3.2.7 Battle the Combatants Until Defeated Using a While Loop

The next objective is to fight each robot multiple times and then defeat them. Currently we can only fight each robot once. So how do we go about fighting each robot until someone wins? It sounds like we need another loop to continue the fighting.

Earlier we determined that to defeat a robot, we need to fight it until its health points reduce to zero or less. Conversely, we can say that we'll fight the enemy robot while it's alive. We can translate this into code by using the following conditional:

```
if (enemyHealth > 0) // if the enemy robot has health points, continue
```

Luckily, we can introduce another type of control flow statement that loops or repeatedly executes a statement while a condition remains true. This is called the `while` loop. Like the `for` loop, the `while` loop repeatedly executes a code block if a condition remains true.

The code block is the statement or statements contained within the curly braces:

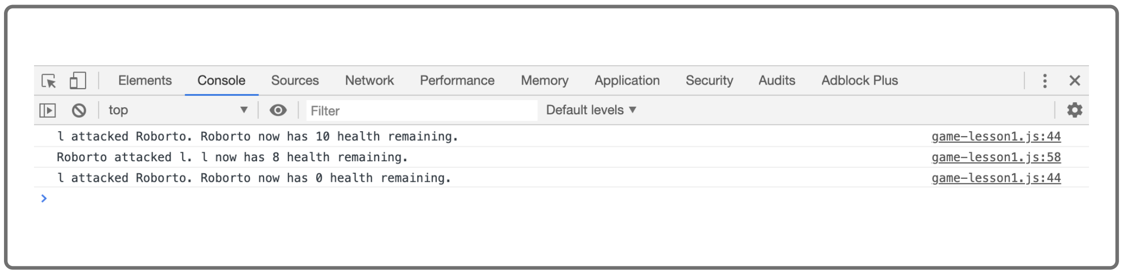
```
while([Condition]) {
  statement
}
```

In a `while` statement, if the condition evaluates to true, the statement executes. Then the condition is reevaluated on the loop's next pass, and then so on. Let's write our code within the `fight()` function to look like the following:

```
var fight = function(enemyName) {
  // repeat and execute as long as the enemy robot is alive
  while(enemyHealth > 0) {
    // place fight function code block here . . .
  }
```

Remove all the code from within the `fight()` function and move it to within the `while` loop. Now when we call the `fight()` function, our robot will fight the enemy robot again until defeated. We can also remove the `alert("Welcome to Robot Gladiators!")` statement from the `fight()` function. This message will be unnecessary at the beginning of every fight, so let's remove it for now.

Save `game.js` and reload `index.html` in Chrome. Check out the console:



```
l attacked Roborto. Roborto now has 10 health remaining.     game-lesson1.js:44
Roborto attacked l. l now has 8 health remaining.            game-lesson1.js:58
l attacked Roborto. Roborto now has 0 health remaining.      game-lesson1.js:44
>
```

This is partially what we want. The `while` loop correctly loops until the enemy robot is defeated however why doesn't another enemy robot appear as it did before for the next round of battle?

We can debug the code by asking a few questions:

- Is the `for` loop iterating through the `enemyNames` array correctly?
- Is the `while` loop functioning as expected?
- Is the `fight()` function accepting additional enemy robots?

Seeing some of the variable's values will help us to resolve these questions and identify what part of the code we need to fix. We could strategically place `console.log()` statements in critical spots in the program, such as in the `for` loop or we could learn a new way to debug the program using Chrome DevTool's Debugger. Let's proceed with the latter and learn a new concept.

Use the Debugger

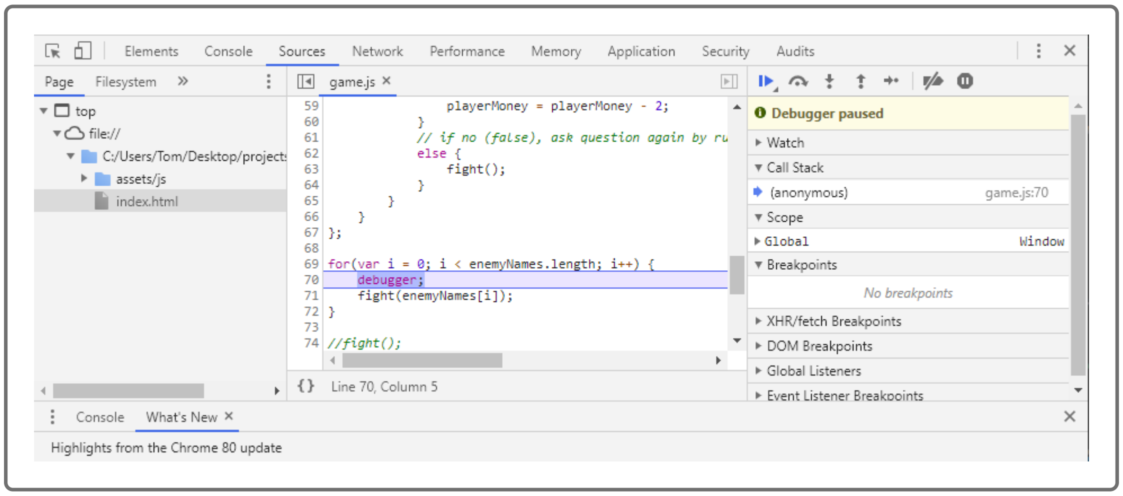
Let's use a different technique this time to reveal the variable values. The `for` loop is an excellent place to use a tool called the **debugger** because this is a focal point where the program stops progressing.

Type the following expression in the `game.js` file before the `fight()` function call inside the block of the `for` loop:

```
for (var i = 0; i < enemyNames.length; i++) {
  debugger;
  // call fight function with enemy robot
  fight(enemyNames[i]);
}
```

This will inform us whether the `for` loop actually iterates through the `enemyNames` array.

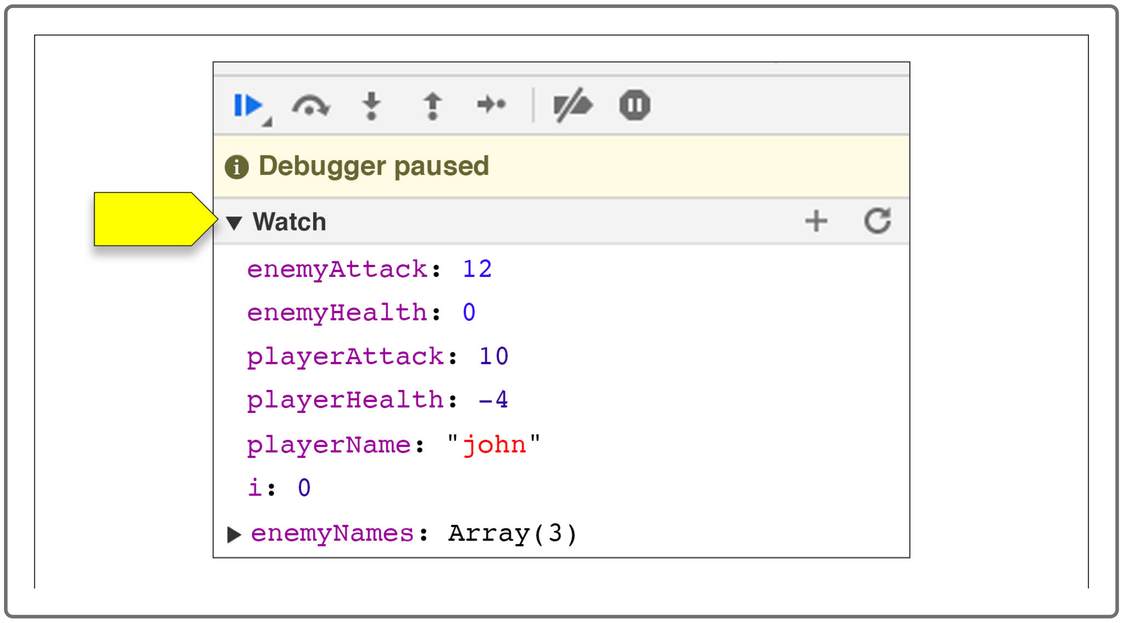
Save `game.js` and reload `index.html` in Chrome. Open Chrome DevTools to see the following in the Sources tab of the browser:



```
                playerMoney = playerMoney - 2;
            }
            // if no (false), ask question again by ru
            else {
                fight();
            }
        }
    }
};

for(var i = 0; i < enemyNames.length; i++) {
    debugger;
    fight(enemyNames[i]);
}

//fight();
```

Notice that the script with the `debugger;` statement highlighted appears in the **Code Editor pane**. To the left, the File Navigator pane displays the folder tree in the Page tab. To the right is the JavaScript Debugging pane. The current status notes that the debugger has been paused; the program will remain frozen until we click the resume button.

Our main focus should be on the debugger tool panel on the right. Let's expand the Watch option as illustrated below:



```
enemyAttack: 12
enemyHealth: 0
playerAttack: 10
playerHealth: -4
playerName: "john"
i: 0
▶ enemyNames: Array(3)
```

The Watch tool allows us to track the state of many variables at once. We can see a snapshot at the moment the program reaches the `debugger` statement. We will see many variables already being tracked. To add variables to the Watch panel, add the variable name by selecting the `+` button and typing the variable name.

If you don't see the `enemyNames` array or the `i`, add them to your Watch panel now.

The `i` value in the first iteration of the `for` loop is zero, so the first element of the array is accessed. Because we placed the `debugger` statement before the `fight()` function call, we've found the point in the program just as the robots go into battle.

HIDE PRO TIP

For shorter battles and faster results, we can manipulate the health points for the player and the enemy in the JavaScript file.

Because the robots' health has a major impact on the battle, we should watch how these values change. It appears that the `enemyHealth` value is currently at zero from the image above.

Debugger Controller

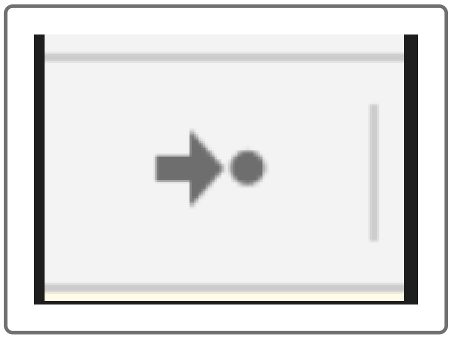
The debugger controls are located at the top of the Debugger panel in the Source panel.

DEEP DIVE

DEEP DIVE

Other controls include resuming the program's execution and stepping into, out of, and over functions. For a closer look at these controls and other tools like breakpoints in the Chrome DevTools debugger, take a look at **Chrome's DevTools JavaScript debugger tutorial. (https://developers.google.com/web/tools/chrome-devtools/javascript/)**

Place your cursor over each control to find a quick explanation. We'll focus on the step control, which is located at the top of the debugger looks like this:

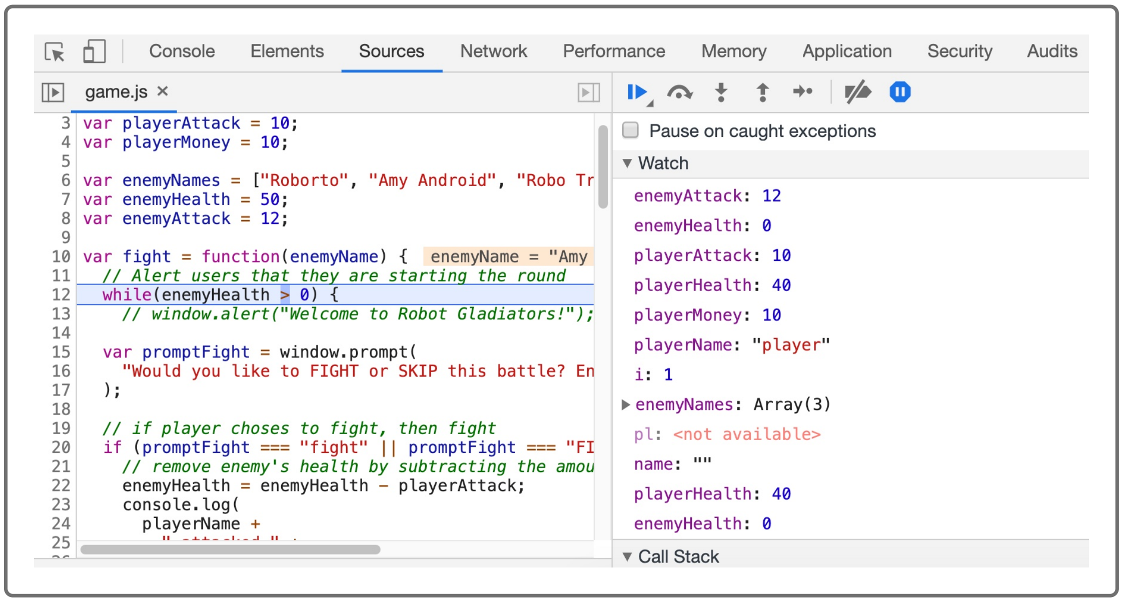


Click on the step button and progress slowly through the program noting value changes and the evaluation of each control flow statement.

It is important to continue to select the fight option when we come to the fight-or-skip prompt so we can observe the `enemyHealth` value change.

In the next step, the `i` value increments by one, due to the `for` statement increment expression `i++`. Next, the `i` conditional statement is evaluated in the `for` loop `i < enemyNames.length`. This conditional statement is true because `1 < 3`, so we proceed into the `fight()` function. Once inside the `fight()` function again we'll step to the `while` loop. But once there, instead of progressing with the new robot enemy, we're immediately kicked out of the `while` loop and we return to the `for` loop. What gives?

Take a look at the following image and pay attention to the values and the conditional currently being evaluated:



```
var playerAttack = 10;
var playerMoney = 10;

var enemyNames = ["Roborto", "Amy Android", "Robo Tr
var enemyHealth = 50;
var enemyAttack = 12;

var fight = function(enemyName) {  enemyName = "Amy
  // Alert users that they are starting the round
  while(enemyHealth > 0) {
    // window.alert("Welcome to Robot Gladiators!");

  var promptFight = window.prompt(
    "Would you like to FIGHT or SKIP this battle? En
  );

  // if player choses to fight, then fight
  if (promptFight === "fight" || promptFight === "FI
    // remove enemy's health by subtracting the amou
    enemyHealth = enemyHealth - playerAttack;
    console.log(
      playerName +
```

```
enemyAttack: 12
enemyHealth: 0
playerAttack: 10
playerHealth: 40
playerMoney: 10
playerName: "player"
i: 1
▶ enemyNames: Array(3)
pl: <not available>
name: ""
playerHealth: 40
enemyHealth: 0
```

We didn't satisfy the `while` loop's condition that `enemyHealth > 0` because the current value of `enemyHealth` is zero.

Ah ha! So that's why the player robot won't fight the other enemy robots.

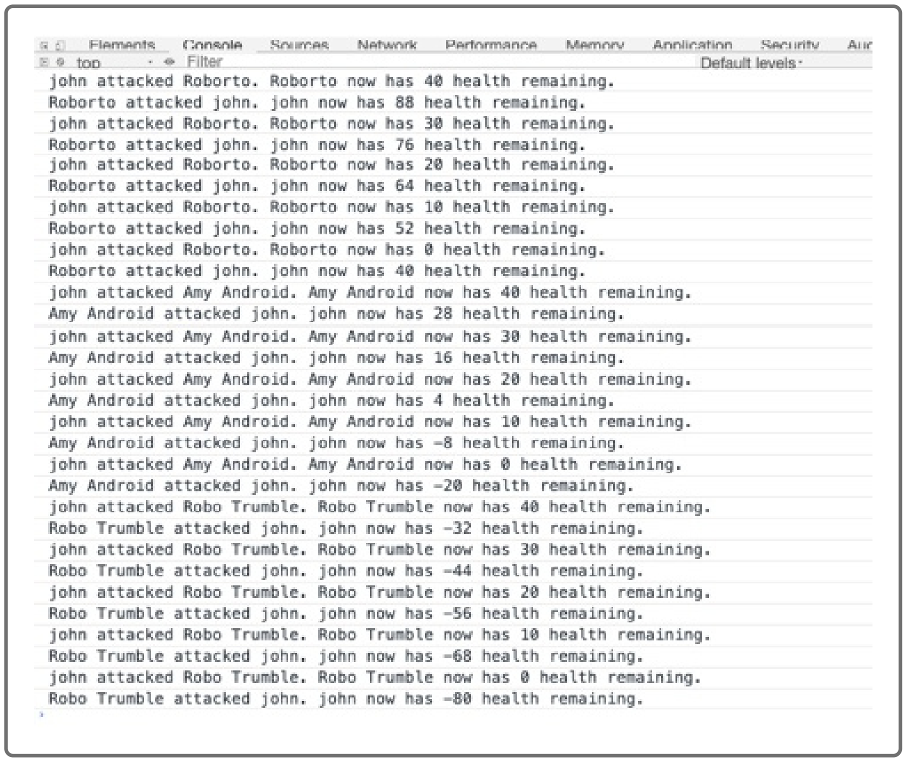
Although initially the `enemyHealth` variable is set at 50, we never reset this value when another robot appears. Currently attacking one robot effectively attacks all the enemy robots: when we've defeated one, we've defeated them all. That doesn't mirror how life actually works so let's adjust our code and adhere to a more realistic scenario.

To allow each new robot to start at full strength, reset the `enemyHealth` value before each new battle round. Let's reset the enemy robot's health by assigning it right before the `fight()` function call inside the `for` loop. In addition, let's assign the element in the enemy robot array to a better name that's more semantic than `enemyNames[i]` to store the current enemy robot. Let's choose `pickedEnemyName` to store the current name in the `enemyNames` array.

Make sure your `for` loop looks like this:

```
for (var i = 0; i < enemyNames.length; i++) {
    var pickedEnemyName = enemyNames[i];
    enemyHealth = 50;
    fight(pickedEnemyName);
}
```

Save your work and reload `index.html` in the browser to see if our changes have solved the problems. We have removed the `debugger` statement for now, but keep this in mind if you run into any trouble. We should see the following in the console:



```
john attacked Roborto. Roborto now has 40 health remaining.
Roborto attacked john. john now has 88 health remaining.
john attacked Roborto. Roborto now has 30 health remaining.
Roborto attacked john. john now has 76 health remaining.
john attacked Roborto. Roborto now has 20 health remaining.
Roborto attacked john. john now has 64 health remaining.
john attacked Roborto. Roborto now has 10 health remaining.
Roborto attacked john. john now has 52 health remaining.
john attacked Roborto. Roborto now has 0 health remaining.
Roborto attacked john. john now has 40 health remaining.
john attacked Amy Android. Amy Android now has 40 health remaining.
Amy Android attacked john. john now has 28 health remaining.
john attacked Amy Android. Amy Android now has 30 health remaining.
Amy Android attacked john. john now has 16 health remaining.
john attacked Amy Android. Amy Android now has 20 health remaining.
Amy Android attacked john. john now has 4 health remaining.
john attacked Amy Android. Amy Android now has 10 health remaining.
Amy Android attacked john. john now has -8 health remaining.
john attacked Amy Android. Amy Android now has 0 health remaining.
Amy Android attacked john. john now has -20 health remaining.
john attacked Robo Trumble. Robo Trumble now has 40 health remaining.
Robo Trumble attacked john. john now has -32 health remaining.
john attacked Robo Trumble. Robo Trumble now has 30 health remaining.
Robo Trumble attacked john. john now has -44 health remaining.
john attacked Robo Trumble. Robo Trumble now has 20 health remaining.
Robo Trumble attacked john. john now has -56 health remaining.
john attacked Robo Trumble. Robo Trumble now has 10 health remaining.
Robo Trumble attacked john. john now has -68 health remaining.
john attacked Robo Trumble. Robo Trumble now has 0 health remaining.
Robo Trumble attacked john. john now has -80 health remaining.
```

Now that looks more like a battle! The rounds are much longer and more extensive. Excellent work! You can see the fight with each robot and the reduction in health points as the battle progresses. After the player robot defeats an enemy robot, a new enemy robot joins the battle at full health, just as planned.

Pause and preserve this work in GitHub before you proceed.

Break Out of the Loop

We're not done yet. The console reveals some oddities that don't resemble an actual battle:

- The player robot's health drops into negative numbers.
- Enemy robots can still attack after being defeated.
- The "skip" option doesn't function correctly.

Let's see what we can do to correct these.

Adjust the "LOSE" Game State

To prevent the player robot from attacking after its health is depleted, we need to review one of our objectives:

```
// Game States
// "LOSE" - Player robot's health is zero or less
```

Once the player robot has been defeated, they should no longer be able to fight. To accomplish this, we need to exit the `while` loop in the `fight()` function.

In the last lesson, we created a condition to check the status of the player robot's health inside the `fight()` function:

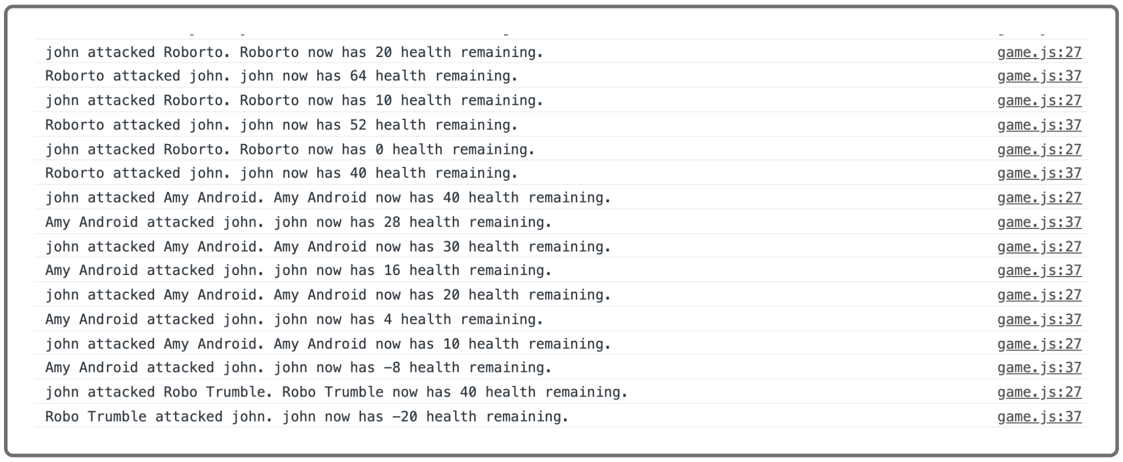
```
if (playerHealth <= 0) {
  window.alert(playerName + " has died!");
}
```

Currently a `window.alert()` notifies the player if the player robot has been defeated. We need to add a way to stop fighting as well after meeting this condition.

Luckily we can use a keyword in JavaScript called `break`. The `break` keyword allows us to exit the current loop. Let's add this beneath the `window.alert` and within the `if` statement so that it looks like the following:

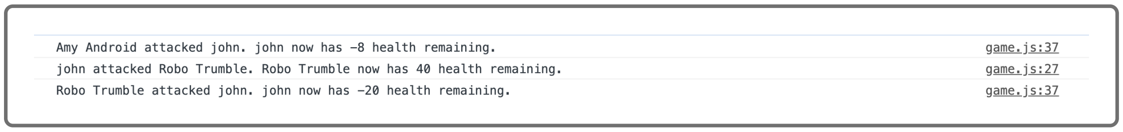
```
if (playerHealth <= 0) {
  window.alert(playerName + " has died!");
  break;
}
```

Now let's run the game and see what happens. The console should look like this:



```
john attacked Roborto. Roborto now has 20 health remaining.                game.js:27
Roborto attacked john. john now has 64 health remaining.                   game.js:37
john attacked Roborto. Roborto now has 10 health remaining.                game.js:27
Roborto attacked john. john now has 52 health remaining.                   game.js:37
john attacked Roborto. Roborto now has 0 health remaining.                 game.js:27
Roborto attacked john. john now has 40 health remaining.                   game.js:37
john attacked Amy Android. Amy Android now has 40 health remaining.        game.js:27
Amy Android attacked john. john now has 28 health remaining.               game.js:37
john attacked Amy Android. Amy Android now has 30 health remaining.        game.js:27
Amy Android attacked john. john now has 16 health remaining.               game.js:37
john attacked Amy Android. Amy Android now has 20 health remaining.        game.js:27
Amy Android attacked john. john now has 4 health remaining.                game.js:37
john attacked Amy Android. Amy Android now has 10 health remaining.        game.js:27
Amy Android attacked john. john now has -8 health remaining.               game.js:37
john attacked Robo Trumble. Robo Trumble now has 40 health remaining.      game.js:27
Robo Trumble attacked john. john now has -20 health remaining.             game.js:37
```

Although we've revised our code to exit the `while` loop when the player robot has lost all its health, the last few lines of the console show that it's still fighting.



```
Amy Android attacked john. john now has -8 health remaining.               game.js:37
john attacked Robo Trumble. Robo Trumble now has 40 health remaining.      game.js:27
Robo Trumble attacked john. john now has -20 health remaining.             game.js:37
```

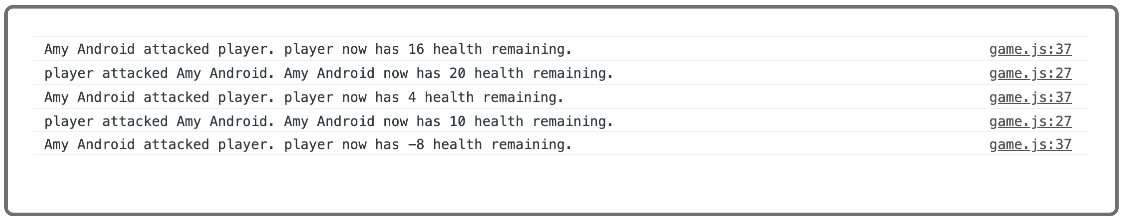
The player robot has reentered the while loop to fight the last robot, despite a health level of zero. Why? Because our while loop repeats if the *enemy* health is greater than 0, but it doesn't consider our own player's health.

Let's add a condition to the `while` loop to check the player robot's health, too:

```
while (enemyHealth  > 0 && playerHealth > 0)
```

With the logical AND operator, `&&`, we can have the `while` loop set two conditions that must both resolve to true to execute the fighting rounds in the `while` loop. The AND operator differs from the OR operator (`||`), which must have either condition evaluate to true. The AND operator must satisfy *both* conditions to execute the block.

The conditionals state that the enemy and the player robot must both have health in order to fight:



```
Amy Android attacked player. player now has 16 health remaining.        game.js:37
player attacked Amy Android. Amy Android now has 20 health remaining.   game.js:27
Amy Android attacked player. player now has 4 health remaining.         game.js:37
player attacked Amy Android. Amy Android now has 10 health remaining.   game.js:27
Amy Android attacked player. player now has -8 health remaining.        game.js:37
```

Nice job—now the player robot can no longer fight after being defeated!

No More Cheap Shots

Let's move on to the next oddity: the enemy robot can still fight after being defeated. Like the last step, we need a condition to allow us to break out of the `while` loop in the `fight()` function. Let's consider some pseudocode that describes this operation:

```
// if the enemy robot's health is zero or less, exit from the fight lo
```

PAUSE

Take a moment and try to translate that into JavaScript.

```
if (enemyHealth <= 0) {
 break;
}
```

Hide Answer

We have a condition from the last lesson that checks for the enemy robot's health. We just need to add a `break` to exit the fight loop and prevent the robot from fighting after it's defeated. Let's add this `break` statement to the condition's block so that it looks like this:

```
if (enemyHealth <= 0) {
  window.alert(enemyName + " has died!");
  break;
}
```

It's time to test the code and see if we get the expected result. Let's run the game and watch the browser console.

When you get to the end of the battle with Roberto and begin your battle with Amy Android, it should look like this:

```
Roborto attacked player. player now has 52 health remaining.
player attacked Roborto. Roborto now has 0 health remaining.
player attacked Amy Android. Amy Android now has 40 health remaining.
```

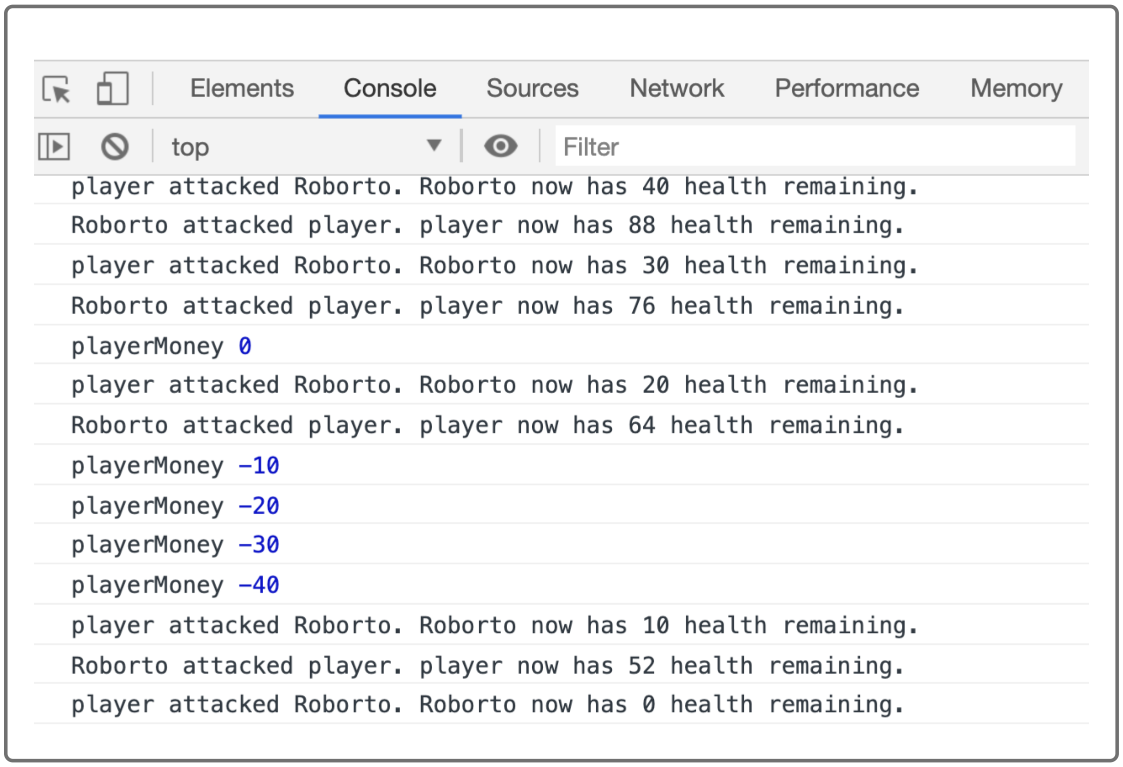
The `while` loop exits after the enemy robot has been defeated, and the next robot combatant confronts us. No longer are we subject to cheap shots from defeated enemy robots. Much better!

Give Peace a Chance

The last oddity that we need to fix is that the skip function isn't working quite right. Let's make sure that the player loses the correct amount of money when they skip. Update the `if (confirmSkip) {...}` code block to deduct 10 from `playerMoney`, and also log the new value:

```
if (confirmSkip) {
  window.alert(playerName + " has decided to skip this fight. Goodbye!
  // subtract money from playerMoney for skipping
  playerMoney = playerMoney - 10;
  console.log("playerMoney", playerMoney);
}
```

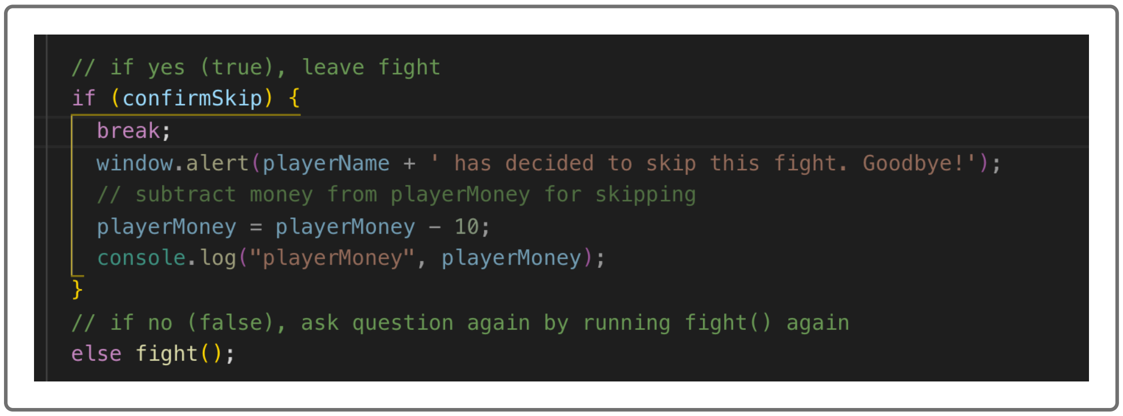
Save `game.js` and reload `index.html` in the browser. Fight for a couple rounds and then enter skip a round. The console shows that although we lose money for choosing to skip, we don't actually skip but resume fighting the same enemy robot:



```
player attacked Roborto. Roborto now has 40 health remaining.
Roborto attacked player. player now has 88 health remaining.
player attacked Roborto. Roborto now has 30 health remaining.
Roborto attacked player. player now has 76 health remaining.
playerMoney 0
player attacked Roborto. Roborto now has 20 health remaining.
Roborto attacked player. player now has 64 health remaining.
playerMoney -10
playerMoney -20
playerMoney -30
playerMoney -40
player attacked Roborto. Roborto now has 10 health remaining.
Roborto attacked player. player now has 52 health remaining.
player attacked Roborto. Roborto now has 0 health remaining.
```

Just as in the previous steps, we need to break out of the `while` loop to quit fighting the enemy robot. So where in the conditional code block will the `break` statement go?

If we place the `break` statement before the other statements in the conditional code block, the VS Code editor actually dims the following lines of code:



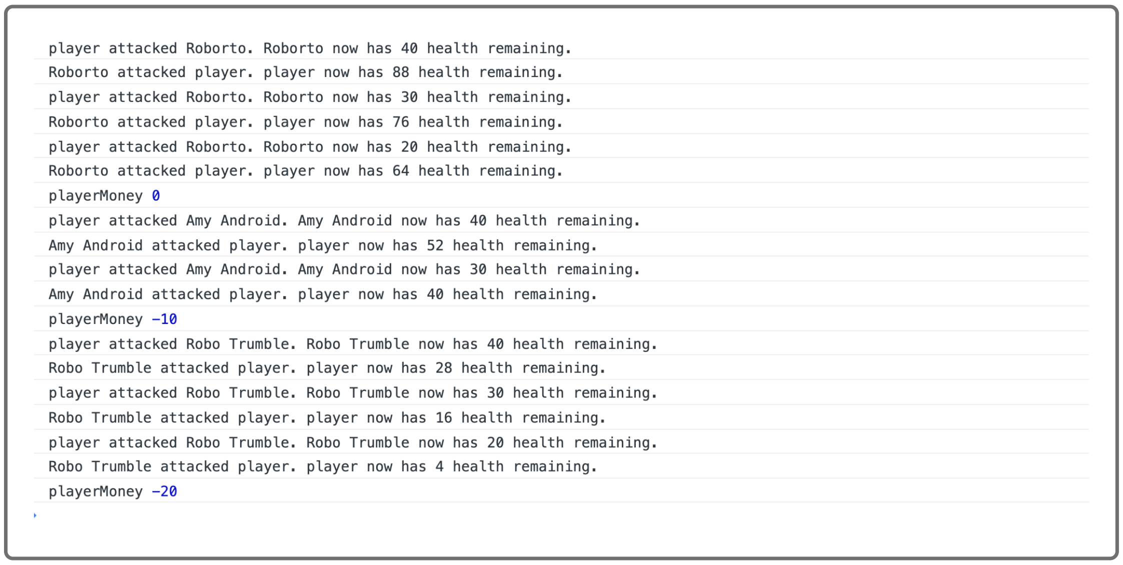
```
// if yes (true), leave fight
if (confirmSkip) {
  break;
  window.alert(playerName + ' has decided to skip this fight. Goodbye!');
  // subtract money from playerMoney for skipping
  playerMoney = playerMoney - 10;
  console.log("playerMoney", playerMoney);
}
// if no (false), ask question again by running fight() again
else fight();
```

VS Code is letting us know that the code beneath the `break;` statement will never run. The `break` statement will immediately step out of the loop, so any subsequent lines of code will never execute. That's why we must always position `break` statements at the bottom of the conditional code block:

```
if (confirmSkip) {
   window.alert(playerName + " has decided to skip this fight. Goodbye
   // subtract money from playerMoney for skipping
   playerMoney = playerMoney - 10;
   console.log("playerMoney", playerMoney);
   break;
 }
```

Let's run the game and see if we fixed the skipping issue. Run the game from the top again, and this time skip battling from time to time during play by entering "skip" when prompted to fight or skip.

The console should look something like this:



```
player attacked Roborto. Roborto now has 40 health remaining.
Roborto attacked player. player now has 88 health remaining.
player attacked Roborto. Roborto now has 30 health remaining.
Roborto attacked player. player now has 76 health remaining.
player attacked Roborto. Roborto now has 20 health remaining.
Roborto attacked player. player now has 64 health remaining.
playerMoney 0
player attacked Amy Android. Amy Android now has 40 health remaining.
Amy Android attacked player. player now has 52 health remaining.
player attacked Amy Android. Amy Android now has 30 health remaining.
Amy Android attacked player. player now has 40 health remaining.
playerMoney -10
player attacked Robo Trumble. Robo Trumble now has 40 health remaining.
Robo Trumble attacked player. player now has 28 health remaining.
player attacked Robo Trumble. Robo Trumble now has 30 health remaining.
Robo Trumble attacked player. player now has 16 health remaining.
player attacked Robo Trumble. Robo Trumble now has 20 health remaining.
Robo Trumble attacked player. player now has 4 health remaining.
playerMoney -20
```

So now when we reply with "skip," 10 `playerMoney` credits are deducted from our total and we no longer face the same opponent just as planned.

Now we can apply our understanding of the `break` statement to change the conditional statements in the `while` loop regarding the fight or skip prompt. Because we can use the `break` statement to exit the loop, we'll rearrange the prompt for the fight or skip conditional statements. Essentially, we'll check whether the prompt was replied to with a skip. If not, we'll let the fight round continue.

Let's move the skip conditional statement to the top and convert it from an `else if` to an `if` statement:

```
// if user picks "skip" confirm and then stop the loop
if (promptFight === "skip" || promptFight === "SKIP") {
  // confirm user wants to skip
  var confirmSkip = window.confirm("Are you sure you'd like to quit?")

  // if yes (true), leave fight
  if (confirmSkip) {
    window.alert(playerName + " has decided to skip this fight. Goodby
    // subtract money from playerMoney for skipping
    playerMoney = playerMoney - 10;
    console.log("playerMoney", playerMoney)
    break;
  }
}
```

This conditional statement will directly follow the prompt at the beginning of the `while` loop:

```
    // ask user if they'd liked to fight or run
    var promptFight = window.prompt("Would you like FIGHT or SKIP this
```

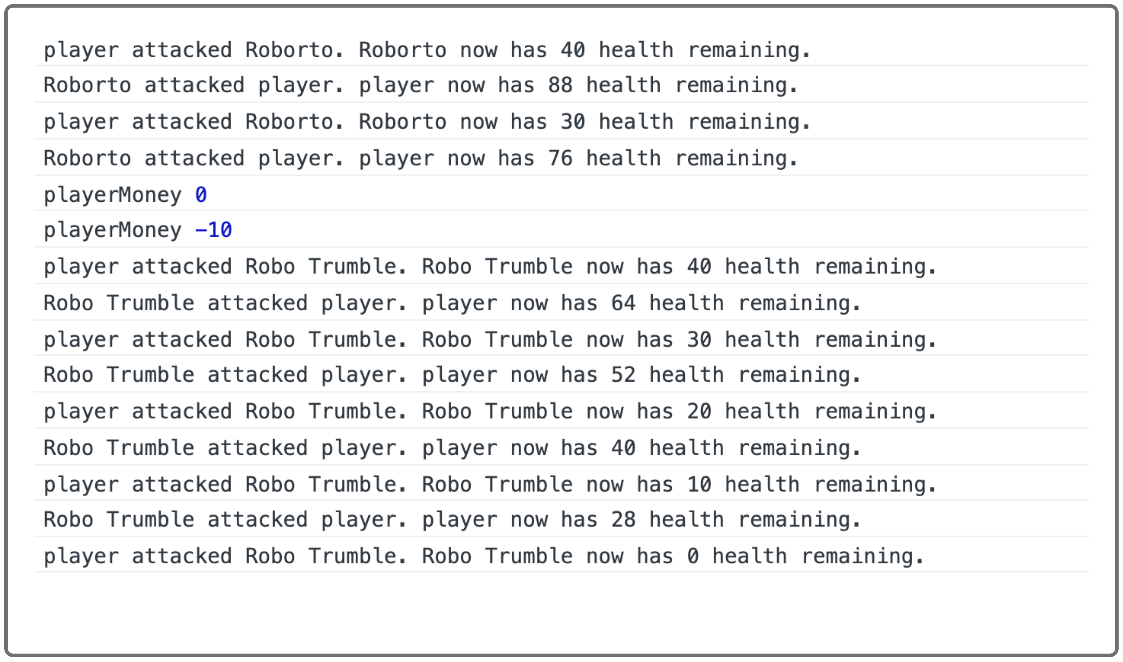
The rest of the `while` loop contains the code for robots to fight. To simplify things, we could set the "fight" option as the default action and then only have to check if skip was explicitly requested.

The entire `fight()` function should now look like this:

```
var fight = function(enemyName) {
  while (playerHealth > 0 && enemyHealth > 0) {
    // ask user if they'd liked to fight or run
    var promptFight = window.prompt('Would you like FIGHT or SKIP this

    // if user picks "skip" confirm and then stop the loop
    if (promptFight === "skip" || promptFight === "SKIP") {
      // confirm user wants to skip
      var confirmSkip = window.confirm("Are you sure you'd like to qui

      // if yes (true), leave fight
      if (confirmSkip) {
        window.alert(playerName + ' has decided to skip this fight. Go
        // subtract money from playerMoney for skipping
        playerMoney = playerMoney - 10;
        console.log("playerMoney", playerMoney)
        break;
      }
    }

    // remove enemy's health by subtracting the amount set in the play
    enemyHealth = enemyHealth - playerAttack;
    console.log(
      playerName + ' attacked ' + enemyName + '. ' + enemyName + ' now
    );

    // check enemy's health
    if (enemyHealth <= 0) {
      window.alert(enemyName + ' has died!');

      // award player money for winning
      playerMoney = playerMoney + 20;

      // leave while() loop since enemy is dead
      break;
    } else {
      window.alert(enemyName + ' still has ' + enemyHealth + ' health
    }

    // remove players's health by subtracting the amount set in the er
    playerHealth = playerHealth - enemyAttack;
    console.log(
      enemyName + ' attacked ' + playerName + '. ' + playerName + ' no
    );
    // check player's health
    if (playerHealth <= 0) {
      window.alert(playerName + ' has died!');
      // leave while() loop if player is dead
      break;
    } else {
      window.alert(playerName + ' still has ' + playerHealth + ' healt
    }
  }
};
```

Save and reload the browser to start game and test the code.

Try fighting twice, and then skipping twice, the proceed with fighting to the end. It should look like this in the console:



```
player attacked Roborto. Roborto now has 40 health remaining.
Roborto attacked player. player now has 88 health remaining.
player attacked Roborto. Roborto now has 30 health remaining.
Roborto attacked player. player now has 76 health remaining.
playerMoney 0
playerMoney -10
player attacked Robo Trumble. Robo Trumble now has 40 health remaining.
Robo Trumble attacked player. player now has 64 health remaining.
player attacked Robo Trumble. Robo Trumble now has 30 health remaining.
Robo Trumble attacked player. player now has 52 health remaining.
player attacked Robo Trumble. Robo Trumble now has 20 health remaining.
Robo Trumble attacked player. player now has 40 health remaining.
player attacked Robo Trumble. Robo Trumble now has 10 health remaining.
Robo Trumble attacked player. player now has 28 health remaining.
player attacked Robo Trumble. Robo Trumble now has 0 health remaining.
```

Excellent work! This would be a great place to preserve your work in GitHub. Do that now, then proceed to the final step of this lesson.