CreateJS - Animacija i rad sa resursima

Prethodna lekcija prevashodno je bila posvećena različitim tehnikama za crtanje, stilizovanje i obradu korisniče interakcije, kada se canvas elementom operiše korišćenjem CreateJS, odnosno EaselJS biblioteke. U ovoj lekciji nastavljamo upoznavanje sa CreateJS skupom funkcionalnosti, pa ćete tako imati prilike da čitate o različitim načinima za realizaciju animacije i rad sa resursima. Takav posao podrazumevaće i upoznavanje sa preostalim bibliotekama CreateJS-a: TweenJS, SoundJS i PreloadJS.

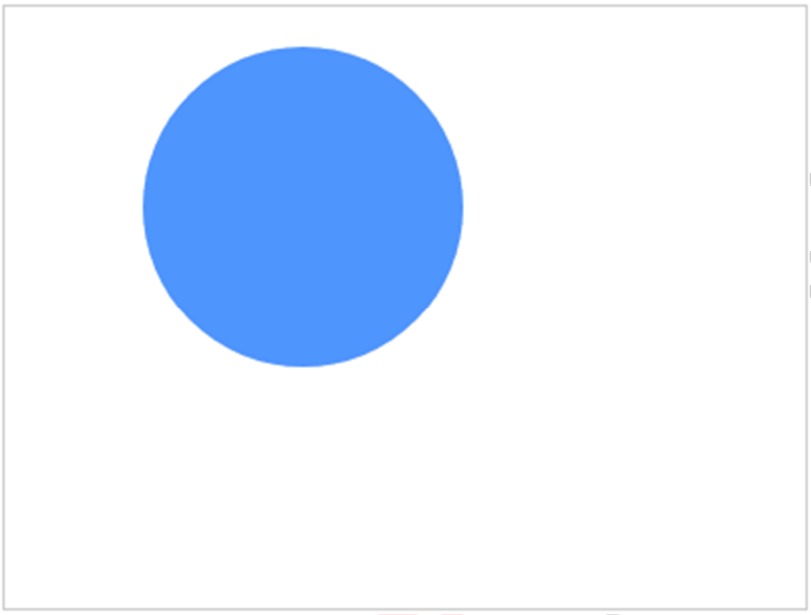
Animacija korišćenjem Ticker klase

U uvodnim modulima ovoga kursa imali ste prilike da vidite šta je animacija i na koji način se postiže kada je u pitanju frontend skup tehnologija. Ciklično ažuriranje osobina grafike koja se animira, obavljali smo korišćenjem različitih tajming funkcija.

Kada je u pitanju grafika koja se crta korišćenjem EaselJS biblioteke, animacija se može postići kreiranjem petlje animacije upotrebom već ilustrovanih, izvornih tajming funkcija:

```
window.onload = init;

let circle;
let stage;
function init() {
    stage = new createjs.Stage("my-canvas");
    let g = new createjs.Graphics();
    g.beginFill("#4F95FF");
    g.drawCircle(100, 100, 80);
    circle = new createjs.Shape(g);
    stage.addChild(circle);
    stage.update();
    update(stage);
}
let direction;
function update(stage) {
    if (circle.x > 50) {
        direction = -1;
    } else if (circle.x <= 0) {
        direction = 1;
    }
    circle.x = circle.x + direction;

    stage.update();

    requestAnimationFrame(function() {
        update(stage);
    });
}
```

Ovo je primer animacije CreateJS grafike, koji je realizovan korišćenjem već viđenih pristupa, odnosno upotrebom `requestAnimationFrame()` metode. Unutar `update()` funkcije, obavlja se ažuriranje `x` koordinate nacrtanog kruga. Nakon svakog ažuriranja `x` koordinate, poziva se `update()` metoda `Stage` klase, kako bi se osvežio prikaz unutar pozornice EaselJS biblioteke. Efekat koji se dobija je kao na animaciji 13.1.



Animacija 13.1 - Animacija kruga koji je nacrtan korišćenjem EaselJS biblioteke

Kada se koristi CreateJS skup funkcionalnosti, identičan efekat je mnogo lakše moguće dobiti korišćenjem jedne specijalne klase koja je deo CreateJS-a. Reč je o klasi `Ticker`.

Klasa Ticker

Klasa `Ticker` predstavlja apstrakciju oko izvornih tajming funkcija web pregledača. Reč je o klasi koja je deo globalnog CreateJS modula, što praktično znači da je na raspolaganju svim bibliotekama iz CreateJS skupa.

Klasa `Ticker` omogućava kontinuirano pozivanje određene logike, sa unapred definisanim razmakom između dva takva pozivanja. Praktično, klasa `Ticker` omogućava da se obavi isto ono što i korišćenjem već ilustrovanih tajming funkcija `setTimeout()` i `requestAnimationFrame()`.

Klasa `Ticker` se ne mora instancirati, već je nju moguće direktno koristiti. Kako bi se određena logika prosledila na kontinuirano izvršavanje korišćenjem ove klase, dovoljno je obaviti pretplatu na `tick` događaj:

```
createjs.Ticker.on("tick", update);
```

Pretplatu je moguće obaviti i na sledeći način:

```
createjs.Ticker.addEventListener("tick", update);
```

Klasa `Ticker`, podrazumevano emituje `tick` događaj 20 puta u jednoj sekundi, odnosno na svakih 50ms. Ukoliko je tako nešto potrebno promeni, mogu se koristiti dva svojstva `Ticker` klase:

- `interval` - definiše vremenski razmak između dva susedna emitovanja `tick` događaja
- `framerate` - definiše frekvenciju osvežavanja animacije (*fps, engl.*)

Ovakva svojstva je moguće upotrebiti na sledeći način:

```
createjs.Ticker.interval = 16;
createjs.Ticker.framerate = 60;
```

Dve upravo prikazane naredbe su ekvivalentne, odnosno imaju identičan efekat.

Funkciji za obradu `tick` događaja, automatski se prosleđuje jedan parametar, koji se odnosi na objekat koji predstavlja osobine događaja. Takav objekat, poseduje i jedno vrlo upotrebljivo svojstvo, koje omogućava lako kreiranje vremenski bazirane animacije. Reč je o svojstvu **delta**, koje sadrži vreme koje je proteklo od prethodnog emitovanja `tick` događaja. Tako `Ticker` klasa za nas automatski obavlja računanje takvog perioda, koji je od presudne važnosti za postizanje vremenski bazirane animacije.

`Ticker` klasa u pozadini, podrazumevano koristi `setTimeout()` metodu. Tako nešto je moguće promeniti korišćenjem svojstva `timingMode` koje može imati sledeće vrednosti:

- `Ticker.TIMEOUT` - podrazumevana vrednost kojom se definiše upotreba `setTimeout()` metode
- `Ticker.RAF` - vrednost kojom se definiše upotreba `requestAnimationFrame()` metode; kada se postavi ovakva vrednost, vrednost svojstva `framerate` se ignoriše, s obzirom da frekvenciju osvežavanja kontroliše web pregledač
- `Ticker.RAF_SYNCHED` - vrednost kojom se definiše upotreba `requestAnimationFrame()` metode, pri čemu `Ticker` klasa pokušava da samostalno uskladi frekvenciju osvežavanja sa vrednošću svojstva `framerate` koju smo samostalno definisali

Animacija iz uvodnog primera ove lekcije, korišćenjem `Ticker` klase se može realizovati na sledeći način:

```
window.onload = init;

let circle;
let stage;

function init() {
    stage = new createjs.Stage("my-canvas");

    let g = new createjs.Graphics();
    g.beginFill("#4F95FF");
    g.drawCircle(100, 100, 80);
```

```
        circle = new createjs.Shape(g);
        stage.addChild(circle);

        stage.update();

        createjs.Ticker.addEventListener("tick", update);
        createjs.Ticker.framerate = 60;

    }

    let direction;

    function update() {

        if (circle.x > 50) {
            direction = -1;
        } else if (circle.x <= 0) {
            direction = 1;
        }

        circle.x = circle.x + direction;

        stage.update();
    }
```

U primeru je sada direktno korišćenje `requestAnimationFrame()` metode zamenjeno `Ticker` klasom. Obavljena je pretplata na `tick` događaj, a kao funkcija za obradu takvog događaja, definisana je ona sa nazivom `update()`. U primeru je frekvencija osvežavanja postavljena na 60FPS.

Nešto ranije je spomenut parametar `delta`, koji funkcija za obradu `tick` događaja automatski dobija, a koji se može koristiti za kreiranje vremenski bazirane animacije. Iz prethodnih modula mi znamo šta je vremenski bazirana animacija, a kako bismo je postigli na upravo prikazanom primeru, dovoljno je da `delta` vrednost uvrstimo u formulu za računanje dužine pomeranja kruga:

```
    function update(event) {

        if (circle.x > 50) {
            direction = -1;
        } else if (circle.x <= 0) {
            direction = 1;
        }

        circle.x = circle.x + direction * (event.delta / 16);

        stage.update();
    }
```

Funkcija `update()` sada prihvata parametar `event`. Takav parametar čuva referencu na objekat događaja. Njegovo svojstvo `delta` koristi se prilikom računanja prostora za koji je potrebno pomeriti krug. Kada je razmak između dve iteracije 16ms, krug se pomera za po jedan piksel. Povećavanjem perioda između iteracija, povećava se i prostor za koji se pomera krug.

Primer - Animacija kretanja kruga

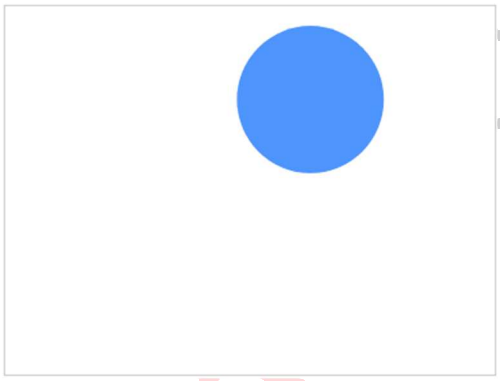
U nastavku će biti prikazana animacija slobodnog kretanja kruga kao u jednoj od prethodnih lekcija, ali ovoga puta korišćenjem EaselJS biblioteke i `Ticker` klase.

Kod kompletnog primera izgleda ovako:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta   name="viewport"   content="width=device-width,   initial-
scale=1.0">
    <title>CreateJS - Circle Animation</title>
    <style>
        #my-canvas {
            border: #cacaca 1px solid;
        }
    </style>
</head>
<body>
    <canvas id="my-canvas" width="400" height="300">
        Your web browser does not support canvas element.
    </canvas>
    <script
src="https://code.createjs.com/1.0.0/easeljs.min.js"></script>
    <script>
        window.onload = init;
        let stage;
        let circle;
        let circleRadius = 60;
        let speed = 2;
        let directionX = 1;
        let directionY = 1;
        function init() {
            stage = new createjs.Stage("my-canvas");
            let g = new createjs.Graphics();
            g.beginFill("#4F95FF");
            g.drawCircle(0, 0, circleRadius);
            circle = new createjs.Shape(g);
            stage.addChild(circle);
            circle.x      =      Math.floor(randomNumber(circleRadius,
stage.canvas.width - (circleRadius)));
            circle.y      =      Math.floor(randomNumber(circleRadius,
stage.canvas.height - (circleRadius)));
            createjs.Ticker.addEventListener("tick", update);
            createjs.Ticker.interval = 16;
        }
        function update(event) {
            if (circle.x > stage.canvas.width - circleRadius)
                directionX = -directionX;
            if (circle.x < circleRadius)
                directionX = -directionX;
            if (circle.y > stage.canvas.height - circleRadius)
                directionY = -directionY;
            if (circle.y < circleRadius)
```

```
                    directionY = -directionY;
                circle.x = circle.x + directionX * (event.delta / 16) *
    speed;
                circle.y = circle.y + directionY * (event.delta / 16) *
    speed;
                stage.update();
            }
            function randomNumber(min, max) {
                return Math.random() * (max - min) + min;
            }
        </script>
    </body>
    </html>
```

Efekat je kao na animaciji 13.2.



Animacija 13.2 - Animacija slobodnog kretanja kruga

Kod primera možete da preuzmete sa sledećeg linka:

`createjs_circle_animation.rar`

TweenJS

Pored `Ticker` klase, animacija se korišćenjem CreateJS skupa funkcionalnosti može postići na još jedan način, koji podrazumeva korišćenje nezavisne biblioteke koja se zove TweenJS.

Slika 13.1- TweenJS (izvor https://www.createjs.com/)

TweenJS je biblioteka koja je namenjena postizanju takozvane *Tween animacije*. Reč je o skraćenici za pojam *Inbetweening*, odnosno za proces kojim se automatski obavlja generisanje svih međustanja kojima se dočarava prelazak nekog grafičkog elementa iz jednog u neko drugo vizuelno stanje.

S obzirom da je reč o zasebnoj biblioteci iz skupa CreateJS funkcionalnosti, TweenJS je neophodno uključiti u HTML dokument u kome će se koristiti:

```
<script src="https://code.createjs.com/1.0.0/tweenjs.min.js"></script>
```

Nakon uključivanja, moguće je preći na korišćenje funkcionalnosti takve biblioteke. Efekat identičan onom iz uvodnog primera ove lekcije, korišćenjem biblioteke TweenJS je moguće postići na sledeći način:

```
window.onload = init;

let circle;
let stage;

function init() {
    stage = new createjs.Stage("my-canvas");

    let g = new createjs.Graphics();
    g.beginFill("#4F95FF");
    g.drawCircle(100, 100, 80);
    circle = new createjs.Shape(g);
    stage.addChild(circle);

    createjs.Tween.get(circle, {loop: true}).to({x: 50}, 1000).to({x: 0},
1000);

    createjs.Ticker.framerate = 60;
    createjs.Ticker.addEventListener("tick", stage);

}
```

Nakon koda koji smo već imali prilike da vidimo u uvodnom delu ove lekcije, definišu se i dve naredbe kojima se utvrđuju osobine Tween animacije i obavlja pokretanje takve animacije. Efekat će biti identičan kao onaj prikazan animacijom 13.1.

Bitno je da primetite da se u primeru ne obavlja eksplicitno pozivanje `update()` metode za ažuriranje pozornice. Takav posao se sada obavlja na nešto drugačiji način, Koristi se logika za pretplatu na `tick` događaj, ali se umesto reference na funkciju koja će takav događaj obraditi, prosleđuje referenca na objekat koji predstavlja pozornicu.

Definisanje osobina Tween animacije, funkcioniše na sledeći način:

- koristi se klasa `Tween` koja se nalazi unutar prostora imena `createjs`
- metodom `get()` obavlja se definisanje objekta nad kojim će biti primenjena animacija
- korišćenjem metode `to()` obavlja se definisanje ključnih tačaka animacije; definiše se svojstvo koje će biti animirano i njegova nova vrednost; to praktično znači da će početna vrednost svojstva biti ona koju element trenutno ima, a završna ona koja se definiše korišćenjem metode `to()`; pored svojstva i vrednosti, korišćenjem metode `to()` obavlja se i definisanje dužine tranzicije

Napomena

TweenJS biblioteka nije ograničena na animiranje objekata koji se crtaju unutar canvas elementa. Takvu biblioteku je moguće koristiti za animiranje bilo kog CSS ili JavaScript svojstva, pa tako i za animiranje regularnih HTML elemenata.

Rad sa resursima

Nakon demonstracije različitih pristupa za postizanje animacije, nastavak lekcije biće posvećen radu sa resursima, pri čemu se misli na eksterne fajlove koji se mogu koristiti prilikom kreiranja bogatog multimedijalnog sadržaja. Pri tom se prevashodno misli na slike i zvučne efekte. Prvo ćemo se upoznati sa različitim načinima za crtanje slika i reprodukciju zvučnih efekata, a nakon toga će biti prikazano korišćenje jednog posebnog sistema za efikasno učitavanje takvih resursa.

Crtanje slika

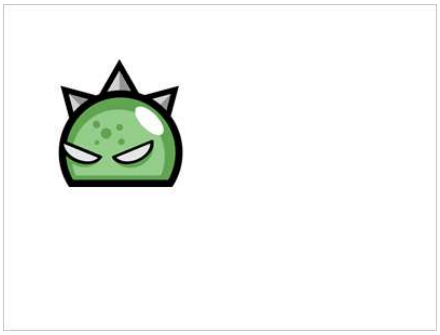
Korišćenjem CreateJS-a, slike je moguće crtati uz pomoć klase `Bitmap`, biblioteke EaselJS.

```
let stage = new createjs.Stage("my-canvas");

let bitmap = new createjs.Bitmap("enemy.png");
bitmap.x = 50;
bitmap.y = 50;
stage.addChild(bitmap);

stage.update();
```

U primeru je obavljeno instanciranje klase `Bitmap`. Tom prilikom je konstruktoru prosleđena putanja na kojoj se nalazi slika. Zatim su definisane koordinate na kojima će se unutar canvas-a naći slika i `Bitmap` objekat je dodeljen pozornici. Ipak, ukoliko primer isprobate unutar web pregledača, videćete da se slika ne prikazuje unutar canvas elementa. Ponašanje je očekivano, ukoliko se prisetimo da sliku nije moguće nacrtati ukoliko njeno učitavanje prethodno nije završeno u potpunosti. Stoga je primer potrebno transformisati na sledeći način:

```
window.onload = init;

function init() {
    let image = new Image();
    image.onload = imageLoaded;
    image.src = "enemy.png";
}

function imageLoaded(event) {
    let stage = new createjs.Stage("my-canvas");
    let bitmap = new createjs.Bitmap(event.target);
    bitmap.x = 50;
    bitmap.y = 50;
    stage.addChild(bitmap);

    stage.update();

}
```

Sada se obavlja logika koja je već viđena u prethodnim lekcijama. Kreira se `Image` objekat i obavlja se pretplata na `load` događaj. Kada se slika učita, aktivira se funkcija `imageLoaded()` i tek tada se izvršava kod koji je prikazan u prethodnom primeru. U `canvas` elementu se prikazuje slika (slika 13.2).



Slika 13.2 - Slika nacrtana EaselJS bibliotekom

S obzirom da koristimo CreateJS skup funkcionalnosti, logično je da ćemo ovakve poslove obavljati korišćenjem jedne posebne biblioteke koja je sastavni deo CreateJS skupa. Reč je o biblioteci PreloadJS.

PreloadJS

PreloadJS je jedna od biblioteka koje pripadaju CreateJS skupu. Namenjena je učitavanju resursa koji će biti korišćeni od strane HTML dokumenta. Pod pojmom resursa misli se na širok dijapazon fajlova različitih tipova (slike, zvukovi, tekst, css, JavaScript, JSON, XML, SVG...).

Slika 13.3 - PreloadJS (izvor https://www.createjs.com/)

Osnovna prednost PreloadJS-a, jeste centralizovani sistem za učitavanje većeg broja resursa i dobijanje dojave kada se takvo učitavanje završi. Vi ste u prethodnim modulima ovoga kursa, mogli da vidite koliki izazov može predstavljati obavljanje takvog posla i to konkretno na primeru razvoja *Jump & Run* igre. Takva igra je koristila veliki broj spritesheet-ova i zvukova i Vi ste mogli da vidite da smo mi morali samostalno da kreiramo sistem uz pomoć koga bismo dobili dojavu o završetku učitavanja svih takvih resursa. Obavljanje takvog posla značajno je olakšano korišćenjem PreloadJS biblioteke.

Prvi korak u korišćenju PreloadJS-a, jeste uključivanje takve biblioteke u HTML dokument:

```
<script src="https://code.createjs.com/1.0.0/preloadjs.min.js"></script>
```

Definisanje logike za učitavanje jednog fajla, može se obaviti korišćenjem sledećih naredbi:

```
let queue = new createjs.LoadQueue();
queue.addEventListener("fileload", imageLoaded);
queue.loadFile("runner.png");
```

Prvo se obavlja instanciranje klase `LoadQueue`. Nad objektnom ove klase obavlja se pretplata na događaj `fileload` i tom prilikom se prilaže referenca na funkciju koja će se izvršiti kada se učitavanja fajla završi. Iniciranje učitavanja se obavlja pozivanjem metode `loadFile()` kojoj se prilaže putanja do fajla. Kada se učitavanje završi, aktiviraće se funkcija `imageLoaded()`.

Naravno, prava moć PreloadJS biblioteke dolazi do izražaja kada je potrebno učitati veći broj fajlova. Takav primer imaćete prilike da vidite u nastavku ove lekcije.

Rad sa spritesheet-ovima

Skup funkcionalnosti za rad sa slikama, biblioteka EaselJS dodatno proširuje mogućnošću direktnog rada sa sprajtovima i spritesheet-ovima. Ova dva pojma su nama poznata iz prethodnih lekcijama u kojima smo kreirali *Jump & Run* igru. Tom prilikom ste mogli da vidite da smo logiku za rad sa sprajtovima morali samostalno da napišemo. Kada se koristi CreateJS, tako nešto nije obaveza, pa je rad sa ovom vrstom rasterske grafike značajno uprošćen.

Klasa SpriteSheet

Logika za rad sa spritesheet-ovima enkapsulirana je unutar klase `SpriteSheet`. konstruktor ove klase izgleda ovako:

```
SpriteSheet (data)
```

Konstruktor `SpriteSheet` klase prihvata jedan objekat kojim se konfiguriše `SpriteSheet` koji će biti kreiran. Spritesheet je moguće konfigurisati sledećim svojstvima:

- `images` - obavezno svojstvo kojim se definiše jedna ili više slika od kojih je sačinjen spritesheet; mi smo do sada koristili spritesheet-ove sačinjene iz jedne slike i to je najčešća situacija u praksi; ipak, EaselJS omogućava da se spritesheet kreira korišćenjem većeg broja nezavisnih slika
- `frames` - obavezno svojstvo kojim se definišu pojedinačni kadrovi jednog spritesheet-a
- `animations` - svojstvo kojim se definišu sekvence kadrova koji će biti korišćeni za konstruisanje animacije
- `framerate` - svojstvo kojim se definiše frekvencija osvežavanja animacije koja nastaje korišćenjem spritesheet-a

Primer kreiranja jednog spritesheet-a, može da izgleda ovako:

```
let data = {
    images: ["runner.png"],
    frames: {
        width: 153,
        height: 199
    },
    animations: {
        stand: {
            frames: [0, 1],
            speed: 0.5
        },
        run: {
            frames: [2, 6],
            speed: 0.3
        },
        jump: {
            frames: [7, 8],
            next: "run",
            speed: 0.5
        }
    }
};

let spriteSheet = new createjs.SpriteSheet(data);
```

Prikazanim kodom je prvo obavljeno kreiranje jednog objekta za konfigurisanje spritesheet-a, koji smo već koristili u jednoj od prethodnih lekcija. Reč je o spritesheet-u kojim se predstavlja glavni karakter *Jump & Run* igre.

Korišćenjem svojstva `images`, definisana je putanja na kojoj se nalazi slika koja predstavlja spritesheet. Dimenzije pojedinačnih kadrova, definisane su upotrebom svojstva `frames`. Definisana je i vrednost svojstva `animations`. Takvim svojstvom su kreirana tri stanja u kojima se može naći karakter koji se predstavlja ovakvim spritesheet-om. Nešto kasnije ćete videti kako se koriste ovako definisane animacije.

Kako bi se grafika predstavljena `SpriteSheet` klasom, prikazala unutar pozornice, neophodno je koristiti klasu `Sprite`.

Klasa Sprite

Klasa `Sprite` koristi se za prikaz grafike definisane spritesheet-om. Pri tom, klasa `Sprite` omogućava prikaz jednog statičkog kadra, ali i sekvence kadrova, odnosno animacije, koja nastaje brzim smenjivanjem kadrova jednog spritesheet-a.

Konstruktor klase `Sprite` izgleda ovako:

```
Sprite (spriteSheet, [frameOrAnimation])
```

Referenca na `SpriteSheet` objekat je obavezan parametar, dok se drugi parametar može, ali i ne mora proslediti. Drugim parametrom je moguće proslediti informacije o animaciji koja će da započne odmah nakon konstruisanja objekta. Ukoliko se prosledi broj, biće to indeks prvog kadra animacije, a ukoliko se prosledi `string`, ona će se odnositi na naziv animacije koji je definisan unutar `animations` svojstva `SpriteSheet` klase.

Kadrove koje će `Sprite` prikazivati moguće je kontrolisati i korišćenjem nekoliko metoda:

- `play()` - započinje prikaz animacije
- `stop()` - zaustavljanja izvršavanja animacije
- `gotoAndPlay(frameOrAnimation)` - započinje prikaz neke konkretne animacije, koja se prosledi kao parametar; kada se prosledi broj kadra, animacija započinje od tog kadra
- `gotoAndStop(frameOrAnimation)` - zaustavlja izvršavanje animacije i pozicionira se na kadar sa prosleđenim indeksom ili prvi kadar animacije sa prosleđenim imenom

Nakon kreiranja `SpriteSheet` objekta u prethodnim redovima, moramo obaviti konstruisanje i jednog objekta tipa `Sprite`:

```
let theRunner = new createjs.Sprite(spriteSheet);
theRunner.gotoAndPlay('stand');
stage.addChild(theRunner);
```

Prikazanim naredbama, prvo se obavlja kreiranje `Sprite` objekta. Pokreće se animacija koja je imenovana nazivom `stand` i na kraju se kreirani `Sprite` dodaje pozornici. Ipak, s obzirom da je reč animaciji, neophodno je definisati i logiku koja će kontinuirano da obavlja ažuriranje pozornice:

```
createjs.Ticker.on("tick", handleTick);

function handleTick(event) {
      stage.update(event);
}
```

Pitanje

Klasa koja omogućava da se grafika jednog spritesheet-a prikaže unutar pozornice, zove se:

a) Sprite
b) Image
c) Picture
d) Blob

Objašnjenje:

Klasa Sprite koristi se za prikaz grafike definisane spritesheet-om. Pri tom, klasa Sprite omogućava prikaz jednog statičkog kadra, ali i sekvence kadrova, odnosno animacije, koja nastaje brzim smenjivanjem kadrova jednog spritesheet-a.

SoundJS

SoundJS je još jedna biblioteka iz skupa CreateJS funkcionalnosti. Ona olakšava rad sa zvučnim zapisima.

Slika 13.4 - SoundJS (izvor https://www.createjs.com/)

Uključivanje SoundJS biblioteke se može obaviti korišćenjem sledeće linije:

```
<script src="https://code.createjs.com/1.0.0/soundjs.min.js"></script>
```

Nakon uključivanja, potrebno je definisati putanje na kojima se nalaze zvučni zapisi:

```
let audioPath = "sounds/";
let sounds = [
      { id: "Click", src: "click.wav" },
      { id: "Jump", src: "jump.wav" },
      { id: "Fail", src: "fail.wav" }
];
```

Unutar `sounds` niza definisana su 3 objekta koja definišu osobine 3 zvučna zapisa, koja se nalaze unutar foldera `sounds`. Iniciranje učitavanja se može obaviti na sledeći način:

```
createjs.Sound.addEventListener("fileload", handleLoad);
createjs.Sound.registerSounds(sounds, audioPath);
```

Na ovaj način, kada se svaki od definisanih zvukova učita, biće aktivirana funkcija sa nazivom `handleLoad()`:

```
function handleLoad(event) {
      //handle sound loaded; use event.src to find out which sound is
loaded
}
```

Bitno je znati da će se funkcija `handleLoad()` aktivirati nakon učitavanja svakog pojedinačnog zvuka, odnosno tri puta u našem primeru. Za dobijanje putanje do zvuka koji je upravo učitan, može se koristiti `src` svojstvo objekta događaja.

Reprodukcija zvuka se obavlja pozivanjem metode `play()`. Njoj je moguće proslediti putanju do zvuka ili `id` vrednost, koja je zvuku dodeljena prilikom inicijalizacije:

```
createjs.Sound.play('Jump');
```

SoundJS i PreloadJS

S obzirom da smo se već upoznali sa PreloadJS bibliotekom, najbolja praksa jeste korišćenje objedinjene logike za učitavanje svih resursa koji će biti potrebni HTML dokumentu na kome radimo. Stoga je najbolje koristiti PreloadJS i za učitavanje zvučnih zapisa. Evo kako se tako nešto može obaviti:

```
let queue = new createjs.LoadQueue();
queue.installPlugin(createjs.Sound);
queue.addEventListener("fileload", assetLoaded);
queue.addEventListener("complete", allAssetsLoaded);

queue.loadFile("runner.png");
queue.loadFile({
    id: "Click",
    src: "sounds/click.wav"
});
queue.loadFile({
    id: "Jump",
    src: "sounds/jump.wav"
});
queue.loadFile({
    id: "Fail",
    src: "sounds/fail.wav"
});
```

Primer ilustruje logiku za učitavanje jedne slike i tri zvučna zapisa, korišćenjem PreloadJS biblioteke.

Kako bi PreloadJS znao kako da učita i obavi integraciju učitanog zvučnog zapisa, neophodno je aktivirati *plugin* koji je namenjen za obavljanje takvog posla. To se u prikazanom primeru obavlja korišćenjem druge naredbe u kojoj se poziva metoda `installPlugin()` i njoj se prosleđuje referenca do `Sound` klase, `createjs` prostora imena.

U primeru je obavljena pretplata na dva događaja:

- `fileload` - aktiviraće se prilikom završetka učitavanja svakog fajla
- `complete` - aktiviraće se kada se završi učitavanje svih fajlova

Događaj `complete` je više nego koristan, jer nam daje signal kada su svi resursi učitani, što je posebno značajno, pogotovu prilikom razvoja igara.

Na kraju, s obzirom da smo aktivirali Sound plugin PreloadJS biblioteke, nakon učitavanja zvučnih zapisa, za njihovu reprodukciju je dovoljno pozivati metodu `play()` kao i do sada:

```
function allAssetsLoaded(event) {

            createjs.Sound.play('Jump');
}
```

Primer - Jump & Run

Za kraj ove lekcije biće prikazan još jedan primer kojim ćemo pokušati da sumiramo tehnike za rad sa spritesheet-ovima, koje su prikazane u prethodnim redovima. Iskoristićemo grafiku iz *Jump & Run* igre, ali će umesto pokretne pozadine, korisniku sada biti omogućeno da direktno upravlja glavnim karakterom igre. Glavni karakter će moći da se kreće levo i desno i da obavlja skok. Sve to će izgledati kao na videu 13.1.

https://youtu.be/cFbfWGU6A00

Video 13.1 – Ponašanje koje je postignuto primerom

Kompletan kod za realizaciju ovakvog primera, možete da preuzmete sa sledećeg linka:

`jump_and_run.rar`

Projekat je neophodno da podignete na neki HTTP server kako bi mogao da funkcioniše. Naime, to je slučaj i sa svim ostalim projektima u kojima ćete koristiti PreloadJS. Web pregledač blokira direktan pristup fajlovima koji se na taj način učitavaju, pa je neophodno koristiti HTTP server.

Rezime

- grafika koja se crta korišćenjem EaselJS biblioteke, može se animirati korišćenjem izvornih tajming funkcija
- klasa `Ticker` predstavlja apstrakciju oko izvornih tajming funkcija web pregledača
- klasa `Ticker` omogućava kontinuirano pozivanje određene logike, sa unapred definisanim razmakom između dva takva pozivanja
- kako bi se određena logika prosledila na kontinuirano izvršavanje korišćenjem klase `Ticker`, dovoljno je obaviti pretplatu na `tick` događaj
- klasa `Ticker`, podrazumevano emituje `tick` događaj 20 puta u jednoj sekundi, odnosno na svakih 50ms
- svojstvom `interval`, `Ticker` klase, definiše se vremenski razmak između dva susedna emitovanja `tick` događaja
- svojstvom `framerate`, `Ticker` klase, definiše se frekvenciju osvežavanja animacije (*fps, engl.*)
- funkciji za obradu `tick` događaja, automatski se prosleđuje i svojstvo `delta`, koje sadrži vreme koje je proteklo od prethodnog emitovanja `tick` događaja
- `Ticker` klasa u pozadini, podrazumevano koristi `setTimeout()` metodu, što je moguće promeniti korišćenjem svojstva `timingMode`
- TweenJS je jedna od biblioteka iz CreateJS skupa, koja je namenjena postizanju takozvane *Tween animacije*
- za postizanje Tween animacije, koristi se `Tween` klasa
- korišćenjem CreateJS-a, slike je moguće crtati uz pomoć klase `Bitmap`, biblioteke EaselJS
- PreloadJS biblioteka je namenjena učitavanju resursa koji će biti korišćeni od strane HTML dokumenta
- EaselJS omogućava direktan rad sa sprajtovima i spritesheet-ovima
- logika za rad sa spritesheet-ovima enkapsulirana je unutar klase `SpriteSheet`
- klasa `Sprite` koristi se za prikaz grafike definisane spritesheet-om
- SoundJS je biblioteka koja olakšava rad sa zvučnim zapisima