Phaser - Animacija, obrada korisničke interakcije, kolizija

U prethodnoj lekciji započeli smo razvoj *Coin Collector* igre korišćenjem softverskog okvira Phaser. Realizovali smo prikaz pozadine, podloge i glavnog junaka i aktivirali osnovni sistem za simulaciju fizike. U ovoj lekciji nastavljamo tako gde smo se u prethodnoj zaustavili, pa ćete tako imati prilike da se upoznate sa još nekoliko vitalnih operacija koje su karakteristične za Phaser razvoj. Na kraju lekcije, dobićemo u potpunosti upotrebljivu igru.

Definisanje animacije glavnog junaka

Naredni korak u realizaciji naše igre, biće kreiranje spritesheet animacije. Kao što znate, reč je o animaciji koja nastaje brzim smenjivanjem susednih kadrova koji su definisani unutar jednog spritesheet-a. Phaser poseduje ugrađeni sistem za obavljanje takvog posla. Kako bi se iskoristio takav sistem, dovoljno je unutar `create()` funkcije, postaviti sledeći kod:

```
this.anims.create({
    key: 'stand',
    frames: this.anims.generateFrameNumbers('runner', {
        start: 0,
        end: 1
    }),
    frameRate: 10,
    repeat: -1
});

this.anims.create({
    key: 'run',
    frames: this.anims.generateFrameNumbers('runner', {
        start: 2,
        end: 6
    }),
    frameRate: 10,
    repeat: -1
});
```

Iz prikazanog koda možete da vidite da se animacija definiše korišćenjem svojstva `anims`, Scene klase.

Spritesheet animacija

Svojstvo `anims`, čuva referencu na objekat klase **AnimationManager**. Reč je o singleton klasi, što praktično znači da kompletna Phaser igra, deli zajednički objekat kojim se upravlja animacijom. Metoda `AnimationManager` klase, koja se koristi za kreiranje animacije je sledeća:

```
create(config)
```

Prilikom kreiranja animacije, metodi `create()` se prilaže objekat za konfigurisanje. On može imati sledeća svojstva:

- `key` - ključ animacije; ova vrednost će kasnije biti korišćena kako bi se referencirala kreirana animacija
- `frames` - objekat koji sadrži podatke koji se koriste za generisanje kadrova animacije
- `frameRate` - brzina reprodukcije animacije, izražena u broju kadrova po sekundi
- `duration` - dužina trajanja animacije izražena u milisekundama
- `delay` - zadrška, odnosno vreme koje je potrebno da protekne od iniciranja do započinjanja animacije
- `repeat` - broj ponavljanja animacije; vrednost -1 označava beskonačno ponavljanje

Za generisanje kadrova animacije, u primeru se koristi još jedna metoda klase `AnimationManager`. Reč je o metodi **generateFrameNumbers()**:

```
generateFrameNumbers(key, config)
```

Parametri imaju sledeće značenje:

- `key` - ključ teksture(slike) koja sadrži konkretan kadar animacije; reč je o ključu koji je definisan prilikom učitavanja resursa
- `config` - objekat za konfigurisanje kadrova
    - `start` - početni kadar animacije
    - `end` - završni kadar animacije

Upravo realizovane animacije biće korišćene kada korisnik pritisne neko dugme na tastaturi za kontrolisanje kretanja glavnog karaktera. Stoga je sledeći korak, realizacija sistema za obradu korisničke interakcije.

Obrada korisničke interakcije

Obrada korisničke interakcije, odnosno preciznije, obrada komandi koje pristižu sa nekih ulaznih uređaja, još jedan je od segmenata razvoja igara za web koji je olakšavan kada se koristi Phaser. Phaser obradu izvornih događaja tastature, miša i džojstika apstrahuje za nas, pa tako nema potrebe za direktnim slušanjem različitih događaja.

Prilikom realizacije naše igre, mi ćemo obrađivati događaje tastature. Konkretno, koristićemo strelice levo, desno i gore za upravljanje glavnim karakterom.

Centralna figura rada sa događajima jeste klasa `InputPlugin` kojoj je moguće pristupiti pomoću svojstva `input`, objekta scene. Za pretplatu na neki događaj, ova klasa se može iskoristiti na sledeći način:

```
this.input.on('pointerdown', callback, context);
```

Na ovaj način se obavlja pretplata na događaj `pointerdown`, bilo gde na površini `canvas` elementa.

Ipak, prava moć Phaser-a, ogleda se u postojanju specijalizovanih klasa, sa funkcionalnostima koje olakšavaju rad sa tastaturom i džojstikom:

- klasa `GamepadPlugin` čiji objekat se dobija korišćenjem svojstva `this.input.gamepad`
- klasa `KeyboardPlugin` čiji objekat se dobija korišćenjem svojstva `this.input.keyboard`

S obzirom da mi želimo da obrađujemo događaje tastature, u nastavku ćemo iskoristiti upravo navedenu klasu `KeyboardPlugin` i to na sledeći način:

```
let cursors = this.input.keyboard.createCursorKeys();
```

Prikazanom naredbom obavljeno je pozivanje metode `createCursorKeys()`, klase `KeyboardPlugin`.

Metoda createCursorKeys() i klasa Key

Korišćenjem metode `createCursorKeys()`, klase `KeyboardPlugin`, obavlja se kreiranje objekta koji će posedovati ukupno 6 svojstava. Biće to svojstva koja će predstavljati 4 strelice na tastaturi i dva dodatna tastera: *space* i *shift*. Tako će svojstva objekta koji će od strane metode `createCursorKeys()` da bude emitovan kao povratna vrednost da budu sledeća:

- `up`
- `down`
- `left`
- `right`
- `space`
- `shift`

Svaki od ovih svojstava popunjava se sa po jednim objektom klase `Key`. Objekti klase `Key` poseduju brojna svojstva koja je moguće koristiti za dobijanje informacija o tasteru na tastaturi koji se modeluje korišćenjem objekta ove klase. Neka od najznačajnijih takvih svojstava su:

- `isDown` - `boolean` svojstvo koje ima vrednost `true` sve dok je taster na tastaturi pritisnut
- `isUp` - `boolean` svojstvo koje ima vrednost `true` sve dok je taster na tastaturi otpušten
- `timeDown` - vreme poslednjeg pritiska tastera
- `timeUp` - vreme poslednjeg otpuštanja tastera
- `duration` - dužina trajanja poslednjeg pritiska tastera, odnosno vreme koje je proteklo između pritiska i otpuštanja tastera
- `shiftKey` - `boolean` svojstvo koje govori da li je pored konkretnog tastera, pritisnut i taster *shift*

Upravo prikazanu naredbu u kojoj se poziva `createCursorKeys()` metoda, najbolje je pozvati unutar `create()` metode. Ipak, s obzirom da ćemo tako dobijeni objekat (`cursors`), koristiti unutar `update()` metode, promenljivu `cursors` je potrebno deklarisati globalno.

Logiku za obradu korisničke interakcije i za ažuriranje stanja našeg glavnog junaka, dodaćemo unutar `update()` metode, unutar koje do sada nismo napisali ni jednu naredbu:

```
function update() {

    if (cursors.left.isDown) {
        runner.setVelocityX(-240);
        runner.flipX = true;
        runner.anims.play('run', true);
    } else if (cursors.right.isDown) {
        runner.setVelocityX(240);
        runner.flipX = false;
        runner.anims.play('run', true);
    } else {
        runner.setVelocityX(0);
        runner.anims.play('stand', true);
    }

    if (cursors.up.isDown) {

        if (runner.body.touching.down) {
            runner.setVelocityY(-500);
        }
    }
    if (!runner.body.touching.down) {
        runner.setFrame(7);
    }

    if (runner.body.velocity.y > 0) {
        runner.setFrame(8);
    }
}
```

Unutar metode `update()` dodata je logika za obradu korisničke interakcije. Definisana su ukupno 4 nezavisna uslovna bloka, koji zavređuju dodatno pojašnjenje.

Napomena

Ne zaboravite da se metoda `update()` poziva po jednom, tokom svake iteracije glavne petlje.

Prvi uslovni blok `if-else...if-else` proverava da li su pritisnuti tasteri levo ili desno. Kada korisnik na tastaturi pritisne strelicu na levo, svojstvo `cursors.left.isDown` dobija vrednost `true`, pa se unutar `update()` metode aktivira prvi if uslovni blok. Unutar prvog if uslovnog bloka, kretanje glavnog karaktera na levo se realizuje na sledeći način:

- brzina po X osi se postavlja na -240
- obavlja se okretanje slike sprajta
- aktivira se animacija sa imenom `run`

Odmah možete da vidite koliko olakšanje donosi upotreba Phaser-a, kada je u pitanju pomeranje objekata koji učestvuju u svetu fizike. Dovoljno je definisati brzinu takvog objekta i on započinje kretanje, čije usmerenje zavisi od vektora brzine. Tako se kretanje na levo, postiže definisanjem negativne brzine po X osi. Već u narednom else if bloku, možete da vidite da je kretanje u desno postignuto definisanjem pozitivne vrednosti brzine po X osi. Na kraju, kada korisnik ne pritiska ni jedan taster na tastaturi, biće aktiviran else uslovni blok. Unutar takvog uslovnog bloka, kako bi se glavni junak zaustavio, njegova brzina se postavlja na 0 i aktivira se nešto ranije definisana `stand` animacija.

Metoda play()

Iz prikazanog koda možete da vidite kako se pokreću spritesheet animacije koje smo kreirali nešto ranije. To se obavlja korišćenjem metode `play()` koja poseduje sledeći potpis:

```
play(key [, ignoreIfPlaying] [, startFrame])
```

Značenje parametara je sledeće:

- `key` - naziv animacije koju je potrebno pokrenuti
- `ignoreIfPlaying - boolean` vrednost kojom je moguće definisati ignorisanje poziva `play()` metode, ukoliko je animacija već pokrenuta
- `startFrame` - indeks početnog kadra animacija

Mi u prikazanom kodu koristimo `play()` metodu sa dva ulazna parametra. Prvim parametrom se definiše naziv animacije, a drugim to da animacija neće svaki put iznova započinjati, kada se pozove `play()` metoda. Ovo je veoma važna činjenica, posebno prilikom kretanja našeg glavnog junaka desno ili levo, s obzirom da će se u takvoj situaciji metoda `play()` pozivati u više sukcesivnih izvršavanja `update()` metode.

Pored prve uslovne konstrukcije `if-else...if-else`, unutar metode `update()` postoje još tri zasebna if uslovna bloka. Za početak, obrada pritiska na taster *up*, izolovana je unutar zasebnog uslovnog bloka, kako bi se omogućila kombinacija skoka i kretanja. Jednostavno, izmeštanjem logike za obavljanje skoka u zaseban blok, omogućava se kretanje glavnog junaka tokom skoka i direktan prelazak iz stanja trčanja u skok.

Skok se postiže slično kao i kretanje levo-desno. Postavlja se vrednost brzine, ali ovoga puta po Y osi. Pri tom, proverava se i da li je naš glavni karakter već u skoku, ispitivanjem vrednosti promenljive `runner.body.touching.down`. Ukoliko je glavni lik već u stanju skoka, neće se pozivati logika za dodeljivanje brzine.

Na kraju, unutar metode `update()` postavljena su još dva if uslovna bloka, kojima se rukuje grafikom glavnog junaka tokom skoka. Naime, Vi znate da se unutar spritesheet-a koji koristimo za predstavljanje glavnoj junaka, 2 nezavisna sprajta koriste za dočaravanje skoka. Jedan sprajt je potrebno koristiti tokom uzlazne faze skoka, a drugi tokom njegove silazne faze. Kako bi se odgovarajući sprajtovi postavili u odgovarajućem trenutku, prvo je definisan uslovni blok kojim se proverava da li je glavni lik u vazduhu:

```
if (!runner.body.touching.down) {
    runner.setFrame(7);
}
```

Ukoliko glavni lik ne dodiruje podlogu, kao kadar se postavlja onaj sa indeksom 7. To je kadar koji ilustruje uzlaznu fazu skoka. Na kraju, kako bi se detektovala silazna faza skoka, postavljen je ovakav uslovni blok:

```
if (runner.body.velocity.y > 0) {
    runner.setFrame(8);
}
```

Uslov if bloka proverava da li je brzina karaktera po Y osi veća od 0. Brzina našeg glavnog karaktera po Y osi je veća od nule onda, kada je on u slobodnom padu, pod uticajem sile gravitacije. To je upravo i silazna faza skoka, koja je nama potrebna, pa se u slučaju zadovoljenja ovakvog uslova, kadar prebacuje na onaj sa indeksom 8 (poslednji kadar).

Velocity, Gravity i Acceleration

Za realizaciju kretanja glavnog junaka naše igre, korišćene su dve osobine sistema arkadne fizike Phaser softverskog okvira. Reč je o brzini (*velocity, engl.*) i gravitaciji (*gravity, engl.*). Pored brzine i gravitacije, svi objekti koji su deo sistema fizike, poseduju i vektorsku veličinu ubrzanja (*acceleration, engl.*) Sve takve veličine je moguće koristiti za podešavanje ponašanja dinamičkih tela, pa na kraju i za dobijanje željenog efekta koji zavisi od zahteva igre.

Gravitacija (*gravity, engl.*) je sila koja deluje na sve dinamičke objekte koji su deo Phaser sistema fizike. Gravitacija se definiše korišćenjem jedinice **px/s^2**. (piksel po sekundi na kvadrat). Ubrzanje sile gravitacije se definiše globalno za sve objekte koji učestvuju u sistemu fizike, prilikom konfigurisanja takvog sistema:

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

Na ovaj način je ubrzanje sile gravitacije postavljeno na 300 piksela po sekundi na kvadrat i to po Y osi. Ovakvo ubrzanje će se primenjivati na sve objekte sa dinamičkim telom. U našem primeru to je objekat koji predstavlja glavnog junaka igre.

Ubrzanje sile gravitacije je moguće definisati i za svaki objekat posebno i to na sledeći način:

```
runner.body.setGravityY(500);
```

Na ovaj način je ubrzanje sile gravitacije eksplicitno definisano za našeg glavnog junaka. Ubrzanje usled sile gravitacije je postavljeno na 500 piksela po sekundi na kvadrat, što praktično znači da će naš glavni junak brže padati na podlogu igre.

Brzina (*velocity, engl.*) je još jedna vektorska fizička veličina koju mogu imati svi objekti sa dinamičkim telom. Brzina se izražava u pikselima po sekundi (**px/s**). Iz prethodnog primera Vi ste mogli da vidite da je mogućnost definisanja brzine iskorišćena za kretanje našeg glavnog junaka. Brzina nekog dinamičkog tela se može definisati korišćenjem sledećih metoda:

- `setVelocity(x [, y])` - omogućava definisanje brzine po X i po Y osi; moguće je proslediti samo prvi parametar i tada se definiše brzina samo po X osi
- `setVelocityX(value)` - omogućava definisanje brzine po X osi
- `setVelocityY(value)` - omogućava definisanje brzine po Y osi

Ukoliko definišete veće apsolutne vrednosti brzine našeg glavnog junaka, on će se kretati brže.

Ubrzanje (*acceleration, engl.*) je vektorska veličina koja definiše promenu brzine tokom nekog vremenskog perioda. Definiše se korišćenjem jedinice **px/s²**. Ubrzanje se može definisati korišćenjem sledećih metoda:

- `setAcceleration(x, y)` - omogućava definisanje ubrzanja po X i Y osi
- `setAccelerationX(value)` - omogućava definisanje ubrzanja po X osi
- `setAccelerationY(value)` - omogućava definisanje ubrzanja po Y osi

Kako biste videli efekat različitih vrednosti ubrzanja, glavnom karakteru naše igre možete postaviti ubrzanje po Y osi na 500, a zatim na -500. Tada ćete moći da vidite razliku u njegovom kretanju prilikom skoka, što je direktna posledica različitih vrednosti ubrzanja po Y osi.

Sve što smo do sada uradili, stvara efekat kao na videu 15.1.

https://youtu.be/9JVoSZhifss

Video 15.1 – Ponašanje koje je postignuto dosadašnjim kodom

Pitanje

Vektorska veličina koja definiše promenu brzine tokom nekog vremenskog perioda, zove se:

a) brzina
b) polet
c) ubrzanje
d) startnost

Objašnjenje:

Ubrzanje (acceleration, engl.) je vektorska veličina koja definiše promenu brzine tokom nekog vremenskog perioda.

Kreiranje novčića

Preostaje da realizujemo još jedan tip objekata u našoj igri. Reč je o objektima koji će predstavljati novčiće. S obzirom da će novčića biti više, sada ćemo po prvi put kreirati objekat koji se koristi za grupisanje većeg broja objekata:

```
coins = this.physics.add.group();
```

Na ovaj način je obavljeno kreiranje jedne grupe, u čijem kontekstu ćemo kreirati sve objekte koji će predstavljati novčiće. Grupe omogućavaju grupisanje srodnih objekata, kako bi njima moglo da se upravlja centralizovano. To praktično znači da mi možemo sada da definišemo određene osobine nad grupom, a takve osobine će automatski da budu primenjene nad svim objektima grupe:

```
this.physics.add.collider(coins, ground, gameOver, null, this);
this.physics.add.overlap(runner, coins, collectCoin, null, this);
```

Korišćenjem ove dve naredbe, definišu se osobine objekata novčića prilikom interakcije sa ostalim objektima koji učestvuju u svetu fizike. Prvo se definiše da će podloga naše igre predstavljati fiksnu barijeru za sve novčiće. Ipak, ovoga puta je metoda `collider()`, kojom se tako nešto obavlja, pozvana sa nešto više parametara nego prošli put.

Metoda collider()

Metoda `collider()` koristi se da konstruiše `Collider` objekat. Reč je o objektu koji automatski, tokom svake iteracije glavne petlje utvrđuje da li je došlo do kolizije ili preklapanja dva objekta koja se prate. Sintaksa ove metode izgleda ovako:

```
collider(object1, object2 [, collideCallback] [, processCallback] [, callbackContext])
```

Prva dva parametra se koriste za definisanje objekata koji se prate. Druga dva parametra služe za definisanje callback funkcija, dok poslednji parametar omogućava da se definiše kontekst u kome će da budu pozvane priložene callback funkcije.

Veoma je bitno da znate da definisane callback funkcije automatski dobijaju reference na objekte između kojih je došlo do kolizije.

Razlika između `collideCallback` i `processCallback` funkcija jeste u neophodnosti emitovanja `boolean` povratne vrednosti u slučaju druge funkcije. Takva osobina nama neće biti potrebna, pa ćemo isključivo koristiti `collideCallback` funkciju.

Kada novčić dodirne podlogu, igra se neuspešno završava. Zbog toga je `collider()` metodi prosleđena referenca na `gameOver()` funkciju koja će obaviti operacije vezane za završetak igre. Nešto kasnije će biti prikazana logika takve metode.

Kako bismo omogućili da naš glavni junak može da prikuplja novčiće, dodata je druga naredba u kojoj se poziva metoda `overlap()`.

Metoda overlap()

Metoda kojom je moguće utvrditi preklapanje dva objekta jeste metoda `overlap()`. Osnovna karakteristika `overlap()` metode jeste ta, da ona ne nastoji da zaustavi objekte kada dođe do kolizije, kao što je to slučaj sa metodom `collider()`. Stoga je ova metoda idealna za realizaciju funkcionalnosti prikupljanja novčića. Sintaksa metoda `overlap()` identična je metodi `collider()`.

Kao i kod metode `collider()` i ovde definisane callback funkcije automatski dobijaju reference na objekte između kojih je došlo do preklapanja.

Prilikom dolaska do preklapanja između glavnog karaktera i nekog novčića, definisano je da će se aktivirati funkcija `collectCoin()`.

```
function collectCoin(runner, coin) {
      coin.disableBody(true, true);
}
```

Upravo ste mogli da pročitate da callback funkcije automatski dobijaju reference na objekte između kojih je došlo do preklapanja. Mi smo njih u prikazanom primeru predstavili promenljivima `runner` i `coin`. Za sada, prilikom kontakta obavljamo uklanjanje novčića sa scene, pozivanjem metode `disableBody()` nad njegovom referencom.

Metoda disableBody()

Metoda `disableBody()` koristi se za stopiranje i skrivanje nekog objekta igre. Ova metoda ima sledeći oblik:

```
disableBody( [disableGameObject] [, hideGameObject])
```

Značenje parametara je sledeće:

- `disableGameObject` - vrednost `true`, deaktivira objekat igre
- `hideGameObject` - vrednost `true`, uklanja objekat igre sa scene

Podrazumevane vrednosti parametara su `false`.

Do sada mi još uvek nismo kreirali logiku za kreiranje novčića. To će biti obavljeno korišćenjem sledeće funkcije:

```
let accelerationY = -250;

function releaseCoin(scene) {

    let x = Phaser.Math.Between(50, 1230);

    let coin = coins.create(x, -100, 'coin');
```

```
        coin.setBounce(0.2);
        coin.setCollideWorldBounds(true);
        coin.setVelocity(Phaser.Math.Between(-60, 60), 0);

        coin.setAccelerationY(accelerationY);
        accelerationY = accelerationY + 5;


        let delay = Phaser.Math.Between(800, 3000)
        let  timer  =  scene.time.delayedCall(delay,  releaseCoin,  [scene],
    this);
    }
```

Upravo prikazanom funkcijom obavlja se sledeće:

- generisanje novog novčića se obavlja najranije 800 milisekundi, a najkasnije 3 sekunde od generisanja prethodnog novčića
- svaki novčić se inicijalno prikazuje izvan okvira canvas elementa, odnosno 100px izvan gornje ivice
- X koordinata na kojoj se generiše novčić nasumično se generiše
- X koordinata brzine novčića, takođe se nasumično generiše i može imati vrednosti između -60 i 60
- pod uticaj sile gravitacije, generisani novčići slobodno padaju ka tlu
- u svakoj iteraciji, vertikalno ubrzanje se povećava za 5 i time se postiže da se svaki sledeći novčić brže kreće ka tlu od prethodnog

Objekti koji predstavljaju novčiće, kreiraju se korišćenjem metode `create()` nešto ranije kreirane grupe.

Metoda create()

Metoda `Group` klase, koja se koristi za kreiranje objekata igre koji će automatski biti dodati takvoj grupi, izgleda ovako:

```
create( [x] [, y] [, key])
```

Parametri imaju sledeće značenje:

- `x` - horizontalna pozicija objekta koji se kreira unutar scene
- `y` - vertikalna pozicija objekta koji se kreira unutar scene
- `key` - ključ koji se odnosi na sliku kojom će da bude predstavljen objekat; ključ se odnosi na vrednost koja je definisana unutar metode `preload()`, prilikom učitavanja resursa

Unutar metode `releaseCoin()` možete da vidite da se X koordinata na kojoj će biti prikazan novčić, generiše nasumično. Isto se može reći i za brzinu novčića po X osi. Za postizanje nasumičnog generisanja vrednosti, koristi se još jedna Phaser funkcionalnost.

Phaser.Math.Between

`Between()` je statička metoda koja se nalazi u prostoru imena `Phaser.Math`. Ona ima sledeći oblik:

```
Between(min, max)
```

Metoda `Between()` omogućava generisanje nasumične vrednosti, ali u određenom opsegu. U prethodnim modulima ovoga kursa, Vi ste mogli da vidite da smo mi ovakvu funkcionalnost samostalno kreirali, s obzirom da ne postoji unutar JavaScript jezika, niti unutar nekog od Web API-a. Prilikom korišćenja Phaser-a, za samostalnim kreiranjem nema potrebe, s obzirom da je takva funkcija deo sistema.

Funkciju `releaseCoin()` je dovoljno pozvati jednom iz `create()` metode:

```
releaseCoin(this);
```

Kada se jednom pozove, naredna pozivanja ove metode će biti obavljana automatski, zahvaljujući logici sadržanoj u dve poslednje naredbe, kojim se kreira jedan tajmer.

Metoda delayedCall()

I kreiranje tajmera je jedna od operacija koja je olakšana prilikom rada sa Phaser-om. Naime, svojstvo `time`, objekta scene, čuva referencu na objekat tipa `Clock`. To je centralna figura sistema za upravljanje vremenom prilikom rada sa Phaser-om. Klasa `Clock` poseduje metodu `delayedCall()` koja se može koristiti za kreiranje tajmera:

```
delayedCall(delay, callback [, args] [, callbackScope])
```

Parametri ove metode imaju sledeće značenje:

- `delay` - vreme nakon koga će biti pozvana funkcija prosleđena drugim parametrom
- `callback` - referenca na funkciju koju je potrebno izvršiti odloženo
- `args` - niz koji predstavlja parametre koje je potrebno proslediti funkciju
- `callbackScope` - kontekst u kome je potrebno da bude pozvana callback funkcija

Score

Prikaz rezultata koji je korisnik ostvario tokom igre, biće obavljen prikazom teksta u gornjem, desnom uglu scene. Za početak je potrebo napraviti dve globalne promenljive:

```
let scoreText;
let score = 0;
```

Promenljiva `scoreText`, čuvaće referencu na objekat teksta, a promenljiva `score` će predstavljati ukupan broj poena koji je igrač ostvario.

Objekat teksta će biti kreiran unutar `create()` funkcije, na sledeći način:

```
scoreText = this.add.text(16, 16, 'score: 0', {
    fontSize: '32px',
    fill: 'white'
});
```

Metoda text() i klasa Text

Metoda `text()` koristi se za kreiranje objekta teksta i za njegovo dodavanje sceni. Tekst je jedan od osnovnih objekata igre, a predstavlja se klasom `Text`. Metoda `text()` poseduje sledeću sintaksu:

```
text(x, y, text [, style])
```

Značenje parametara je sledeće:

- `x` - horizontalna pozicija teksta
- `y` - vertikalna pozicija teksta
- `text` - tekst koji će biti prikazan
- `style` - objekat za stilizovanje teksta

Odmah nakon pozivanja metode `text()`, kreirani tekst će biti prikazan unutar scene. Ukoliko je nakon njegovog prikazivanja potrebno promeniti tekst, dovoljno je pozvati metodu `setText()`, sa novom vrednošću teksta:

```
setText(value)
```

Ukupan broj poena je potrebno uvećavati kada korisnik pokupi novčić:

```
function collectCoin(runner, coin) {
    score = score + 10;
    scoreText.setText('Score: ' + score);
    coin.disableBody(true, true);
}
```

Svakim prikupljenim novčićem, broj poena se uvećava za 10 i obavlja se ažuriranje prikaza unutar scene.

Game Over

Nešto ranije ste mogli da vidite, da se prilikom kontakta novčića i podloge, aktivira funkcija `gameOver()`. Nju još nismo kreirali, a evo kako će izgledati:

```
let isGameOver = false;

function gameOver() {
    this.physics.pause();
    runner.anims.play('stand');
    isGameOver = true;
}
```

Unutar metode `gameOver()` obavlja se nekoliko operacija:

- pauziranje simulacije arkadnog sistema
- aktiviranje animacije sa nazivom `stand`
- postavljanje vrednosti promenljive `isGameOver` na `true`

Promenljive `isGameOver` biće iskorišćena na nekoliko mesta, kako bi se igra zaustavila. Prvo, unutar `update()` metode:

```
function update() {

    if (isGameOver) {
        return;
    }

      ...

}
```

Kada dođe do završetka igre, potrebno je zaustaviti i generisanje novčića:

```
function releaseCoin(scene) {

    let x = Phaser.Math.Between(50, 1230);

    let coin = coins.create(x, -100, 'coin');
    runner.setBounce(0.2);
    coin.setCollideWorldBounds(true);
    coin.setVelocity(Phaser.Math.Between(-60, 60), 0);

    coin.setAccelerationY(accelerationY);
    accelerationY = accelerationY + 5;

    if (!isGameOver) {
        let delay = Phaser.Math.Between(800, 3000)
        let timer = scene.time.delayedCall(delay, releaseCoin, [scene],
this);
    }
}
```

Logika za kontinuirano pozivanje `releaseCoin()` funkcije, sada je upakovana unutar jednog uslovnog bloka, pa se generisanje novčića neće obavljati kada je vrednost promenljive `isGameOver` jednaka `true`.

Dodavanje zvuka

Za kraj, dodaćemo i dva zvučna efekta, odnosno naredbe za reprodukciju dva zvuka koja su učitana na početku, unutar metode `preload()`:

```
function preload() {
            ...

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

}
```

Na osnovu ovako učitanih zvučnih zapisa, potrebno je kreirati objekte, unutar metode `create()`:

```
collectSound = game.sound.add("collect");
failSound = game.sound.add("fail");
```

Naredbe za reprodukciju, postavićemo unutar odgovarajućih metoda. Unutar metode `collectCoin()`:

```
function collectCoin(runner, coin) {
    collectSound.play();
    score = score + 10;
    scoreText.setText('Score: ' + score);
    coin.disableBody(true, true);
}
```

I na kraju, unutar metode `gameOver()`:

```
function gameOver() {
    failSound.play();
    this.physics.pause();
    runner.anims.play('stand');
    isGameOver = true;
}
```

Napomena

Kompletan izvorni kod igre *Coin Collector* možete da preuzmete sa sledećeg linka:

`coin_collector_final.rar`

Rezime

- spritesheet animacija se definiše korišćenjem svojstva `anims`, `Scene` klase i njene metode `create(config)`
- za generisanje kadrova animacije, koristi se `generateFrameNumbers()` metoda klase `AnimationManager`
- Phaser apstrahuje obradu izvornih događaja tastature, miša i džojstika, pa tako nema potrebe za direktnim slušanjem različitih događaja

- centralna figura rada sa događajima jeste klasa `InputPlugin` kojoj je moguće pristupiti pomoću svojstva `input`, objekta scene
- korišćenjem metode `createCursorKeys()`, klase `KeyboardPlugin`, obavlja se kreiranje objekta koji će omogućiti praćenje događaja strelica na tastaturi
- gravitacija (*gravity, engl.*) je sila koja deluje na sve dinamičke objekte koji su deo Phaser sistema fizike
- brzina (*velocity, engl.*) je još jedna vektorska fizička veličina koju mogu imati svi Phaser objekti sa dinamičkim telom
- ubrzanje (*acceleration, engl.*) je vektorska veličina koja definiše promenu brzine tokom nekog vremenskog perioda
- brzina se izražava u pikselima po sekundi (px/s)
- gravitacija i ubrzanje se definišu korišćenjem jedinice piksel po sekundi na kvadrat (px/s^2)
- metoda `collider()` utvrđuje koliziju između dva objekta
- `Collider` objekat automatski, tokom svake iteracije glavne petlje utvrđuje da li je došlo do kolizije ili preklapanja dva objekta koja se prate
- metoda kojom je moguće utvrditi preklapanje dva objekta jeste metoda `overlap()`
- osnovna karakteristika `overlap()` metode jeste ta, da ona ne nastoji da zaustavi objekte kada dođe do kolizije, kao što je to slučaj sa metodom `collider()`
- metoda `disableBody()` koristi se za stopiranje i skrivanje nekog objekta igre
- metoda `Between()` omogućava generisanje nasumične vrednosti, ali u određenom opsegu
- klasa `Clock` poseduje metodu `delayedCall()` koja se može koristiti za kreiranje tajmera
- metoda `text()` koristi se za kreiranje objekta teksta i za njegovo dodavanje sceni
- tekst je jedan od osnovnih objekata igre, a predstavlja se klasom `Text`