Transformacija i animacija

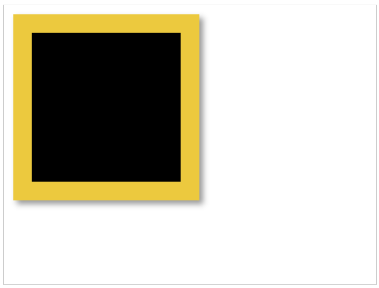
U prethodnim lekcijama ilustrovani su različiti načini za crtanje unutar canvasa. Imali ste prilike da vidite kako se unutar canvas elementa crta i stilizuje proizvoljna grafika, ali i kako se obavlja crtanje već pripremljene grafike predstavljene slikama. U ovoj lekciji biće ilustrovano još nekoliko veoma važnih pristupa kada je u pitanju rad sa canvas grafikom. Takvi pristupi će nam omogućiti da određene aspekte rada sa canvasom dodatno uprostimo. Stoga će u ovoj lekciji prvo biti reči o rukovanju stanjem canvas elementa, a zatim i o različitim transformacijama koje je moguće sprovesti nad koordinatnim sistemom canvasa. Sve to će postaviti temelje za sprovođenje animacije nad canvas grafikom.

Rukovanje stanjem canvasa

U jednoj od prethodnih lekcija imali ste prilike da vidite šta podrazumeva proces stilizacije oblika koji se crtaju. Jednostavno, koriste se različita svojstva konteksta za crtanje, čijim vrednostima se utiče na osobine grafike koja će biti nacrtana nakon takvih intervencija. Takođe, mogli ste da vidite da se jednom definisane vizuelne osobine primenjuju nad svom grafikom koja se crta u budućnosti i da, ukoliko želimo da crtamo više uzastopnih oblika različitih vizuelnih karakteristika, moramo da neposredno pre crtanja obavimo redefinisanje stilizacije. Evo jednog takvog primera:

```
ctx.fillStyle = "#ECC93E";
ctx.shadowOffsetX = 4;
ctx.shadowOffsetY = 4;
ctx.shadowBlur = 6;
ctx.shadowColor = 'rgba(0, 0, 0, 0.4)';

ctx.fillRect(10, 10, 200, 200);

ctx.fillStyle = "black";
ctx.shadowOffsetX = 0;
ctx.shadowOffsetY = 0;
ctx.shadowBlur = 0;
ctx.shadowColor = 'rgba(0, 0, 0, 0)';

ctx.fillRect(30, 30, 160, 160);
```

Prikazanim kodom obavlja se crtanje dva kvadrata. Drugi kvadrat se crta preko prvog. Pritom je prvi kvadrat žute boje i poseduje senku. Drugi kvadrat je crne boje i ne poseduje senku (slika 8.1).



Slika 8.1. Dva kvadrata nacrtana jedan preko drugog

Ono što odmah možete da primetite jeste to da drugi (crni) pravougaonik poseduje podrazumevane vizuelne karakteristike, što praktično znači da, ukoliko bismo crtali samo njega, ne bismo bili u obavezi da definišemo bilo kakvu stilizaciju. Ipak, pošto smo prvo nacrtali žuti pravougaonik sa senkom, dalje je bilo neophodno da pre crtanja drugog kvadrata ponovno redefinišemo vrednosti svih svojstava za stilizovanje kako se identična stilizacija ne bi primenila i nad drugim pravougaonikom. Ovo je samo jedan banalan primer koji ilustruje na koji način funkcioniše definisanje različite stilizacije za više oblika koji se uzastopno crtaju. Potpuno je jasno da se u realnim okolnostima mogu javiti još kompleksniji primeri, koji bi od nas zahtevali dosta ponavljanja koda.

Kako bi se u ovakvim situacijama olakšalo crtanje grafike, `canvas` poznaje pojam stanja. Stanje se odnosi na sve vizuelne osobine koje se definišu korišćenjem različitih svojstava za stilizovanje. Canvas API poseduje i dve metode koje nama omogućavaju da rukujemo takvim stanjem:

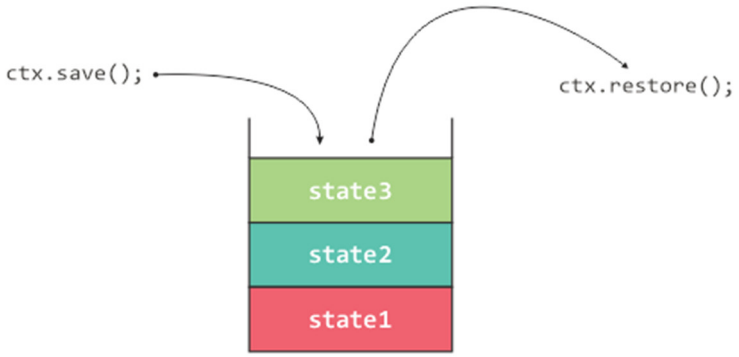
- `save()` – čuva stanje `canvas`a,
- `restore()` – vraća `canvas` u prethodno stanje.

Kako u praksi izgleda rukovanje stanjem i šta nam ono može doneti, biće prikazano na već viđenom primeru crtanja dva kvadrata:

```
ctx.save();

ctx.fillStyle = "#ECC93E";
ctx.shadowOffsetX = 4;
ctx.shadowOffsetY = 4;
ctx.shadowBlur = 6;
ctx.shadowColor = 'rgba(0, 0, 0, 0.4)';

ctx.fillRect(10, 10, 200, 200);

ctx.restore()
ctx.fillRect(30, 30, 160, 160);
```

Odmah na početku primera obavlja se pozivanje metode `save()`, koja obavlja čuvanje stanja `canvas` elementa. Zatim se nastavlja sa već viđenim postupkom stilizovanja i crtanja prvog kvadrata. Kada se crtanje prvog kvadrata završi, više se ne obavlja ručno resetovanje svih svojstava za stilizaciju koja su upotrebljena za stilizovanje prvog kvadrata. Umesto takvih pet naredbi koje su korišćene u prethodnom primeru, sada se obavlja samo pozivanje metode `restore()` i `canvas` se automatski vraća u stanje u kojem je bio prilikom prethodnog pozivanja metode `save()`. S obzirom na to da je metoda `save()` pozvana pre definisanja bilo kakve stilizacije, drugi kvadrat se crta sa podrazumevanim postavkama. Na ovaj način smo na efikasan način uprostili kod koji pišemo, tako što smo 5 naredbi zamenili pozivom samo jedne metode.

Stanje `canvas` elementa je u pozadini realizovano kao stek (engl. *stack*). To je struktura podataka kod koje se poslednje dodati element prvi uklanja (slika 8.2).

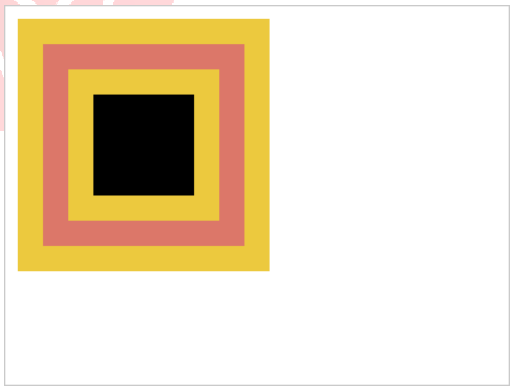


Slika 8.2. Stek stanja canvasa

Sa slike 8.2. možete videti na koji način funkcioniše stek stanja canvas elementa. Možete videti i to da je unutar steka stanja canvas elementa moguće upisivati veći broj stanja, što praktično znači da je metode `save()` i `restore()` moguće pozivati veći broj puta:

```
ctx.save();

ctx.fillStyle = "#ECC93E";
ctx.fillRect(10, 10, 200, 200);

ctx.save();

ctx.fillStyle = "#DC7769";
ctx.fillRect(30, 30, 160, 160);

ctx.restore();
ctx.fillRect(50, 50, 120, 120);

ctx.restore();
ctx.fillRect(70, 70, 80, 80);
```

Ovakav kod proizvodi rezultat kao na slici 8.3.



Slika 8.3. Četiri kvadrata, jedan preko drugog

Isto kao i u prethodnom primeru, odmah na početku se obavlja čuvanje inicijalnog stanja canvas elementa. Nakon toga se obavlja crtanje žutog kvadrata, ali se sada nakon njegovog crtanja obavlja još jedno čuvanje stanja. Stoga nakon drugog pozivanja metode `save()`, unutar steka postoje dva upisana stanja canvas elementa. Sledeći korak jeste crtanje još jednog kvadrata, ovoga puta crvene boje. Poslednja dva kvadrata crtaju se na osnovu stilizacije koja je zapamćena unutar steka stanja. Tako se pre crtanja trećeg kvadrata poziva `restore()` metoda i kvadrat automatski dobija žutu boju. Po identičnom principu i poslednji, četvrti kvadrat dobija crnu boju, odnosno biva nacrtan korišćenjem podrazumevane stilizacije koja je zapamćena na samom početku primera, prvim pozivom `save()` metode.

Upravo prikazani pristupi za rukovanje stanjem canvasa posebno su korisni prilikom obavljanja transformacija nad koordinatnim sistemom canvas elementa, o čemu će biti reči u nastavku ove lekcije.

Transformacije

Canvas API poznaje nekoliko metoda kojima je moguće obavljati transformacije. Pritom se transformacije odnose na sam koordinatni sistem canvas elementa, čime je moguće uticati na njegove osobine. Postoje tri vrste transformacija koje je moguće sprovoditi:

- translacija (pomeranje),
- rotacija,
- promena veličine.

Za sprovođenje opisanih transformacija koristi se nekoliko metoda, ilustrovanih tabelom 8.1.

Transformacija	Metoda	Opis
Translacija (Translating)	`translate(x, y)`	pomera referentnu tačku koordinatnog sistema na neku drugu lokaciju, definisanu prosleđenim koordinatama
Rotacija (Rotating)	`rotate(angle)`	rotira koordinatni sistem canvas elementa oko referentne tačke, za ugao prosleđen kao parametar
Promena veličine (Scaling)	`scale(x, y)`	uvećavanja ili smanjuje jedinice koordinatnog sistema canvas elementa, čime se smanjuje ili uvećava grafika koja se crta

Tabela 8.1. Canvas transformacije

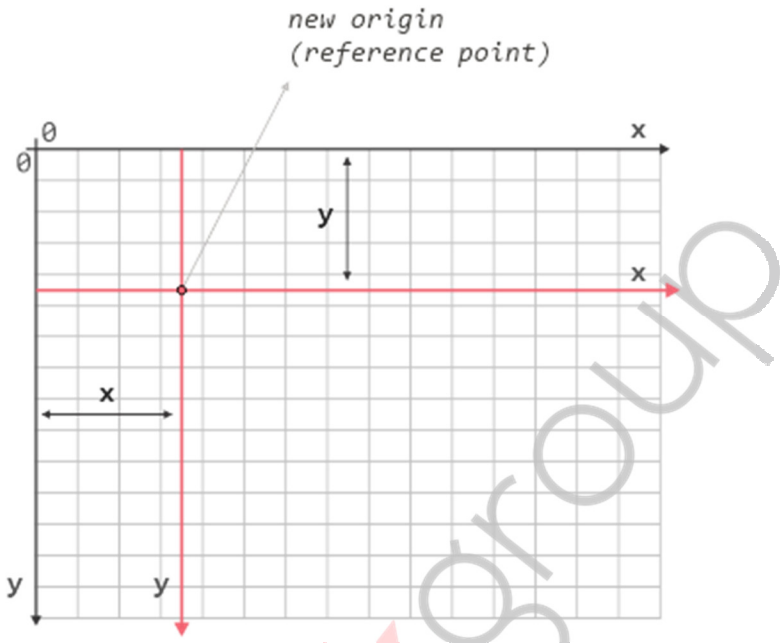
U nastavku ćemo se upoznati sa svakom od navedenih transformacija.

Translacija (pomeranje)

Translacija omogućava pomeranje kompletnog koordinatnog sistema canvas elementa, odnosno postavljanje njegove referentne tačke na neku novu poziciju. Translacija se obavlja korišćenjem istoimene metode, koja prihvata dva parametra:

```
translate(x, y)
```

Parametri x i y predstavljaju koordinate nove referentne tačke. Drugim rečima, x označava koliko će se koordinatni sistem pomeriti po horizontalnoj osi, a y, koliki će pomeraj biti po vertikalnoj osi (slika 8.4).



Slika 8.4. Translacija

U ovom trenutku možda ne možete da uvidite pravu moć translacije i sa pravom se možete pitati zbog čega bismo uopšte pomerali kompletan koordinatni sistem canvasa. Kako biste lakše razumeli moć ove transformacije, dat je sledeći primer:

```
ctx.fillStyle = "#ECC93E";
ctx.fillRect(10, 10, 120, 80);

ctx.save();

ctx.translate(0, 90);
ctx.fillRect(10, 10, 120, 80);

ctx.restore();

ctx.translate(0, 180);
ctx.fillRect(10, 10, 120, 80);
```

Ovakvim kodom dobijaju se tri pravougaonika (slika 8.5).

Slika 8.5. Tri pravougaonika

Šta na osnovu ovakvog primera možemo da zaključimo? Bitno je da primetite da su naredbe za crtanje sva tri pravougaonika identične, odnosno u sva tri slučaja metoda `fillRect()` prihvata identične parametre, što znači da se pravougaonici crtaju na identičnim pozicijama. Ipak, na slici 8.5. možete videti da se oni unutar `canvas` elementa nalaze jedan ispod drugog. To je postignuto pomeranjem koordinatnog sistema `canvasa`.

Prvi pravougaonik je nacrtan bez ikakvog pomeranja i tom prilikom su zabeležene osobine `canvasa`, čuvanjem stanja. Pre crtanja drugog pravougaonika, obavljena je transformacija pomeranja, tako što je referentna tačka koordinatnog sistema spuštena za `90px` nadole:

```
ctx.translate(0, 90);
```

Nakon ovakve transformacije, sva naredna crtanja biće relativna novim osobinama koordinatnog sistema `canvas` elementa. Upravo zbog toga, identična naredba za crtanje rezultuje crtanjem pravougaonika koji se nalazi ispod već nacrtanog.

Crtanje trećeg pravougaonika se obavlja po identičnoj logici. Ipak, pre transformisanja `canvas` se vraća u inicijalno stanje pozivanjem metode `restore()`, čime se obavlja resetovanje svih sprovedenih transformacija. To praktično znači da se referentna tačka vraća u gornji levi ugao `canvas` elementa. Ipak, odmah nakon vraćanja u inicijalno stanje, mi opet obavljamo transformisanje, ali ovoga puta referentnu tačku spuštamo za `180px`, što omogućava da se treći pravougaonik nađe ispod dva već nacrtana.

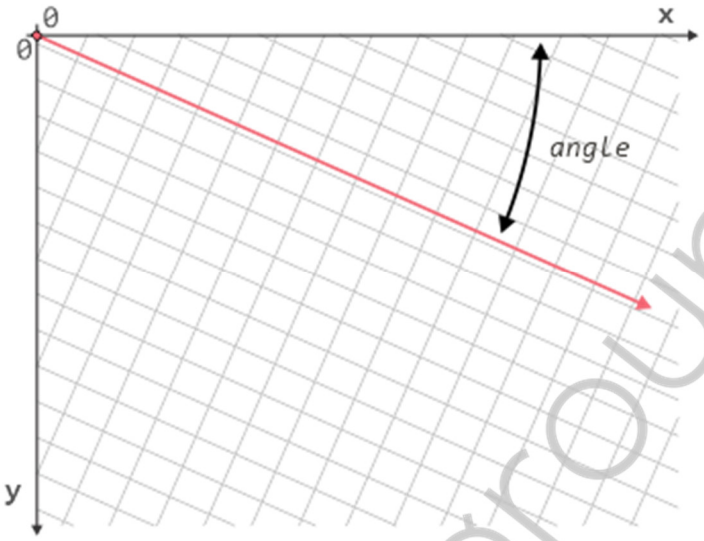
Napomena

Efekat prikazan slikom 8.5. mogao je biti dobijen i bez resetovanja stanja koordinatnog sistema `canvas` elementa između uzastopnih transformacija. Tada bi se svaka naredna transformacija obavljala nad već transformisanim koordinatnim sistemom, pa bi i poslednje pomeranje bilo potrebno obaviti za `90px`:

```
ctx.translate(0, 90);
```

Rotacija

Rotacija je transformacija koja omogućava rotiranje kompletnog koordinatnog sistema. Rotacija se obavlja oko referentne tačke, što podrazumevano izgleda kao na slici 8.6.



Slika 8.6. Rotacija

Rotacija se postiže korišćenjem metode `rotate()` koja prihvata jedan parametar:

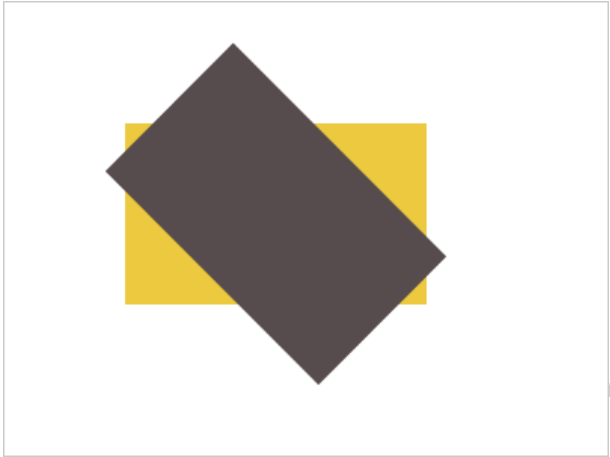
```
rotate(angle)
```

Parametar (`angle`) se odnosi na ugao rotacije i izražava se u radijanima. Radijani se u stepene mogu pretvoriti korišćenjem sledeće formule: `radians = (Math.PI/180)*degrees`.

Rotacija je retko korisna u svom izvornom obliku, odnosno kada se obavlja oko gornje leve ivice `canvas` elementa. Dobra stvar je što se referentna tačka može pomerati, i to upravo korišćenjem translacije sa kojom smo se upoznali u prethodnom poglavlju. Sve to nama omogućava da rotaciju obavljamo oko centara elemenata koji će biti nacrtani. Takvu rotaciju ilustrovaće naredni primer:

```
ctx.fillStyle = "#ECC93E";
ctx.fillRect(80, 80, 200, 120);

ctx.translate(180, 140);
ctx.rotate((Math.PI / 180) * 45);
ctx.translate(-180, -140);

ctx.fillStyle = '#574C4D';
ctx.fillRect(80, 80, 200, 120);
```

Prikazanim kodom se dobija efekat kao na slici 8.7.

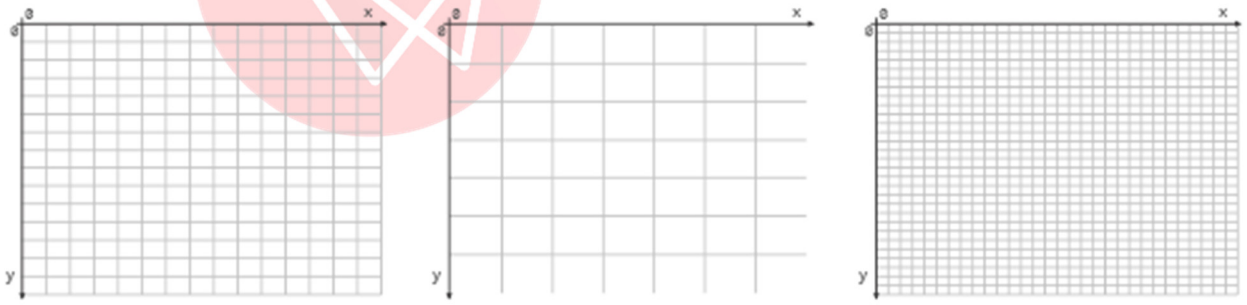


Slika 8.7. Primer rotacije

U prikazanom primeru se obavlja crtanje dva pravougaonika. Prvi se crta bez ikakvih transformacija i ima žutu boju pozadine. Drugi pravougaonik ima sivu boju pozadine, a sam poziv metode za crtanje (`fillRect()`) izgleda identično kao i onaj koji se obavlja crtanje žutog pravougaonika. Ipak, kao što možete videti, sivi pravougaonik je rotiran oko svog centra. Rotiranje oko centra pravougaonika je postignuto kombinacijom translacije i rotacije. Prvo je referentna tačka koordinatnog sistema pomerena u centar pravougaonika. Zatim je obavljeno rotiranje za 45 stepeni, a na kraju je referentna tačka vraćena na originalnu poziciju. Sve ovo nam je omogućilo da poslednjom naredbom obavimo crtanje pravougaonika koji je rotiran oko svog centra.

Promena veličine

Poslednja transformacija koju je moguće obavljati nad koordinatnim sistemom canvas elementa jeste promena veličine, odnosno smanjenje ili uvećanje jedinica od kojih je sačinjen koordinatni sistem (slika 8.8).



Slika 8.8. Promena veličine koordinatnog sistema; sleva nadesno – normalno, uvećano, umanjeno

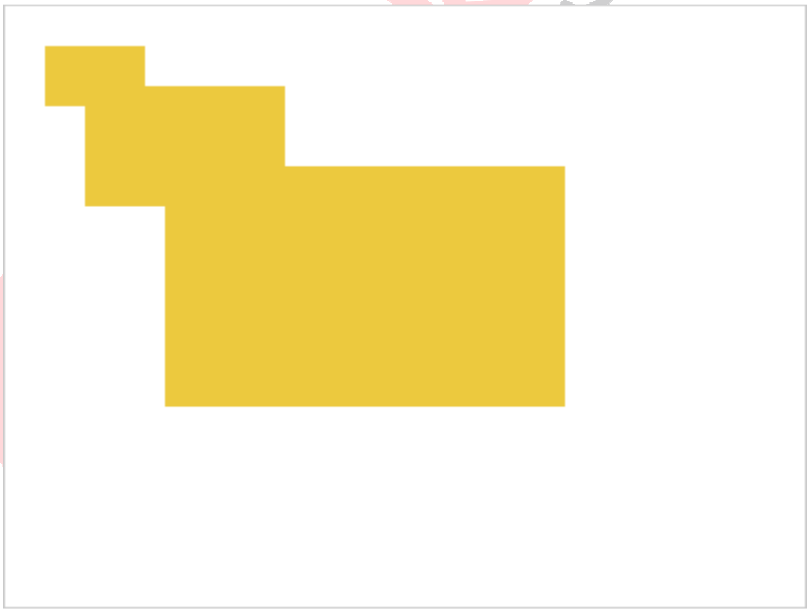
Promena veličine koordinatnog sistema obavlja se upotrebom metode `scale()`:

```
scale(x, y)
```

Metoda `scale()` prihvata dva parametra koji omogućavaju uticanje na veličinu koordinatnog sistema po horizontalnoj i vertikalnoj osi, nezavisno. Metodi `scale()` se prosleđuju numeričke vrednosti. Vrednost 1.0 označava originalnu veličinu. Sve vrednosti manje od 1.0 smanjuju, a vrednosti veće od 1.0 povećavaju veličinu `canvasa`:

```
ctx.fillStyle = "#ECC93E";
ctx.fillRect(80, 80, 200, 120);

ctx.save();

ctx.scale(0.5, 0.5);
ctx.fillRect(80, 80, 200, 120);

ctx.restore();

ctx.scale(0.25, 0.25);
ctx.fillRect(80, 80, 200, 120);
```

Na ovaj način dobija se prikaz kao na slici 8.9.



Slika 8.9. Primer promene veličine elemenata koji se crtaju

Prikazani primer podrazumeva crtanje tri pravougaonika. Sva tri pravougaonika crtaju se na identičan način, odnosno imaju identičnu referentnu tačku i veličinu. Ipak, možete da vidite da oni unutar `canvas` elementa ne izgledaju identično. Razlog je transformacija kojom je vršena promena veličine koordinatnog sistema.

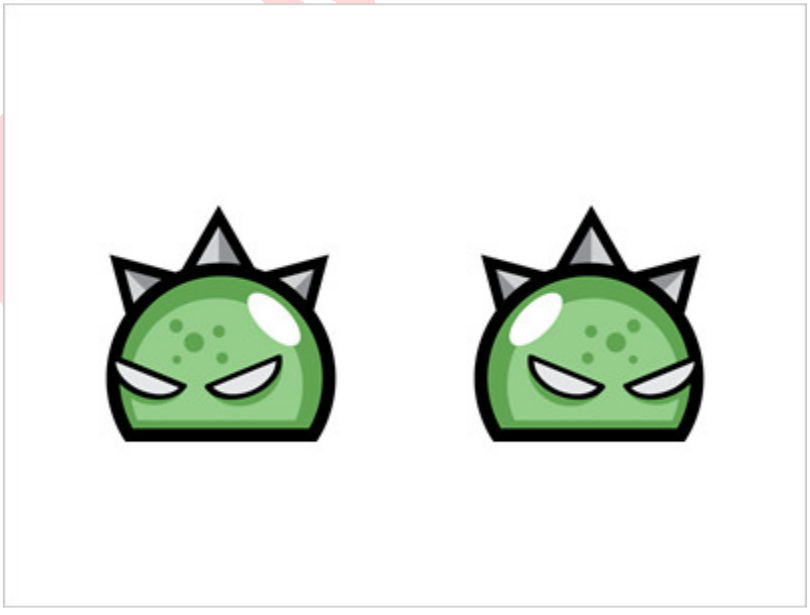
Prvi pravougaonik je nacrtan bez ikakvih transformacija. Drugi je nacrtan nakon što je koordinatni sistem smanjen dva puta. Na kraju, pre crtanja trećeg pravougaonika, koordinatni sistem je smanjen 4 puta.

Bitno je da primetite i to da je nakon crtanja prvog pravougaonika obavljeno čuvanje stanja koordinatnog sistema dok je on još bio u izvornom obliku, pre bilo kakvih transformacija. To nam je omogućilo da nakon crtanja drugog pravougaonika canvas vratimo u originalno stanje. Da tako nešto nije učinjeno, drugom transformacijom canvas ne bi bio smanjen 4, već 8 puta, zato što bi bilo obavljano smanjivanje već smanjenog canvasa.

Transformaciju koja omogućava promenu veličine koordinatnog sistema canvasa moguće je koristiti za obavljanje preslikavanja nekog oblika:

```
let img = new Image();

img.addEventListener('load', function () {

    ctx.drawImage(img, 50, 100);

    ctx.scale(-1, 1);
    ctx.translate(-myCanvas.width, 0);

    ctx.drawImage(img, 50, 100);

}, false);

img.src = 'enemy.png';
```

Ovaj kod proizvodi efekat kao na slici 8.10.



Slika 8.10. Primer reflektovanja jedne slike

U prikazanom primeru obavlja se okretanje, odnosno reflektovanje (engl. *flip*) jedne slike. Pod tim se podrazumeva dobijanje nove slike koja predstavlja refleksiju nastalu na osnovu originalne slike.

Efekat je postignut korišćenjem metode `scale()`, gde je za prvi parametar prosleđena vrednost -1. Na taj način će koordinati sistem u potpunosti biti okrenut horizontalno i izaći će iz vidljivih okvira `canvas` elementa. Zbog toga je u primeru, nakon ove transformacije, obavljena i translacija, tako što je kompletan koordinatni sistem pomeren u levo za celu širinu `canvas` elementa. Tako je okrenuta slika vraćena u vidljive okvire `canvas` elementa.

Primer – crtanje Pie Chart dijagrama (2. način)

Ono što smo naučili o transformacijama sada će biti primenjeno za kreiranje *Pie Chart* dijagrama ne nešto drugačiji način od onoga koji je već prikazan u jednoj od prethodnih lekcija. Evo kako će izgledati funkcija za crtanje dijagrama, čija logika će u ovom slučaju biti kreirana upotrebom transformacija:

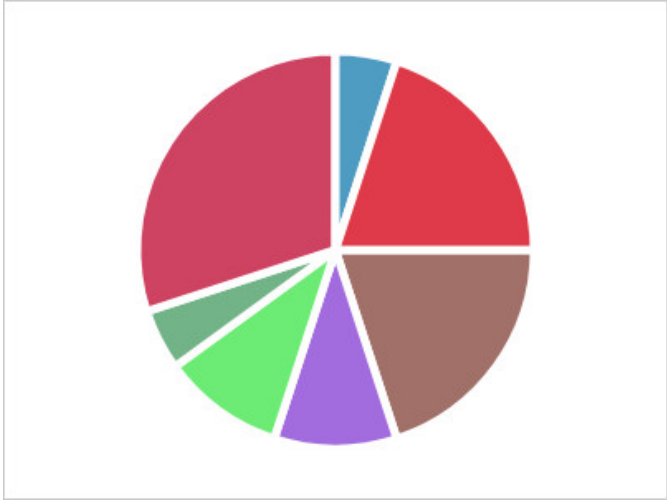
```
function makePieChart(ctx, shares) {
    let sum = shares.reduce((a, b) => a + b, 0);
    if (sum !== 100)
        throw new Error("Sum of all shares must be 100.");
    for (let i = 0; i < shares.length; i++) {
        let angle = (Math.PI / 180) * shares[i] * 3.6;
        let red = Math.floor(Math.random() * 200) + 50;
        let green = Math.floor(Math.random() * 200) + 50;
        let blue = Math.floor(Math.random() * 200) + 50;
        ctx.fillStyle = 'rgb(' + red + ', ' + green + ', ' + blue +
')';
        ctx.beginPath();
        ctx.arc(200, 150, 120, 0, angle);
        ctx.lineTo(200, 150);
        ctx.fill();
        ctx.strokeStyle = "white";
        ctx.lineWidth = 5;
        ctx.beginPath();
        ctx.arc(200, 150, 120, 0, angle);
        ctx.lineTo(200, 150);
        ctx.closePath();
        ctx.stroke();
        ctx.translate(200, 150);
        ctx.rotate(angle);
        ctx.translate(-200, -150);
    }
}
```

Razlike u odnosu na jednu od prethodnih lekcija su sledeće:

- više nema početnog i krajnjeg ugla, već se koristi samo jedan ugao – krajnji; početni ugao je uvek 0,
- nakon crtanja svakog pojedinačnog segmenta obavlja se rotiranje kompletnog `canvas` elementa za ugao upravo nacrtanog segmenta,
- rotiranje se obavlja oko centra kružnog dijagrama, te se stoga pre rotiranja obavlja postavljanje referentne tačke u centar kruga, transformacijom pomeranja,
- odmah nakon rotiranja, pomeranje se poništava, vraćanjem početne tačke koordinatnog sistema na njenu originalnu poziciju.

Na ovaj način, nema potrebe da se vodi računa o početnom i krajnjem uglu i obavlja njihovo sabiranje, kao u jednoj od prethodnih lekcija. U svakoj iteraciji petlje, koordinatni sistem se rotira za ugao upravo nacrtanog segmenta.

Efekat je identičan kao u jednoj od prethodnih lekcija (slika 8.11).



Slika 8.11. Pie Chart dijagram

Kompletan kod primera možete da preuzmete sa sledećeg linka:

`example-piechart-using-transformations.rar`

Pitanje

Transformacija pomeranja se drugačije naziva:

a) skaliranje
b) ekstraponiranje
c) translacija
d) eksproprijacija

Objašnjenje

Translacija omogućava pomeranje kompletnog koordinatnog sistema canvas elementa, odnosno postavljanje njegove referentne tačke na neku novu poziciju.

Osnove Canvas animacije

Sve što smo u prethodnim lekcijama naučili o crtanju grafike unutar canvas elementa korišćenjem Canvas API-ja, poslužiće nam da u narednim lekcijama kreiramo jednu realnu igru. Pre nego što se ozbiljno posvetimo takvom poslu, biće prikazani neki osnovni postulati Canvas animacije.

Animacija unutar canvas elementa funkcioniše po sledećem principu:

1. očisti canvas,
2. sačuvaj stanje,
3. nacrtaj kadar,
4. vrati izvorno stanje.

Animiranje grafike unutar canvas elementa obavlja se konstantnim ponavljanjem 4 upravo navedena koraka. Na početku se u potpunosti uklanja sve što je tokom prethodne iteracije nacrtano unutar canvasa. To nam omogućava da naredni kadar nacrtamo na potpuno čistom canvas elementu. Pre crtanja se obavlja čuvanje stanja kako bi se nakon obavljenog crtanja kadra mogla resetovati stilizacija i transformacije koje su korišćene tokom generisanja kadra. Zatim se obavlja crtanje kadra, a na kraju i povratak stanja canvasa u originalno stanje.

S obzirom na to da animiranje unutar canvasa podrazumeva ciklično ponavljanje upravo opisanih operacija, potpuno je razumljivo da je i ovde neophodno koristiti neku od tajming funkcija koje web pregledači nama izlažu na korišćenje.

Kako sve ovo izgleda u praksi imaćete prilike da vidite kroz nekoliko narednih primera.

Primer – animiranje kretanja kruga

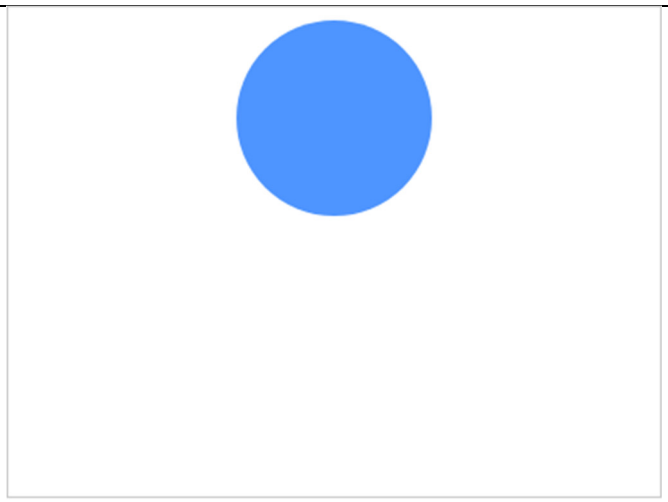
Za početak biće prikazan primer koji je već realizovan u uvodnom modulu ovoga kursa. On podrazumeva animiranje kretanja kruga gore-dole. Inicijalno je takav primer realizovan korišćenjem običnog HTML elementa, a sada ćete imati prilike da vidite kako to isto postići unutar canvasa.

Kod izgleda ovako:

```
      <!DOCTYPE html>
      <html lang="en">

      <head>
          <meta charset="UTF-8">
          <meta name="viewport" content="width=device-width, initial-
scale=1.0">
          <title>Canvas Animation Example - Circle Up-Down</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>

              window.onload = draw;

              let circleRadius = 60;
              let translateY = 0;
```

```
            let speed = 4;
            let direction = +1;

            function draw() {
                let myCanvas = document.getElementById("my-canvas");

                if (myCanvas.getContext) {

                    let ctx = myCanvas.getContext('2d');
                    update(ctx);

                } else {
                    alert("Canvas is not supported.");
                }
            }

            function update(ctx) {

                ctx.clearRect(0, 0, 400, 300);

                ctx.save();
                ctx.translate(0, translateY);

                ctx.beginPath();
                ctx.fillStyle = '#4F95FF';
                ctx.arc(200, 60, 60, 0, 2 * Math.PI);
                ctx.fill();

                ctx.restore();

                if (translateY > ctx.canvas.height - (circleRadius * 2))
{
                    direction = -1;
                } else if (translateY < 0) {
                    direction = 1;
                }

                translateY += direction * speed;

                requestId = requestAnimationFrame(function () {
                    update(ctx);
                });
            }
        </script>
    </body>

    </html>
```

Prikazanim kodom dobija se efekat kao na animaciji 8.1.



Animacija 8.1. Primer animiranja kretanja kruga gore-dole

Kompletna logika animacije sadržana je unutar funkcije `update()`. Funkcija `update()` koristi nekoliko promenljivih koje su deklarisane globalno:

- `circleRadius` – poluprečnik kruga,
- `translateY` – pomeraj po Y osi,
- `speed` – brzina animacije,
- `direction` – usmerenje animacije, odnosno da li se krug kreće nagore ili nadole; kada je vrednost ove promenljive 1, krug se kreće nadole, a kada je vrednost -1, krug se kreće nagore.

Logika za animiranje je istovetna već viđenoj u jednoj od prethodnih lekcija. U svakoj iteraciji animacije obavljaju se 4 nešto ranije opisana koraka:

- čisti se kompletan `canvas`, korišćenjem metode `clearRect()`,
- obavlja se čuvanje stanja `canvasa`, pre obavljanja transformacija,
- obavlja se transformisanje pomeranja, za vrednost `translateY` promenljive,
- crta se krug,
- obavljaju se pripreme za naredni kadar tako što se utvrđuje usmerenje i nova vrednosti pomeranja, odnosno promenljive `translateY`,
- usmerenje postaje negativno kada donji deo kruga dodirne dno `canvas` elementa; da bi se tako nešto utvrdilo, proverava se da li je pomeraj veći od visine `canvasa` od koje se oduzima visina kruga (dva poluprečnika),
- usmerenje postaje pozitivno kada vrh kruga dodirne vrh `canvas` elementa; da bi se tako nešto utvrdilo, proverava se da li je pomeraj manji od 0,
- vrednost `translateY` promenljive se svakom iteracijom uvećava za proizvod usmerenja i brzine.

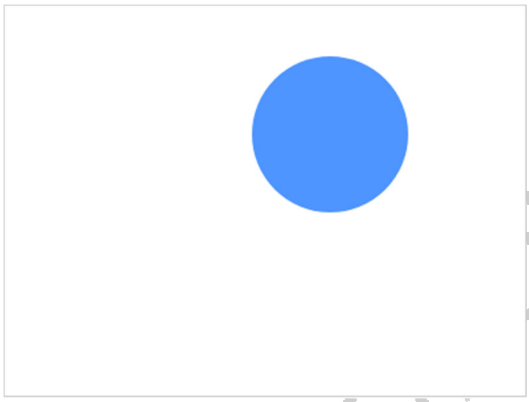
Kompletan primer možete preuzeti sa sledećeg linka:

`example-circle-up-down.rar`

Primer – slobodno kretanje kruga

Naredni primer ilustrovaće nešto naprednije kretanje kruga. Krug će se slobodno kretati u okvirima canvas elementa, pritom menjajući smer kada dođe do okvira. Animacija će se ovoga puta obavljati uticanjem na koordinate centra kruga. Drugim rečima, pomeranje kruga se neće obavljati transformacijom pomeranja, već direktnim uticanjem na centar kruga.

Primer će stvoriti efekat prikazan animacijom 8.2.



Animacija 8.2. Slobodno kretanje kruga

Evo kako će izgledati kod za realizaciju ovakvog primera:

```
      <!DOCTYPE html>
      <html lang="en">
      <head>
          <meta charset="UTF-8">
          <meta name="viewport" content="width=device-width, initial-
scale=1.0">
          <title>Canvas Example - Circle Animtaion</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>
              window.onload = draw;
              let circleX;
              let circleY;
              let circleRadius = 60;
              let speed = 2;
              let directionX = 1;
              let directionY = 1;
              function draw() {
```

```
                    let myCanvas = document.getElementById("my-canvas");
                    if (myCanvas.getContext) {
                        let ctx = myCanvas.getContext('2d');
                        circleX = Math.floor(randomNumber(circleRadius,
ctx.canvas.width - (circleRadius)));
                        circleY = Math.floor(randomNumber(circleRadius,
ctx.canvas.height - (circleRadius)));
                        update(ctx);
                    } else {
                        alert("Canvas is not supported.");
                    }
                }
                function update(ctx) {
                    ctx.clearRect(0, 0, 400, 300);
                    ctx.beginPath();
                    ctx.fillStyle = '#4F95FF';
                    ctx.arc(circleX, circleY, 60, 0, 2 * Math.PI);
                    ctx.fill();
                    if (circleX > ctx.canvas.width - circleRadius)
                        directionX = -directionX;
                    if (circleX < circleRadius)
                        directionX = -directionX;
                    if (circleY > ctx.canvas.height - circleRadius)
                        directionY = -directionY;
                    if (circleY < circleRadius)
                        directionY = -directionY;
                    circleX += directionX * speed;
                    circleY += directionY * speed;
                    requestId = requestAnimationFrame(function () {
                        update(ctx);
                    });
                }
                function randomNumber(min, max) {
                    return Math.random() * (max - min) + min;
                }
            </script>
        </body>
        </html>
```

Kao i u prethodnom primeru, logika za definisanje animacije sadržana je unutar funkcije `update()`. Ova funkcija koristi sledeće promenljive:

- `circleX` – x koordinata centra kruga,
- `circleY` – y koordinata centra kruga,
- `circleRadius` – poluprečnik kruga,
- `speed` – brzina animacije,
- `directionX` – usmerenje kretanja kruga po x osi,
- `directionY` – usmerenje kretanja kruga po y osi.

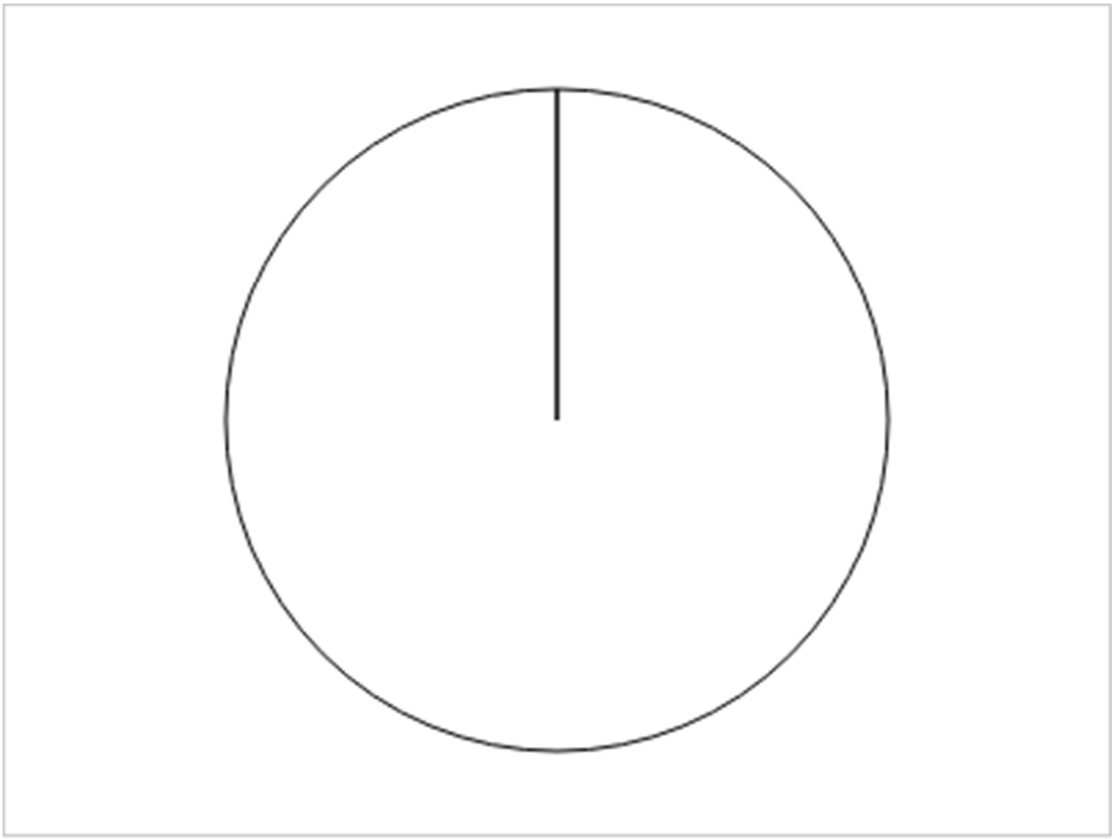
Unutar metode `draw()` pre početka animiranja obavlja se nasumično generisanje koordinata kruga. Na taj način se osigurava da će svakim novim osvežavanjem stranice, animacija započeti sa nove pozicije. Za dinamičko generisanje koordinata centra kruga koristi se pomoćna funkcija `randomNumber()`. Unutar nje je na osnovu `Math.random()` funkcije napravljena funkcionalnost koja omogućava definisanje minimalne i maksimalne vrednosti. Minimalna i maksimalna vrednost se definišu kako bi se osiguralo da početne koordinate kruga ne izađu iz okvira `canvas` elementa.

Kompletan primer možete preuzeti sa sledećeg linka:

`example-circle-free-motion.rar`

Primer – simulacija radara

Sledeći primer će ilustrovati animaciju nalik na onu koja postoji kod radarskih uređaja. Primer će biti maksimalno uprošćen, pa će se tako sastojati iz jedne kružnice i linije. Početna tačka linije će biti u centru kružnice, a završna tačka na njenom obodu. Linija će kružiti oko svoje početne tačke, odnosno oko centra kružnice, a sve to će izgledati kao na animaciji 8.3.



Animacija 8.3. Simulacija radara

Kod za realizaciju ovog 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>Canvas Example - Stopwatch 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>
              window.onload = draw;
              function draw() {
                  let myCanvas = document.getElementById("my-canvas");
                  if (myCanvas.getContext) {
                      let ctx = myCanvas.getContext('2d');
                      update(ctx);
                  } else {
                      alert("Canvas is not supported.");
                  }
              }
              let angle = 0;
              function update(ctx) {
                  ctx.save();
                  ctx.clearRect(0, 0, 400, 300);
                  ctx.beginPath();
                  ctx.arc(200, 150, 120, 0, 2 * Math.PI);
                  ctx.closePath();
                  ctx.stroke();
                  ctx.translate(200, 150);
                  ctx.rotate((Math.PI / 180) * angle);
                  ctx.translate(-200, -150);
                  ctx.beginPath();
                  ctx.moveTo(200, 150);
                  ctx.lineTo(200, 30);
                  ctx.closePath();
                  ctx.stroke();
                  ctx.restore();
                  angle++;
                  requestId = requestAnimationFrame(function () {
                      update(ctx);
                  });
              }
          </script>
      </body>
      </html>
```

Kompletna logika animacije sadržana je unutar funkcije `update()`. Unutar takve funkcije obavljaju se sledeći koraci:

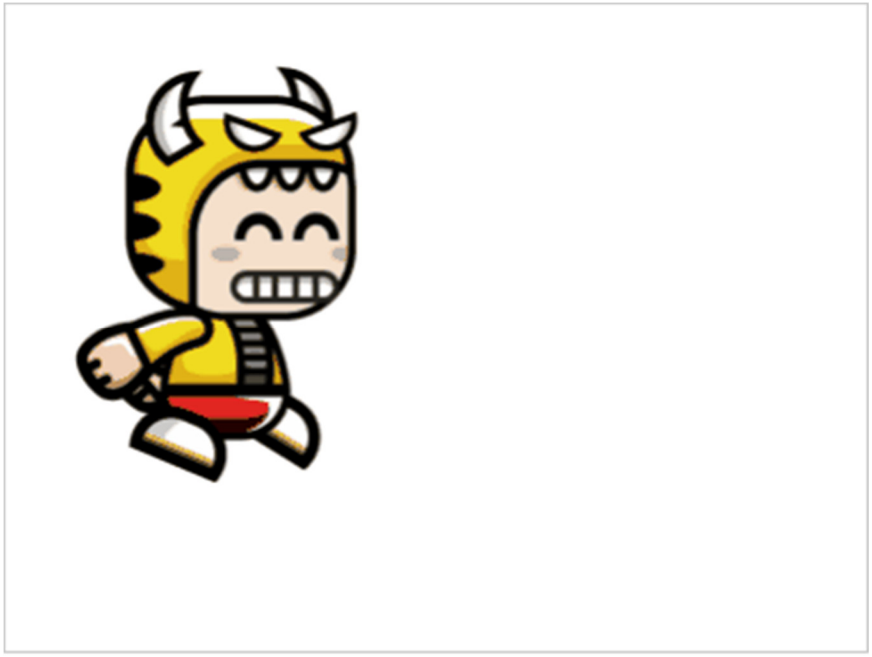
- čuva se stanje `canvas` elementa,
- čisti se kompletan `canvas` element,
- crta se kružnica,
- referentna tačka se pomera u centar kružnice, korišćenjem translacije,
- obavlja se rotiranje koordinatnog sistema korišćenjem metode `rotate()`,
- referentna tačka se vraća na svoju originalnu poziciju,
- crta se linija,
- `canvas` se vraća u originalno stanje i time se resetuje transformacija rotacije,
- ugao rotacije se uvećava za jedan,
- obavlja se ponovno pozivanje `update()` metode.

Kompletan kod primera možete da preuzmete sa sledećeg linka:

`example-radar-animation.rar`

Primer – spritesheet animacija

Kao uvod u ono što nas čeka u narednom modulu, za kraj ove lekcije biće prikazan primer spritesheet animacije. Reč je o animaciji koja podrazumeva brzo smenjivanje sprajtova grupisanih unutar jedne slike, čime se dobija animacija jednog karaktera, odnosno elementa igre. Primer će ilustrovati animiranje trčanja i skoka glavnog junaka, igre koju ćemo kreirati u narednom modulu. Finalni efekat će biti kao na animaciji 8.4.



Animacija 8.4. Primer spritesheet animacije

Kod za realizaciju 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>Canvas Example - Spritesheet 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>
              window.onload = draw;
              const NO_SPRITES = 9;
              let spriteIndex = 0;
              let speed = 0.125;
              function draw() {
                  let myCanvas = document.getElementById("my-canvas");
                  if (myCanvas.getContext) {
                      let ctx = myCanvas.getContext('2d');
                      let spritesheet = new Image();
                      spritesheet.addEventListener('load', function () {
                          update(ctx, spritesheet);
                      }, false);
                      spritesheet.src = 'runner.png';
                  } else {
                      alert("Canvas is not supported.");
                  }
              }
              function update(ctx, spritesheet) {
                  ctx.clearRect(0, 0, 400, 300);
                  ctx.drawImage(spritesheet, Math.floor(spriteIndex) *
(spritesheet.width / NO_SPRITES), 0, spritesheet.width / NO_SPRITES,
spritesheet.height, 25, 25, spritesheet.width / NO_SPRITES,
spritesheet.height);
                  if (spriteIndex < NO_SPRITES) {
                      spriteIndex = spriteIndex + speed;
                  } else {
                      spriteIndex = 2;
                  }
                  requestId = requestAnimationFrame(function () {
                      update(ctx, spritesheet);
                  });
              }
          </script>
      </body>
      </html>
```

Unutar `draw()` metode, ovoga puta, nalazi se kod za učitavanje jedne slike. Reč je o slici unutar koje se grupiše ukupno 9 sprajtova. Kada se takva slika u potpunosti učita, obavlja se pozivanje metode `update()`, čime započinje animacija.

Metoda `update()` prihvata dva parametra – kontekst za crtanje i upravo učitani spritesheet.

Animacija se postiže smenjivanjem sprajtova. Smenjivanje sprajtova se postiže korišćenjem `drawImage()` metode kojom se obavlja izolovanje pojedinačnog sprajta koji treba nacrtati. Sprajtovi se prvo prikazuju od početka do kraja, a zatim se nakon prikazivanja poslednjeg sprajta, animacija nastavlja od 3. sprajta (čiji je indeks 2), zato što se prva dva sprajta koriste za prikaz karaktera u mirovanju.

Kako se animacije ne bi izvršavala suviše brzo, u primeru postoji i promenljiva `speed` kojom se kontroliše brzina promene sprajtova. Naime, promena sprajta u svakom kadru proizvodi isuviše brzu animaciju, stoga se na indeks trenutnog sprajta (`spriteIndex`) u svakoj iteraciji dodaje vrednost definisana svojstvom `speed`. S obzirom na to da je za odabir sprajta neophodna celobrojna vrednost, koristi se `Math.floor()` funkcija koja uvek obavlja zaokruživanje indeksa na manji ceo broj. Stoga će jedan sprajt animacije biti aktivan sve dok indeks ne postane veći od trenutnog, a time se na kraju postiže da jedan sprajt bude prikazan kroz nekoliko kadrova animacije. Sve to na kraju rezultuje nešto sporijom animacijom, čiju je brzinu moguće kontrolisati vrednošću promenljive `speed`.

Kompletan kod primera možete preuzeti sa sledećeg linka:

`example-spritesheet-animation.rar`

Rezime

- Stanje `canvasa` se odnosi na sve vizuelne osobine koje se definišu korišćenjem različitih svojstava za stilizovanje.
- Canvas API poseduje dve metode koje omogućavaju rukovanje stanjem: `save()` i `restore()`.
- Metoda `save()` čuva stanje `canvasa`.
- Metoda `restore()` vraća `canvas` u prethodno stanje.
- Stanje `canvas` elementa je u pozadini realizovano kao stek struktura podataka.
- Canvas API omogućava obavljanje nekoliko transformacija: translaciju, rotaciju i promenu veličine.
- Translacija omogućava pomeranje kompletnog koordinatnog sistema `canvas` elementa, odnosno postavljanje njegove referentne tačke na neku novu poziciju.
- Translacija se obavlja korišćenjem metode `translate(x, y)`.
- Rotacija je transformacija koja omogućava rotiranje kompletnog koordinatnog sistema.
- Rotacija se postiže korišćenjem metode `rotate(angle)`.
- Promena veličine koordinatnog sistema obavlja se upotrebom metode `scale(x, y)`.
- Canvas animacija podrazumeva kontinuirano, ponovno crtanje, ažurirane grafike unutar `canvas` elementa, čime se dobija privid kretanja.