Pits Of Doom Lesson 2: Breaking It Down

One of the best things to do if you've never made a game before or you don't have much programming experience is to break things down into their most basic parts. We'll start off making this game by doing some really really simple programming and then adding functionality until the game is complete. It doesn't look like much now, but at the end of this tutorial you'll be able to download a fully working version with an installation file that you can use under the GNU public license to modify and edit as you wish.

The last thing I want to mention is the teaching style I'll be using throughout all of the lessons in this tutorial. I'm big on letting people try things on their own before I give them the answer. Pretend you're asked to take a history test. If I was to just hand you an answer sheet with all the right bubbles filled in you'd never learn anything from the test. This is a theory I hold to. In each of the lessons in this tutorial you'll find two versions of every file I've made. One file will have no code in it. That way you can try to program this game from scratch by yourself. But if that's too daunting don't worry. The other version of the file will have all the working code inside of it.

Even if you're not a programmer I encourage you to try your hand at programming each lesson. I'll go over some of the basic concepts you'll need to complete the programming before concluding with the reasoning behind what I've done so far.

Lesson Concepts

About PHP

PHP is a server side programming language. That means any code you write on any PHP file will never be visible to anyone who goes to a file with php code on it. The benefits of this are you don't have to worry about people stealing any code you've written. When someone views a PHP page, all they see is the HTML or text you've generated with PHP or typed onto the page yourself.

PHP files have a .php extension. A lot of people I've talked to take that to mean PHP files cannot display HTML or Javascript. That's incorrect. The .php extension is the server's way of knowing it needs to evaluate any PHP code on the file before the person requesting the page sees it.

The biggest draw back with PHP is that it's a server side programming language. That means any time you want to make a change to what the user sees on the page they either have to refresh it, or navigate to a new page. Websites where you see changes applied immediately (google mail, wordpress, to name a few) use something called AJAX. AJAX is an asynchronous relay between a server and a user's browser. This means they don't have to refresh the page to see changes. If you're interested in something like that then this tutorial is still the one for you. PHP is often used in combination with AJAX to create more dynamic web pages. However, I won't go over any AJAX in this tutorial, so once you're done making this game you'll need to find some AJAX tutorials.

Printing to the Screen

All PHP code is delimited by a start and end php tag that looks like this:

```
<?php
echo “Hello World!!”;
print “Hey there!”;
echo “<b>WOW!!</b><br>”;
print “This is sooooo neat…”;
?>
```

The PHP start and end tags are lot like matching HTML tags. If you don’t have any HTML experience I strongly recommend you take some of the tutorials on lissaexplains.com before you continue with the lessons here.
If you don’t have the correct start and end tags that would be one reason why someone could see your PHP code on the screen. Without the PHP tags the server doesn’t have any way to know what you’ve typed needs to be evaluated as PHP.

In the code above you’ll also see print and echo statements. It doesn’t really matter which one you use, but this is how you print something to the screen. Notice on the third print statement there is HTML in the quotation marks. There is nothing wrong with putting HTML inside of your PHP, in fact that’s one of the great things about it.

A few syntax things to keep in mind:

1. You always need matching start and end <?php ?> tags.
2. PHP statements always end with a semicolon ;
3. When you echo something to the screen it should be inside of single or double quotation marks
4. If you need to put quotes in your print statements then you should escape it like this:
5. ```
   <?php
   echo “And he said \”Hi Mom, how are you today?\””;
   ?>
   ```
6. The backslash tells PHP to evaluate the quotation mark as text and not as the end of the string.
7. You won’t see an error in PHP until you go to view the page. That’s because php is only evaluated on runtime. Sometimes this is very frustrating, and you’ve probably seen error codes before. If you get an error message read it, it usually tells you what you did wrong in your code.

Common PHP Errors & Solutions

Parse error: syntax error, unexpected T_STRING in [file location] on line [#]
This means you forgot a ” or a ‘ somewhere. There should always be at the start and end of a string. If you have a matching pair, make sure you don’t have one too many. If you have quotes inside of your string then you need to escape them using a backslash like this: \” or \’

Parse error: syntax error, unexpected ‘}’ in [file location] on line [#]
This means you have some kind of multi-line statement and you either forgot a bracket or having too many brackets. Brackets always come in pairs. Sometimes it mean you’re missing a semicolon at the end of the previous statement.

Parse error: syntax error, unexpected ‘””‘ in [file location] on line [#]
This means you probably forgot a parentheses before of after the DOUBLE quote. Sometimes it could also mean that you forgot to escape the double quote.
   ```

Parse error: syntax error, unexpected “‘” in [file location] on line [#]
This means you probably forgot a parentheses before of after the SINGLE quote. Sometimes it could also mean that you forgot to escape the SINGLE quote.

Parse error: syntax error, unexpected ‘;’ in [file location] on line [#]
This means that you’re missing something before the semicolon at the end of your statement. Usually its a closing parentheses. However, it could also mean you have a semicolon in a position PHP is not expecting one to be in. That usually happens in loops.

Parse error: syntax error, unexpected T_EXIT in [file location] on line [#]
This means you forgot a semicolon before an exit; call

Parse error: syntax error, unexpected $end in [file location] on line [#]
This usually means you forgot a semicolon or your function isn’t returning like it should be.

Variables
In PHP a variable is defined by a $ dollar sign. That means any word with a dollar sign in front of it will usually be evaluated for some other kind of content. The easiest way to think of a variable is to think about a jar. You can put things into a jar and take things out. You can stick a name on a jar but then decide you want to change that name later. Variables also work this way.

```
<?php
//this is a comment, comments are for you
//they aren’t processed as code
//get into the habit of leaving comments for yourself so if you come back later
//you can remember what you did or what you were trying to do
$myvariable = “Hello”;
$mysecond = ” World!”;
echo $myvariable;
echo $mysecond;
print “<br>”;
echo “$myvariable $mysecond<br>”;
$myvariable = 2;
echo $myvariable;
?>
```

In this example I have two variables. You can assign them a value using the equal sign. Variables in PHP are dynamic. That means you can assign them to one type (a string) and then reassing them to another type (an integer) without any kind of conversion. If that just went right over your head, don’t worry. All you really need to know is the syntax to assign a variable.

If-Else Statements

Games are all about decision making and that's the benefit of a programming language. In order for anything to work in this game we'll have to set it up to follow the rules and logic I briefly outlined inLesson 1. To do this we need to respond differently based upon the choice the user playing our game makes.

```
<?php
if ($_POST['button']) //someone clicked a button
{
echo "You pressed a button!";
}
else
{
echo "You haven't pressed a button yet. Go ahead, try it!";
}
?>
<form action="#" method="post">
<input type="submit" name="button" value="Press Me!">

</form>
```

In this example you see a .php file that has a section of PHP code and then below that it has some regular HTML. The part to pay attention to is the if-else section. Here we display a different message to the user based upon if they've clicked on our button or not. IF the button has been pressed, the code between the brackets { } of the if section will execute. Otherwise the code between the else brackets will execute. I'm sure you can already see the power of PHP in this one example. If you'd like more help with If-else statements and some more examples of if-else statements read my tutorial here. This covers some PHP programming concepts more in depth and gives more examples.

Form Basics

A form is one way to get information on what the user is doing back and forth to the server and PHP. All forms must have a start and end <form></form> tags. In order to submit any information from a form to the server you MUST have a submit button. Without a submit button you won't be able to tell the page that the user is trying to send you data about what he/she is doing. You can see a small form in the example above. It has both the start and end form tags, as well as a button. If that looks unfamiliar to you then you really should go back to lissaexplains.com and do some catching up on your HTML skills!

$_GET and $_POST Variables

In order to get data from a form PHP stores two special variables for us. The variables get data depending on the type of METHOD used in the form. If you look at our example above, we use a POST method. That means all the names and values of our form are sent to the server and stored inside of the $_POST variable. That way we can pull information from those variables to see if a user clicked a certain button or link. In the example above, we're checking to see if someone has clicked on our submit button that we've named button (yay for generic names!). If we'd called our submit button BOB and didn't change the if statement to check for $_POST['BOB'] then we'd never see the words telling us we'd pressed the button. As far as PHP is concerned, we no longer have a submit button called button so there's no way it will ever be pressed.

Summary

There you have it. I've not covered in brief (and sorry if it's not brief enough for you — or too brief!!) all of the major concepts in the first part of the tutorial files you can download below. Again, if you're up for the challenge, try programming this file yourself by opening the character2.php file. At the top of the page it will outline everything I'm trying to accomplish in this particular file for this lesson. If you just want to see some working code then download character.php and jump right into the next lesson.

Game Files

character.php
character2.php (no source code)
You'll need to copy/paste the source from one of these files and give it a .php extension for the last part of the tutorial to make sense. All of these files are viewable as .txt files so you can see the PHP code inside of them. If they were .php files you wouldn't ever be able to see the .php code. Do you remember why?

Lesson Reasoning

A lot of the logic behind the game we're going to make has to do with movement of a character around the game world. Remember there is a difference between a member and a character. A member is an actual person who will login and play this game by the time we're done. A character is something the member will use to navigate the game. Each member can have multiple characters.

Character.php is from the point of view of one character. Each character will be faced with the option of which direction they can move — left, right, forward or back. Every time a character moves we'll need to evaluate the move and take some action depending upon the new location they've moved to.

Although this file only checks for a single direction and places the "treasure" in one specific location, it's a step in the right direction. With this file we know which direction has been selected, we can compare that direction to the new position they've moved to, and we can print a message on the screen indicating if the character won or lost based upon finding the treasure.

Find lesson 3 at http://design1online.com!