Comp 125 - Visual Information Processing

Spring Semester 2019 - Week 12 - Friday

Dr Nick Hayward

HTML Canvas

abstracted function - draw a well-known mouse

- abstract the drawing of the well-known mouse
  - *e.g. vary size according to a relative scale for each circle*

```
// define function to draw mouse - x & y = centre of head, radius = head size, color1 = he
function mouse(x, y, radius, fill, color1, color2) {
  //draw left ear
  circle(Math.floor(x/1.28), Math.floor(y/1.3), Math.floor(radius/1.8), fill, color2);
    //draw right ear
    circle(Math.floor(x*1.22), Math.floor(y/1.3), Math.floor(radius/1.8), fill, color2);
    //draw head
  circle(x, y, radius, fill, color1);
}
```

HTML Canvas

abstracted function - draw a well-known mouse - part 2

- we might define the base mouse size as follows

```
// base small size for mouse
mouse(150, 130, 34, true, 'DarkRed', `black`);
```

- then scale by a factor of 2 for a large mouse size

```
// scale by 2 - x, y & radius
mouse(300, 260, 68, true, 'DarkBlue', `green`);
```

- we may also now specify colours for the mouse as well
  - `color1` *for the head*
  - `color2` *for the ears*
- Example - variant mouse colours
  - *http://linode4.cs.luc.edu/teaching/cs/demos/125/drawing/basic-animation/animation3.2/*

HTML Canvas - basic animations

random movement - part 4

- to animate a shape in a random direction and path
  - *create a custom function to `update` this position*
- function will randomly change the `x` and `y` coordinates
  - *create effect of shape moving around the canvas*

```
// update the x and y coordinates for shape animation
function update(coord) {
    var offset = Math.random()*5-2;
    coord += offset;

    if (coord > 400) {
        coord = 0;
    }
    if (coord < 0) {
        coord = 0;
    }

    return coord;
}
```

- check if coordinates go beyond the width or height of the canvas
  - *if yes, reset back to the top using a value of 0*

HTML Canvas - basic animations

random movement - part 5

- then use this `update` function to radomly animate the shape
  - *use standard `setInterval` timer*

```
// animate our well known mouse
setInterval(function() {
    context.clearRect(0, 0, 400, 400);

  // 1. base small size for mouse
  circle(x, y, 40, true, 'green');
    x = update(x);
    y = update(y);

}, 20);
```

- start by clearing `context` for defined size of `canvas`
- then draw required shape to animate per frame
- `x` and `y` coordinates will be random by calling the `update` function
  - *call function with previous frame's `x` and `y` coordinates*
- Example - random movement and animation
  - *http://linode4.cs.luc.edu/teaching/cs/demos/125/drawing/basic-animation/animation3.3/*

HTML Canvas - add an image

basic image rendering - part 1

- draw image to canvas using `drawImage()` method

```
// image drawn full size from source to x & y in destination
context.drawImage(image, dx, dy)
// image drawn with scaled width and height for destination
context.drawImage(image, dx, dy, dw, dh)
// image drawn with source cropped...
context.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)
```

- `d` represents the destination canvas
- `s` represents the source image

HTML Canvas - add an image

basic image rendering - part 2

- add a static image using `drawImage()` method
- use `Image()` constructor to create an image object
- use `img` object to set `src` for image file
  - *local and remote URL for image is OK*
- draw image to context
  - *`context.drawImage(image, dx, dy)`*

```
// 1. define optional image size
var img = new Image();

// image source file
img.src = './assets/images/philae1.jpg';

img.onload = function() {
  context.drawImage(img, 0, 0);
}
```

- image is not scaled to canvas width and height
- Example - draw image to canvas from local file
  - *http://linode4.cs.luc.edu/teaching/cs/demos/125/drawing/basic-image/basic1/*

HTML Canvas - add an image

basic image rendering - part 3

- draw image to canvas with scaled source image
  - *`context.drawImage(image, dx, dy, dw, dh)`*

```
// 1. define optional image size
var img = new Image();

// image source file
img.src = './assets/images/philae1.jpg';

img.onload = function() {
    // context.drawImage(image, dx, dy, dw, dh)
  context.drawImage(img, 0, 0, 116, 77);
}
```

- Example - draw image to canvas from local file - dw & dh
  - *http://linode4.cs.luc.edu/teaching/cs/demos/125/drawing/basic-image/basic2/*

HTML Canvas - add an image

basic image rendering - part 4

- draw part of the source image
- define source x, y, width and height
  - *i.e. crop part of source image*
- define destination x, y, width and height
  - *i.e. where to draw image on canvas*
  - *& scaled size on canvas*

```
// image source file
img.src = './assets/images/philae1.jpg';

img.onload = function() {
    // context.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)
  context.drawImage(img, 200, 200, 232, 144, 0, 0, 464, 288);
}
```

- Example - draw image to canvas from local file - `dw` & `dh` plus source crop
  - *http://linode4.cs.luc.edu/teaching/cs/demos/125/drawing/basic-image/basic3/*

HTML Canvas - Canvas interaction

move a ball with keyboard controls

- create a new example to allow a user to move a ball
  - *move ball around canvas using keyboard controls*
- requirements include
  - *need to draw a **ball***
  - *listen for specific keypress commands, e.g UP, DOWN, LEFT, RIGHT*
  - *then update animation of ball to reflect each keypress*
- allowing a user to directly control animation of shape on canvas
- setup our initial example with a `canvas` and `context`
  - *use `Ball` constructor and **prototype** methods*
  - *start to add logic to control the `ball`, update animation...*
  - *extend the prototype for user control of the `ball` object*

HTML Canvas - Canvas interaction

keyboard listeners

- add listeners to the `canvas` for specifc keypress events
  - *e.g. up, down, `left`, and `right`*

```
// add event listener for keypress - e.g. up arrow key...
window.addEventListener('keydown', function (event) {
    // get code for key presses
  var key = event.keyCode;
  console.log("key pressed = " + key);
  ball.userControl(key);
})
```

- each keypress event returns a unique code
  - *use code to identify key pressed by user*
  - *37 = LEFT arrow*
  - *38 = UP arrow*
  - *39 = RIGHT arrow*
  - *40 = DOWN arrow*
- call `userControl()` method for each keypress

HTML Canvas - Canvas interaction

extend Ball prototype - `userControl()`

```
// 4. update prototype - user control
Ball.prototype.userControl = function( key ) {
  // key - UP arrow
  if (key === 38) {
    this.xSpeed = 0;
    this.ySpeed = -10;
    context.clearRect(0, 0, 400, 400);
    ball.draw();
    ball.move();
  }
};
```

- conditional check for key code - `38` = UP arrow
  - *`x` set to `0` to prevent horizontal move*
  - *`y` set to `-10` to move up canvas*
  - *canvas cleared to allow animation frames to be drawn*
  - *call **prototype** method `draw()` on `ball` object*
  - *call **prototype** method `move()` on `ball` object*
- Example - move ball with keyboard control
  - *http://linode4.cs.luc.edu/teaching/cs/demos/125/drawing/basic-game/basic-ball-move1/*

HTML Canvas - Canvas interaction

update `move()` method

- update `move()` to check `canvas` boundaries
- stop `ball` from leaving canvas

```
// check ball relative to boundaries - canvas edge
  if (this.x < 0) {
      this.x = canvas.width;
  } else if (this.x > canvas.width) {
      this.x = 0;
  } else if (this.y < 0) {
      this.y = canvas.height;
  } else if (this.y > canvas.height) {
    this.y = 0;
  }
```

- Example - update `move()` to check canvas boundaries
  - *http://linode4.cs.luc.edu/teaching/cs/demos/125/drawing/basic-game/basic-ball-move2/*

HTML Canvas - Canvas interaction

abstract width and height

- canvas height and width need to be used throughout JS logic
  - *abstract to variables*

```
// define canvas width and height
var cHeight = canvas.height;
var cWidth = canvas.width;
```

References

- W3Schools - HTML5
  - *canvas element*