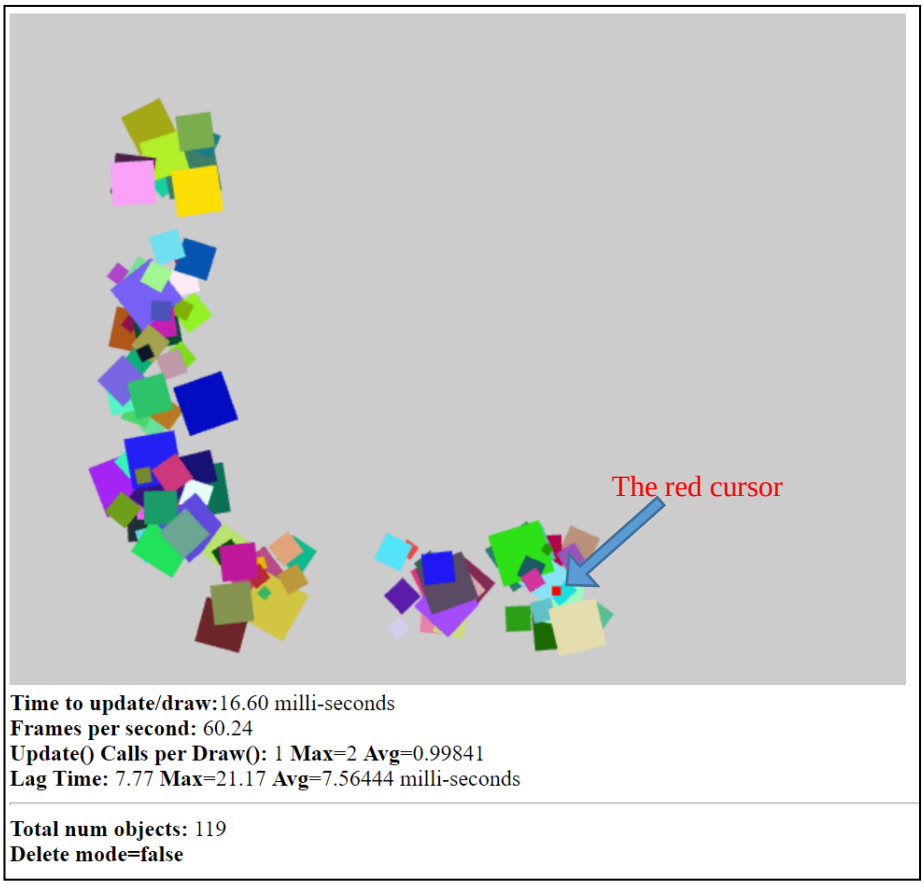
CSS 452: Programming Assignment #2
Game loop and Keeping track of time

Due time: Please refer to our course web-site

Objective
In this programming assignment we will develop a little depth of understanding of the provided API while involving the user and working with the inner-most component: the game loop, or, working with time.

In addition to solving the specified challenges, make sure you have an absolutely and thorough understanding of every single file in your solution. Remember, complexity will increase very rapidly.

Assignment Specification:
Here is an example of the results from this assignment:



Notice:

- **Text output**: There are text output under the drawing. These are output from the HTML index file. More on this later.
- **Red Cursor**: There is a red square above the mouse cursor, this is our red cursor.
- **Cluster of shapes:** There are cluster of shapes, these are created by the space bar.

Here is the specification for the assignment.

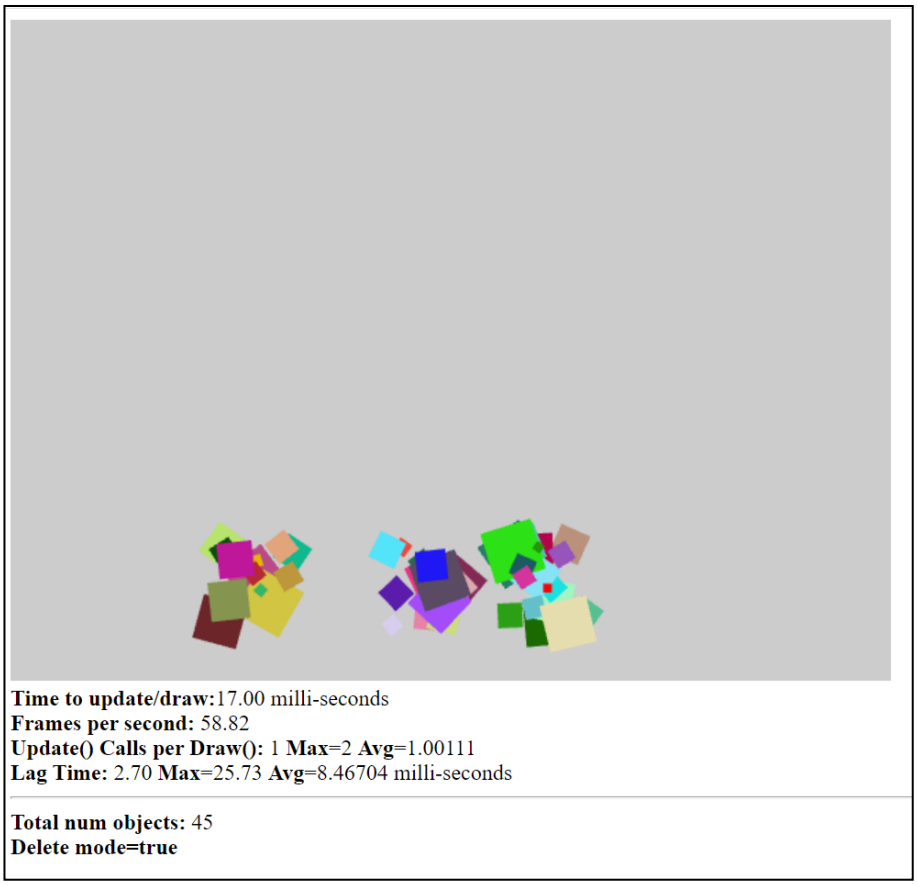
- You will begin with the API from Example 4.2 (4.2.keyboard_support).
- You will create a new my_game.js test case and modify (very slightly) the API to support/export new functionality.
- MyGame test case:
    - The world: canvas: 640x480, world: 100x75
    - Your app will draw the ***Red Cursor*** initially at the center of the viewport.
        - Red Cursor: size 1x1
    - **Arrow keys**: (Left/Right/Up/Down) moves the ***Red Cursor*** accordingly.
    - **Space bar**: creates n squares where:
        - n is a random number between 10 to 20
        - The center of the squares are randomly distributed within 5 units of the ***Red Cursor***.
        - The size of each square is a random number between 1 to 6.
        - The squares are randomly rotated.
    - **D-key:** the clicking of the D-key will send your app into **delete mode**. Your app will remind in delete mode until all the created squares are deleted.
        - The delete mode: will cause the squares to be deleted in the order and according to the time-interval of their creation. E.g.,
            - Set-1: at time t0, you pressed the first space-bar and squares Set-1 is created
            - Set-2: 3-seconds after t0, you press the space bar again and square Set-2 is created
            - Set-3:10-seconds after t0, you press the space bar again and square Set-3 is created
        - Now, the deletion order will follow the creation order, that is,
            - Delete Set-1, when D-key is clicked
            - 3-seconds after t1, you will delete Set-2
            - 10-seconds after t1, you will delete Set-3
        - When all squares are deleted, your application will exit from delete mode. At this point, the time interval reference is reset, that is, the subsequent creation will be considered as creating at time 0.
        - **Note:** while in delete mode, your system must continue to support the space bar creation functionality. That is, while deleting, new squares can still be added, and, the deletion sequence and time duration as described above must be maintained.
- API modification (very slightly):
    - **engine/core/loop.js**: you will examine the core of Game Loop and decide what are the important information to pass out so that you can report on
        - Time to call update/draw
        - Number of frames drawn in a second
        - Number of update() calls per each draw() function call. Also, print out the max, and average over the entire session
        - Lag Time: current lag time, the max, and average over the entire session.
            - The definition of lag time is elapsed time since last update that exceeded the desired frame time. Or, in the ***loopOnce()*** function, before the while-loop calling to update():

```
mLagTime – kMPF
// ideally, this should be close to zero
```

    - **renderer.js**: this is where I modify the API to include a relative object creation time, this relative time is used to support time-ordered object deletion.

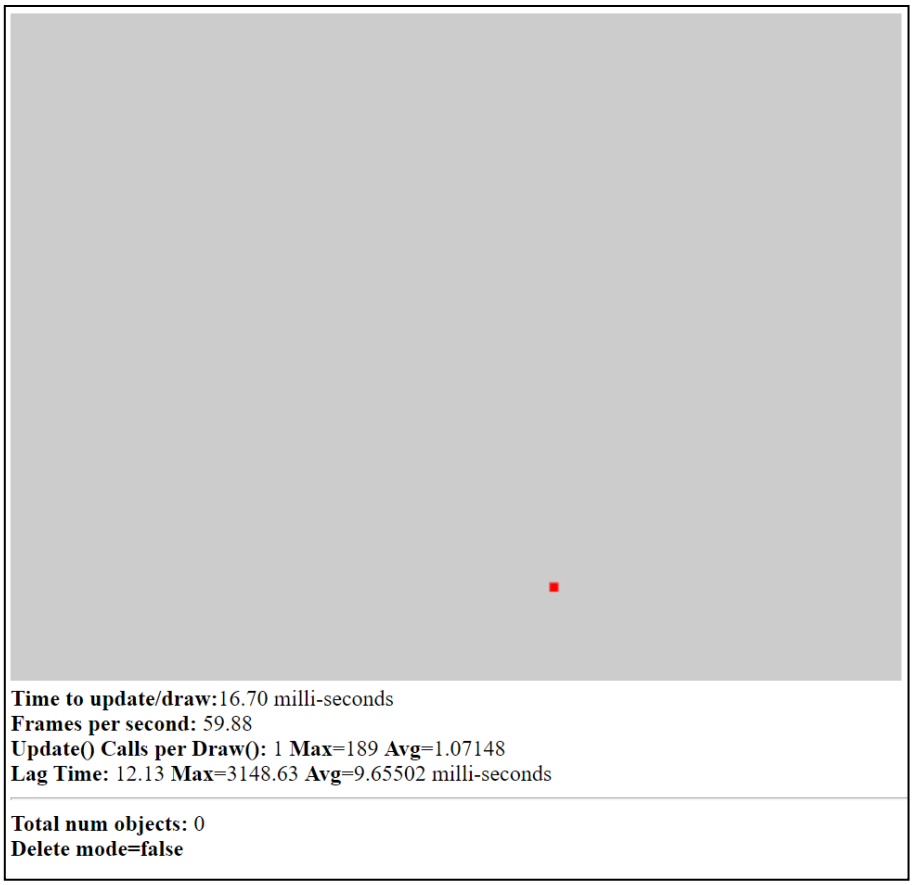
- HTML page text output
    - You will modify index.html to support text output of information concerning your test case (e.g., number of squares on screen) and the engine state (re-draw frame rate). More about this later.

The above given screenshot shows pressing the space-bar will create a set of squares (a number between 10 to 20) around the ***Red Cursor*** position. The following screenshot shows that after the D-key, square sets disappear in the same order and time interval as they were created. Notice the **Total num objects** output and the **Delete mode** status.



Loop based on actual wall clock time. Since our game loop is defined against real-world time, you can examine loop behavior by exaggerate elapse time. Try this, iconize your web-browser and then maximize it again. When iconized, the *requestAimationFrame()* function will not be called and there are no updates, however, during this time, wall-clock continues to tick. That's why when you maximize your browser again, you will notice one *huge* lag time and large number of update() calls. This observation tells us, if I iconize a browser window when an object begins to drop in my game, the object will continue to drop after I iconize the window, and, when I maximize my window again, the object will likely have come to a stop on the floor.

Try opening this example (Example 9.8) in your browser and immediately iconize the browser. Now, maximize the browser to observe all objects are resting on various platforms. This behavior may or may not be desirable. In all cases, we need to recognize this is the behavior we have defined. In the following screen shot, I iconized/maximized my browser, notice the max number of update calls (189), and lag time (3148 mSec).



Hints:

1. Printing Javascript variables from HTML. The following HTML-code fragment is in the <body> tag in my index.html file. Notice that the code fragment defines two global functions: **gUpdateFrame**, and **gUpdateObject**. These two functions can be called from anywhere in our JavaScript. You can decide if you want to modify these functions and where you want to call these two functions from. The following HTML code is here in plain text.

```
<body onload="new MyGame('GLCanvas');">

        <canvas id="GLCanvas" width="640" height="480">
            <!-- GLCanvas is the area we will draw in: a 640x480 area. -->
            Your browser does not support the HTML5 canvas.
            <!-- this message will show only if WebGL clearing failed -->
        </canvas>

        <br>
        <div id='UpdateFrame'></div>
        <hr>
        <div id='UpdateObject'></div>

        <script>
            let mMaxUpdatePerDraw = 0;  // These are global variables: initialized once per page
load
            let mAvgUpdate = 0;
            let mMAxLagTime = 0;
            let mAvgLag = 0;
            function gUpdateFrame(elapsed, numUpdatePerDraw, lagTime) {

                … you need to include logic here to keep track of max/avg update/lagtime

                let elm = document.getElementById("UpdateFrame");
                elm.innerHTML =
```

```
                            "<b>Time to update/draw:</b>" + elapsed + " milli-seconds<br>" +
                            "<b>Frames per second: </b>" + (1000/elapsed).toFixed(2) + "<br>" +
                            "<b>Update() Calls per Draw(): </b>" + numUpdatePerDraw +
                                " <b>Max</b>=" + mMaxUpdatePerDraw +
                                " <b>Avg</b>=" + mAvgUpdate.toFixed(5) + "<br>" +
                            "<b>Lag Time: </b>" + lagTime.toFixed(2) +
                                " <b>Max</b>=" + mMaxLagTime.toFixed(2) +
                                " <b>Avg</b>=" + mAvgLag.toFixed(5) + " milli-
seconds<br>";            }

            function gUpdateObject (n, deleteMode) {
                let elm = document.getElementById("UpdateObject");
                elm.innerHTML =
                        "<b>Total num objects: </b>" + n + "<br>" +
                        "<b>Delete mode=" + deleteMode;

            }
        </script>
    </body>
```

a. You can find out more about getElementById from here: http://www.w3schools.com/jsref/met_document_getelementbyid.asp
b. You can find out more about innerHTML from here: http://www.w3schools.com/jsref/prop_html_innerhtml.asp
c. My approach to learning about these stuff: google, and then look for: W3-School, and/or StackOverflow.com

2. You will need to define an array to keep track of all the created squares.
    a. Do a google on: "*javascript array*". Look at: http://www.w3schools.com/js/js_arrays.asp
        i. You can initialize an array by:
```
var myArray = [];  // notice the empty "[]"
```
    b. To add to the array:
```
myArray.push(newObj); // add to the end of the array
```
    c. When removing elements from an array
        i. I iterate from the back to the front
```
for (i=myArray.length-1; i>=0; i--)
```
        ii. To delete, I call:
```
myArray.splice(i, 1); // remove from position i, one element
```

3. You will need to call Javascript: `Math.random()` to work with random numbers. Remember:
    a. This function returns a number between 0 to 1
    b. To create a random number between 0.5A and -0.5A, you can do this:
```
A * (Math.random() – 0.5)
```
    c. To create a random number between X and X+Y, you can do this:
```
X + (Y * Math.random())
```

4. Make sure you understand the difference between `engine.input.isKeyPressed()` and `engine.input.isKeyClicked()`. In this MP, we mostly (or only?) work with clicks.

5. This is how I recommend to begin attacking MP1:
    a. Step 1: Copy Example 4.2 to your local folder, and make sure you can compile/run the project
    b. Step 2: Edit my_game.js to remove everything except the Red Cursor, where you can use the arrow keys to move the Red Cursor.
    c. Step 3: Space bar to create one square at the Red Cursor location, try moving the Red Cursor while typing space bar to create many squares
    d. Step 4: Add in randomness (random number of squares, random size/rotations)

e. Step 5: Implement delete mode

Make sure you can run/test your system after each of the above steps.

6. Remember to submit the ***minimum*** number of files that will allow the grade and I to open and run your project. Please refer to the projects you are provided with, notice you can run each project individually (without having to add or remove any files). Pay special attention to the fact that unused files are always removed. You are expected to be professional when submitting your work as programming assignments. Last warning, bad submission will result in loss in credits, **submission that does not run will result in zero credit**.

Credit Distribution

Here is how the credits are distributed in this assignment:

1.	Initial canvas and viewport of 640x480 with red cursor		10%
	a. Correct Canvas size b. Red Cursor initially in the middle of canvas c. Arrow keys move the red cursor	10% 5% 10%	
2.	***Space bar*** creation of squares		30%
	a. Proper drawing of created squares b. Create between 10 to 20 each time c. Squares are within 5 unites from Red Cursor d. Squares are size between 1 to 6 e. Squares are rotated	25% 10% 20% 10% 5%	
3.	***Delete mode*** support		30%
	a. Deletion begins immediately after D-key b. Delete sequence according to creation c. Delete time-interval according to creation d. Exit delete mode when all deleted e. Support for creation in Delete Mode	15% 10% 10% 10% 10%	
4.	Text display of system info		25%
	a. Reporting of time to update/draw b. Reporting of num/max/avg update calls per draw c. Reporting of lag time: current/max/avg d. Reporting of total num objects e. Reporting of deletion mode state	10% 15% 15% 20% 10%	
5.	Proper submission		5%
	a. Zip file names with **NO SPACES** b. No extra unused files/folders (E.g., Test folder) c. Proper coding style, including project names, web-page title (how to change this?) etc.	5% 5% 5%	

This programming assignment will count 11% towards your final grade for this class.

Warnings:

1. Start **today:** You should start working on this assignment ***today!***
    a. Test out on-line submission.
    b. Set up your development environment
    c. Download and make sure you can run the book examples

2. **Understand your work and the system!** DO NOT stop working after you have proper results. Make sure to go through every single file in your project carefully, make sure you understand what the roles of each file are, and what every function is doing. This system will evolve very quickly with complexity increasing very rapidly!

3. **Pace of this class:** The pace of this class will be *fast*, and there will be a *large*, no I mean ***gargantuan***, amount of programming (at least that is what I plan to do). If you fall behind by this weekend, it will be difficult to catch up! The next assignment will be quite a bit more complicated than this one, and it requires you to completely understand this assignment and the system you are working with.

Creativity and Extra Credits: Please be as creative as you can. Make sure your solution is different from mine. If you find this assignment underwhelming, good. How about work on:

- Key to enable all created objects in random movements (but bounded by the canvas)?
- Key to enable automatic creation of squares at fixed interval along where the red cursor is located?
- Key to enable creation of something interesting looking? ☺, etc. etc.
- Integrate results from your MP1 (to support additional shapes to rectangles)

Make sure you work on extra ***AFTER*** you are done with the basic assignment, you can work on anything based on the game engine (talk to me first), and I will find ways of giving you extra credits. Extra credits will help people who get them and NOT hurt anyone else's grade.