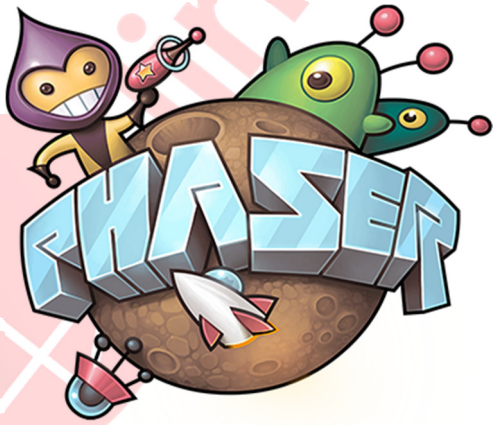
Phaser – integracija, resursi, crtanje i fizika

U jednom od prethodnih modula imali ste prilike da vidite kako izgleda proces kreiranja jedne 2D igre koja se izvršava unutar web pregledača, kada se za razvoj koristi Vanilla JavaScript. Neki od najznačajnijih zadataka koje smo morali samostalno da obavimo bili su kreiranje glavne petlje, realizacija vremenski bazirane animacije, rukovanje spritesheetovima, obrada korisničke interakcije, učitavanje resursa i detekcija završetka učitavanja. Prilikom realizacije manje kompleksnih igara može imati smisla da se sve nabrojane operacije realizuju samostalno. Ipak, kreiranje kompleksnijih igara korišćenjem Vanilla JS-a ubrzo može postati vrlo komplikovan posao. Stoga se u pomoć može pozvati neki od softverskih okvira koji su specijalno namenjeni tome da proces kreiranja igara učine bržim i jednostavnijim.

Modul koji je pred vama u potpunosti će biti posvećen jednom softverskom okviru koji je primarno namenjen razvoju igara koje se izvršavaju unutar web pregledača, preciznije, unutar canvas elementa. Reč je o softverskom okviru (engl. *framework*) koji se zove Phaser.

Šta je Phaser?

Phaser je softverski okvir za razvoj igara koje se izvršavaju unutar cavnas elementa web pregledača. Reč je o potpuno besplatnom softverskom okviru otvorenog koda, koji je moguće koristiti za ličnu ili komercijalnu upotrebu, bez ikakvih naknada.



Slika 14.1. Phaser (izvor: https://phaser.io/)

Phaser je namenjen razvoju 2D igara, a kreirala ga je kompanija Photon Storm. Za crtanje grafike unutar canvas elementa Phaser u pozadini može da koristi Canvas API, sa kojim smo se već upoznali, ali i WebGL skup funkcionalnosti, kada tako nešto podržava web pregledač unutar koga se igra izvršava. Odabir odgovarajućeg sistema za crtanje obavlja se potpuno autonomno, što na kraju igrama kreiranim korišćenjem Phasera omogućava da maksimalno iskoriste sve pogodnosti hardvera klijentskog uređaja. Naravno, Phaser poseduje uniformni aplikativni programski interfejs koji u potpunosti apstrahuje proces crtanja grafike unutar canvasa, bez obzira na skup funkcionalnosti koji se koristi u pozadini.

Pored skupa funkcionalnosti za crtanje, sastavni delovi Phasera su i razni drugi sistemi koji su neizostavni deo procesa razvoja web igara:

- **sistem za učitavanje resursa** (engl. *preloader*) – reč je o sistemu koji će nam u nastavku pomoći da definišemo sve slike i zvučne zapise koje će naša igra koristiti i koji će za nas automatski obaviti njihovo učitavanje pre nego što započne izvršavanje igre,
- **nekoliko sistema za simuliranje fizike** (engl. *physics systems*) – Phaser poseduje tri sistema za simuliranje fizike, čime se u igre koje kreiramo može dodati logika koja simulira ponašanje objekata u prirodi,
- **sistem za rad sa spritesheetovima** – reč je o sistemu koji omogućava da se veoma lako rukuje spritesheetovima i sprajtovima, koji čine osnovu 2D igara,
- **sistem za rad sa česticama** (engl. *particles*) – sistem koji olakšava kreiranje efekata kao što su eksplozija, kiša ili vatra,
- **Phaser Pointer sistem za obradu korisničke interakcije** – Phaser olakšava obradu korisničke interakcije, bez obzira na vrstu uređaja koji se koristi za kontrolisanje igre (miš, tastatura, džojstik, displej osetljiv na dodir),
- **sistem za rad sa zvukom** – sistem koji olakšava kreiranje, reprodukciju i kontrolu zvučnih zapisa.

Uspostavljanje okruženja za razvoj Phaser igara

Pre nego što budemo u mogućnosti da Phaser iskoristimo za kreiranje igara, neophodno je da njega integrišemo u tekući projekat. To podrazumeva uključivanje jednog `.js` fajla unutar HTML dokumenta. Takav fajl je moguće preuzeti sa sledeće adrese:

```
phaser.min.js
```

Preuzeti fajl je u HTML dokument moguće uključiti na sledeći način:

```
<script src="js/phaser.min.js"></script>
```

Prikazana linija podrazumeva da se fajl `phaser.min.js` nalazi unutar foldera `js` na korenoj putanji projekta.

Phaser je u HTML moguće integrisati i korišćenjem fajlova hostovanih na CDN serveru:

```
      <script
src="//cdn.jsdelivr.net/npm/phaser@3.24.1/dist/phaser.js"></script>

      <script
src="//cdn.jsdelivr.net/npm/phaser@3.24.1/dist/phaser.min.js"></script>
```

Prva linija ilustruje uključivanje razvojne, a druga uključivanje produkcione verzije Phasera.

Napomena

S obzirom na to da je sastavni deo Phasera sistem za učitavanje resursa, baš kao i prilikom rada sa PreloadJS bibliotekom, Phaser projekte je neophodno hostovati na HTTP serveru, bilo udaljenom ili lokalnom. Preporuka je korišćenje lokalnog Apache servera, koji je sastavni deo XAMPP paketa.

Kompletan HTML dokument u koji je uključen Phaser izgleda ovako:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Phaser Coin Collector</title>
</head>

<body>
    <div id="game-container"></div>
    <script src="js/phaser.min.js"></script>
    <script> </script>
</body>

</html>
```

Odmah možete da primetite da unutar HTML dokumenta ne postoji `canvas` element, za razliku od primera iz prethodnih modula. Jednostavno, Phaser obavlja dinamičko kreiranje takvog elementa, pa nema potrebe za njegovim fizičkim smeštanjem unutar HTML strukture.

Unutar `script` elementa postavili smo jedan `div` element sa `id-jem game-container`, unutar koga će biti postavljen naš `canvas` element. Takođe, HTML dokument poseduje i jedan `script` element koji je trenutno prazan, a unutar koga ćemo u nastavku mi pisati kod za kreiranje igre.

Igra Coin Collector

U nastavku ovoga modula sa Phaserom ćemo se upoznati na primeru kreiranja igre *Coin Collector*. Biće to igra koja će imati osobine primera koji je prikazan na kraju prethodnog (CreateJS) modula. Igrač će moći da upravlja glavnim karakterom igre, koji će moći da se kreće levo i desno i da obavlja skok. Igrač će skupljati novčiće, a ukoliko ne stigne da neki od novčića uhvati pre nego što on dodirne tlo, doći će da kraja igre. Sve to će izgledati kao na videu 14.1.

https://youtu.be/i-wCbMg6u3I

Video 14.1. Igra Coin Collector

S obzirom na to da ćemo odmah preći na praktičan razvoj igre, neophodno je odmah na početku uspostaviti projektnu strukturu sa svim neophodnim resursima:

- **img**
    - background.png
    - coin.png
    - ground.png
    - runner.png
- **js**
    - phaser.min.js
- **sound**
    - collect.mp3
    - fail.mp3
- index.html

Kompletan kostur projekta možete da preuzmete sa sledećeg linka:

`coin_collector_skeleton.rar`

Konfigurisanje igre

Prvi korak u razvoju Phaser igre biće definisanje osnovnih konfiguracionih parametara:

```
let config = {
    type: Phaser.AUTO,
    parent: 'game-container',
    width: 1280,
    height: 720,
    scene: {
        preload: preload,
        create: create,
        update: update
    }
};

var game = new Phaser.Game(config);

function preload() {}
function create() {}
function update() {}
```

U prikazanom kodu prvo je obavljeno kreiranje jednog konfiguracionog objekta, čija je referenca smeštena unutar promenljive `config`. Takav objekat je iskorišćen za prosleđivanje konstruktoru klase `Phaser.Game`.

Prostor imena Phaser i klasa Game

Funkcionalnosti Phaser softverskog okvira smeštene su unutar prostora imena sa nazivom `Phaser`. Klasa `Game` pripada ovom prostoru imena, a koristi se za kreiranje instance koja predstavlja kompletnu Phaser igru. Instanca `Game` klase rukovodi procesom učitavanja, parsiranja konfiguracionih vrednosti, kreiranja sistema za crtanje i pokretanje ostalih Phaser sistema zaduženih za upravljanje zvukom i za obradu korisničke interakcije. Nakon svih navedenih koraka koji se obavljaju prilikom učitavanja igre, klasa `Game` je zadužena za pokretanje glavne petlje.

Konstruktor `Game` klase izgleda ovako:

```
Game( [GameConfig])
```

GameConfig je objekat za konfigurisanje `Game` instance i on može imati sledeća najznačajnija svojstva:

- `type` – tip sistema za crtanje koji će biti korišćen; moguće vrednosti su sledeće:
    - `Phaser.CANVAS` – za crtanje se koristi Canvas API,
    - `Phaser.WEBGL` – za crtanje se koristi WebGL,
    - `Phaser.AUTO` – koristi se WebGL ukoliko je podržan od strane web pregledača; u protivnom se koristi Canvas API.
- `width` – širina `canvas` elementa u pikselima,
- `height` – visina `canvas` elementa u pikselima,
- `scene` – svojstvo kojim se definiše scena,
- `parent` – referenca na DOM element ili `id` vrednost nekog HTML elementa unutar koga je potrebno smestiti `canvas` element; bez ovoga svojstva `canvas` se podrazumevano postavlja na kraj `body` elementa,
- `canvas` – referenca na DOM element koji predstavlja `canvas`, ukoliko želimo da Phaser koristi neki postojeći `canvas` element,
- `canvasStyle` – CSS stilizacija koja će biti primenjena nad `canvas` elementom umesto podrazumevane Phaser stilizacije.

Prikaz grafike Phaser igre apstrahuje se korišćenjem pojma scene (engl. *Scene*). Mi smo upravo prikazanim kodom obavili kreiranje jedne scene tako što smo svojstvu `scene`, objekta za konfigurisanje, dodelili referencu na jedan objekat.

Scena

Scena je osnovni Phaser pojam u čijem se kontekstu prikazuje animirana grafika igara. Upravo prikazanim kodom obavili smo kreiranje jedne scene koja je definisana sa tri svojstva. Reč je o svojstvima koja upućuju na tri metode čija su tela trenutno prazna:

- `preload()` – metoda unutar koje je potrebno definisati kod za učitavanje resursa koji će biti korišćeni od strane igre,
- `create()` – metoda unutar koje se kreiraju objekti igre, odnosno oni objekti koje želimo da prikažemo unutar scene,
- `update()` – metoda koja se poziva po jednom tokom svake iteracije glavne petlje, sve dok je jedna scena aktivna; koristi se za ažuriranje osobina elemenata igre; može da prihvati dva ulazna parametra:
    - `time` – trenutno vreme, odnosno vreme pozivanja trenutne iteracije petlje,
    - `delta` – vreme u milisekundama koje je proteklo od prethodne iteracije glavne petlje.

Tri upravo opisane metode scene trenutno su prazne, tako da je naš sledeći korak da definišemo njihovu logiku. Počećemo od metode koja na osnovu redosleda pozivanja ima prioritet. Reč je o metodi za učitavanje resursa.

Responsive canvas

Podrazumevano, `canvas` koji Phaser kreira poseduje dimenzije koje su u prethodnim redovima definisane svojstvima `width` i `height`. To su dimenzije koje Phaser na `canvas` element postavlja korišćenjem istoimenih HTML atributa. Iz prethodnih lekcija znamo da se na taj način definiše prostor za crtanje, ali takođe znamo i to da `canvas` element podrazumevano ne poseduje responsive osobine.

Prilikom rada sa Phaserom ne preporučuje se direktno stilizovanje `canvas` elementa koji Phaser kreira. Stoga je za definisanje ponašanja koja se odnose na veličinu `canvas` elementa potrebno koristiti `ScaleManager` sistem, koji ovaj softverski okvir poseduje:

```
scale: {
    mode: Phaser.Scale.FIT,
    parent: 'game-container',
    width: 1280,
    height: 720
}
```

Ovo je primer korišćenja takvog sistema, odnosno kod kojim se Phaseru govori da je potrebno da `canvas` element prati promenu veličine roditeljskog elementa. To je postignuto definisanjem vrednosti svojstva **mode**. Ono može imati vrednosti nekoliko predefinisanih konstanti:

- `Phaser.Scale.FIT` – širina i visina `canvas` elementa automatski se računaju na osnovu prostora unutar roditeljskog elementa; pri tome se uvek čuva izvorni odnos stranica i kompletan `canvas` element uvek je u potpunosti vidljiv unutar web pregledača,

- `Phaser.Scale.HEIGHT_CONTROLS_WIDTH` – širina se automatski prilagođava dostupnoj visini unutar roditelja, pa se može dogoditi da deo `canvas` elementa bude izvan vidnog polja,

- `Phaser.Scale.WIDTH_CONTROLS_HEIGHT` – visina se automatski prilagođava dostupnoj širini, pa se može dogoditi da deo `canvas` elementa bude izvan vidnog polja,
- `Phaser.Scale.RESIZE` – `canvas` uvek zauzima kompletan prostor i po visini i po širini koji je dostupan unutar roditelja, pa tako može doći do deformacije odnosa stanica.

`ScaleManager` dimenzijama `canvas` elementa rukuje se isključivo korišćenjem CSS svojstava `width` i `height`, dok veličina prostora za crtanje uvek ostaje nepromenjena.

S obzirom na to da `ScaleManager` prilagođava `canvas` roditeljskom elementu, dobra praksa je da se na takvom elementu definiše maksimalna i eventualno minimalna veličina:

```
#game-container {
    max-width: 1280px;
    margin: 0 auto;
}
```

Učitavanje resursa

Sve resurse koje će naša igra koristiti potrebno je definisati unutar `preload()` metode, objekta scene. Na taj način će Phaser znati koji su sve resursi neophodni za funkcionisanje igre i automatski će obaviti njihovo učitavanje, pre nego što scenu prikaže unutar `canvas` elementa i obavi aktiviranje glavne petlje.

Logika za definisanje resursa naše igre izgledaće ovako:

```
function preload() {
    this.load.image('background', 'img/background.png');
    this.load.image('ground', 'img/ground.png');
    this.load.image('coin', 'img/coin.png');
    this.load.spritesheet('runner',
        'img/runner.png', {
            frameWidth: 100,
            frameHeight: 130
        }
    );

    this.load.audio('fail', 'sound/fail.mp3');
    this.load.audio('collect', 'sound/collect.mp3');
}
```

Unutar metode `preload()` dodata je logika za učitavanje svih resursa koji će biti korišćeni u nastavku. Praktično, to su svi fajlovi koji se nalaze unutar foldera `img` i `sound`. Možete videti da se za učitavanje koristi svojstvo `load`, objekta scene i tri metode, u zavisnosti od tipa resursa:

- `image()` – za učitavanje slika,
- `spritesheet()` – za učitavanje slika koje predstavljaju spritesheetove, pri čemu se definišu širina i visina pojedinačnog sprajta,
- `audio()` – za učitavanje zvučnih zapisa.

Prvi parametar svake od navedenih metoda jeste ključ (engl. *key*) pod kojim će resurs biti dostupan ostatku logike.

Crtanje statičke grafike pozadine

Uskoro ćete videti da Phaser omogućava kreiranje brojnih specijalizovanih objekata koji se mogu koristiti tokom razvoja igre, odnosno objekata koji poseduju vrlo napredne osobine. Ipak, prvi grafički objekat koji ćemo mi nacrtati u ovoj lekciji biće u potpunosti statičan, što znači da neće koristiti nikakve specijalne Phaser sisteme. Reč je o pozadini igre koja neće biti animirana niti će učestvovati u bilo kakvoj interakciji sa igračem. Stoga ćemo nju unutar Phaser scene nacrtati kao običnu sliku.

Crtanje slike je moguće obaviti na sledeći način:

```
function create() {
      this.add.image(640, 360, 'background');
}
```

Unutar `create()` metode postavljena je jedna naredba. Nju ćemo sada raščlaniti na delove:

- `this` – referenca na objekat scene,
- `add` – svojstvo koje omogućava pristup `GameObjectFactory` objektu, koji će nam na kraju obezbediti funkcionalnost za kreiranje `GameObject` objekta,
- `image()` – metoda koja obavlja kreiranje objekta igre na osnovu reference na prosleđenu sliku.

Metoda `image()` prihvata tri parametra, pri čemu se prva dva odnose na koordinate centra slike, a poslednji na ključ pod kojim je slika nešto ranije, unutar metode `preload()` registrovana.

Koordinate za pozicioniranje objekata relativne su njihovom centru

Prilikom rada sa Phaserom sve koordinate za pozicioniranje objekata relativne su centru takvih objekata. Upravo se zbog toga za definisanje koordinata u upravo prikazanom primeru uzima polovina širine i visine koje slika izvorno poseduje.

Kao svoju povratnu vrednost, metoda `image()` emituje objekat igre koji se prikazuje unutar scene.

Objekti igre (Game Objects)

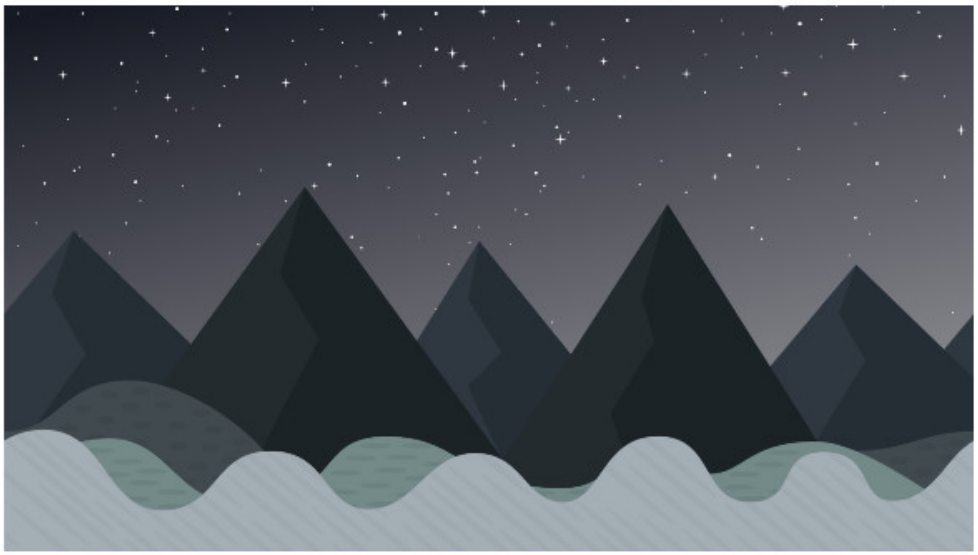
Vizuelni elementi koji se prikazuju unutar jedne scene u kontekstu Phaser okruženja nazivaju se objekti igre, odnosno Game Objects.

Phaser poznaje dosta različitih vrsta objekata igre, ali svi oni imaju zajedničku, baznu klasu **Game Objects**. Reč je o klasi koja se ne koristi direktno. Direktno se koriste neke od mnogobrojnih konkretnih klasa koje Phaser poznaje:

- `Image` – za prikaz neanimiranih slika unutar scene,
- `Sprite` – za prikaz animiranih slika; osnovna razlika u odnosu na klasu `Image` odnosi se na mogućnosti animiranja, kontrolisanja od strane korisnika i definisanja fizike,
- `Graphics` – za prikaz geometrijskih oblika, kao što su linija, pravougaonik, krug...,
- `Text` – za prikaz teksta,
- `Container` – za grupisanje većeg broja nekih drugih objekata igre u jednu celinu
- ...

Iako se mogu kreirati direktnim instanciranjem svojih klasa, veoma česta praksa za kreiranje objekata igre jeste korišćenje objekta tipa **GameObjectFactory**. Reč je o objektu koji je sastavni deo svake scene i koji omogućava lako kreiranje objekata igre koji se automatski registruju unutar scene koja ih je kreirala. To je i pristup koji je upotrebljen u prethodnoj naredbi.

Prikazanom naredbom unutar HTML dokumenta dobićemo prikaz kao na slici 14.2.



Slika 14.2. Prikaz statičke pozadine unutar Phaser scene

Na slici 14.2. možete da primetite da unutar upravo nacrtane pozadine ne postoji podloga po kojoj je predviđeno da se kreće glavni junak. Mi smo je ovoga puta sa razlogom uklonili iz pozadine zato što će prilikom kreiranja Phaser igre takva podloga biti jedan poseban objekat igre.

Dodavanje podloge i aktiviranje fizike

Kao što je upravo rečeno, podloga po kojoj će trčati glavni junak naše igre biće zasebni element igre zato što Phaser omogućava da se on definiše kao deo sistema fizike. Na taj način će Phaser sliku koja predstavlja podlogu prepoznati kao objekat po kome će se kretati naš glavni karakter, bez potrebe da mi ručno simuliramo njegovu poziciju na podlozi. O čemu se zapravo radi, imaćete prilike da vidite u narednim redovima.

Objekat koji će predstavljati podlogu naše igre kreiraćemo na sledeći način:

```
let ground = this.physics.add.staticImage(640, 665, 'ground');
```

I ovoga puta koristimo objekat scene koji je dostupan preko ključne reči `this`, ali sada upotrebljavamo svojstvo **physics**. Reč je o svojstvu koje nam omogućava pristup `ArcadePhysics` objektu, što je osnovni tip unutar koga su objedinjene funkcionalnosti za rad sa arkadnom fizikom.

Fizika igre

Na početku priče o Phaseru rečeno je da su njegov sastavni deo i nekoliko sistema fizike. U svetu razvoja igara sistemi fizike omogućavaju simulaciju zakona koji važe u prirodi. Phaser poseduje tri različita sistema za simulaciju zakona fizike, a mi ćemo prilikom razvoja igre *Coin Collector* koristiti najjednostavniji, koji se zove *Arcade Physics*.

Arkadni sistem fizike omogućava kreiranje dve osnovne vrste objekata:

- Physics Image – slika koja ne može biti animirana,
- Physics Sprite – slika koja može biti animirana.

Objekti unutar arkadnog sistema fizike mogu se podeliti na dve grupe:

- statički (`StaticBody`) objekti – objekti koji se ne pomeraju i na koje ne deluju fizičke sile, kao što je gravitacija; statički objekti ne mogu imati brzinu niti ubrzanje; drugi objekti se odbijaju od njih, pri čemu se oni ne pomeraju,
- dinamički (`DynamicBody`) objekti – objekti koji se mogu kretati posredstvom fizičkih sila, kao što je gravitacija; ovakvi objekti mogu imati brzinu i ubrzanje; takođe, mogu su sudarati i odbijati od drugih objekata, u čemu učestvuje njihova masa.

Uskoro ćete moći da vidite da su tip objekta i tip tela takvog objekta dva različita pojma. To praktično znači da je moguće kreirati `Image` objekat koji može biti statički ili dinamički. Po istom principu i `Sprite` objekat može biti statički ili dinamički.

Kako bi se upravo opisani sistem fizike aktivirao, neophodno je dodati sledeći odeljak koda unutar objekta za konfigurisanje igre:

```
physics: {
    default: 'arcade',
    arcade: {
        gravity: {
            y: 300
        },
        debug: false
    }
}
```

Prikazani odeljak koda je neophodan kako bi nešto ranije prikazana naredba mogla da funkcioniše.

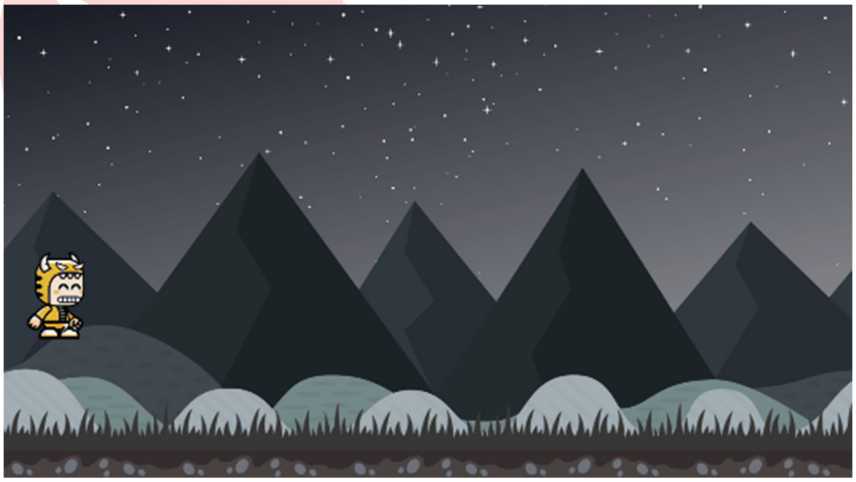
Korišćenjem metode `staticImage()` iz prethodne naredbe dobija se jedan statički `Image` objekat. To su upravo i osobine koje mi želimo za podlogu naše igre. Podloga neće moći da se pomera kako bi glavni junak mogao da trči po njoj.

Kreiranje glavnoj junaka

Nakon kreiranja statičke pozadine, biće kreiran i jedan dinamički objekat koji će učestvovati u Phaser sistemu arkadne fizike. Reč je o glavnom junaku naše igre:

```
let runner = this.physics.add.sprite(100, 450, 'runner');
```

Na ovaj način obavlja se kreiranje `Sprite` objekta sa dinamičkim telom. To znači da će na našeg glavnog junaka uticati fizičke sile. Da je to stvarno tako, možete da vidite na animaciji 14.1, koja ilustruje prikaz unutar web pregledača, a njega možete da dobijete ukoliko sada pokrenete igru.



Animacija 14.1. Posredstvom sile gravitacije, naš glavni junak je u slobodnom padu

Kao što možete da vidite, naš glavni junak slobodno pada, s obzirom na to da na njega deluje sila gravitacije. Loša stvar je što se naš glavni junak ne zaustavlja, pa tako izlazi iz okvira canvas elementa.

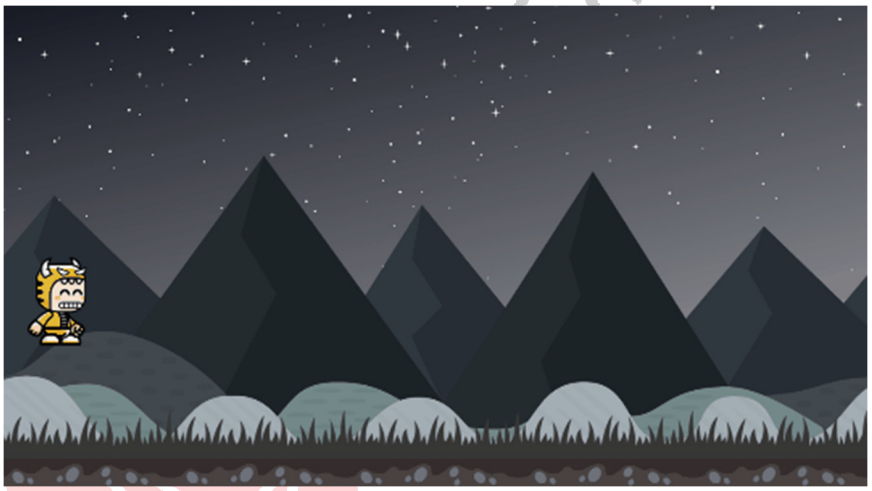
Dobijeni efekat se može promeniti na sledeći način:

```
runner.setBounce(0.2);
runner.setCollideWorldBounds(true);
```

Na ovaj način su definisane dve dodatne osobine `Sprite` objekta, koji predstavlja našeg glavnog junaka:

- metodom `setBounce()` definisan je koeficijent odbijanja kada dođe do kontakta našeg glavnog junaka i nekog drugog objekta,
- metodom `setCollideWorldBounds()`, kojoj je prosleđena vrednost `true`, definisano je da naš glavni junak neće moći da izađe iz okvira scene.

Ukoliko u ovom trenutku pokrenete igru unutar web pregledača, moći ćete da vidite da glavni junak na izlazi iz okvira canvas elementa (animacija 14.2).



Animacija 14.2. Glavni junak ne može da izađe iz okvira canvas elementa

Ipak, još uvek nismo dobili ono što želimo, odnosno to da se naš glavni junak zaustavi na podlozi za trčanje. Za obavljanje takvog posla neophodno je iskoristiti još jednu od funkcionalnosti Phaser arkadnog sistema fizike.

Detekcija kolizije

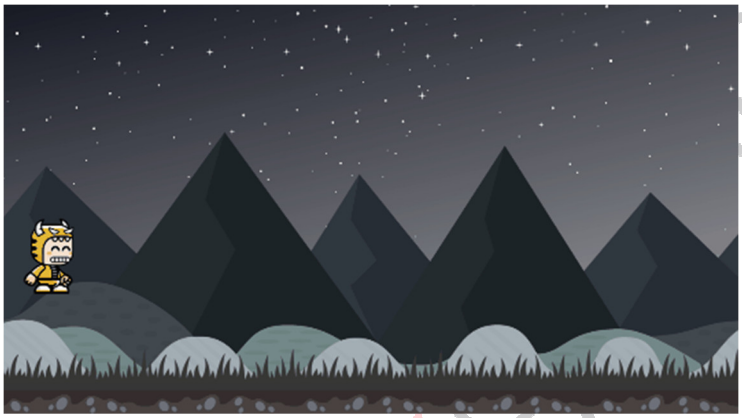
Kako bismo učinili da se naš glavni junak zaustavi u kontaktu sa podlogom igre, biće potrebno iskoristiti jednu posebnu metodu `Arcade.Factory` klase:

```
collider(object1, object2)
```

Metoda `collider()` koristi se kako bi se Phaser arkadnom sistemu fizike reklo da je potrebno da prati dva objekta i da utvrđuje da li je došlo do kolizije između njih. Ovu metodu mi ćemo upotrebiti na sledeći način:

```
this.physics.add.collider(runner, ground);
```

Prikazana naredba je dovoljna kako bi se aktivirao sistem za praćenje kolizije između dva objekta koja učestvuju u sistemu fizike. To praktično znači da će se naš glavni junak zaustaviti u kontaktu sa podlogom (animacija 14.3).



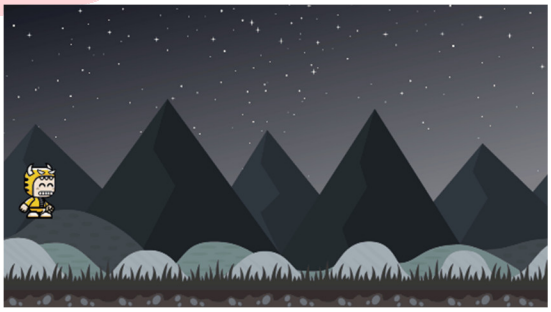
Animacija 14.3. Glavni karakter se zaustavlja prilikom kontakta sa podlogom

U animaciji 14.3. možete da vidite da se naš glavni junak sada zaustavlja, ali i da imamo novi problem. Naime, s obzirom na to da naša podloga poseduje i travu, potrebno je da naš glavni junak kroz nju trči. Trenutno, on trči po površini trave.

Kako bi se ovakav problem rešio, neophodno je izvršiti uticaj na osobine `DynamicBody` objekta, koji predstavlja telo koje učestvuje u sistemu fizike. To je moguće obaviti na sledeći način:

```
ground.body.setOffset(0, 60);
```

`DynamicBody` objektu pristupa se korišćenjem svojstva `body`, a zatim se korišćenjem metode `setOffset()` vrši pomeranje imaginarnih okvira u kome se nalazi telo objekta. Na ovaj način se region tela objekta spušta za `60px`, pa ćemo na kraju dobiti efekat koji smo želeli (animacija 14.4).



Animacija 14.4. Glavni junak se sada zaustavlja na predviđenom delu podloge

Pitanje

Osnovni sistem fizike unutar Phasera naziva se:

a) Arcade Physics
b) Matter Physics
c) Game Objects
d) Photon Storm

Objašnjenje

U svetu razvoja igara sistemi fizike omogućavaju simulaciju zakona koji važe u prirodi, a osnovni takav sistem u svetu Phaser razvoja zove se Arcade Physics.

Rezime

- Phaser je softverski okvir za razvoj igara koje se izvršavaju unutar `canvas` elementa web pregledača.
- Phaser je namenjen razvoju 2D igara, a razvila ga je kompanija Photon Storm.
- Za crtanje grafike unutar `canvas` elementa Phaser može da koristi Canvas API, ali i WebGL skup funkcionalnosti.
- Pored skupa funkcionalnosti za crtanje, sastavni delovi Phasera su i sistem za učitavanje resursa, nekoliko sistema za simuliranje fizike, sistem za rad sa spritesheetovima, sistem za rad sa česticama, Phaser Pointer sistem za obradu korisničke interakcije, sistem za rad sa zvukom...
- Funkcionalnosti Phaser softverskog okvira smeštene su unutar prostora imena sa nazivom `Phaser`.
- Klasa `Game` koristi se za kreiranje instance koja predstavlja kompletnu Phaser igru.
- `GameConfig` je objekat za konfigurisanje `Game` instance.
- Scena je osnovni Phaser pojam u čijem se kontekstu prikazuje animirana grafika igara.
- `preload()` je metoda unutar koje je potrebno definisati kod za učitavanje resursa koji će biti korišćeni od strane igre.
- `create()` je metoda unutar koje se kreiraju objekti igre, odnosno oni objekti koje želimo da prikažemo unutar scene.
- `update()` je metoda koja se poziva po jednom tokom svake iteracije glavne petlje.
- Prilikom rada sa Phaserom koordinate za pozicioniranje objekata relativne su njihovom centru.
- Vizuelni elementi koji se prikazuju unutar jedne scene u kontekstu Phaser okruženja nazivaju se objekti igre, odnosno *Game Objects*.
- U svetu razvoja igara sistemi fizike omogućavaju simulaciju zakona koji važe u prirodi, a osnovni takav sistem u svetu Phaser razvoja zove se *Arcade Physics*.