Animation

- Frames and Frame Rate
- Simulation
- Timer
- Tweening
- Easing
- Keyframes
- System Timers (and UI Threading)

Animation

Animation is the simulation of movement using a series of images (or drawings, models, etc.)

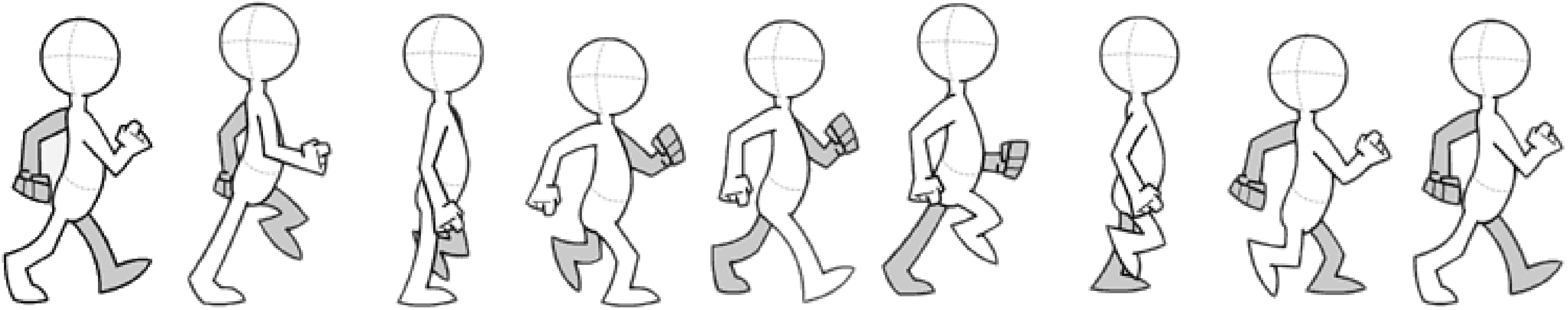


IMAGE SOURCE: https://www.angryanimator.com/word/2010/11/26/tutorial-2-walk-cycle/

Animation Terminology

Frame: each image (or state) of an animation sequence

Frame rate: number of frames to display per second

Tweening: interpolation of key frames into frames

Easing: a function that controls how tweening is calculated

Key Frame: defines the beginning and ending of a tween

In user interface programming, we typically animate numerical parameters that change how graphics are drawn over time

- parameters are often related to transformations (e.g. translate X and Y position to animate drawing position)
- parameters can be anything numeric: fill, stroke weight, etc.
- animating non-numeric values (e.g. a String or Image) is possible, but custom tweening methods are needed

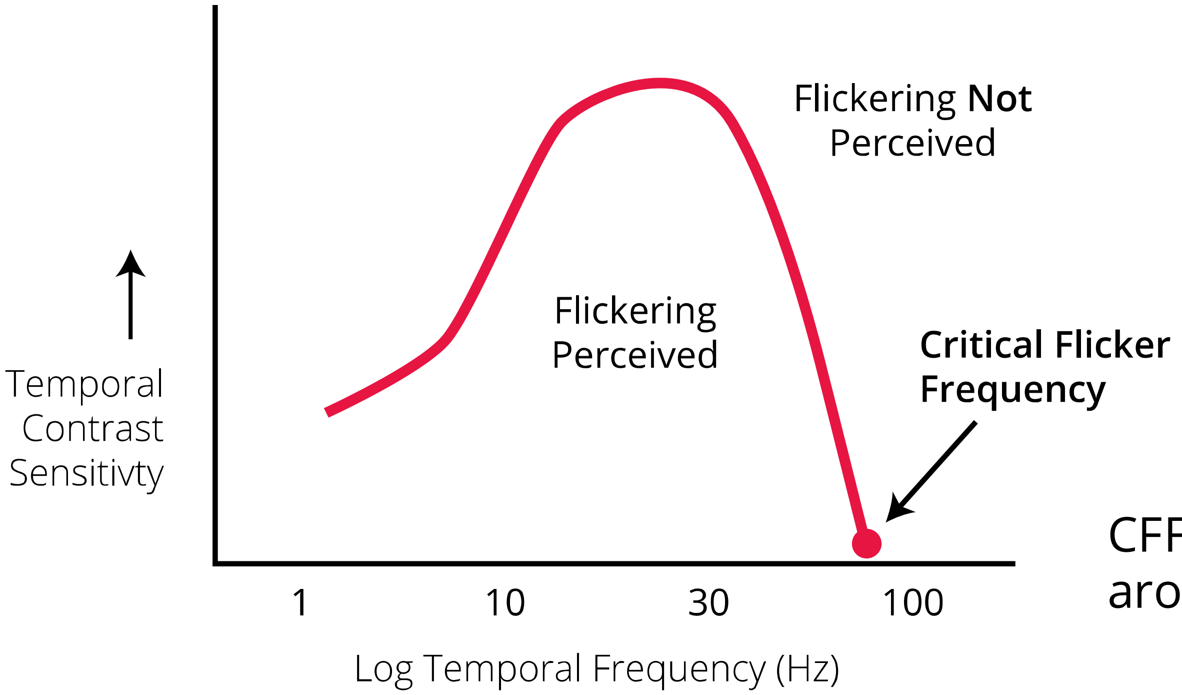
Frame Rate

- Measured in **frames-per-second (fps)**
  - can be expressed as Hertz (Hz): International System of Units (SI) measure defined as one cycle per second (e.g. 60 FPS = 60 Hz)
- Common device and media frame rates:
  - hand drawn animation: as low as 12 FPS, usually 24 fps
  - GIFs: usually 15 to 24 fps
  - Film: standard 24 fps, high def 60 fps
  - Legacy Broadcast Television: NTSC: 30 fps*, PAL 25 fps*
  - Computer displays: 60 fps or more
  - Computer games: 60 fps or more
  - Virtual Reality displays: 90 fps, 120 fps, or more

* each frame is sent progressively in two parts: odd “scanlines”, even “scanlines”, so communication speed is technically 60 Hz for NTSC and 50 Hz for PAL

Critical Flicker Frequency* (CFF)

- when perception of intermittent light source changes from flickering to continuous light
  - dependent on brightness of stimulus, wavelength, ...
  - varies by individual



CFF is typically around 60 – 90 Hz

*also called “flicker fusion threshold”, “temporal contrast sensitivity”

IMAGE SOURCE: https://entokey.com/temporal-properties-of-vision/

Zoetrope, mechanical example of CFF

https://youtu.be/8UC8j4pg1iA

Animation in SimpleKit

- SimpleKit lets you define a single callback for updating animations
  - callback is called with time (in milliseconds) as argument
- in simplekit

```
type AnimationCallback = (time: number) => void;

function setSKAnimationCallback(animate: AnimationCallback) ...
```

- in your program

```
setSKAnimationCallback((time) => { /* animate in here */ });
```

Animation by Simulation

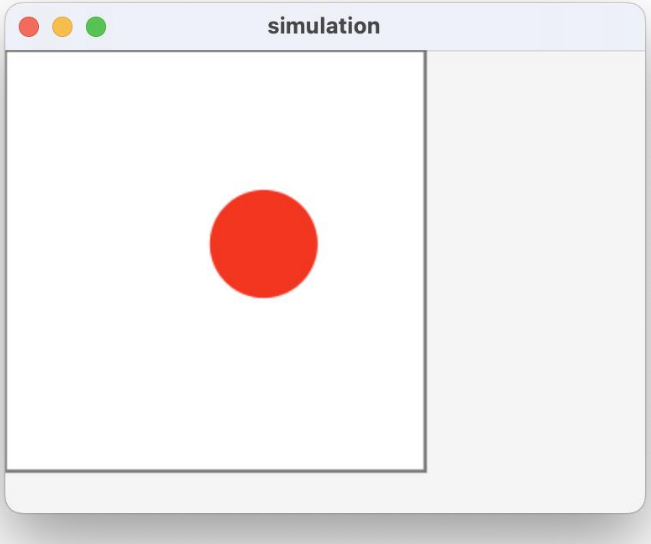
- Animation can be created through real time simulation
  - using functions, conditionals, etc.
- Typically, no start and end, it just loops or continues
  - conceptually simpler, just need a function and/or some rules

simulation

- dot moves in direction with speed, bounces when boundary hit
  - set animation callback bounce( )

```
// if it hits the edge of the box, change direction
if (dot.x < dot.r || dot.x > box.width - dot.r) {
    dx *= -1.0;
}
...

// update the dot position
dot.x += dx;
dot.y += dy;
```

- also demo of moving dot in circle
  - Call `circle( )` in animation callback



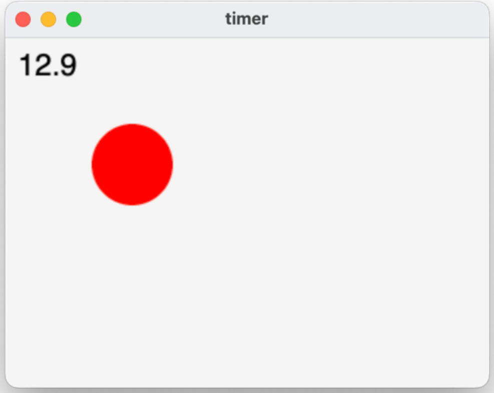
Timers

- A *timer* can be considered a simple kind of animation
  - State 1 ... wait ... State 2
- A timer needs
  - a duration
  - a start time
  - an update function to check if timer is finished
  - a method to check if timer is running
  - (usually) a callback function to call when timer is finished

timer simpleTimerDemo()

- Useful to trigger UI changes after some time
- Basic timer object
  - construct it with duration
  - start it with current time (from animation callback or **`skTime`**)
  - update with current time in animation callback
  - use **`isRunning`** property to trigger event

```
export class BasicTimer {
  constructor(public duration: number) {}
  ...
```



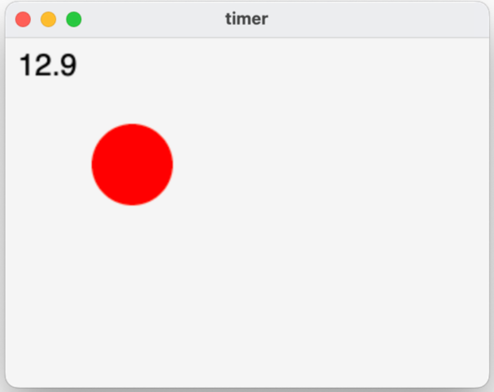
12.9

timer callbackTimerDemo()

- CallbackTimer to call function when time finishes
  - construct it with duration and callback function
  - start it with current time
  - update with current time in animation callback
  - calls callback when finished with time

```
export class CallbackTimer extends BasicTimer {
constructor(
  public duration: number,
  public callback: (t: number) => void) {}
  ...
```

- How to make dot pulse on and off every second?



Tweening

Interpolation between keyframes to create individual frames

- we'll consider "keyframes" as numeric values

Tweening Parameters

startValue is the starting value for the tween (keyframe 1)

endValue is the end value for the tween (keyframe 2)

duration is the duration for the tween

startTime is when the tween begins

Tweening Calculation

1) Calculate proportion of time completed so far:

```
t = (time – startTime) / duration
```

2) Interpolate start and end value to get current tweened value:

```
value = startValue + (endValue – startValue) * t
```

lerp function

linear int**erp**olation

- smoothly interpolate changes from one value to another
- lerp is a fundamental part of animation tweening

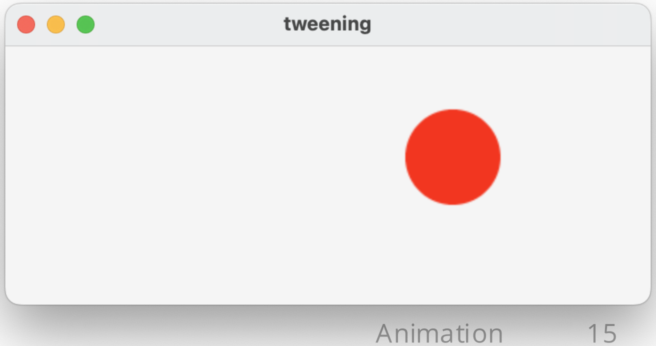
```
// linear interpolation from start to end
// using normalized time t (in [0, 1])
const lerp = (start: number, end: number, t: number) =>
  start + (end - start) * t;
```

tweening

- Basic animation object
  - construct it with animation parameters and **update value callback** (called every frame with new interpolated value)
  - start it with current time, update with time in animation callback

```
export class Animator {
  constructor(
    public startValue: number,
    public endValue: number,
    public duration: number,
    public updateValue: (p: number) => void
  ) {}
  ...
```

- Why does Animator force t to be 1 when time is elapsed? → remove that code and animate for 200ms to see what happens



Easing Functions

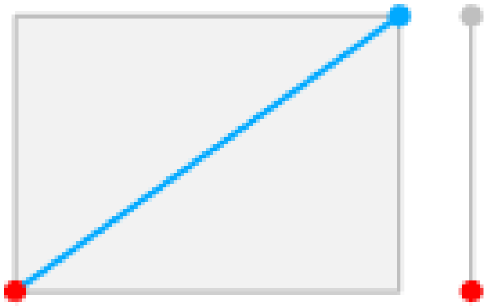
Controls *how* tweening is calculated

Lerp generates a **linear change** in value over time t

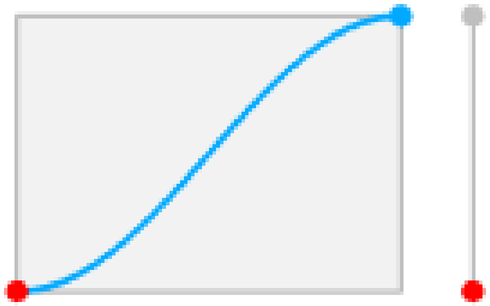
```
value = lerp(startValue, endValue, t)
```

An **easing function** changes how value is interpolated over time t

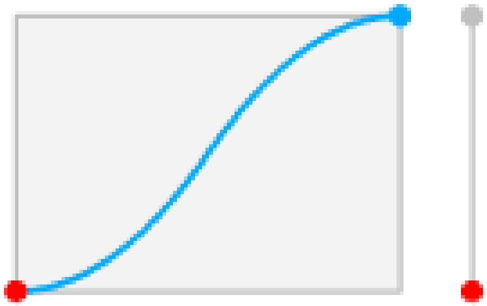
```
value = lerp(startValue, endValue, easing(t))
```



LINEAR



SINE



POWER: 2

[VIDEO SOURCE] https://www.alanzucconi.com/2021/01/24/piecewise-interpolation/easing-curves//

Easing Functions

- Type

```
type EasingFunction = (t: number) => number;
```

- Common functions

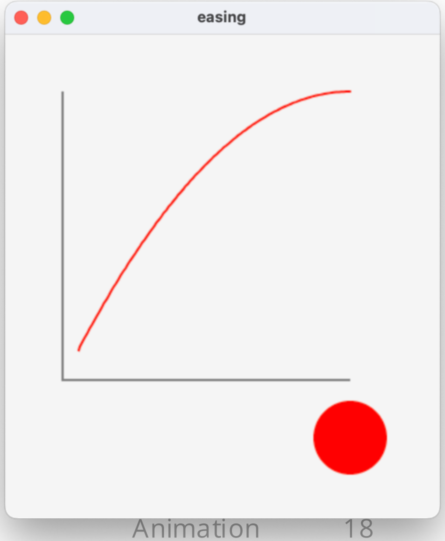
```
const flip = (t) => 1 - t;

const easeOut = (t) => Math.pow(t, 2);

const easeIn = (t) => flip(easeOut(flip(t)));

const easeInOut = (t) => lerp(easeOut(t), easeIn(t), t);
```

exponent changes the "amount" of easeOut

easing

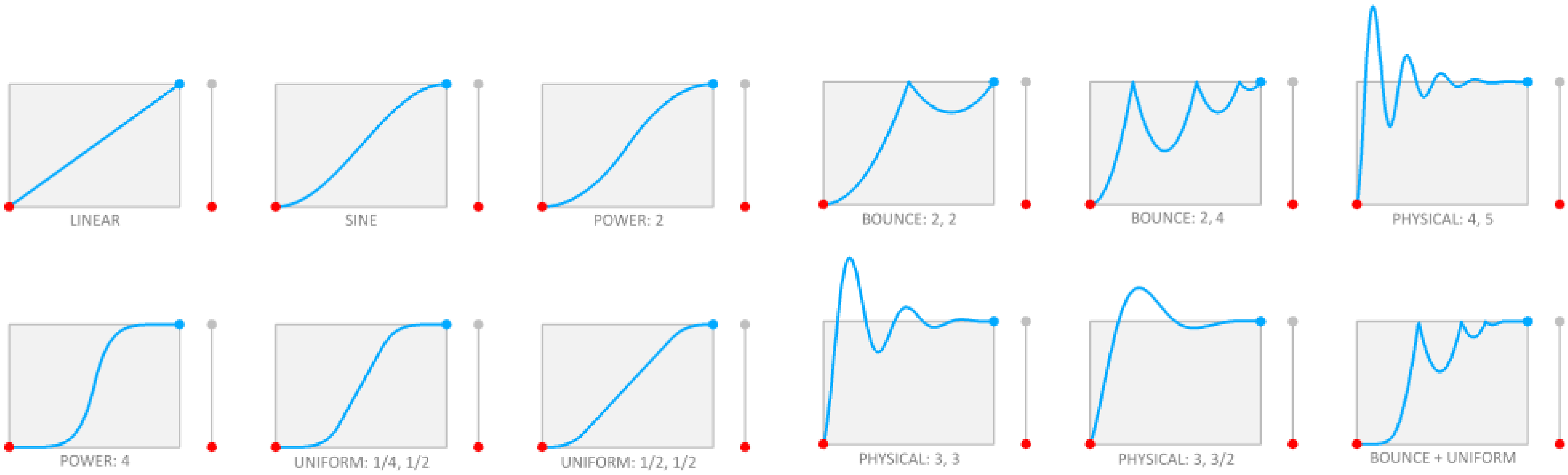
- Animation object with optional easing function

```
export class Animator {
  constructor(
    public startValue: number,
    public endValue: number,
    public duration: number,
    public updateValue: (p: number) => void,
    public easing: EasingFunction = (t) => t
  ) {}

...

}
```



Easing Function Resources

- http://robertpenner.com/easing/
- https://greensock.com/docs/v3/Eases
- https://www.febucci.com/2018/08/easing-functions/



[VIDEO SOURCE] https://www.alanzucconi.com/2021/01/24/piecewise-interpolation/easing-curves//

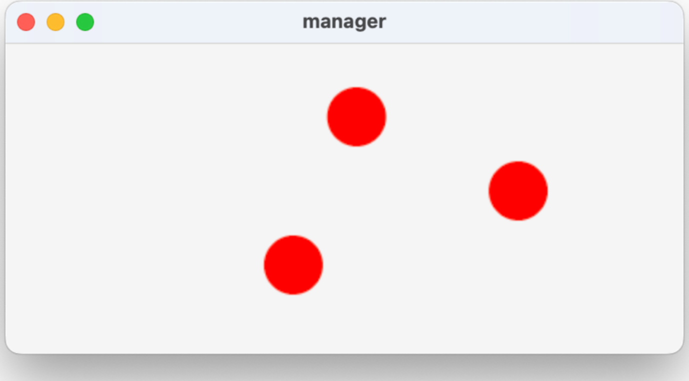
UI Toolkit Animation Architecture

- Some toolkits provide an **animation manager**
  - keep a list of active animations
  - update each animation every frame
  - remove animations when they finish
- Programmer sets animation, lets toolkit manage everything

manager

- AnimationManager singleton with list of active Animator objects

```
class AnimationManager {
  protected animations: Animater[] = [];

  add(animation: Animator) {
    this.animations.push(animation);
    ...
  }

  update(time: number) {
    // update every animation currently running
    this.animations.forEach(
        (a) => a.update(time));

    // remove animations that finished
    this.animations =
      this.animations.filter(
        (a) => a.isRunning);
  }
```



Keyframing

- A tween is essentially two *keyframes*:
  - *keyframe 1:* start time, starting value
  - *keyframe 2:* end time, ending value
- We can generalize this to a *list of keyframes:*
  - *keyframe 1:* time1, value1
  - *keyframe 2:* time2, value2
  - *keyframe 3:* time3, value3

  …
  - *keyframe N:* timeN, valueN
- A sequence of keyframes enables animations over time:
  - find keyframe i and keyframe i + 1 for current time

    ```
    (time >= keyframe[i].time) && (time <= keyframe[i+1].time)
    ```
  - tween value as keyframe i and value as keyframe i + 1

Keyframing Example

keyframes

	time	*targetValue*
0:	0	100
1:	1000	300
2:	2500	400
3:	5000	50
4:	6000	50
5:	6500	100

Example 1

when time is 800,

tween keyframe 0 and keyframe 1:

```
t = (800 – 0) / (1000 – 0)
value = 100 + t * (300 - 100)
```

Example 2

when time is 3000,

tween keyframe 2 and keyframe 3:

```
t = (3000 – 2500) / (5000 – 2500)
value = 400 + t * (50 - 400)
```

Practice:

when time is 5250

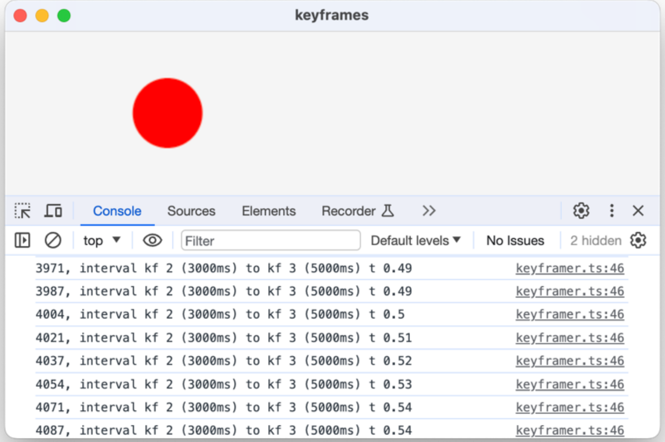
when time is 7000

keyframes

- Array of KeyFrame objects

```
const keyframes: KeyFrame[] = [
  { time: 0, targetValue: 50 },
  { time: 1000, targetValue: 450 },
  ...
```

- Keyframer object
  - find interval in keyframe array using time
  - tween to get current value
  - call callback
- Comments
  - how to insert pauses as keyframes?



```
3971, interval kf 2 (3000ms) to kf 3 (5000ms) t 0.49    keyframer.ts:46
3987, interval kf 2 (3000ms) to kf 3 (5000ms) t 0.49    keyframer.ts:46
4004, interval kf 2 (3000ms) to kf 3 (5000ms) t 0.5     keyframer.ts:46
4021, interval kf 2 (3000ms) to kf 3 (5000ms) t 0.51    keyframer.ts:46
4037, interval kf 2 (3000ms) to kf 3 (5000ms) t 0.52    keyframer.ts:46
4054, interval kf 2 (3000ms) to kf 3 (5000ms) t 0.53    keyframer.ts:46
4071, interval kf 2 (3000ms) to kf 3 (5000ms) t 0.54    keyframer.ts:46
4087, interval kf 2 (3000ms) to kf 3 (5000ms) t 0.54    keyframer.ts:46
```

Animation Using Built-in Timers

A **timer** triggers an event after some time period

1. Set time period to time interval for desired frame rate
   e.g. 30 FPS has an interval of 1/30 seconds (~33 milliseconds)

2. In the timer "finished" event handler, do:
   - update parameters you want to animate
   - (optional) redraw an updated image for the frame

3. restart the timer for the next interval
   - some timers can repeat automatically at a set interval

```
timer = Timer(() => {
    x += 1              // animate parameter
    draw()              // redraw scene
}
```

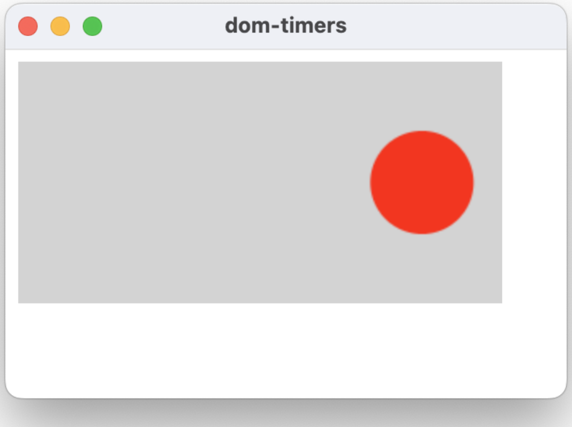
pseudo code

dom-timers demoIntervalTimer()

- The DOM / HTML engine has a general-purpose interval timer
  - try different frame rates by changing interval, what happens?

```
const duration = 2000;
let timer = setInterval(() => {
  const timePassed = performance.now() - start;
  dot.x = lerp(50, 250, timePassed / duration);
  gc.clearRect(0, 0, canvas.width, canvas.height);
  dot.draw(gc);
  // stop after certain time
  if (timePassed > duration) {
    clearInterval(timer);
  }
}, 1000 / 60);
```

1000 ms / 60 frames = 16.666 ms/frame = 60 FPS

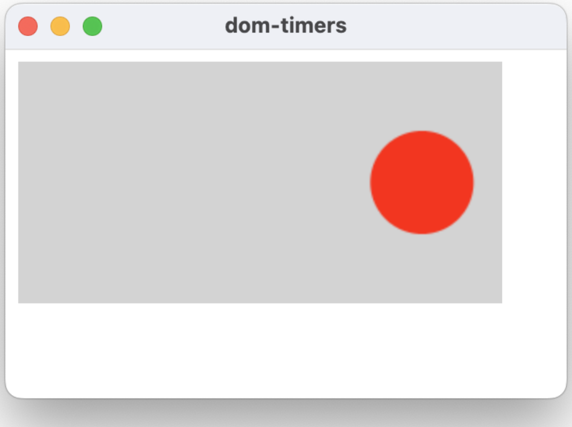


Check assignment specification, using this timer may not be allowed.

dom-timers demoRequestAnimationFrame()

- The DOM / HTML engine provides a special animation callback
  - this is what SimpleKit uses to create the run-loop

```
const duration = 2000;
requestAnimationFrame(function animate(timePassed) {
  dot.x = lerp(50, 250, timePassed / duration);
  gc.clearRect(0, 0, canvas.width, canvas.height);
  dot.draw(gc);
  // continue unless done animation
  if (timePassed < duration) {
    requestAnimationFrame(animate);
  }
});
```



Check assignment specification,
using this timer may not be allowed.

Timers and the UI Thread

- Most (all?) UI frameworks are **single-threaded** (e.g. JavaFX)
  - partly because its simpler and multiple threads isn't needed
  - a single threaded dispatch queue avoids deadlocks and race conditions due to unpredictable user-generated events
- Most (all?) UI frameworks are typically not thread-safe
  - to reduce execution burden, reduce complexity, etc.
- Most modifications to the UI must be on the UI execution thread
  - otherwise, behaviour may be unexpected
  - or in some cases, an exception is thrown

This has implications for animation timers in those frameworks

- HTML DOM interval and animation timers run on the UI thread
- Other platforms may have timers running on a non-UI thread

Animation using Java Util Timer

This code is **Kotlin** with **JavaFX**

```
import java.util.*

// create timer
val timer = Timer()

// schedule a task to repeat
timer.scheduleAtFixedRate(
  // WARNING! This task is NOT executed on the JavaFX thread!
  object : TimerTask() {
    override fun run() {
      aniScene.x += 1.0        // animate parameter
      aniScene.draw()          // redraw updated scene
    }
  },
  0, 1000/60
)
```

This type of timer does not run on the JavaFX UI thread: it may cause an exception if modifications to the scene graph are attempted in the event handler.

Animation using Java Util Timer with JavaFX Runnable

This code is **Kotlin** with **JavaFX**

```
import java.util.*
import javafx.application.Platform


// create timer
val timer = Timer()

// schedule a task to repeat
timer.scheduleAtFixedRate(

  object : TimerTask() {
    override fun run() {
      // runs the code on the JavaFX thread
      Platform.runLater {
        aniScene.x += 1.0       // animate parameter
        aniScene.draw()         // redraw updated scene
      }
    }
  },
  0, 1000/60
)
```

Standard Animation API and Libraries

- CSS Transitions and Animation
  - https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animations/Using_CSS_animations
  - can control with TypeScript by creating and manipulating styles
- Many 3rd party animation libraries
  - GreenSock https://greensock.com/
  - can animate any JavaScript object field
  - many easing functions
  - keyframing system

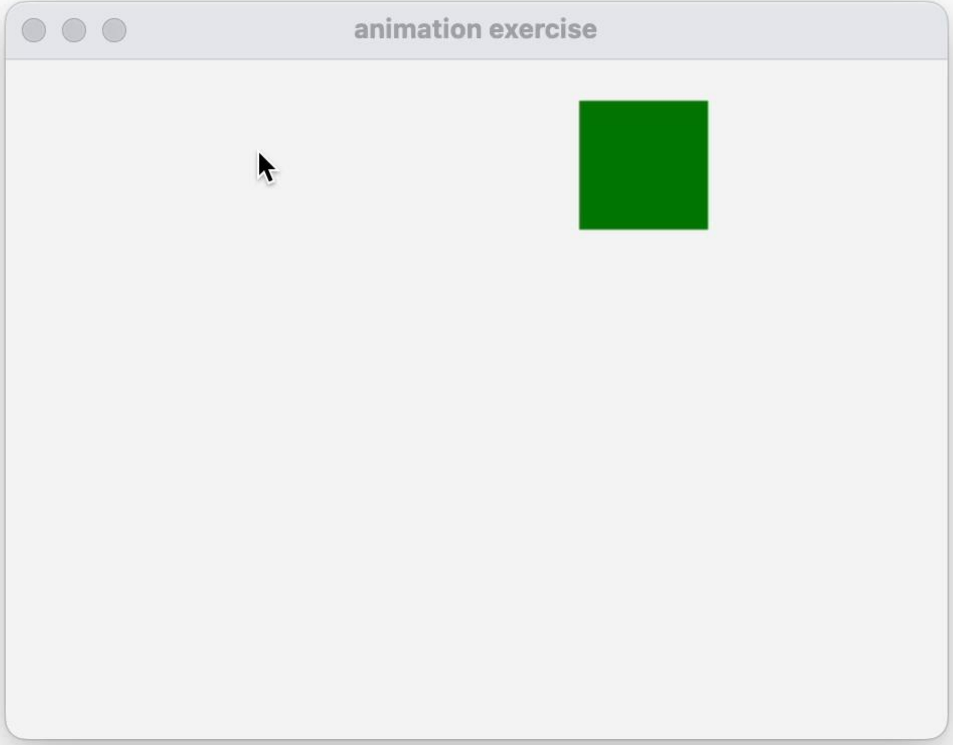
Check assignment specification, using CSS animations or external animation libraries may not be allowed.

Exercise

EXERCISE

- Draw green 64px square
- If mouse click on square:
  - turn square to blue for 2s
- If mouse click not on square:
  - animate square centre to click position over 1s
- Use SimpleKit
  - draw callback
  - event handler
  - animation update

Hint: create a Square shape class that holds *all* of its state (position, fill, **timer, animations**, ...).



Video demo:
https://vault.cs.uwaterloo.ca/s/trNbEgP24p4mbas