Autori: © Divna Krpan, Saša Mladenović, Goran Zaharija

OOP Vježbe 11
Događaji

—

Bilješke

HTML stranice i elementi

Osnovni jezik koji koristimo za izradu web stranica je HTML (engl. HyperText Markup Language). On našim web preglednicima (engl. web browser) daje podatke o sadržaju i strukturi učitane web stranice, a preglednik od tih podataka oblikuje stranicu kakvu mi vidimo. HTML kôd sastoji se od oznaka (engl. tag). HTML element definiran je početnom oznakom, sadržajem i završnom oznakom.

```
<oznaka>Sadržaj</oznaka>
```

Primjer naslova poglavlja (eng. heading):

```
<h1>My First Heading</h1>
```

Drugi primjer je ***canvas*** element kojeg smo također već upoznali:

```
<canvas></canvas>
```

Osnovna struktura stranice izgleda ovako:

```
<!DOCTYPE html>
<head>
  <title>Naslov stranice</title>
</head>
<body>
  <h1>OOP</h1>
</body>
</html>
```

HTML stranica sastoji se od zaglavlja (head) i tijela (body). Unutar **body** dijela dodajemo sadržaj u obliku teksta ili različitih HTML elemenata. Osim *canvas* elementa malo detaljnije ćemo pogledati ***button*** jer se često koristi za pokretanje akcija. Već smo ga koristili u početnim vježbama.

```
<button>Klikni me</button>
```

Unutar tijela stranice dodajemo *button* element. Koristit ćemo ga za povezivanje s događajima, a da bi to mogli napraviti, moramo ga na neki način "dohvatiti". To ćemo napraviti na sljedeći način:

- Dodajemo atribut kojim ga možemo identificirati: ***id***
- Pozovemo metodu ***getElementById()*** koja pripada objektu ***document***.

Kad se HTML dokument učita u web preglednik, onda postaje objekt ***document*** koji sadrži HTML elemente. Objekt *document* je svojstvo objekta ***window***, a *window* predstavlja prozor web preglednika te osim same stranice sadrži i razne druge stvari (detalji nam trenutno nisu bitni). Objektu *document* možemo pristupiti pozivanjem *window.document* ili samo *document*.

Ako definiramo *button* element s jedinstvenim atributom ***id***:

```
<button id="btnPrvi">Klikni me</button>
```

onda element možemo dohvatiti sljedećom naredbom:

```
let btn = document.getElementById("btnPrvi");
```

To nam je dovoljno kako bi mogli koristiti *button* element za povezivanje s događajima.

Događaji

Događaji su najčešće rezultat neke akcije koju napravi korisnik (klik mišem, pritisak tipke na tipkovnici, povlačenje (eng. drag and drop), pomak miša, ...), ali može ih izazvati i web preglednik (npr. učitavanje stranice i sl.). Razlikujemo *ugrađene* (eng. built-in) događaje od korisnički definiranih.

Ugrađeni događaji

Primjeri ugrađenih ili zadanih događaja su: *click, mousemove, mousedown, mouseup, load, resize, scroll, ...*

Događaje povezujemo s nekim HTML elementom definiranjem "osluškivača" (eng. event listener) kao i posljedice tog događaja. Nakon izvršenog događaja poziva se akcija ili funkcija za obradu ili upravljanje događajem (eng. event handler). *Event handler* je zapravo "callback" funkcija koja se aktivira kao posljedica nekog događaja.

Postoji više načina na koje možemo povezati funkciju (ili metodu) za upravljanje događajem s odabranim HTML elementom, ali ovdje ćemo promatrati samo jedan način koji se često preporučuje, a to je korištenjem metode ***addEventListener()***:

```
btn.addEventListener("tipDogađaja", funkcija_obrade);
```

Primjenom ove metode možemo dodati više funkcija koje će reagirati na isti događaj.

Primjer:

```
let btn = document.getElementById("btnPrvi");

btn.addEventListener("click", poruka);
btn.addEventListener("click", ispis);

function poruka() {
    alert("Klik!");
}

function ispis(event) {
    console.log(event);
}
```

U gornjem primjeru imamo dvije funkcije koje će reagirati na događaj klik. Kad god se dogodi neki događaj, svi podaci spremaju se u odgovarajući objekt (npr. gdje je bila strelica miša, koja tipka je pritisnuta, na koji element je korisnik kliknuo i sl.). Taj objekt se uvijek prosljeđuje *callback* funkciji s podacima (kao što možemo vidjeti u gornjem primjeru za funkciju ***ispis(event)***.

Korisnički definirani događaji

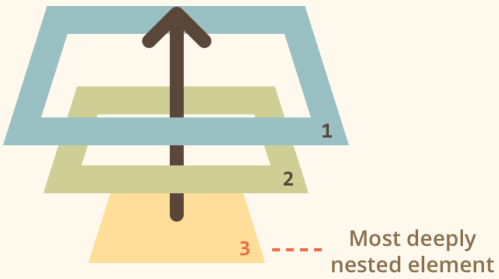
Korisnički definirane događaje sami definiramo stvaranjem odgovarajućih objekata. Možemo ih stvoriti pomoću konstruktora ***Event*** ili ***CustomEvent***. Konstruktor ***Event*** pozivamo na sljedeći način:

```
new Event(tip)
new Event(tip, opcije)
```

Parametri:

- **tip** - predstavlja string s nazivom događaja,
- **opcije** - predstavlja objekt sa sljedećim svojstvima:
    - ***bubbles*** - logička vrijednost (true/false),
    - ***cancelable*** - logička vrijednost (true/false),

Dodavanje opcija nije obavezno. Ako ih ne dodamo, zadana vrijednost je *false*. Npr. u slučaju ugniježđenih HTML elemenata, opcija *bubbles* određuje hoće li se neki događaj proslijediti nadređenom elementu (roditelju).



To je važno ako imamo takvu situaciju i želimo precizno odrediti koji element će reagirati na događaj. Primjer:

```
const novi = new Event("proba");
// osluškivanje
btn.addEventListener("proba", fun);
function fun(){ //ne radi ništa}
```

Inače web preglednik pokreće (eng. fire) ugrađene događaje, ali ako želimo programski pokrenuti događaj moramo koristiti metodu ***dispatchEvent()***. Metoda prima objekt tipa **Event**.

```
btn.dispatchEvent(novi);
```

Na taj način možemo pokrenuti i neki od ugrađenih događaja, odnosno simulirati događaj, npr. klik:

```
btn.dispatchEvent(new Event("click"));
```

U gornjem primjeru događaj *"click"* pokrećemo naredbom u JavaScript-u, a funkcija koja reagira na događaj će se pokrenuti kao da je korisnik stvarno kliknuo na *button* element. Ako događaj nećemo opet koristiti onda ga ne moramo spremati u varijablu.

Postoji način za razlikovanje događaja koje je stvarno pokrenuo korisnik u odnosu na one koje pokreće "skripta". Svojstvo ***isTrusted*** će biti *true* ako je događaj posljedica akcije korisnika, a *false* ako ga je generirala neka skripta (JavaScript program).

Ako je potrebno dodati još informacija uz sam događaj, onda možemo koristiti **CustomEvent** konstruktor koji ima svojstvo **detail**. Unutar ***detail*** možemo spremiti vlastite podatke. U sljedećem primjeru definiramo dva različita događaja **istog tipa**:

```
const catFound = new CustomEvent("animalfound", {
    detail: {
        name: "cat"
    }
});
const dogFound = new CustomEvent("animalfound", {
    detail: {
        name: "dog"
    }
});
```

Svaki od navedenih događaja ima svoje posebne informacije ovisno o tome koja životinja je pronađena. Element *btn* će osluškivati sve sve događaje tipa *"animalfound"* te će ispisati naziv iz polja *detail*:

```
btn.addEventListener("animalfound", zivotinja);
function zivotinja(ev) {
    console.log(ev.detail.name);
}
```

Nakon pokretanja događaja ispisuje se odgovarajući podatak:

```
btn.dispatchEvent(catFound); //cat
btn.dispatchEvent(dogFound); //dog
```

Na taj način možemo razlikovati različite događaje istog tipa te nije potrebno definirati dva različita tipa događaja (npr. *catfound* i *dogfound*) već je dovoljno dodati dodatne informacije.

- Zadatak 1.
  - Dodati novi *button* element pored ostalih elemenata tog tipa (kako ne bi promijenili strukturu stranice).
    - id: btnGame
  - Definirati novi događaj ***"gameover"*** kojeg je potrebno povezati s *btnGame* elementom.
  - Događaje će se pokrenuti kad se izgube svi životi.
- Zadatak 2.
  - Na isti način definirajte novi događaj tipa ***"levelup"*** kojeg ćemo pozvati u trenutku kad želimo prijeći na novu razinu igre.
  - Potrebno je odrediti uvjet kojeg igrač mora ispuniti kako bi prešao na iduću razinu pa neka je to broj bodova.
    - Zbog bržeg testiranja neka se nova razina pokrene kad skupi dva novčića (20 bodova).
  - U novoj razini igre opet se nalazi isti glavni lik, ali imat će samo jedan život (bez obzira na stanje koje je imao prije toga).
  - Lik *coin* više ne postoji već se pojavljuje lik ***cilj*** kojeg možemo vidjeti u mapi u Tiled-u.
  - Ako glavni lik dotakne *cilj*, onda je igra gotova.
- Zadatak 3.
  - U prethodnim zadacima došli smo do dvije situacije kad igra završava:
    - Kad glavni lik potroši sve živote (znači da nije uspio pa to možemo smatrati porazom).
    - Kad glavni lik dotakne cilj (ostvario je sve uvjete za uspješan završetak igre).
  - Premda su oba slučaja isti tip događaja "*gameover*", razlikuju se u tome je li glavni lik pobijedio ili izgubio igru.

- U slučaju pobjede ispišite poruku: “Pobjeda”.
- U slučaju poraza ispišite poruku: “Izgubili ste!!!”.

➢ Zadatak 4.

- Neka u drugoj razini igre lik postane dinosaur.

Napomena: Nakon završetka prve razine, potrebno je zaustaviti igru, pobrisati postojeće likove i stvoriti nove te ponovo pokrenuti igru. Ako to ne napravimo, može se dogoditi da neki od likova iz prve razine ostane na canvas-u te se čak nastavi kretati.