

Lecture 6: Advanced Animation

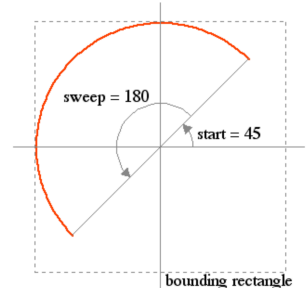
Written by Ben Yan, from materials by Jerry Cain

GArc Geometry

The **GArc** class represents an arc formed by taking a section from the perimeter of an oval. An arc is defined by its **start angle**, its **sweep angle**, and its bounding box, i.e. the top-left coordinate **(x,y)**, and width and height of its corresponding oval.

The following method can be used to create a custom **GArc** object.

```
let arc = GArc(x, y, width, height,
               startAngle, sweepAngle);
```



Angles are measured in degrees going counterclockwise from the positive x axis. Negative angles signify a clockwise direction. Also, if you'd like to procrastinate setting the arc's location, you can write:

```
let arc = GArc(width, height, startAngle, sweepAngle);
gw.add(arc, x, y); // choose location only when we add it to the canvas!
```

The **GArc** class also supports this gallery of methods, which allow you to animate an arc's appearance.

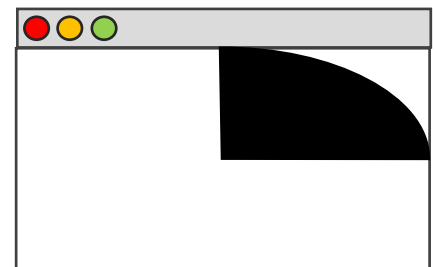
setStartAngle(start)	Sets the start angle for the arc.
getStartAngle()	Returns the start angle for the arc.
setSweepAngle(sweep)	Sets the sweep angle for the arc.
getSweepAngle	Returns the sweep angle of the arc.
setFrameAngle(x, y, width, height)	Resets the bounds for the corresponding oval's frame.

Filled Elliptical Arc

The following program displays a shaded, pie-shaped wedge formed by the center and the endpoints of arc sweeping 90 total degrees.

```
function FilledEllipticalArc() {
    let gw = GWindow(GWINDOW_WIDTH,
                     GWINDOW_HEIGHT);
    let start = 0; // angle in degrees
    let sweep = 90; // angle distance in degrees

    let arc = GArc(0, 0, gw.getWidth(), gw.getHeight(), start, sweep);
    arc.setFilled(true);
    gw.add(arc); // note top-left coordinate of oval is the origin (0,0)!
}
```

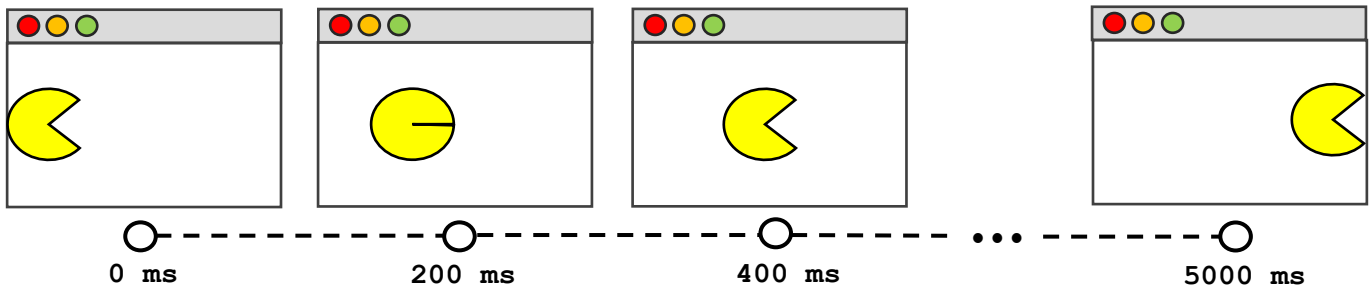


Note the **GArc** class supports functions **setFilled(boolean)** and **setFillColor(color)**, which are similar to their **GOval** and **GRect** counterparts.

Pac-Man

We'll write a program that uses a **GArc** object to display a **PacMan** figure (like the one from the arcade game), placing him initially at the left edge of the graphics window. From there, a timer animation enables **PacMan** to glide horizontally to the right edge of the window, and as he moves, the program should change the start and sweep angles of the **GArc** so the mouth appears to open and close.

The mouth is fully open when the start and sweep are **45** and **270** degrees, respectively, and the mouth is fully closed when the start and sweep are **0** and **360** degrees (i.e. full circle), as illustrated below.

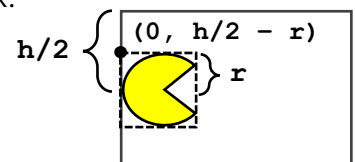


Assume the following constants have been defined. Here, the diameter of **PacMan**'s oval is **60** pixels, and every **20** milliseconds, he moves at **2** pixels in the **x** direction, and the start angle of his arc changes by **DTHETA** = **5** pixels—positively if his mouth is opening, and negatively if his mouth is closing.

```
const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 300;
const TIME_STEP = 20;
const PACMAN_SIZE = 60; /* diameter in pixels */
const PACMAN_DX = 2;
const PACMAN_DTHETA = 5;
```

First, we define the graphics window, as well as **PacMan**'s initial bounding box.

```
let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
let x0 = 0;
let y0 = gw.getHeight() / 2 - PACMAN_SIZE / 2;
```



Next, we create **PacMan** ☺, placed at his starting location, and with his mouth fully open, ready to eat!!

```
let pacman = GArc(x0, y0, PACMAN_SIZE, PACMAN_SIZE, 45, 270);
pacman.setFilled(true);
pacman.setFill("Yellow");
gw.add(pacman);
```

In anticipation of the **step** function that'll animate PacMan to life, we'll want to keep track of two state variables. One is the start **angle** of his mouth. The other is a **sign** value, which is positive (**+1**) if his mouth is opening (**angle** is increasing), and negative (**-1**) if closing (**angle** is decreasing).

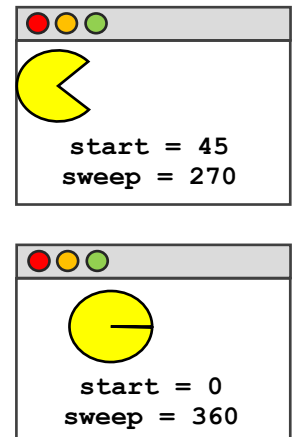
```
let angle = 45;
let sign = -1;
```

We proceed to implement **step**, which is called on regular intervals of **TIME_STEP** milliseconds.

```
let step = function() {
    angle += sign * PACMAN_DTHETA;
    if (angle === 0 || angle === 45) sign = -sign;

    pacman.setStartAngle(angle);
    pacman.setSweepAngle(360 - 2 * angle);
    pacman.move(PACMAN_DX, 0); // horizontal movement

    if (pacman.getX() + PACMAN_SIZE >= gw.getWidth()) {
        clearInterval(timer);
    } // reaches right side of window
};
let timer = setInterval(step, TIME_STEP);
```



In each step, we increase or decrease **angle** by **DTHETA** pixels, and change the start and sweep angles accordingly. Note for his mouth's opening to be vertically symmetric, the end angle must be $360 - \text{angle}$, meaning the sweep angle adjusts to $(360 - \text{angle}) - \text{angle} = 360 - 2 * \text{angle}$.

When the start **angle** reaches either **0** (fully close) or **45** (fully open), we reverse the **sign** so the mouth moves the other way in future steps. Lastly, once the bounding box of **PacMan** kisses the right side of the window, we call **clearInterval** to stop the timer animation.

Assembling all the knobs and screws together, we get this vintage implementation of arcade PacMan ☺.

```
function PacMan() {
    let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
    let x0 = 0;
    let y0 = (gw.getHeight() - PACMAN_SIZE) / 2;

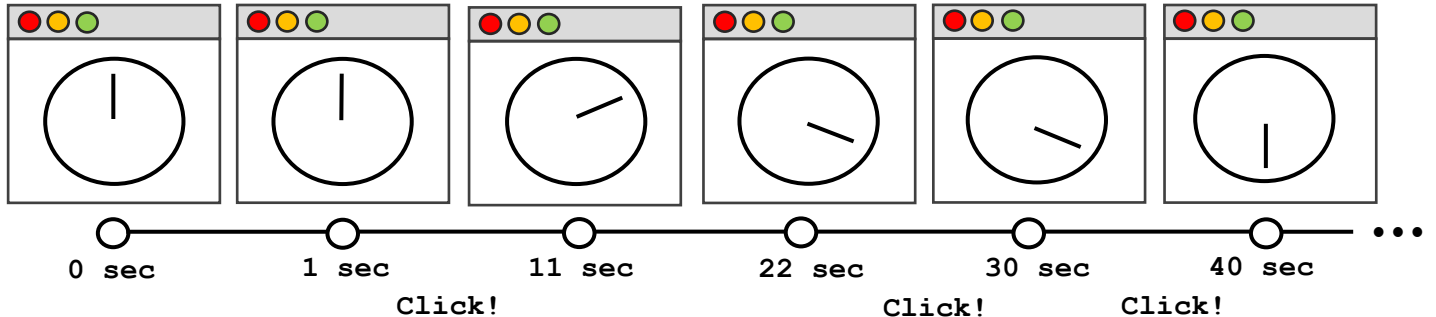
    let pacman = GArc(x0, y0, PACMAN_SIZE, PACMAN_SIZE, 45, 270);
    pacman.setFilled(true);
    pacman.setFill("Yellow");
    gw.add(pacman);

    let angle = 45;
    let sign = -1;
    let step = function() {
        angle += sign * PACMAN_DTHETA;
        if (angle === 0 || angle === 45) sign = -sign;

        pacman.setStartAngle(angle);
        pacman.setSweepAngle(360 - 2 * angle);
        pacman.move(PACMAN_DX, 0);
        if (pacman.getX() + PACMAN_SIZE >= gw.getWidth()) {
            clearInterval(timer);
        }
    };
    let timer = setInterval(step, TIME_STEP);
}
```

Stopwatch

Next, we'll write a graphical program that places a stopwatch so that it's centered within a graphics window. The stopwatch is stationary until the user clicks the mouse anywhere, at which point the stopwatch hand rotates at a speed of 360 degrees per minute. When the user clicks a second time, the stopwatch is halted, but a third click prompts it to continue rotating as if never interrupted, and so on.



Note we'll need a marriage of both mouse and timer events! 🤖 Assume the definition of these constants.

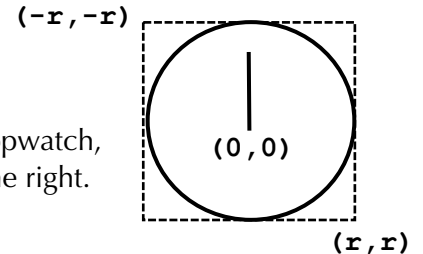
```
const GWINDOW_WIDTH = 500; // in pixels
const GWINDOW_HEIGHT = 300;
const STOPWATCH_RADIUS = 80;
const STEP_TIME = 60000/360; // rotate 360 degrees over course of 60 sec
const DTHETA = 1; // in degrees! change in angle every step
```

We also derive another pair of constants. Note the stopwatch's hand is slightly shorter than its radius.

```
const STOPWATCH_DIAMETER = 2 * STOPWATCH_RADIUS;
const STOPWATCH_HANDLENGTH = 0.8 * STOPWATCH_RADIUS;
```

First, we write a helper function that creates a **GCompound** object for the stopwatch, which consists of a **GOval** and a **GLine** for the watch hand, as shown on the right.

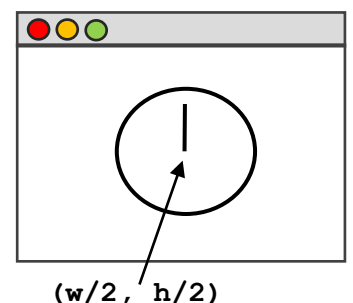
```
function createStopwatch() {
    let stopwatch = GCompound();
    stopwatch.add(GOval(-STOPWATCH_RADIUS, -STOPWATCH_RADIUS,
                        STOPWATCH_DIAMETER, STOPWATCH_DIAMETER));
    stopwatch.add(GLine(0, 0, 0, -STOPWATCH_HANDLENGTH));
    return stopwatch;
}
```



Note: In the coordinate system of **JSGraphics**, the **y** coordinate increases as we go down, not up 😊.

Next, we create the graphics window, and enlist the helper function above to create the stopwatch **GCompound** object, and add it so that its origin or center coincides with the center of the window.

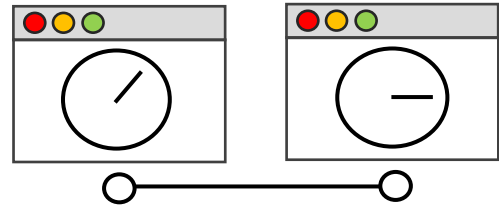
```
let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
let stopwatch = createStopwatch();
gw.add(stopwatch, GWINDOW_WIDTH / 2, GWINDOW_HEIGHT / 2);
```



To proceed, we'll draft an excerpt of code to animate the watch hand to move. We do this by **rotating the entire GCompound about its origin**, using the `rotate(angle)` method, where *angle* is in degrees.

```
let step = function() {
    stopwatch.rotate(-DTHETA);
};

let timer = setInterval(step, STEP_TIME);
```



Note that the angle, `-DTHETA`, is negative to rotate **clockwise** instead of counterclockwise, and that `STEP_TIME` is intentionally calculated so the hand rotates a full 360 degrees around in one minute ☺.

Next, we embed this animation as part of a "**click**" event listener—so that a click toggles between stopping the timer (if a timer is currently active), and starting the timer (if the hand is stationary).

```
let timer = 0; // sentinel value that says there's no active timer

let clickAction = function(e) {
    if (timer === 0) {
        // hand is stationary, so start animating it
        let step = function() {
            stopwatch.rotate(-DTHETA);
        };
        timer = setInterval(step, STEP_TIME);

    } else {
        // hand is moving, so make it stationary by stopping animation
        clearInterval(timer);
        timer = 0;
    }
};

gw.addEventListener("click", clickAction);
```

Note that we declare `timer` outside both the event and timer callback functions, so that all calls access, and potentially modify, the same exact `timer` value. Moreover, we use a `0` value as a sentinel or signal that no `timer` is actively running, so that `clickAction` knows whether to start or stop the clock. We alternatively could have defined a Boolean variable, e.g., `clockIsRunning`, and used that instead.

The full solution—with all the moving clock parts and stationery above—is shown on the next page ☺.



Little Ben

"I love clocks!"
– Ben



Big Ben

```

/* File: Stopwatch.js
 * -----
 * This program presents a simple animated stopwatch that can be started,
 * stopped, and continued by repeatedly clicking the mouse.
 */

/* Constants */
const GWINDOW_WIDTH = 500; // in pixels
const GWINDOW_HEIGHT = 300;
const STOPWATCH_RADIUS = 80;
const STEP_TIME = 60000/360; // rotate 360 degrees over course of 60 sec
const DTHETA = 1; // in degrees! change in angle every step

/* Derived Constants */
const STOPWATCH_DIAMETER = 2 * STOPWATCH_RADIUS;
const STOPWATCH_HANDLENGTH = 0.8 * STOPWATCH_RADIUS;

function Stopwatch() {
    let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
    let stopwatch = createStopwatch();
    gw.add(stopwatch, GWINDOW_WIDTH/2, GWINDOW_HEIGHT/2);

    let timer = 0; // sentinel value that says there's no active timer

    let clickAction = function(e) {
        if (timer === 0) {
            // hand is stationary, so start animating it
            let step = function() {
                stopwatch.rotate(-DTHETA);
            };
            timer = setInterval(step, STEP_TIME);
        } else {
            // hand is moving, so make it stationary by stopping timer
            clearInterval(timer);
            timer = 0;
        }
    };
    gw.addEventListener("click", clickAction);
}

function createStopwatch() {
    let stopwatch = GCompound();

    stopwatch.add(GOval(-STOPWATCH_RADIUS, -STOPWATCH_RADIUS,
                        STOPWATCH_DIAMETER, STOPWATCH_DIAMETER));
    stopwatch.add(GLine(0, 0, 0, -STOPWATCH_HANDLENGTH));

    return stopwatch;
}

```

