Project_1_2_Tutorial_ALL
EMDA 203

Instructions

- Download the `Project_1_2_Tutorial_2020` folder from the Project_1_2 assignment section in Moodle.
- Copy it into your named folder on the "users" shared drive (your "P drive") just like you did last week. Delete the downloaded copy, so you don't get confused!
- Open the `Project_1_2_tutorial_code.html` file in both Chromeyou're your text editor of choice (Visual Studio Code, SublimeText, TextWrangler, etc.).

Workflow Check

Before doing any work, check that:

1. The file you are editing in TextWrangler is the same file you are viewing in Chrome.

Placing Your Hero on the Stage

Uncommment the seven lines around `45-51` to add the hero to the stage. Save and reload the page. What happens? Right-click and view the console information - what does it tell you to do? Use the console to figure out what the error is and fix it. When the error is fixed, reload the page and congratulate yourself on your first successful debugging.

Question: What does this tell you about the importance of descriptive variable naming? How is, `heroStartPositionX` a better variable name than, `htspsX` ?

Handling Keyboard Input

At about line `102`, uncomment the four lines of keyMonkey code and reload the page. Try using the arrow keys to move your hero around the stage. What happens? Right click and use the console to see if it can help you figure out what is wrong. Is the console helpful? Debug the error remembering that functions need ***two parts*** to run - the function *definition* and the ***function call*** which is the *command* to run the function. Get the page working and start exploring the stage with your hero using the arrow or WASD keys.

Question: Was the console helpful in finding this error?

Adding Targets to the Stage

Uncomment the three chunks of code starting at about line `54` to add the three targets to the stage and reload the page. Do you see all three targets? If one of the targets is missing, how would you go about figuring out why the third one isn't showing up? Is the console useful in

this situation? Debug the code and get the third target to show up, remembering that sometimes there are small typos that aren't actual errors, and will not show up in the console.

Javascript Display Layers
Move your hero around and interact with the targets. Notice that your hero passes UNDER the targets. Why do you think this is? Fix it so that your hero passes over the targets for better gameplay.

Question: What did you learn about how javascript orders the layers on the stage?

Curly Braces
-Go to around line `83` and remove the curly brace that ends the init function (simulating a typo that you could easily make while coding) and reload the page. Look in the console and note that "Unexpected end of input" is what the console displays for a missing curly brace – remember this. Put the curly brace back to get your code working again. Good coding practice is to make sure that all of your curly braces have descriptive comments next to them.

Question: Would the line number data in the console be useful for fixing a missing curly brace?

Tiny Typos Make Big Results
-Remove one of the close parentheses - ")" - at the end of line `34` and reload the page. Look at the data in the console and see if it would be helpful in finding this type of error. Put the parenthesis back and get things working again.

Question: How useful is the console for tracking down a missing parenthesis?

Passing an Argument
Around line `92`, uncomment the function call to `moveTarget`. Note that we've placed our `target01` variable in the parentheses of the function call. A specific value passed via the parentheses of a function call is known as an *argument.* In other words, `target01`, is an *argument* passed with this `moveTarget` function call.

Parameters are looking for Arguments (They're feisty that way)
Take a look around line `111`, where we define our `moveTarget` function. Note that the parentheses after the function name contain a custom variable named, `thisTarget`. A variable in the parentheses of a function *definition* is known as a *parameter*. Parameters allow a function to be more flexible because you can pass different *arguments* in your function calls. Save and test your code. What happens? Can you make the target fall *down* instead of sideways?
Try changing the *argument* passed around line `92` to `target02`. Can you add two more *function calls* with different *arguments* to get *all* of the targets to move?

Question: Which of the following is a function definition and which is a function call? Identify the *parameter* and the *argument*.

```
function sayHello(aName){
      console.log("Hi there, " + aName + "!");
}

sayHello("Nick");
```

Put Your Assets in the Game

Import the assets for *your* game from Assignment 1_2 (hero, background, targets etc.) by placing them into the /images/ folder for the tutorial and reloading the page. If some of your assets do not show up, make sure that the name of the asset and the name of the call in the code matches exactly.

Working in X and Y Space

Notice the targets are given a random x position by using Math.random( ) to generate a number between 0 & .999999 and then multiplying that number by the stage width. Could you refine this code, so the targets are never partially off screen? Also, experiment with adjusting the start position for your hero to get the feel for your game right. Can you add a conditional to the `moveTarget` function to stop the targets from traveling off-screen and into Miles' dreams?

Review

1. Workflow is critical, double check your file paths and always work in Chrome.
2. If your page doesn't work properly, always check the console. Sometimes, the console will tell you what the error is along with the specific line number of the error. Often, you have to do some detective work on your own. Unexpected end of input errors tend to revolve around misplaced curly braces **{ }**. If nothing is showing up in the browser and you're not getting any errors, you might have the wrong file loaded, commented out a key section of code, forgotten to re-save your file in TextWrangler, or forgotten to reload your browser.
3. Running a function requires two steps:
    a. The *definition* of the function - which uses the keyword `function`
    b. A function *call* that tells the computer to run the function. Functions are called/invoked/run by typing the name of the function followed by ( ).
4. Some function definitions include special variables known as *parameters*. A function's *parameters* are defined in the ( ) after the function name. The specific values passed to the function from a particular function call are known as *arguments*. For example, the function definition below has two *parameters*, `n1` and `n2`:

```
function addTwoNumbers(n1, n2){
      return n1 + n2;
}
```

A function *call* to this function would take two specific *arguments*, in this case, 3 and 7:

```
var sum = addTwoNumbers(3,7);
```

The value of `sum` would be `10`.

5. Items are stacked on the screen in the order that they are added to the stage. The code below would place the `hero` on top of the `bg`.

```
myStage.addChild(bg);
myStage.addChild(hero);
```