Advanced Canvas

Week 9 Session 2

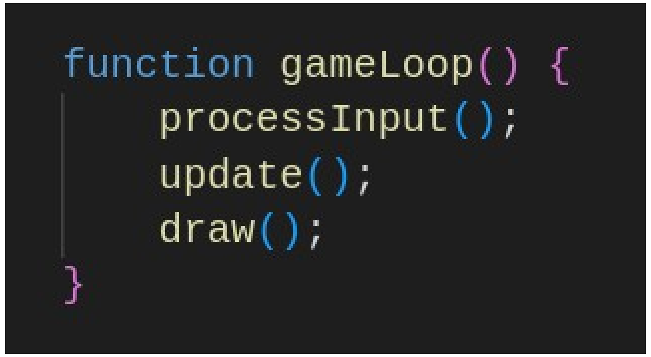
Contents of This session

- Video Game Concepts
  - Game Loop
  - Update Performance Variance
  - Tick
  - Input Event vs Input Checking
  - Sprites and Sprite Sheets
  - Animation Tracks and Frames

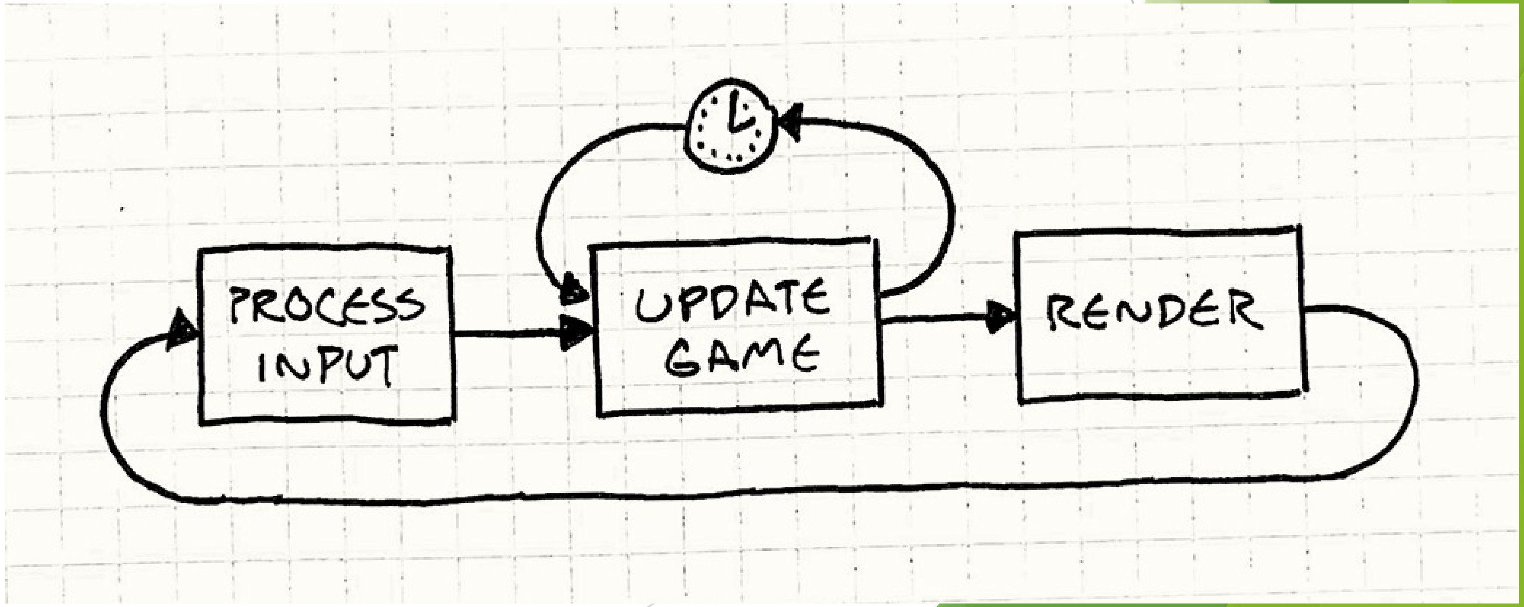
Video Game Loop

Video games use a central loop of functionality that is repeated continuously. It consists of three main components:

- Process Input
- Update
- Render / Draw



```
function gameLoop() {
    processInput();
    update();
    draw();
}
```



Game Loop Components

- Process Input:
  - Take input from the user (asynchronously via Event callbacks)
  - Translate user input into game-usable instructions
    - [W, A, S, D] => Player character movement
- Update:
  - Apply changes to game components
    - Calculate forces on physical objects (acceleration, velocity)
    - Implement variable adjustments from user input

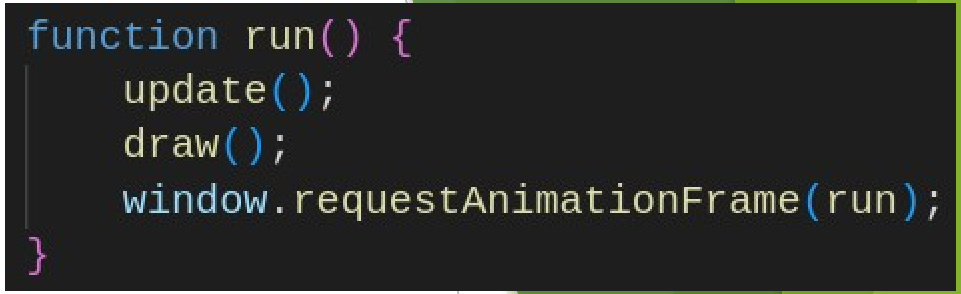
Game Loop Components

- Render:
  - Calculate where game elements will be on screen
  - Calculate transparency, lighting, shadows
  - Create a 2D array of pixels containing the screen data
- Draw:
  - Send the screen data to the display

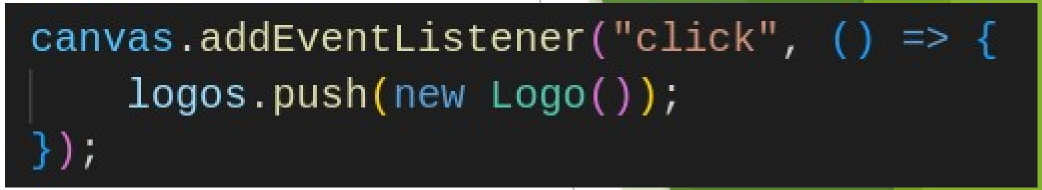
Game Loop with Canvas

- LogoAnimation.js from Week 7 Demos:

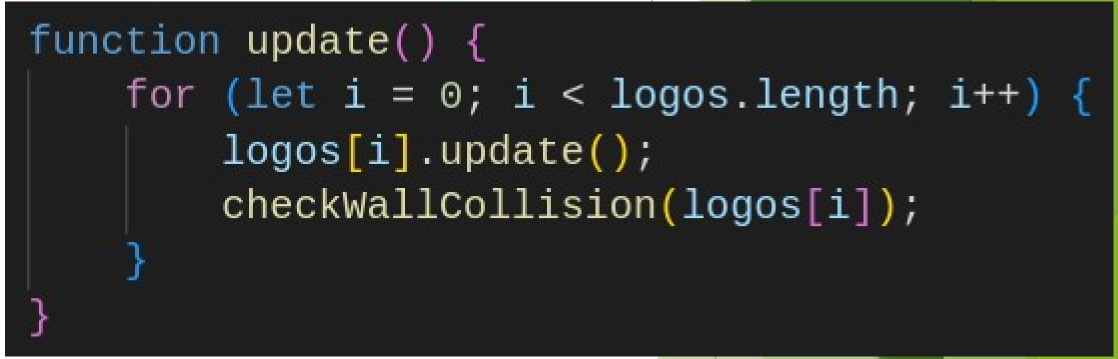
- Game Loop = run()
- User input = click event listener
- Update = update()
- Render/Draw = draw()



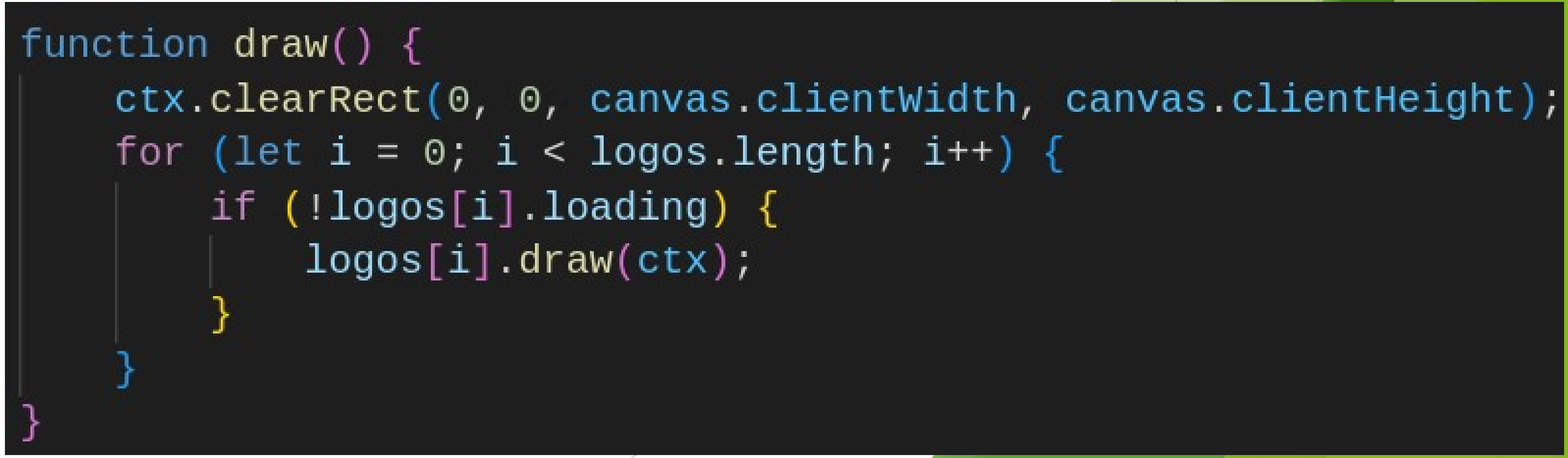
```javascript
function run() {
    update();
    draw();
    window.requestAnimationFrame(run);
}
```



```javascript
canvas.addEventListener("click", () => {
    logos.push(new Logo());
});
```



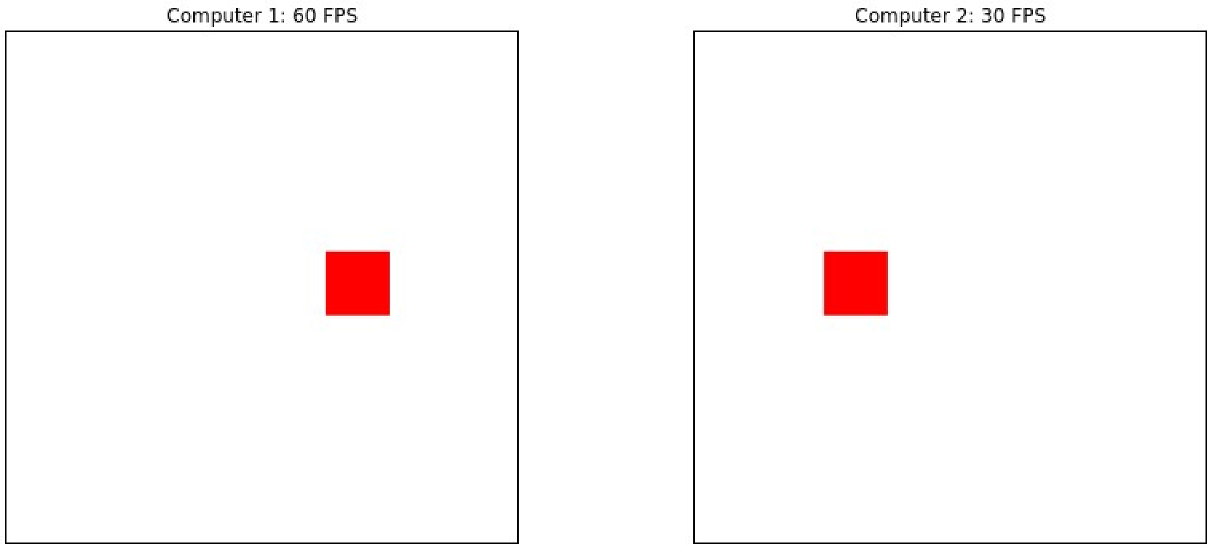
```javascript
function update() {
    for (let i = 0; i < logos.length; i++) {
        logos[i].update();
        checkWallCollision(logos[i]);
    }
}
```



```javascript
function draw() {
    ctx.clearRect(0, 0, canvas.clientWidth, canvas.clientHeight);
    for (let i = 0; i < logos.length; i++) {
        if (!logos[i].loading) {
            logos[i].draw(ctx);
        }
    }
}
```

Game Loop Variance

- This approach has a major flaw. Update speed is directly tied to render speed. A device that can draw the canvas faster will update the entire game faster.
- Scenario: Video game where the player moves a red square to the right by pressing a key.
  - Computer 1 runs a game at 60 Frames per Second
  - Computer 2 runs a game at 30 Frames per Second



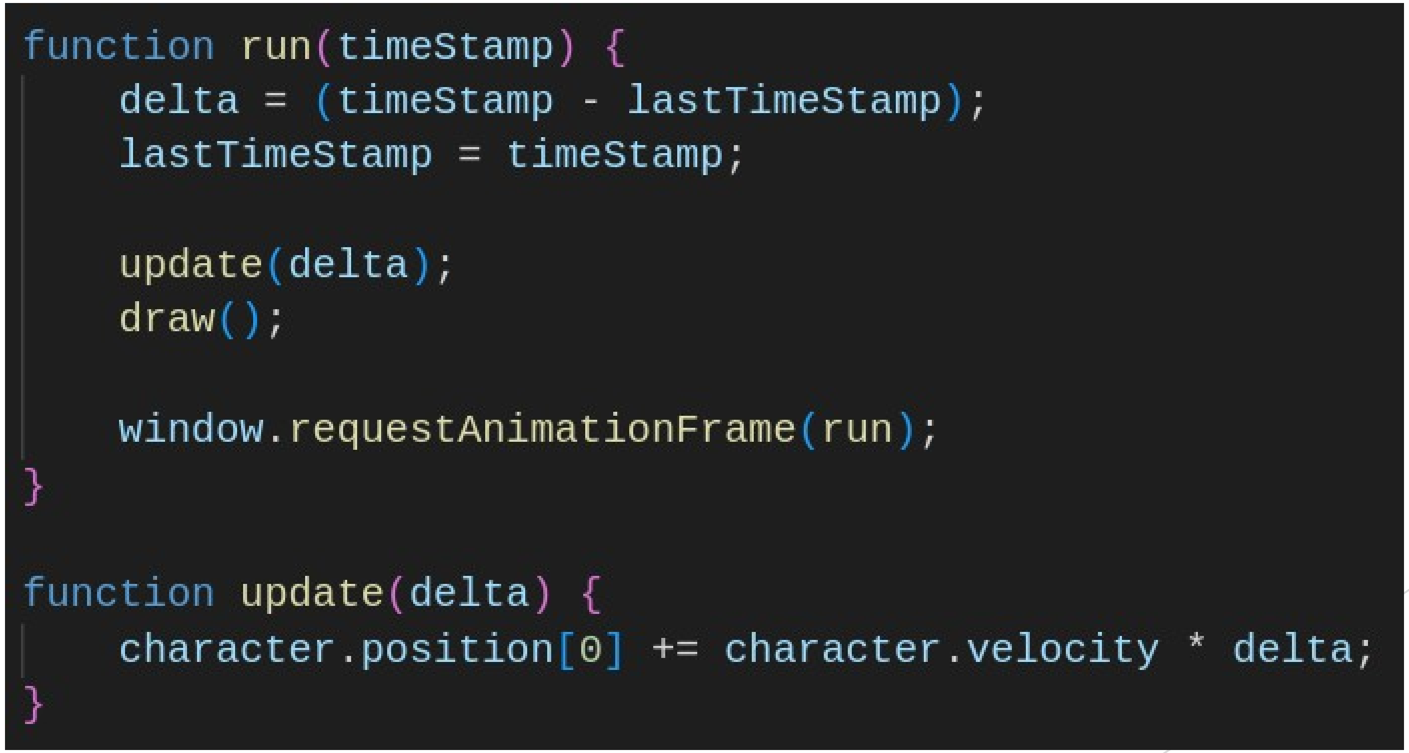
Frame-independent Updates

- To fix the update speed issue, game updates should be de-coupled from the frame rate.

- Solutions:
  - Run the updates and renders on separate threads
    - Multithreading complications, async, race conditions
  - Calculate the time of each frame and adjust updates by it
    - Easier to implement

Time Delta

- At the beginning of the game loop, check the time and compare it to the time check of the previous game loop.
- Pass the time delta (also called tick) to the updating code.
- Multiply variable adjustments by the delta.



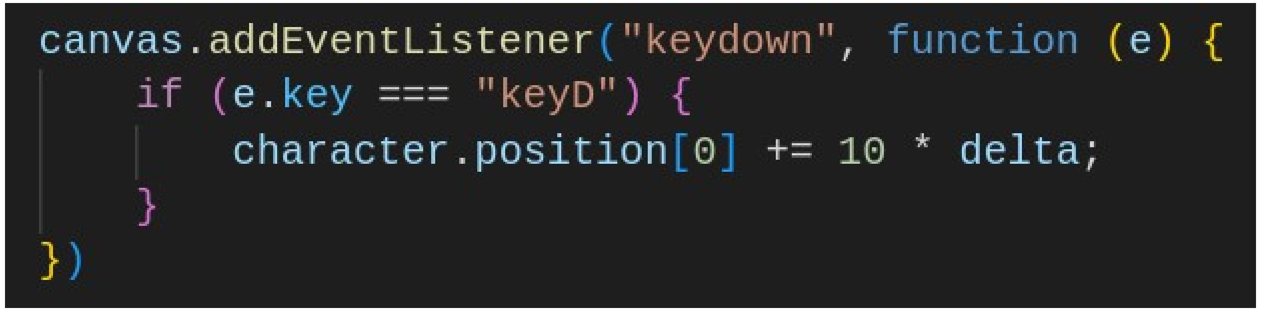
```
function run(timeStamp) {
    delta = (timeStamp - lastTimeStamp);
    lastTimeStamp = timeStamp;

    update(delta);
    draw();

    window.requestAnimationFrame(run);
}

function update(delta) {
    character.position[0] += character.velocity * delta;
}
```

Input Events vs Input Checking

- If we perform game updates in the input event listener callbacks we face another issue. Update pace is dictated by the system input polling rate.
- Most operating systems have a feature called Character Repeat Delay. It stops users from accidentally sending multiple key input values when a key is held for >1 poll cycle.
- Game input events are affected by this delay and polling rate.

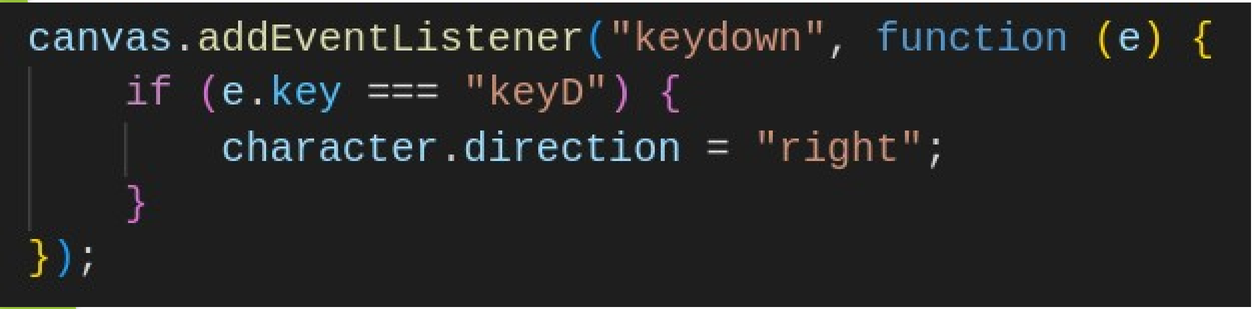
Input Events vs Input Checking

- Instead of update on input event:

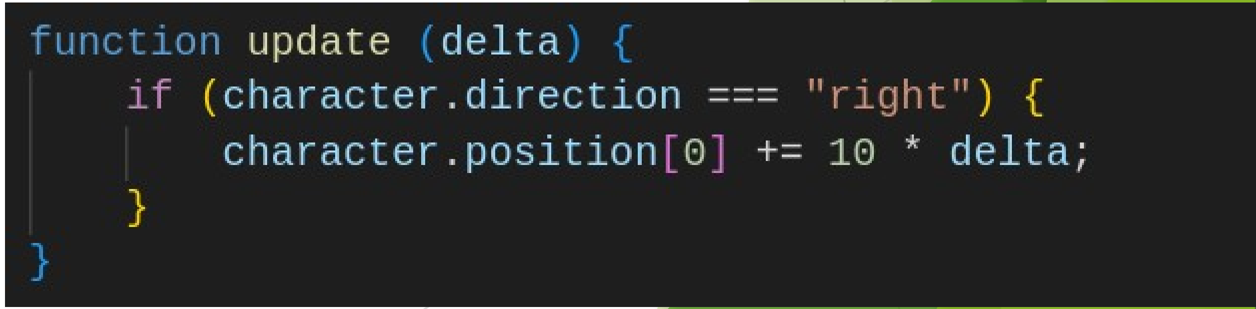


```
canvas.addEventListener("keydown", function (e) {
    if (e.key === "keyD") {
        character.position[0] += 10 * delta;
    }
})
```

- Separate the input command from the update:



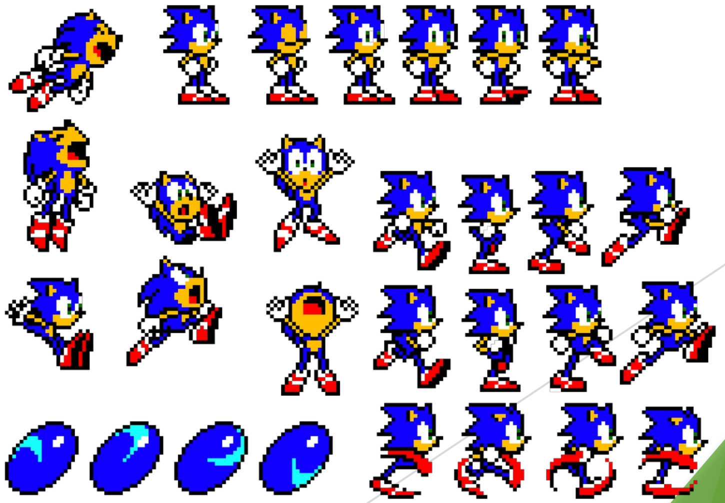
```
canvas.addEventListener("keydown", function (e) {
    if (e.key === "keyD") {
        character.direction = "right";
    }
});
```



```
function update (delta) {
    if (character.direction === "right") {
        character.position[0] += 10 * delta;
    }
}
```

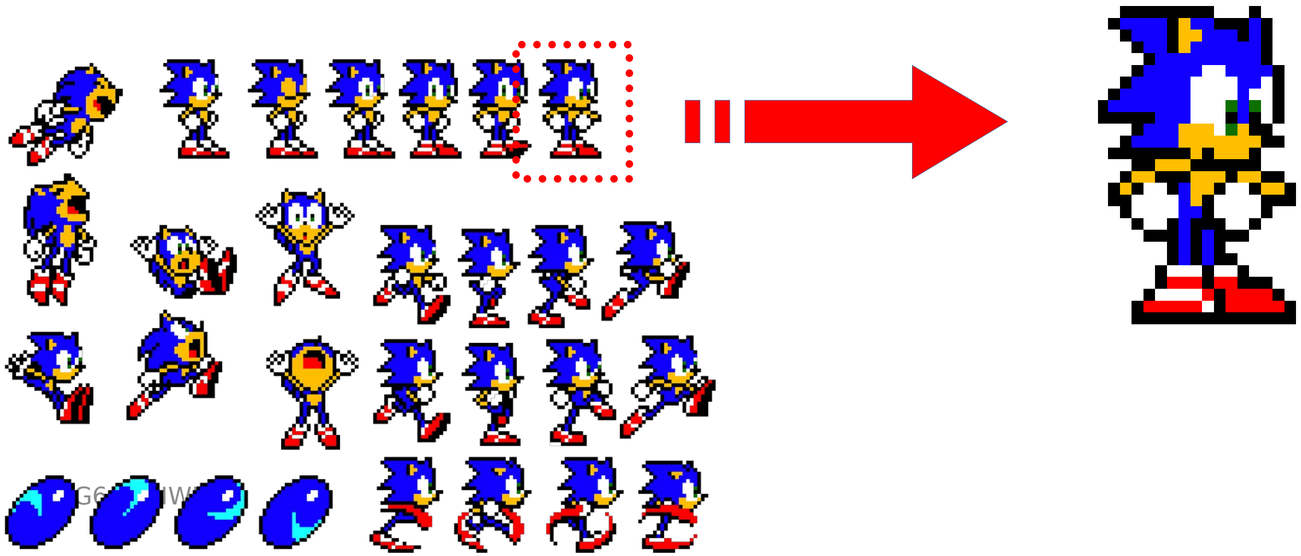
Sprites and Sprite Sheets

- 2D game element textures are called Sprites. For resource efficiency, many sprites can be stored in a single file called a Sprite Sheet.
- Sprite sheets are commonly used in tile and voxel level designs like classic Pokemon or Minecraft, and for characters with animations like Mario or Sonic the Hedgehog



Sprites and Sprite Sheets

- Using a Sprite sheet involves the following steps:
  - Load the Sprite sheet asset.
  - Save the coordinates of each sprite.
  - Draw a sub-image at render time using the coordinates.



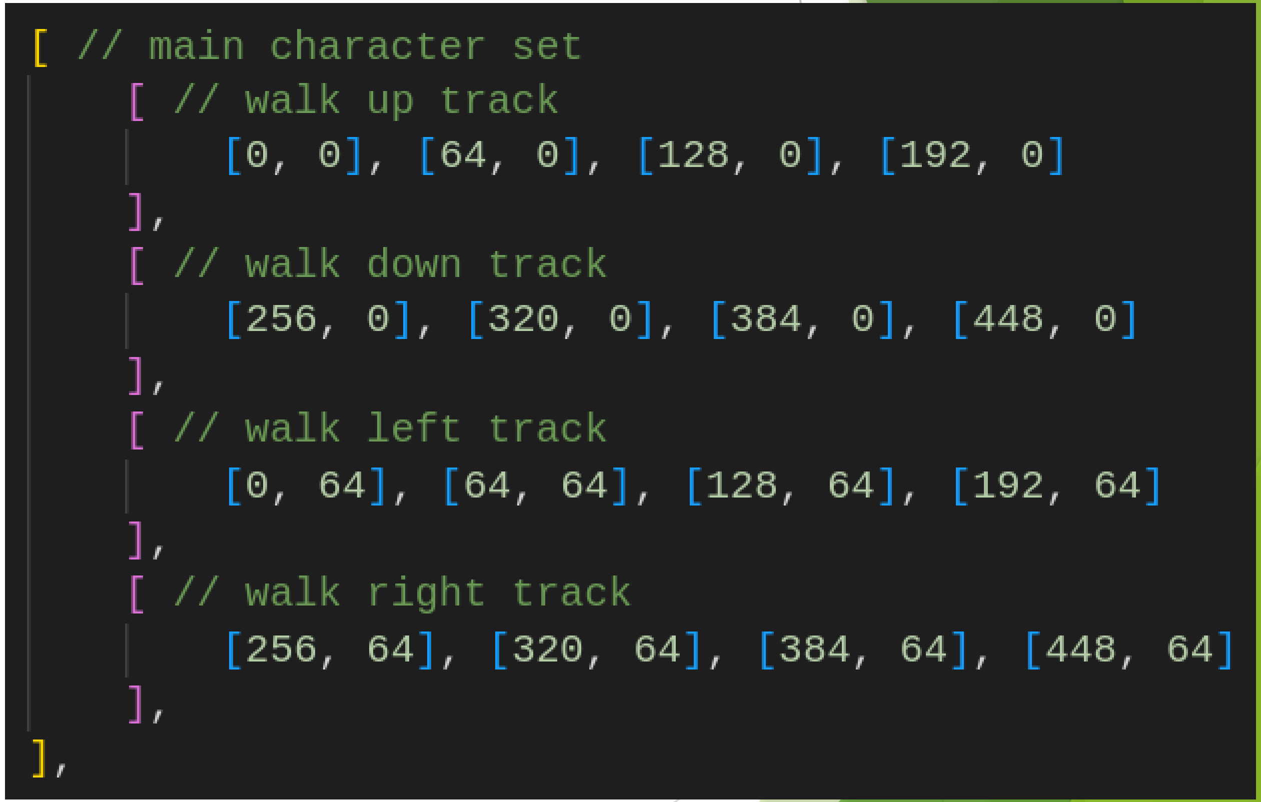
Sprite Sheet Animations

- Sprite sheets provide an easy way to implement flip-book animations.
- Flip-book animation is achieved by having each frame of an animation as a sprite, and an object’s texture is rapidly changed to each sequential animation sprite

[Sonic Sprite Animation](#)

Animation Tracks and Frames

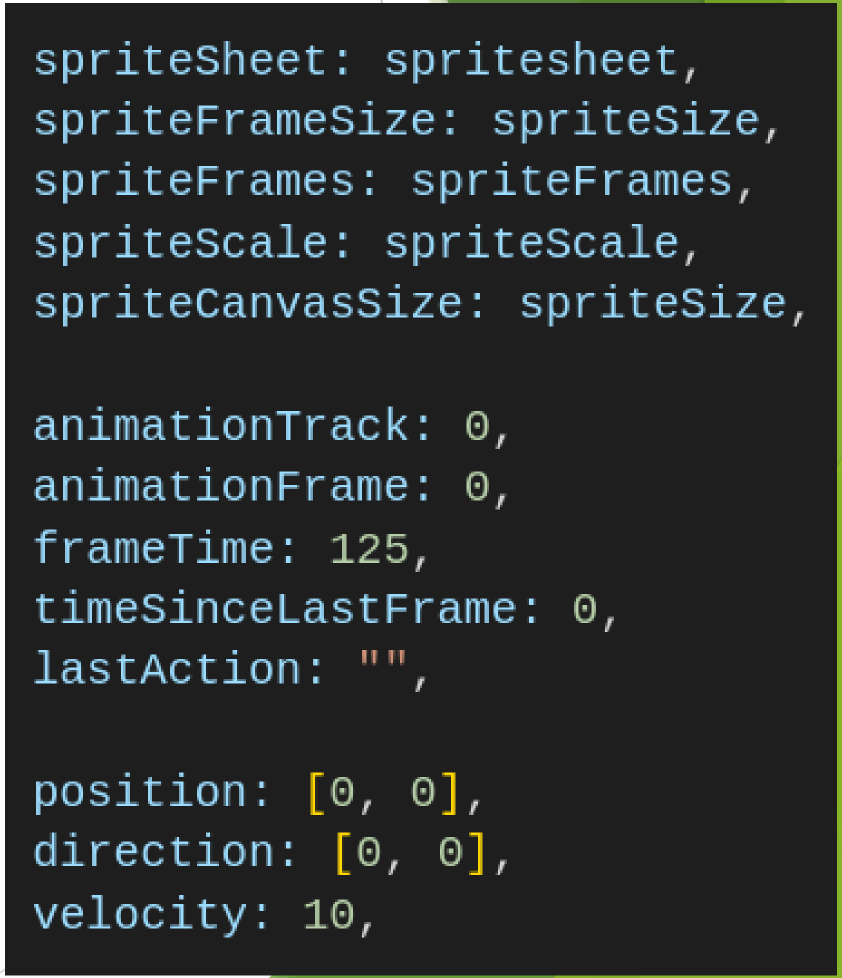
- Sprite sheet animation frame data can be stored in nested arrays:
- Coordinates: Array of X and Y value for a frame
- Track: Array of coordinates for an animation sequence
- Set: Array of Tracks for the animations of an object



```
[ // 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]
    ],
],
```

Animation Tracks and Frames

- Variables to operate the animation system:
  - Sprite sheet Image
  - Animation Set
  - Current Track Index
  - Current Frame Index
  - Sprite Frame Size
  - Sprite Canvas Size (if scaling)
  - Frame Time



```
spriteSheet: spritesheet,
spriteFrameSize: spriteSize,
spriteFrames: spriteFrames,
spriteScale: spriteScale,
spriteCanvasSize: spriteSize,

animationTrack: 0,
animationFrame: 0,
frameTime: 125,
timeSinceLastFrame: 0,
lastAction: "",

position: [0, 0],
direction: [0, 0],
velocity: 10,
```

Animation Tracks and Frames

- On character update:
  - Check time since last animation
  - If next animation is due, increment the animation Frame index by 1 (wrapped).
  - If character state has changed (such as direction change), set current animation track to match the state.
  - If an animation update has occured, reset the time since last animation to 0.

Array Index Wrapping

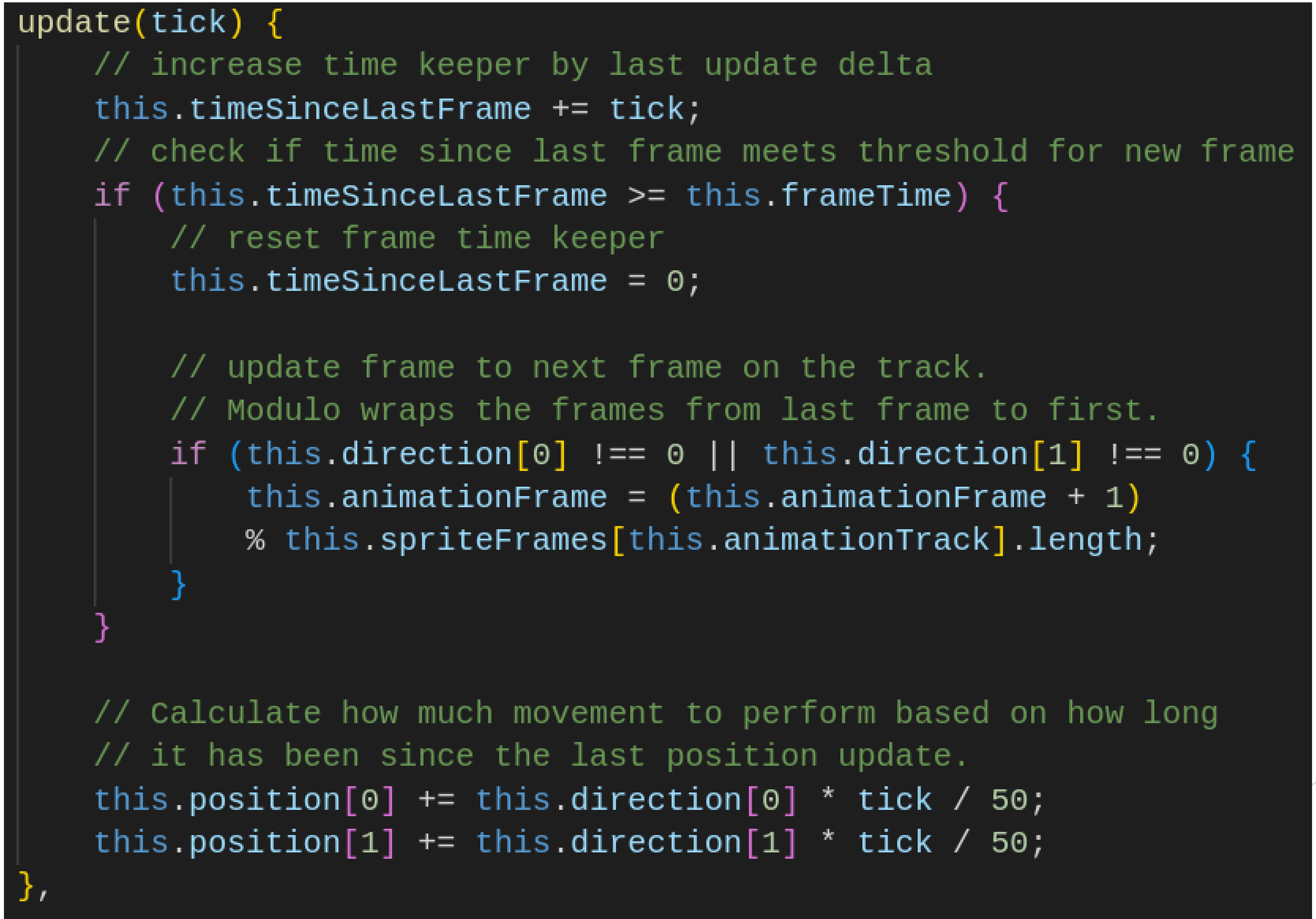
Array = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Index = 9

Index ++

- This will put the index outside the range of data.
- To wrap the end indices to the other end of the array:
  - Index = Index % array.length
- Modulus operator provides remainder after integer division
  - 10 % 10 = 0

Animation Tracks and Frames

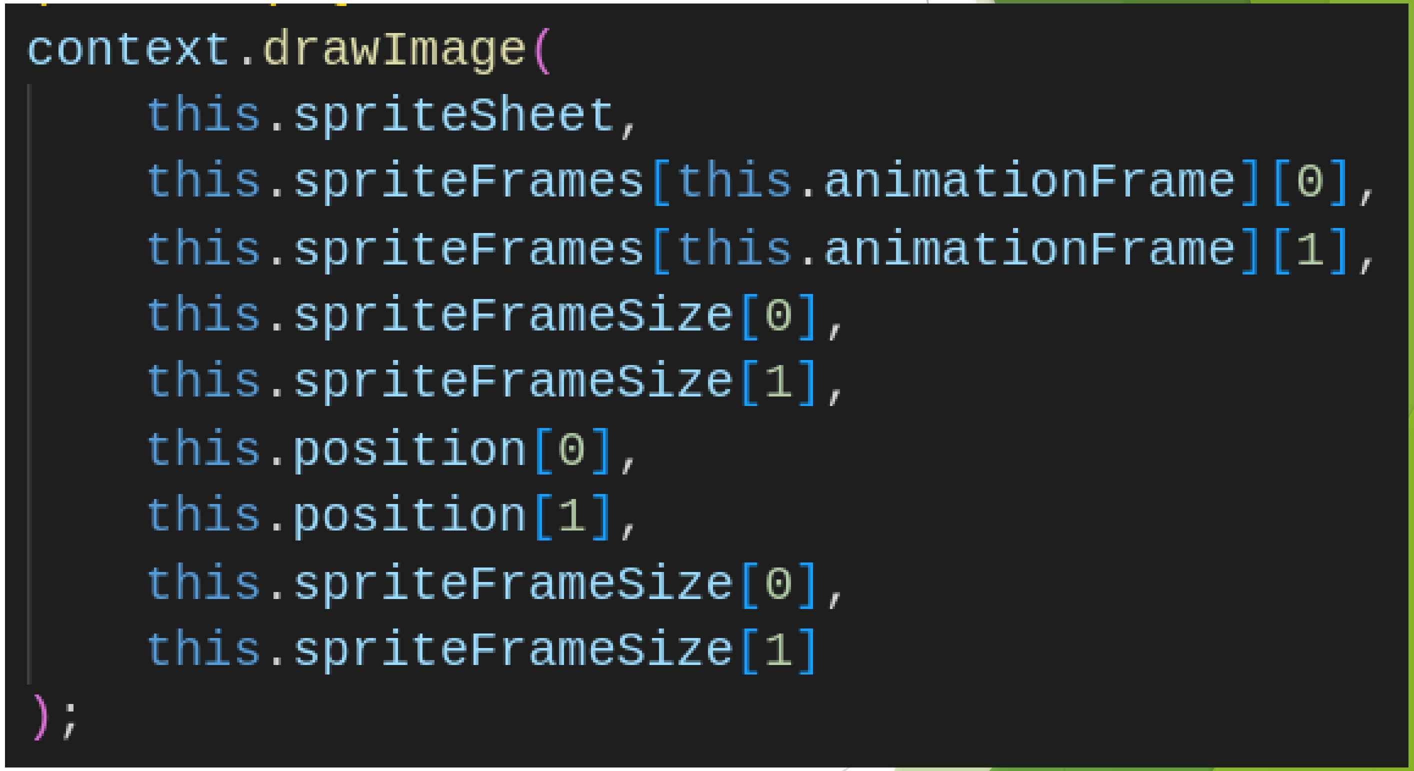


```
update(tick) {
    // increase time keeper by last update delta
    this.timeSinceLastFrame += tick;
    // check if time since last frame meets threshold for new frame
    if (this.timeSinceLastFrame >= this.frameTime) {
        // reset frame time keeper
        this.timeSinceLastFrame = 0;

        // 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 / 50;
    this.position[1] += this.direction[1] * tick / 50;
},
```

Drawing sprites from Sprite Sheet

context.drawImage(

- Spritesheet image,
- Sprite coordinate X,
- Sprite coordinate Y,
- Sprite width,
- Sprite height,
- Canvas coordinate X,
- Canvas coordinate Y,
- Canvas Sprite width,
- Canvas Sprite height)



```
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]
);
```

Sprite Sheet Animation in Action

https://jschollitt.github.io/week9/week9.html

Exercise

- Canvas Sprite Sheet Exercise on Moodle

End of The Session 2

Week 9