# Designing and Developing a simulation of 2-Dimensional Motion using HTML5 Canvas, CSS and JavaScript

Georgia State University | PERIMETER COLLEGE

*Isaiah Philip, Dr. Taha Mzoughi[1].*
*[1]Georgia State University Perimeter College*

## Abstract

This presentation describes the method for writing a web application to simulate 2-Dimensional Motion. The web application was based on HTML5, CSS, and JavaScript.

## Introduction

The HTML5 Canvas element was used with JavaScript to create the animation. As this was my first time working in front end development, I learned the basics of HTML5, JavaScript, and CSS on CodeAcademy. To learn how to use the Canvas element I used MDN Web Docs and when I had more specific questions, I used W3Schools. I frequently referenced Dr. Mzoughi's "Alo Academy" for formatting of Canvas elements, required mathematics, and unique aspects that would be implemented such as ghosting. The purpose of this program is to act as an educational aid for students who want to get a better understanding of the basics of 2D Motion. The purpose of this presentation is to explain what the program does and how it was written.

## Methodology

The visible elements of the program can be categorized into three main groups: the **Canvases**, the **Sliders** and the **Buttons**.

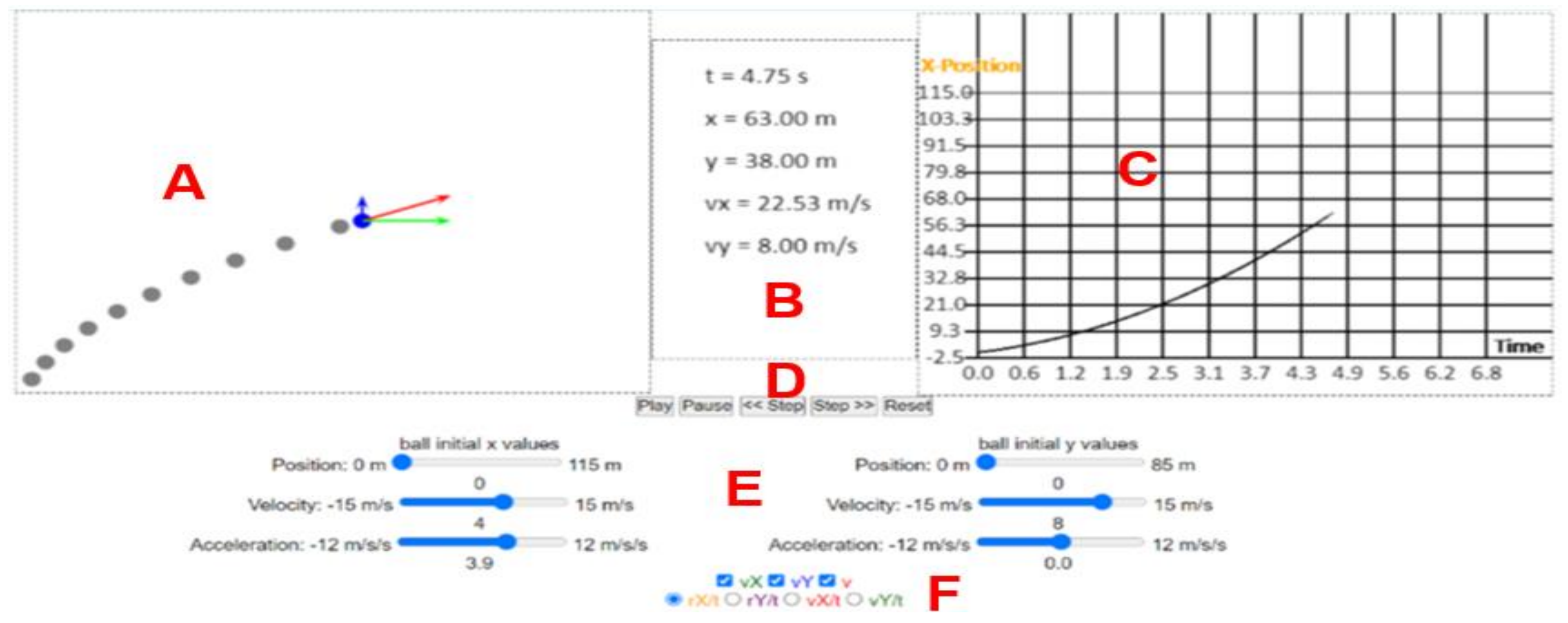


Figure 1: This a snapshot of the simulation. The elements of the program can be categorized into three main groups: the Canvases as shown in parts A, B, C of Figure 1, the Sliders as shown in part E and the Buttons as shown in parts D and F.

There are four **canvases**. Starting from the left, section A, is the animation canvas. As expected, this is the canvas that displays the actual animation. The second is section B, the display canvas. In descending order, the canvas displays the time, x-position, y-position, x-velocity, and y-velocity. The third and fourth canvases are layered on top of each other to create section C, the graph and the corresponding line. The graph canvas displays a graph with time as the x-axis, and either , x-position, y-position, x-velocity, or y-velocity as the y-axis.

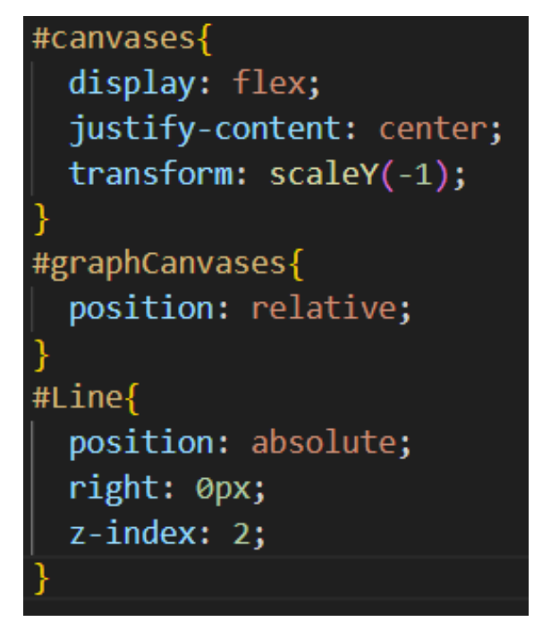
The **buttons** can be further separated into the basic buttons and the specialized buttons. The basic buttons, section D, control basic functions integral to the animation. Specialized buttons, section F, can be selected based on what the user wants to see. There are two rows of specialized buttons, in descending order, the first is the arrow row and the second is the graph row. The arrow row allows the user to select which velocities they would like to see a visual representation of (in section A), in the form of arrows. The graph row allows the user to select which graph they would like to be displayed on the graph canvas (section C). Multiple arrow buttons can be selected at once, but only one graph button can be selected at any given time. Each specialized button has a corresponding color that is mirrored by the element it creates. In the example, the selected graph button, rX/t, is yellow. As such, the color for the Y-axis label is also yellow.

The **sliders**, section E, set the values before the animation plays. The sliders are in two groups, the initial x values, and the initial y values. The user can change the position, velocity, and acceleration values before the animation. The maximum and minimum values of each slider are displayed on their left(min) and their right(max). The current value is displayed under.

The program is achieved through the interaction of the HTML, the CSS, and the JavaScript code.

CSS was used to layer graphs 3 and 4 (section C). The flex display was used to place the canvases together laterally and the center justify-content orientated the canvases to the center. Canvases 1, 3, and 4 are also scaled on their Y axis by -1 to give them cartesian coordinates and make them easier to work with.

To place the line canvas on top of the graph canvas both canvases were placed within a div, id: graphCanvases. The div's position was set to relative allowing its contained elements (graph & line canvases) to share positions. The line canvas is given an absolute position and moved zero pixels to the right. This places the two canvases in the same position. Setting the line canvas on z-index 2 places it on top of the graph canvas, which is on z-index 1.



```
#canvases{
  display: flex;
  justify-content: center;
  transform: scaleY(-1);
}
#graphCanvases{
  position: relative;
}
#Line{
  position: absolute;
  right: 0px;
  z-index: 2;
}
```

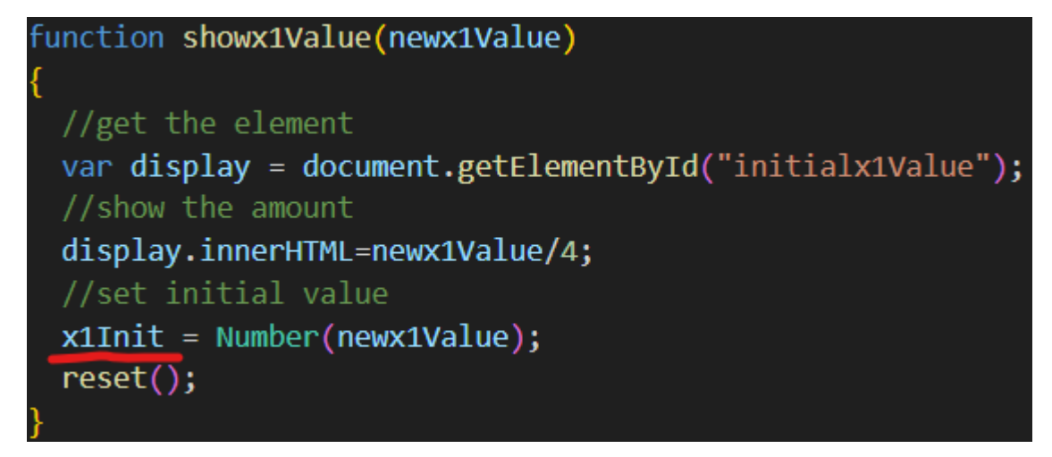
Figure 2: This is the CSS necessary to layer canvaes. applied (1) to all canvases/canvas related divs, (2) to the 'graphCanvases'(graph and line canvases), and (3) to the Line canvas.

Within the HTML, the canvases are canvas elements, and the sliders and buttons are input elements. There is a lot of variety in the input element due to its type attribute. All the button subcategories and sliders are different types: Basic buttons are type 'button', graph buttons are type 'radio', arrow buttons are type 'checkbox', and sliders are type 'range'. Every input element within the program calls a JavaScript function upon user interaction(ie. clicking).

The JavaScript functions can be split into three categories: The **input**(button/slider) functions, the **minor** function, and the **major** functions

The **input** functions are the most numerous category with 18 in total. If you look back at Figure 1, you will notice that the total number of buttons and sliders is also 18; this is no coincidence. Each input function corresponds to an input element and is called upon a user interacting with said element. The input functions can be further separated into the slider functions, the graph functions, the arrow functions, and the basic functions. The 'corresponding variables' mentioned in the next few explanations are variables specific to the role of a function and are used in the drawMotion() function. They are underlined in red in each example image.

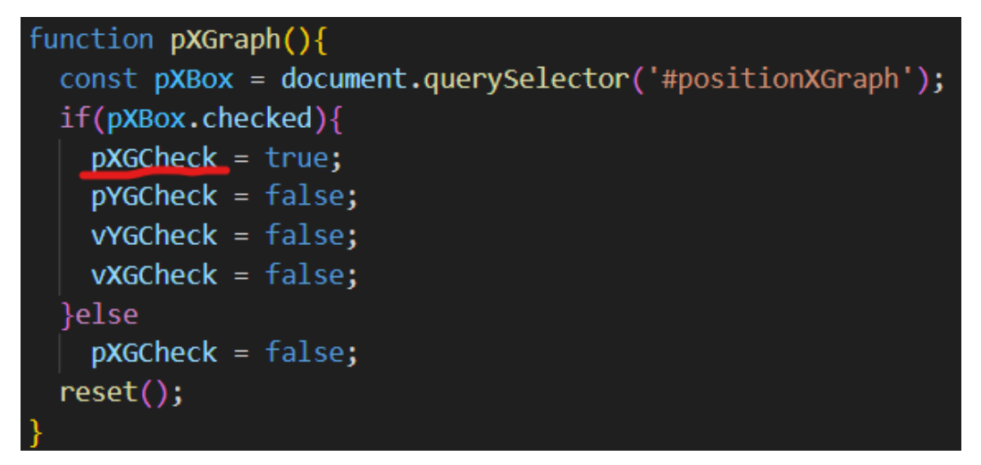
The 6 slider functions correspond to the 6 slider elements. Each function gets the value created by movement of the slider, then changes the value underneath the slider to said value and sets the corresponding variable to said value.



```
function showx1Value(newx1Value)
{
  //get the element
  var display = document.getElementById("initialx1Value");
  //show the amount
  display.innerHTML=newx1Value/4;
  //set initial value
  x1Init = Number(newx1Value);
  reset();
}
```

*Figure 3*: Example of a Slider function

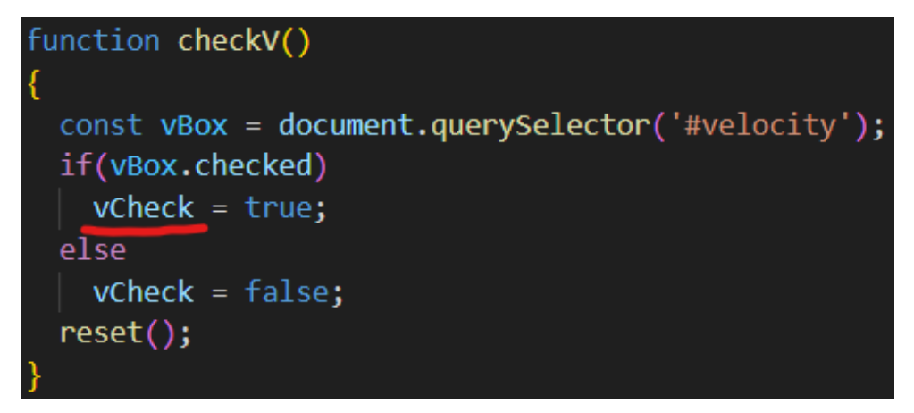
The 4 graph functions give their corresponding Boolean variable a true value and give all other graph function variables a false value.



```
function pXGraph(){
  const pXBox = document.querySelector('#positionXGraph');
  if(pXBox.checked){
    pXGCheck = true;
    pYGCheck = false;
    vYGCheck = false;
    vXGCheck = false;
  }else
    pXGCheck = false;
  reset();
}
```

*Figure 4*: Example of a graph function

The 3 arrow functions check if the corresponding velocity box is checked. If it is, the corresponding variable is given a true value. If it is not checked then the corresponding variable is given a false value.



```
function checkV()
{
  const vBox = document.querySelector('#velocity');
  if(vBox.checked)
    vCheck = true;
  else
    vCheck = false;
  reset();
}
```

*Figure 5*: Example of an arrow function

The basic functions are as follows: play(), pause(), stepBack(), stepForward(), and reset(). The play function initiates the animation. The pause function pauses the animation. The 'step' functions move the animation backward or forward by 0.05 seconds. You may have noticed that every example of the input functions shown above ends with a call to the reset() function.

The **minor** function is the arrow constructor, arrow(). The arrow drawing function creates arrows of varying position, color and size. In this program its only use is to create arrows for representing the velocities that are affecting motion in the animation canvas.

The **major** functions are the most important functions in the program. They include: drawMotion(), runMotion(), drawGrid(), displayValues(), ghost() and reset(). Notice that reset() is both a major and an input function.

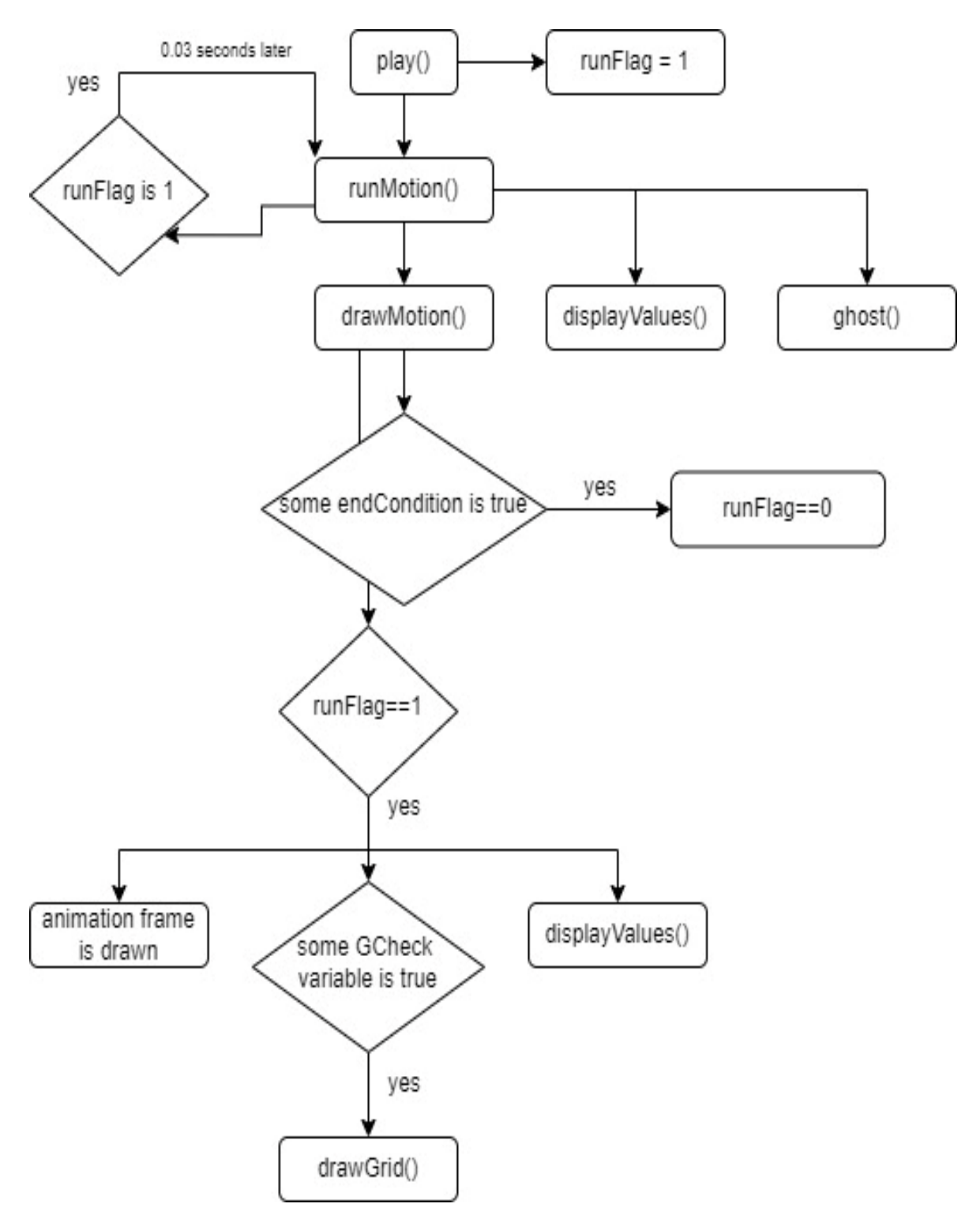


*Figure 6*: Flow diagram that displays the major functions and how they interact to create the program. Note that the rhombus shape indicates an if(conditional) statement while a rectangle indicates a call to a function or an assignment of value (with the one exception being 'animation frame is draw').

Throughout the program, the variable runFlag acts as a sentinel value, meaning it is the condition of termination. If runFlag equals one, the program continues, and if runFlag does not equal one the program ends.

The animation begins when the play button is pressed, calling the play() function. The play function sets the runFlag to 1 and calls the runMotion() function. The runMotion() function calls drawMotion(), displayValues(), and ghost(). The drawMotion function first checks if any of the end conditions[1] are true. If any are true then the runFlag variable is set to zero. The drawMotion function then checks if the runFlag is equal to one, in which case it completes a frame of the animation, displays the new values, and checks each graph button to see if any have been selected.

Once the drawMotion() is completely ran through, we move back to the runMotion function, which checks if the runFlag is one. This is where the program either completes or continues. If runFlag is one, runFlag calls itself 0.03 seconds later and continues. If runFlag was set to zero earlier in the drawMotion function, the program completes.

## Results/Conclusion

The purpose of this program is to act as an educational aid for students who want to get a better understanding of the basics of 2D Motion. It can accurately display the position, velocity, and acceleration of an object, under conditions determined by the user. If you are interested in the full source code, follow this link: https://github.com/IsaiahPhilip/2D-Motion-Web-App.git, or scan the QR code.

## References

1. Bachman, Zoe. "Learn HTML." *Codecademy*, Codecademy, www.codecademy.com/enrolled/courses/learn-html. Accessed 30 Nov. 2023.
2. GregoireGregoire, et al. "HTML5 - Canvas Element - Multiple Layers." *Stack Overflow*, 9 June 2010, stackoverflow.com/questions/3008635/html5-canvas-element-multiple-layers.
3. Lin, Kenny. "Learn Javascript." *Codecademy*, Codecademy, www.codecademy.com/learn/introduction-to-javascript. Accessed 30 Nov. 2023.
4. MozDevNet. "Canvas API." *MDN Web Docs*, developer.mozilla.org/en-US/docs/Web/API/Canvas_API. Accessed 30 Nov. 2023.
5. Mzoughi, Taha. "HTML5 Simulations." *Alo Academy*, 2005, alo.academy/mzoughi/Simulations-html5.shtm.

## Acknowledgements

Taha Mzoughi
Department of Physics
Georgia State University
Georgia State, GA 30302, USA
Email: tmzoughi@gsu.edu

UAP Program
This project was sponsored by the University Assistantship Program of the Georgia State University Honors College.