Pits Of Doom Lesson 5: Easy Map Editor

In the last lesson we made some real progress. Now, instead of having a boring page with a click of one button before the game is over, we have a character that can move around a much larger area, be blocked by walls, fall into pits and die, and then win if they find the treasure.

That's all fine and dandy but our game needs more substance. It's time to add more maps, map levels, and our very own easy map editor so we can quickly make maps, generate map files, and play our game with a whole new level of complexity.

Lesson Concepts

Including Files

One of the best things about programming is being able to break bigger, more complex problems down into smaller more manageable pieces. Include files are one way to do that. Let's take our custom function we wrote last lesson to display the map on the screen. Did you notice that our character.php file is becoming a bit crowded and hard to read? Wouldn't it be nice if we could take the displayMap() function out off of the code in that page but still use it on there?

That's what an include file does. Using an include file is easy. First you create a new file. I like to call files where I keep a lot of functions something really generic: functions.php. That way when you go back looking for things later and you find a function being used that's not in the file you have a good idea of where to check to find it.

The next thing you do is put some code inside of functions.php. In our case, the display map function. So all we do is cut/paste the function from our character.php file to the functions.php file.

Last but not least, we go back to character.php and at the top of the page we write:

```
<?php
session_start();
include('functions.php');
//rest of the code here like normal….
?>
```

What does this do? It literally takes the contents of the functions.php file and dumps it onto the character.php page. That way any piece of code you have inside of functions.php is now accessible to the page just like if we'd written the displayMap() function directly inside the character.php page. How neat is that?

I'm sure you can already see the benefits of doing this like this. It makes everything much easier to read.

Question: When is it a good time to make a function?

Functions are great things. My rule of thumb is, if you have a piece of code, or even some kind of page formatting that you plan on using again and again, make it into a function. That way, instead of having to type the same thing over and over and over again all you do is call your function and pass it unique parameters.

Take our character movement for example. Every time someone clicks on a button we have to check to see which button they pressed, what's in the direction they're moving, and how to respond to what's in that direction. Do you notice the similarities in the code for EVERY direction?? If you run into situations like this, it's a good time to stop and think function.

Give it a try! Can you convert the character movement into a function? Here's a hint: Every time the player moves all you really need to know is what direction it's moving in. Then, according to that direction, you can check the x and y values of the map to see what's there.

Automatic File Downloads

In this lesson we're creating a map editor we can use to create map files fairly quickly. In our map editor, we want to take the map file we created and automatically download it, or export it, once we're done. In order to do that you'll need to know some information about files and http headers. Despite how hard these seems like it should be, it's only a few lines of code.

Let's talk about HTTP headers. HTTP is Hypertext Transfer Protocol, or one way we use to view files online. You may be familiar with other things like FTP, file transfer protocol or TCP/IP, Transfer Control Protocol/Internet Protocol. But we're only concerned about the one.

Every time you request a file from someplace on the internet a lot of different things go on. If you're interested in how everything works I highly suggest going to wikipedia's networking section. However, the more important thing you need to know is that for any file you view online, it starts with header information that describes the type of file it is, the length of the file, and the content of the file.

PHP, since it's a server side programming language, can be used to setup the headers on any file you create before someone else sees the file online. In this way, we can use PHP to tell the browser anything we write on a HTML file should be treated like it's a music file, or an application, a PDF, or a text file. Whatever we want! Create a file and call it testdownload.php:

```
<?php
//let's create a name for the file you'll download
$filename = "My Funky File.txt";
//Tell PHP that we want to change the header information for this .php file and instead it'll become a .txt file
header("Content-Type: text");
//Now we add an additional statement that's meant to make your browser automatically download this file's content
header('Content-Disposition: attachment; filename="' . $filename .'"');
//now anything we echo on this page will become a part of the file we're downloading! think of all the possibilities!
echo "This is my totally awesome, funky cool file that I made download automatically using .php! You can't even tell this was actually inside a .php script because the headers are being set to text :)";
?>
```

That's it! See, with only a few lines of code you can create a file that automatically downloads anything you echo on the page with the name, and extension you want it to have. We can use this concept with our map editor, to echo all the options someone selects to the page after we've set the headers and create automatically downloads for the map files.

Reading From A File

Reading from a file is a bit trickier. Instead of going over the concept I'm going to put the actual code I wrote specifically for the map editor I made with a ton of comments explaining what I've done and why. Please note that I wrote this file reader for the format generated by the map editor. It probably won't work for any map file you've made yourself so you'll need to re-create it using the map editor first.

Another thing I'll warn you about. I keep my map files in another directory called maps. If you keep your map files in a different location you'll need to change the $directory variable to reflect that. If you keep the map files in the same place as the game and map editor files then uncomment the first $directory line and comment out the second $directory line.

```
<?php
/**************
* Load Maze File
* NOTE: This was designed specifiically for files made with the map editor
*               it's not guaranteed to work for a file you made and formatted yourself.
*               Please use the editor to generate these files for you.
**************/
function loadMap($filename, $level) {
        $map = array();
        //if you keep the map files in a particular directory
        //fill that in below. Otherwise you can leave this blank
        //$directory = "";
        $directory = "maps/";
        if (file_exists($directory . $filename . "_" . $level . ".txt"))   {
                //open our file for reading the contents
                $fileline = file($directory . $filename . "_" . $level . ".txt");
                $x = 0;
                //while there is data in the file
                //read it one line at a time
                foreach ($fileline as $line_num => $line)
                {
                        $i = 1;
                        $y = 0; //we need to reset this each time
                                //so our row always starts at zero
                                //if this data is the start of our map
                                //it should always be a wall
                                if (substr($line, $i, 1) == "W")
```

```
                {
                        //start pulling the info for the first row
                        //keep loading in info until we reach the end of the line
                        //in the row
                        while (substr($line, $i, 1) != "\n")
                        {
                                if ($i % 2 != 0) //we do this so we don't load
                                                //in any of the spaces between characters
                                {
                                        $map[$x][$y] = substr($line, $i, 1);
                                        $y++;
                                }
                                $i++; //take the next character in the row
                        }
                        $x++; //increment to the next row
                }
        }
        return $map; //return the array with all the map data
    }
    else
        return "File not found!";
}
?>
```

This function does three major things. First, it checks to see if the file exists and if it doesn't it gives us an error message. Otherwise it opens the file, searches through it until if finds the first line of the map. We know when we find the first line of the map because it'll be a W character, the wall that runs all around the outside of the map. Once it finds a line in the file from the map it inserts it into the array and increments the array variables by 1. That way each row and column of the map are read in until it doesn't find any more.

We know that we've reached the end of a line (or the end of a row in the map) when we reach a \n character. This is the same thing as a carriage return in the file, or an end of line character. This tells us there's nothing else left to read on the line, so we increment the array to start reading from the next row.

If this seems like it's a bit out of your understanding don't worry. This is actually much easier to do when you're reading or inserting values into a database that we'll cover next lesson.

Some Simple Javascript

I know I said I wouldn't go into AJAX and Javascript but in creating the map editor I realized it would be easier to use visually if I added some Javascript for automatic color coding. So I'll go over both functions I wrote and how they work below.

Javascript starts with different tags then what you see in PHP but it has both a start and end tag.

```
<script language="javascript">
 function setColor(object) {
        object.style.backgroundColor = object.options[object.selectedIndex].style.backgroundColor;
        object.style.color = object.options[object.selectedIndex].style.color;
 }
 function resetColors(object) {
        var i;
        //we need to make sure we skip the buttons at the
       //bottom of the page, so we have to subtract 2 from
       //all the items in the form
       for (i = 0; i < (object.length-2); i++)
              setColor(object[i]);
 }</script>
```

The very first function is simple. It takes the color of the option in the select box and changes the select box to match that color. The second function is a little more complicated. When we call it we pass it a form, or all of the things between the <form> and </form> tags in our map editor file. This includes all of the select boxes we're using to let you create the map. Then it goes through all of those select boxes and calls the setColor function.

The reason we use this second function has to do with first loading the editor. The first time the editor is loaded we run a PHP script so that the borders of the map are automatically walls. When we do this, we still want to update the color of the walls we set. That's what the resetColors function is for, it goes through every select box on the page and sets it to the appropriate color. You'll notice when you first start the editor it appears with all white select boxes after a few seconds the borders of the map changes to gray. That change is caused by the javascript call to reset the colors. Try removing the resetColors call from the map editor. Do you see the difference when you try to run it now?

Pits of Doom Map Editor

I know I haven't done this before, but I feel like we're at a point where more explanation is necessary. Now I'm going to talk about the map editor, some of it's features, and how you use it.

The map editor is a great, useful tool we can use to build this game quickly and efficiently. It has three major features to it. The first page you see when you open it is setting up the name and the level of the map. If you're making multiple levels on the same map then they all need the same name. Then you can enter a width and height for the map. I suggest using a size 15 x 15 or smaller, otherwise you'll have to do a lot of back and forth scrolling, but any size you pick will work.

Once you've filled out the map details click on the start map editor. The screen you'll see next has several neat features.

- To help make building a map faster, all the edges are automatically turned into a wall. For the sake of the map file loader make sure you always have walls around the edge of your map.
- The map editor is also color coded, to make everything more visually pleasing.

- S is the starting position of the character. You can have more then one, but the files I've included always pick the starting position closest to the top of the board. If you'd like to add in functionality to randomly pick the starting position by all means go ahead.
- An addition to the map editor is the L, or ladder. Ladders will allow the character to move up or down a level if they land on one. A character cannot move up a level if they're on the top map level, or level 1. In the other direction, a character cannot move down a level if they're on the bottom level of the map. Right now we'll assume all maps have 5 levels.
- The last (and best) function of the map editor is the export file button. This lets you automatically download your map as a .txt file. Then you can use these .txt files to load maps into your game and play them!

Ladders, Pits & Level Changes

In this version of Pits of Doom we need to add the functionality of moving up and down ladders and falling down pits. The easiest thing to do first is create a function for all of our movement in the game.

Once you've done that, we need to make our $_SESSION variable global to our function. That means the function can access the values we already have inside of the $_SESSION variable. By setting the map name and level in a session we'll always know which map we're on and which level we're on. If we apply this same concept to the map array values, we'll always know what's in the map once we've loaded it.

Any time we're on a pit we now need to change the level the character is on so they fall down a level (unless they're not on the last level of the map!). To make things easier for ourselves, we'll make sure the map at level 5 doesn't have any pits in it.

Once the character's level has been dropped to the map file below, we load the map for that new level. Before the character can start playing this level again we need to find the starting position on this new map. To do that I've written two simple functions:

```
<?php
/**************
* This finds the position of the starting location
* marked with the value S and returns the X
* coordinate. It MUST be passed an entire array
* filled with maze data for it to work.
*
* Tip!: The variable name in the function parameter
*       (in this case its $array) does not have to match
*       the name of the variable you pass to the function
*       when you call it.
**************/
function startPositionX($array) {
        for ($x = 0; $x < height($array); $x++)
                for ($y = 0; $y < width($array); $y++)
                        if ($array[$x][$y] == "S")
                                return $x;
```

```
}

/***************
* This finds the position of the starting location
* marked with the value S and returns the Y
* coordinate. It MUST be passed an entire array
* filled with maze data for it to work.
***************/
function startPositionY($array) {
        for ($x = 0; $x < height($array); $x++)
                for ($y = 0; $y < width($array); $y++)
                        if ($array[$x][$y] == "S")
                                return $y;
}
?>
```

One of these finds the X position of the starting position and the other finds the Y position. Using these we can reset the character's default position on the new map we loaded. Note that height and width are additional functions I've written to find the height and width of the two-dimensional array.

Ladders use a similar concept with one difference. If someone is on a ladder they now have the option of moving up or down instead. To do that, I've written a special if statement for ladders. When someone is on a ladder they can use the up and down buttons to move up or down the ladder instead of moving backwards or forwards on the map. If the character is on the bottom level, we allow them to move back instead of moving down the ladder, and if they're on the top level we let them move forward instead of moving up a level.

Summary

Our game is really coming along! I'm quite pleased with what I have right now. Not only can I move up and down ladders, fall into pits, and find the treasure, I'm having fun doing it. Right now I feel like we have a really solid foundation for how character movement is going to work. I think it's time to take things to a new level of complexity by adding in the database.

Game Files

Working Version:

character.php
mapeditor.php

Source Code:

character.txt
functions.txt
mapeditor.txt
mapsymbols.txt

Map Files
Round Hill_1.txt
Round Hill_2.txt
Round Hill_3.txt
Round Hill_4.txt
Round Hill_5.txt

Try it Yourself:
character.php
functions.php

Lesson Reasoning

In this lesson we're really starting to tie a lot more things together. Now we know how to load a map from an outside data source and swap those maps as they get to certain points on the map. We now have a fairly decent skeleton of what our map/character movement functionality is going to be like once the game is finished.

But this still isn't what we want!! We need everything to be done from the database, loaded and ready to go so we can change it as and how we see fit. In our next lesson we'll work on an introduction to SQL, setting up and connecting to a database, creating tables, and changing over everything we've done so far (including our easy map editor) to pull from and save to the database.

Look at that! Only 5 lessons and we're already moving on to interacting with a database.

Find lesson 6 at http://design1online.com!