**Plan the UI tests**

In this section, you follow along with Amita and Andy as they talk about how to incorporate Selenium UI tests into the release pipeline. They begin by walking through the tests that Amita normally does manually. Then they map Amita's manual steps to automated test cases.

**Run UI tests manually**

Amita is waiting for Andy to show up. Andy is going to help Amita write a UI test that will be added to the *Test* stage of the pipeline. When he arrives, Andy sees Amita scribbling in her notebook, crossing out something, muttering, and then tearing out the page.

**Andy:** Hi. You don't look happy.

**Amita:** I'm not happy. I'm trying to figure out how to write an automated test, but I don't know where to start. I don't code. I feel like I'm obsolete.

**Andy:** Wow, I don't think it's that bad. For one thing, we'll always need someone who has the user's perspective in mind. There's no way to automate that. For another, no one starts out knowing how to automate tests. We were all beginners at some point. Hopefully, I can make the learning process a bit easier.

I think the best way to start is to automate something you regularly do manually. Pick a UI test. Then let's walk through it and write down the steps. Next we'll figure out how to automate those steps. What test should we pick?

Amita takes a deep breath.

**Amita:** Let's automate the modal window tests. When I click certain things, like the **Download game** button for example, I want to verify that the correct modal window appears. Then when I click away from the modal window, I want to verify that the modal window disappears and that the main window is active again.

**Andy:** That sounds like a great place to start. You run the test. I'll write down the procedure.

Amita opens a Windows laptop and launches Google Chrome. She goes to the web app and verifies that the home page opens.

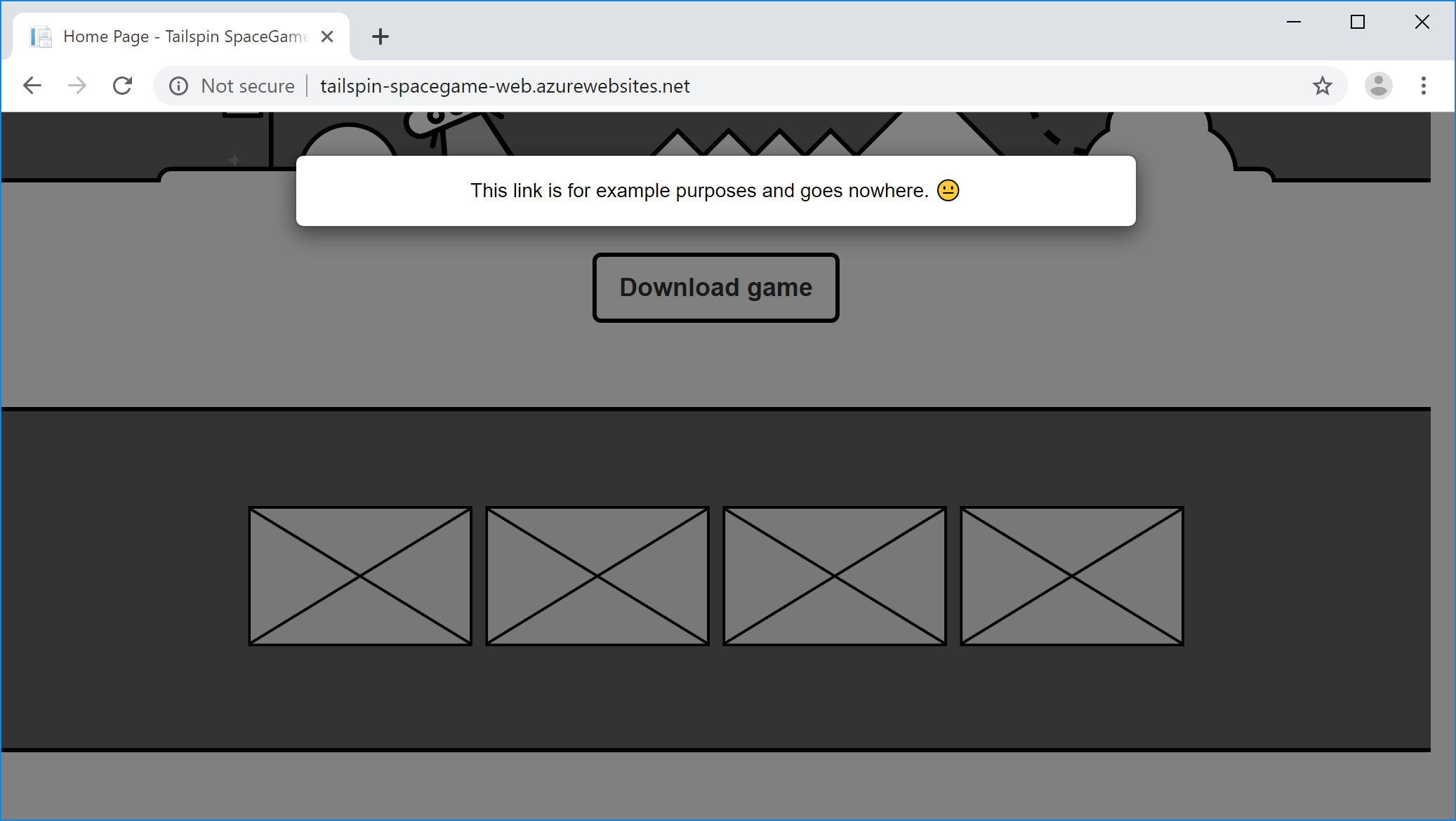
**Tip**

For reference, Microsoft hosts a version of the **Space Game** website. Go there if you want to follow along with Amita's manual tests. Otherwise, you can just read along.

**Andy:** OK. What do you check next?

**Amita:** I check that when I click the **Download game** button, the correct modal window appears.

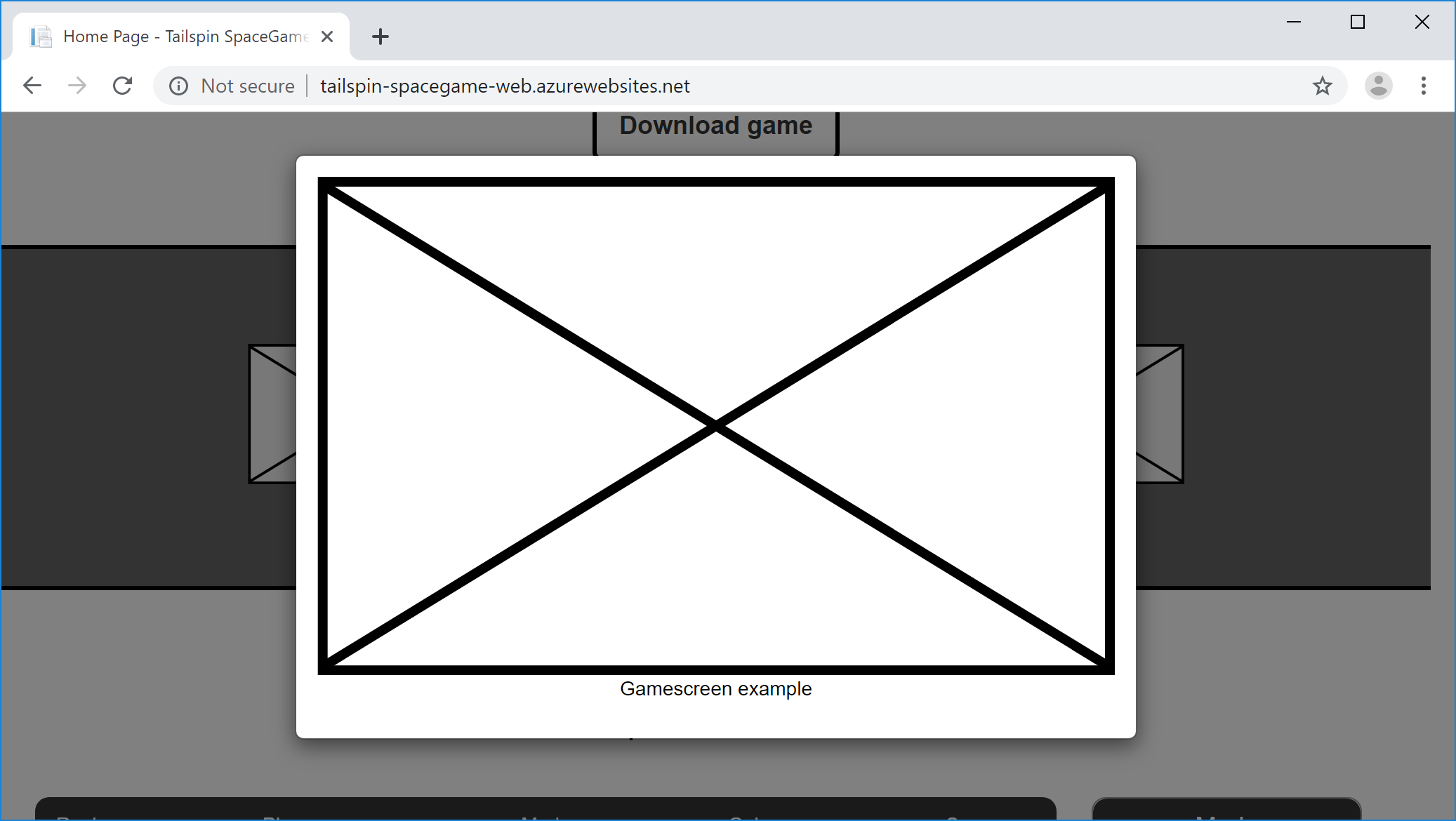
Amita clicks the **Download game** button. The modal window appears.



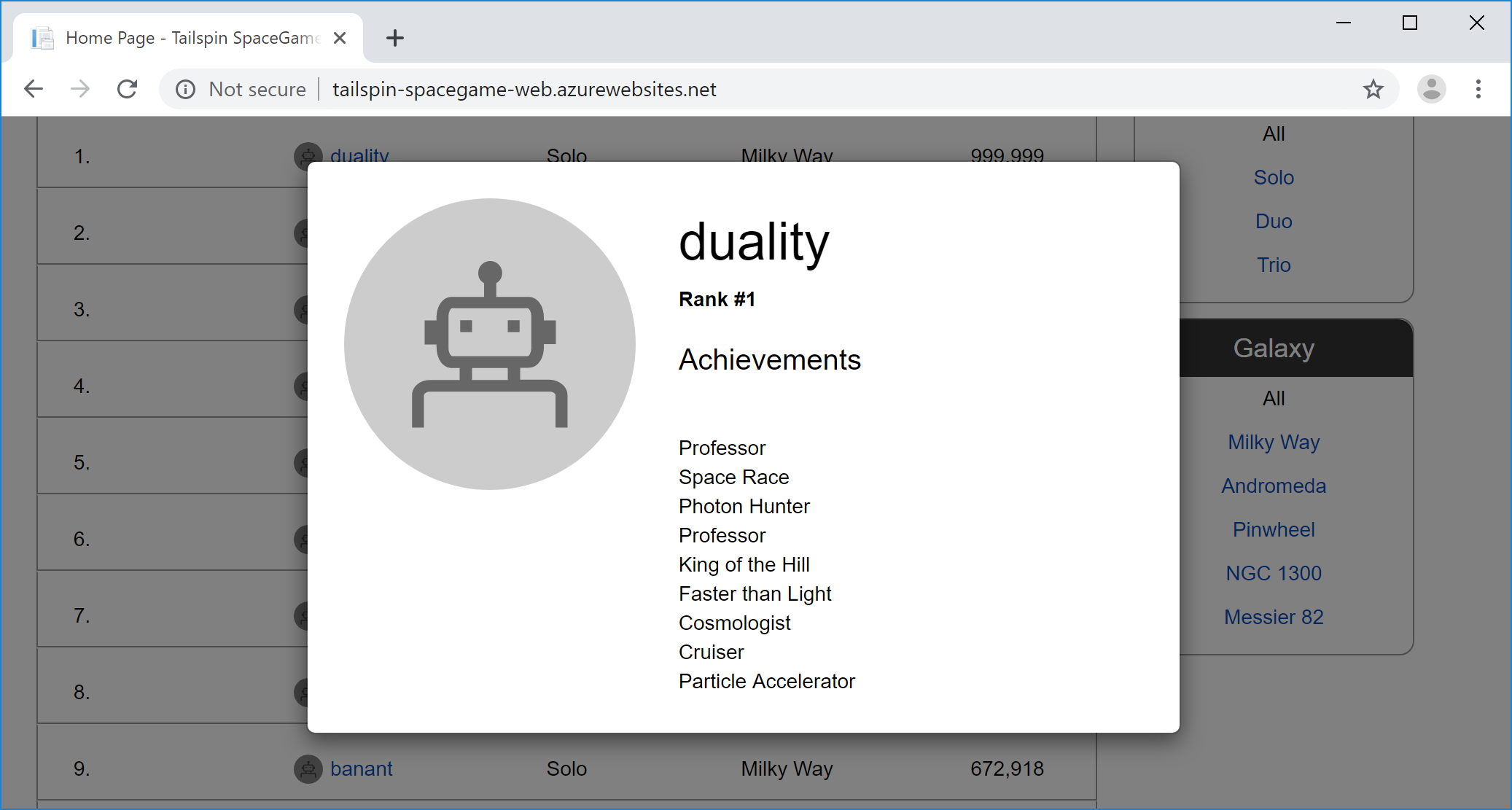
**Andy:** Great. What modal windows do you check next?

**Amita:** Next I check the four game screens. After that, I click the top player on the leaderboard. I verify that the player's profile appears.

Amita clicks each of the four thumbnail images to show the example game screens.



Next Amita clicks the top player on the leaderboard. The player's profile appears.



**Amita:** That covers the modal window tests. I run these tests on Windows because that's what most players use to visit our site. I run the tests on Chrome, and when I have time I also run them on Firefox and Microsoft Edge.

If I had time, I would run all the tests again on macOS and Linux, just to ensure we're compatible with any operating system that the players use to visit the site. But I need to run many other tests.

**What are locators in Selenium?**

In a Selenium test, a *locator* selects an HTML element from the DOM (Document Object Model) to act on. Think of the DOM as a tree or graph representation of an HTML document. Each node in the DOM represents a part of the document.

In a Selenium test, you can locate an HTML element by its:

* id attribute.
* name attribute.
* XPath expression.
* Link text or partial link text.
* Tag name, such as body or h1.
* CSS class name.
* CSS selector.

The locator you use depends on the way your HTML code is written and the kinds of queries you want to perform.

In an HTML document, the id attribute specifies a unique identifier for an HTML element. Here you'll use the id attribute to query for elements on the page because each identifier must be unique. This makes the id attribute one of the easiest ways to query for elements in a Selenium test.

**Get the ID for each HTML element**

Here you follow along with Amita and Andy as they collect the ID for each button that Amita clicks and for each resulting modal window.

**Andy:** I can see why these tests take so long and can be so frustrating. You're going to love automating them. I promise.

Here's what we'll do. We'll get the id attribute for each button you click and for the modal window that appears. The automated tests that we write can use these expressions to know which buttons to click and which modal windows to expect.

Let's start by getting the id attribute for the **Download game** button.

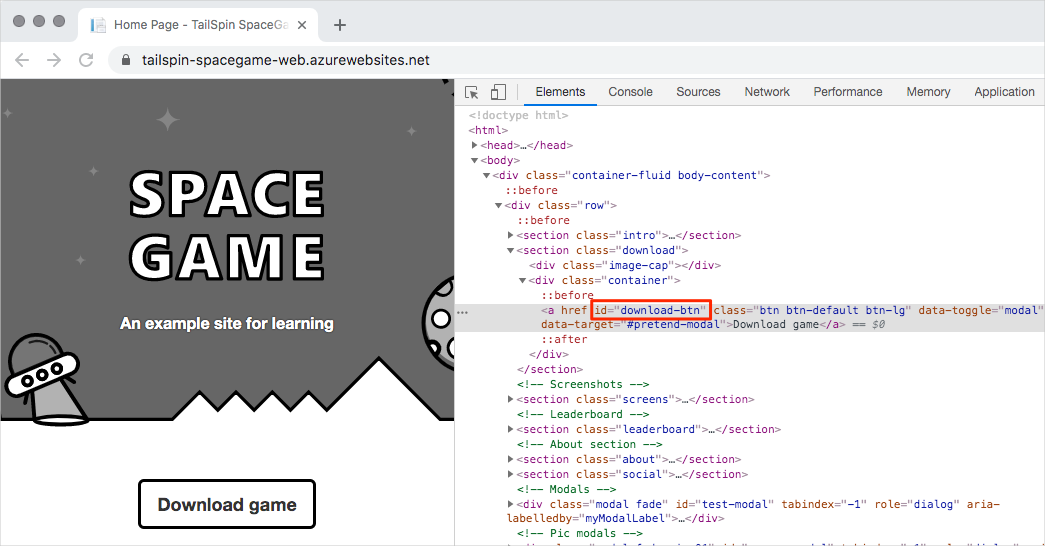
**Note**

You can follow these steps if you want to, or just read along. The next section provides all of the id attributes that you need when you run the automated tests.

1. In Google Chrome, go to the *Space Game* home page.
2. Right-click the **Download game** button. Then select **Inspect**.

The developer tools window opens. The HTML code for the **Download game** button is highlighted.

1. Examine the highlighted code and note the id attribute. Copy the id for later.



1. Select the **Download game** button. Then repeat steps 2 and 3 to get the id attribute for the modal window that appears.
2. Repeat the process for the four game screens and the top player on the leaderboard.

Amita opens Microsoft Word and adds a table. The table contains the id attribute for each link and the id attribute for the corresponding modal window. To keep the table basic, Amita records:

* The **Download game** button.
* Just one of the game screens.
* The top player on the leaderboard.

Here's what Amita's table looks like:

|  |  |  |
| --- | --- | --- |
|  | | |
| Feature | **Link id** | **Modal id** |
| Download game button | download-btn | pretend-modal |
| First game screen | screen-01 | screen-modal |
| Top leaderboard player | profile-1 | profile-modal-1 |

**Plan the automated tests**

**Amita:** OK. We have the id attribute for each button I click. We also have the resulting modal window. What's next?

**Andy:** I think we're ready to write our tests. Here's what we'll do:

1. Create an NUnit project that includes Selenium. The project will be stored in the directory along with the app's source code.
2. Write a test case that uses automation to click the specified link. The test case verifies that the expected modal window appears.
3. Use the id attribute we saved to specify the parameters to the test case method. This task creates a sequence, or series, of tests.
4. Configure the tests to run on Chrome, Firefox, and Microsoft Edge. This task creates a *matrix* of tests.
5. Run the tests and watch each web browser come up automatically.
6. Watch Selenium automatically run through the series of tests for each browser.
7. In the console window, verify that all the tests pass.

**Amita:** I'll be excited to see how quickly the tests run. Can we try the tests now?

**Andy:** Absolutely. Let's move over to my laptop. I have the app code ready.