

Mushroom Makers – Lesson 3: Moving Characters & Simple Collision Detection

Ready to add movement to your text story? Let's take your RPG to the next level by controlling a character on the screen using your keyboard! Get ready for more JavaScript.

Step 1: Update Your HTML (*index.html*)

Add two images to your folder along with your existing project files (so your folder should include *index.html*, *script.js*, *style.css*, *image1* and *image2*). One image will be the player. The other will be treasure. Then add these images to your *index.html* inside a game container area (game-area). This should be placed between `<body>` and `</body>`:

```
<div id="game-area">

  <!-- Player Character -->
  

  <!-- Target / Treasure Item -->
  

</div>
<p id="game-status">Use Arrow Keys to move your character to the
treasure!</p>
```

Note: Change the words in `src="my_image.jpg"` to whatever your image files are called.

Step 2: Add some CSS (*style.css*)

In your *style.css* file, add the following styles to the new html elements you just created:

```
/* Character Movement Styles */
#game-area {
  position: relative;
  width: 500px;
  height: 300px;
  background-color: #0f172a;
  border: 2px solid #38bdf8;
  overflow: hidden;
}
```

```
#player {  
position: absolute;  
top: 100px;  
left: 100px;  
width: 40px;  
height: 40px;  
}  
#target {  
position: absolute;  
top: 200px;  
left: 300px;  
width: 40px;  
height: 40px;  
}
```

Key Concept: Using *position: absolute* on the images allows us to move them precise distances using X (left) and Y (top) pixel values inside the game-area.

Step 3: Add Movement Logic (*script.js*)

To make things move in the browser screen, we need to use JavaScript. We will define the X and Y coordinates for the player and create **constants** (speed and the player object).

const (Constant):

const is a keyword used to create a **fixed variable** - a container whose value **cannot change** while the game is running.

Think of it like a name label permanently stuck onto a box. We use **const** for things in your game that stay the same the whole time, like *const speed = 10;* (the player's movement speed) or

const player = document.getElementById('player');

That is a reference to an element called 'player' on the screen.

- **let** is for values that **will change** (like your score or *playerX* position).
- **const** is for values that **stay constant** (like game rules or speed limits).

Add X and Y position variables to track your player's coordinates, and 'listen' (*addEventListener*) for keyboard presses to your *script.js* file:

```
// 1. Define player coordinates and speed
let playerX = 100; // Left position in pixels
let playerY = 100; // Top position in pixels
const speed = 10; // How many pixels to move per keypress

// 2. Listen for arrow key presses – we use addEventListener to do this
window.addEventListener('keydown', function(event) {
  const player = document.getElementById('player');

  // Simple math to change coordinates based on key pressed
  if (event.key === 'ArrowUp') {
    playerY -= speed; // Subtract Y to move UP
  } else if (event.key === 'ArrowDown') {
    playerY += speed; // Add Y to move DOWN
  } else if (event.key === 'ArrowLeft') {
    playerX -= speed; // Subtract X to move LEFT
  } else if (event.key === 'ArrowRight') {
    playerX += speed; // Add X to move RIGHT
  }

  // Update the image position on screen
  player.style.left = playerX + 'px';
  player.style.top = playerY + 'px';

  // Check if player hit the target
  checkCollision();
});
```

Open your *index.html* in the browser and have a look.

Step 4: Simple "Same Location" Collision Detection

There is no purpose having a target if we cannot detect collision. Let's add a collision that happens when the distance between the Player and Target is smaller than their size (e.g., within 30 pixels of each other). Add the code below - a simple collision detection function - to your *script.js*.

```
// 4. The checkCollision function to detect the collision

function checkCollision() {
  const targetX = 300;

  // Target's left position
  const targetY = 200;
  // Target's top position
  // Calculate distance between Player and Target
  const xDistance = Math.abs(playerX - targetX);
  const yDistance = Math.abs(playerY - targetY);

  // If player is within 30px on both X and Y axes:
  if (xDistance < 30 && yDistance < 30) {
    document.getElementById('game-status').textContent = "🏴‍☠️ You collected the TREASURE!";
    document.getElementById('target').style.display = 'none'; // The style.display is another way to do CSS which here hides the target
  }
}
```

Extension 1: Adding a Score Counter 🏆

Now that your player can collect the treasure, let's keep track of how many items the player collects by adding a Score!

First, update HTML (*index.html*) by adding a score display right above or below your game arena:

```
<h2 id="score-board">Score: 0</h2>
```

Of course, that won't do anything until we have added some JavaScript to make it active. We need a variable to hold the score. We use ***let score = 0;*** (not *const*) at the beginning because the score **changes** every time you collect an item! We also need to add logic to add points when a collision happens. Add a *score* variable before your *checkCollision* function (you can add it to the top of your *script.js* with the other variables):

```
// Add a score tracker variable
let score = 0;

// Must go BEFORE this:
// 4. The checkCollision function to detect the collision
function checkCollision() { ...
```

The score variable exists but doesn't do anything. We need to add an update to the 'If player is within 30px on both X and Y axes' conditional that tells us what happens if the player collides with the treasure. Replace your existing 'If player is within 30px on both X and Y axes' conditional with this code (note we are also adding code that will move the treasure when a collision is detected):

```
// If player is within 30px on both X and Y axes:
if (xDistance < 30 && yDistance < 30) {

  // 1. Increase the score by 1
  score = score + 1;

  // 2. Update the HTML score display by identifying the html element and updating its content
  document.getElementById('score-board').textContent = 'Score: ' + score;

  // 3. Move the target to a new position so player can catch it again!
  const target = document.getElementById('target'); target.style.left = '50px'; target.style.top = '220px';
}
```

Open your *html* in the browser and try. You should see the score counter increases when you collect the treasure. However, you will probably find that the 'game' is very limited because the treasure moves to the new spot and then becomes inactive. It would be better if the treasure moved to a new location every time it is hit. We can do this by using JavaScript's built-in **Math.random()** function.

By calculating a random left (x) and top (y) position within the arena boundaries, the treasure spawns in a new spot every single time the player touches it!

Extension 2: Spawning the Treasure in a Random Position! 🎲

Instead of sending the treasure to a fixed spot like 50px, 220px, we can make it jump to a **random location** inside the 500x300 arena!

First, we need to give the treasure X and Y variables using *let* because they will change. Add this near the top of your *script.js* where you define the other variables:

```
// Track target's dynamic position with variables
let targetX = 300;
let targetY = 200;
```

We will change the '4. The checkCollision function to detect the collision' block again (study the code in blocks B and C so you understand the changes):

```
// 4. The checkCollision function to detect the collision

function checkCollision() {
// Calculate distance between Player and Target
const xDistance = Math.abs(playerX - targetX);
const yDistance = Math.abs(playerY - targetY);

// The conditional logic
if (xDistance < 30 && yDistance < 30) {

// A. Add 1 point to score and update screen
score = score + 1;
document.getElementById('score-board').textContent = 'Score: ' + score;

// B. Generate new random coordinates inside arena (500x300)
// We leave a 40px margin so the target stays inside the box!
targetX = Math.floor(Math.random() * (500 - 40));
targetY = Math.floor(Math.random() * (300 - 40));

// C. Move the target image to the new random spot
const target = document.getElementById('target');
target.style.left = targetX + 'px';
target.style.top = targetY + 'px';
}
}
```

Note: As the code becomes more complex, be careful not to miss any of these `{}`. You can count to check that every *opening* `{` curly brace has a *closing* `}` curly brace. Any that open but do not close will break the code.

Try it! Now you have the basis of a game. You also have the code you need to extend this simple game.

Extension Challenges for Fast Finishers!

1. **Player 2 Controls:** Add a second player image (player2) controlled using event.key === 'w', 'a', 's', and 'd'. Create separate player2X and player2Y variables to track Player 2's movement!
2. **Moving Target:** Can you make the treasure move in a random direction when a player hits a key?

3. **Win:** Can you add a conditional within the '4. The checkCollision function to detect the collision' conditional that will check if score=10 and fire the "You WIN!" message? You could then hide the treasure (look back at how we did that in step 4).

Helpful conditional logic:

```
if (score === 10) {  
  document.getElementById('game-status').textContent = "🏆 You Win the Game!"; }  
if (score >= 10) {  
  document.getElementById('game-status').textContent = "🏆 You Win the Game!"; }
```

4. **Add a Baddie:** Can you add an obstacle that, when your player hits it, the player disappears and you get a "You LOSE!" message? You could extend this by adding lives (study how score is displayed and increased on collision and create a lives display with a variable that decreases on collision).