**Writing the Poker Dice Program**

It's time to take another look at the Poker Dice program that made its debut at the beginning of this chapter. As usual, this program doesn't do anything you haven't already learned. It is a little more complex than the trivial sample programs I've been showing you in this chapter, but it's surprisingly compact considering how much it does. It won't surprise you that arrays and loops are the secret to this program's success.

**Setting up the HTML**

As always, a basic HTML page serves as the foundation for the PHP program. I chose to add a simple style sheet to this page to make tan characters on a green background.

<html>

<head>

<title>poker dice</title>

<style type = "text/css">

body {

background: green;

color: tan;

}

</style>

</head>

<body>

<center>

<h1>Poker Dice</h1>

<form>

<?

**Building the Main Code Body**

The poker dice program is long enough to merit functions. I've broken it down into smaller segments to describe it here, but you may also want to look at the code in its entirety from the CD-ROM that accompanies this book.

The main part of the code is used to set up the general flow of the program. Most of the work is done in other functions called from this main area.

//check to see if this is first time here

if (empty($cash)){

$cash = 100;

} // end if

rollDice();

if ($secondRoll == TRUE){

print "<h2>Second roll</h2>\n";

$secondRoll = FALSE;

evaluate();

} else {

print "<h2>First roll</h2>\n";

$secondRoll = TRUE;

} // end if

printStuff();

The first order of business is to see if this is the first time the user has come to this page. It's important to understand how timing works in this program. The user will feel like he or she is playing the same game for several turns, but actually each time he or she rolls the dice, the entire program runs again. The program will have different behavior based on which form elements (if any) have values. If the user has never been to the page before, the value for the $cash variable will be null. The first if statement checks this condition. If the $cash variable has not yet been created, the user will get a starting value of $100. (I wish real casinos worked like this...)

The program then calls the rollDice() function, which will be described momentarily. This function rolls the dice and prints them to the screen.

If you look carefully at the program as it is running, you'll see it runs in two different modes. Each turn consists of two possible rolls. On the first roll, the user is given the ability to save a roll with a checkbox, and no scoring is performed. On the second roll, there are no checkboxes (because the user will need to start with all fresh dice on the next turn) and the program keeps track of the player's score by adding money for various combinations.

The $secondRoll variable is used to keep track of whether the user is on the second roll. I chose to give it the value TRUE when the user is on the second roll and FALSE when on the first roll. If $secondRoll is TRUE, the program will call the evaluate() function, which will tally any losses or winnings. Regardless, I inform the user which roll it is, and change the value of $secondRoll so it reflects what should happen the next time this program is called (which will happen when the user clicks on the Submit button).

**Making the rollDice() Function**

The job of the rollDice() function is—well—to roll the dice. It's a somewhat long function, so I'll print it all out for you here, then I'll explain it in smaller chunks. Essentially, this function builds an HTML table based on five die rolls. It is able to determine if the user has chosen to keep any previous dice, and only rolls a new die if the user did not choose to keep it. If it is the first roll, the program prints a checkbox, which allows the user to select a die to keep.

function rollDice(){

global $die, $secondRoll, $keepIt;

print "<table border = 1><td><tr>";

for ($i = 0; $i < 5; $i++){

if ($keepIt[$i] == ""){

$die[$i] = rand(1, 6);

} else {

$die[$i] = $keepIt[$i];

} // end if

$theFile = "die" . $die[$i] . ".jpg";

//print out dice images

print <<<HERE

<td>

<img src = "$theFile"

height = 50

width = 50><br>

HERE;

//print out a checkbox on first roll only

if ($secondRoll == FALSE){

print <<<HERE

<input type = "checkbox"

name = "keepIt[$i]"

value = $die[$i]>

</td>

HERE;

} // end if

} // end for loop

//print out submit button and end of table

print <<<HERE

</tr></td>

<tr>

<td colspan = "5">

<center>

<input type = "submit"

value = "roll again">

</center>

</td>

</tr>

</table>

HERE;

} // end rollDice

The checkboxes that appear sometimes are special. The general strategy for them is this: If it's the first turn, I'll print out a checkbox under each die. The checkboxes are all called keepIt, and all have an index. When PHP sees these variables with the same name but different indices, it will automatically create an array. Checkboxes in PHP are a little different than some of the other form elements, because they only send a value if they are checked. Any checkbox the user does not check will not be passed to the program. If the checkbox has been checked, the value associated with that checkbox will be passed to the program.

**Rolling the Dice if Necessary**

The program uses two arrays to keep track of the dice. The $die array is used to store the current values of all the dice. The $keepIt array will contain no values unless the user has checked the corresponding checkbox (which will only happen on the first roll, because the checkboxes will not be printed on the second roll).

if ($keepIt[$i] == ""){

$die[$i] = rand(1, 6);

} else {

$die[$i] = $keepIt[$i];

} // end if

$theFile = "die" . $die[$i] . ".jpg";

For each die, if the user chose to keep the previous value, that previous value will be stored in the appropriate element of the $keepIt array. If so, the $keepIt value will be transferred over to the $die array. Otherwise, the program will roll a new random value for the die.

**Printing Out the Table Contents**

Once the function has determined a value for each die (by copying it from $keepIt or rolling a new value as appropriate) it is time to print out the image corresponding to each die.

//print out dice images

print <<<HERE

<td>

<img src = "$theFile"

height = 50

width = 50><br>

HERE;

//print out a checkbox on first roll only

if ($secondRoll == FALSE){

print <<<HERE

<input type = "checkbox"

name = "keepIt[$i]"

value = $die[$i]>

</td>

HERE;

} // end if

If it's the first roll, the function also prints out the keepIt checkbox corresponding to this die. Note how the name of the checkbox will correspond to the die name. (Remember, the value $i will be translated to a number before the HTML page is printed.) The value of the current die is stored as the value of the keepIt checkbox.

|  |  |  |
| --- | --- | --- |
|  | TRICK? | If you're still baffled, that's okay. It can be hard to see how all this works together. It might help to run the program a couple of times and look carefully at the HTML source that's being generated. To fully understand a PHP program, you can't always look at it simply on the surface as the end user will. You may need to see the HTML elements that are hidden to the user to fully understand what's going on. |

**Printing Up the End of the Table**

After the loop that rolls and prints out the dice, it's a simple matter to print the Submit button and the end of table HTML.

//print out submit button and end of table

print <<<HERE

</tr></td>

<tr>

<td colspan = "5">

<center>

<input type = "submit"

value = "roll again">

</center>

</td>

</tr>

</table>

HERE;

Note that since no action was specified in the form, PHP will default to the same page that contains the form. This is convenient for programs like this that call themselves repeatedly.

**Creating the evaluate() Function**

The purpose of the evaluate() function is to examine the $die array and see if the user has achieved patterns worthy of reward. Again, I'll print out the entire function here, and then show you some highlights below.

function evaluate(){

global $die, $cash;

//set up payoff

$payoff = 0;

//subtract some money for this roll

$cash -= 2;

//count the dice

$numVals = array(6);

for ($theVal = 1; $theVal <= 6; $theVal++){

for ($dieNum = 0; $dieNum < 5; $dieNum++){

if ($die[$dieNum] == $theVal){

$numVals[$theVal]++;

} // end if

} // end dieNum for loop

} // end theVal for loop

//print out results

// for ($i = 1; $i <= 6; $i++){

// print "$i: $numVals[$i]<br>\n";

// } // end for loop

//count how many pairs, threes, fours, fives

$numPairs = 0;

$numThrees = 0;

$numFours = 0;

$numFives = 0;

for ($i = 1; $i <= 6; $i++){

switch ($numVals[$i]){

case 2:

$numPairs++;

break;

case 3:

$numThrees++;

break;

case 4:

$numFours++;

break;

case 5:

$numFives++;

break;

} // end switch

} // end for loop

//check for two pairs

if ($numPairs == 2){

print "You have two pairs!<br>\n";

$payoff = 1;

} // end if

//check for three of a kind and full house

if ($numThrees == 1){

if ($numPairs == 1){

//three of a kind and a pair is a full house

print "You have a full house!<br>\n";

$payoff = 5;

} else {

print "You have three of a kind!<br>\n";

$payoff = 2;

} // end 'pair' if

} // end 'three' if

//check for four of a kind

if ($numFours == 1){

print "You have four of a kind!<br>\n";

$payoff = 5;

} // end if

//check for five of a kind

if ($numFives == 1){

print "You got five of a kind!<br>\n";

$payoff = 10;

} // end if

//check for flushes

if (($numVals[1] == 1)

&& ($numVals[2] == 1)

&& ($numVals[3] == 1)

&& ($numVals[4] == 1)

&& ($numVals[5] == 1)){

print "You have a flush!<br>\n";

$payoff = 10;

} // end if

if (($numVals[2] == 1)

&& ($numVals[3] == 1)

&& ($numVals[4] == 1)

&& ($numVals[5] == 1)

&& ($numVals[6] == 1)){

print "You have a flush!<br>\n";

$payoff = 10;

} // end if

print "You bet 2<br>\n";

print "Payoff is $payoff<br>\n";

$cash += $payoff;

} // end evaluate

The general strategy of the evaluate() function is to subtract two dollars for the player's bet each time. (Change this to make the game easier or harder.) Then I created a new array called $numVals, which tracks how many times each possible value appears. Analyzing the $numVals array is an easier way to track the various scoring combinations than looking directly at the $die array. The rest of the function checks each of the possible scoring combinations and calculates an appropriate payoff.

**Counting Up the Dice Values**

When you think about the various scoring combinations in this game, it's important to know how many of each value the user rolled. The user will get points for pairs, three, four, and five of a kind, and flushes (five values in a row). I made a new array called $numVals which has six elements. $numVals[1] will contain the number of ones the user rolled. $numVals[2] shows how many twos, and so on.

//count the dice

for ($theVal = 1; $theVal <= 6; $theVal++){

for ($dieNum = 0; $dieNum < 5; $dieNum++){

if ($die[$dieNum] == $theVal){

$numVals[$theVal]++;

} // end if

} // end dieNum for loop

} // end theVal for loop

//print out results

// for ($i = 1; $i <= 6; $i++){

// print "$i: $numVals[$i]<br>\n";

// } // end for loop

To build the $numVals array, I stepped through each possible value (1 through 6) with a for loop. I then used another for loop to look at each die and determine if it showed the appropriate value. (In other words, I checked for ones the first time through the outer loop, then twos, then threes, and so on.) If I found the current value, I incremented $numVals[$theVal] appropriately.

Notice the lines at the end of this segment that are commented out. There was no need to move on with the scorekeeping code if the $numVals array was not working as expected, so I put in a quick loop that would tell me how many of each value the program found. This was a quick way to make sure my program was working properly before I added new functionality to it. It's smart to periodically check your work and make sure that things are working as you expected. When things were working correctly, I decided to place comments in front of each line to temporarily turn it off. By doing this, I removed the code, but it is still there if something goes wrong and I need to look at the $numVals array again.

**Counting Up Pairs, Twos, Threes, Fours, and Fives**

The $numVals array has most of the information I need, but it's not quite in the right format yet. The user will earn cash for pairs, and for three, four, and five of a kind. To check for these conditions, I'll use some other variables and another loop to look at $numVals.

//count how many pairs, threes, fours, fives

$numPairs = 0;

$numThrees = 0;

$numFours = 0;

$numFives = 0;

for ($i = 1; $i <= 6; $i++){

switch ($numVals[$i]){

case 2:

$numPairs++;

break;

case 3:

$numThrees++;

break;

case 4:

$numFours++;

break;

case 5:

$numFives++;

break;

} // end switch

} // end for loop

First, I created variables to track pairs, and threes, fours, and fives of a kind. I initialized all these variables to zero. I then stepped through the $numVals array to see how many of each value occurred. If, for example, the user rolled 1, 1, 5, 5, 5, $numVals[1] will equal 2 and $numVals[5] will equal 3. After the switch statement executes, $numPairs will equal 1 and $numThrees will equal 1. All the other $num variables will still contain zero. Creating these variables will make it very easy to determine which scoring situations (if any) have occurred.

**Looking for Two Pairs**

All the work setting up the scoring variables pays off, because it's now very easy to determine when a scoring condition has occurred. I chose to award the user one dollar for two pairs (and nothing for one pair.) If the value of $numPairs is 2, the user has gotten two pairs, and the $payoff variable is given the value 1.

//check for two pairs

if ($numPairs == 2){

print "You have two pairs!<br>\n";

$payoff = 1;

} // end if

Of course, you're welcome to change the payoffs as you wish. As it stands, this game is incredibly generous, but that makes it fun for the user.

**Looking for Three of a Kind and Full House**

I decided to combine the checks for three of a kind and full house (which is three of a kind and a pair). The code first checks for three of a kind by looking at $numThrees. If the user has three of a kind, it then checks for a pair. If both these conditions are true, it's a full house, and the user is rewarded appropriately. If there isn't a pair, there is still a (meager) reward for the three of a kind.

//check for three of a kind and full house

if ($numThrees == 1){

if ($numPairs == 1){

//three of a kind and a pair is a full house

print "You have a full house!<br>\n";

$payoff = 5;

} else {

print "You have three of a kind!<br>\n";

$payoff = 2;

} // end 'pair' if

} // end 'three' if

**Checking for Four and Five of a Kind**

Checking for four and five of a kind is trivial. All that is necessary is to look at the appropriate variables.

//check for four of a kind

if ($numFours == 1){

print "You have four of a kind!<br>\n";

$payoff = 5;

} // end if

//check for five of a kind

if ($numFives == 1){

print "You got five of a kind!<br>\n";

$payoff = 10;

} // end if

**Checking for Straights**

Straights are a little trickier, because there are two possible straights. The player could have the values 1–5 or 2–6. To check these situations, I used two compound conditions.

//check for straights

if (($numVals[1] == 1)

&& ($numVals[2] == 1)

&& ($numVals[3] == 1)

&& ($numVals[4] == 1)

&& ($numVals[5] == 1)){

print "You have a straight!<br>\n";

$payoff = 10;

} // end if

if (($numVals[2] == 1)

&& ($numVals[3] == 1)

&& ($numVals[4] == 1)

&& ($numVals[5] == 1)

&& ($numVals[6] == 1)){

print "You have a straight!<br>\n";

$payoff = 10;

Notice how each if statement has a condition made of several sub-conditions joined by the && operator. The && operator is called a boolean and operator. You can read it as "and." The condition will be evaluated to true only if all the sub conditions are true.

The two conditions are very similar to each other. They simply check the two possible flush situations.

**Printing Out the Results**

The last function in the program prints out variable information to the user. The $cash value describes the user's current wealth. Two hidden elements are used to store information the program will need on the next run. The secondRoll element contains a true or false value indicating whether the next run should be considered the second roll. The cash element describes how much cash should be attributed to the player on the next turn.

function printStuff(){

global $cash, $secondRoll;

print "Cash: $cash\n";

//store variables in hidden fields

print <<<HERE

<input type = "hidden"

name = "secondRoll"

value = "$secondRoll">

<input type = "hidden"

name = "cash"

value = "$cash">

HERE;

} // end printStuff