Stilizovanje geometrijskih oblika

U prethodnoj lekciji bavili smo se pristupima za crtanje raznih geometrijskih oblika unutar canvas elementa. U ovoj lekciji imaćete prilike da naučite kako se obavlja stilizacija nacrtane grafike, što podrazumeva definisanje boje, stila okvira, gradijenata, senki i slično.

Rad sa bojama

Boje je moguće definisati korišćenjem dva svojstva, objekta koji predstavlja kontekst za crtanje:

- **fillStyle** - za definisanje boje ispune
- **strokeStyle** - za definisanje boje okvira

Ova svojstva kao svoju vrednost mogu da prihvate `string` podatak koji se odnosi na boju, koja može biti definisana u nekoliko različitih oblika:

- imenom - `'red'`
- heksadecimalnom vrednošću - `#4F95FF`
- rgb() funkcijom - `'rgb(79, 149, 255)'`
- rgba() funkcijom - `'rgb(79, 149, 255, 0.5)'`

Primer definisanja boje pozadine jednog kruga može da izgleda ovako:

```
ctx.beginPath();
ctx.fillStyle = '#4F95FF';
ctx.arc(200, 150, 120, 0, 2 * Math.PI);
ctx.fill();
```

Na ovaj način dobija se prikaz kao na slici 6.1.

Slika 6.1 - Krug sa definisanom bojom pozadine (ispune)

Napomena

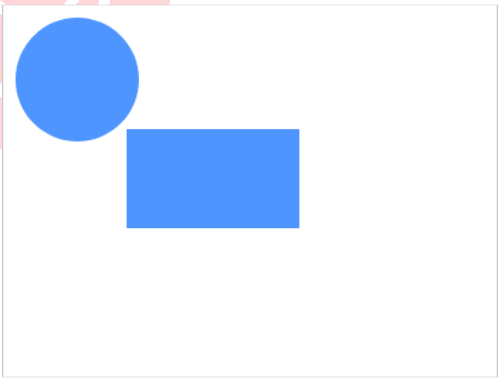
I dalje koristimo šablon koji smo uspostavili na početku upoznavanja sa Canvas API-em. Stoga je prikazani kod potrebno pisati unutar `draw()` metode koja se aktivira kada je kompletan HTML dokument učitan, nakon naredbe za dobijanje konteksta za crtanje:

```
<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) {
            var ctx = myCanvas.getContext('2d');
            ctx.beginPath();
            ctx.fillStyle = '#4F95FF';
            ctx.arc(200, 150, 120, 0, 2 * Math.PI);
            ctx.fill();
        } else {
            alert("Canvas is not supported.");
        }
    }
</script>
```

Bitno je da znate da kada jednom postavite boju pozadine korišćenjem `fillStyle` svojstva, takva boja se upotrebljava za boju pozadine svih oblika koji će biti crtani u nastavku:

```
ctx.beginPath();
ctx.fillStyle = '#4F95FF';
ctx.arc(60, 60, 50, 0, 2 * Math.PI);
ctx.rect(100, 100, 140, 80)
ctx.fill();
```

Kao što možete da vidite, boja ispune je definisana jednom, a nakon toga, svi nacrtani oblici imaju takvu boju ispune (slika 6.2).



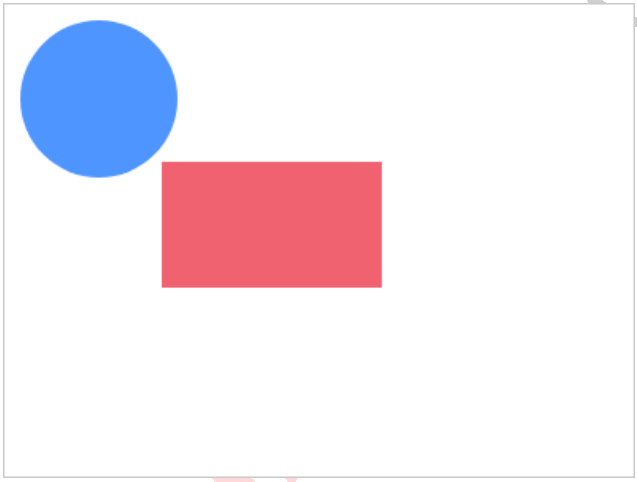
Slika 6.2 - Jednom definisana boja pozadine primenjuje se nad svim oblicima koji se crtaju u nastavku

Ukoliko želite da neki naredi oblik ima drugačiju boju pozadine, neophodno je da redefinišete vrednost svojstva `fillStyle`, neposredno pre crtanja takvog oblika:

```
ctx.beginPath();
ctx.fillStyle = '#4F95FF';
ctx.arc(60, 60, 50, 0, 2 * Math.PI);
ctx.fill();

ctx.beginPath();
ctx.fillStyle = '#F06270';
ctx.rect(100, 100, 140, 80)
ctx.fill();
```

Obratite pažnju da je sada oblike neophodno crtati kao zasebne putanje, kao bi bilo moguće definisati različite boje pozadine. Na ovaj način se dobija prikaz kao na slici 6.3.



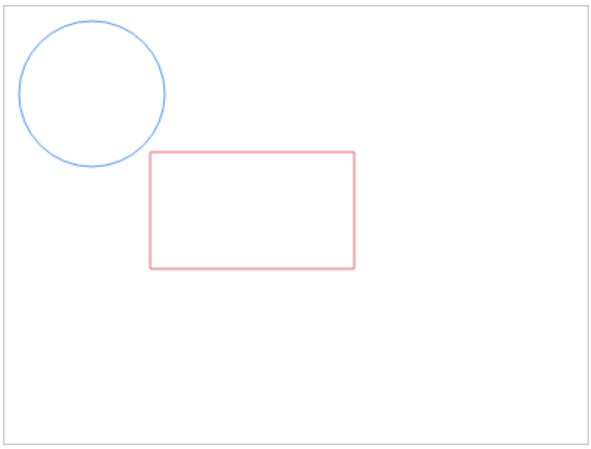
Slika 6.3 - Dva oblika sa različitim bojama ispune

Za definisanje boje okvira koristi se svojstvo `strokeStyle`, po istom principu kao i upravo ilustrovano svojstvo `fillStyle`:

```
ctx.beginPath();
ctx.strokeStyle = '#4F95FF';
ctx.arc(60, 60, 50, 0, 2 * Math.PI);
ctx.stroke();

ctx.beginPath();
ctx.strokeStyle = '#F06270';
ctx.rect(100, 100, 140, 80)
ctx.stroke();
```

Nacrtani oblici će izgledati kao na slici 6.4.



Slika 6.4 - Oblici sa definisanim bojama okvira

Primer - Stilizovanje Pie Chart dijagrama (1. deo)

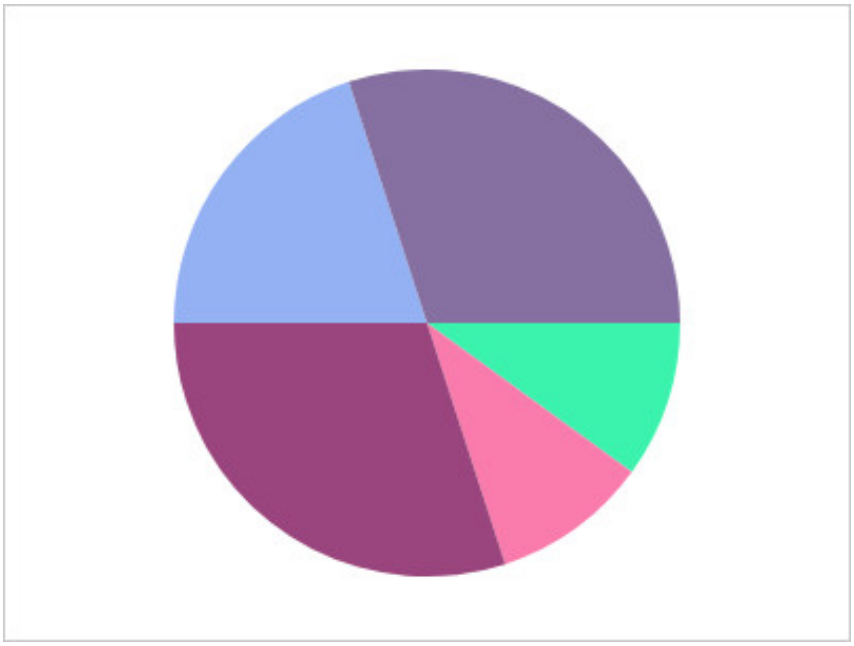
Upravo prikazani pristupi za definisanje boje ispune grafike koja se crta unutar canvas elementa, biće iskorišćeni za stilizovanje *Pie Chart* dijagrama koji je kreiran u jednoj od prethodnih lekcija. U postojeći kod će biti dodata logika za dinamičko generisanje boja kojima će biti obojeni segmenti kružnog dijagrama koji se crtaju:

```
window.onload = draw;
function draw() {
    let myCanvas = document.getElementById("my-canvas");
    if (myCanvas.getContext) {
        var ctx = myCanvas.getContext('2d');
        makePieChart(ctx, [10, 10, 30, 20, 30]);
    } else {
        alert("Canvas is not supported.");
    }
}
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.");
    let startAngle = 0;
    for (let i = 0; i < shares.length; i++) {
        let endAngle = startAngle + (Math.PI / 180) * shares[i]
* 3.6;

        let red = Math.floor(Math.random() * 256);
        let green = Math.floor(Math.random() * 256);
        let blue = Math.floor(Math.random() * 256);
        ctx.fillStyle = 'rgb(' + red + ', ' + green + ', ' +
blue + ')';

        ctx.beginPath();
        ctx.arc(200, 150, 120, startAngle, endAngle);
        ctx.lineTo(200, 150);
        ctx.fill();
        startAngle = endAngle;
    }
}
```

Na ovaj način može se dobiti efekat kao na slici 6.5 (*boje će se kod vas verovatno razlikovati, s obzirom da se dinamički generišu, prilikom svakog novog izvršavanja koda*).



Slika 6.5 - Pie Chart dijagram

Pitanje

Koje svojstvo se koristi za definisanje boje ispune elemenata koji se crtaju?

a) fillStyle
b) beginPath
c) colorStyle
d) fillColor

Objašnjenje

Svojstvo fillStyle koristi se za definisanje boje ispune.

Providnost

Providnost je moguće definisati korišćenjem dva pristupa:

- `globalAlpha` - svojstvo kojim je moguće definisati providnost koja će da bude primenjena nad svim bojama koje budu korišćene u nastavku, bilo za definisanje boje ispune ili okvira
- `rgba()` - format za definisanje boje, koji omogućava da se boja definiše korišćenjem četiri kanala - *red*, *green*, *blue*, *alpha*; stoga poslednji parametar ove funkcije omogućava definisanje providnosti

Primer definisanja providnosti korišćenjem `globalAlpha` svojstva, može da izgleda ovako:

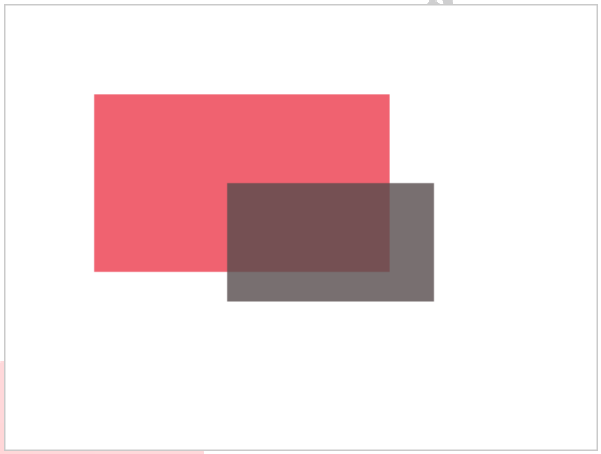
```
ctx.beginPath();
ctx.fillStyle = '#F06270';
ctx.rect(60, 60, 200, 120)
ctx.fill();

ctx.globalAlpha = 0.8;

ctx.beginPath();
ctx.fillStyle = '#574C4D';
ctx.rect(150, 120, 140, 80)
ctx.fill();
```

Prikazanim kodom se crtaju dva pravougaonika. Nakon crtanja prvog pravougaonika, korišćenjem svojstva `globalAlpha` definiše se providnost. Vrednost mora biti između 0 i 1, gde 0 označava potpunu providnost, a 1 potpuno odsustvo providnosti.

Prikazanim kodom se dobija prikaz kao na slici 6.6.



Slika 6.6 - Primer providnosti definisane nad pozadinom drugog pravougaonika

Drugi način za definisanje providnosti, jeste definisanje boje u rgba() obliku. Stoga se identičan efekat može postići na sledeći način:

```
ctx.beginPath();
ctx.fillStyle = '#F06270';
ctx.rect(60, 60, 200, 120)
ctx.fill();

ctx.beginPath();
ctx.fillStyle = 'rgba(87, 76, 77, 0.8)';
ctx.rect(150, 120, 140, 80)
ctx.fill();
```

Stilizovanje okvira

Stilizovanje linija, odnosno okvira, može se obaviti korišćenjem sledećih svojstava i metoda objekta koji predstavlja kontekst:

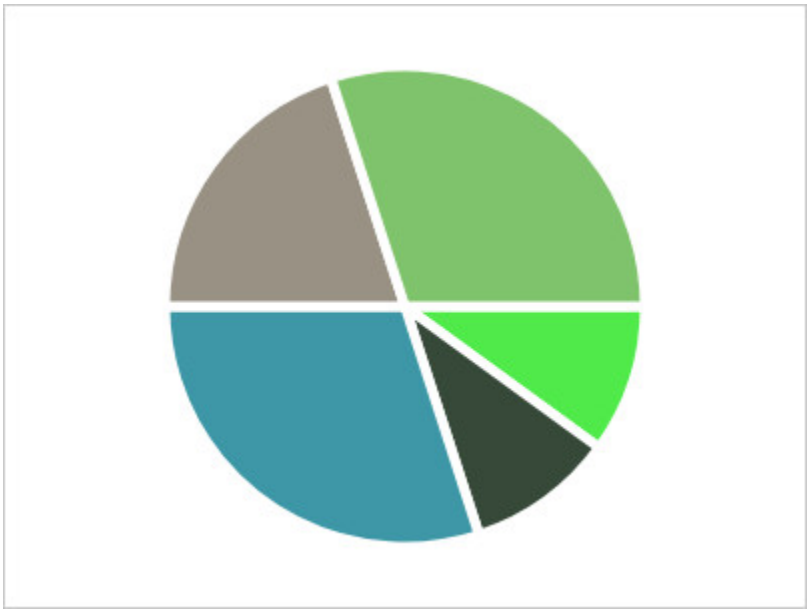
- **lineWidth** - širina linije; definiše se korišćenjem celih, pozitivnih numeričkih vrednosti
- **lineCap** - stil krajeva linija; definiše se korišćenjem `string` vrednosti `'butt'`, `'round'` ili `'square'`:
- **lineJoin** - stil sastava linija; može imati vrednosti `'round'`, `'bevel'` ili `'miter'`
- **miterLimit** - definiše rastojanje između spoljne i unutrašnje tačke sastava linija, kada se za stil sastava koristi opcija `'miter'`
- **setLineDash()/getLineDash()** - metode za rad sa isprekidanim linijama
- **lineDashOffset** - definiše gde će započeti isprekidana linija

Primer - Stilizovanje PieChart dijagrama (2. deo)

Upravo prikazani pristupi za stilizovanje linija, biće iskorišćeni za dodatno unapređivanje *Pie Chart* dijagrama. Na svaki od segmenata, sada ćemo postaviti i okvire i na taj način definisati rastojanje između susednih segmenata. Ipak, bitno je da znate da jedna putanja unutar `canvas`-a, ne može u istom trenutku imati i ispunu i okvire. Jednostavno, putanja se može nacrtati ili `fill()` ili `stroke()` metodom, što nama ostavlja mogućnost da dobijemo ili samo ispunu, ili samo okvire bez ispune. Stoga ćemo okvire oko postojećih segmenata da dodamo kreiranjem dodatnih putanja koje će biti nacrtane metodom `stroke()`:

```
            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.");
                let startAngle = 0;
                for (let i = 0; i < shares.length; i++) {
                    let endAngle = startAngle + (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, startAngle, endAngle);
                    ctx.lineTo(200, 150);
                    ctx.fill();
                    ctx.strokeStyle = "white";
                    ctx.lineWidth = 5;
                    ctx.beginPath();
                    ctx.arc(200, 150, 120, startAngle, endAngle);
                    ctx.lineTo(200, 150);
                    ctx.closePath();
                    ctx.stroke();
                    startAngle = endAngle;
                }
            }
```

Na ovaj način *Pie Chart* dijagram dobija izgleda kao na slici 6.7.



Slika 6.7 - PieChart dijagram sa okvirima oko pojedinačnih segmenata

Kompletan kod primera, možete da preuzmete sa sledećeg linka:

`example-pie-chart-styling.rar`

Gradijenti

Canvas API omogućava definisanje gradijenata koji će biti iskorišćeni kao boja okvira ili ispune. Omogućeno je kreiranje dve vrste gradijenata, pa samim tim postoje i dve različite metode:

- `createLinearGradient(x1, y1, x2, y2)` - za kreiranje linearnih gradijenata
- `createRadialGradient(x1, y1, r1, x2, y2, r2)` - za kreiranje radijalnih gradijenata

Obe upravo prikazane metode koriste se za definisanje početka i završetka gradijenta. Kod lineranih gradijenata to su dve tačke, početna i završna, dok je kod radijalnih gradijenata reč o dva kruga, takođe početnom i završnom.

Kreiranje gradijenata jeste posao koji se sastoji iz nekoliko koraka. Prvi korak jeste kreiranje objekta gradijenta i za obavljanje takvog posla koriste se upravo prikazane metode. S obzirom da se dve prikazane metode koriste za definisanje početka i završetka gradijenta, boje od kojih će gradijent biti sačinjen dodaju se korišćenjem posebne metode - `addColorStop(position, color)`. Na kraju, tako kreirani objekat gradijenta postavlja se za boju ispune ili okvira, kao vrednost svojstava `fillStyle` ili `fillStroke`.

Kako sve ovo izgleda u praksi, biće prikazano u narednim redovima. Prvo će biti ilustrovano kreiranje linearnih gradijenata. Prvi korak jeste kreiranje objekta gradijenta:

```
let linearGradient = ctx.createLinearGradient(10, 10, 140, 140);
```

Kreiranje objekta linearnog gradijenta podrazumeva korišćenje funkcije `createLinearGradient()`. Njoj se prosleđuju koordinate početne i krajnje tačke. Bitno je znati da su koordinate relativne koordinatnom sistemu kompletnog `canvas` elementa, a ne nekog konkretnog oblika nad kojim će se gradijent primeniti.

Drugi korak u kreiranju linearnog gradijenta jeste dodavanje boje i pozicija na kojima će se one naći unutar gradijenta:

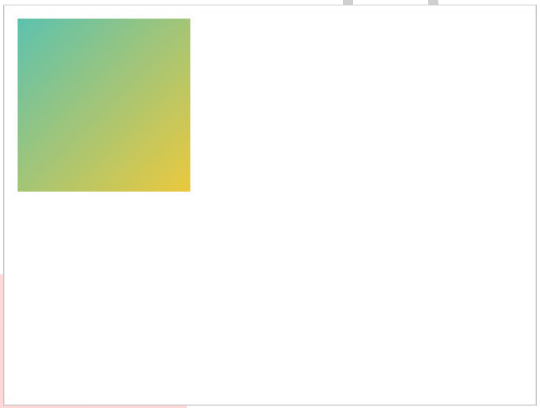
```
linearGradient.addColorStop(0, '#5DC2B0');
linearGradient.addColorStop(1, '#ECC93E');
```

Na ovaj način su definisane dve boje koje će ravnomerno biti rasprostranjene duž ravni gradijenta.

Treći i poslednji korak, jeste postavljanje ovako kreiranog gradijenta za boju ispune ili okvira:

```
ctx.fillStyle = linearGradient;
ctx.fillRect(10, 10, 130, 130);
```

Na ovaj način se unutar `canvas` elementa dobija prikaz kao na slici 6.8.



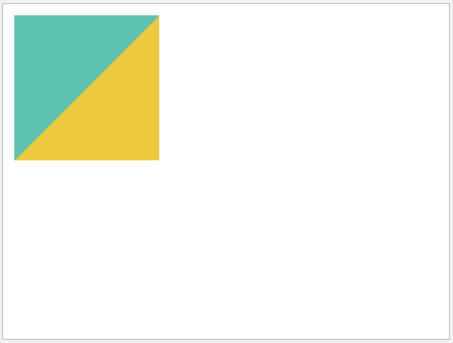
Slika 6.8 - Primer lineranog gradijenta unutar canvas-a

Primer - Linearni gradijent sa oštrim prelaskom između boja

Oštar prelazak između boja gradijenta se može postići ponavljanjem susednih boja na istoj poziciji unutar gradijenta:

```
let ctx = myCanvas.getContext('2d');
let linearGradient = ctx.createLinearGradient(10, 10, 140, 140);
linearGradient.addColorStop(0, '#F06270');
linearGradient.addColorStop(0.5, '#F06270');
linearGradient.addColorStop(0.5, '#ABD486');
linearGradient.addColorStop(1, '#ABD486');
ctx.fillStyle = linearGradient;
ctx.fillRect(10, 10, 130, 130);
```

Na ovaj način se dobija prikaz kao na slici 6.9.



Slika 6.9 - Oštar prelaz između boja gradijenta

Koraci za kreiranje radijalnog gradijenta su slični. Prvo se kreira objekat radijalnog gradijenta:

```
let radialGradient = ctx.createRadialGradient(75, 75, 30, 75, 75, 60);
```

Ovoga puta se koristi metoda `createRadialGradient()` koja prihvata 6 parametara. Prva tri parametra se odnose na podatke prvog kruga - koordinate njegovog centra i poluprečnik, respektivno. Druga tri parametra se odnose na osobine drugog kruga. Opet važe identična pravila kao i kod lineranih gradijenata. Koordinate se odnose na kompletan `canvas` element.

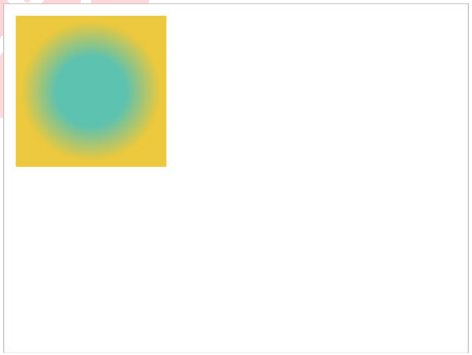
Nakon kreiranja objekta gradijenta, njemu je potrebno dodati boje i njihove pozicije:

```
radialGradient.addColorStop(0, '#5DC2B0');
radialGradient.addColorStop(1, '#ECC93E');
```

Na kraju je kreirani gradijent potrebno dodeliti `fillStyle` ili `strokeStyle` svojstvu:

```
ctx.fillStyle = radialGradient;
ctx.fillRect(10, 10, 130, 130);
```

Efekat primera je kao na slici 6.10.



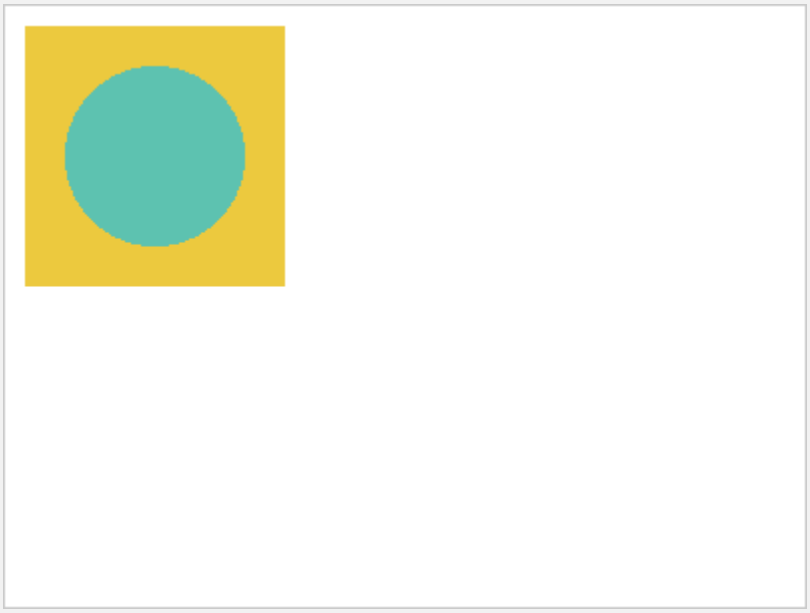
Slika 6.10 - Primer radijalnog gradijenta

Primer - Radijalni gradijent sa oštrim prelazom između boja

Po identičnom principu po kome je nešto ranije postignut oštar prelaz boja unutar linearnog gradijenta, može se obaviti isto i kada je u pitanju radijalni gradijent:

```
          var radialGradient = ctx.createRadialGradient(75, 75, 30, 75, 75,
60);
          radialGradient.addColorStop(0, '#5DC2B0');
          radialGradient.addColorStop(0.5, '#5DC2B0');
          radialGradient.addColorStop(0.5, '#ECC93E');
          radialGradient.addColorStop(1, '#ECC93E');
          ctx.fillStyle = radialGradient;
          ctx.fillRect(10, 10, 130, 130);
```

Efekat je kao na slici 6.11.



Slika 6.11 - Radijalni gradijent sa oštrim prelazom između boja

Senke

Na elemente koji se crtaju unutar `canvas`-a, moguće je postavljati senku. Senka se definiše korišćenjem sledećih svojstava:

- `shadowOffsetX` - dužina senke po x osi, odnosno koliko senka treba da izađe iz okvira elementa nad kojim se postavlja; podrazumevana vrednost je 0
- `shadowOffsetY`- dužina senke po y osi; podrazumevana vrednost je 0
- `shadowBlur` - jačina efekta zamućenja senke; podrazumevana vrednost je 0
- `shadowColor` - boja senke; podrazumevano je senka potpuno providna

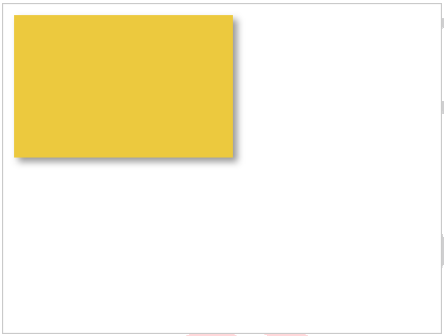
Primer definisanja senke na jednom pravougaoniku može da izgleda ovako:

```
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, 130);
```

U primeru je prvo postavljena boja ispune koju će imati elementi koji se crtaju nakon takve naredbe. Zatim slede naredbe u kojima se postavljaju vrednosti svojstava za definisanje osobina senke. Na kraju se obavlja crtanje jednog pravougaonika, koji unutar canvas-a izgleda kao na slici 6.12.



Slika 6.12 - Senka na elementu unutar canvas-a

Rezime

- svojstvo `fillStyle` koristi se za definisanje boje ispune
- svojstvo `strokeStyle` koristi se za definisanje boje okvira
- prilikom definisanja boje grafike koja se crta unutar canvas-a, boje je moguće definisati korišćenjem imena, heksadecimalnih vrednosti, ili upotrebom rgb() i rgba() funkcija
- jednom definisana boja pozadine ili okvira, odnosi se na sve elemente koji se crtaju u nastavku
- `globalAlpha` je svojstvo kojim je moguće definisati providnost koja će da bude primenjena nad svim bojama koje budu korišćene u nastavku
- definisanje providnosti je moguće obaviti i korišćenjem rgba() funkcije
- stilizovanje linija, odnosno okvira, može se obaviti korišćenjem različitih svojstava i metoda: `lineWidth`, `lineCap`, `lineJoin`, `miterLimit`, `setLineDash()`/`getLineDash()`, `lineDashOffset`
- metoda `createLinearGradient()` koristi se za kreiranje linearnih gradijenata
- metoda `createRadialGradient()` koristi se za kreiranje radijalnih gradijenata
- gradijentima se boje dodaju korišćenjem metode `addColorStop(position, color)`
- osobine senke moguće je definisati svojstvima `shadowOffsetX`, `shadowOffsetY`, `shadowBlur` i `shadowColor`