There are a number of features of the site that are not initially apparent.

* Getting three most recent games for home page
  + The homepage includes a short description of the website, but more importantly, it includes the three most current games. Through PHP, the games with the three most recent release dates are featured on the home page. This is automatically updated if a new game is added with a more recent release date. I designed and implemented the following code to iterate through the array of games and compare dates to find the indexes with the three most recent dates.

<span class=”code codeBlock”>//get index of most recent game

for($i = 0; $i < count($gameData); $i++) {

if(strtotime($gameData[$i][4]) > strtotime($gameData[$firstRecentIndex][4])) {

$firstRecentIndex = $i;

}

}

//get index of second most recent game

for($i = 0; $i < count($gameData); $i++) {

if($i != $firstRecentIndex) {

if(strtotime($gameData[$i][4]) > strtotime($gameData[$secondRecentIndex][4])) {

$secondRecentIndex = $i;

}

}

}

//get index of third most recent game

for($i = 0; $i < count($gameData); $i++) {

if($i != $firstRecentIndex AND $i != $secondRecentIndex) {

if(strtotime($gameData[$i][4]) > strtotime($gameData[$thirdRecentIndex][4])) {

$thirdRecentIndex = $i;

}

}

}</span>

The data for these indexes is then stored in a new array for use on the homepage.

* Side bar images
  + The images of game characters on the sides of site (when the window is big enough) were designed to make them as responsive as possible. Each side of the site has its own <span class=”code”>div</span> element, with classes of <span class=”code”>characterContainerLeft</span> and <span class=”code”>characterContainerRight </span>. Each image’s width is set to 75 pixels, with a bottom padding of 70 pixels. This ensures that the column that they take up is uniform in width and there is adequate spacing in between images. The <span class=”code”>div</span> elements are positioned absolutely, so that they do not interrupt the flow of the page. The left container is positioned with a left value of 2%, and the right container is positioned by using the following code: <span class=”code”>left: calc(98% - 75px);</span>. This allows the right edge of the <span class=”code”>div</span> to be 2% from the right side, mirroring the alignment of the <span class=”code”>div</span> on the left side. Both containers are given <span class=”code”>top: 150px;</span> so that the first image starts below the header. The code that makes the format work so well is that the <span class=”code”>div</span> elements are given heights of <span class=”code”>calc(100% - 150px);</span> and have their overflow property set to hidden. This makes the <span class=”code”>div</span> as tall as the page, and hides any pictures that do not fit on the page. Because this is done dynamically, the window can change size or zoom and more or less images will be shown to accommodate the size of the page. One final feature that keeps the appearance fresh is that the array is shuffled every time a page is loaded, so the images are randomized.
* Alphabetize games on genre page
  + Because future versions of this website could contain many, many games, it made sense to alphabetize the order that they appear on their genre pages. Because all of the game data is stored in a two-dimensional array, it was very difficult to sort. The solution was to turn the indexed array of game data into an associative array. This was accomplished using the following code:

<div class=”code codeBlock”>$gameDataSorted= array();

for($i = 0; $i < count($gameData) - 1; $i++) {

$gameDataSorted[] = array\_combine($headers, $gameData[$i]);

}</div>

This created a new array called <span class=”code”>$gameDataSorted</span> which was an associative array with the keys being the headers of the CSV file. This array could then be sorted by using the following code:

<div class=”code codeBlock”> function compareByName($a, $b) {

return strcmp($a["Name"], $b["Name"]);

}

usort($gameDataSorted, 'compareByName');</div>

This defines the function <span class=”code”>compareByName</span> which compares the values associated with the <span class=”code”>”Name”</span> key. This comparison is then used in the <span class=”code”>usort()</span> function which sorts the array <span class=”code”>$gameDataSorted</span> by the comparison defined in <span class=”code”>compareByName</span>. Then, this sorted array is iterated through in a for each loop, and if the genre of that game matches the genre specified in the <span class=”code”>$\_GET</span> array.

* Game page
* Function for changing color of rating box
  + In order to make the game page more visually appealing, I designed a function to set the background color of the number rating. The following function takes a rating as input and returns an array containing red, green, and blue values for the color. The color is red if the rating is less than 20, and goes from red to yellow between 20 and 70, and goes from yellow to green as it approaches 100.<span class=”code codeBlock”> function ratingGradient($rating) {

//set red content

if($rating < 70) { //rating < 70, red or yellow

$r = 255;

}else { //rating > 70, remove red to make green

$r = 255 - 7 \* ($rating - 63);

}

//set green content

if($rating < 20) { //rating < 20, all red

$g = 0;

}else { //rating > 20, add green to make yellow

$g = 7 \* ($rating - 20);

}

//set blue content

$b = 0;

//return array with rgb values

return [$r, $g, $b];

}</span>

* Getting footer to stick to bottom
* Slider – designed by me
  + It seemed logical to display screenshots in a slider, but I did not want to use someone else’s code. For the labs, I used a slider supplied by Bob and one that I found online, but I never felt that I was truly able to get the slider to do what I wanted it to do. For that reason, I did some research and wrote my own code for a slider. I ended up using a few concepts from a few different sources and combining what I did like from them and removing what I did not. The main function for moving to the next or previous slide is displayed below. It is called with n being 1 or -1, depending on the desired direction.<span class=”code codeBlock”> function moveImg(n) {

//initialize index

var i;

//get all images in slides

var x = document.getElementsByClassName("screenshotSlides");

//get all buttons from slider

var dots = document.getElementsByClassName("dot");

//hide all images

for (i = 0; i < x.length; i++) {

x[i].style.display = "none";

}

//move to next (or previous if n = -1) image

slideIndex += n;

//remove active button class from all buttons

for (i = 0; i < dots.length; i++) {

dots[i].className = dots[i].className.replace(" activeButton", "");

}

//if index is greater than the number of images, go back to start

if (slideIndex > x.length-1) {slideIndex = 0}

//if index is less than 0, go to the end

if (slideIndex < 0) {slideIndex = x.length-1}

//show current image

x[slideIndex].style.display = "block";

//add activeButton class to current slide's button

dots[slideIndex].className += " activeButton";

//clear and reset timer to go to next image after 3 seconds

clearInterval(slidesTimer);

slidesTimer = setInterval('moveImg(1)', 3000);

}</span>There is also a separate function called <span class=”code”>setImg(n)</span>, which sets the slider to a certain index, rather than moving forwards or backwards. It is essentially the same function except the line <span class=”code codeBlock”>slideIndex += n;</span> becomes <span class=”code codeBlock”>slideIndex = n;</span>

* Csv
  + <p>Nearly all of the content on the website is based off of information from a CSV file. Each game is a row in the CSV and is given the following information:</p>

<ul>

<li>Identification number (ID)</li>

<li>Name</li>

<li>Genre</li>

<li>Rating</li>

<li>Release date</li>

<li>Developer</li>

<li>Publisher</li>

<li>Platforms</li>

<li>Age rating</li>

<li>Tags</li>

<li>Summary</li>

<li>Trailer link</li>

</ul>

This CSV is read into the array <span class=”code”>$gameData</span> and is manipulated and accessed on most pages on the website. The content items that are not included in the CSV are images. The solution to how we handled images is explained in the following section.

* Getting images function
  + When creating the first few pages of games, it became clear that we would be displaying a large number of images, and entering every single filename into the CSV file would not be efficient. After a bit of research I was able to design a function to take a path and directory name as parameters and return an array containing all the file paths of that folder. The foundation of this function is the <span class=”code”>FilesystemIterator</span> class. A new instance of the class is created by passing in the path to the desired folder. The class is then manipulated and an array containing the paths of all files in that folder is returned, as outlined in the code below.

This function is used to produce the paths for all images on the website. A folder named <span class=”code”>cover-art</span> contains subfolders of with names corresponding to the game’s ID. For example, the path to the cover art folder for a game with an ID of 1 would be <span class=”code”>/images/cover-art/1</span>. A similar structure is used for screenshots, and the path to the screenshot folder for a game with an ID of 1 would be <span class=”code”>/images/screenshots/1</span>.

* Breakpoints
  + In order to make the website as scalable as possible, multiple breakpoints were put implemented using CSS. On the game page, a break point was implemented to display the cover art and information box in one column when the screen was too small, as shown in the following code.<span class=”code codeBlock”>@media screen and (max-width: 970px) {

.coverArt {

display: block;

text-align: center;

}

.gameInfoContainer {

float: none;

margin: 0.5em auto 1em auto;

}

}</span>Similarly, the sidebar images are not displayed when the screen becomes too small to accommodate them properly, as shown in the code below.<span class=”code codeBlock”>@media screen and (max-width: 1150px) {

.characterContainerLeft, .characterContainerRight {

display: none;

}

}</span>

<span class=”code”></span>