Explorathon 2018

Using Computer Science to Make Games

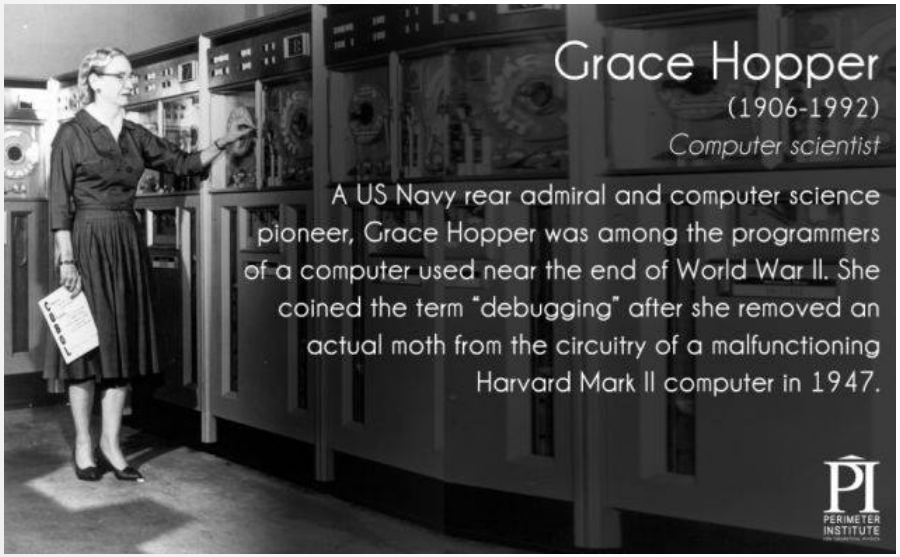
Women in Computer Science

In 1844 a paper was released by an "A.A.L." that discussed a theoretical machine being worked on by mathematician Charles Babbage.

A.A.L. theorized that this machine could one day be able to perform all sorts of complex equations and even create music. A.A.L wrote a set of commands for the still-incomplete machine that would allow it to generate Bernoulli numbers.

It wasn't until 20 after her death that A.A.L. was revealed to be Augusta Ada Lovelace.

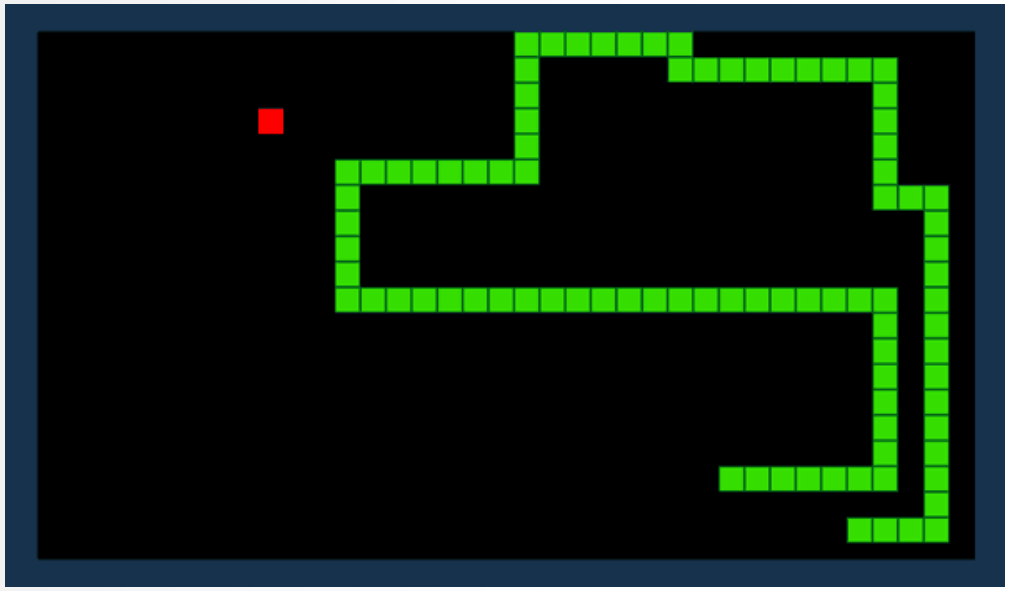
Today she is considered the world's first computer programmer .



Careers in Computer Science

- Software Engineer
- Web Application Developer
- Mobile Application Developer
- Video Game Developer
- ...and so much more!
- Today we will show you what its like to develop a video game and write real code!

Our Inspiration



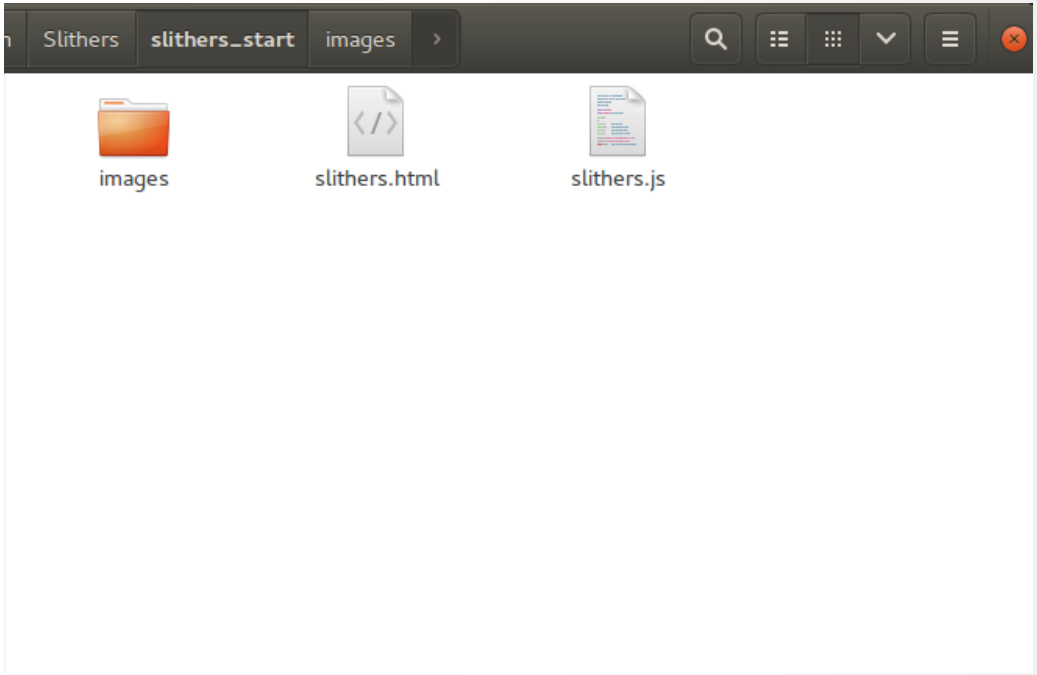
The game we will create today is based on the classic video game Snake. We have also supplied a finished version of our game.

Crash Course in Writing Code

- We are going to write a lot of code very quickly today. Here are some rules to help you along:
  - It’s called a “language” because we are literally talking to a computer
  - If you forget a symbol or misspell a word, the program will show an error or do something weird, because the computer doesn’t understand
  - We can help you find some of the symbols on your keyboard since we don’t use them often

Open the Starter Project

- Open up the file browser
- Find the slithers_start folder in the flash drive
- The initial setup of the project comes with:
  - An HTML file that calls our code, and displays our game
  - A JavaScript file that does all the work!
  - A folder with all of our game images



Our Simple HTML File

slithers.js × slithers.html ×

```
1   <html>
2   <head>
3     <script type="text/javascript" src="slithers.js"></script>
4   </head>
5   <body onload="onLoad();" style="text-align: center; zoom: 300%">
6     <img src="images/apple_piece.png" id="apple_piece" style="display: none;"/>
7     <img src="images/body_piece.png" id="body_piece" style="display: none;"/>
8     <img src="images/head_piece_left.png" id="head_piece" style="display: none;"/>
9     <img src="images/wall_piece.png" id="wall_piece" style="display: none;"/>
10    <img src="images/floor_piece.png" id="floor_piece" style="display: none;"/>
11    <canvas id="canvas" width="300px" height="200px"></canvas>
12  </body>
13  </html>
```

- The canvas element is what we draw our game on
- A function named onLoad() is called on the body during the “onload” event
- We also load the different images we are going to use

Our onLoad() Event

- Toward the middle of our JavaScript file, we can see our onLoad() function
- The onLoad() function initializes our game loop
- The gameLoop() function calls methods to handle input, update the position of our assets, and draw the game

slithers.js | slithers.html

```
124  /**
125   * Sets up our game. Stores a reference to the canvas, sets up key listening, and starts our game loop.
126   */
127  function onLoad() {
128    // Setup our game loop to listen for a change every 16 milliseconds.
129    window.setInterval(gameLoop, 16);
130
131    // We draw via the canvas's 2d context, so we store a reference for drawing.
132    canvas = document.getElementById("canvas").getContext("2d");
133
134    // We load in all of our images from the HTML file.
135    wallImage = document.getElementById("wall_piece");
136    floorImage = document.getElementById("floor_piece");
137    headImage = document.getElementById("head_piece");
138    bodyImage = document.getElementById("body_piece");
139    appleImage = document.getElementById("apple_piece");
140
141    // Every time a key is pressed, store it to be processed in the game loop.
142    document.addEventListener('keydown', function (event) {
143      playerKeyPress = event.keyCode;
144    });
145
146    // Make sure to set the key press to false when it's done being "pressed".
147    document.addEventListener('keyup', function (event) {
148      playerKeyPress = 0;
149    });
150
151    newGame();
152  }
```

- Then, it prepares our canvas, and gets ready for us to draw
- Next it loads all of the different images we are going to use
- Lastly, this code does is listen for key presses, and records the unique code

Our Board

- We store our board in a 2 dimensional array
- This makes it really easy to walk through the array and draw it on our canvas

slithers.js

```
/**
 * We define the structure of the board in a method so a new one can be created whenever a new game begins. A constant for
 * the board's definition would require us to copy to a board array each time the game starts, creating a new instance of
 * the data is simpler.
 */
function getNewBoard() {
  board =  [
    [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1],
    [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]
  ];
```

Let's Draw Something!

- The first thing we are going to do is draw our board
- Find the function called `drawBoard()` at the bottom of the file
- We need to choose/draw different images for the floor and the walls based on the 2 dimensional array
- Since our images are 8 pixels by 8 pixels, we need to multiply our x and y positions by 8

Filling in drawBoard()

- After we have written the highlighted code below in drawBoard(), open the HTML file in the browser

```
function drawBoard() {
  for (var x = 0; x < board[0].length; x++) {
    for (var y = 0; y < board.length; y++) {
      var value = board[y][x];
      var image;
      if (value == -1) {
        image = wallImage;
      }
      else {
        image = floorImage;
      }
      drawImage(image, x * 8, y * 8, 0, false);
    }
  }
}
```

Now We Draw Our Snake

- We can store a lot of different information under our snake as properties, and change them as the snake moves
- The properties help to keep track of where they are on the board, and how many segments there are
- Our snake properties are already loaded with their starting positions
- We access the properties like this: snake[0].x

```
function resetSnake() {
  return [
    {
      x: 23,
      y: 17
    },
    {
      x: 24,
      y: 17
    },
    {
      x: 25,
      y: 17
    },
    {
      x: 26,
      y: 17
    }
  ];
}
```

How to Draw the Snake

- Now that we know where the snake's positions are stored, we can write the drawing code!
- Inside the drawGame() function, we are going to write the highlighted lines shown below
- Our head image should be drawn for the very first segment
- After we write this code, we can refresh our HTML file in the browser

```
for (var segment = 0; segment < snake.length; segment ++) {
  var image;
  if (segment == 0) {
    image = headImage;
  }
  else {
    image = bodyImage;
  }
  drawImage(image, snake[segment].x * 8, snake[segment].y * 8, 0, false);
}
```

What does drawImage() do?

- To draw our board and snake segments, we used a helper function called drawImage()
- This function takes your image, where you want it drawn in the canvas, and a couple other options to correctly draw the image
- We passed in the snake image we wanted, and the (x,y) position on the board

```
320
321  /**
322   * Draws an image to the canvas at the specified location with rotation and mirroring.
323   */
324  function drawImage(image, x, y, angle, mirror) {
325    canvas.translate(x + (image.width / 2), y + (image.height / 2));
326
327    canvas.rotate(angle * Math.PI / 180);
328
329    if (mirror) {
330      canvas.scale(-1, 1);
331    }
332    else {
333      canvas.scale(1, 1);
334    }
335
336    canvas.drawImage(image, -4, -4);
337
338    canvas.setTransform(1, 0, 0, 1, 0, 0);
339  }
```

Drawing the Apple

- Our apple is very similar to our snake
- The apple attributes are also saved in a game state variable towards the top of the file, with it's x and y positions
- After we add the following line of code at the bottom of our drawGame() function, we can refresh our browser

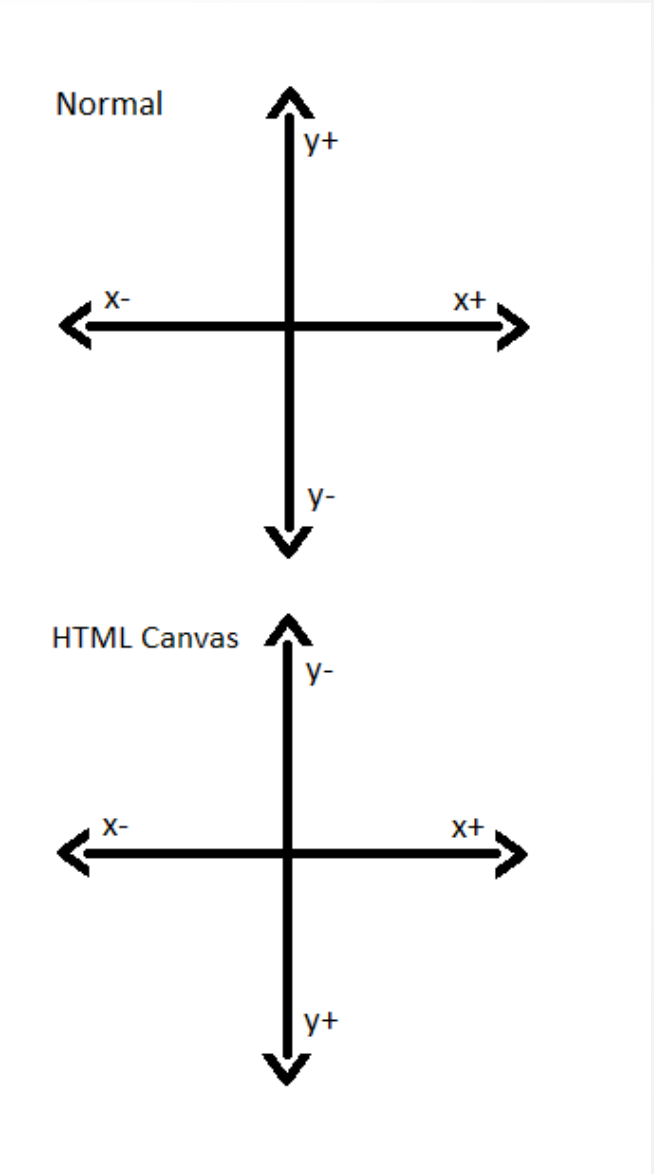
```
for (var segment = 0; segment < snake.length; segment ++) {
  var image;
  if (segment == 0) {
    image = headImage;
  }
  else {
    image = bodyImage;
  }
  drawImage(image, snake[segment].x * 8, snake[segment].y * 8, 0, false);
}

drawImage(appleImage, applePosition.x * 8, applePosition.y * 8, 0, false);
```

Time to Make Things Move

- In Snake, we can move our snake’s head all around the board, and the body follows
- The next step is to check key presses, and change our snake’s position based on that
- Before we write our code, we need to review Cartesian coordinates

Coordinates - Quick Review

- We can move things around on the screen according to x and y positions
- Because positive y is downward, when we press **down** we want to **increase** our snake's y property
- When we press **up**, we want to **decrease** our snake's y property
- When we press **right**, we want to **increase** our snake's x property
- When we press **left**, we want to **decrease** our snake's x property



Checking Key Presses

- At the top of our file are a lot of variables
- Some of our variables - constants - are numbers that never change
- We are using all of these constants to track which keyboard key was pressed

```
/**
 * These are our constant values.
 *
 * Constants are a necessity to good programming as they allow later changes to be made system wide as long as the constant is used
 * instead of "magic values/numbers" across the code.
 */
// ASCII key codes stored as constants for readability.
var KEY_PRESS_LEFT = 37;
var KEY_PRESS_RIGHT = 39;
var KEY_PRESS_DOWN = 40;
var KEY_PRESS_UP = 38;
var KEY_PRESS_ENTER = 13;
```

Make Things Move

- Go to the handleInput() function above the drawGame() function
- We are going to determine which key is pressed, then change our snake's position based on that with the code below
- The ++ and -- are shorthand for adding 1 to our variable, e.g. xDelta = xDelta + 1;

```
var xDelta = 0;
var yDelta = 0;

if (playerKeyPress == KEY_PRESS_LEFT) {
  xDelta --;
}
if (playerKeyPress == KEY_PRESS_RIGHT) {
  xDelta ++;
}
if (playerKeyPress == KEY_PRESS_DOWN) {
  yDelta ++;
}
if (playerKeyPress == KEY_PRESS_UP) {
  yDelta --;
}
```

Applying Our New Position

- If both xDelta and yDelta are 0, then that means our snake isn't moving, so we check their values before doing anything
- We use a helper function to move the snake body, then apply the new position to the snake's head
- Once we write this new code below in handleInput() in between the given check of xDelta and yDelta, we can refresh our browser

```
if (xDelta != 0 || yDelta != 0) {
  moveSnakeBody(xDelta, yDelta);
  snake[0].x += xDelta;
  snake[0].y += yDelta;
}
```

It’s a Little Jumpy

- We only want our snake moving 1 piece length at a time
- Once a button is pressed and we record the xDelta and yDelta, we should reset the playerKeyPress variable so it stops saying that the key was pressed
- Add this line below and it should be a much smoother snake movement

```
playerKeyPress = 0;

if (xDelta != 0 || yDelta != 0) {
  moveSnakeBody(xDelta, yDelta);
  snake[0].x += xDelta;
  snake[0].y += yDelta;
}
```

How moveSnakeBody() Works

- The code to move the snake's body looks a little tricky, but its actually pretty clever
- We start from the end of the snake, and move each piece into the position of the piece before it
- That way there's no extra calculations, just a copy + paste of each segment!

slithers.js × slithers.html ×

```
226   /**
227    * Steps through our snake's body, and moves each segment forward by one space.
228    */
229   function moveSnakeBody() {
230     for (var x = snake.length - 1; x >= 1; x --) {
231       snake[x].x = snake[x - 1].x;
232       snake[x].y = snake[x - 1].y;
233     }
234   }
235
```

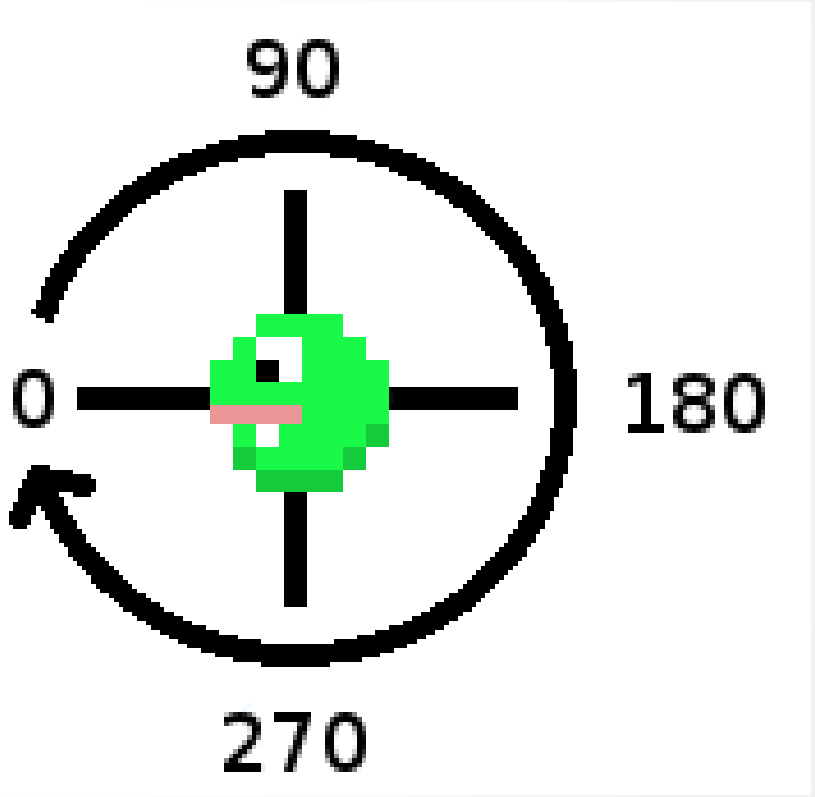
Changing Our Snake's Head

- Our snake head looks weird! He should look in the direction he's heading
- We need to keep track of the last position during key presses
- Add the yellow highlighted code to the handleInput() function

```
if (playerKeyPress == KEY_PRESS_LEFT) {
  lastSnakeDirection = "left";
  xDelta --;
}
if (playerKeyPress == KEY_PRESS_RIGHT) {
  lastSnakeDirection = "right";
  xDelta ++;
}
if (playerKeyPress == KEY_PRESS_DOWN) {
  lastSnakeDirection = "down";
  yDelta ++;
}
if (playerKeyPress == KEY_PRESS_UP) {
  lastSnakeDirection = "up";
  yDelta --;
}
```

Rotate Our Snake Head

- Our drawImage() helper function lets us specify the number of degrees to rotate our image, or mirror our image
- Right now we're drawing our snake head at 0 degrees, but depending on the lastSnakeDirection, we need to calculate a new rotation angle



Code for Snake's Rotation

```
for (var segment = 0; segment < snake.length; segment ++) {
  var image;
  var angle = 0;
  if (segment == 0) {
    image = headImage;
    if (lastSnakeDirection == "up") {
      angle = 90;
    }
    else if (lastSnakeDirection == "right") {
      angle = 180;
    }
    else if (lastSnakeDirection == "down") {
      angle = 270;
    }
  }
  else {
    image = bodyImage;
  }

  drawImage(image, snake[segment].x * 8, snake[segment].y * 8, angle, false);
}
```

We can refresh the browser after adding this highlighted code in the drawGame() function

Facing Right

- That doesn't look right!
- We don't want to rotate the snake's head for him to face right, instead we want a mirror of his headImage
- We can refresh after updating this code

```
for (var segment = 0; segment < snake.length;
segment ++) {
  var image;
  var angle = 0;
  var mirror = false;
  if (segment == 0) {
    image = headImage;
    if (lastSnakeDirection == "up") {
      angle = 90;
    }
    else if (lastSnakeDirection == "right") {
      mirror = true;
    }
    else if (lastSnakeDirection == "down") {
      angle = 270;
    }
  }
  else {
    image = bodyImage;
  }

  drawImage(image, snake[segment].x * 8,
snake[segment].y * 8, angle, mirror);
}
```

Eat & Regenerate Apple

- Next, we want to make the apple disappear if the head overlaps the apple's position
- When the apple gets eaten, it needs to appear in another random spot
- Our checkAppleEaten() helper function will randomly generate a new apple if the previous one gets eaten
- We need to add this line in the handleInput() code

```
if (xDelta != 0 || yDelta != 0) {
  moveSnakeBody(xDelta, yDelta);
  checkAppleEaten(xDelta, yDelta);
  snake[0].x += xDelta;
  snake[0].y += yDelta;
}
```

checkAppleEaten()

- The checkAppleEaten() function will compare the (x,y) position of the apple and the snake's head
- If they have the same (x,y) coordinates, then the apple has been eaten, and we should regenerate another one

```
226  /**
227   * Checks the position of our snake's head and the apple, returns TRUE if they're at the same position.
228   */
229  function checkAppleEaten(xDelta, yDelta) {
230    if (snake[0].x + xDelta == applePosition.x && snake[0].y + yDelta == applePosition.y) {
231      newAppleLocation();
232      return true;
233    }
234    return false;
235  }
236
```

Grow the Body

- When we eat an apple, our body should grow by 1 segment
- To keep with our trick, we need to keep track of the last segment, and assign the new segment to that position
- We need to modify our code in handleInput() for moving our snake and eating our apple

```
if (xDelta != 0 || yDelta != 0) {
  var lastSegmentX = snake[snake.length - 1].x;
  var lastSegmentY = snake[snake.length - 1].y;
  moveSnakeBody();
  if (checkAppleEaten(xDelta, yDelta)) {
    growSnakeBody(lastSegmentX, lastSegmentY);
  }
  snake[0].x += xDelta;
  snake[0].y += yDelta;
}
```

Our growSnakeBody() function

- The growSnakeBody() function will take the (x,y) values we saved for the very last piece of the snake before we moved the body
- Then it will add a new segment onto the end of the snake array with that saved (x,y) position

```
216  /**
217   * Push a new segment onto the end of our snake array.
218   */
219  function growSnakeBody(newX, newY) {
220    snake.push({
221      x: newX,
222      y: newY
223    });
224  }
```

Wrap Around the Board

- There are a couple holes in the wall!
- When our snake goes through a hole, we want him to appear on the other side of the board
- Add the code below to make the snake wrap around the board at the bottom of the handleInput() function

```
if (snake[0].x == -1 && lastSnakeDirection == "left") {
  snake[0].x = board[0].length - 1;
}
if (snake[0].x == board[0].length &&
  lastSnakeDirection == "right") {

  snake[0].x = 0;
}
```

Steps to Finish Your Game

- Prevent the player from moving the snake in the exact opposite direction its currently heading
- Cause a gameOver when the snake's head hits a wall
- Cause a gameOver when the snake's head touches a piece of his body
- Show Game Over text
- Force the snake to move after a certain amount of time if no keys were pressed

Please keep an eye on https://github.com/hillylew/Slithers for additions to this tutorial

Thank you!

- Included on your flash drive are the following programs
  - Notepad++ – a text editor with syntax highlighting, good for writing code
  - Gimp – A free and open source image editor
  - Audacity – A free sound editing program
  - Reaper – A digital audio workstation program providing convenient licensing
  - Blender – A free and open source 3d modeling tool
  - Tiled – A free and open source Tile Map editing tool, used for creating complex levels
- Please visit https://github.com/hillylew/Slithers for future updates to this project or to help in its development. Slithers is a GPL'd open source project that anyone can contribute to!
- We hope you enjoyed Slithers!