



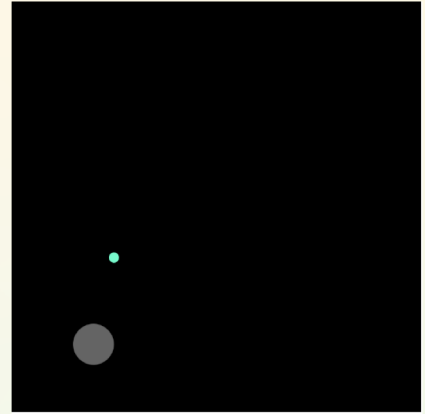
Girls Who Code en casa

Juego del Meteor Catcher: Parte 4
Capturar el meteorito

Descripción de la actividad

Al final de la Parte 3, aprendiste a crear y usar variables en p5.js para mover el meteorito por la pantalla a una velocidad especificada. ¡Es hora de interactuar! En esta parte, aprenderás cómo agregar la entrada del jugador programando un capturador que responda al movimiento del ratón. Este capturador es una elipse blanca translúcida que se mueve con el ratón. Haz clic [aquí](#) para obtener una vista previa de lo que aprenderás al final de la actividad.

Ya deberías haber completado la [Parte 1](#), la [Parte 2](#), y la [Parte 3](#) de la **serie de juegos Meteor Catcher** antes de embarcarte en esta actividad.



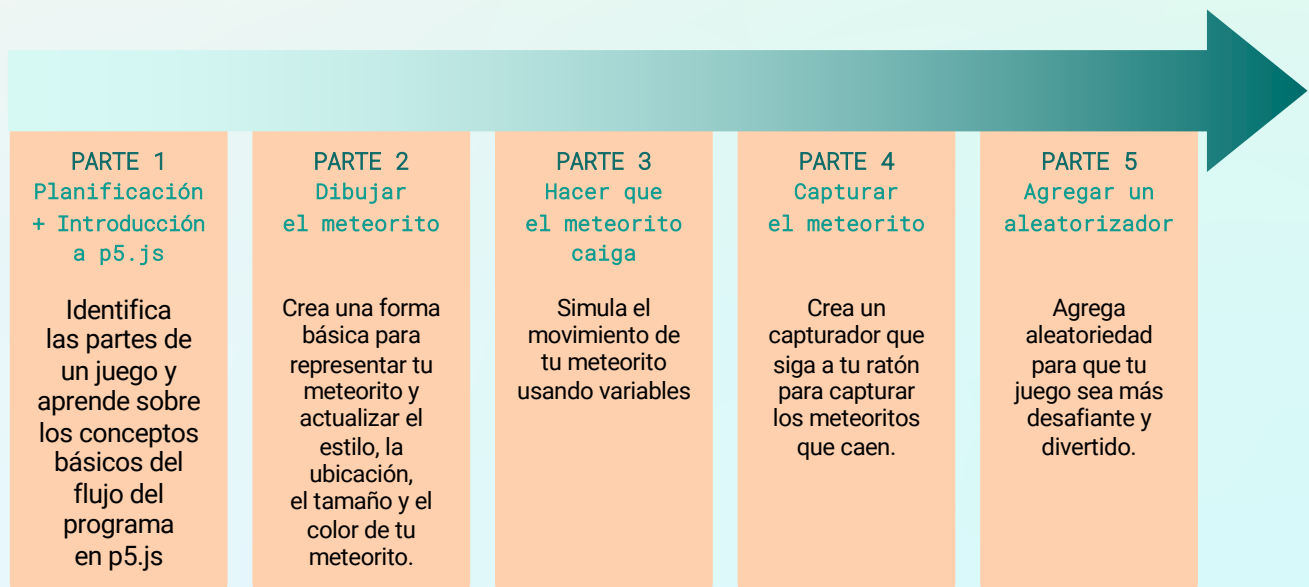
Objetivos del aprendizaje

Al finalizar esta actividad, serás capaz de:

- ☐ describir cómo simular el movimiento básico en un programa.
- ☐ programar diferentes comportamientos en elementos usando variables y operadores aritméticos.

Materiales

- [Editor en línea de p5.js](#)
- [Proyecto de muestra del juego Meteor Catcher](#)
- [Guía de referencia de la parte 4 del juego Meteor Catcher](#)



Artículo destacado sobre “Mujeres en tecnología”: Rebecca Cohen Palacios



Fuente de la imagen:
GamesIndustry.biz

Si alguna vez has jugado a los siguientes videojuegos: Elder Scrolls: Blades, Assassin's Creed Origins, Assassin's Creed Syndicate o Shape Up (Kinect), ¡entonces ya has visto algunos de los trabajos que Rebecca ha contribuido a crear! Hay más mujeres trabajando en la industria de los videojuegos, y parte de esto se debe al trabajo de Rebecca y a la organización que cofundó con Tanya Short, [Pixelles](#).

Pixelles es una organización sin fines de lucro dedicada a alentar a más mujeres en el desarrollo de videojuegos. Rebecca se dio cuenta de que, además de que los videojuegos aún se percibían como algo “para niños”, también había un gran porcentaje de mujeres en la industria del videojuego que abandonaron el campo debido al sesgo, la falta de apoyo y a las barreras laborales. Pixelles aborda estos problemas al ofrecer a las mujeres sin experiencia o a mitad de su carrera a través de talleres mensuales gratuitos, tutorías, programas de “creación de su primer juego”, un acelerador de carreras, redes sociales y mucho más...

Además de ser codirectora de Pixelles, Rebecca también trabaja actualmente en [Behaviour Interactive](#) como diseñadora sénior de IU. Antes de comenzar en la industria del juego en Ubisoft, Rebecca pasó 6 años como desarrolladora de Gráficos y Web.

Obtén más información sobre Rebecca y cómo su trabajo con [Pixelles](#) trata de cerrar la brecha de género en la industria del juego.

- ["Pixelles está ayudando a las madres con experiencia profesional a permanecer en los videojuegos"](#)
- ["Los 7 motivos principales por los que las mujeres abandonan el desarrollo de videojuegos"](#)
- ["Una organización sin fines de lucro de Montreal brinda a las mujeres la oportunidad de entrar en la industria de los videojuegos dominada por hombres"](#)
- ["Empoderamiento a través del desarrollo de videojuegos": La incubadora de videojuegos Pixelles"](#)

Reflexión

Ser un experto informático es más que sencillamente ser bueno programando. Toma unos minutos para reflexionar sobre cómo Rebecca y su trabajo reflejan las características que todos los verdaderos expertos informáticos deben desarrollar en sí mismos: valentía, resiliencia, creatividad y propósito.



RESILIENCIA

¿Cuáles son las dificultades que enfrentan las mujeres en la industria del videojuego? ¿Cómo ayuda Pixelles a las mujeres a perseverar en sus carreras?

Comparte tus respuestas con un familiar o amigo. Anima a otras personas para que lean sobre Rebecca y se unan a la charla.

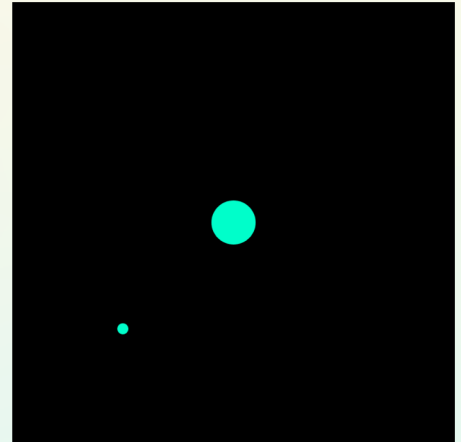
Paso 1: Agregar el capturador (5 a 10 minutos)

Dibujar el capturador (3 a 5 minutos)

Primero, debemos dibujar el capturador a la pantalla usando la función `ellipse()`. Tal como hicimos con el meteorito, usaremos variables para almacenar el ancho y la altura del capturador. Utilizaremos variables especiales para la posición x e y en el siguiente paso, por lo que no necesitamos crear variables para almacenar estos valores.

- ☐ **Agrega una nueva variable por encima de `setup()` para almacenar el ancho y la altura de nuestro capturador.** Ya que nuestro capturador es un círculo, usaremos el mismo valor para ambos. Nombramos a nuestra variable `catcherDiameter`, pero puedes nombrarla como desees. Solo asegúrate de hacer referencia a ella correctamente más adelante en tu código.
- ☐ **Asigna la variable de ancho y altura de tu capturador a un valor de 40.**
- ☐ **En la función `draw()` dibuja el capturador usando la función `ellipse()`** Agrega esto debajo de tu código para el meteorito. Agrega un comentario breve para recordar que este es el capturador.
- ☐ Establece los parámetros x e y en `200`, luego establece los parámetros de ancho y altura en la variable que creaste anteriormente.
- ☐ Ejecuta tu código.

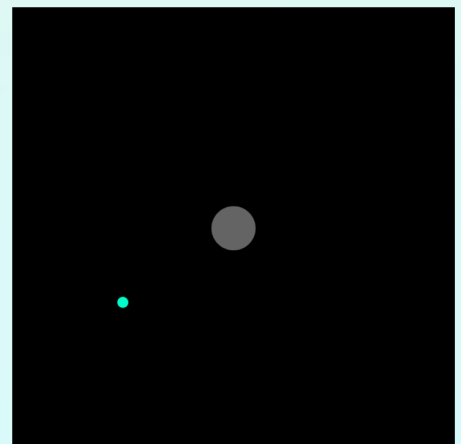
Deberías ver un círculo verde azulado en el medio del lienzo:



Cambiar el color del capturador (3 a 5 minutos)

Queremos que nuestro capturador sea blanco y semitransparente para que podamos ver nuestro meteorito a través del capturador. Esto significa que debemos usar nuestra función `fill()` nuevamente. Anteriormente mencionamos que `fill()` agregará color a todas las formas debajo de ese comando hasta que le diga al programa lo contrario, como limpiar un pincel y sumergirlo en otro color.

- ☐ Crea una nueva función `fill()` sobre la elipse del capturador. Esto aplicará el nuevo color a todas las formas que se encuentran debajo.
- ☐ Agrega los valores RGB para que tu capturador sea blanco.
- ☐ Agrega un cuarto parámetro para controlar la transparencia y establecer este valor en `100`. Este parámetro opcional se denomina valor alfa. Puedes establecer cualquier valor entre 0 y 255 con 0 como completamente transparente y 255 como opaco. Puedes intentar entrenar con algunos valores diferentes para comprender cómo cambia la visualización del capturador.
- ☐ Ejecuta tu código.



Deberías ver que el color del capturador cambia a blanco semitransparente. En la siguiente imagen, el capturador puede aparecer en gris. Ya que es semitransparente, parte del color de fondo se filtra.

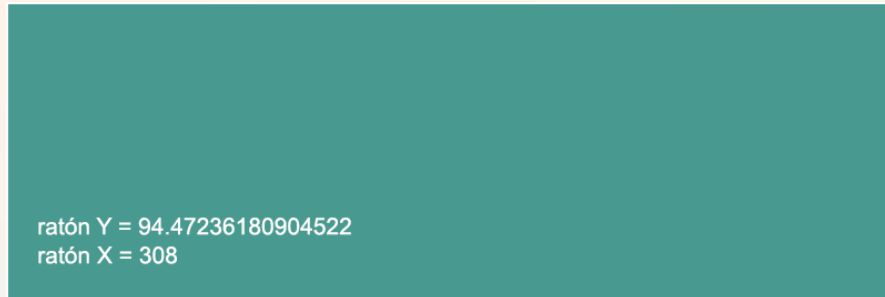


No olvides verificar tu código con la Guía de referencia en la página 2.

Paso 2: Añadir entrada del jugador (2-4 minutos)

En este momento, el capturador no está capturando mucho. Queremos que un jugador pueda controlar su movimiento con el ratón. Esto significa que necesitamos encontrar una manera de cambiar los valores x e y del capturador para que coincidan con los valores x e y del ratón.

Por suerte para nosotros, hay dos variables incorporadas en p5.js que hacen esto por nosotros. Las variables `mouseX` y `mouseY` contienen la posición de la posición horizontal y vertical del ratón. Intenta mover el ratón en el [bordado de ejemplo](#) para ver cómo cambian los valores.



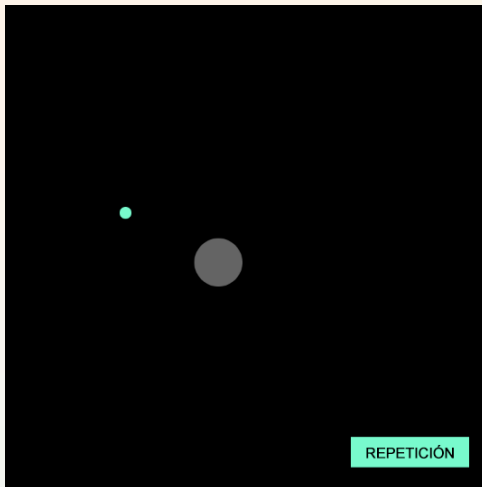
Podemos usar `mouseX` y `mouseY` para conectar el capturador al ratón. Esto significa que la posición x e y del ratón se convierte en la posición x e y del capturador. Actualicemos nuestro capturador:

- ☐ Reemplaza el valor actual de x en la elipse del capturador con `mouseX`.
- ☐ Reemplaza el valor actual de y en la elipse de tu capturador con `mouseY`.

Paso 3: Probar tu código (2 a 5 minutos)

Pongamos a prueba lo que hemos escrito hasta ahora para asegurarnos de que nuestro programa funcione de la manera que queremos. Haz clic en el botón Reproducir para ejecutar tu bordado. Deberías tener:

- Un capturador que sigue el movimiento del ratón.
- El capturador debe ser de color blanco y semitransparente.
- No deberías tener un botón Volver a reproducir.



Haz clic [aquí](#) para ejecutar el bordado del ejemplo.

Nota: Incluimos un botón Volver a reproducir para que puedas restablecer el comportamiento del meteorito. Si no incluyéramos esto, solo verías un cuadro negro una vez que el meteorito salga de la parte inferior de la pantalla. ¡Lo solucionaremos en la siguiente parte con una sentencia condicional, pero todavía no lo hemos hecho!

¿No funciona como deseas? Prueba estos consejos de depuración:

- ¿Tu código está dentro de las llaves correctas?
- ¿Hay punto y coma al final de cada línea de código?
- ¿Escribiste correctamente los nombres de la función y de la variable? Recuerda que JavaScript distingue entre mayúsculas y minúsculas.
- ¿Tus funciones están en la ubicación y secuencia correctas? ¡Recuerda que el orden es importante en el flujo del programa!
- ¿Tienes funciones `fill()` separadas por encima de las funciones `ellipse()` para tu meteorito y capturador?
- Si tu capturador no se muestra, aumenta el valor alfa.
- ¿Agregaste `mouseX` y `mouseY` en la ubicación correcta del parámetro en tu `ellipse` del capturador?



No olvides verificar tu código con la Guía de referencia en la página 3.

Paso 4: Verificar la comprensión

Describe cómo esta línea de código cambiaría el comportamiento de nuestro capturador:

```
ellipse(200, 200, mouseX, mouseY);
```

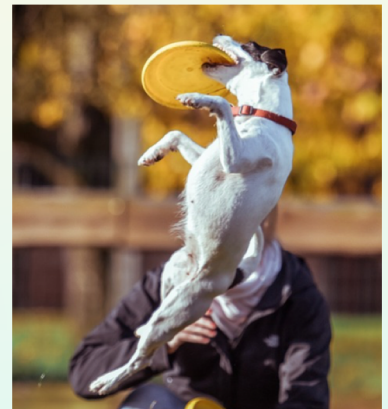


No olvides verificar tu código con la Guía de referencia en la página 3.

Paso 5: Conceptualizar la “captura” (2 minutos)

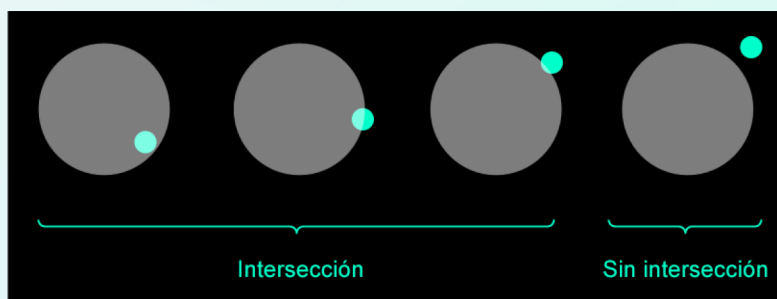
En el último paso, descubrimos una manera de hacer que un jugador interactúe con nuestro juego. Ahora queremos descubrir cómo hacer que interactúen entre sí los diferentes componentes del juego. Queremos determinar cuándo el capturador ha “atrapado” el meteorito y cuándo el meteorito ha “chocado” contra la parte inferior de la pantalla.

¿Qué sucede realmente cuando capturas algo? Imagina lanzar una pelota o hacerla rodar por el piso hasta un amigo. Cuando toca su cuerpo y lo sostiene, decimos que ha atrapado la pelota. También podríamos decir que la pelota chocó o intersectó con su mano o pie (o pata).



Fuente de la imagen: [Pixabay](#)

Queremos escribir un código que pruebe si el meteorito y el capturador se han intersectado. Esto también se conoce como detección de colisión, un término para cuando se tocan dos formas. Estas son algunas maneras en las que podríamos considerar la detección de colisiones en nuestro programa:



Pensemos en la información a la que tenemos acceso en nuestro programa que podría ayudarnos: conocemos la posición x e y del meteorito, la posición x e y del capturador y el tamaño de cada uno. Aunque sus posiciones cambian constantemente, podemos acceder a esos valores a través de sus variables x e y. ¡Variables al rescate, una vez más!

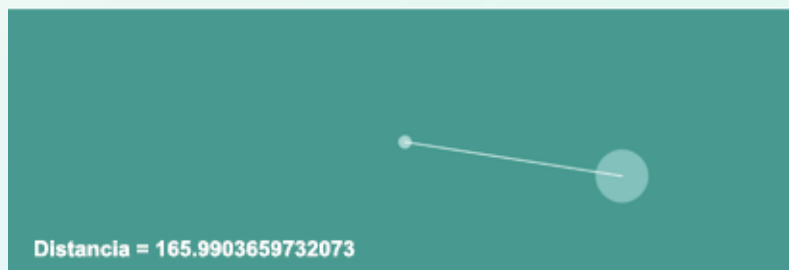
Paso 6: Calcular la distancia (10-15 min)

Cumplir con la función `dist()` (3-5 minutos)

Podemos usar la información anterior para calcular qué tan lejos está el capturador del meteorito en cualquier momento. p5 tiene una función que hace estos cálculos para nosotros: la función `dist()`. Esta función calcula la distancia entre dos puntos. Todo lo que tenemos que hacer es conectar los parámetros. Esta es la sintaxis:

JAVASCRIPT	DESCRIPCIÓN
<code>dist(x1, y1, x2, y2);</code>	<ul style="list-style-type: none">→ <code>dist</code>: El nombre de la función.→ <code>()</code>: Utilizamos paréntesis para indicar a nuestro programa que necesita llamar a la función. A veces incluimos parámetros o entradas en la función dentro de nuestros paréntesis.→ <code>x1</code>: La coordenada x del primer punto.→ <code>y1</code>: La coordenada y del primer punto.→ <code>x2</code>: La coordenada x del segundo punto.→ <code>y2</code>: La coordenada y del segundo punto.→ <code>;</code>: todas las líneas de código en p5.js deben terminar con punto y coma.

Inspecciona el [bordado de demostración](#). El texto muestra el valor devuelto de la función `dist()` entre el círculo central y el círculo del ratón. Intenta mover el círculo del ratón alrededor de él hasta que se intercepte con el círculo central y observa cómo cambian los valores de distancia.



Piensa en ello: ¿Qué rango de valores de distancia representa la intersección? Más adelante, nos haremos esta pregunta cuando completemos nuestro bordado del proyecto para escribir una sentencia condicional para que podamos hacer un seguimiento de la "captura".

Paso 6: Calcular la distancia (cont.)

Agregar `dist()` a tu código (5-8 minutos)

Agreguemos la función `dist()` a nuestro borrador del proyecto. Primero, crearemos una variable para almacenar el valor devuelto de la función (es decir, la distancia entre el centro del capturador y el centro del meteorito). A continuación, agregaremos la función. Por último, utilizaremos una nueva función, la función `print()` para seguir el valor de la distancia a medida que cambia.

- ☐ **Agrega una nueva variable por encima de `setup()` para almacenar el valor de distancia.** Hemos nombrado nuestra variable “distancia”, pero puedes nombrarla como quieras. Solo asegúrate de hacer referencia a ella correctamente más adelante en tu código. No es necesario asignarle un valor.
- ☐ **Llama a la función `dist()` bajo el código del capturador.** Si es útil, puedes agregar un comentario breve para recordar lo que hace esta línea de código.
- ☐ Establece los parámetros para `x1` y `y1` en las variables que almacenan la posición `x` e `y` de tu meteorito.
- ☐ Establece los parámetros para `x2` y `y2` en `mouseX` y `mouseY`.
- ☐ Almacena los resultados de la función `dist()` en tu variable de distancia.



No olvides verificar tu código con la Guía de referencia en la página 4.

Paso 6: Calcular la distancia (cont.)

Imprimir el valor `dist()` a la consola (3 a 5 minutos)

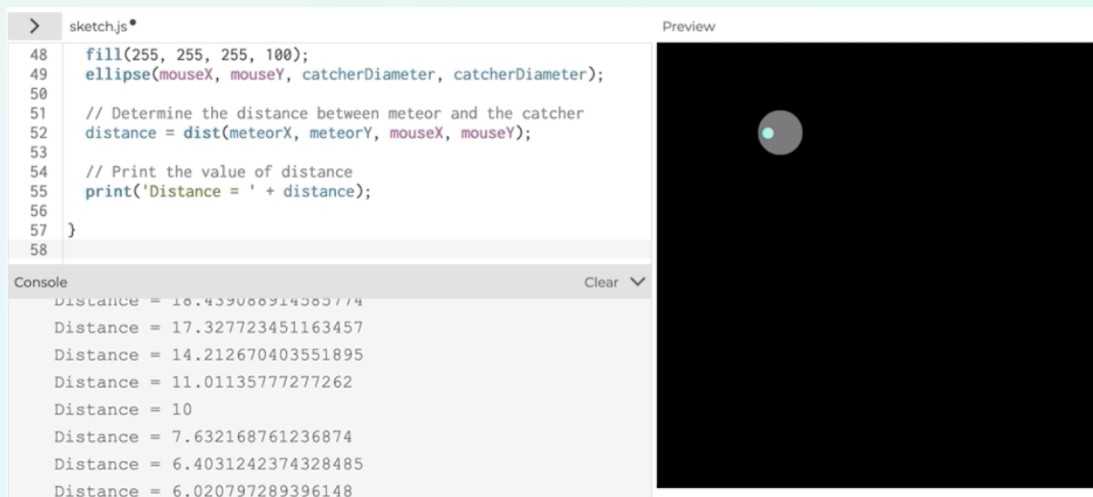
Ahora nuestro programa está calculando constantemente la distancia entre el centro del meteorito y el centro del captador. Pero, ¿cómo podemos verificar esos valores? La forma más fácil de hacerlo es la función `print()`. Imprime valores alfanuméricos, como palabras y números, en la consola debajo del editor. Esta es la sintaxis:

JAVASCRIPT	DESCRIPCIÓN
<pre>print('Distance = ' + distance);</pre>	<ul style="list-style-type: none">→ <code>print</code>: El nombre de la función que imprime mensajes en la consola.→ <code>()</code>: Utilizamos paréntesis para indicar a nuestro programa que necesita llamar a la función. A veces incluimos parámetros o entradas en la función dentro de nuestros paréntesis.→ <code>' '</code>: Las comillas simples o dobles le indican al programa que estamos imprimiendo cadenas o texto. Puedes usar cualquier tipo de comillas, simplemente sé coherente.→ <code>+</code>: Une elementos para imprimirlos juntos.→ <code>distancia</code>: Imprime el valor actual almacenado en la variable de distancia.→ <code>;</code>: todas las líneas de código en p5.js deben terminar con punto y coma.

Escribamos esto en nuestro borrador:

- ☐ Agrega esta línea de código bajo la función `dist()` para imprimir el valor de la variable de distancia:
`print('Distance = ' + distance);`
- ☐ Ejecuta el borrador.

El texto `Distance =` y una serie de valores cambiantes deberán imprimirse en la consola.

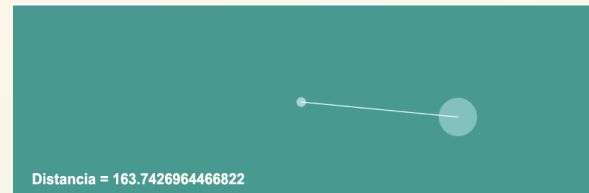


Paso 7: Determinar la intersección (5 minutos)

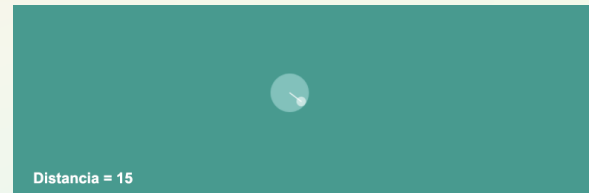
Definir la intersección usando el valor de distancia (2 a 3 minutos)

Sabemos que queremos que nuestro programa se comporte de cierta manera si un capturador y un meteorito se cruzan. Para hacer esto, necesitamos un valor numérico que represente la intersección. Podemos usar el valor almacenado en `distance` (recuerda que puedes haber usado un nombre de variable diferente) para determinar cuándo se cruzan el capturador y el meteorito.

Considera el [bordado de demostración de distancia](#) anterior de nuevo: ¿Qué valor es la distancia igual a cuando el capturador cubre parcial o totalmente el “meteorito”?



Después de explorar el bordado de demostración, podemos decir que la intersección ocurre **si la distancia es inferior a 15**. Ahora podemos comenzar a tomar algunas decisiones en nuestro programa usando sentencias condicionales.



Revisar las sentencias condicionales (2 minutos)

Ahora podemos decirle a nuestro bordado qué hacer si se produce una intersección. Para esto, necesitamos una sentencia condicional. Hagamos un repaso rápido: las **sentencias condicionales** nos permiten controlar el flujo de nuestro programa. Utilizan afirmaciones `if` para verificar si una condición es **verdadera** o **falsa**. Si una condición es verdadera, la computadora ejecutará el código dentro de la afirmación `if`.

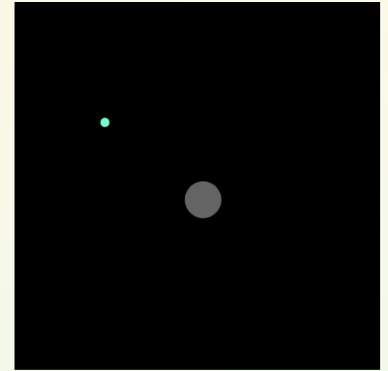
A continuación se muestra un ejemplo para ilustrar la sintaxis de una sentencia condicional en JavaScript (recuerda que p5.js es una biblioteca para JavaScript). Le indica a nuestro programa que dibuje un círculo en la esquina superior izquierda del lienzo si la posición x del ratón es mayor que 100.

JAVASCRIPT	DESCRIPCIÓN
<pre>if(mouseX > 100){ ellipse(0,0,50,50); }</pre>	<ul style="list-style-type: none">→ <code>if</code>: Palabra clave para decirle al programa que esto es una sentencia condicional.→ <code>()</code>: Utilizamos paréntesis para informar a nuestro programa que todo lo que se encuentre en su interior forma parte de la condición que evaluará.→ <code>mouseX > 100</code>: La sentencia condicional que el bordado probará para determinar si es verdadera o falsa. Puedes usar operadores de comparación como <code>></code> (mayor que) o <code><=</code> (menor que o igual a) u operadores lógicos como <code> </code> (o) o <code>&&</code> (y) para configurar su afirmación.→ <code>{}</code>: Las llaves indican a nuestro programa qué líneas de código ejecutar si la condición es verdadera.→ <code>ellipse(0,0,50,50)</code>: La afirmación que se ejecutará si la condición se evalúa como verdadera. También puedes incluir otras afirmaciones <code>if</code> aquí.→ <code>;</code>: todas las líneas de código en p5.js deben terminar con punto y coma.

Paso 8: Preparar la sentencia condicional del capturador (5-10 minutos)

Planear la sentencia condicional del capturador (2 a 4 minutos)

La primera sentencia condicional que creamos será para el capturador. Tenemos que escribir una afirmación `if` que probará para ver si el meteorito y el capturador se cruzan. Antes de escribir cualquier código, describamos lo que queremos que suceda. Juega nuevamente al juego y luego escribe un pseudocódigo para la sentencia condicional. Intenta ser lo más específica posible. Puedes usar el pseudocódigo que escribiste en la sección **Planear la Parte 1** para comenzar. Revisa el [bordado del ejemplo](#) antes de comenzar.



Pista: Puedes usar las variables para la distancia y la posición y del meteorito que creaste en los pasos anteriores. También necesitarás un número que represente la intersección.



No olvides verificar tu código con la Guía de referencia en la página 5.

Agregar la sentencia condicional del capturador (3 a 5 minutos)

Luego, traduce tu pseudocódigo al código real:

- ☐ Agrega una afirmación `if` que verifique el valor de la variable de distancia.
- ☐ **Agrega una línea de código que actualice la posición y de tu meteorito.** Asegúrate de que esté dentro de las llaves de las sentencias condicionales.
- ☐ Ejecuta tu código.
- ☐ Trata de hacer que tu meteorito y tu capturador se entrecrucen. Cuando se crucen, el meteorito actual deberá desaparecer y un meteorito “nuevo” deberá aparecer en la parte superior de la pantalla.

Si esto no sucede, comprueba los valores de distancia que se imprimen en la consola. Cuando el capturador y el meteorito se cruzan, ¿los valores están en el rango especificado en la sentencia condicional? Si no es así, es posible que debas ajustar el número en tu sentencia.

La función `print()` es muy útil para eliminar bugs. Si en algún momento no estás segura del valor de una variable en tu bordado, coloca una afirmación `print()` debajo para volver a verificar.

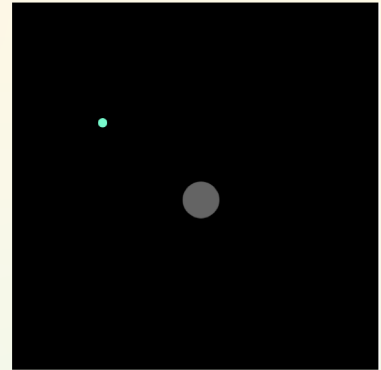


No olvides verificar tu código con la Guía de referencia en la página 5.

Paso 9: Configurar la parte inferior de la sentencia condicional (5-10 minutos)

Planear la parte inferior de la sentencia condicional (2 a 3 minutos)

Ahora escribamos la sentencia condicional para probar si el meteorito se ha intersectado con la parte inferior de la pantalla. Si se ha intersectado con la parte inferior del lienzo, queremos que el programa lo vuelva a dibujar en la parte superior del lienzo. Si buscas una forma rápida de hacer referencia a la altura o al ancho del lienzo en tu código, puedes usar las palabras clave `height` y `width` en lugar del valor numérico. Revisa el [bordado de ejemplo](#) antes de comenzar.



Recuerda: la altura de nuestro lienzo es de 400 píxeles.

Escribe el pseudocódigo para esta sentencia condicional. Al igual que antes, trata de ser lo más específica posible. Puedes usar el pseudocódigo que escribiste en el **paso anterior** o la **sección Planear de la Parte 1** para comenzar.



No olvides verificar tu código con la Guía de referencia en la página 5.

Agrega la parte inferior de la sentencia condicional (3-5 minutos)

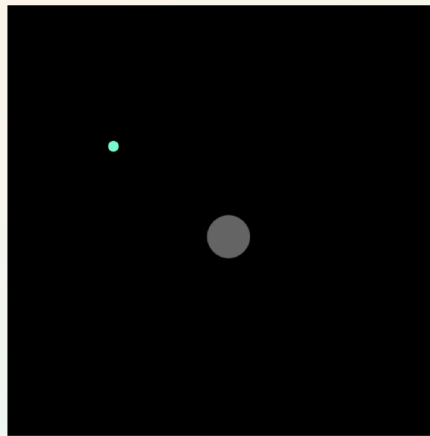
Ahora, traduce tu pseudocódigo al código real:

- ☐ Escribe la afirmación `if` y la sentencia condicional.
- ☐ Agrega el enunciado que se ejecutará si la sentencia condicional es verdadera. Asegúrate de que esté dentro de las llaves de las sentencias condicionales.
- ☐ Ejecuta tu código.
- ☐ Espera a que el meteorito se cruce con la parte inferior de tu lienzo. Cuando se crucen, el meteorito actual deberá desaparecer y un meteorito "nuevo" deberá aparecer en la parte superior de la pantalla.

Paso 10: Probar tu código (5 a 10 minutos)

Pongamos a prueba lo que hemos escrito hasta ahora para asegurarnos de que nuestro programa funcione de la manera que queremos. Haz clic en el botón Reproducir para ejecutar tu bordado. Debería:

- Imprimir el valor de distancia a la consola.
- Volver a dibujar el meteorito hasta la parte superior del lienzo cuando el capturador y el meteorito se crucen.
- Volver a dibujar el meteorito hasta la parte superior del lienzo cuando el meteorito y la parte inferior del lienzo se crucen.



Haz clic [aquí](#) para ejecutar el bordado del ejemplo.

¿No funciona como deseas? Prueba estos consejos de depuración:

- ¿Tu código está dentro de las llaves correctas?
- ¿Hay punto y coma al final de cada línea de código?
- ¿Escribiste correctamente los nombres de la función y de la variable?
- ¿Tus funciones están en la ubicación y secuencia correctas? ¡Recuerda que el orden es importante en el flujo del programa!
- ¿Son correctos los parámetros de tu función `dist()`?
- Imprime los valores en la consola con `print()` y luego marca cualquier afirmación `if`.



No olvides verificar tu código con la Guía de referencia en la página 6.

Paso 11: Verificar la comprensión

Digamos que deseas “atrapar” el meteorito cuando el captador apenas toca el borde exterior del meteorito. ¿Aumentarías o disminuirías el valor en la expresión de tu primer enunciado de sentencia condicional?



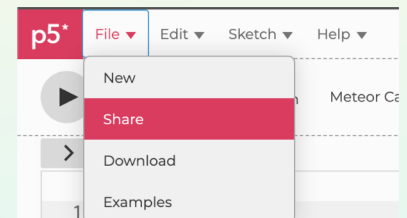
No olvides verificar tu código con la Guía de referencia en la página 7.

Paso 12: Compartir tu proyecto de Girls Who Code en casa (5 minutos)

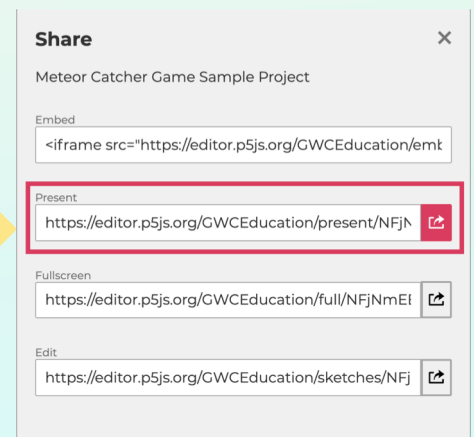
Nos encantaría ver tu trabajo y sabemos que a otros también les gustaría. ¡Comparte tu juego con nosotros! No olvides etiquetar [@girlswhocode](#) [#codefromhome](#), ¡y quizá te destaquemos en nuestra cuenta!

Sigue estos pasos para compartir tu proyecto:

- Primero guarda tu proyecto.
- En el menú **Archivo**, elige la opción **Compartir** en el menú desplegable.
- Elige la opción **Vínculo** en el menú desplegable.
- Copie el enlace **Presente** y pégalo donde quieras compartirlo.



Enlace al proyecto



¡Espera más proyectos de Girls Who Code en casa!

