# GD 11
# Build The Roller Coaster, Part 1

Roller coasters are fun because the ramp is so wild. Some roller coasters have multiple loops while others focus on rising and falling.

In this lesson, we will combine art with game development. We will use our mouse to draw a custom ramp and then use the keyboard to active the game.

**Build The Roller Coaster** – Press down on the left button of the mouse. Next, move the mouse around to draw your ramp. The goal is to match the national flag with the correct nation. Answer quickly and more ramp is build to extend the or else the roller coaster will fall into the pit of misery and the game is over.

1. Event listener 2. Pixel & Path

---

**1. Write the code below in between <script> </script>.** The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

---

```
/* ===================================
* Draw Ramp Event Handlers
* =================================== */
window.addEventListener("mousedown", path_start);

window.addEventListener("mouseup", path_end);

window.addEventListener("mousemove", new_point);

window.addEventListener( "keypress", check_key );
```

**Explanation**

The code above sets up our event and event handler. The code in **red** is our event. Let's focus on the "mousedown" event. We don't start drawing until the user presses down on the left button of the computer mouse. So, we have to set up an event listener to detect this and the code in red sets up the detection.

Next, when the game does detect that the user has pressed down on the computer mouse, we have to react to the event. The code in **green** is a JS function call and it is our event handler ( ie. reaction to an event )

When we start to draw a shape, the code will create a path based on our mouse movement. The code in **blue** sets up the event listener for when the mouse moves to a different location.

The code in **orange** sets up an event listener for when we stop drawing. In our case, when the user no longer presses down on the left button of the mouse, then the drawing is done. The code in **orange** will detect and stop the drawing by calling the JS function `path_end`.

**2. Write the code below in between <script> </script>.** The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
 /* ============================
* class PointXY
*
* ============================ */
class PointXY
{
  constructor( xP = 0, yP = 0 )
  {
    this.xPoint = xP;
    this.yPoint = yP;
  }

  get_profile()
  {
    return { x: this.xPoint, y: this.yPoint };
  }
}
```

**Explanation**

The code above is the class definition for PointXY. We are using a class to combine together the (x,y) position of a single pixel. Every time the user presses down on the left button of the mouse and then moves the mouse, it will create a new (x,y) point. The new (x,y) point used to create an object.

Later on, we connect all of the objects together to create the shape of our ramp.

**3. Write the code below in between <script> </script>.** The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ==================================
* First Point
* ==================================*/
if( sampleCount <= 0 )
{

  prevMouseX = eventObj.offsetX;
  prevMouseY = eventObj.offsetY;

  // -- put into array
   pObj = new PointXY( eventObj.offsetX, eventObj.offsetY );
   pointArr[sampleCount] = pObj.get_profile();

   sampleCount++;

}
```

**Explanation**

When the game starts, the player has not drawn the ramp. So, the code above sets up the starting point of the ramp. Notice that the code in **grey** uses the `PointXY()` class definition that we previously written code for.

The code in **grey** uses the (x,y) position of the mouse as input commands to create a new object of type `PointXY()`. This is the first pixel of the ramp. Next, we store the first pixel into the array that is named `pointArr`. This time, we manually select a single position of the array using square brackets.

Since we already have the starting point, the code in **blue** increments to the next point. In the end, we connect all of the points together to create the ramp.


# CONTINUE TO THE NEXT PAGE

**4. Write the code below in between <script> </script>.** The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
       /* ==================================
       * Second Point & Beyond
       * ==================================*/
      else
      {
        pathString = "M " + prevMouseX + " " + prevMouseY + " L " +
eventObj.offsetX + " " + eventObj.offsetY;


        prevMouseX = eventObj.offsetX;
        prevMouseY = eventObj.offsetY;

        // -- put into array
        pObj = new PointXY( eventObj.offsetX, eventObj.offsetY );
        pointArr[sampleCount] = pObj.get_profile();
        console.log( "x: " + pointArr[sampleCount].x + ", y: " +
pointArr[sampleCount].y );
        console.log( "pointArr[" + sampleCount + "]: " + pointArr[sampleCount] );

        sampleCount++;
        console.log( "sampleCount: " + sampleCount );


        // -- draw
        var pathObj = new Path2D( pathString );
        ctx.beginPath();
        ctx.lineWidth   = "3";
        ctx.strokeStyle = "#ffa500";
        ctx.stroke(pathObj);

      }
```

**Explanation**

In the previous step, we began to draw the ramp by saving the (x,y) position of the mouse.

In the code above, we continue to save the (x,y) position of the ramp in order to **continue** to draw the ramp. The code looks very similar except the code in **red is new**.

How do we make a shape? We have to connect the points together with a line. The code in **red** connects the previous point with the point that was recently drawn. **When all points are connected together, we create the shape of our ramp.**

The code in **red** uses special symbols. The "**M** " → move to the previous point and the "**L** " means to make a line. So, this translates into "**Move to the previous mouse position and draw a line to connect the previous mouse position with the current mouse position.**"