2018/2019

RAČUNALNA GRAFIKA

kORIŠTENJE ANIMACIJA U WEB TEHNOLOGIJAMA

ivo kovačević

SADRŽAJ

[1 UVOD 1](#_Toc532641541)

[2 CSS3 2](#_Toc532641542)

[2.1 Što je točno CSS3? 2](#_Toc532641543)

[2.2 Upoznavanje s CSS3 animacijama 2](#_Toc532641544)

[2.2.1 Transformacije 2](#_Toc532641545)

[2.2.2 Tranzicije 3](#_Toc532641546)

[2.2.3 Animacije 3](#_Toc532641547)

[3 MEMORY IGRA 4](#_Toc532641548)

[4 ZAKLJUČAK 5](#_Toc532641549)

[5 DODATAK A 6](#_Toc532641550)

# UVOD

Cilj ovog seminarskog rada je pokazati snagu CSS3 stilskog jezika.

U prošlosti, Flash tehnologija dominirala je webom. Preko 30% stranica koristilo je ovaj alat za razne animacije, video sadržaje, igre itd. Napredovanjem CSS stilskog jezika, broj stranica koje koriste Flash pao je ispod 4% u samo par godina. Razni Internet preglednici više ne podržavaju Flash te je njegovo korištenje strogo nepreporučljivo.

CSS stilski jezik je toliko napredovao da omogućava stvaranje svakakvih animacija (rotacija, translacija, skaliranja…) gdje je jedina granica mašta kreatora.

U ovom seminarskom radu pokazat ćemo kako korištenjem CSS3 stilskog jezika možemo vrlo jednostavno kreirati animacije.

# CSS3

## Što je točno CSS3?

CSS (engl. Cascading Style Sheets) je stilski jezik za opis prezentacije dokumenta napisanog pomoću markup jezika.

Razlika između CSS i CSS3 je u tome što je CSS3 naprednija verzija CSS-a koja donosi velike promjene u svijetu web animacija i izgleda.

CSS3 proširuje CSS zakrivljenim kutovima elemenata, sjenama, gradijentima, tranzicijama i animacijama. Također dodaje Flex i Grid rasporede elemenata.

Kreiranje navedenih pojmova u prošlosti bilo je nezamislivo bez korištenja Flash tehnologije, a danas je sve to moguće uz CSS3 bez potrebe za instaliranjem dodatnih proširenja.

## Upoznavanje s CSS3 animacijama

U ovom potpoglavlju upoznat ćemo se s osnovnim CSS3 animacijama.

Pretpostavka je da postoji temeljno znanje o CSS tehnologiji pa ćemo odmah prijeći na stvar.

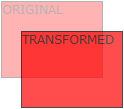
### Transformacije

CSS transformacije omogućavaju nam da translatiramo, rotiramo, skaliramo i iskrivljavamo elemente.

Funkcije koje možemo koristiti su:

* translate()
* rotate()
* scale()
* skewX()
* skewY()

**Translacija** nam omogućava da translatiramo element po x i y osi.



Slika . Transformacija elementa

Primjer:

Neka imamo element klase element1 kojeg treba pomaknuti u desno 50px i dolje 100px:

.element1 { transform: translate(50px, 100px); }

**Rotacija** nam omogućava rotiranje elemenata u smjeru kazaljke na satu ili u kontra smjeru korištenjem funkcije rotate().

Primjer:



Slika . Rotacija elementa

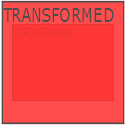
Neka imamo element klase element2 kojeg treba rotirati za 30 stupnjeva:

.element2 { transform: rotate(20deg); }

**Skaliranje** nam omogućava skaliranje elemenata korištenjem funkcije scale().

Primjer:

Neka imamo element klase element3 kojeg želimo uduplati i po dužini i po širini:



Slika . Primjer skaliranja

.element3 { transform: scale(2, 2); }

### Tranzicije

Tranzicije nam omogućavaju da promijenimo određene atribute „glatko“ (engl. smoothly) iz jedne vrijednosti u drugu.

Primjer: Želimo promijeniti boju i dimenzije botuna kada se preko njega pređe mišem.

 (hover) 🡪 

.btn {

Color: green;

Width: 50px;

Height: 30px;

transition: all linear 2s; }

.btn:hover {

Color: blue;

Width: 100px;

Height: 70px; }

„All“ označava da tranziciju želimo primijeniti na sve atribute, „linear“ označava da će promjena biti linearna tj. jednolika od početka do kraja, a „2s“ označava da će promjena trajati 2 sekunde.

### Animacije

CSS3 animacije omogućuju animaciju HTML elemenata bez korištenja Javascript-a ili Flash-a. Animacije nam omogućavaju mijenjanje koliko god hoćemo atributa i koliko god puta hoćemo. Animacije se deklariraju ključnom riječi @keyframes animation\_name. Da bi animacija proradila moramo je vezati za neki element.

Sintaksa i primjer:

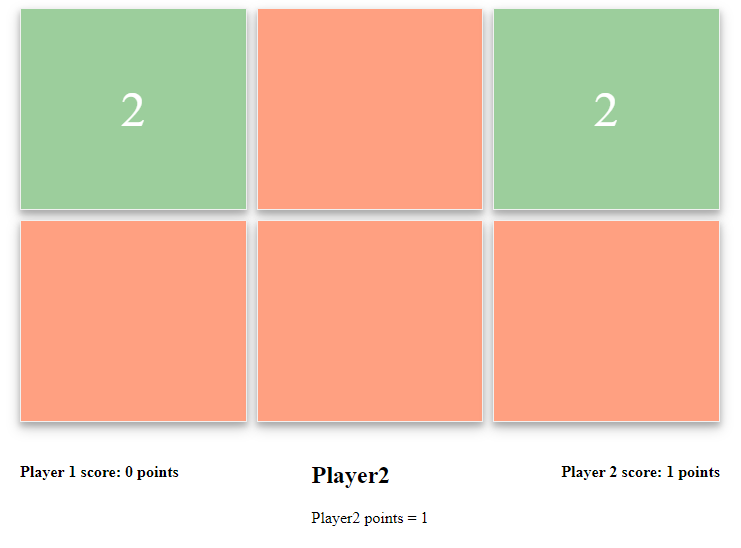
@keyframes colorChange {  
  from {background-color: red;}  
  to {background-color: yellow;}  
}

/\* The element to apply the animation to \*/  
div {  
  width: 100px;  
  height: 100px;  
  background-color: red;  
  animation-name: colorChange;  
  animation-duration: 4s;  
 animation-delay: 1s;  
 animation-iteration-count: 2;  
}

Prvo smo deklarirali animaciju „colorChange“ ključnom riječi @keyframes te smo specificirali početne i krajnje atribute s „from“ i „to“. Potom smo animaciju vezali na element div s „animation-name“. Duration, delay i count govore da će animacija trajati 4 sekunde, da će kasnit 1 sekundu i da će se ponoviti 2 puta.

# MEMORY IGRA

Na konkretnom primjeru memory igre pokazat ćemo kako stvarne primjere primjene animacija. Memory igra isprogramirana je korištenjem Javascript-a, HTML-a i CSS-a.



Slika . Memory igra

Kod memory igre bit će dostupan u DODATAK A. U slijedećim potpoglavljima biti će objašnjene sve bitnije stavke memory igre.

## Javascript

Javascript u memory-u se koristi za funkcionalnosti same igre. Postoje dvije klase Player (igrač) i Game (igra) iz kojih instanciramo objekte. Igra ima dva igrača Player1 i Player2.

Klasa Player:

class Player {

constructor(name) {

this[name\_secret] = name;

this[points\_secret] = 0;

}

getName() {

return this[name\_secret];

}

getPoints() {

return this[points\_secret];

}

incrementPoints() {

this[points\_secret] += 1;

}

}

Klasa Player vrlo je jednostavna klasa koja ima atribute name (ime igrača) i points ( bodovi igrača) te metode getName(), getPoints() i incrementPoints() koje vraćaju ime igrača, bodove i inkrementiraju trenutni broj bodova.

Ukoliko promotrite klasu Player jedino što se na prvi pogled čini zbunjujuće je kod konstruktora gdje imamo this[name\_secret] = name i this[points\_secret] = 0. Naime, u javascriptu još ne postoje privatni i zaštićeni (engl protected) tipovi atributa klase. Iz tog razloga, da bi se zaštitilo neovlašteno mijenjanje imena i broja bodova igrača te atribute klase moralo se zaštititi uz pomoć ovog malog trika.

Slijedeća na redu je klasa Game koja je nešto složenija te sadrži svu glavnu logiku ove igre.

class Game {

constructor(players) {

this.card\_content = [1, 1, 2, 2, 3, 3];

this.cards = document.getElementsByClassName('flip-card');

this.players = players;

this.currentPlayer = players[0];

this.tryNo = 0;

this.openedCards = [];

this.discoveredCards = [];

this.shuffle();

this.initGame();

}

initGame() {

currentPlayerPrint.innerHTML = this.currentPlayer.getName();

Array.from(this.cards).forEach((card, index) => {

let p = card.getElementsByClassName('card-text')[0];

p.innerHTML = this.card\_content[index];

setTimeout(() => {

card.addEventListener('click', () => {

this.onCardClick(index, card);

})

}, 0);

});

}

shuffle() {

let a = this.card\_content;

for (let i = a.length - 1; i > 0; i--) {

const j = Math.floor(Math.random() \* (i + 1));

[a[i], a[j]] = [a[j], a[i]];

}

return a;

}

onCardClick(cardIndex, card) {

if(this.tryNo < 2) {

if(this.discoveredCards.includes(cardIndex)) {

return;

} else {

if(this.openedCards.includes(cardIndex)) {

return;

} else {

this.tryNo += 1;

this.openedCards.push(cardIndex);

let flip\_card\_inner = card.getElementsByClassName('flip-card-inner');

flip\_card\_inner[0].classList.toggle('rotate-card');

if(this.tryNo == 2) {

this.tryNo = 0;

if(this.isDiscovered()) {

this.discoveredCards.push(this.openedCards[0]);

this.discoveredCards.push(this.openedCards[1]);

this.currentPlayer.incrementPoints();

para.innerHTML = "SCORE!!!";

gameCommentDiv.appendChild(para);

para.innerHTML = this.currentPlayer.getName() + " points = " + this.currentPlayer.getPoints()

gameCommentDiv.appendChild(para);

if(this.currentPlayer == this.players[0]) {

let scoreSpan = document.getElementById('p1-points');

scoreSpan.innerHTML = this.currentPlayer.getPoints();

} else {

let scoreSpan = document.getElementById('p2-points');

scoreSpan.innerHTML = this.currentPlayer.getPoints();

}

this.openedCards = [];

if(this.discoveredCards.length == 6) {

para.innerHTML = "GAME OVER!!!"

gameCommentDiv.appendChild(para);

this.showWinner();

}

} else {

setTimeout(() => {

this.changePlayer();

for(let no of this.openedCards) {

this.cards[no].getElementsByClassName('flip-card-inner')[0].classList.toggle('rotate-card');

}

this.openedCards = [];

}, 1000);

}

}

}

}

}

}

changePlayer() {

if(this.currentPlayer == this.players[0]) {

this.currentPlayer = this.players[1];

} else {

this.currentPlayer = this.players[0];

}

currentPlayerPrint.innerHTML = this.currentPlayer.getName();

}

isDiscovered() {

if(this.openedCards.length == 2) {

if(this.cards[this.openedCards[0]].getElementsByClassName('card-text')[0].innerHTML == this.cards[this.openedCards[1]].getElementsByClassName('card-text')[0].innerHTML) {

return true;

} else {

return false;

}

} else {

return false;

}

}

showWinner() {

if(this.players[0].getPoints() > this.players[1].getPoints()) {

para.innerHTML = this.players[0].getName() + " wins with " + this.players[0].getPoints() + " points";

gameCommentDiv.appendChild(para);

} else {

para.innerHTML = this.players[1].getName() + " wins with " + this.players[1].getPoints() + " points";

gameCommentDiv.appendChild(para);

}

}

}

Samo ćemo objasniti bitne stavke ove klase.

U konstruktoru klase osim inicijalizacije nekih atributa pozivamo i dvije metode shuffle() i initGame(). Shuffle kao što i sama riječ kaže zamiješa karte slučajnim redoslijedom.

InitGame() postavi „click“ eventListener na sve kartice igre. Dakle kada igrač klikne na neku karticu pozvat će se funkcija koja je zadužena za taj click događaj.

Ta funkcija je onCardClick() koja je ujedno i najsloženija metoda ove klase. U ovoj metodi prvo provjeravamo je li kliknuta karta već otkrivena ( nađen joj je par) ili je već otvorena (karta je prva kliknuta za dani potez) te ukoliko je neki od ovih uvjeta ispunjen izlazimo iz metode. Ukoliko uvjeti nisu ispunjeni povećavamo broj pokušaja trenutnog igrača te rotiramo kartu. Rotacija karte obavlja se specijalnim CSS3 efektima što će biti naknadno objašnjeno.

Nadalje, ukoliko korisnik drugim potezom otkrije karticu s istim brojem kao i kod prvog poteza, broj bodova mu se povećava te i dalje ostaje na potezu. U protivnom, pričeka se neko vrijeme (setTimout() funkcija) te se karte okreću u prvobitni položaj.

Metode changePlayer() i showWinner() rade točno ono što i samo ime sugerira tj. obavljaju promjenu igrača i prikazuju pobjednika.

Metoda isDiscover vraća istinu ili laž ovisno o tome je li korisnik pogodio dvije iste karte.

I na samom kraju nam ostaje samo inicijalizirati objekte klasa Player i Game.

let player1 = new Player("Player1");

let player2 = new Player("Player2");

let game = new Game([player1, player2]);

Klasa Game u konstruktor prihvaća niz objekata klase Player.

# ZAKLJUČAK

# DODATAK A