SE 319

Assignment 3 Report

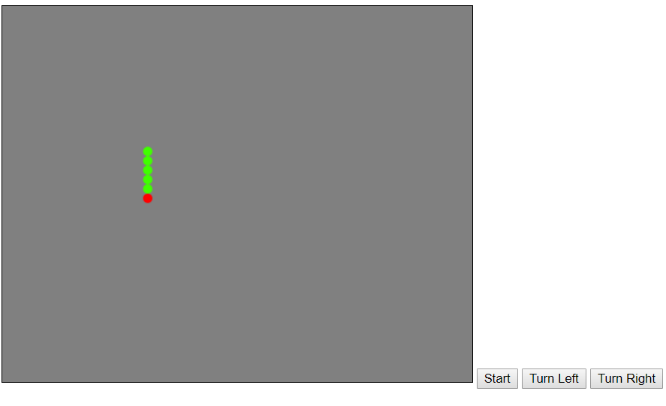
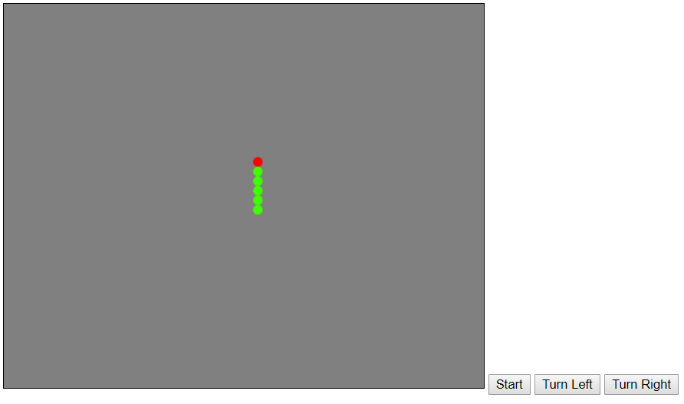
Brad Gannon

**Part 1 (snake.html, snake.js)**

My approach for this part was to reference HTML code that had been provided to us to format the page, then use basic logic in the JavaScript portion to create the game component. For the HTML section, I started by creating the canvas portion via CSS and used a gray background, which I found to be more appealing than the white. After that, I created the dimensions of the canvas and a border like what was given as a sample HTML for the assignment. Lastly, I created the ‘Start’, ‘Turn Left’, and ‘Turn Right’ buttons and linked them to specific functions in the JavaScript file.

For the JavaScript file, I kept the logic very simple. Firstly, I initialized some variables, such as the current direction of the snake, some constants for the dimensions of the canvas, and arrays to keep track of the location of the front of the snake. I chose to use some dots (imported as PNG files) for the snake, as I thought that had more visual appeal than the simple rectangle given in the example output. I started the JavaScript file with an initialization function, which loaded the images and created the snake, followed by setting up a quick delay before the actual game started. The loadImages() and createSnake() functions were very simple, as it simply placed the snake at a given location and set the arrays as the appropriate values. I used a drawing function to update each time the snake moved, which was done by clearing the canvas and placing the updated snake at the new location, all while updating the arrays. I created a gameOver() function, which displays the text “Game over” in the center of the canvas if the front of the snake hits a boundary. The move() function considers what direction the snake is moving and updates the arrays accordingly. The direction the snake is moving is represented via Boolean variables. I implemented a function to check if the head of the snake has reached a border of the canvas, which then called the gameOver() function if necessary. The gameCycle() function checks if the snake is within the boundaries, which then checks if there has been a collision with the border, moves the snake, updates the drawing, and uses the timer to delay slightly. Lastly, I used two functions, leftTurn() and rightTurn(), to update the Boolean values if the button had been clicked. Overall, the logic of this program was to essentially update the variables based on button clicks and re-draw the snake to reflect its movement. Here are outputs of the game:

Before ‘Start’ button has been clicked



Game over

Snake moving left

Snake moving up

Snake moving down

Snake moving right

**Part 2 (hw3.js)**

This part was straightforward, as I simply created four variables, which were inputted by the user when prompted on the console. From there, I passed each variable to its corresponding function and printed the result on the console. For the factorial computation, I used recursion to simply multiply the passed in number times the number minus one, until the number was equal to one. For the sum of the digits, I used the modulus, taking the sum as the number modulus ten, and then dividing the number by ten. For reversing the digits, I turned the number into a string and used the reverse method that is built into JavaScript. Lastly, for the palindrome check, I turned the number into a string and used a for-loop to determine if the first and last digits were equal, then the first and last – 1, and so on. Then I returned a Boolean value as the result. Screenshots of the output are on the next page:

