Univerzitet u Beogradu
Fakultet organizacionih nauka
Laboratorija za elektronsko poslovanje

RAZVOJ IGARA KORIŠĆENJEM TEHNOLOGIJA HTML5, CSS3 I JAVASCRIPT

Seminarski rad iz Elektronskog poslovanja

Nastavnik: Doc. dr Zorica Bogdanović

Saradnik: Konstantin Simić

Student: Miloš Stojanović 223/11

Beograd, 2014.

Sadržaj

1. UVOD

HTML5, kao najnovija verzija dobro poznate, standardne tehnologije za izradu veb stranica, je svojom pojavom privukao mnogo pažnje u svetu veb dizajna. Svedoci smo da, u savremenom internetu, sve više veb sajtova gubi tradicionalni oblik statičnih dokumenata sa tekstom, slikama i hipervezama. Veb sajtovi sve više postaju veoma moćne veb aplikacije, sa mogućnostima i karakteristikama klasičnih desktop aplikacija (obrada teksta, slika, videa). Novine koje donosi HTML5 predstavljaju nastavak tog trenda. Jedna od značajnih novina su šire mogućnosti kada su u pitanju multimedijalne i interaktivne veb lokacije. Recimo, HTML je sada u stanju da direktno reprodukuje video i audio sadržaj (novi <*audio*> i <*video*> elementi) bez potrebe za raznim dodacima kao što su Adobe Flash ili Microsoft Silverlight. Pored ovih vidljivih novina, HTML5 donosi mnogo poboljšanja "pod haubom" koja značajno optimizuju komunikaciju servera i veb čitača. [1]

U duhu promena koje je predstavio HTML5, i CSS je doneo mnogo poboljšanja i novina u verziji CSS3. Projektanti su podelili CSS3 u module koji grupišu određene funkcionalnosti. To je olakšalo postupak prilagođavanja proizvođača internet čitača i omogućilo je da se specifikacija treće verzije i dalje razvija, a da se određeni stabilni delovi već koriste, što ne bi bilo moguće da je u pitanju monolitna struktura specifikacije. Neki od modula su:

- Selectors
- Layout
- Media Queries
- Borders and Backgrounds
- Color

JavaScript je još jedna veoma popularna i moćna veb tehnologija. To je jednostavan, svestran, efektivan, skriptni jezik koji se koristi da se proširi funkcionalnost veb sajtova. Podržava i paradigmu objektno-orjentisanog programiranja (bazirano na prototipovima). Upotreba seže od kreiranja vizualnih efekata, pružanja većeg stepena interaktivnosti do procesiranja podataka sa veb stranica. JavaScript koristi DOM (Document Object Model) za prezentovanje i interakciju sa elementima i objektima HTML, XHTML dokumenata. DOM predstavlja uređenu hijerarhiju objekata kojoj se pristupa kroz odgovarajući API. Neke od prednosti JavaScript-a su: izvršava se na klijentskoj strani i time ne

opterećuje server, relativno je lak za korišćenje i učenje, široko je prihvaćen od strane proizvođača internet čitača. Jedna od mana koju kritičari navode je problem sigurnosti zbog činjenice da se kod izvršava na strani klijenta što otvara priliku za delovanje malicioznog softvera. [2]

Zahvaljujući ovim karakteristikama, navedene tehnologije se sve više koriste i za razvoj igrica, posebno u veb i mobilnom okruženju. U ovom radu će biti predstavljene neke mogućnosti i načini kako se u ovim tehnologijama može implementirati jednostavna igrica.

2. OPŠTA ANALIZA I OPIS TEHNOLOGIJA SA ASPEKTA RAZVOJA IGARA

U ovom poglavlju će biti prezentovane neke prednosti, opšte tehnike i elementi za razvoj igirca uz pomoć navedenih tehnologija.

2.1 Prednosti korišćenja tehnologija HTML5, CSS3 i JavaScript za razvoj igara

Koje su to prednosti korišćenja HTML5, CSS3 i JavaScript-a u odnosu na klasični, već dokazani native pristup?

PLATFORMSKA NEZAVISNOST

Jedna od prvih i najočiglednijih prednosti je činjenica da će igrice koje su razvijene u navedenim tehnologijama raditi na svim modernim uređajima, od računara do tableta i mobilnih telefona i raznih vrsta operativnih sistema koji se na njima koriste (uz vrlo malo dodatnih podešavanja). Imajući u vidu mnoštvo hardverskih i softverskih platformi koje se pojavljuju, jasno je da platformska nezavisnost omogućava veću fleksibilnost i brzinu u razvoju aplikacija kao i širi pristup tržištu korisnika. Kod native pristupa, potrebno je krajnji proizvoid prilagoditi za sve ciljne platforme što iziskuje dodatne troškove i napore.

UNIVERZALNA FRONT-END PLATFORMA

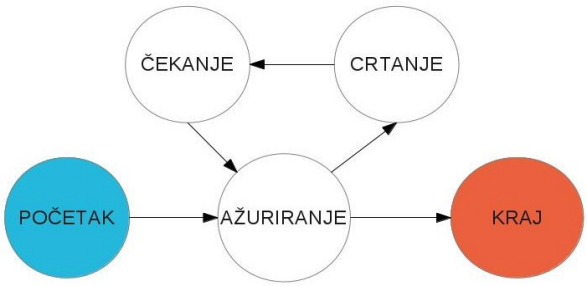
Ukoliko je potrebno da koristite neku serversku tehnologiju, bez obzira za koju se odlučite (PHP, Python, Ruby, ASP.NET) HTML5 i JavaScript mogu da implementiraju bogatu i svestranu klijentsku stranu aplikacije. Vaš front-end ostaje otporan na promene back-end tehnologija koje možete da menjate po potrebi.

BRŽI RAZVOJNI PROCES I DISTRIBUCIJA

Nema komplikovanih i dugih procesa kompajliranja i debagovanja, aplikacija je brže spremna za ažuriranje i distribuciju. [3]

2.2 Elementi i koncepti koji se koriste u razvoju igara uz pomoć HTML5, CSS3 i JavaScript-a

Jedna od glavnih komponenti kod programiranja igrica, bez obzira o kojoj tehnologiji se radi, je tkz. petlja igre *(game loop).* Petlja igre omogućava da se igra kontinuirano odvija i u trenucima kada nema inputa od strane igrača. Svaka igra sadrži određeni broj kako grafičkih, tako i logičkih elemenata, a ti elemnti mogu da menjaju svoja stanja i pozicije u zavisnosti od logike igre i reakcija igrača. Zadatak petlje je da u definisanom intervalu poziva odgovarajuću funkciju čiji je zadatak ažuriranje. Dakle, nakon samog početka igre, u jednom ciklusu izvršavanja, petlja poziva odgovarajuću funkciju (često se ova funkcija naziva *update function* ili *animation function*) koja ažurira stanje elemenata, menja njihovu poziciju, detektuje koliziju, upravlja zvukom, reaguje na inpute igrača (ukoliko postoje u datom trenutku), pokreće AI algoritme, i na kraju, na osnovu svega urađenog iscrtava korisniku novu grafičku prezentaciju stanja u kome se igra nalazi. Ta grafička prezentacija trenutnog stanja naziva se okvir *(frame)*. U petlji je potrebno definisati koliko puta će se funkcija ažuriranja pozvati u toku sekunde, i samim tim koliko okvira po jednoj sekundi će biti prikazano (*FPS – frames per second*). FPS bi trebao da bude veći od 15 da bi ljudski mozak stvorio sliku kontinuirane animacije prilikom igranja. FPS se obično definiše u rasponu od 30 do 60. Na samom kraju cilklusa petlja pauzira određeno vreme (u zavisnosti od vrednosti FPS-a) i ciklus se ponavlja sve dok se igranje ne završi. [4]



Slika 1: Petlja igre (game loop)

Sledeći važan element kao deo jezika HTML5 koji je, između ostalog, jako pogodan za izradu igrica je element *<canvas>. Canvas* predstavlja grafički kontejner pravougaonog oblika koji pruža mnoge mogućnosti, a neke od njih su:

- crtanje raznih oblika
- bojenje
- pravljenje gradijenata i paterna
- renderovanje teksta
- manipulacija pikselima

Sve ove mogućnosti ostvaruju se pomoću nekog skriptnog jezika (najčešće JavaScript-a) i upotrebom bogatog API-ja koji ovaj element obezbeđuje. Važno je napomenuti da *canvas* ne podržava koncept slojeva. Ako, recimo, nacrtate neku liniju zelene boje na površini i u nekom trenutku prefarbate tu liniju crvenom, ona se praktično gubi i ne možete da je vratite. Ako to uporedimo sa realnim svetom, *canvas* se u suštini ponaša kao obično slikarsko platno. Kao što je već rečeno, ovaj element, kroz svoj API, obezbeđuje metode za crtanje linija, eliptičnih i pravougaonih oblika, teksta, dodavanje slika, itd. Važna metoda koja se korsti u radu sa *<canvas>* elementom je *getContext()* metoda kojoj prosleđujemo parametar u vidu string-a *“2d”*. Ova metoda vraća ugrađeni HTML5 objekat sa mnogim atributima i funkcijama za crtanje i rad sa *canvas*-om u 2d formatu. *Canvas* u html fajlu kreiramo na sledeći način:

```
<canvas id=”myCanvas”>  </canvas>
```

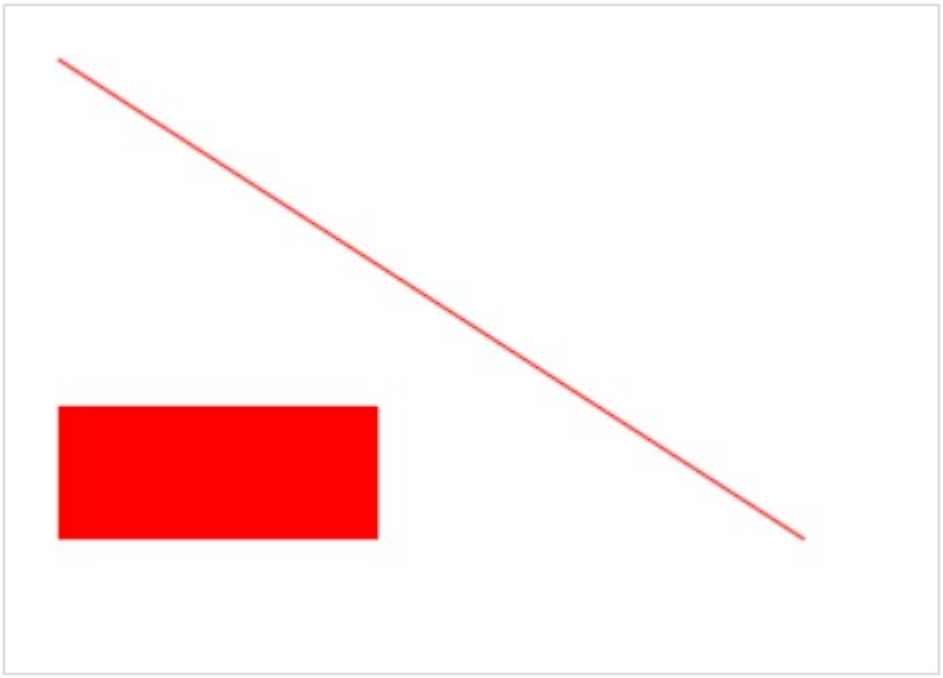
Zatim, potrebno je u js fajlu pronaći odgovarajući *canvas* i preuzeti context objekat u neku promenljivu koju ćemo koristiti za pozivanje metoda za crtanje:

```
var c = document.getElementById(“myCanvas”);
var ctx = c.getContext(“2d”);
```

Međutim, pre nego što počnemo sa crtanjem, potrebno je u uvesti koncept putanje (*path*) pri radu sa *canvas* elementom. Koncept putanje najlakše je objasniti ako se napravi analogija sa realnim svetom. Recimo da ispred sebe imate slikarsko platno i želite nešto da nacrtate. Verovatno ćete prvo uzeti običnu grafitnu olovku i napraviti skicu oblika. Tek onda ćete stvarno početi da slikate, a ta jedva vidljiva skica služiće vam kao putanja kojom ćete slikati dati oblik kistom i željenom bojom. Sličan princip se koristi i u radu sa *canvas* elementom. Prvo je potrebno da započnete putanju metodom *beginPath().* Zatim, pomoću određenih metoda možemo da pomeramo “vrh olovke” *(moveTo(x,y)* gde su *x* i *y* koordinate na *canvas* elementu*),* crtamo linije *(lineTo(x,y)),* kružne i eliptične oblike *(arc(x,y,radius,startAngle,stopAngle))* i druge. Zatim možemo da zatvorimo putanju metodom *closePath()*. Međutim, željeni oblik ćemo dobiti tek pozivom metode *stroke()* koja preko prethodno definisane putanje crta oblik koji ona opisuje. Pomoću atributa *strokeStyle, fillStyle,* i drugih, možemo da podešavamo boju, stil i ostale parametre crteža. Takođe, tu su i metode za crtanje pravougaonika (obojenih i praznih) *fillRect(), strokeRect(), rect()* i metoda *clearRect()* za brisanje definisane površine koje ne zahtevaju da se prethodno definiše putanja. Sve ove metode kao parametre primaju koordinate *canvas* elementa i eventualno neke dodatne parametre. Primer upotrebe nekih od navedenih metoda:

```
<!-- Primer 1. - rad sa canvas elemntom -->
<!DOCTYPE html>
<html>
	<body>
	<canvas id="myCanvas" width="350" height="250" style="border:1px solid #d3d3d3;">
	Your browser does not support the HTML5 canvas tag.</canvas>
		<script>
			var c=document.getElementById("myCanvas");
			var ctx=c.getContext("2d");

			ctx.beginPath();
```

```
                ctx.moveTo(20,20);
                ctx.lineTo(300,200);
                ctx.closePath();
                ctx.strokeStyle="#FF0000";
                ctx.stroke();

                ctx.fillStyle="#FF0000";
                ctx.fillRect(20,150,120,50);
            </script>
        </body>
</html>
```



Slika 2: Rezultat primera 1

3. SPECIFIKACIJA IGRE “ALIEN SHOOTER”

U ovom poglavlju biće predstavljenja specifikacija igrice “Alien Shooter” na čijem primeru će biti objašnjeni i predstavljeni osnovni koncepti razvoja igara u već navedenim tehnologijama.

“Alien Shooter” je jednostavna akciona igrica tipa *light gun shooter* u kojoj igrač treba da, koristeći miš računara, nišani i gađa vanzemaljce koji padaju sa vrha ekrana ka dnu. Vanzemaljaci padaju u formaciji od po 7 u svakom talasu. Da bi došao do kraja igrice, igrač mora da pogodi ukpuno preko 500 vanzemaljaca. Igra ima 7 nivoa težina. Pri svakom nivou, vanzemaljaci menjaju svoj oblik i kreću se sve brže. Na drugi nivo igrač prelazi nakon što pogodi 51. vanzemaljca, treći nakon 101., četvrti nakon 171., peti nakon 271., šesti nakon 381. i sedmi nakon 451. Igrač ima na raspolaganju 10 života, a gubi život svaki put kada jedan vanzemaljac padne na dno ekrana. Kao dodatnu mogućnost, igrač ima priliku da pogodi kutiju koja pada najviše dva puta u toku svakog nivoa, ne računajući prvi nivo. Kada igrač pogodi kutiju postoji mogućnost da dobije projektil ili da aktivira vremensku mašinu koja usporava vanzemaljace na nekoliko sekundi pružajući igraču mogućnost da ih više pogodi. Maksimalan broj projektila koje igrač može da ima je 3, i aktivira ih po potrebi, dok se vremenska mašina aktivira odmah pri pogotku kutije. Kada se projektil aktivira, svi trenutno vidljivi vanzemaljci na ekranu bivaju pogođeni. Verovatnoća da, pri pogotku kutije, igrač dobije projektil je 70% a vremensku mašinu 30%. Ukoliko igrač već ima 3 projektila, pri pogotku kutije sigurno dobija vremensku mašinu. Trenutak pojave kutije nije unapred određen i odigrava se primenom generatora slučajnih brojeva. U okviru glavnog ekrana za igru postoji i deo na kome je ispisan trenutni broj pogođenih vanzemaljaca, trenutni nivo, broj preostalih života i projektila. Igra treba da ima i odgovarajući glavni meni, prikaz kontrola, opis igrice i link ka izvornom kodu na GitHub repozitorijumu. Igrica treba da bude pokrivena odgovarajućim zvučnim efektima.

Dakle, potrebni elementi u igri su:

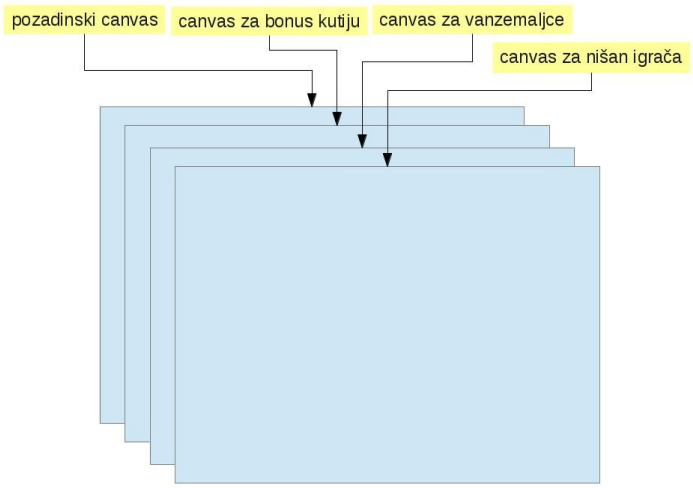
- grafički elementi za vanzemaljace, kutiju, nišan igrača, projektil, pozadina ekrana
- elementi menija
- zvučni elementi

4. IMPLEMENTACIJA IGRE “ALIEN SHOOTER”

Potpun izvorni kod dostupan je na adresi https://github.com/somi92/alien-shooter pod GNU GENERAL PUBLIC LICENSE licencom. U ovom poglavlju su definisani i objašnjeni samo ključni elementi razvoja ove igre.

4.1 Postavljanje igračkog okruženja

Prva stvar koju je potrebno implementirati je samo igračko okruženje, tj. površinu na kojoj se odvija radnja igrice. U tu svrhu koristi se *canvas* element. Važno je napomenuti da u ovoj igrici postoje četiri glavna grafička elementa: pozadina igračke površine, bonus kutija, vanzemaljci i nišan igrača. Za svaki od četiri elementa pridružujemo jedan *canvas* element istih dimenzija. Ti elementi će se preklapati jedan preko drugog u navedenom redosledu, s tim da će svi osim pozadine biti providni. Zašto je potrebno koristiti poseban *canvas* za svaki element igre? Pristup sa višestrukim *canvas* elementima može da donese značajna poboljšanja u performansama u slučaju kada je potrebno istovremeno crtati više elemenata i kada su velika preklapanja među elementima čije je ponašanje relativno nepovezano. Recimo, u slučaju ove igrice, ako bi nišan igrača i pozadina igračke površine bili na jednom *canvas*-u svaki put kada igrač nišaneći pomeri miš (što se praktično dešava neprestano tokom igre) potrebno je ponovo nacrtati nišan u novom položaju. A pošto *canvas* ne podržava više slojeva, to će da utiče i na pozadinsku sliku, što znači da bi i nju trebalo iscrtavati svaki put iako se ona u stvari ne menja. Kada bismo tome dodali i vanzemaljce i bonus kutiju, jasno da bi često dolazilo do sličnih preklapanja i neki elementi bi se u određenim trenucima morali bez potrebe ponovo crtati. Imajući u vidu da je operacija crtanja na *canvas* elementu relativno “skupa” što se tiče vremena i računarskih resursa, svakako je korisno smanjiti količinu crtanja na najmanji mogući nivo. Na taj način može se postići značajna optimizacija performansi igrice. [5]



Slika 3: Višeslojna igračka površina

Izvorni kod višeslojne pozadine u HTML-u (fajl *game.html*):

```
<canvas id="background" width="700" height="460"></canvas>
<canvas id="bonus" width="700" height="460"></canvas>
<canvas id="alien" width="700" height="460"></canvas>
<canvas id="shooter" width="700" height="460"></canvas>
```

Kod u CSS-u koji obezbeđuje pravilan raspored *canvas* elemenata (preko *z-index* atributa, fajl *style.css*):

```
#alien {
	z-index: -2;
	background: transparent;
}
#shooter {
	z-index: -1;
```

```
        background: transparent;
}
#background {
        z-index: -4;
}
#bonus {
        z-index: -3;
        background: transparent;
}
```

Takođe, potrebno je obezbediti odgovarajuće grafičke reprezentacije elemenata igre:

Slika 4: Grafički elementi igre

Slika 5: Pozadinska slika

U JavaScript fajlu *(game/game.js)* definišemo klasu GamePlay. U okviru ove klase biće definisan niz promenjivih i atributa klase koji će čuvati stanja i parametre igrice. Značenje atributa i promenjivih dato je u komentarima sledećeg isečka koda:

```
function GamePlay() {


    // atribut koji predstavlja petlju igre
    this.gameLoop;
    // promenjiva koja predstavlja nišan igraca
    var shooter;

    // canvas element za vanzemaljace i odgovarajuci context objekat
    this.canvasAlien;
    this.contextAlien;

    // canvas za nisan igraca i odgovarajuci context objekat
    this.canvasShooter;
    this.contextShooter;

    // canvas za bonus kutiju i odgovarajuci context objekat
    this.canvasBonus;
    this.contextBonus;

    // element koji prikazuje eksploziju projektila
    this.explosion;
    // promenjiva koja kontrolise duzinu eksplozije
    var explosionCounter;
    // da li je eksplozija u toku ili ne
    this.explosionSwitch;

    // canvas element za pozadinu i odgovarajuci context objekat
    var canvasBg;
    var contextBg;

    // element koji se prikazuje na kraju igre
    var finishScreen;
    // poruka koja se prikazuje na kraju
    var message;
    // slika koja se prikazuje na kraju
    var finishImage;
    // slika trenutne vrste vanzemaljaca (menja se u svakom nivou)
    var alienImage;

    // broj pogodjenih vanzemaljaca
    var aliensKilled;
    // broj preostalih zivota igraca
```

```
    var health;
    // trenutni nivo igrice
    var level;

    // prati pritiske na mis racunara
    this.mouseIsDown;
    // niz koji cuva vertikalne pozicije vanzemaljaca na ekranu
    var alien = [];

    // koordinate bonus kutije na canvasu
    var boxX;
    var boxY;

    // horizontalna i vertikalna koordinata nisana igraca na canvasu
    this.cX;
    this.cY;

    // niz koji cuva projektile (tj. njihovu graficku reprezentaciju)
    this.slots = [];
    // broj raspolozivih projektila (0-3)
    var missiles;
    // prati da li je projektil trenutno aktiviran
    this.missileActivated;

    // prati da li je bonus kutija trenutno omogucena
    this.bonusEnabled;
    // da li je bonus kutija u padu ili ne
    this.bonusFalling;
    // broj bonus kutija koji je pao u jednom nivou
    this.bonusInLevel;

    // prati da li je vremenska masina ukljucena ili ne
    this.timeMachineSwitch;
    // trajanje vremenske masine
    var timeMachineCounter;
    // kontrolise brzinu vremenske masine
    var timeReverse;
    // prati da li je igra pauzirana ili ne
    this.gamePaused;
```

Klasa sadrži i funkciju *setUpGame()* koja treba pa postavi odgovarajuće početne vrednosti za sve navedene atribute i promenjive (pogledati https://github.com/somi92/alien-shooter/blob/master/game/game.js).

U fajlu *game.html* definisani su elementi koji treba da igraču obezbede prikaz

trenutnog stanja igre, tj. broj pogođenih vanzemaljaca, živote, nivo igre i broj raspoloživih projektila (pogledati i fajl *style.css* u folderu *style*).

```
<div id="panel">
      <p class="labels">ALIENS KILLED: </p>
      <p id="score" class="labels"></p>
      <p class="labels">HEALTH: </p>
      <p id="health" class="labels"></p>
      <p class="labels">LEVEL: </p>
      <p id="level" class="labels"></p>
      <div id="missile">
            <p>Missiles:</p>
            <img id="slot3" src="images/none.png" />
            <img id="slot2" src="images/none.png" />
            <img id="slot1" src="images/none.png" />
      </div>
</div>
```

Prikaz panela na igračkoj površini:

Slika 6: Panel sa informacijama o tekućoj igri

4.2 Implementacija nišana igrača

Klasa *Shooter* implementira nišan igrača i njegove funkcionalnosti:

```
// klasa koje implementira nisan igraca
function Shooter() {

      // promenjive koje cuvaju koordinate nisana na canvasu
      var x;
      var y;

      // setter za koordinate
      this.setCoordinates = function(xCord, yCord) {
            x = xCord;
            y = yCord;
      };
```

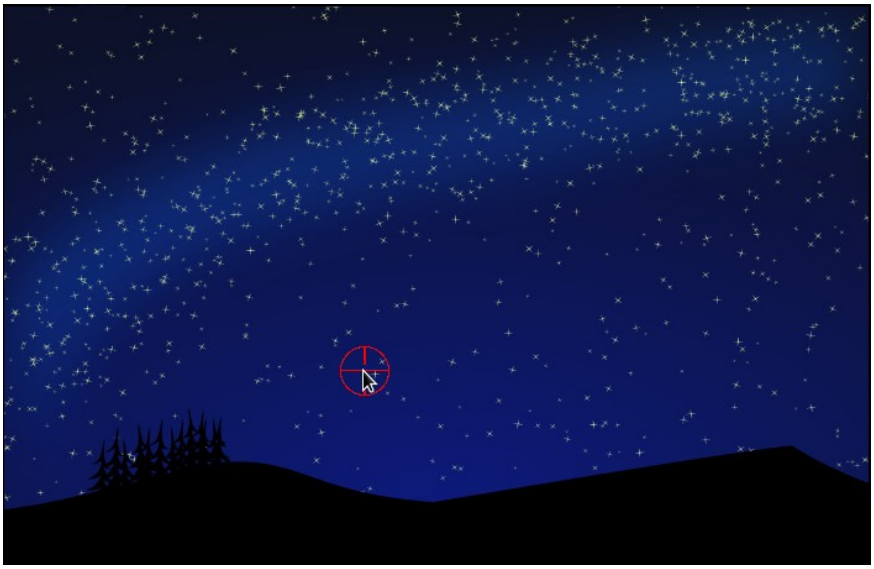
```
	// funkcija koja se poziva svaki put kad igrac pomeri mis (nisan)
	this.move = function(e) {

		if(!e) {
			var e = event;
		}

		if(game.isGamePaused()==false) {

			x = e.pageX - game.canvasShooter.offsetLeft;
			y = e.pageY - game.canvasShooter.offsetTop;

			game.contextShooter.clearRect(x-400,y-400,600,600);
			game.contextShooter.drawImage(imageStorage.scope, x-20,
y-20, 40, 40);
		}
	};

	// poziva se kada igrac pritisne dugme misa
	this.mouseDown = function(e) {
		var shotSound = new Audio("sounds/shot.ogg");
		shotSound.play();
		game.mouseIsDown = 1;

		if (!e)
		e = event;
	  game.cX = e.pageX - game.canvasShooter.offsetLeft;
	  game.cY = e.pageY - game.canvasShooter.offsetTop;
	};

	// poziva se kada igrac otpusti dugme misa
	this.mouseUp = function() {
		game.mouseIsDown = 0;

		if (!e)
		e = event;
	  game.cX = e.pageX - game.canvasShooter.offsetLeft;
	  game.cY = e.pageY - game.canvasShooter.offsetTop;
	};
}
```

Funkcija *move()* poziva se svaki put kada igrač pomeri miš i ažurira koordinate. If uslov koji proverava da li je igra pauzirana služi da se koordinate ne ažuriraju u trenutku pauziranja igre tako da bi nišan ostao na istom mestu dok se igra ne nastavi. Parametar *e* je *MouseEvent* objekat preko kojeg uzimamo koordinate miša *(e.pageX* i *e.pageY)*. Međutim, ti koordinati su u odnosu na celi HTML dokument pa je zato potrebno da od njih oduzmemo pomeraj *canvas*-a u odnosu na vrh ekrana i levu ivicu *(x = e.pageX – game.canvasShooter.offsetLeft;* i *y =*

e.pageY – game.canvasShooter.offsetTop;). Zatim, metodom *clearRect()* briše deo canvasa oko nišana i crta novi položaj metodom *drawImage()* koja kao parametar uzima sliku nišana iz objekta koji služi kao skladište slika kao i poziciju na kojoj će biti nacrtana. Brisanje nišana se vrši metodom "dirty rectangle" što znači da se briše samo površina u neposrednoj okolini trenutnog položaja. To dovodi do poboljšanja performansi jer nije potrebno brisati celokupnu površinu canvasa. Metode *mouseUp()* i *mouseDown()* funkcionišu na sličnom principu i ažuriraju atribute klase *GamePlay* koji predstavljau koordinate u kojima se nišan nalazi u trenutku kada igrač pritisne i otpusti dugme miša. Adekvatno se ažurira i atribut *mouseIsDown*. Važno je napomenuti da se u *setUpGame()* funkciji klase *GamePlay* dodaju "osluškivači" *(listeners)* koji omogućavaju samo praćenje pozicije miša i na osnovu svojih parametara pozivaju metode implementirane u klasi *Shooter*. Isečak koda iz *setUpGame* metode klase *GamePlay:*

```
this.canvasShooter.addEventListener("mousemove", shooter.move,false);

this.canvasShooter.addEventListener("mousedown",shooter.mouseDown,false);

this.canvasShooter.addEventListener("mouseup",shooter.mouseUp,false);
```

Prvi parametar predstavlja događaj koji se osluškuje, drugi je funkcija koja se poziva pri detektovanju događaja, a treći parametar je default false i nije bitan u ovom slučaju.



Slika 7: Igračko okruženje i nišan

4.3 Implementacija petlje igre i animacione funkcije

Igra se pokreće iz funkcije *init()* koja prvo poziva dve metode klase *GamePlay*, prvo *setUpGame()* a zatim *startTheGame()* koja implementira i pokreće petlju igre:

```
this.startTheGame = function() {

      this.gameLoop = setInterval(function(){game.animate();},30);
};
```

Atribut *gameLoop* klase *GamePlay* predstavlja identifikator tajmera kojeg dobijamo pozivom *setInterval()* funkcije koja je deo JavaScript API-ja. Funkcija kao prvi parametre dobija funkciju koju treba da poziva, a drugi parametar je vremenski period čekanja pre svakog poziva. Dakle, u ovom slučaju, petlja igre će pozivati funkciju *animate()* na svakih 30 milisekundi, sve dok se ne pozove metoda *clearInterval(gameLoop).* To znači da FPS igrice iznosi 33.333 (1000ms/33.333 = 30ms).

Najvažnija funkcija koja predstavlja suštinu igre je funkcija *animate()* klase *GamePlay*. Ova funkcija će biti objašnjena deo po deo. Za potpun prikaz funkcije posetiti https://github.com/somi92/alien-shooter/blob/master/game/game.js .

```
this.animate = function() {

        this.contextAlien.strokeStyle = "transparent";
        this.contextAlien.clearRect(0,0, this.canvasAlien.width,
                                    this.canvasAlien.height);

        this.contextBonus.strokeStyle = "transparent";
        this.contextBonus.clearRect(0,0, this.canvasAlien.width,
                                    this.canvasAlien.height);

        if(this.timeMachineSwitch==true && timeMachineCounter<=80) {
                  timeReverse = 3;
                  timeMachineCounter++;
        } else {
            this.timeMachineSwitch = false;
            if(this.explosionSwitch==false) {
                  timeReverse = 0;
            }
```

```
            timeMachineCounter = 0;
        }
```

U prvoj liniji funkcije, na *canvas*-u na kojem se crtaju vanzemaljci atribut *strokeStyle* se postavlja na *"transparent"* što znači da će svi oblici koji se crtaju biti providni. To je potrebno zato što će biti korišćene određene pomoćne linije i oblici koji ne treba da budu vidljivi igraču (objašnjeno u nastavku). U drugoj liniji, cela površina se briše kako bi se napravio prostor za iscrtavanje novog stanja igre. Treća i četvrta linija koda rade ekvivalentne operacije ali za *canvas* bonus kutije. Nakon toga, proverava se da li je uključena vremenska mašina i ako jeste postavlja se vrednost promenjive koja se kasnije koristi za usporavanje vanzemaljaca i inkrementira se brojač koji reguliše trajanje vremenske mašine. Ukoliko je uslov netačan, tj. vremenska mašina ne radi ili joj je isteklo trajanje, postavljaju se odgovarajuće vrednosti promenjivih. Potrebno je takođe proveriti i da li je u toku eksplozija projektila zato što se promenjiva *timeReverse* koristi i za usporavanje vanzemaljaca i u slučaju trajanja eksplozije.

```
for (i = 0; i < 7; i++) {

      var speed = Math.floor((Math.random()*level)+1);
      alien[i] += (speed-timeReverse);

      if (alien[i] >= this.canvasAlien.height - 50) {
            health--;
            var al = new Audio("sounds/aliens.ogg");
            al.play();
            var pos = Math.floor((Math.random()*100)+1);
            alien[i] = -pos;
            this.updateGame();
      }

      var y = alien[i];
      var x = (i+0.5) * 90;
      var radius = 33;
      this.contextAlien.beginPath();
      this.contextAlien.arc(x+35, y+35, radius, 0, 2 * Math.PI);
      this.contextAlien.drawImage(alienImage,x,y,70,70);
      this.contextAlien.closePath();
      var pos1 = Math.floor(10+(1+120-10)*Math.random());

      if ((this.contextAlien.isPointInPath(this.cX, this.cY) &&
                  this.mouseIsDown) || this.missileActivated == true) {
            alien[i] = -pos1;
            aliensKilled++;
            this.updateGame();
      }
```

```
        this.contextAlien.stroke();
 }
```

Implementacija vanzemaljaca koncipirana je tako da je u svakom trenutku postoji sedam elemenata koji predstavljaju vanzemaljce. Kada igrač pogodi vanzemaljca, beleži se rezultat i element se pomera na vrh ekrana i ponovo počinje da pada. Formira se for petlja koja se izvršava sedam puta (po jednom za svakog vanzemaljca). Prvo se određuje brzina kretanja vanzemaljca uz pomoć generatora slučajnih brojeva. Pored toga, brzina zavisi od nivoa na kojem se igrač nalazi. Generator slučajnih brojeva korsitiće se u određivanju brzine i pozicije vanzemaljaca kako bi se smanjilja uniformnost i predvidivost u kretanju. U narednoj if komandi proverava se da li je vanzemaljac dosegao dno ekrana. Ako jeste, igraču se oduzima život, dati vanzemaljac se vraća na vrh ekrana i poziva se funkcija *updateGame()* koja treba da ažurira podatke o igri i upravlja tranzicijom između nivoa.

Nakon toga, formira se putanja za svakog vanzemaljca. Koristi se funkcija *drawImage()* za crtanje sličice vanzemaljca. Potrebno je nacrtati krug oko svake sličice (funkcija *arc()*) kao pomoćnog oblik, koji će se korstiti da se proveri da li je igrač pogodio vanzemaljca. Ovaj krug je nevidljiv za igrača (zbog *strokeStyle* = *“transparent”*). U narednom if uslovu proverava se da li je igrač pogodio ili je ispaljen projektil. U skladu sa tim, ažurira se rezultat, poziva se funkcija *updateGame()*. Ovde se koristi funkcija iz *canvas* API-ja *isPointInPath()* koja za dobijene koordinate proverava da li se ta tačka nalazi u okviru putanje, u ovom slučaju, u okviru nevidljivog kruga koji je prethodno definisan.

```
if(this.explosionSwitch == true) {

      explosionCounter++;
      timeReverse = 3;
}

if(explosionCounter>50) {
      this.explosion.style.visibility="hidden";
      explosionCounter = 0;
      this.explosionSwitch = false;
      timeReverse = 0;
}

this.missileActivated = false;
```

Ukoliko je u toku eksplozija projektila, brojač eksplozije se inkrementira i postavlja se vrednost za usporavanje. Kada istekne brojač eksplozije, prikaz eksplozije se sklanja sa ekrana i promenjive se adekvatno ažuriraju.

```
if(this.bonusFalling==false) {

      if(this.bonusInLevel<2 && level>2) {

            var mysteryP = Math.floor((Math.random()*1000)+1);
            if(mysteryP>=998 && mysteryP<=1000 && level>1) {
                  this.bonusFalling = true;
                  this.bonusInLevel++;
                  boxX = (Math.random()*550)+100;
                  boxY = -20;
                  this.contextBonus.beginPath();
                  this.contextBonus.arc(boxX+35, boxY+35,33,0,2*Math.PI);
this.contextBonus.drawImage(imageStorage.mystery_box,boxX,boxY,70,70);
                        this.contextBonus.closePath();
                  }
            }

  } else {
      if (boxY >= this.canvasAlien.height - 50) {

            this.bonusFalling = false;
      } else {
            boxY += 5;
            this.contextBonus.beginPath();
            this.contextBonus.arc(boxX+35, boxY+35, 33, 0, 2 * Math.PI);

this.contextBonus.drawImage(imageStorage.mystery_box,boxX,boxY,70,70);
            this.contextBonus.closePath();
            if(boxY>50 && boxY<70) {
                  var box_sound = new Audio("sounds/box_falling.ogg");
                  box_sound.play();
            }
            if (this.contextBonus.isPointInPath(this.cX, this.cY) &&
this.mouseIsDown) {
                  this.bonusFalling = false;
                  this.openTheBox();
                 }
         }
  }
```

Ovaj deo koda zadužen je za upravljanje bonus kutijom. Prvi if uslov određuje da li bonus kutija već pada. Ukoliko to nije slučaj, ispituje se da li je već bilo kutija u trenutnom nivou. Na osnovu generatora slučajnih brojeva određuje se verovatnoća

pada kutije. Preostali postupak animacije padanja kutije je dosta sličan padanju vanzemaljaca.

```
if(KEY_STATUS[KEY_CODES[32]]==true && missiles>0 && missiles<4) {

      this.slots[-1+missiles].setAttribute("src","images/none.png");
      missiles--;
      var a1 = new Audio("sounds/explosion.ogg");
      a1.play();
      a1.play();
      KEY_STATUS[KEY_CODES[32]] = false;

      this.missileActivated = true;
      this.explosion.style.visibility="visible";
      this.explosionSwitch = true;
 }
```

Pomoću ovog koda igrač lansira projektil. U if uslovu se proverava da li je igrač pritisnuo SPACE tipku u da li je broj projektila u odgovarajućem rasponu. Ako jeste, slot se prazni, pušta se odgovarajući zvučni efekat i ažuriraju se odgovarajuće promenjive. Takođe se omogućava prikaz animacije eksplozije *(this.explasion.style.visibility="visible").* U okviru game.html fajla definisan je element sa id *explosion* koji služi za prikaz eksplozije.

```
if(health<=0 && this.explosionSwitch==false) {
      clearInterval(this.gameLoop);
      this.gameOver();

}

if(aliensKilled>500 && this.explosionSwitch==false) {
      clearInterval(this.gameLoop);
      this.gameFinished();
}
```

Poslednji deo funkcije *animate()* proverava da li je igrač završio igru. Prvi uslov odnosi se na gubitak zivota, a drugi na kraj igre kada igrač ostvari cilj, a to je preko 500 pogođenih vanzemaljaca.

Na kraju ovog podpoglavlja, biće objašnjena funkcija koja ažurira panel sa informacijama o igri. Ova funkcija se poziva pri svakom pogodtku ili gubitku života. Funkcija pristupa odgovarajućim elementima u panelu preko promenjive *score,* ažurira elemente i nizom if uslova upravlja tranzicijom između nivoa igrice. U okviru uslova se ažuriraju promenjive i menja se slika vanzemaljaca.

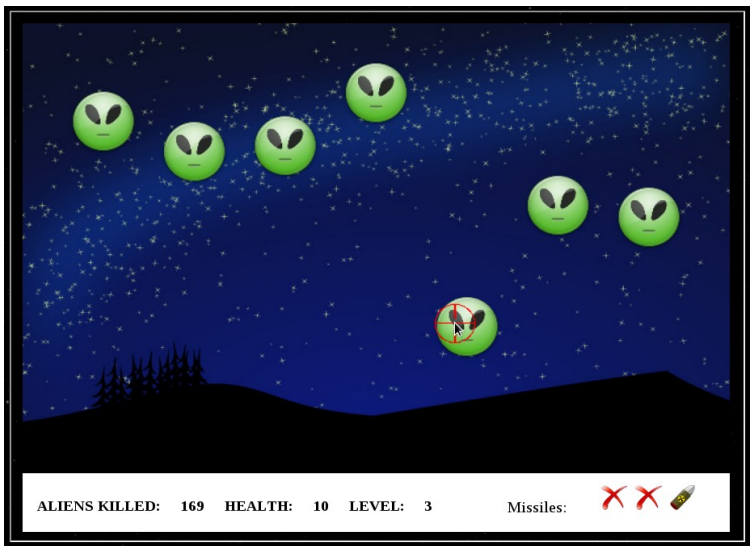
```
this.updateGame = function() {

            var score = document.getElementById('score');
            score.innerHTML=aliensKilled;

            var score = document.getElementById('health');
            score.innerHTML=health;

            var score = document.getElementById('level');
            score.innerHTML=level-1;

            if(aliensKilled>50 && aliensKilled<52) {
                  level = 3;
                  alienImage = imageStorage.ufo;
                  this.bonusInLevel = 0;
            }

            if(aliensKilled>100 && aliensKilled<102) {
                  level = 4;
                  alienImage = imageStorage.alien2;
                  this.bonusInLevel = 0;
            }

            if(aliensKilled>170 && aliensKilled<172) {
                  level = 5;
                  alienImage = imageStorage.alien3;
                  this.bonusInLevel = 0;
            }

            if(aliensKilled>270 && aliensKilled<272) {
                  level = 6;
                  alienImage = imageStorage.alien4;
                  this.bonusInLevel = 0;
            }

            if(aliensKilled>380 && aliensKilled<382) {
                  level = 7;
                  alienImage = imageStorage.alien5;
                  this.bonusInLevel = 0;
            }

            if(aliensKilled>450 && aliensKilled<452) {
                  level = 8;
```

```
                alienImage = imageStorage.alien6;
                this.bonusInLevel = 0;
            }
        };
```

Postoji još nekoliko dodatnih funkcija i elemenata u implementacije igre od kojih će neki biti navedeni:

- funkcija *openTheBox()* koja implementira pogodak bonus kutije, tj. određivanje bonusa koji igrač dobija
- funkcija *gameOver()* koja implementira kraj igre u situaciji u kojoj igrač izgubi sve živote, prikazuju advekvatnu poruku pristupajući odgovarajućim HTML elementima u *game.html*
- funkcija *gameFinished()* koja implementira sličnu funkcionalnost kao i prethodna funkcija, ali u slučaju da igrač "pređe" igricu
- funkcije za osluškivanje pritiska dugmeta
- objekat *imageStorage* koji omogućava da se sve slike učitaju pre početka igre
- elementi sa id-jem *screen* u fajlu game.html koji omogućavaju prikaz odgovarajućih poruka na kraju igre
- implementacija menija u fajlu *menu.html*

Za potpun prikaz implementacije igre posetiti *https://github.com/somi92/alien-shooter* gde su dostupni svi fajlovi. Pogledati fajl *style.css* u folderu *style.*



Slika 8: Igra u toku

5. ANALIZA DODATNIH MOGUĆNOSTI

U ovom radu korisćen je pristup samostalne izrade i implementacije svih elemenata i temeljnih koncepata razvoja igara u cilju da se objasni njihova suština i da se predstave osnove. Međutim, razmatranje ove teme svakako ne bi bilo potpuno bez osvrta na neka već gotova rešenja za razvoj igara u tehnologijama HTML5, CSS3 i JavaScript. Na internetu postoji mnoštvo radnih okvira *(framework)* i tkz. g*ame engine-a* koji implementiraju razne funkcije potrebne za razvoj igara. Ova rešenja pružaju API koji omogućava programerima da na najbolji i najlakši način iskoriste sve mogućnosti framework-a. Neke od prednosti korišćenja ovakvih gotovih rešenja su: brži, lakši i efikasniji razvojni proces (naročito kod većih i kompleksnijih projekata), koristi se rešenje koje već testirano i provereno, dostupnost dokumentacije i podrške od strane zajednice korisnika, kod je obično lakši za održavanje i izmenu. Jedna od najbitnijih prednosti, ako ne i najbitnija, je ta što su ova rešenja uglavnom jako dobro optimizovana što se tiče upotrebe računarskih resursa, što je dosta bitna karakteristika imajući u vidu da neke igre mogu biti mnogo zahtevne. Naravno, postoje i neke mane: framework se generički ponaša i ograničena je mogućnost njegove modifikacije pa možda neće biti u mogućnosti da zadovolji neke potrebe, programer može da se previše fokusira na framework i izostavi iz vide druge aspekte razvoja. [6]

Framework	Cena	Poslednje izdanje	Kratki opis
Construct 2	zavisi od opcija	12.11.2013.	Razvojno okruženje (game maker). Nema potrebe za pisanjem JavaScript koda. Pruža mnoštvo opcija.
ImpactJS	99$	28.8.2013.	Dobro se održava, ima veliku dokumentaciju i zajednicu. Radi na svim platformama.
Three.js	besplatan	14.1.2014.	Najpopularniji game engine za razvoj 3d/WebGL igirca.
EaselJS	besplatan	24.1.2014.	Omogućava lakše upravljanje elementima na canvas-u. Raspolaže sa dosta alata. Inspirisan Flash-om.

Tabela 1: Pregled nekih popularnih framework i game engine rešenja

6. ZAKLJUČAK

U ovom radu pružen je uvod u osnove razvoja igara primenom tehnologija HTML5, CSS3 i JavaScript. Imajući u vidu sve veći rast i širenje mobilnih platformi i Web-a kao univerzalne platforme za razne aplikacije, ne iznenađuje sve veća privlačnost ovih tehnologija. Na internetu postoji mnoštvo servisa i tehnologija koje omogućavaju HTML5 i JavaScript igricama integraciju za razne platforme (Android, iOS), povezivanje sa društvenim mrežama, distribuciju, analitiku, servise za plaćanje i reklamiranje. Jedan od najpopularnijih servisa te vrste je Clay.io koji pruža i još dodatnih opcija. Možete, na primer, veoma lako distribuirati svoju igru na Facebook platformu, Windows 8, Chrome Web Store, Firefox Marketplace. S obzirom na sve navedeno, realno je očekivati dalji rast i širenje ovih tehnologija u domenu razvoja igara za mobilne uređaje i Web.

7. LITERATURA

[1] Lawson B., Sharp R., *Uvod u HTML5 za programere*, Mikro knjiga, 2012.

[2] Flanagan D., *JavaScript: sveobuhvatni vodič*, Mikro knjiga, 2008.

[3] *Getting Started With HTML5 Game Development*, Hallock A., https://hacks.mozilla.org/2013/09/getting-started-with-html5-game-development/ (poslednja poseta 3.2.2014.)

[4] *Game Development Using JavaScript*, Raiten S., http://www.codeproject.com/Articles/563425/Game-Development-Using-JavaScript (poslednja poseta 3.2.2014.)

[5] *Galaxian Style HTML5 Game*, Lambert S., http://blog.sklambert.com/galaxian-html5-game/ (poslednja poseta 2.2.2014.)

[6] *HTML5 Game Engines*, http://html5gameengine.com/ (poslednja poseta 4.4.2014.)

[7] *Let's call it a draw(ing surface)*, http://diveintohtml5.info/canvas.html (poslednja poseta 1.2.2014.)