used likewise to position an element relative to the window bottom.
- `element.style.left` is used likewise to position an element relative to the window left side.
- `element.style.right` is used likewise to position an element relative to the window right side.
- `window.innerWidth` returns the number of pixels that measure the width of the browser window.
- `window.innerHeight` returns the number of pixels that measure the height of the browser window.
- `parseInt(foo)` converts a string foo into its integer value. If foo="12", the result is 12. if foo="12.3", the result is 12.
- `Date.now()` returns the number of milliseconds between 1 January 1970 00:00:00 UTC and the current time (now). Date is a general time and date service with many methods for retrieving the date, day, year, month, hour, minute, second and millisecond.
- `Math.random()` return a random floating point number between 0 and 1. Typically the result is multiplied by the range of desired values; so the multiplied result is between 0 and the maximum range.

Stepwise Javascript Implementation

When an programmer has a problem that is too big to understand all at once, the problem is broken down into steps that are easier to understand and implement. Each successive step builds on the prior steps.

Hitting Mole Increments Score

Having the score increment when the mole is hit is fundamental to this game. This snippet shows the definition of the score variable with `var score`, its initialization to the value 0 with `var score = 0`, and it being incremented by the `molePressed()` function. To get the `molePressed()` function to be called, it is attached the `onclick` event listener of the document DOM element named "mole" with the statement: `document.getElementById( "mole").onclick = molePressed;.` The source is found in moleLesson5.html.

```
//*** GLOBALS ***
var score = 0;


//*** INITIALIZATION ***
```

```
    document.getElementById( "mole").onclick = molePressed;
    document.getElementById("gameOver").style.display="none"

    //*** FUNCTIONS ***

    function molePressed () {
      score = score + 1;
      document.getElementById( "score").innerHTML = score;
    }
```

Start Button Resets Score

This snippet adds functionality to the start button to reset the score to 0. It is similar to the preceding example, except that the start button element is being effected.

```
    //*** GLOBALS ***
    var score = 0;


    //*** INITIALIZATION ***

    document.getElementById( "start").onclick = startPressed; //**NEW
    document.getElementById( "mole").onclick = molePressed;
    document.getElementById("gameOver").style.display="none"

    //*** FUNCTIONS ***

    function startPressed () {                              //**NEW
      score = 0;                                            //**NEW
      document.getElementById( "score").innerHTML = score;  //**NEW
    }                                                       //**NEW

    function molePressed () {
      score = score + 1;
      document.getElementById( "score").innerHTML = score;
    }
```

Isoloate Access to the Score Element

Sometimes it is useful to isolate access to a particular data item or element. Here the function `setScore()` is used to limit access to the score element and the score variable. The resulting code is also somewhat cleaner. The source file is moleLesson6.html.

```
    //*** GLOBALS ***
    var score = 0;


    //*** INITIALIZATION ***

    document.getElementById( "start").onclick = startPressed;
    document.getElementById( "mole").onclick = molePressed;
    document.getElementById("gameOver").style.display="none"
```

```
//*** FUNCTIONS ***

function setScore (count) {                                 //**NEW
  score = count;                                            //**NEW
  document.getElementById( "score").innerHTML = score;    //**NEW
}

function startPressed () {
  setScore (0);                                             //**NEW
}

function molePressed () {
  setScore (score + 1); );                                  //**NEW
}
```

The Mole Moves

Critical to the Mole Wack game is the movement of the mole itself. Remember that each element of the web page is accessible using the DOM ant that includes the style attributes of the element. We saw how CSS can be used to position an element like the score or the start button on the display. This snippet shows how Javascript can change the style positioning attributes to dynamically move the mole element on the screen.

Look at the moveMole () function. The `width` and `height` variables are set to the width and height of the `window` object which is the display area of the browser. These have been decreased by 75 which is the width of the mole element. This sets up the bounds for moving the mole. The mole variable is set to the object which is the element on the document named "mole"). The style top attribute of this object is then set to a string which is the number of pixels from the top of the window. Wait a second, break this down a little bit. The function `Math.random()` returns a floating point number between 0 and 1. This is multiplied by the variable `height`, which is the window height in pixels. This result is a number in pixels between 0 and the window height. To convert that result into a string it is concatenated, usingthe "+" operator, to the string "px". The "px" string is required by the top attribute to let it know what measurement units to apply. A similar thing is done with the width positioning for the mole. This listing is moleLesson7.html.

```
//***GLOBALS ***

var score = 0;


//*** INITIALIZATION ***

document.getElementById( "start").onclick = startPressed;
document.getElementById( "mole").onclick = molePressed;


//*** FUNCTIONS ***

function setScore (count) {
  score = count;
  document.getElementById( "score").innerHTML = score;
}

function startPressed () {
  setScore (0);
}

function moveMole () {                                              //**NEW
  var width = window.innerWidth - 75; // decrease by mole size      //**NEW
  var height = window.innerHeight - 75; // decrease by mole size    //**NEW
  var mole = document.getElementById("mole");                       //**NEW
  mole.style.top = height * Math.random() + "px";                   //**NEW
  mole.style.left = width * Math.random() + "px";                   //**NEW
  moleTime = Date.now() + moleDuration;                             //**NEW
};                                                                  //**NEW

function molePressed () {
  setScore (score + 1);
  moveMole();                                                       //**NEW
}
```

The Mole Moves By Itself

In the previous exercise the mole is moved whenever the mole is clicked. But that isn't too challenging. You have all the time in the world to move to the mole and then click on it. The mole needs to be moved all of the time. To do this we introduce a timing loop function `gameLoop()`. This is started by the `startPressed()` function when it calls it. The loop then calls itself with the `setTimeout()` function. This built-in function is special in that the call is delayed by the number of milliseconds specified in the second parameter. This allows the browser to do other things like detecting clicks while the loop waits. The loop checks to see if it is due to move the mole. When it is, the loop calls the `moveMole()` function. Whether it is due or not is checked by looking at the system time using the `now()` method of the built-in `Date` object which returns the number of milliseconds since the beginning of time, January 1, 1970. The due time for the mole movement is set in the startPressed() function to be immediate (the current time) and after a delay in the moveMole function. Note that the loop runs at a different pace than the mole. This is a general mechanism that allows the timing loop to time many things without having its period set to the periods of the things that it wants to time. In general the timing for the loop does not have to be faster than the eye can perceive changes (say around 30 times a second or 33 milliseconds). This listing is in moleListing8.html.

```
      //***GLOBALS ***

      var moleTime;                                       //**NEW
      var moleDuration = 0800; // .8 second               //**NEW


      //*** INITIALIZATION ***

      document.getElementById( "start").onclick = startPressed;
      document.getElementById( "mole").onclick = molePressed;
      document.getElementById( "gameOver").style.display = "none"

      //*** FUNCTIONS ***

      function startPressed () {
        setScore( 0);
        moleTime = Date.now();                            //**NEW
        gameLoop();                                       //**NEW
```

```
    }

    function gameLoop () {                              //**NEW
      if (Date.now() >= moleTime) {                     //**NEW
        moveMole();                                     //**NEW
      }                                                 //**NEW
      setTimeout(gameLoop, 300);                        //**NEW
    }                                                   //**NEW

    function moveMole ()
      var width = window.innerWidth
      var height = window.innerHeight;
      var mole = document.getElementById("mole");
      mole.style.top = height * Math.random() + "px";
      mole.style.left = width * Math.random() + "px";
      moleTime = Date.now()   + moleDuration;
    }

    function molePressed () {
      setScore( score + 1);
      moveMole();
    }
```

The Game Ends

Now the game is getting a bit more interesting, but it can go on forever. Your maximum score is determined by how long you want to sit around playing the game. On way to fix this is to have a fixed amount of time for playing the game. When the start button is pressed, the gametime variable is set to the time when the game should end. A check for this is added to the gameLoop() function to either continue with the game or to end it. Now it is also appropriate to make different elements appear an disappear depending on the game state: idle, playing, end. This listing is in moleListing9.html.

```
    //***GLOBALS ***

    var gameTime;                                                 //**NEW
    var moleTime;
    var gameDuration = 10000; // 10 seconds                       //**NEW
    var moleDuration = 0800; // .5 second


    //*** INITIALIZATION ***

    document.getElementById( "start").onclick = startPressed;
    document.getElementById( "mole").onclick = molePressed;
    document.getElementById("gameOver").style.display="none"


    //*** FUNCTIONS ***

    function startPressed () {
      document.getElementById( "score").innerHTML = 0;

      var baseTime = Date.now();                                  //**NEW
      gameTime = baseTime + gameDuration;                         //**NEW
```

```
        moleTime = baseTime;                                    //**NEW
        document.getElementById("gameOver").style.display="none"; //**NEW
        document.getElementById("start").style.display="none";    //**NEW
        document.getElementById("mole").style.display="block";    //**NEW
        gameLoop();
      }

      function gameLoop () {
        var currentTime = Date.now();                           //**NEW
        if (currentTime >= moleTime) {
          moveMole();
        }
        if (currentTime < gameTime) {;                          //**NEW
          setTimeout(gameLoop, 300); ;                          //**NEW
        } else {;                                               //**NEW
          document.getElementById("gameOver").style.display="block";//**NEW
          document.getElementById("start").style.display="block"; //**NEW
          document.getElementById("mole").style.display="none";   //**NEW
        }
      }

      function moveMole () {
        var currentTime = Date.now()
        var width = window.innerWidth - 75; // decrease by mole size
        var height = window.innerHeight - 75; // decrease by mole size
        var mole = document.getElementById("mole");
        mole.style.top = height * Math.random() + "px";
        mole.style.left = width * Math.random() + "px";
        moleTime = currentTime + moleDuration;
      }

      function molePressed () {
        var score = document.getElementById( "score");
        score.innerHTML = parseInt(score.innerHTML) + 1;
        moveMole();
      }

```

What is the Next Step???

- Add an image for the mole
- Add sounds
  - One for hitting the mole
  - One for missing the mole
  - One when the mole moves
- Add a high score
- Make into an app.
  - find a wrapper like PhoneGap for smart phones or investigate Node.js for other platforms like Linux, MacOS or Windows.
  - wrap code in the wrapper
  - download wrapped code into phone
  - install the code
- Make the page web accessible
  - add to a server, perhaps on a Raspberry Pi with Apache.
- Learn more about Javascript, HTML, and CSS using resources:
  - Read a book from it-ebooks.info,
  - Take a course from Khan Academy
  - Get hands on experience with Code.org
  - Find a particular feature at W3School
- Learn about code development tools
  - Browser based debugging tools
  - "lint" programs to check CSS and HTML syntax
  - "minify" programs to make your final code smaller

Possible Carreers in Information Technology

- help desk / computer support
- system administrator
- system analyst
- coder
- web developer
- front-end developer (HTML, CSS, Javascript and many more)
- back end developer (PHP and many more)
- web designer (heavy CSS with HTML and Javascript)
- product developer/engineer
- software engineer
- system engineer
- network engineer
- protocol engineer
- engineering management
- chief information officer