CS/INFO 3300; INFO 5100
Homework 4
Due 11:59pm **Wednesday** February 27

Goals: Practice using d3 to create SVG elements and set their aesthetic properties. Work with scales some more, and practice data cleaning. Use events to begin making interactive charts.

Your work should be in the form of an HTML file called index.html with one <p> element per (sub)problem. Wrap any SVG code for each problem in a <svg> element following the <p> element(s). For this homework we will be using d3.js. **In the <head> section of your file, please import d3 using this tag: <script src="https://d3js.org/d3.v5.min.js"></script>**

Create a zip archive containing your HTML file plus associated data files (such as blockbusters.json) and upload it to CMS before the deadline.

1. You've now seen a few scatterplots in class as well as developed your own in HW3. In this problem you will once again be developing a scatterplot, however this time many of the specific **design features of the plot will be left to you**. Inside the homework ZIP file you will find **blockbusters.json**, a dataset of the top 10 highest grossing movies for each year from 2018 to 1975. Please use this dataset for the rest of the problem. For each of the following sub-problems, please briefly **explain your procedure or design rationale for that step** (e.g. How did you decide on axis scale labels? What made you choose a linear/log scale? What compromises did you make [such as hiding some numbers so the axes are easier to read)?].

A. This data file isn't exactly perfect. In fact, we've gone ahead and **added some more points with confusing, missing, or bad data values**. Worse, we've not bothered fixing any types or standardizing values. Begin by loading the data file using an asynchronous request with **d3.json**. **Implement the rest of this problem in the promise function**. Save the data array in a variable blockbustersData that is defined **outside** the scope of the callback function as a var. Within your promise, use **filter** and a **forEach** loop to **hide or correct any important data quality issues**. For efficiency's sake, feel free to gather maximum and minimum values at this time as well, or you can use d3.max and d3.min later on in step B. Describe what data issues you **found** and **how you fixed them** in your <p> tag.

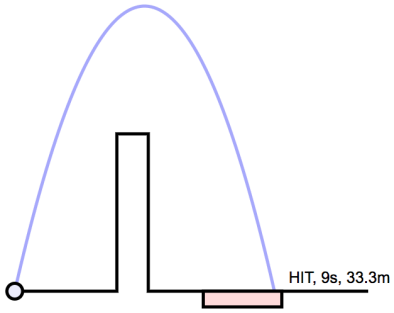
B. Create an SVG canvas **800 pixels in width and 500 pixels in height**. You are going to be graphing **year of release on the x-axis**, showing **worldwide gross in dollars on the y-axis**, creating and varying the **radius of circles based on imdb rating**, and **coloring them by main genre**. In this step, please **construct scales for your chart**. Create **<g> elements**, use **translate** to move them to an appropriate place, and **populate them with d3.axis labels**. Use a **second set of d3.axis objects** to create **corresponding x- and y-axis gridlines** in a light color.
Feel free to choose whatever margin/padding values, domain, range, log/linear scales, colors, and axis formatting styles you like. Use the <p> tag to **explain the choices you made**

in designing the axes/scales. We will reward scales/axes that are legible, show the distribution of data clearly (or as clearly as possible), use color effectively, and avoid visual clutter. A good rationale in the <p> for your design will outweigh problems in these areas.

C. Create a **<g> element** and **translate** it so that it can act as your **main chart region**. Now, populate the chart with **<circle> elements corresponding to valid data points**. Move, scale, and color them as necessary using the structures you built in step B.

D. Using **d3.on("mouseover") and d3.on("mouseout"),** provide users a way to move their mouse **onto points** and see the **title of the movie they are hovering over**. At minimum, points should **grow in size** and a **floating <text> label should appear nearby** when the mouse enters the inside of a circle. Do not use HTML elements; your event actions must happen entirely within SVG canvas elements. When the mouse leaves the circle, it must **return to its normal appearance**. Feel free to add more complexity to make the highlighting of points more obvious or the text more legible, but this is not explicitly required. Write 2-3 sentences in your <p> tag describing the possible **benefits to users** from this approach and identifying places in the chart where it may be **ineffective or confusing** (or, if you fixed them, what you did to improve the user experience).

2. In this problem you will simulate **projectile motion under the influence of gravity** using a finite approximation, where we estimate a ball's position every 0.1 seconds. Physics review: keep track of the position (displacement), velocity, and acceleration for the x and y dimensions separately. The finished work should look ***similar*** to the diagram, but does not need to replicate it exactly.



In this scenario, a Martian is trying to play catch with their neighbor. Unfortunately, some jerk went and built a wall between their houses. The pitcher is **13 meters away from the wall**, which is **20 meters tall and 4 meters thick**. On the other side of the wall, their neighbor's field is between **8 and 18 meters away** from the wall (25-35 meters from the pitcher). The neighbor is happy to run any distance in that range, so anything landing (at y=0) between **8 and 18 meters from the wall will be caught**. Like many physics story problems, we'll ignore the Martian wind and atmosphere, friction, the height of the pitchers, and any other realistic considerations. We will only use Mars gravity, which is a downward acceleration of 3.71m/s/s.

A. Contextualize the data. Usually we do this with axes and other guides. Here we will show a simplified SVG drawing. Create an SVG canvas **400 pixels high and 400 pixels wide**. Choose your pixels such that **5 pixels in both the x and y axes corresponds to 1 meter of distance** in the scenario. While an example drawing has been provided at the end of the assignment **(you are welcome to copy it into your submission),** feel free to make your **own drawing of the scenario if you'd like**. To make the calculations in the next sections easier, treat the location of the pitcher as (0,0). This makes the top corners of the wall (13,20) and (17,20), and the target

area between (25,0) and (35,0). **Create linear scales for the x and y dimensions that map meters to pixels.** Be careful to make sure that your domains and ranges account for the difference on origin (lower left for problem, upper left for SVG canvas [hint: reverse a range]).

B. Generate a data array. Create a function ***trajectory*** that takes an **initialVelocity** in meters per second and an initial **angle**, returning an array of objects. Each object in this array should have seven variables: **ground, x, y, xVelocity, yVelocity, xAcceleration, and yAcceleration**. You will need to set the initial conditions (at array index 0) specially: **set the initial x velocity to the initial velocity times the cosine of the initial angle, use the sine for the y angle**. The acceleration in both dimensions will be constant, and will represent the change in the ball's speed every tenth of a second. In the x dimension acceleration should be zero. **Acceleration in the y dimension should be -3.71 / 10, to account for Mars gravity.** For each successive object in the array, **set velocity in the x and y dimensions equal to the previous velocity plus the current acceleration. The x and y positions should be equal to their previous values plus 0.1 × their current velocities.** Finally, to see whether the ball has landed or hit the wall, calculate the **current height of the ground by setting the ground variable to either 0 or 20 depending on where the x position is located** (i.e. the ball is over the wall or not). The array should comprise **exactly as many elements as you need to hit the ground** (i.e. y <= ground).

C. Display data. Create a function ***plotTrajectory*** that takes an **array** of the format created by the function in part B and a **string containing a color** (e.g. "#abb043") for the plot. This function should use **d3.line()** to create a 25% opacity 5-pixel-wide <path> element tracing this trajectory, colored using the provided string. Place a **<text> element near the point of impact** with a label showing if the ball **HIT or MISSed the target**, the **time it took in flight**, and the **distance from the take-off point to the landing point** (consult Pythagoras and Euclid).

D. Display the trajectory for a toss thrown at **14 meters per second and a 75 degrees angle** (this is about 1/3 to 1/2 the speed of an Earthling amateur baseball pitch). Show **two** additional trajectories with **velocities and angles of your choosing**. Provide a blue color for the lines for these trajectories. Avoid overlapping text labels.

E. Create a second version of trajectory, ***trajectoryWithWind***, that takes an additional parameter: **constant acceleration due to wind in m/s/s**. Like trajectory, this function will generate an array of objects with 7 properties, but this time accounting for a windy day. The math will work out the same, with the exception that **(wind acceleration / 10) becomes x acceleration in the initial conditions before iteration** (wind **towards** the target is positive). Using this new function, calculate a **13 meters per second throw at 80 degrees with a 1m/s/s wind speed towards the target.** Use your plot function to draw a new d3.line() for this throw, this time using a **red** <path> element. There is no need to create a <text> element for this throw, but you are welcome to do so if it will fit on screen.

SVG FOR PROVIDED DIAGRAM (FEEL FREE TO USE IN SUBMISSION)
5px = 1m,   (0,0)m from scenario is at (25,10)m or [125,350]px

```
<svg height="400" width="400">
  <g id="background" style="stroke: #000; stroke-width: 2px; fill: none;">
    <!--- Wall is 13 meters from pitcher. 20 meters tall and 4 meters thick.
          Target is between 8 meters and 18 meters from the wall.  --->
    <path d="M75 350 H 140 V 250 H 160 V 350 H 350"/>
    <!--- 1 meter is 5px, so our pitcher is at (15,10) from the lower left corner --->
    <circle id="pitcher" cx="75" cy="350" r="5" style="fill: #EEF"/>
    <!-- Our target is between (40,10) and (50,10) from the lower left --->
    <rect id="target" x="200" y="350" width="50" height="10" style="fill: #FDD"/>
  </g>
</svg>
```