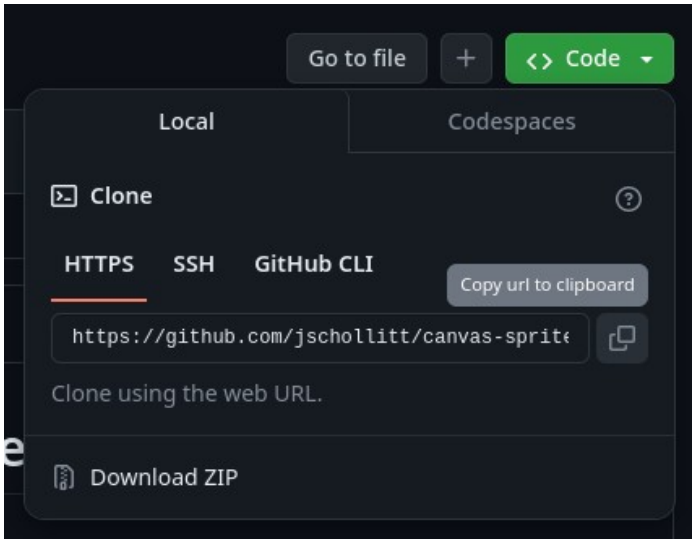
ISCG6420 Semester 1 2024

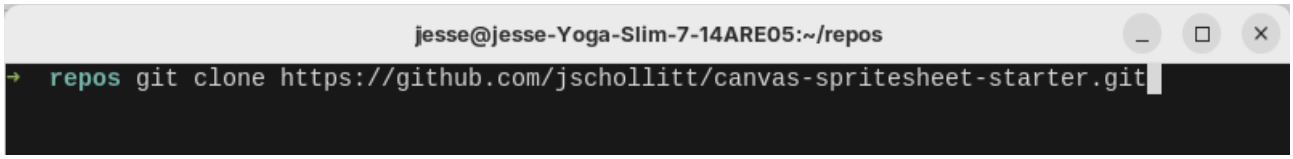
Exercise: Advanced Canvas

These instructions will take you through 3 exercises demonstrating advanced canvas animation techniques.

Open Exercise Starter

Clone the content from the starter project repository:
Link: https://github.com/jschollitt/canvas-spritesheet-starter
Complete version: https://github.com/jschollitt/ISCG6420-2024-S1/tree/main/week9





Open the project folder in VSCode, and open "week9.html". Note the ID for each canvas:

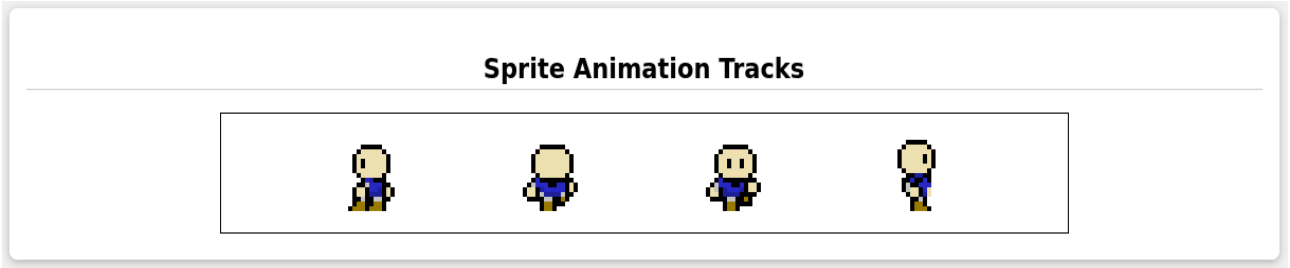
```
<!-- Section 1 -->
<section class="txt-center">
    <h2 class="underline">Sprite Animation Tracks</h2>
    <canvas id="ex1canvas" width="800" height="100"></canvas>
</section>

<!-- Section 1 -->
<section class="txt-center">
    <h2 class="underline">Sprite Flip Speed</h2>
    <canvas id="ex2canvas" width="800" height="382"></canvas>
</section>

<!-- Section 1 -->
<section class="txt-center">
    <h2 class="underline">Input-based Track Selection</h2>
    <canvas id="ex3canvas" width="800" height="800"></canvas>
</section>
```

Open **"js/week9.js"** for editing. Each exercise has its own function to contain all the code for a specific canvas.

```
Example1();
Example2();
Example3();

function Example1() {···
}

function Example2() {···
}

function Example3() {···
}
```

Starting code has been provided for each exercise.

Exercise 1 – Sprite Tracks



Exercise 1 demonstrates using four different sprite animation tracks from one sprite sheet.

At the bottom of the Exercise 1 function is a function called **"Character()"**. This function will create a character object and return it.

Inside the function, return an empty object:

```
// Character object creator function
function Character(spritesheet, spriteSize, spriteFrames, position) {
    return {

    };
}
```

Inside the object, declare the following object members:

- **spriteSheet** equal to **spritesheet** (parameter)
- **spriteFrameSize** equal to **spriteSize** (parameter)
- **spriteFrames** equal to **spriteFrames** (parameter)
- **animationFrame** equal to **0** (index of track)
- **frameTime** equal to **125** (milliseconds)

- timeSinceLastFrame equal to 0 (starting time)
- position equal to position (parameter)

```
return {
    spriteSheet: spritesheet,       // image containing the sprites
    spriteFrameSize: spriteSize,    // dimensions of the sprites in the spritesheet
    spriteFrames: spriteFrames,     // 3d array.
                                    // X = animation track, Y = animation frame, Z = coords of frame

    animationFrame: 0,              // current frame in animation to draw
    frameTime: 125,                 // milliseconds to wait between animation frame updates
    timeSinceLastFrame: 0,          // track time since the last frame update was performed

    position: position,               // position of the character (X, Y)
```

NOTE: Object member values are declared using a colon (:) instead of equals (=), and members are separated using a comma (,) with no semi-colon (;) on the end of each line.

Underneath the members, declare two member functions: update(tick) and draw(context):

```
  position: position,               // position of the character (X, Y)

  update(tick) {···
  },

  draw(context) {···
  },
```

Inside the update function add the following code:

```
update(tick) {
    // increase time keeper by last update delta
    this.timeSinceLastFrame += tick;
    // check if time since last frame meets threshold for new frame
    if (this.timeSinceLastFrame >= this.frameTime) {
        // reset frame time keeper
        this.timeSinceLastFrame = 0;

        this.animationFrame = (this.animationFrame + 1) % this.spriteFrames.length;
    }
},
```

This code increments the frame time tracker by the update delta tick, compares it to the animation time threshold (frameTime), and if ready, resets the timer and updates the animation frame to the next index (wrapped).

Inside the draw function add the following code:

```
draw(context) {
    context.drawImage(
        this.spriteSheet,
        this.spriteFrames[this.animationFrame][0],
        this.spriteFrames[this.animationFrame][1],
        this.spriteFrameSize[0],
        this.spriteFrameSize[1],
        this.position[0],
        this.position[1],
        this.spriteFrameSize[0],
        this.spriteFrameSize[1]
    );
},
```

Save your work. Character object definition complete.

Inside the Example1 init function, add the following code:

```
function init() {
    console.log("init");
    canvas = document.getElementById('ex1canvas');
    ctx = canvas.getContext('2d');

    character1 = Character(
        characterSpriteSheet,
        [64, 64],
        [[0, 64], [64, 64], [128, 64], [192, 64]],
        [108, 18]
    );
```

Check the name of the Canvas 1 ID matches the value in "week9.html".

Create three more character objects for the variables: character2, character3, character4.

Use these parameter values for them:

- character2
  - Sprite Sheet: characterSpriteSheet
  - Sprite Size: [64, 64]
  - Sprite Frames: [[0, 0], [64, 0], [128, 0], [192, 0]] (this is a 2D array)
  - Canvas Position: [281, 18]

- character3
  - Sprite Sheet: characterSpriteSheet
  - Sprite Size: [64, 64]
  - Sprite Frames: [[256, 0], [320, 0], [384, 0], [448, 0]](this is a 2D array)
  - Canvas Position: [454, 18]

- character4
  - Sprite Sheet: characterSpriteSheet
  - Sprite Size: [64, 64]
  - Sprite Frames: [[256, 64], [320, 64], [384, 64], [448, 64]] (this is a 2D array)
  - Canvas Position: [627, 18]

Complete the init function by calling "window.requestAnimationFrame(run);".

Your init function should look like this:

```
    // initialise canvas and game elements
function init() {
        console.log("init");
        canvas = document.getElementById('ex1canvas');
        ctx = canvas.getContext('2d');

        character1 = Character(
            characterSpriteSheet,
            [64, 64],
            [[0, 64], [64, 64], [128, 64], [192, 64]],
            [108, 18]
        );

        character2 = Character(
            characterSpriteSheet,
            [64, 64],
            [[0, 0], [64, 0], [128, 0], [192, 0]],
            [281, 18]
        );

        character3 = Character(
            characterSpriteSheet,
            [64, 64],
            [[256, 0], [320, 0], [384, 0], [448, 0]],
            [454, 18]
        );

        character4 = Character(
            characterSpriteSheet,
            [64, 64],
            [[256, 64], [320, 64], [384, 64], [448, 64]],
            [627, 18]
        );

        window.requestAnimationFrame(run);
    }
```

Inside the update function, call each character object's update function with "tick" passed as a parameter:

```
function update() {
    character1.update(tick);
    character2.update(tick);
    character3.update(tick);
    character4.update(tick);
}
```

Inside the draw function, clear the screen using clearRect, then call each character object's draw function with "ctx" passed as a parameter:

```
function draw() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    character1.draw(ctx);
    character2.draw(ctx);
    character3.draw(ctx);
    character4.draw(ctx);
}
```

Inside the run function, add the following code:

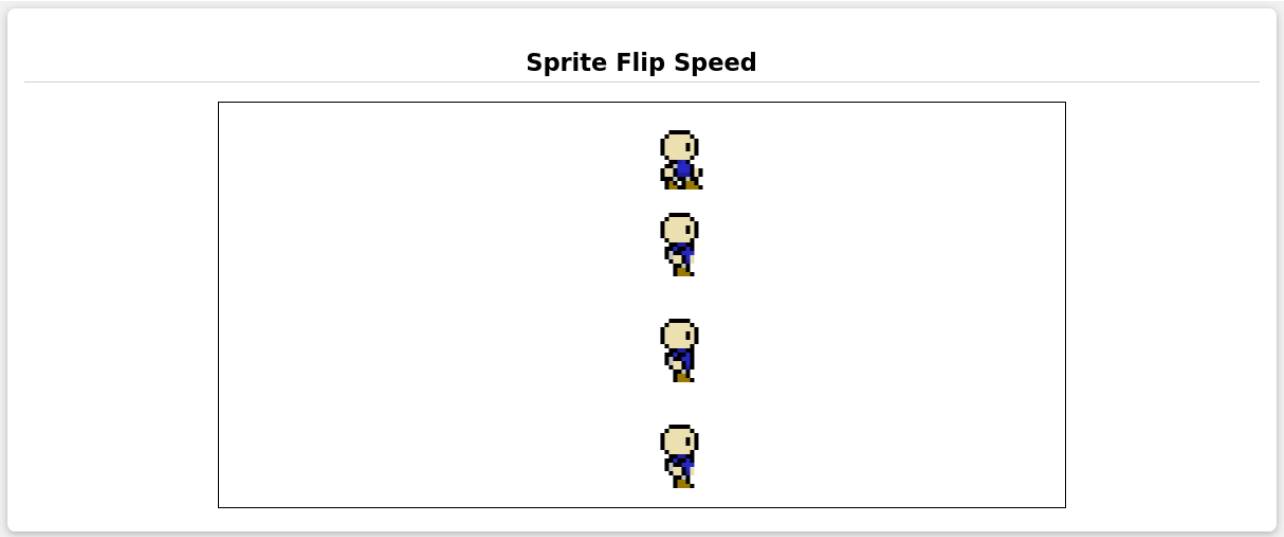
```
// Game loop function
function run(timeStamp) {
    // calculate time delta since last update
    tick = (timeStamp - lastTimeStamp);
    // save current time for next loop
    lastTimeStamp = timeStamp;

    update(tick);
    draw();

    window.requestAnimationFrame(run);
}
```

This function calculates the time since the last loop, calls the update and draw functions, then requests to have the run function called recursively.

Save your work and test the ouput. You should have your four character objects being drawn to the screen with their spritesheet animation frames cycling through a different animation track.

Exercise 1 complete.

Exercise 2 – Animation speed and motion



Exercise 2 demonstrates how to move a sprite-animated object, and the effect animation speed has on the appearance of motion.

Inside the "Example2" function, at the bottom is another Character() function. Add a member to the returned object, called "direction" and set its value to "[10, 0]"

```
animationFrame: 0,              // current frame in animation to draw
frameTime: frameTime,                 // milliseconds to wait between animation frame u
timeSinceLastFrame: 0,          // track time since the last frame update was performed

position: position,               // position of the character (X, Y)
direction: [0.2, 0],
```

Inside the update() function, add the following code after the if statement block:

```
if (this.timeSinceLastFrame >= this.frameTime) {
    // reset frame time keeper
    this.timeSinceLastFrame = 0;

    this.animationFrame = (this.animationFrame + 1) % this.spriteFrames.length;
}

this.position[0] = (this.position[0] + (this.direction[0] * tick)) % canvas.width;
```

This calculates how far left the character should move for the length of time since last update, and uses modulo to wrap the position from the end of the canvas back to the beginning.

In the init() function, update all four characters to use the following animation track:

- [[256, 64], [320, 64], [384, 64], [448, 64]]

Set the character positions to:

- Character 1: [0, 18]
- Character 2: [0, 100]
- Character 3: [0, 200]
- Character 4: [0, 300]

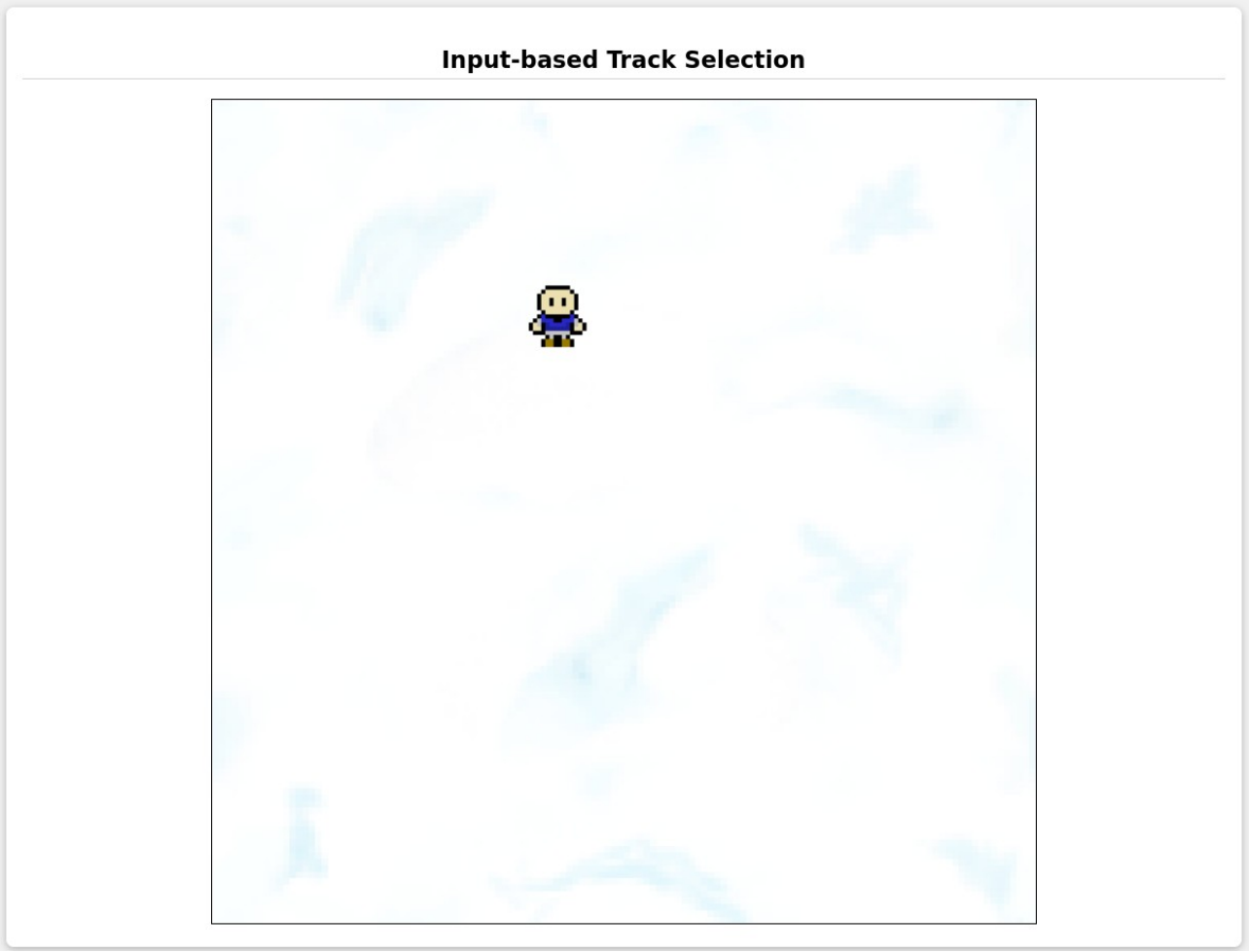
Add a fifth parameter to each character to serve as the animation frame time milliseconds:

- Character 1: 80
- Character 2: 120
- Character 3: 160
- Character 4: 200

Save your work and test the output. Canvas 2 should have four characters distributed vertically, with each moving right at the same velocity, but with different animation speeds.

Exercise 2 complete.

Exercise 3 – User Input



Exercise 3 implements animation track switching and user input.

Inside the Example3 function, inside the init() function is a character object declaration. Unlike exercise 1 and 2, this character has been modified to accept more than one animation track.

Add the following missing tracks to the character:

- Walk left: [0, 0], [64, 0], [128, 0], [192, 0]
- Walk down: [256, 0], [320, 0], [384, 0], [448, 0]
- Walk left: [0, 64], [64, 64], [128, 64], [192, 64]

```
[ // main character set
    [ // walk up track
        [0, 0], [64, 0], [128, 0], [192, 0]
    ],
    [ // walk down track
        [256, 0], [320, 0], [384, 0], [448, 0]
    ],
    [ // walk left track
        [0, 64], [64, 64], [128, 64], [192, 64]
    ],
    [ // walk right track
        [256, 64], [320, 64], [384, 64], [448, 64]
    ],
],
```

Underneath the character object declaration, add two event listeners for the "keydown" and "keyup" events. Have each listener callback to the "doKeyDown()" and "doKeyUp()" functions respectively:

```
    );
    character.init();

    document.addEventListener("keydown", doKeyDown);
    document.addEventListener("keyup", doKeyUp);

    window.requestAnimationFrame(run);
}
```

Below the "draw()" function are the two event listener callback functions. Add the following code to the functions:

```
function doKeyDown(e) {
    e.preventDefault();
    if (character != undefined) { character.doKeyInput(e.key, true); }
}

function doKeyUp(e) {
    e.preventDefault();
    if (character != undefined) { character.doKeyInput(e.key, false); }
}
```

These functions will tell the character to process a user input with a boolean determining if the key was pressed to start an action, or released to stop an action.

Inside the Example3 Character "init()" function, add the following code to scale the sprite:

```
// Initialise variables that cannot be calculated during
// object creation.
init() {
    console.log("init");
    // Apply scale multiplier to sprite frame dimensions
    this.spriteCanvasSize = [
        this.spriteFrameSize[0] * this.spriteScale,
        this.spriteFrameSize[1] * this.spriteScale
    ];
},
```

NOTE: A scale value of 1 is set in the example code (. This will not scale by default, but changing the value from 1 will apply a scaling factor.

Inside the character "action()" function, add the following code:

```
action(action) {
    console.log(`action: ${action}. Animation Frame ${this.animationFrame}`);
    // ignore duplicate actions.
    if (action === this.lastAction) return;

    // Handle each action type as cases.
    switch (action) {···
    }

    // keep track of last action to avoid reinitialising the current action.
    this.lastAction = action;
},
```

This function will take the provided action string parameter and apply responses using the switch block.

Add the following cases to the switch block:

```
switch (action) {
    case "moveLeft":
        this.animationTrack = 2;
        this.animationFrame = 0;
        this.direction[0] = -this.velocity;
        break;
    case "moveRight":
        this.animationTrack = 3;
        this.animationFrame = 0;
        this.direction[0] = this.velocity;
        break;
    case "moveUp":
        this.animationTrack = 0;
        this.animationFrame = 0;
        this.direction[1] = -this.velocity;
        break;
    case "moveDown":
        this.animationTrack = 1;
        this.animationFrame = 0;
        this.direction[1] = this.velocity;
        break;
    case "noMoveHorizontal":
        this.direction[0] = 0;
        this.animationFrame = 0;
        break;
    case "noMoveVertical":
        this.direction[1] = 0;
        this.animationFrame = 0;
        break;
    default:
        this.direction = [0, 0];
        break;
}
```

Inside the character update function, add the following code:

```
update(tick) {
    // increase time keeper by last update delta
    this.timeSinceLastFrame += tick;
    // check if time since last frame meets threshold for new frame
    if (this.timeSinceLastFrame >= this.frameTime) {
        // reset frame time keeper
        this.timeSinceLastFrame = 0;

        // update frame to next frame on the track.
        // Modulo wraps the frames from last frame to first.
        if (this.direction[0] !== 0 || this.direction[1] !== 0) {
            this.animationFrame = (this.animationFrame + 1) % this.spriteFrames[this.animationTrack].length;
        }
    }

    // Calculate how much movement to perform based on how long
    // it has been since the last position update.
    this.position[0] += this.direction[0] * tick;
    this.position[1] += this.direction[1] * tick;
```

The first block checks if the character is moving, and if so updates the animation frame for the current track, and wraps the index with modulo.

The second block adds position updates according to the current direction the character is moving in.

Finally, add the following code to the "doKeyInput()" function at the bottom of the character:

```
doKeyInput(e, isKeydown = true) {
    switch (e) {
        case "w":
            if (isKeydown) this.action("moveUp");
            else this.action("noMoveVertical");
            break;
        case "a":
            if (isKeydown) this.action("moveLeft");
            else this.action("noMoveHorizontal");
            break;
        case "s":
            if (isKeydown) this.action("moveDown");
            else this.action("noMoveVertical");
            break;
        case "d":
            if (isKeydown) this.action("moveRight");
            else this.action("noMoveHorizontal");
            break;
        default:
            if (!isKeydown) this.action("stop");
            break;
    }
```

This switch block takes the key from the input event listeners and calls an action type based on the key and whether the key is down or up. Key down actions set the character movement direction, and key up actions stop the character moving in the respective direction.

Save your work and test the output. Canvas 3 should have a character that is able to be moved using [W, A, S, D] keys, and the respective animation tracks should play when moving in each direction.

NOTE: Exercise 3 has a large amount of code to introduce. Check with your lecturer if you find issues with your implementation and are having difficulty debugging.

Exercise 3 complete.

Next Steps

- Use code from these exercises as a starting point (or supplimental source) for your group project – part 2.
- Try different sprite sheets
- Add more actions to the character, such as jump or interact.

Exercises complete.