**Step 4**

Before you can begin building out the functionality for the game, you will need to set up the ability to add 2D graphics.

The *Canvas API* can be used to create graphics in games using JavaScript and the HTML canvas element.

You will need to use the getContext method which will provide the context for where the graphics will be rendered.

Example Code

canvas.getContext("2d");

Assign that getContext method to a const variable called ctx.

**Step 5**

The canvas element has a width property which is a positive number that represents the width of the canvas.

Example Code

canvas.width

Below your const declarations, append the width property to the canvas variable.

# Step 6

The innerWidth property is a number that represents the interior width of the browser window.

Assign innerWidth to canvas.width.

# Step 7

The innerHeight property is a number that represents the interior height of the browser window.

Below your canvas.width, append the height property to the canvas variable and assign it innerHeight.

# Step 8

In your platformer game, the main player will need to jump between the different platforms. When the player jumps, you will need to apply gravity to bring them back down.

Create a new const variable called gravity and assign it the number 0.5.

# Step 9

In the game, the player will have the opportunity to cross different checkpoints. You will need to keep track of the status for the checkpoint collision detection.

Use let to create a new variable called isCheckpointCollisionDetectionActive and assign it the value of true.

# Step 10

As you are designing the game, you will need to make sure that the size of the elements in the game are responsive and adapt to different screen sizes.

Start by creating an arrow function called proportionalSize that takes in a size parameter.

# Step 11

The width and the height of the main player, platforms and checkpoints will be proportional sized relative to the innerHeight of the browser screen. The goal is to make the game responsive and visually consistent across different screen sizes.

Inside your proportionalSize function, you will need to return a ternary that checks if innerHeight is less than 500. If so, return Math.ceil((size / 500) \* innerHeight), otherwise return size.

# Step 11

The width and the height of the main player, platforms and checkpoints will be proportional sized relative to the innerHeight of the browser screen. The goal is to make the game responsive and visually consistent across different screen sizes.

Inside your proportionalSize function, you will need to return a ternary that checks if innerHeight is less than 500. If so, return Math.ceil((size / 500) \* innerHeight), otherwise return size.

# Step 15

Inside your position object, add a new key called x with a value of proportionalSize(10). After that, add another key called y with a value of proportionalSize(400).

You need to use the proportionalSize function here to make sure that the player's position is always proportional to the screen size. This is important because you want the player to be able to move around the screen regardless of the screen size.

# Step 18

The next step is to create a draw() method, which will be responsible for creating the player's width, height, position, and fill color.

Below your constructor, create an empty draw() method.

**Step 20**

Below your ctx.fillStyle, you need to create the player's shape by calling the fillRect() method on the ctx object which you instantiated earlier.

Example Code

fillRect(x, y, width, height)

Inside the fillRect() method add the this.position.x, this.position.y, this.width and this.height values.

# Step 33

For the last condition, you will need to check if the player's x position has exceeded the right edge of the canvas. If it has, you will need to set the player's x position to the maximum value so the player does not accidentally go off screen to the right.

Inside your update method, create an if statement that checks if this.position.x >= canvas.width - this.width \* 2.

# Step 41

The requestAnimationFrame() web API, takes in a callback and is used to update the animation on the screen. The animate function will be responsible for updating the player's position and continually drawing it on the canvas.

Inside the animate function, call the requestAnimationFrame() API and pass animate as the argument.

# Step 42

As the player moves through the game, you will need to clear the canvas before rendering the next frame of the animation.

You can use the clearRect() Web API to accomplish this. It takes in an x, y, width, and height arguments.

Below your requestAnimationFrame, call the clearRect() method on the ctx variable and pass in 0, 0, canvas.width, canvas.height as the arguments.

# Step 44

To manage the player's movement in the game, you will need to monitor when the left and right arrow keys are pressed.

Create a new const variable called keys and assign it an empty object.

# Step 46

The next step is to add the logic for increasing or decreasing a player's velocity based on if they move to the left or right of the screen.

Inside the animate function, create an if statement where the condition checks if the right key was pressed and the player's x position is less than proportionalSize(400).

You need to use the proportionalSize function here to make sure the player's x position is always proportional to the screen size.

# Step 51

In the game, the player will interact with different checkpoints. If the isCheckpointCollisionDetectionActive is false, then you will need to stop the player's movements on the x and y axes.

Start by creating an if statement where the condition checks if the isCheckpointCollisionDetectionActive is false.

Remember that you can use the ! operator to check if the variable is false.

**Step 59**

Now it is time to add the event listeners that will be responsible for calling the movePlayer function.

Start by adding an addEventListener to the global window object.

For the arguments, pass in the keydown event and an arrow function that uses the destructuring assignment to get the key property from the event object in the event listener parameter.

Here is the syntax for using the destructuring assignment in the parameter list of the arrow function:

Example Code

btn.addEventListener('click', ({ target }) => {

console.log(target);

});



Check Your Code (Ctrl + Enter)

Sorry, your code does not pass. Keep trying.

2. Your event listener should listen for the keydown event.

Reset



Navigated to Step 59

No results found for ','

Cursor added: line 104 column 28

# Step 61

Add another addEventListener to the global window object and pass in the keyup event and use destructuring to pass in the key property from the event.

# Step 63

Before you can start moving your player across the screen, you will need to use the animate function.

Inside the startGame function, delete player.draw() and call the animate function.

Click the Start Game button and use the left and right arrow keys to move the player across the screen. You can also use the spacebar or the up arrow key to jump up.

**Step 66**

When working with objects where the property name and value are the same, you can use the shorthand property name syntax. This syntax allows you to omit the property value if it is the same as the property name.

Example Code

// using shorthand property name syntax

obj = {

a, b, c

}

The following code is the same as:

Example Code

obj = {

a: a,

b: b,

c: c

}

Inside the constructor, add this.position and assign it an object with the x and y coordinates. Make sure to use the shorthand property syntax .