Izrada 2D retro videoigre pomoću Javascripta

Juić, Tim

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tim Juić

IZRADA 2D RETRO VIDEOIGRE POMOĆU JAVASCRIPTA

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tim Juić

Studij: Informacijski / poslovni sustavi

IZRADA 2D RETRO VIDEOIGRE POMOĆU JAVASCRIPTA

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Mario Konecki

Varaždin, 2023.

Tim Juić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom završnom radu istražuje se implementacija dvodimenzionalne retro pucačke video igre pomoću klasičnih web tehnologija poput HTML-a (eng. Hypertext Markup Language), CSS-a (eng. Cascading Style Sheets) i JavaScripta. Rad se temelji na izradi igrice koja simulira preživljavanje igrača protiv beskonačnih valova napada svemiraca. Kroz detaljnu analizu HTML5 platna, objašnjeni su procesi crtanja grafičkih elemenata, animacije i detekcije sudara. Ključne teme obuhvaćaju petlju igre, generiranje valova neprijatelja, upravljanje korisničkim unosom i resursima igre. Kroz praktični primjer implementacije igre, rad ilustrira primjenu teorijskih koncepta u stvarnom okruženju. Cilj je pružiti uvid u tehničke aspekte razvoja igara bez upotrebe gotovih okvira, biblioteka ili motora igara.

Ključne riječi: videoigra; JavaScript; animacija; HTML5 platno; prepoznavanje sudara; generiranje protivnika; upravljanje resursima

Sadržaj

1. Uvod

Završni rad obrađuje implementaciju dvodimenzionalne (2D) retro pucačke igrice pomoću jezika JavaScript i ostalih klasičnih web tehnologija. Igrica je izgrađena od temelja te se na taj način detaljno razrađuju mnogi tehnički i konceptualni aspekti izrade koji su u tom procesu korišteni. Riječ je o dvodimenzionalnoj video igri preživljavanja gdje se igrač suočava sa beskonačnim valovima napada svemiraca. Rad je podijeljen na teorijski i praktični dio.

Teorijski dio rada fokusira se na teorijsku osnovu koja obuhvaća temeljne koncepte HTML-a (eng. Hypertext Markup Language), CSS-a (eng. Cascading Style Sheets) i programskog jezika JavaScript. Nakon toga pažnja se posvećuje upotrebi HTML5 platna za crtanje grafičkih elemenata i animaciju. Detaljno je objašnjena tehnika i postupak crtanja raznih oblika i slika na platno te načini animiranja tih elemenata. Kratko se istražuje i postupak prepoznavanja sudara između elemenata na platnu koji je ključan u razvoju videoigre.

Praktični dio rada obrađuje strukturu, proces i tehnologije korištene u izradi igrice. Upotrebljavajući teorijske koncepte iz prvog dijela, detaljno se prikazuje kako igra funkcionira te od kojih se komponenata sastoji. Najbitnije teme uključuju petlju igre, proces generiranja valova neprijatelja, upravljanje korisničkim unosom i resursima igre.

Motivacija za odabir ove teme proizlazi iz prethodnog znanja i iskustvu u izradi računalnih igara u web preglednicima, dobro poznavanje web tehnologija te osobne naklonosti prema retro pucačkim igrama. Kroz ovaj rad, cilj mi je dublje istražiti tehničke aspekte izrade ove vrste igara te podijeliti dio tog znanja s drugima.

2. Metode i tehnike rada

Temeljne tehnologije korištene u ovom radu su klasični alati za izradu web stranica. To uključuje HTML, CSS i JavaScript. HTML i CSS su prvenstveno služili za dizajn i strukturu vizualnih elemenata, iskočnih prozora i izbornika, dok je programski jezik JavaScript služio za implementaciju programske logike video igrice. Osim njih nisu korištene dodatne biblioteke, okviri ili već gotovi motori za razvoj igara. Igrica je u potpunosti izgrađena od nule kako bi se pokazalo funkcioniranje stvari u pozadini. Za uređivanje teksta koristila se aplikacija Visual Studio Code. Sav programski kod je verzioniran pomoću alata git i platforme GitHub, omogućavajući praćenje promjena i lako vraćanje na prethodne verzije. Kod je javno dostupan i može mu se pristupiti preko poveznice https://github.com/timjuic/js-retro-game. Video igra poslužena je preko alata Firebase te joj se pristupa preko poveznice https://alien-invasion-2a9d4.web.app/.

3. Teorijski dio

3.1 HTML, CSS i JavaScript

HTML (eng. Hypertext Markup Language), CSS (eng. Cascading Style Sheets) i JavaScript su temeljne 3 tehnologije koje se koriste za izradu web stranica i aplikacija. Slijedi kratak opis gore navedenih tehnologija.

HTML je najosnovniji građevni blok weba. On definira značenje i strukturu web sadržaja te govori web pregledniku kako prikazati sadržaj („HTML: Hypertext Markup Language“, 2023). Može ga se zamisliti kao kostur web stranice koji se sastoji od raznih elemenata. Ti elementi se koriste kako bi se definirao sadržaj i struktura stranice. Na primjer to su odlomci teksta, naslovi, liste, poveznice, slike, videozapisi i drugi multimedijalni sadržaji. Osim što definira strukturu stranice, HTML je ključan za pristupačnost na webu te pruža semantičke informacije o sadržaju stranice. One su bitne kako bi preglednik na ispravan način interpretirao sadržaj stranice. Svaki element može sadržavati i atribute. Atributi se u HTML elemente postavljaju nakon naziva oznake elementa te oni pružaju dodatne informacije o elementima. Obično se definiraju u parovima ime – vrijednost. Najčešće korišteni atributi su klasa i identifikacijska oznaka (eng. „id“). Oni se definiraju kako bi se elementima u kojima se nalaze lakše pristupilo sa strane JavaScripta ili CSS-a o kojima će sada biti riječi.

CSS (Cascading Style Sheets) je stilski jezik koji se koristi za mijenjanje i uređivanje izgleda stranice, odnosno sadržaja napisanog u HTML-u („CSS: Cascading Style Sheets“, 2023). On omogućava postavljanje fontova, boja, raspoređivanje i premještanje elemenata, mijenjanje njihove veličine i oblika te još mnoge druge vizualne aspekte u web stranicama. CSS upute se obično pišu u odvojenoj datoteci od HTML-a. Tako se razdvaja sadržaj stranice od prezentacije što povećava fleksibilnost i olakšava održavanje web stranica. Ipak, on se može i direktno primijeniti na HTML elemente tako da se stilovi definiraju unutar ranije spomenutih atributa.

JavaScript je skriptni, interpreterski, jednodretveni programski jezik više razine koji je razvijen za rad u web preglednicima, za izradu interaktivnih i dinamičkih web stranica („JavaScript“, 2023). Danas se koristi na mnogim mjestima pa se s njim može razvijati i serverski kod, pa čak i mobilne i desktop aplikacije. Javascript je objektno orijentirani jezik, ali podržava i druge paradigme programiranja kao što su proceduralno i funkcionalno programiranje. On je jednodretveni jezik baziran na događajima (eng. Events). Osmišljen je 1995. godine, i isprva nije imao puno mogućnosti, ali danas podržava mnoge koncepte modernog programiranja kao što su asinkrono programiranje, obećanja (eng. Promises), skraćeni zapis funkcija i još mnoge druge.

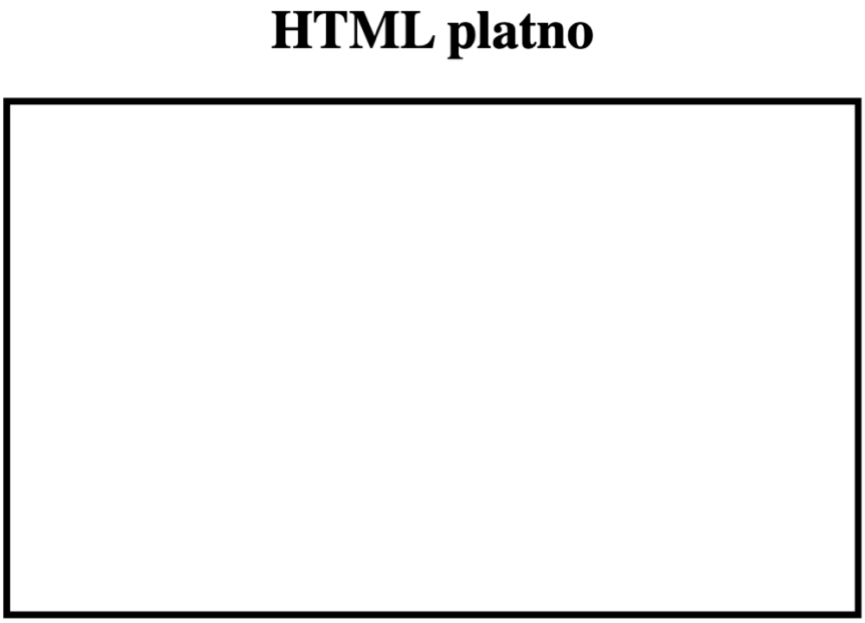
„Bilo koja aplikacija koja može biti napisana u JavaScriptu, na kraju će biti napisana u JavaScriptu.“ (Jeff Atwood, 2007).

3.2 Canvas

Jedan HTML element u ovom radu je posebno bitan, a to je platno (engl. Canvas). Platno je pravokutno polje na kojem se pomoću programskog jezika JavaScript može crtati rasterska grafika („HTML Canvas Tag“, bez dat.). Platno je element od iznimne važnosti jer omogućuje dinamičko crtanje i animiranje elemenata što je ključno za interaktivnost i fluidnost svake videoigre. Pomoću platna, mogu se izravno manipulirati pojedini pikseli na ekranu, stvarati složeni objekti i oblici te isti i animirati. Najčešće služi za crtanje dvodimenzionalnih (2D) oblika, ali ima mogućnost crtanja i vrlo naprednih trodimenzionalnih (3D) oblika.

Prije nego što se krene u daljnji opis platna, bitno je spomenuti da je platno samo spremnik ili polje na kojem se može crtati, dok se crtanje odvija pomoću jezika JavaScript. Platno je u početku prazno, bez obruba i sadržaja. Za razliku od ostalih HTML elemenata, platno ne može imati unutarnje elemente, odnosno elemente djecu. Ukoliko se postavi element dijete unutar platna, ono neće biti vidljivo na stranici. Slijedi isječak HTML koda koji prikazuje dodavanje platna u web stranicu pri čemu mu je dodijeljen atribut „id“ sa vrijednošću „moje-platno“ te nekoliko CSS uputa koje postavljaju boju pozadine platna u crnu te ga horizontalno centriraju u sredinu stranice. Ukoliko nije drugačije postavljeno, početna veličina platna je 300 piksela u širinu i 150 piksela u visinu.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body style="text-align: center;">
    <h1>HTML platno</h1>
    <canvas id="moje-platno" style="background-color: black; display:
inline-block;"></canvas>
</body>
</html>
```



Slika 1. Prikaz praznog platna sa crnim obrubom na web stranici (localhost).

3.3 Crtanje osnovnih oblika

Kao što je ranije spomenuto, po HTML platnu se crta pomoću programskog jezika JavaScript. U njemu postoji posebno sučelje za programiranje (eng. Application Programming Interface - API) koji koje sadrži razne metode za crtanje na platnu. Kako bismo pristupili tom sučelju, prvo je potrebno stvoriti referencu na njega, odnosno potrebno je dohvatiti HTML element platna iz strukture stranice koja je prethodno napravljena. Platnu je dodijeljen atribut „id“ sa vrijednošću „moje-platno“. Sada ga je iz JavaScripta moguće dohvatiti sa metodom „getElementById“ iz objekta „document“ koji predstavlja cijeli html dokument, odnosno strukturu. To je globalni objekt u JavaScriptu kojemu izravno može pristupiti. Spomenutoj metodi prosljeđuje se naziv identifikatora elementa kojemu pristupamo. Time je dobivena referenca platna iz HTML strukture. Na dobivenoj referenci je potrebno definirati kontekst pomoću kojeg će se crtati. Za to služi metoda „getContext“ koja traži argument vrste crtaćeg konteksta. Trenutno postoje četiri opcije koje je ovdje moguće definirati („HTMLCanvasElement: getContext() method“, 2023):

- Dvodimenzionalni kontekst - „2d“. Ovo je najčešće korišteni kontekst i onaj koji će ovdje biti korišten i najdetaljnije objašnjen. Sadrži velik skup metoda i svojstava

za crtanje 2D grafike, uključujući razne oblike, putanje i linije, tekst, slike i još mnogo toga.

- WebGL kontekst - „webgl“. To je kontekst za oblikovanje 3D grafike u web preglednicima koristeći grafički procesor. WebGL omogućuje crtanje naprednih 3D oblika, elemenata, efekata, sjena i ostalog. Direktno komunicira sa korisnikovom grafičkom karticom i zbog toga pruža dobre performanse. Koristi se za oblikovanje grafike u web alatima kao što su Google Karte. Nedostatak WebGL-a je što je težak za shvatiti i koristiti te je potrebno dobro znanje matematike o 3D grafici. Svi današnji web preglednici podržavaju WebGL.
- WebGL2 kontekst – „webgl2“. Ovo je evolucija WebGL-a te pruža poboljšanje funkcionalnosti i performanse za 3D grafiku.
- „webgpu“. WebGPU je je moderno grafičko sučelje za web preglednike i razvijen je od strane W3C skupine developera. WebGPU je nasljednik WebGL-a, pruža bolju kompatibilnost s modernim grafičkim karticama, brže operacije računanja i brži pristup naprednijim GPU opcijama. Ovo je relativno nova tehnologija i u većini web preglednika još nije podržana.

Kratak programski isječak ispod prikazuje dohvaćanje platna te odabir konteksta nad kojim će se crtati. U tijeku ovoga rada biti će korišten 2D kontekst.

```
let canvas = document.getElementById('moje-platno');
let context = canvas.getContext("2d");
```

3.3.1 Crtanje pravokutnika

Crtanje pravokutnika ili kvadrata je najlakše te je potrebna samo jedna linija koda sa metodom „fillRect“ koja prima četiri argumenta. Prva dva argumenta su početna horizontalna koordinata koja se obično naziva „x“ te početna vertikalna koordinata na platnu koja se naziva „y“. Naime, koordinatni sustav na platnu malo je drugačiji od tradicionalnog Kartezijevog sustava. Poznato je da u Kartezijevom sustavu x-os raste, odnosno gleda prema desno, y-os raste prema gore, a ishodišna točka je u sredini. Na platnu je ishodište u gornjem lijevom kutu platna, bez obzira gdje se ono nalazi na stranici. Lijevi gornji kut platna će uvijek biti pozicija (0, 0). Zbog toga se na platnu koordinate povećavaju prema dolje (koordinata y) i prema desno (koordinata x).

Sada je jasno što prva dva argumenta predstavljaju, a to je udaljenost od gornjeg desnog kuta platna te istovremeno početna pozicija crtanja pravokutnika. Druga dva argumenta su redom širina i visina pravokutnika, odnosno drugim riječima koliko daleko će se

pravokutnik crtati prema desno od početne pozicije (širina), te koliko prema dolje (visina). Prije instrukcije crtanja, može se definirati boja u kojoj se želi crtati. Ukoliko nije definirano, početna boja je crna. S obzirom da je platno u ovom slučaju crno, boja je postavljena na zelenu. Sada će svi elementi koji su nacrtani nakon definiranja boje biti zeleni, sve dok nije drugačije određeno.

```
context.fillStyle = 'green';
context.fillRect(150, 100, 100, 100); // x, y, širina, visina
```

3.3.2 Crtanje kruga

Crtanje kruga također je vrlo jednostavno, ali poznavajući osnove geometrije, jasno je da se on crta na drugačiji način. Krug se crta pomoću putanje slično kao i na papiru. Putanja se započinje metodom „beginPath“ nad kontekstom platna. Zatim se metodom „arc“ crta krivulja kruga koja prima pet argumenata koji su redom:

- Horizontalna koordinata sredine kruga
- Vertikalna koordinata sredine kruga
- Polumjer kruga
- Početni kut crtanja kruga izražen u radijanima s početkom mjerenja na pozitivnijem dijelu x-osi
- Završni kut crtanja kruga izražen u radijanima s početkom mjerenja na pozitivnijem dijelu x-osi

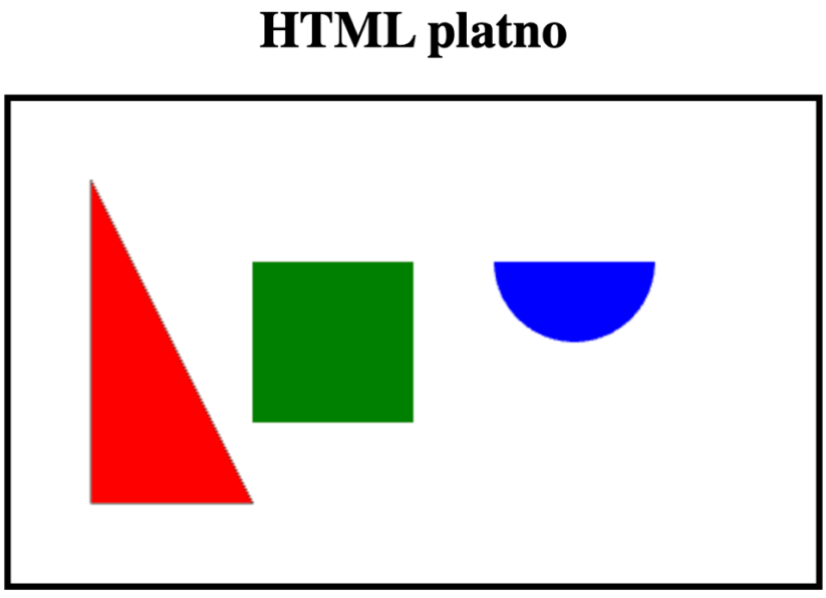
Konačno, ukoliko se crta krug, potrebno je pozvati funkciju koja će popuniti obilježenu putanju sa definiranom bojom. Slijedi objašnjeni isječak programskog koda, koji će nacrtati plavi krug sa središtem u točki (150, 150) sa polumjerom 50 piksela. U slučaju da je potrebno nacrtati samo polovicu kruga, kao zadnji argument može se proslijediti upola manja vrijednost u radijanima.

```
context.fillStyle = 'blue';
context.beginPath();
context.arc(350, 100, 50, 0, 1 * Math.PI); // x, y, radijus, početni kut, završni kut
context.fill();
```

3.3.3 Crtanje trokuta

Trokut se može jednostavno nacrtati povezivanjem tri točke sa linijom. Dakle opet se koristi „beginPath“ metoda koja započinje novu putanju. Metoda „moveTo“ se koristi za pomicanje na početnu lokaciju za crtanje trokuta koja će ujedno biti i jedna od kutova trokuta. Nakon toga pozivanjem metode „lineTo“ crtaju se stranice trokuta tako da joj proslijede koordinate točke. Za svaku stranicu potrebno je pozvati spomenutu metodu, no za zadnju stranicu ipak se može iskoristiti metoda „closePath“ koja će zatvoriti trokut, odnosno putanju tako da povuće liniju do početne pozicije. Time se izbjegava upisivanje iste koordinate kao argument dva puta. Na kraju je potrebno pozvati metodu „stroke“ koja će nacrtati definiranu putanju, odnosno „fill“ koja će obojati cijelu unutrašnjost trokuta.

```
context.fillStyle = "red"
context.beginPath();
context.moveTo(50, 50);    // vrh trokuta
context.lineTo(50, 250);    // lijevi kut
context.lineTo(150, 250);   // desni kut
context.closePath();        // zatvara putanju tako da se vrati na početnu
točku
context.stroke();
context.fill();
```



Slika 2. Platno sa nacrtanim geometrijskim oblicima (localhost)

Iako je u pokazanim primjerima objašnjeno crtanje samo jednostavnijih oblika, treba znati da se pomoću HTML5 platna može crtati bilo kakav oblik ili krivulja. Pitanje je samo koliko složen i kompleksan postupak crtanja postane. Nekad neće biti isplativo crtati neki jako složen objekt ili krivulju. U tom slučaju može se iskoristiti opcija platna za crtanje slika.

3.4 Crtanje slika

HTML5 platno ima još jednu izuzetno korisnu mogućnost, a to je crtanje slika. Time je moguće dodavanje kompleksnijih efekata, grafika ili pozadina za platno, bez potrebe za ručnim crtanjem svakog detalja. Postupak crtanja slike vrlo je jednostavan i sličan načinu kako se crta i pravokutnik u prethodnom primjeru.

Za crtanje željene slike prvo je potrebno imati referencu na nju koja će se proslijediti metodi za crtanje. Referenca se može dobiti kreiranjem nove instance slike te definiranjem putanje do željene slike. Nakon toga se može pozvati metoda „drawImage“ koja je zadužena za crtanje. Metoda je vrlo slična metodi za crtanje pravokutnika. Jedina razlika je taj što traži referencu slike. Dakle, redom su to argumenti:

- Referenca, odnosno instanca željene slike
- Koordinata početka crtanja na x-osi
- Koordinata početka crtanja na y-osi
- Željena širina slike na platnu u pikselima
- Željena visina slike na platnu u pikselima

Iako je postupak crtanja slika jednostavan, potrebno je imati na umu jednu bitnu stvar. Naime, proces učitavanja slike može zahtijevati određeno vrijeme. Jednom kada je definirana putanja do slike, ona odmah neće biti spremna za prikaz. Točno trajanje toga ovisi o dimenzijama slike, brzini internetske veze, ali i odzivu servera ako se podaci preuzimaju s vanjske lokacije. Stoga je ključno osigurati da se slika ne pokušava prikazati prije nego što je potpuno učitana. To se može postići korištenjem događaja "onload" koji je povezan s objektom slike. Ovaj događaj aktivira se čim je slika spremna. Postavljanjem metode za crtanje unutar njega, osigurano je da se neće pokušati crtati prije nego što je učitana. Slijedi opisani isječak programskog koda.

```
// Stvaranje nove instance slike
```

```
const slika = new Image();

// Postavljanje putanje do slike
slika.src = 'slike/slika1.jpg';

// Crtanje slike na canvasu kada je slika učitana
slika.onload = function() {
    context.drawImage(slika, 0, 0, canvas.width, canvas.height);
}
```

3.5 Crtanje nakošenih elemenata

Do sad su svi elementi koji su se crtali na platnu bili paralelni sa pozicijom platna, odnosno nisu bili nakošeni. Često će biti potrebno da oblici koji se crtaju budu nakošeni ili da se vrte u nekakvoj animaciji. Stoga će u nastavku biti objašnjeno kako ovo postići. Crtat će se pravokutnik koji je zarotiran 40 stupnjeva u smjeru kazaljke na satu. Kako bi stvari bile jednostavnije, kreiran je objekt pravokutnik sa sljedećim atributima koje će se koristiti kao parametri za crtanje:

- Pozicija na x-osi
- Pozicija na y-osi
- Širina pravokutnika
- Visina pravokutnika
- Rotacija pravokutnika izražena u stupnjevima
- Boja pravokutnika

Crtanje ukošenih elemenata temelji se na rotaciji konteksta platna za određeni broj stupnjeva. Zbog toga svi elementi, nacrtani nakon postavke rotacije, automatski postaju ukošeni. Vrijednost rotacije u metodi “rotate” mora biti definirana u radijanima. Osim toga, ne smije se zanemariti ishodište rotacije. Koristeći samo metodu “rotate”, oblici će se ukositi u odnosu na ishodišnu točku platna, odnosno gornji lijevi kut. No u ovom slučaju se želi ukositi element s obzirom na njegovo središte. Kako bi se to postiglo, potrebno je prilagoditi kontekst platna središtu pravokutnika. Središte pravokutnika određuje se zbrojem njegove početne koordinate i polovine njegove širine, odnosno visine. Kontekst platna pomiče se pomoću

metode „translate“, koja prima informacije o tome za koliko se jedinica ili piksela treba pomaknuti po svakoj osi. Budući da se početna točka nalazi na koordinatama (0, 0), samo je potrebno proslijediti koordinate središta pravokutnika. Nakon što se platno pomakne, može se primijeniti metoda rotacije. Prije samog crtanja pravokutnika, preporučljivo je vratiti kontekst platna na početnu poziciju kako se ne bi trebao korigirati položaj prilikom crtanja. Kontekst se može vratiti istom metodom „translate“, no s negativnim vrijednostima. Kontekst platna ostaje ukošen, i sada je spremno za crtanje.

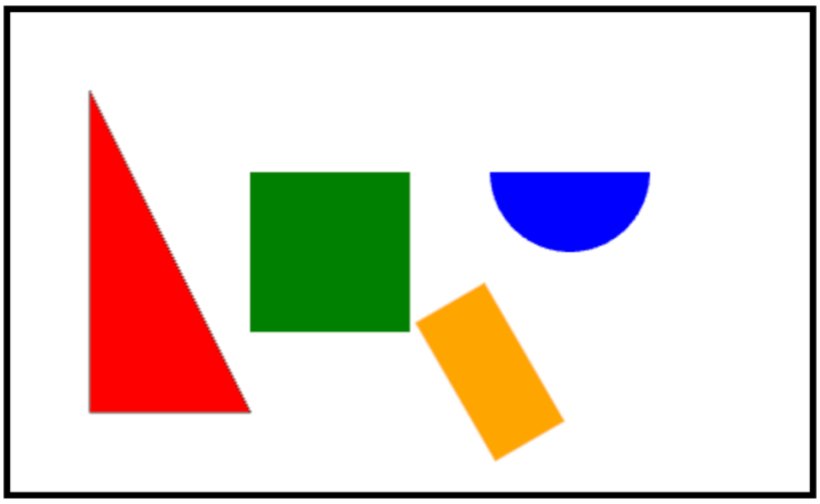
Nakon crtanja svakog ukošenog elementa, preporučljivo je vratiti kontekst platna u prvotno stanje, odnosno stanje prije rotacije. Ako ga se ručno ne vrati, platno će zadržati rotaciju. U tu svrhu se koriste metode „spremi“ (eng. Save) i „vrati“ (eng. Restore) koje se nalaze u objektu konteksta platna. Prva metoda čuva trenutno stanje konteksta, dok druga vraća prethodno spremljeno stanje. Stoga je bitno stanje pohraniti prije početka crtanja ukošenih elemenata, a obnoviti ga nakon što je crtanje završeno. Na taj način se osigurava da se svaki put prilikom crtanja radi s platnom u njegovom izvornom stanju. U nastavku slijedi isječak programskog koda koji navedeno prikazuje.

```
let pravokutnik = {
    pozicijaX: 50,
    pozicijaY: 50,
    sirina: 100,
    visina: 50,
    rotacijaStupnjevi: 50,
    boja: 'orange',
}

let sredisteX = pravokutnik.pozicijaX + pravokutnik.sirina / 2;
let sredisteY = pravokutnik.pozicijaY + pravokutnik.visina / 2;


context.fillStyle = pravokutnik.boja;
context.save();
context.translate(sredisteX, sredisteX); // pomakni koordinatni sustav u
središte pravokutnika
context.rotate(pravokutnik.rotacijaStupnjevi * (Math.PI / 180)); //
zarotiraj koordinatni sustav za zadani kut (potrebni radiani)
context.translate(-sredisteX, -sredisteY); // pomakni koordinatni sustav u
središte pravokutnika
context.fillRect(pravokutnik.pozicijaX, pravokutnik.pozicijaY,
pravokutnik.sirina, pravokutnik.visina); // nacrtaj pravokutnik
```

```
context.restore();
```



Slika 3. Platno sa novo dodanim nakošenim pravokutnikom (localhost)

3.6 Pomicanje i animacija elemenata

Sada kad je poznato kako se razni elementi i oblici na platno, bitno je naučiti i kako ih animirati. Osnovna svrha korištenja HTML5 platna je mogućnost kreiranja glatkih i brzih animacija. Jasno je da se animacije mogu raditi i pomicanjem pravih HTML elemenata, ali su znatno sporije u usporedbi sa crtanjem na platnu.

Animacije na platnu funkcioniraju na sljedeći način. Recimo da postoji kvadrat čije pomicanje se želi animirati. Proces animiranja sastoji se od tri ključna koraka:

1. Brisanje prošle pozicije kvadrata sa platna, odnosno čišćenje platna
2. Pomicanje kvadrata na novu poziciju
3. Crtanje nove pozicije kvadrata

Te tri stvari se moraju konstantno ponavljati kako bi se stvorila animacija. Animacije se obično stvaraju pomoću petlji animacije (eng. Animation Loop). Metoda te petlje se obično poziva jako brzo kako bi animacija bila glatka i primamljiva ljudskom oku. Danas je to obično 60 puta u sekundi, s obzirom da većina ekrana osvježava sliku tom brzinom. Drugim riječima, trajanje svake pojedinačne slike u ovom slučaju traje otprilike 16 milisekundi. Poznato je da

sekunda ima tisuću milisekundi, kada se to podijeli sa 60 sličica u sekundi, dobije se navedeni rezultat. Vrijedi spomenuti da se animacije mogu odvijati i brže i sporije od spomenute brzine, a sve to ovisi o cilju i potrebi animacije i brzini osvježavanja ekrana.

U programskom jeziku JavaScript petlje animacije se obično rade pomoću jednog od sljedeća dva načina.

Prvi način je korištenjem metode „setInterval“. Ona omogućuje periodično izvršavanje željene funkcije u jednakim vremenskim intervalima („setInterval() global function“, 2023). Pri pozivanju, kao prvi argument zahtjeva funkciju koja će se izvršavati, a kao drugi razmak između njezinih poziva izražen u milisekundama. Uobičajeno, predana funkcija obuhvaća radnje kao što su crtanje, brisanje i pomicanje elemenata na platnu čime se stvara animacija. Nedostatak ovog pristupa je da će brzina animacije biti konstantna i neće ovisiti o brzini osvježavanja korisnikovog ekrana. To znači da ukoliko se brzina pozivanja metode postavi na 60 poziva u sekundi, a korisnikov monitor može prikazati samo 30, metoda će se prebrzo pozivati i time trošiti duplo više resursa nego što je zapravo potrebno. Isto tako se može dogoditi i obratno, da korisnik ima moderan monitor sposoban za prikazivanje 144 sličice u sekundi. U tom slučaju animacija će biti prespora za taj monitor.

Drugi način je korištenjem „requestAnimationFrame“ metode. Poziva ju se kada se želi nacrtati nova slika u animaciji. Prosljeđuje joj se samo metoda koju se želi pozvati za iscrtavanje nove slike animacije. Ova metoda zakazuje povratni poziv na dobivenu metodu i govori pregledniku da se želi nacrtati sljedeći korak animacije („Window: requestAnimationFrame() method”, 2023). Kada je preglednik spreman, poziva se proslijeđena metoda i proces se ponavlja. Bitno je da ta metoda sadrži unutar sebe poziv „requestAnimationFrame“ funkcije jer ona zakazuje samo jedan poziv, za razliku od „setInterval“ koji konstantno poziva željenu metodu. Ovo je preporučeni način za stvaranje animacija u modernom JavaScriptu. Prednosti korištenja ovog načina su da preglednik upravlja brzinom animacije ovisno o brzini osvježavanja monitora.

U sljedećem primjeru će biti objašnjeno kako stvoriti animaciju u kojoj se kvadrat kreće po ekranu i odbija od rubova ekrana. Animacija će biti identična onima na klasičnim zaštitnicima ekrana (eng. Screen Saver). U implementaciji spomenutoga koriste se klase „Animacija“ i „Pravokutnik“. Pravokutnik je oblik koji će se animirati te posjeduje svojstva kao što su pozicija na x osi, pozicija na y osi, širina i visina, brzina kretanja na x i y osi te boja. Uz to, sadrži i metode „nacrtaj“, „obriši“ i „pomakni“ koje odgovaraju ranije navedenim koracima potrebnim za animiranje. Metoda za crtanje crta pravokutnik na njegovoj lokaciji uzimajući u obzir njegovu širinu i visinu. Metoda za brisanje čisti cijelo platno od prijašnjeg crtanja. Metoda

za pomicanje pomiče pravokutnik na novu lokaciju tako da njegove koordinate poveća za iznos odgovarajuće brzine.

Klasa animacija sadrži metodu „animiraj“ stvara animaciju pozivanjem gore spomenutih metoda te zakazuje sljedeći korak animacije pomoću „requestAnimationFrame“ funkcije. Osim metode za animiranje, klasa posjeduje i metode za zaustavljanje i pokretanje animacije i ima pristup objektu pravokutnika kako bi pozivala njegove funkcije. Slijedi objašnjeni isječak JavaScript programskog koda.

```
class Pravokutnik {
    constructor(pozicijaX, pozicijaY, sirina, visina, brzinaX, brzinaY, boja) {
        this.pozicijaX = pozicijaX;
        this.pozicijaY = pozicijaY;
        this.sirina = sirina;
        this.visina = visina;
        this.brzinaX = brzinaX;
        this.brzinaY = brzinaY;
        this.boja = boja;
    }

    nacrtaj() {
        context.fillStyle = this.boja;
        context.fillRect(this.pozicijaX, this.pozicijaY, this.sirina, this.visina);
    }

    obrisi() {
        context.clearRect(0, 0, canvas.width, canvas.height); // Brisanje cijelog platna
    }

    pomakni() {
        this.pozicijaX += this.brzinaX;
        this.pozicijaY += this.brzinaY;

        if (this.pozicijaX <= 0 || this.pozicijaX + this.sirina >= canvas.width) {
            this.brzinaX = -this.brzinaX;
        }
```

```
        if (this.pozicijaY <= 0 || this.pozicijaY + this.visina >=
canvas.height) {
            this.brzinaY = -this.brzinaY;
        }
    }
}

class Animacija {
    constructor() {
        this.pravokutnik = new Pravokutnik(0, 0, 20, 20, 3, 3, "blue");
        this.aktivno = false;
    }

    pokreni() {
        if (!this.aktivno) {
            console.log('Pokretanje');
            this.aktivno = true;
            this.animiraj();
        }
    }

    zaustavi() {
        this.aktivno = false;
    }

    animiraj() {
        this.pravokutnik.pomakni();
        this.pravokutnik.obrisi();
        this.pravokutnik.nacrtaj();

        if (this.aktivno) {
            requestAnimationFrame(this.animiraj.bind(this));
        }
    }
}

let animacija = new Animacija();
animacija.pokreni()
```

3.7 Prepoznavanje sudara

Prepoznavanje sudara igra ključnu ulogu u raznim videoigrama, simulacijama i animacijama. To je proces, odnosno algoritam koji omogućuje identifikaciju trenutaka kada se dva ili više objekata preklapaju ili dodiruju.

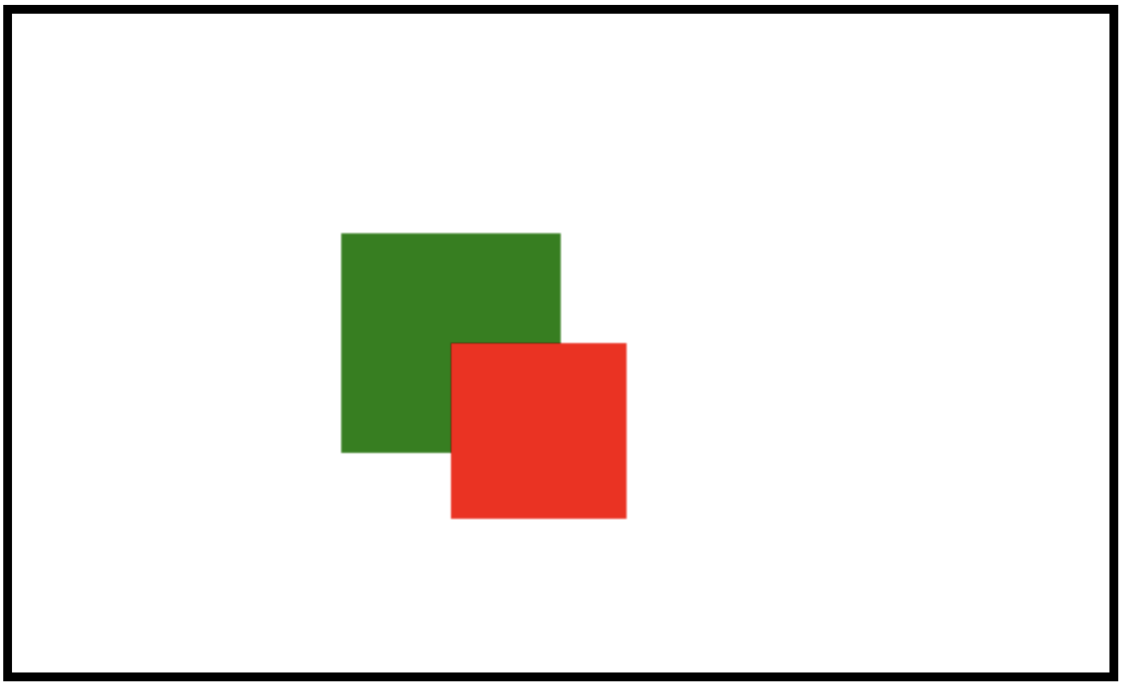
Detekcija sudara je računalni problem otkrivanja sjecišta ili preklapanja dvaju ili više objekata. Prepoznavanje sudara klasično je pitanje računalne geometrije te ima primjenu u mnogim područjima računalstva, ponajviše u računalnoj grafici, računalnim igrama, simulacijama i robotici („Collision Detection“, 2023). Postoje razne vrste algoritama za prepoznavanje sudara. Mogu se podijeliti na 2D i 3D algoritme te se još razlikuju po tome između kakvih oblika prepoznaju sudar. To može biti od običnih pravokutnika i krugova do izuzetno složenih i nepravilnih 3D oblika.

Ovdje će biti riječ o prepoznavanju sudara između pravokutnika poravnatim sa koordinatnim osima (eng. Axis-Aligned Collision Detection). Ovo je jedan od najjednostavnijih, ali i najčešće korištenih algoritama. Svrha mu je utvrditi sudaraju li se ili preklapaju dva pravokutna objekta koji su poravnati, odnosno paralelni sa koordinatnim osima. Korištenje ovog algoritma smanjuje potrebu za složenijim i sporijim geometrijskim proračunima čime se postiže brza i efikasna provjera sudara. U sljedećem programskom primjeru implementirana je provjera sudara između dva objekta, odnosno pravokutnika. Implementacija je u obliku jednostavne metode koja prima dva argumenta koji su u ovom okruženju nazvani „entitet“ (eng. Entity). To su instance klase „Pravokutnik“ identične onoj u primjeru u prethodnom poglavlju. Prepoznavanje sudara se sastoji od ukupno četiri provjere po dvije za svaku koordinatnu os.

1. Provjera nalazi li se lijeva strana prvog objekta (njegova X koordinata) lijevo od desne strane drugog objekta (njegova X koordinata plus širina). Ako je to istina, to znači da postoji horizontalni preklapajući dio između ta dva entiteta.
2. Provjera nalazi li se desna strana prvog objekta (njegova X koordinata plus širina) desno od lijeve strane drugog objekta (njegova X koordinata). Ako je to istina, to ukazuje na drugi horizontalni preklapajući dio između entiteta.
3. Provjera nalazi li se gornja strana prvog objekta (njegova Y koordinata) iznad donje strane drugog objekta (njegova Y koordinata plus visina). Ako je to istina, to ukazuje na vertikalni preklapajući dio između entiteta.

4. Provjera nalazi li se donja strana prvog objekta (njegova Y koordinata plus visina) ispod gornje strane drugog objekta (njegova Y koordinata). Ako je to istina, to ukazuje na drugi vertikalni preklapajući dio između entiteta.

```
collidesWithEntity(entity1, entity2) {
      return (
         entity1.posX < entity2.posX + entity2.width &&
         entity1.posX + entity1.width > entity2.posX &&
         entity1.posY < entity2.posY + entity2.height &&
         entity1.posY + entity1.height > entity2.posY
      );
}
```



Slika 4. Platno sa dva preklapajuća kvadrata (localhost)

Ako su sve 4 od ovih provjera točne, metoda vraća pozitivan rezultat što znači da se objekti preklapaju, odnosno sudaraju. Ukoliko je samo jedna od provjera netočna, vraća se negativan rezultat. Algoritam se isprva može činiti malo zbunjujući u smislu zašto su potrebne baš četiri provjere. Razlog tomu je taj što se provjera odvija na dvodimenzionalnom sustavu. Teoretski, kada bi bila riječ o jednodimenzionalnom sustavu govorilo bi se o

mogućem sudaranju između dvije dužine na pravcu. Jasno je da je pravac jednodimenzionalni sustav. U tom slučaju bile bi dovoljne samo prve dvije provjere od gore spomenutih četiri.

4. Praktični dio

4.1 Uvod u igru

U ovom poglavlju opisan je proces implementacije 2D retro pucačke igrice. Igrica je napravljena pomoću već spomenutih HTML, CSS i JavaScript tehnologija te se za crtanje koristio HTML element platno. Osim spomenutih tehnologija nije korištena niti jedna dodatna biblioteka, okvir, motor igre ili nešto slično. Sve je u potpunosti implementirano ispočetka. S obzirom na korištene tehnologije, igrica se igra preko web preglednika. U implementaciji igre korišteni su koncepti objašnjeni u teorijskom dijelu rada i još mnogi drugi koji će tek biti spomenuti u ovom dijelu. Iako je ovo dvodimenzionalna igrica, implementacija nije sasvim jednostavna i riječ je o tisućama linija koda te više od stotinu različitih datoteka. Stoga će biti objašnjeni osnovni i najbitniji dijelovi programskog koda i logike.

4.2 Opis i mehanika igre

Riječ je o dvodimenzionalnoj pucačkoj igrici preživljavanja gdje se igrač suočava sa beskonačnim valovima napada svemiraca. Igrica je inspirirana i ima elemente starijih arkadnih igara. Radnja se odvija na ravnoj statičnoj 2D površini koja je prikazana odozgo. Cilj igrice je što duže preživjeti beskonačne valove napada svemiraca. Svemirci se stvaraju sa svih strana i napadaju u sve većim, jačim, bržim i općenito težim valovima. Postoji više vrsta svemiraca gdje svaki od njih ima svoje karakteristike, a neki imaju i super moći. Razlikuju se u izgledu, veličini, brzini kretanja, količini energije i snazi, dok oni jači imaju i specijalnu sposobnost. Primjerice neki imaju mogućnost teleportacije, neki eksplodiraju kada su napadnuti, neki pucaju na igrača, a neki čak stvaraju svoje manje svemirce koji ih štite. Kako bi se zaštitio, igrač mora koristiti svoje lasersko oružje te se strateški kretati po površini kako bi izbjegao neprijatelje. Kreće se pomoću tipkovnice sa klasičnim kontrolama strelica, a moguće je ići prema gore, dolje, lijevo, desno i u svim dijagonalama. Kretanje je ograničeno unutar granica površine i nije moguće izaći izvan. Miš se koristi za ciljanje sa laserskim oružjem prema neprijateljima. Držanjem lijeve tipke miša se aktivira oružje i počinje pucati. Pomoću srednje tipke miša igrač može birati između više načina pucanja. Neprijatelji gube snagu kada su pogođeni, a kada im snaga dosegne nulu, umru sa zanimljivim efektom eksplozije. Na jednak način igrač gubi zdravlje kada dođe u kontakt sa vanzemaljcima ili bude pogođen od strane njihovih projektila. Preostalo zdravlje igrača i vanzemaljaca je prikazano pomoću indikatora zdravlja koji se nalazi ispod njih. Igra završava kada igrač izgubi svo zdravlje. Mjeri se vrijeme koliko dugo je igrač preživio napade što izravno predstavlja njegovu uspješnost u igranju.

Igrici se može pristupiti preko poveznice https://alien-invasion-2a9d4.web.app/ te je smještena na Firebase-u. Delaney (2022) ga u svojem videozapisu opisuje kao Google-ov skup alata za izradu aplikacija i upravljanje njihovom infrastrukturom. Može ga se opisati i kao „softver kao usluga“ (eng. Software as a Service - SaaS). Sadrži servise za autentifikaciju, upravljanje bazama podataka, analitiku i praćenje napretka i prometa aplikacije. Uz to, ima i mogućnost za besplatno posluživanja web stranica i to je u ovom slučaju korišteno.

4.2 Struktura programskog koda

Sva logika implementirana je pomoću programskog jezika JavaScript i on je glavni alat korišten za izradu ove igrice. HTML i CSS služili su kao pomoćni alati za postavljanje početne pozicije platna, dizajniranje korisničkih sučelja i sličnog. Struktura igre čvrsto se zasniva na objektno orijentiranom pristupu. Temelj tog pristupa je korištenje klasa koje djeluju kao predlošci za kreiranje specifičnih objekata unutar igre. Vrlo često se koristi koncept nasljeđivanja gdje pojedina klasa nasljeđuje drugu i time poprima njezine metode i atribute. Tako se izbjegava ponavljanje programskog koda, a omogućuje dijeljenje zajedničkih karakteristika između različitih klasa. Ova objektno orijentirana paradigma pruža modularnost i skalabilnost igrice, a njezin cilj je da se pojedina klasa usredotoči na svoje specifične uloge. Tako je igrica podijeljena na mnoge komponente, odnosno klase gdje svaka od njih ima svoju ulogu. Mogu se izdvojiti sljedeće najbitnije klase:

- Igra
- Upravitelj platna
- Učitavač resursa
- Upravitelj unosa
- Detektor sudara
- Upravitelj težinom
- Upravitelj zvukom
- Generator valova protivnika
- Protivnik
- Igrač
- Entitet
- Oružje
- Val protivnika (eng. Enemy Wave)
- Postavke igre

Igra (eng. Game) je najveća i najbitnija klasa u programu. Ona je predložak za kreiranje instance nove igre. Jednom kada je objekt te klase napravljen, igra može započeti. Igra ima pristup i upravlja svim ostalim komponentama tako što njihove instance pohranjuje u svoje atribute. Ona sadrži i strukture podataka za pohranu svih protivnika, metaka i efekata te metode za zaustavljanje i ponovno pokretanje. U nastavku slijedi isječak koda koji prikazuje klasu igre sa njezinim atributima i najbitnijim metodama.

```
export default class Game {
    constructor() {
        this.settings = settings
        this.events = new EventEmmiter();
        this.canvasManager = new CanvasManager(this);
        this.assetLoader = new AssetLoader();
        this.crosshairManager = new CrosshairManager(this)
        this.inputManager = new InputManager(this);
        this.collisionDetector = new CollisionDetector(this)
        this.levelManager = new LevelManager(this)
        this.soundManager = new SoundManager(this);
        this.statsManager = new StatsManager(this);
        this.canvas = this.canvasManager.getCanvas('playerCanvas');
        this.player = new Player(this, 500);
        this.enemies = []
        this.playerBullets = []
        this.enemyBullets = []
        this.isPaused = false;
        this.ticksElapsed = 0;
        this.canvasManager.scaleEntities()
        this.play()
    }

    play() {
        if (this.isPaused || !this.loopId) {
            this.isPaused = false;
            pauseModal.style.display = 'none'
            this.loopId = setInterval(() => this.tick(),
this.settings.TICK_DURATION_MS);
            if (!this.statsManager.timer.isStarted()) {
```

```
                this.statsManager.startTimer();
            } else {
                this.statsManager.unpauseTimer();
            }
        }
    }

    pause() {
        if (!this.isPaused) {
            this.isPaused = true;
            clearInterval(this.loopId)
            this.loopId = null;
            pauseModal.style.display = 'flex'
            this.statsManager.pauseTimer();
        }
    }
}
```

Unutar konstruktora igre moguće je vidjeti kako igra stvara instance drugih klasa i pohranjuje ih kao svoje atribute. Time su sve komponente na jednom mjestu i lako ih je dohvatiti. Vrijedi primijetiti da se instanca igre prosljeđuje većini tih modula. Taj postupak se zove ubrizgavanje ovisnosti (eng. Dependency Injection). To je pristup u programiranju gdje se ovisnosti poput objekata unose u drugu komponentu umjesto da ih komponenta samostalno stvara. Ovaj postupak je nužan iz razloga što su pojedinoj komponenti potrebni podaci prisutni iz objekta igre. Primjerice komponenta koja upravlja platnom je zadužena za održavanje omjera platna i svih elemenata nacrtanih na njemu, stoga su joj potrebni ti elementi. U ovom slučaju to su neprijatelji trenutno vidljivi na ekranu, igrač i ostali efekti ukoliko ih ima. Igra sadrži sve te podatke i to je moguće vidjeti u donjem dijelu konstruktora. U nastavku slijedi opis ostalih komponenata i klasa.

4.3 Petlja igre

U teorijskom dijelu bilo je riječi o animacijama. Igra kao takva je u stvari jedna velika složena animacija s dodatkom da reagira na korisnički unos. Baš kao i u animacijama, za funkcioniranje igre mora postojati petlja ili metoda koja se konstantno poziva i pokreće igru. To se naziva petlja igre (eng. Game Loop). Petlja igre je ustvari srce igre, dok je metoda od koje se ona sastoji njen otkucaj (eng. Tick). U ovoj implementaciji igre ta metoda se baš tako

i zove, „tick“. Svaki puta kada se ona pozove, igra se pomiče za jedan korak unaprijed. To znači da se u svakom otkucaju te metode:

- Provjerava se postoji li novi nadolazeći val neprijatelja
- Generira se novi val neprijatelja i stvaraju novi neprijatelji
- Briše se platno od prethodnog otkucaja
- Igrač se pomiče na novu lokaciju ukoliko ju je zatražio unosom
- Odvija se logika za pomicanje neprijatelja
- Odvija se logika za aktivaciju neprijateljskih moći i pucanja
- Metci se pomiču za vrijednost njihove brzine
- Provjerava se postoje li sudaranja između metaka, neprijatelja i igrača
- Događa se obrada udaraca i primjenjuje se gubitak zdravlja
- Stvaraju i obrađuju se efekti i eksplozije
- Crtaju se svi elementi igre na novim lokacijama, uključujući protivnike, igrača, metke, eksplozije i ostalo
- Puštaju se zvukovi
- Računaju se statistike i prati igračev napredak

Ovo su samo neke od stvari koje se odvijaju u tijeku igre ukratko objašnjene. Postoji još dosta druge logike, ali sada je jasno kako otprilike igra funkcionira. Brzina otkucaja igre je 60 puta u sekundi, a implementirana je pomoću metode „setInterval“ koja je već spomenuta u teorijskom dijelu. Iako je preporučeno koristiti „requestAnimationFrame“, ovdje se koristi postavljanje konstantnog intervala kako bi se osiguralo da je igra uvijek radi jednakom brzinom, neovisno o brzini osvježavanja igračevog monitora.

Ipak, ne događaju se sve stvari unutar petlje igre. Primjerice pomicanje korisnikovog pokazivača, odnosno nišana događa se izvan nje. Razlog tomu je taj što nišan mora biti izuzetno brz i s najmanjim mogućim odzivom pokazivati na trenutnu lokaciju igračevog miša. Postoji još nekoliko stvari koje se odvijaju izvan petlje igre, o kojima će biti riječi kasnije. U programskom isječku slijedi metoda o kojoj se govori. Nalazi se u klasi „Game“ iz prethodnog poglavlja. Izostavljena je iz prošlog programskog koda zbog jednostavnosti.

```
tick() {
    if (this.isPaused) return

    this.levelManager.checkForUpcomingWaves();
```

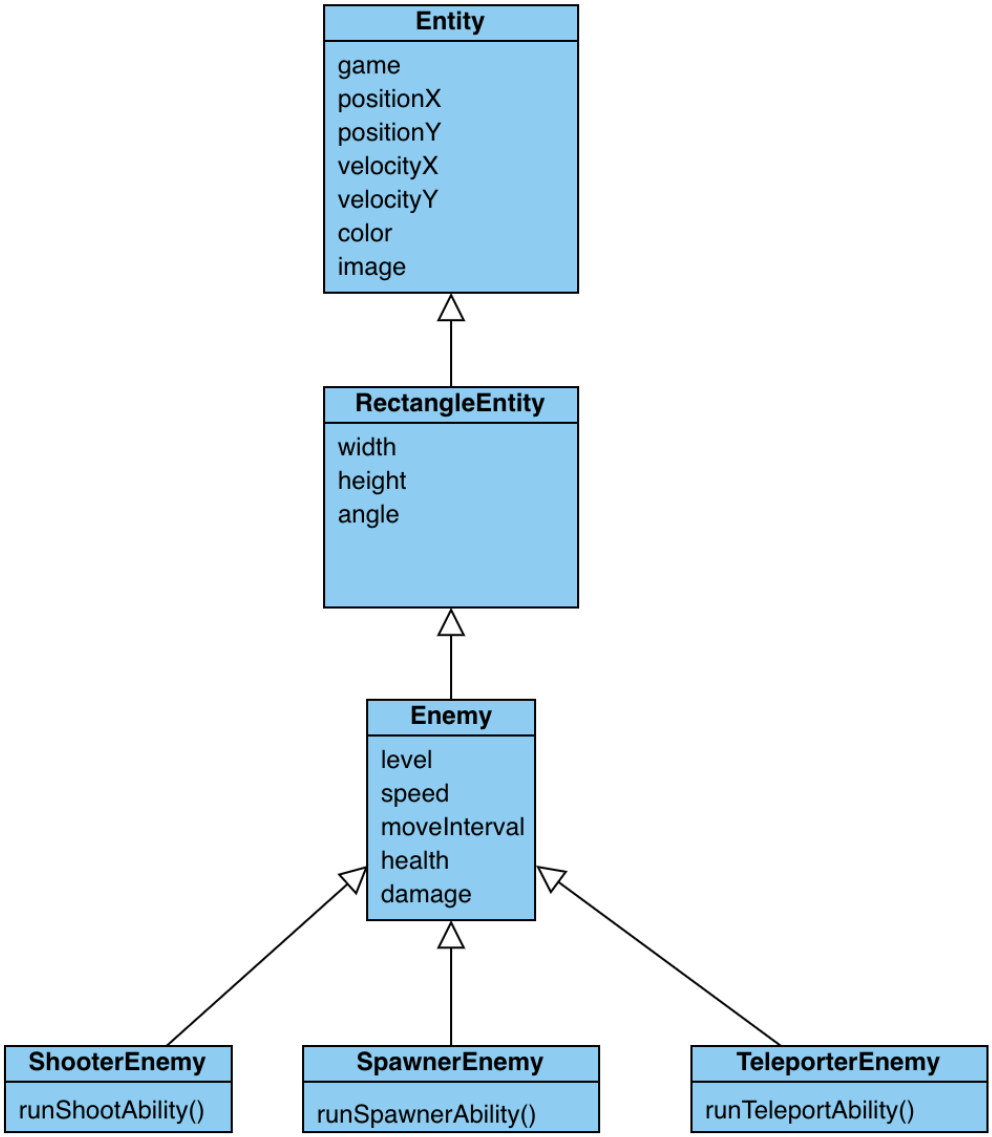
```
    this.canvasManager.clearCanvases()
    this.borderManager.drawBorders('playerCanvas')
    this.player.updatePosition();
    this.playerBullets.forEach(bullet => bullet.updatePosition());
    this.enemyBullets.forEach(bullet => bullet.updatePosition());
    this.enemies.forEach(enemy => enemy.move())
    this.enemies.forEach(enemy => {
        if (enemy.canShoot) enemy.runShootAbility();
    })
    this.runHitDetection()

    this.explosions.forEach(explosion => explosion.update())
    this.particleManagers.forEach(pm => pm.particles.forEach(particle => {
        particle.updatePosition();
    }));
    this.particles.forEach(particle => particle.update())

    this.drawGameElements();
    this.ticksElapsed++;
}
```

4.4 Neprijatelji

Neprijatelji su ključni elementi u igri. Oni su predstavljeni kao svemirci ili vanzemaljci koji se generiraju sa svih strana igračke površine. Postoji deset različitih tipova protivnika koji se razlikuju po izgledu, brzini, veličini, zdravlju i snazi, a neki od njih posjeduju i posebne sposobnosti. Svaki protivnik je definiran u posebnoj klasi s karakteristikama kao što su visina, širina, brzina kretanja, napadna snaga itd. Svaka od tih klasa nasljeđuje osnovnu klasu "Neprijatelj" (engl. Enemy), koja opet nasljeđuje klasu "Pravokutni Entitet" (engl. Rectangle Entity). Svi trenutno postojeći protivnici dijele zajedničku karakteristiku pravokutnog oblika. Klasa pravokutnika sadrži osnovne atribute pravokutnih oblika u dvodimenzionalnom prostoru, uključujući podatke s mogućim brzinama kretanja po osima. Osim toga, implementira i metodu za crtanje oblika na platnu, olakšavajući klasama koje je nasljeđuju brigu o tome. Klasa neprijatelj implementira dodatne metode za logiku kretanja protivnika i ponašanje nakon što su pogođeni ili uništeni. Svaka klasa specifičnog tipa protivnika dodatno dodaje kod za posebne sposobnosti, ukoliko ih imaju.



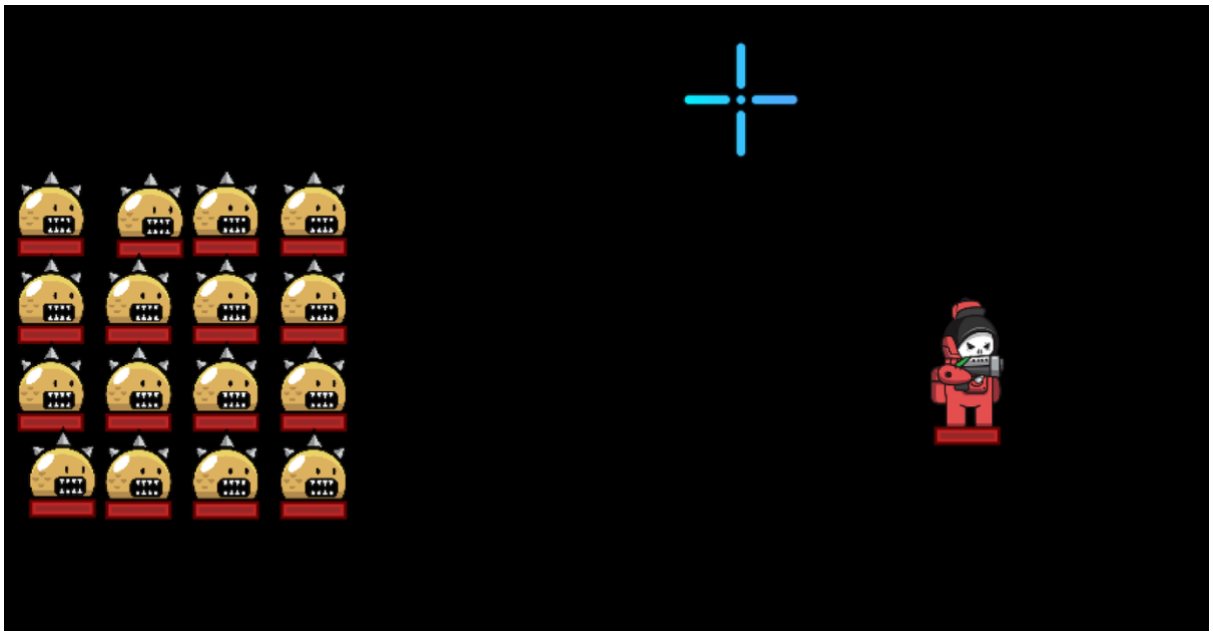
Slika 5. Pojednostavljeni dijagram klasa koji prikazuje strukturu nasljeđivanja i osnovne atribute

4.5 Generiranje valova neprijatelja

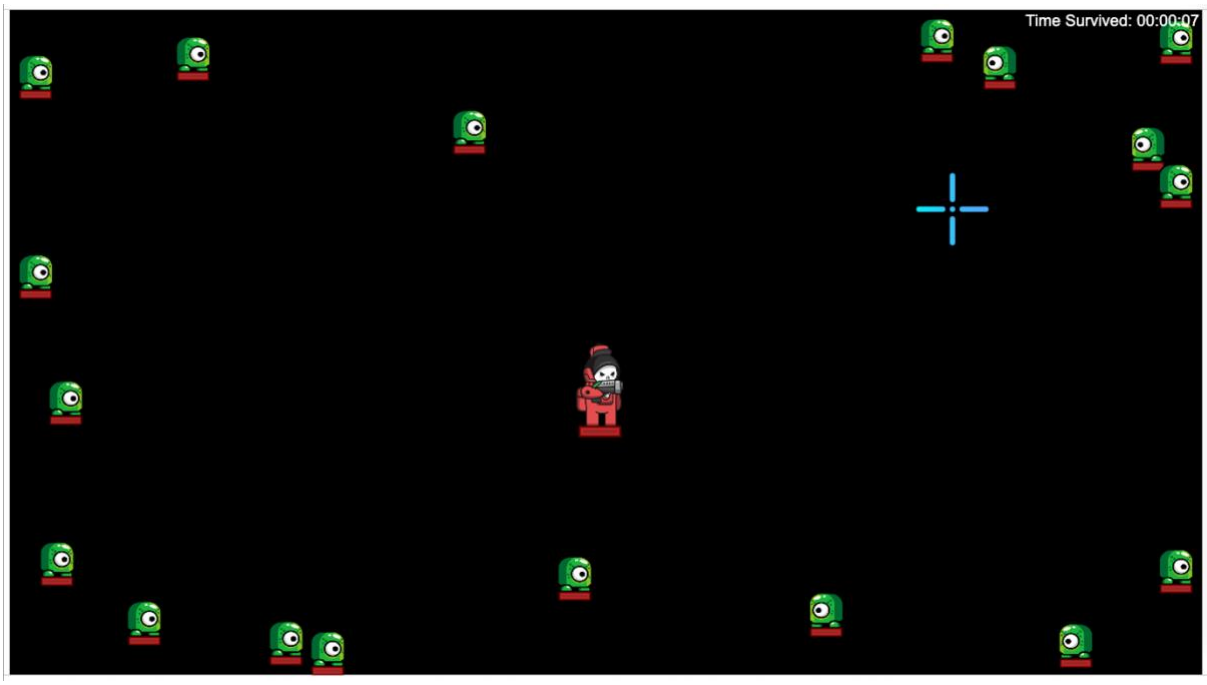
Generiranje valova neprijatelja bio je jedan od najsloženijih elemenata u razvoju igre. Ono pritom predstavlja ključan i najbitniji aspekt iste te izravno utječe na iskustvo i način igranja. Kako bi generiranje funkcioniralo, bilo je potrebno osmisliti i implementirati mnogo stvari kao što su mjesto stvaranja neprijatelja, veličinu vala, raspored, interval između njihovih pojava i same vrste neprijatelja. Prije nego što se započne s detaljnom razradom samog generiranja valova, bitno je pružiti dodatne informacije o samim valovima neprijatelja.

Val neprijatelja je ustvari skupina neprijatelja koji se stvaraju zajedno u određenoj formaciji. Postoje četiri različite vrste vala:

- Val u kutu
- Val u obliku kvadrata
- Nasumični val
- Val u liniji



Slika 6. Slika ekrana koja prikazuje val neprijatelja u obliku kvadrata



Slika 7. Slika ekrana sa nasumičnim valom neprijatelja

Jedan val može sadržavati samo jednu vrstu neprijatelja, odnosno može se sastojati od neprijatelja iste vrste. U samoj implementaciji, val je definiran kao klasa koja nosi sa sobom informacije o broju protivnika koji će biti stvoreni, vrsti tih protivnika, trenutku kad će generiranje vala započeti te vremenskom razmaku između stvaranja svakog pojedinačnog protivnika.

Unutar te strukture, definirane su metode za inicijalizaciju instanci protivnika i algoritmi za upravljanje njihovim redoslijedom stvaranja. Sama organizacija valova je modularna, gdje svaki tip vala predstavlja svoju posebnu klasu koja nasljeđuje osnovnu klasu vala. Ovaj pristup omogućava svakom tipu vala da dodatno razradi vlastiti algoritam raspoređivanja i stvaranja protivnika na igralištu. Na primjer, val smješten u kutu igrališta može stvarati protivnike koji se pojavljuju iz jednog specifičnog kuta i kreću se prema središtu platna. Sljedeći programski kod prikazuje dio implementacije tog vala sa metodom koja sadrži algoritam za stvaranje protivnika u gornjem lijevom kutu.

```
export default class CornerWave extends Wave {
    constructor(game, startSummoningTicks, waveSize, enemyType,
delayBetweenSummonsMs, summoningPosition) {
        super(game, startSummoningTicks, waveSize, enemyType,
delayBetweenSummonsMs, summoningPosition);
        this.borderWidth = this.game.getBorderManager().getLeftBorder();
```

```
        this.gapFromBorder = playerCanvas.width / 150
        this.summoningPosition = summoningPosition
        this.calculateDuration(this.getTotalEnemies());
    }


    createTopLeftWave() {
        let playerCanvas = this.game. getCanvas('playerCanvas');
        let [startX, startY] = this.getStartPosition(Corners.UPPER_LEFT);
        let gap = playerCanvas.width / 130;
        let numCols = this.calculateColumnsForEnemies(this.waveSize)
        let elementNumber = 1;

        for (let col = 0; col < numCols; col++) {
            for (let row = 0; row <= col && elementNumber <= this.waveSize;
row++) {
                let actualRow = row;
                let actualCol = col - row;
                this.spawnQueue.push({ x: startX + actualCol * (gap +
this.enemyWidth), y: startY + actualRow * (gap + this.enemyHeight) })
                elementNumber++;
            }
        }
    }
}
```

Sada kada je poznato značenje i funkcioniranje pojedinog vala. Moguće je objasniti proces generiranja valova. Valovi se generiraju dinamički tijekom igre pomoću klase „generator valova“. Jednom kada aktivni val završi, ili istekne njegovo dodijeljeno vrijeme, generator stvara novi. Ključna značajka generatora je da valove stvara na način da su sve teži i jači, kako bi igraču s vremenom bilo sve teže preživjeti. To se postiže na način da generator s vremenom bira sve jače protivnike, protivnici postaju jači i brži te se povećava njihov broj. Generator valova generira sve podatke potrebne za stvaranje instance vala. Ti podaci uključuju:

- Klasu tipa vala
- Klasu vrste protivnika
- Broj protivnika
- Vremenski razmak između stvaranja protivnika unutar jednog vala izražen u milisekundama

- Lokacija stvaranja vala. Ukoliko je odabrani tip val u kutu, ovaj podatak može biti jedan od kutova ekrana.

Sljedeća metoda prikazuje srž tog generiranja.

```
generateNextWave() {
        let waveType = this.pickWaveType();
        let enemyType = this.pickEnemy();
        let enemyAmount = this.getEnemySpawnAmount(waveType, enemyType);
        let summonDelay = this.getEnemySpawnDelay(waveType, enemyAmount);
        let summonPosition = this.getSummonPosition(waveType);

        return [waveType, enemyAmount, enemyType, summonDelay,
summonPosition]
}
```

Treba primijetiti da ona samo stvara podatke za sljedeći val, a instanca se kreira u drugoj klasi pod nazivom „Upravitelj nivoa“ (eng. Level Manager). Metoda za pripremu vala koja se nalazi u upravitelju nivoa koristi podatke generirane u prethodnoj metodi. Jedan od tih podataka je klasa odabranog vala te se ona u programskom kodu ispod izvlači u posebnu varijablu kako bi se u liniji ispod napravila instanca vala. Val se zatim dodaje u polje sa nizom valova. Na kraju metode, trenutak stvaranja sljedećeg vala se pomiče za trajanje novostvorenog vala.

```
prepareNextWave() {
        let generatedWaveData = this.waveGenerator.generateNextWave();
        let waveType = generatedWaveData.splice(0, 1)[0];
        let wave = new waveType(this.game, this.waveSpawnTick,
...generatedWaveData);
        this.waves.push(wave);
        this.waveSpawnTick += wave.tickDuration;
}
```

U pozadini ove implementacije postoje i mnogi konfiguracijski podaci koji djeluju na proces generiranja. Ti podaci uključuju brzinu povećanja težine valova, maksimalne količine stvaranja određene vrste protivnika, faktor slučajnosti pri odabiru protivnika, moguće lokacije stvaranja određenog vala te druge važne postavke.

4.6 Upravljanje korisničkim unosom

Korisnik sa igrom komunicira pomoću tipkovnice i miša. Tipkovnicom se može kretati dok miš služi za ciljanje i pucanje na protivnike. Za upravljanje korisničkim unosom zadužena je klasa pod nazivom upravitelj unosa (eng. Input Manager). Ona učitava kontrole igre u objekt gdje su ključevi moguće akcije koje korisnik može napraviti svojim unosom. Vrijednosti postavljene na svaki ključ su ime ili imena tipki koje je potrebno kliknuti za određenu akciju. Primjerice za kretanje prema gore definirane su tipka „w“ i tipka sa strelicom prema gore. Točnije, postavljeni su znakovni nizovi koji predstavljaju te fizičke tipke na tipkovnici („KeyboardEvent: code property“, 2023.).

Učitane kontrole se interpretiraju u slušateljima događaja (eng. Event Listeners) koji su postavljeni da osluškuju događaje pritiska i puštanja tipki na tipkovnici („Element: keydown event“, 2023.), („Element: keyup event“, 2023.). Slušatelj događaja provjerava pripada li kod pritisnute tipke onome definiranome u kontrolama igre. Ukoliko pripada, javlja igri koja tipka je pritisnuta postavljanjem zastavice. Nakon toga igra može na odgovarajući način reagirati na taj unos. Slijedi isječak programskoga koda o kojemu je riječ.

```
export default class InputManager {
    constructor(game) {
        this.game = game;
        this.pressedControls = {}
        this.loadDefaultControls()
        this.registerListeners()
    }

    loadDefaultControls() {
        // Učitavanje kontrola
        this.controls = {};
        this.controls[`${InputType.UP}`] = ['w', 'ArrowUp'];
        this.controls[`${InputType.RIGHT}`] = ['d', 'ArrowRight'];
        this.controls[`${InputType.DOWN}`] = ['s', 'ArrowDown'];
        this.controls[`${InputType.LEFT}`] = ['a', 'ArrowLeft'];
        this.controls[`${InputType.SHOOT}`] = [0]
        this.controls[`${InputType.TOGGLEPAUSE}`] = ['Escape']
    }

    registerListeners() {
```

```
        // Postavljanje slušatelja događaja
         window.addEventListener('keydown', (event) => {
             let control = this.#getPressedControl(event)
             if (!control) return
             if (this.pressedControls[control]) return;
             this.pressedControls[control] = true; // Postavljanje zastavice
da je tipka stisnuta
             this.game.events.emit('playerInput', control, true)
         })
         window.addEventListener('keyup', (event) => {
             let control = this.#getPressedControl(event)
             if (!control) return
             this.pressedControls[control] = false; // Postavljanje
zastavice da je tipka puštena
             this.game.events.emit('playerInput', control, false)
         })
    }
}
```

4.7 Upravljanje resursima

U kontekstu razvoja ove igrice, upravljanje resursima odnosi se prvenstveno na upravljanje sa slikama i zvukom. Sličice se koriste za prikaz karaktera samog igrača, svih neprijatelja, metaka i nišana u igrici. Kvaliteta i izgled sličica uvelike utječu na percepciju i doživljaj igrice. Kako bi se neprijatelji i ostali elementi mogli pravovremeno i ispravno prikazivati na ekranu, njihove sličice moraju biti učitane do trenutka njihovog pojavljivanja. One se učitavaju jednom na početku igre, i ostaju učitane tijekom cijele igre.

Slikama upravlja klasa pod nazivom „Učitavač resursa“ (eng. Asset Loader). Ona dohvaća slike iz strukture datoteka u projektu igrice, stvara objekte slika i učitava ih u memoriju. Time one postaju spremne za korištenje. Glavna metoda koja je za to zadužena je u sljedećem programskom isječku. Prima kao prvi argument tip resursa kojeg učitava koji je ujedno i naziv mape u kojoj se traženi resursi nalaze. Drugi argument koji prima su nazivi svih slika koje je potrebno učitati, zajedno za njihovim nastavkom.

```
loadImages(assetType, assetNames) {
         this[assetType] = {}
         assetNames.forEach(assetName => {
             let assetImage = new Image();
```

```
            assetImage.src = `./assets/${assetType}/${assetName}`;
            let assetNameBeforeExtension = assetName.split('.')[0]
            this[assetType][assetNameBeforeExtension] = assetImage;
        });
    }
}
```

4.8 Zvučni efekti

U svrhu poboljšanja doživljaja igre, dodani su i raznovrsni zvučni efekti. Na primjer, kada igrač koristi svoje lasersko oružje za pucanje, eliminira neprijatelja ili mišem pređe preko opcija u meniju, reproducira odgovarajući zvuk. Reproduciranje zvukova izuzetno je jednostavno, a sljedeća metoda to prikazuje.

```
playSound(soundName) {
    if (!this.soundEnabled) return;
    let fullPath = `./assets/sounds/${soundName}`
    let sound = new Audio(fullPath);
    sound.play();
}
```

Prikazana metoda nalazi se u klasi „upravitelj zvuka“ (eng. Sound Manager). U trenutku kada je potrebno pustiti zvuk, dio programskog koda može pozvati gornju metodu pri čemu mora proslijediti naziv zvuka. Na primjer zvuk za pucanje oružjem naziva se „blaster.wav“. Zvukovi se mogu uključiti i isključiti preko izbornika za pauzu.

5. Zaključak

Implementacija videoigre je vrlo složen postupak, čak i ako se radi o 2D igrici. Igrice su vrlo dinamičan sustav sa puno programske logike i gdje igrač direktno utječe na rezultat igre. Stoga je potrebno je misliti na svaku sitnicu i voditi računa da sve ispravno funkcionira,

radi glatko i brzo te pravilno reagira na igračev unos. Korištenje web tehnologija za razvoj igara ima dobrih i loših strana. Odlukom korištenja web tehnologija omogućuje se lakši pristup igrici te igrači ne trebaju instalirati igricu kako bi je mogli igrati. Sve što im je potrebno je web preglednik te pristup internetu. S druge strane, poznato je da postoji puno različitih web preglednika. Svaki od njih ima određene specifičnosti i drugačiji način rada te zbog toga treba obratiti posebnu pažnju da igra radi jednako dobro na različitim preglednicima. Neki preglednici jednostavno ne podržavaju određene tehnologije ili ugrađene metode pa je potrebno prilagoditi igricu tako da se osigura rad na svim preglednicima.

Smatram da je korištenje web tehnologija jedan od najboljih načina za razvoj 2D, ali i jednostavnijih 3D igara. Time se osigurava lakši pristup igrici sa bilo kojeg uređaja ili preglednika. Uz to, ne treba se misliti na operacijski sustav kojeg korisnik koristi te implementirati videoigru za različite uređaje. Internet se jako brzo širi te njegov napredak omogućuje sve sofisticiranije i složenije igre unutar web preglednika. Neprestano izlaze noviji i bolji alati koji se mogu koristiti te bi bilo šteta da ostanu neistraženi.

Literatura

A full overview of HTML Canvas (2019). freeCodeCamp. Preuzeto 22.07.2023. s https://www.freecodecamp.org/news/full-overview-of-the-html-canvas-6354216fba8d/

CanvasRenderingContext2D: arc() method (2023). MDN web docs. Preuzeto 05.08.2023. s https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/arc

CanvasRenderingContext2D: drawImage() method. (2023). MDN web docs Preuzeto 26.07.2023. s https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage

CanvasRenderingContext2D: save() method. (2023). MDN web docs. Preuzeto 26.07.2023. s https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/save

Collision detection (2023). Wikipedija. Preuzeto 16.08.2023. s https://en.wikipedia.org/wiki/Collision_detection

CSS: Cascading Style Sheets (2023). MDN web docs. Preuzeto 30.06.2023. s https://developer.mozilla.org/en-US/docs/Web/CSS

Delaney, J. (17.02.2022). Firebase in 100 seconds [Video file]. Preuzeto 21.08.2023. s https://www.youtube.com/watch?v=vAoB4VbhRzM&t

Drawing Shapes with canvas. (2023). MDN web docs. Preuzeto 23.07.2023 s https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_shapes

Element: keydown event (2023). MDN web docs. Preuzeto 17.08.2023 s https://developer.mozilla.org/en-US/docs/Web/API/Element/keydown_event

Element: keyup event (2023). MDN web docs. Preuzeto 17.08.2023. s https://developer.mozilla.org/en-US/docs/Web/API/Element/keyup_event

Firebase (2023). [Web aplikacija]. Preuzeto s https://firebase.google.com/

HTML: HyperText Markup Language (2023). MDN web docs. Preuzeto 29.06.2023. s https://developer.mozilla.org/en-US/docs/Web/HTML

HTML Canvas Graphics (bez dat.) W3Schools. Preuzeto 22.07.2023. s https://www.w3schools.com/html/html5_canvas.asp

HTML Canvas Drawing (bez dat.) W3Schools. Preueto 22.07.2023. s https://www.w3schools.com/graphics/canvas_drawing.asp

HTML Canvas Tag (bez dat.) Javatpoint. Preuzeto 22.07.2023. s https://www.javatpoint.com/html-canvas

HTML Introduction (bez dat.) W3Schools. Preuzeto 30.06.2023. s https://www.w3schools.com/html/html_intro.asp

HTML Tags (2021). Javatpoint. Preuzeto 22.07.2023. s https://www.javatpoint.com/html-tags
HTML: What is HTML – Definition and Meaning of Hypertext Markup Language (2021). freeCodeCamp. Preuzeto 22.07.2023. s https://www.freecodecamp.org/news/what-is-html-definition-and-meaning/

HTMLCanvasElement: getContext() method (2023). MDN web docs. Preuzeto 22.07.2023. s https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/getContext

JavaScript (2023). MDN web docs. Preuzeto 30.06.2023. s https://developer.mozilla.org/en-US/docs/Web/JavaScript

KeyboardEvent: code property (2023). Preuzeto 17.08.2023. s https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/code

Canvas Tutorial (2023). MDN web docs. Preuzeto 29.06.2023. s https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial

Transformations (2023). MDN web docs. Preuzeto 26.07.2023. s https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Transformations

WebGL: 2D and 3D graphics for the web. (2023). MDN web docs. Preuzeto 23.07.2023 s https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API#webgl_2

WebGPU API. (2023). MDN web docs. Preuzeto 23.07.2023 s https://developer.mozilla.org/en-US/docs/Web/API/WebGPU_API#browser_compatibility

WebGPU. (2023). Wikipedija. Preuzeto 23.07.2023. s https://en.wikipedia.org/wiki/WebGPU

Window: requestAnimationFrame() method (2023). MDN web docs. Preuzeto 05.08.2023. s https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

Popis slika