# K2L Collision Detection

## Today we are going to be refreshing on our Javascript skills, and also going over what “Collision Detection” is, and how we can use it for your game. We used it in your Pong game, although this time you will be creating it yourself.

Part 1 (Drawing, and Logic)

In this part, we’ll be going over what we already learned to draw some shapes and get your shapes moving. If you find all of this too easy, or remember what you are doing, skip to part 2. If you get stuck on part 2, finish part 1 BEFORE you ask questions.

**To start, Download the files by searching “K2L\_Middleschool” on Github.com and opening the K2L\_Collision Folder. Hold Ctrl and click the “index” file, and open with Google Chrome or Safari. Hold Ctrl and click the “mycode” file and open it with your Text Edit. Make sure to turn off Smart Quotes. If you don’t know how, raise your hand.**

1. First, we are going to draw some shapes on the screen.
   1. Inside your “draw” method, insert “c.fillRect(0,0,50,50);” **(DO NOT COPY, TYPE IT OUT)***function draw(elapsedTime){*

*c.fillRect(0,0,50,50);*

*}*

* 1. Refresh your page and see if it works. You should see a small black square.
  2. Next, make the Rectangle **wider**.
     1. Remember! c.fillRect(x,y,**width**,**height**);
  3. Now, make it **taller.**
  4. Now move it somewhere else on the screen.
     1. Remember! c.fillRect(**x,y**, width, height);

1. Now, let’s try making those shapes move!
   1. Above your update method (At the very top!) add in this code:  
      *var x = 0;  
      var y = 0;*
   2. These are going to be the **x** and **y** coordinates of your square!
   3. Next, **inside** your update method, add the following code:  
      *x = x + 1;  
      y = y + 1;*
   4. It should look like this:  
      *function update(elapsedTime){*

*collisionDetection();*

*x = x + 1;*

*y = y + 1;*

*}*

* 1. Refresh your page and make sure it works! If you have any errors, **make sure you did NOT copy and paste the code.** Retype it all out, make sure everything is spelled correctly!.

1. Lastly, let’s use a variable type called a “Boolean”, or true and false variable, when a key is pressed. We used this briefly before, so if you don’t remember what it is, that’s fine.
   1. Above your “update” method, type the following code:  
      *var move = false;*
   2. Now go down to the very bottom and you should see some pressed functions, find one you want to start the movement of your square and modify it like so:   
      *function downPressed(){*

*move = true;*

*}*

* 1. This means that when you press the “down” arrow key or “s”, the move variable will be true.
  2. Next, go into your update method and add:  
     *if(true){  
     …  
     }*
  3. The “…” is where your movement code goes from the last part (step 2). The whole method should now look like this:  
     *function update(elapsedTime){*

*collisionDetection();*

*if(move){*

*x = x + 1;*

*y = y + 1;*

*}*

*}*

* 1. Save your file, and refresh your game page. The square should **NOT** be moving. Press the key you selected, and see if it moves. If so, you are ready to start Part 2. Delete everything you added in Part 1.

Part 2 (Creating Collision Detection)

In this part, we are going to create your own collision detection method! When two objects touch on your screen, this method will be able to tell and allow you to perform an action when it happens! This is a major part of ANY game, so make sure you do your best to understand how it works! Feel free to work with a partner if you believe it will make it easier.

1. To start off, we are going to create Two Rectangles. We also need them to have x, y, width, and height variables.
   1. Above your update method, add in x, y, width, and height variables for **BOTH** rectangles. An example is seen below (Feel free to use what I do)  
      *var s1X = 0;*

*var s1Y = 0;*

*var s1Width = 20;*

*var s1Height = 20;*

*var s2X = 100;*

*var s2Y = 100;*

*var s2Width = 75;*

*var s2Height = 75;*

* 1. Next, lets draw these rectangles. Inside your draw method, draw the two rectangles using c.fillRect(x, y, width, height) for **BOTH** rectangles.
     1. Remember! If you get stuck, **see Part 1 : Step 1**
     2. When you save and refresh, you should see **2** rectangles.

1. Next, lets make one of these Rectangles Move. We are going to use the arrow keys, similar to our pong game.
   1. If you remember how to do it, **try to do it without looking below**. If not, I’ll show you how to make it move…
      1. **UP**
         1. To make if move up, modify the upPressed method like so:  
            *function upPressed(){*

*s1Y = s1Y - 10;*

*}*

* + 1. **Down**
       1. To make it move down, do the same as above, but for downPressed and opposite (+10). Try to figure it out, if you can’t just raise your hand.
    2. **Right**
       1. To make it move right, modify the rightPressed method like so:  
          *function rightPressed(){*

*s1X = s1X + 10;*

*}*

* + 1. **Left**
       1. Similar to how we did “Down”, you can make it move left by doing the same thing you did for rightPressed, but opposite (-10)
  1. Now save your code, and refresh your page. One of your squares should now be able to move in any direction (only one should be moving).

1. The next part is where we will be calculating the collision detection… I will have talked briefly of the math behind detecting a collision, and here is where you will try to figure it out! I recommend working with a partner to try and figure it out!
   1. First, go into your update method, and change “collisionDetection()” to:  
      *collisionDetection(s1X, s1Y, s1Width, s1Height, s2X, s2Y, s2Width, s2Height);*
   2. This is going to be **your** Collision Detection Method
   3. Now go down until you see the old collisionDetection method. Change it so that it matches what you changed in the update method. (See Below):  
      *function collisionDetection(x1, y1, width1, height1, x2, y2, width2, height2){*

*//Your code goes here*

*}*

* 1. If there was a collision, have it return **true**, so it would look like this:  
     *function collisionDetection(x1, y1, width1, height1, x2, y2, width2, height2){*

*//Your code goes here  
return true;*

*}*

* 1. And in your update method, do something like this:  
     *if(collisionDetection(s1X, s1Y, s1Width, s1Height, s2X, s2Y, s2Width, s2Height)){*

*background = "#FF0000";*

*}else{*

*background = "#FFFFFF";*

*}*

1. **HINTS** 
   1. In order to tell if there was a collision, you have to use if() statements. We used them briefly, but I’ll explain again how they work.
   2. Examples:
      1. You can use them to compare numbers, for example  
         if(1 > 2){  
          **//If it was true**}else{ **//If it was false**

}

* + 1. In the above example, it is **FALSE** because 1 is **NOT** greater than 2! So, if you used your variables from before…  
       if(s1X > s2X){  
        **//The left corner of square 1 is on the right side of the left corner of square 2**}else{ **//The left corner of square 1 is on the left side of the left corner of square 2**}
    2. You can chain them together, if you wish.   
       if(1 > 2 && 1 > 0){  
        //If 1 is greater than 2 **AND** 1 is greater than 0  
       }else{  
        //if it’s false  
       }
    3. In the above example, the “&&” signs mean **AND**. So if both are true, the statement is true. If **EITHER** are false, it returns false. It is false because 1 is **LESS** than 2, even though it’s greater than 0! Below we have **OR** which means as long as 1 of them is true, then it’s all true:  
       if(1 > 2 || 1 > 0)  
        //If 1 is greater than 2 **OR** 1 is greater than 0  
       }else{  
        //if it’s false  
       }
    4. In the above example, the “||” signs mean **OR**. Because 1 is greater than 0, this statement is **TRUE** even though 1 is **NOT** greater than 2.
  1. If you want help trying to figure it out, raise your hand. If you are confused by the Boolean logic, try visiting and reading <http://www.quirksmode.org/js/boolean.html>
  2. If you can’t figure it out, and just want something that works, you can copy this code and replace your method:  
     *function collisionDetection(x1, y1, width1, height1, x2, y2, width2, height2){*  
     *return !(*

*((y1 + height1) < y2) ||*

*(y1 > (y2 + height2)) ||*

*((x1 + width1) < x2) ||*

*(x1 > (x2 + width2))*

*);  
}*