Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Mikael Gousetis

Mobiilipelien kehitys Phaser-sovellus-kehyksen avulla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

16.4.2019

Tekijä Otsikko Sivumäärä Aika	Mikael Gousetis Mobiilipelien kehitys Phaser-sovelluskehyksen avulla 39 sivua 16.4.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Mobile Solutions
Ohjaaja	Yliopettaja Kari Salo

Insinöörityön tarkoituksena oli tutkia ja kartoittaa erilaisia pelikehyksiä ja löytää niistä sopiva, jonka avulla projektissa kehitettiin kaksi yksinkertaista mobiilipeliä. Asiakkaan tavoitteena oli käyttää insinöörityötä osana toisen asteen opiskelijoille pidettävää web-kehityksen alkeiskurssia, minkä vuoksi projektia varten kehitettyjen mobiilipelien piti olla yksinkertaisia, mutta samalla tarpeeksi houkuttavia opiskelijoiden mielenkiinnon ylläpitämiseksi.

Insinöörityössä kartoitettiin ja vertailtiin ensin eri sovelluskehyksiä, joita harkittiin projektin tekoon suunnitteluvaiheessa. Koska projektia varten kehitetyt mobiilipelit ovat web-pohjaisia, toimi ohjelmointikielenä JavaScript. Sovelluskehykseksi valikoitui Phaser, jonka avulla oli helppo kehittää yksinkertaisia 2D-mobiilipelejä.

Phaser-sovelluskehys sisältää paljon hyviä ominaisuuksia, kuten kolme erilaista fysiikkamoottoria, pelitilat ja niiden elinkaarimetodit sekä kattavan syötönohjauksen. Phaserin ydinominaisuus on sen pelimoottori, joka tekee mobiilipelien kehittämisestä helppoa ja nopeaa.

Insinöörityötä varten kehitettiin kaksi yksinkertaista mobiilipeliä sen kohderyhmän takia. Projektia varten kehitetyistä peleistä ensimmäinen oli endless runner -tyyppinen lentopeli ja jälkimmäinen oli tap to kill -henkinen nopeuspeli. Yksi pelien tärkeimmistä vaatimuksista oli pelien toimivuus ja skaalautuminen eri mobiililaitteissa. Tämä tavoite saavutettiin työssä hyvin.

Phaserin ominaisuuksia hyödyntäen insinöörityön mobiilipeleistä saatiin toteutettua helppokäyttöisiä ja toimivia. Insinöörityö täytti asiakkaan vaatimukset ja opetti sen toteuttajaa mobiilipelien kehittämisessä.

Avainsanat	Phaser, mobiilikehitys, web-kehitys, JavaScript

Author Title Number of Pages Date	Mikael Gousetis Mobile game development with Phaser game framework 39 pages 16 April 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Mobile Solutions
Instructor	Kari Salo, Principal Lecturer

The purpose of this thesis was to study and map different game frameworks and to find the most suitable for this project that would be used for the development of two simple mobile games. The client's objective was to use the thesis as part of a beginner's web development course for upper secondary students which is why the games developed for this project needed to be simple, but at the same time entertaining to keep students interested.

In the thesis, the various application frameworks that were considered for the project at the planning stage were first identified and compared. Because the mobile games that were developed for the project are web-based, the programming language of choice was JavaScript. The game framework chosen was Phaser, which made it easy to develop simple 2D mobile games.

Phaser game framework contains a lot of good attributes, such as three different physics engines, game states and their life cycle methods and a comprehensive input. The main attribute of Phaser is its game engine, which makes the development of mobile games easier and faster.

For this thesis, two simple mobile games were developed because of its target group. The first one of these games was an endless runner-type flight game and the latter was a tap to kill-type speed game. One of the most important requirements for the games was to be functional and scaling on different mobile devices. This goal was well achieved in this thesis.

Utilizing the features of Phaser the mobile games were made easy to use and functional. The thesis met the customer's requirements and taught its implementer in the development of mobile games.

Keywords	Phaser, mobile development, web development, JavaScript

Sisällys

1 Johdanto

Insinöörityön tarkoituksena on luoda helppo kehitysympäristö web-pohjaisten mobiilisovellusten luomiseen. Työssä kartoitetaan ja tutkitaan erilaisia pelikehyksiä ja valitaan niistä sopivin kahden yksinkertaisen mobiilipelin kehittämistä varten. Työ toimii osana toisen asteen opiskelijoille pidettävää web-kehityksen peruskurssia. Tarkoituksena on luoda mielenkiintoinen kehittämisympäristö käyttämällä hyväksi valitun pelikehyksen pelillisiä ominaisuuksia.

Insinöörityön on tilannut Metropolian Ammattikorkeakoulun yliopettaja Kari Salo. Ohjelmoinnista on tullut vuosi vuodelta tärkeämpi osa nykyaikaista opetussuunnitelmaa. Nuoria kannustetaan jatkuvasti ohjelmoimaan ja kehittämään ohjelmoinnillista ajattelutapaa. Tämän takia työtä varten kehitetyt mobiilisovellukset toimivat osana hänen lukiolaisille pitämäänsä web-kehityksen peruskurssia, jonka avulla pyritään luomaan helposti lähestyttävä ja houkutteleva kehitysympäristö opiskelijoille, joilla ei mahdollisesti ole aikaisempaa sovelluskehittämiskokemusta.

Insinöörityötä varten kehitettyjen mobiilisovellusten rakenne on työn suunnitteluvaiheessa muotoutunut melko yksinkertaiseksi. Tämän takia on tärkeä valita työn vaatimustasoon nähden sopivat työkalut mobiilisovellusten toteuttamiseen.

2 Pelikirjastojen vertailu

Ennen mobiilipelien suunnittelua piti insinöörityölle löytää oikeanlainen JavaScript-kehys, jota käyttäen peli tehtäisiin. Tässä luvussa käydään läpi ne kehykset, jotka eivät tulleet valituksi projektin tekoon, ja vertaillaan niiden hyviä ja huonoja puolia (taulukko 1). Työhön valituksi tullut sovelluskehys esitellään luvussa 3.

Taulukko 1. Sovelluskehysten vertailu [1; 2; 3; 4].

Sovelluskehys	Edut	Puutteet
PixiJS-grafiikkakirjasto	• Hyvä dokumentaatio • Hyvät esimerkit • Yksinkertainen	• Ei sisällä pelimoottoria
EaselJS-grafiikkakirjasto	• Hyvä dokumentaatio • Hyvät esimerkit	• Ei sisällä pelimoottoria • Liian monimutkainen • Vaatisi myös SoundJS-, TweenJS- ja PreloadJS-kirjastot
ThreeJS-JavaScript-kirjasto	• Hyvä dokumentaatio • Hyvät esimerkit	• Ei sisällä pelimoottoria • Sallii vain 3D-grafiikan luomisen • Tukee vain WebGL:n käyttöä • Liian yksipuolinen
PlayCanvas-editori	• Oma editori • Sisältää pelimoottorin	• Tukee vain WebGL:n käyttöä • Liian monimutkainen

PixiJS-grafiikkakirjasto

PixiJS on Mat Grovesin kehittämä avoimen lähdekoodin grafiikkakirjasto, jonka avulla käyttäjä pystyy helposti luomaan, näyttämään ja animoimaan interaktiivista 2D-grafiikkaa. Kirjasto on alustariippumaton, mikä tekee kehittämisestä eri alustoille yksinkertaisempaa. PixiJS tukee täysin WebGL:n käyttöä, mutta mahdollistaa myös HTML5:n Canvas-elementin käytön sitä tarvitsevissa tilanteissa. [1.]

Vaikka kirjastoa käytetään web-pohjaisten pelien tekemiseen, ei PixiJS silti ole pelimoottori. Kirjasto ei sisällä pelimekaniikkaa itsessään, vaan keskittyy pelkästään asioiden liikuttelemiseen näytöllä. Tämä syy johti myös siihen, miksi kirjastoa ei valittu mobiilipelien kehittämiseen insinöörityössä.

EaselJS-grafiikkakirjasto

EaselJS on GSkinnerin kehittämä grafiikkakirjasto JavasScriptille. Se toimii osana isompaa CreateJS-JavaScript-kirjastopakettia. EaselJS mahdollistaa graafisen toteutuksen esittämisen käyttäen HTML Canvas -elementtiä. EaselJS tukee myös WebGL:n käyttöä. [2.]

Vaikka EaselJS vaikutti hyvin ammattimaiselta ja sisälsi kattavan dokumentaation ja hyviä esimerkkejä, se jätettiin pois insinöörityön toteuttamisesta. Mobiilipelien toteutus EaselJS:llä vaikutti erittäin työläältä, ja kirjasto itsessään ei sisältänyt minkäännäköistä pelimekaniikkaa. Päätökseen vaikutti myös se, että EaselJS olisi myös vaatinut TweenJS-, SoundJS- ja PreloadJS-kirjastot.

Three.js-JavaScript-kirjasto

Three.js on Ricardo Cabellon vuonna 2010 GitHubiin julkaisema JavaScript-kirjasto. Three.js:n avulla käyttäjä pystyy luomaan animoitua 3D-grafiikkaa web-selaimelle. Three.js on alustariippumaton, ja se käyttää WebGL:ää grafiikan renderöimiseen. [3.]

Vaikka tätä kirjastoa pystytään käyttämään web-pohjaisten pelien kehittämiseen, se ei valikoitunut tämän insinöörityön toteuttamiseen. Suurin syy oli kirjaston yksipuolisuus. Pelien kehittäminen tällä kirjastolla olisi ollut hyvin rajattua, koska se sallii vain 3D-pelien luomisen. Kirjasto ei myöskään sisällä minkäännäköistä pelimoottoria, mikä tekee kehittämisestä haastavampaa.

PlayCanvas-editori

PlayCanvas on avoimen lähdekoodin editori, joka sallii 2D- ja 3D-pelien kehittämisen JavaScriptin avulla. Sen ovat luoneet Dave Evans ja Will Eastcott, ja se julkaistiin vuonna

2014. PlayCanvas sisältää täysin toimivan pelimoottorin, johon sisältyy graafisten toteutusten esittäminen, oma fysiikkamoottori, animaatioiden luonti ja erilaisten syöttölaitteiden tukemisen, kuten näppäimistö, hiiri tai peliohjain. PlayCanvas toimii selaimissa, jotka käyttävät WebGL:ää grafiikan renderöimiseen. [4.]

PlayCanvas ei valikoitunut tämän insinöörityön projektien toteuttamiseen, koska pelien kehittäminen sen avulla vaikutti liian monimutkaiselta. PlayCanvas ei myöskään tue HTML Canvas -elementin käyttöä.

3 Phaser-sovelluskehys

Phaser-sovelluskehys valittiin tämän insinöörityön toteuttamiseen, koska tärkeää oli luoda toisen asteen opiskelijoille helppo oppimisympäristö, jossa he pystyvät tutustumaan ja kehittämään helposti yksinkertaisia mobiilipohjaisia sovelluksia ilman aikaisempaa ohjelmointikokemusta. Phaser täytti nämä vaatimukset, ja koska pelillistäminen on Phaserin ydinominaisuus, se oli osuva valinta tämän insinöörityön toteuttamiseen.

Phaser on ilmainen, avoimen lähdekoodin pelikehys, jonka avulla käyttäjä pystyy luomaan 2D-HTML-pelejä joko mobiilialustoille tai perinteisille pöytäkoneille [5]. Phaser käyttää sekä Canvas- että WebGL-renderoijaa grafiikan esittämiseen ja vaihtelee näiden kahden välillä riippuen selaimen tukemasta muodosta. Tämä edesauttaa nopeaa renderoimista pöytäkone- ja mobiilialustalla. Kehittäminen Phaserilla on mahdollista käyttäen joko JavaScript- tai TypeScript-ohjelmointikieliä.

3.1 Tausta

Phaserin on kehittänyt Photon Storm. Sen ensimmäinen versio julkaistiin syyskuussa vuonna 2013, minkä jälkeen se on saanut lukuisia uusia päivityksiä. Phaserin ensimmäiset versiot sisälsivät Pixi.js-kirjaston grafiikan renderoimiseen. [6.] Myöhemmin se on poistettu Phaserin uusimmasta versiosta (Phaser 3). Phaser 2:n viimeisin virallinen versio oli 2.6.2, ja sen jälkeen Photon Storm loi uuden GitHub-tietolähteen nimeltä Phaser CE (Community Edition).

Phaser CE:n tarkoitus on, että se toimii stabiilina versiona, jota päivittää ja ylläpitää Phaser-yhteisö. Phaser CE:n luominen auttoi Photon Stromia keskittymään täysin Phaser 3:n kehittämiseen. Phaser CE:n ensimmäinen virallinen versio oli 2.7.0. [7.]

Phaser 3:n ensimmäinen versio julkaistiin helmikuussa vuonna 2018. Sitä varten Photon Storm loi uudestaan kehyksen jokaisen elementin käyttäen täysin modulaarista rakennetta, joka yhdistää dataorientoituneen lähestymistavan. Phaser 3 sisältää myös uuden WebGL-renderoijan, joka on suunniteltu juuri modernien 2D-pelien kehittämiseen. [8.]

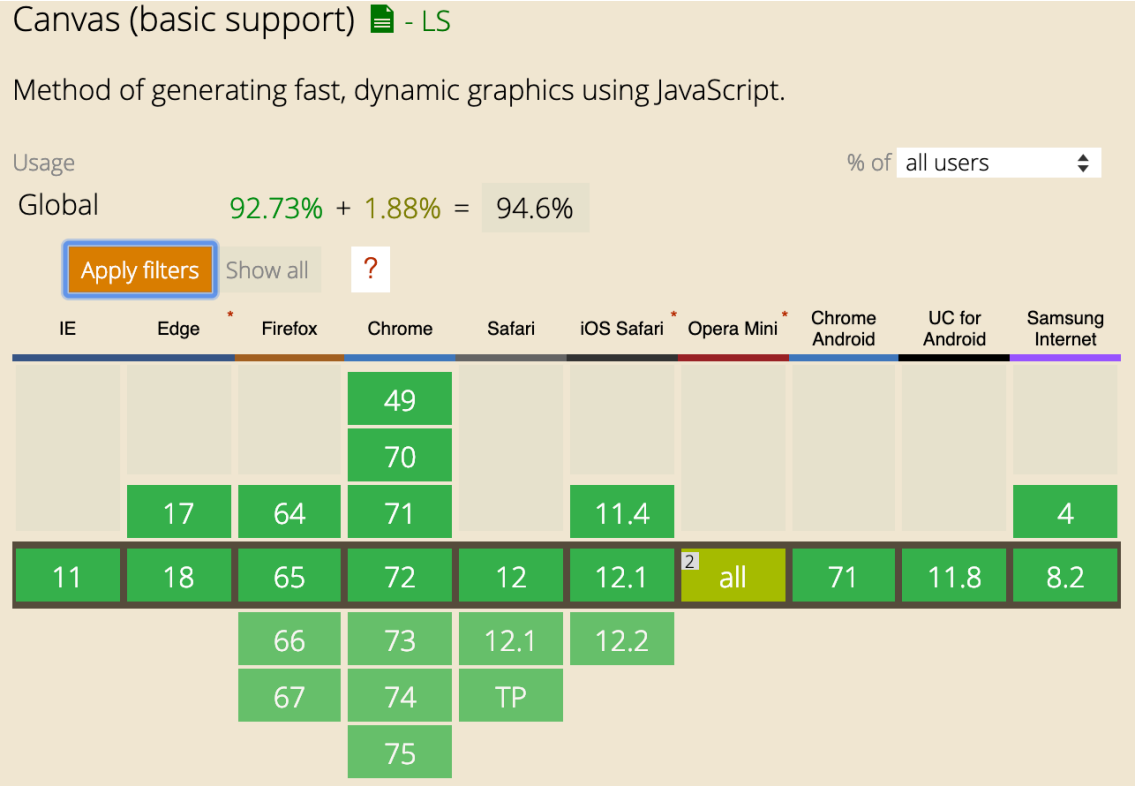
3.2 Ominaisuudet

Tämän insinöörityön kannalta tärkeimmät Phaserin ominaisuudet, jotka käydään läpi, perustuvat kehitettyihin sovelluksiin ja niissä käytettyihin ominaisuuksiin. Nämä ominaisuudet olivat Phaserin vähimmäisedellytykset, joiden avulla kehittäminen on mahdollista: pelimoottorin käyttämät eri fysiikkamoottorit, pelitila ja sen elinkaari sovelluksessa, spirte-kuvatiedosto, tween-siirtymä ja syötönohjaus.

Phaserin vaatimustaso

Toimiakseen Phaser tarvitsee tietynlaiset edellytykset. Se vaatii verkkoselaimen, joka tukee vähintään HTML Canvas -elementtiä. Phaser käyttää verkkoselaimissa joko WebGL-ohjelmointirajapintaa tai Canvas-elementtiä peligrafiikan esittämiseen JavaScriptin avulla.

Canvas-ohjelmointirajapinta sallii grafiikan esittämisen käyttämällä JavaScript-ohjelmointikieltä ja HTML Canvas -elementtiä. Sitä voidaan käyttää esimerkiksi tiedon visualisointiin, peligrafiikan esittämiseen, animaatioiden tuottamiseen ja valokuvien muokkaamiseen. Canvas-ohjelmointirajapinta keskittyy pääasiassa yksinkertaisen 2D-grafiikan esittämiseen. [9.] Canvas luo HTML-elementtiin piirtoalueen, jolle se määrittää korkeuden ja leveyden. JavaScriptia käyttämällä Canvas-ohjelmointirajapinta pystyy muokkaamaan tätä HTML-elementtiä. [10.] Canvas-elementin esitteli alun perin Apple sen käyttämälle OS X -käyttöjärjestelmälle vuonna 2004. Myöhemmin se on standardisoitunut käytettäväksi kaikille nykyaikaisille verkkoselaimille (kuva 1).



Kuva 1. Verkkoselaimet, jotka tukevat Canvas-elementin käyttöä [11].

WebGL on JavaScript-ohjelmointirajapinta, joka mahdollistaa interaktiivisen 2D- ja 3D-grafiikan esittämisen sille yhteensopivissa verkkoselaimissa ilman erillisiä liitännäisiä. WebGL on täysin integroitu muiden web-standardien kanssa, mikä mahdollistaa grafiikkaprosessorin hyödyntämisen kuvien muokkaamisessa ja grafiikan esittämisessä. Kuten Canvas-ohjelmointirajapinta, niin myös WebGL käyttää HTML Canvas -elementtiä alustana grafiikan renderöimiseen. [12.]

Phaser-sovelluksia kehittäessä voidaan valita, käyttääkö pelimoottori Canvasia, WebGL:ää vai molempia sovellusten renderöimiseen. Molempia käytettäessä pelimoottori yrittää automaattisesti käyttää WebGL:ää grafiikan esittämiseen, mutta jos verkkoselain tai laite ei tue WebGL:n käyttöä, ottaa pelimoottori automaattisesti Canvasin käyttöön grafiikan esittämistä varten. [13.]

Sovellusten kehittäminen Phaserillä onnistuu joko JavaScript- tai TypeScript-ohjelmointikielillä. TypeScript on Microsoftin kehittämä ja ylläpitämä avoimen lähdekoodin ohjelmointikieli. Se toimii JavaScriptin supersetinä, mikä tarkoittaa, että monet JavaScriptilla

kehitetyt sovellukset voidaan helposti kääntää TypeScriptille. [14.] TypeScriptin ja JavaScriptin välillä on joitakin eroja. Suurin niistä on se, että TypeScript on olio-ohjelmointikieli, siinä missä JavaScript on komentosarjakieli. Se sisältää toiminnallisuuden, jota kutsutaan staattiseksi kirjoittamiseksi, jota ei sisälly JavaScriptiin. TypeScript tukee myös moduulien käyttöä ja mahdollistaa rakenteiden luomisen objekteille. TypeScript ei tue abstraktien luokkien käyttöä, ja sen kääntäminen vie enemmän aikaa suhteessa JavaScriptiin. [15.] Tämän insinöörityön tekoon valittiin ohjelmointikieleksi JavaScript, koska kokemus sen käytöstä oli paljon kattavampi, joten sovellusten kehittäminen sitä käyttämällä oli luontevampaa.

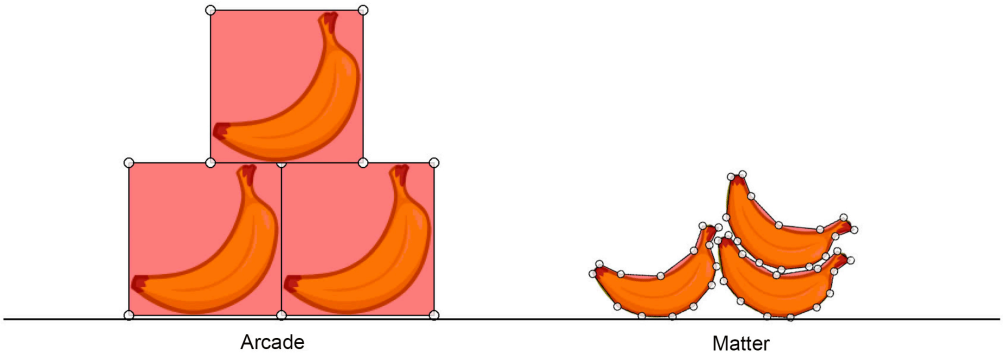
Fysiikkamoottori

Phaserin pelimoottoriin on sisäänrakennettu kolme erilaista fysiikkamoottoria, jotka ovat seuraavat:

- **Arcade-fysiikkamoottori** koostuu rajatuista suorakulmioista ja ympyröistä, joita ei pysty kääntämään. Tämä tarkoittaa, että pelimoottori piirtää kuvatiedoston ympärille suorakulmion tai ympyrän, jolla se merkitsee kuvatiedoston rajat. Tämän avulla Phaser tarkkailee, ilmeneekö toisen kuvan kanssa, joka myös on suorakulmio tai ympyrä, päällekkäisyyttä mahdollisen törmäyksen havaitsemiseksi. Arcade-fysiikkamoottori on kevyt ja nopea fysiikkamoottori, joka ei vaadi isoa suoritustehoa sovelluksilta, mutta sen yksinkertaisuuden takia se ei aina takaa varminta mahdollista törmäyksen havaitsemista. [16.]

- **Impact-fysiikkamoottori** pohjautuu ImpactJS-pelimoottorin käyttämään fysiikkamoottoriin. Kuten Arcade-fysiikkamoottori, myös Impact piirtää suorakulmion tai ympyrän muotoiset rajat kuvatiedoston ympärille, mutta erona Arcade-fysiikkamoottoriin, Impact-fysiikkamoottorin avulla pelimoottori pystyy kääntämään kuvatiedostoa pelitason vaatimusten mukaan. Jos esimerkiksi pelitasossa on kaltevuuseroja, pystyy Impact-fysiikkamoottori asettamaan oikean kaltevuuskulman kuvatiedostolle. Impact-fysiikkamoottori vaatii enemmän suoritustehoa kuin Arcade-fysiikkamoottori, mutta se pystyy havaitsemaan tarkemmin törmäykset. [17.]

- **Matter-fysiikkamoottori** on Matter.js:n luoma fysiikkamoottori, joka on implementoitu Phaser 3:een. Se on Phaserin fysiikkamoottoreista kehittynein. Sen sijaan, että se piirtäisi pelkästään suorakulmion kuvatiedoston ympärille, Matter-fysiikkamoottori pystyy luomaan monikulmaiset rajat kuvatiedoston ympärille ja sallii paljon kompleksisemman vuorovaikutuksen objektien välillä (kuva 2). Tämä sallii paljon tarkempia ja monimutkaisempia törmäyksiä objektien välillä. Vaikka Matter-fysiikkamoottori on kaikista tarkin havaitsemaan törmäykset, se on myös suoritusteholtaan vaativin sovellukselle. [18.]



Kuva 2. Arcade-fysiikkamoottori piirtää kuvatiedoston ympärille suorakulmion, minkä takia kuvatiedostot kasaantuvat kuin laatikot. Matter-fysiikkamoottori sallii paljon tarkemman monikulmiomuodon kuvatiedoston ympärille ja realistisemman kasaantumisen. [19.]

Pelitila

Pelitila on yksi Phaserin ydinominaisuuksista. Pelitilat jakavat sovelluksen koodin loogisiin palasiin, joita käyttämällä sovellus etenee. Sovellus käyttää aina vain yhtä pelitilaa kerrallaan. Yleisesti jokaiselle sovelluksen tapahtumalle on oma pelitilansa: erilliset pelitilat tasoille, päävalikolle, alustukselle ja pelin päättymiselle. Näiden eri pelitilojen välillä sovellus pystyy liikkumaan eteenpäin, ja se myös pystyy palaamaan jo käytettyihin tiloihin tarpeen mukaan. [20.]

Siirtyessään uuteen pelitilaan Phaser huolehtii, että aikaisemmin aktiivisena ollut pelitila sulkeutuu taustalta eikä jää viemään suoritustehoa [20]. Käyttämällä tiloja Phaser myös vapauttaa resursseja, poistaa kuuntelijoita käytöstä ja hallinnoi muistin vapautumista [21].

Sovellus voi koostua monestakin erilaisesta tilasta. Tässä insinöörityössä luodut sovellukset käyttivät seuraavia tiloja pelien luomiseen:

- **Boot**-tilassa voidaan määritellä pelille sen käyttämä fysiikkamoottori ja asettaa esimerkiksi mitat pelille [21].

- **Preload**-tilassa ladataan pelissä käytettävät kuvatiedostot [22].

- **GameTitle**-tilassa esitetään pelin otsikko. Pelaaja pystyy tässä tilassa aloittamaan pelin ja näkemään pelin toimintaohjeet. [20.]

- **Main**-tila on sovelluksen päätila, joka huolehtii pelilogiikasta [20].

- **GameOver**-tila kutsutaan, kun pelaaja häviää. Tila tulostaa näytölle ”Game Over” -tekstin ja esittää pelaajan tuloksen. Tila myös sallii pelaajan käynnistää pelin uudestaan halutessaan. [20.]

Pelitilojen käytön suurin hyöty Phaserissä on helppo koodin organisointi, luominen ja ylläpitäminen. Koska pelitiloja pystytään hallitsemaan erikseen, sovellukset vievät myös vähemmän suoritustehoa. [21.]

Kun Photon Storm julkaisi Phaser 3:n, kehittäjät nimesivät pelitilan (State) uudelleen pelinäkymäksi (Scene). Pelitilan ja pelinäkymän nimelliseron lisäksi ovat itse elinkaarimetodit ja niiden toiminnat pysyneet samanlaisina kuin aiemmin. [23.] Koska tätä insinöörityötä varten kehitetyt pelit käyttävät vanhempaa versiota Phaseristä (2.6.2), puhutaan tässä insinöörityössä pelitiloista.

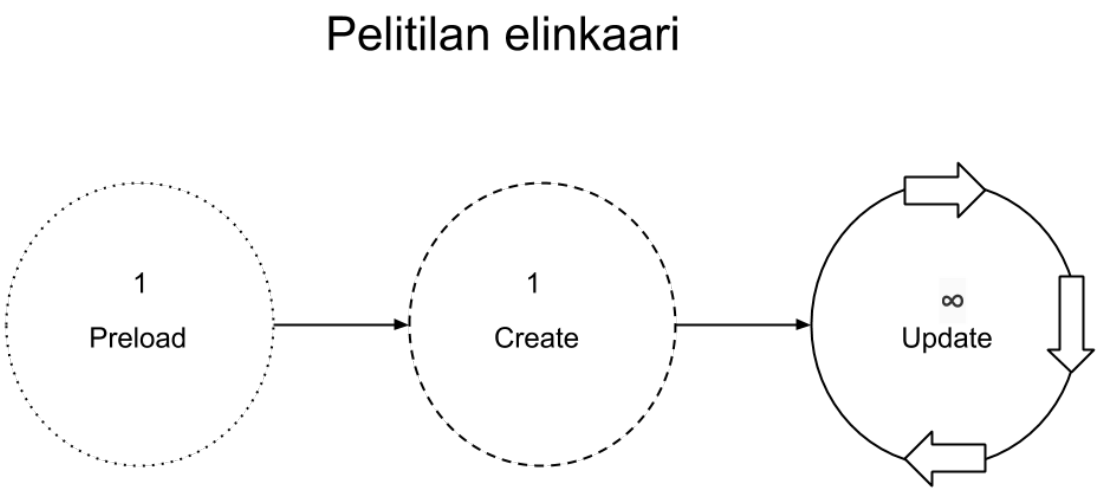
Pelitilan elinkaari

Phaserin pelitilat koostuvat tietyistä elinkaarimetodeista, joita käyttämällä pelitila pystyy alustamaan, luomaan ja suorittamaan sovelluksen pelilogiikan. Näistä elinkaarimetodeista (kuva 3) tärkeimmät ovat seuraavat:

- **Preload**-metodia kutsutaan ensimmäisenä. Tässä metodissa Phaser alustaa kaikki pelitilan tarvitsemat resurssit, kuten kuvatiedostot ja äänitiedostot. [24.]

- **Create**-metodi kutsutaan automaattisesti sen jälkeen, kun Preload-metodi on suorittanut. Tässä metodissa Phaser luo aiemmin Preload-metodissa alustetut kuva- ja äänitiedostot. Create-metodissa luodaan myös suurin osa sovelluksen asennuskoodista, kuten pelaaja ja pelitasot. Myös peliobjektien luominen tapahtuu tässä metodissa. [24.]

- **Update**-metodi on pelitilan elinkaaren ydin. Toisin kuin Preload- ja Create-metodit, Phaser kutsuu Update-metodia toistuvasti noin 60 kertaa minuutissa. Tämä metodi sisältää yleisesti pelin tärkeimmän logiikan, ja sen avulla sovellus voi tehdä päätöksiä, miten peli etenee. Update-metodi hoitaa myös objektien törmäyksen havaitsemisen. [24.]



Kuva 3. Pelitilan elinkaarta kutsutaan myös pelisilmukaksi (Game loop). Phaser kutsuu Preload- ja Create-metodeja vain kerran. Update-metodia kutsutaan 60 kertaa minuutin aikana.

Edellä mainitut elinkaarimetodit ovat pelitilalle tärkeimmät. Muita elinkaarimetodeja, joita pystytään käyttämään Phaserissä, ovat init, loadRender, loadUpdate, paused, pauseUpdate, preRender, render, resize, resumed ja shutdown.

Sprite

Sprite on Phaserin käyttämä kuvatiedosto, joka sisältää sarjan koordinaatteja ja tekstuureja, jotka pelimoottori renderöi canvas-elementtiin. Sprite sisältää myös lisäominaisuuksia, jotka sallivat fysiikan, animaation ja tapahtumien lisäämisen kuvatiedostolle. [25.] Spriten renderöiminen vie enemmän latausaikaa ja tehoa prosessorilta, minkä takia Phaser suosittelee perinteisten kuvatiedostojen käyttämistä staattisten kuvien esittämiseen. [26.]

Tween

Tween-siirtymällä Phaserissä tarkoitetaan toimintoa, jonka avulla pystytään muuttamaan yhtä tai useampaa ominaisuutta peliobjektissa tietyn ajanjakson välillä [27].

Phaser käyttää Tween-siirtymää peliobjektin automaattiseen liikuttamiseen muokkaamalla sen x- ja y-koordinaatteja tietyn ajanjakson aikana [16].

Syötönohjaus

Peliobjektien manuaaliseen liikkeiden hallintaan Phaser pystyy käyttämään erilaisia syötönohjausmenetelmiä. Peliobjekteja pystytään liikuttamaan joko hiiri- tai näippäinkomennoilla pöytäkoneilla tai kosketustapahtumilla (touch event) mobiililaitteissa. [28.]

4 Sovelluksien suunnittelu

Insinöörityön tavoite oli luoda helppo ja yksinkertainen kehitysympäristö web-pohjaisten mobiilipelien luomiseen Phaser-kehystä käyttämällä. Insinöörityö toimi osana lukioikäisille oppilaille suunnattua ohjelmointikurssia. Insinöörityön osana tehtiin kaksi yksinkertaista mobiilipeliä, jotka toimivat esimerkkeinä web-pohjaisten sovellusten kehittämisestä. Mobiilipelien rakenne piti pitää mahdollisimman yksinkertaisena, koska kurssi oli oppilaille, joilla mahdollisesti ei ollut ollenkaan aikaisempaa sovelluskehityskokemusta.

4.1 Pelikategoriat

Insinöörityötä varten piti valita sopivanlainen mobiilipelikategoria, jonka pohjalta työn pelit tehtäisiin. Mobiilipelit ulottuvat erittäin moneen pelikategoriaan, ja niiden avulla oikean pelityypin löytäminen voi olla hankalaa.

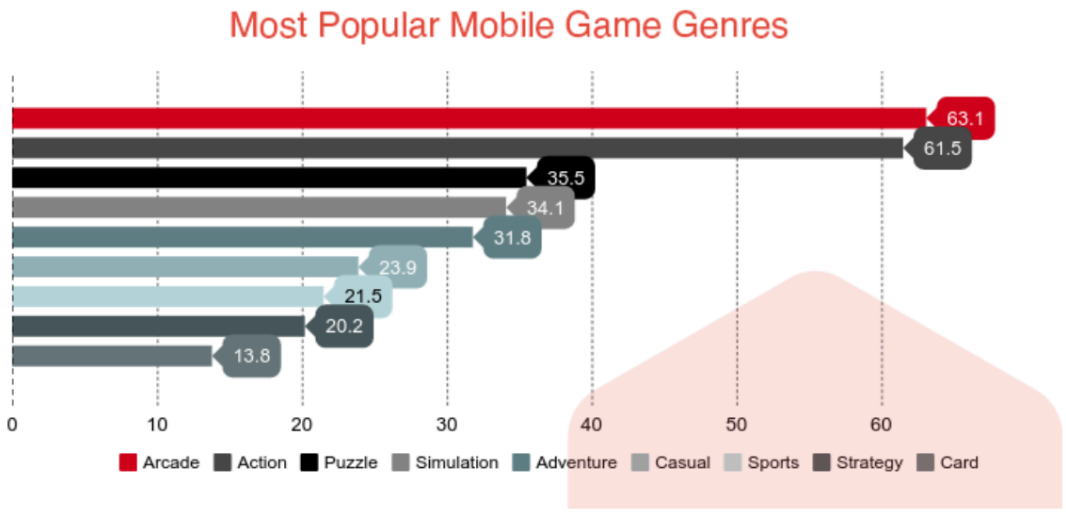
Suosituimmat mobiilipelikategoriat tätä insinöörityötä tehdessä olivat (kuva 4) seuraavat:

- **Arcade-pelit** ovat yleisesti luonteeltaan yksinkertaisia, eivätkä ne sisällä monimutkaista pelimekaniikkaa. Pelit keskittyvät enemmän pelattavuuteen kuin tarinankerrontaan tai sisällön tuottamiseen luomalla lyhyitä pelitasoja, jotka vaikeutuvat pelien edetessä. Nämä pelit eivät vaadi liikaa käyttäjältä, joten jokainen voi pelata näitä pelejä pelikokemuksesta riippumatta. [29.]

- **Toimintapelit** painottuvat yleisesti fyysisiin haasteisiin, mikä vaatii hyvät refleksit, nopean reaktioajan ja hyvän käden ja silmän koordinaation pelin pelaajalta. Toimintapelit keskittyvät pääasiassa pelattavan hahmon hengissä pitämiseen erilaisissa haastavissa ympäristöissä. Toimintapelien tasot kovenevat ja vastusten määrä pelikentissä kasvaa, mitä pidemmälle pelaaja pääsee. [30.]

- **Pulmapelit** keskittyvät käyttäjän ongelmanratkaisemiskykyihin. Pulmapelit sisältävät monesti sanakilpailuja, lukujonojen ratkomista ja kuvioiden tunnistamista. Monesti pelaaja saa rajattoman ajan ja yrittämiskerrat pulmien ratkaisemiseksi. [30.]

- **Seikkailupelit** keskittyvät tarinankerrontaan. Ne ovat monesti yksinpelejä, joissa pelaajan ohjaama hahmo etenee ratkomalla erilaisia pulmia ja vaikuttaa näin tarinan lopputulemaan. Seikkailupelien keskittyminen tarinankerrontaan mahdollistaa moniulotteisten juonien luomisen, mikä tekee pelikokemuksesta paljon syvemmän. [30.]



Kuva 4. Suosituimmat mobiilipelikategoriat vuonna 2017. Pelien latausmäärät on ilmoitettu miljoonina. [30.]

Koska tämä insinöörityö toimi osana kurssia, joka toteutettiin toisen asteen opiskelijoille, piti kehitettävien mobiilipelien olla yksinkertaisia, mutta samalla antaa peruskäsitys mobiilipelien kehittämisestä. Pelien käyttöliittymän piti olla intuitiivinen ja helposti toteutettava. Näiden vaatimusten perusteella päätyi insinöörityötä varten kehitettäväksi pelikategoriaksi arcade-pelit. Suosituimmat pelit viime vuosina arcade-kategoriassa ovat olleet Crossy Roads, Temple Run, Angry Birds ja Fruit Ninja.

Edellä mainittujen pelien suosion takia pelityypit, jotka valittiin tämän insinöörityön toteutukseen, olivat

- endless runner
- tap to kill.

Endless runner

Endless runner -pelit ovat tasohyppelypelejä, joissa pelaajan hahmo liikkuu koko ajan eteenpäin loputtomiin jatkuvassa, menettelyllisesti luodussa pelimaailmassa. Pelaajan ohjaaman hahmon liikkeet on yleensä rajoitettu vain muutamaan liikkeeseen, kuten hyppääminen, väistäminen tai hyökkääminen. Endless runner -pelien tarkoitus on päästä pelihahmon kanssa mahdollisimman pitkälle, ennen kuin pelaaja häviää. Pelikentät koostuvat yleisesti erilaisista esteistä, joita onnistuneesti väistämällä peli nopeutuu ja vaikeutuu vähitellen. Endless runner -tasohyppelypelit ovat saavuttaneet todella ison suosion mobiilialustoilla. Ne ovat hyvin sopivia näiden pelien rajallisiin ohjainvaatimuksiin, kuten yksittäiseen näytön näpäytykseen pelihahmon liikuttamiseksi. [31.]

Tap to kill

Tap to kill -peleissä pelin vastustajia ilmaantuu peliruudulle ja pelaajan pitää koskettamalla tuhota ne. Peli nopeutuu ja vastustajien määrä kasvaa sitä mukaa, kuin pelaaja niitä tuhoaa. Peli päättyy, kun vastustaja saavuttaa mahdollisen päämääränsä pelitasolla. Tap to kill -pelit ovat yleensä hyvin nopeatempoisia ja soveltuvat hyvin mobiilialustoille yksinkertaisen pelilogiikkansa takia.

Insinöörityössä päädyttiin näiden pelityyppien valitsemiseen, koska työ vaati pelien olevan helposti kehitettäviä ja sisältävän yksinkertaisen pelimekaniikan rajoitetuilla ohjainkomennoilla. Nämä pelityypit toimivat myös hyvänä esimerkkinä muista tämän pelikategorian suosituimmista peleistä, mikä teki pelien kehittämisestä lähestyttävämpää ja viihdyttävämpää.

4.2 Työkalut sovelluksien luomiseen

Sopivien pelikategorioiden löytämisen jälkeen alkoi projektin kehitysympäristön valitseminen. Insinöörityölle piti löytää sopiva Phaser-versio, jota käytettiin pelin pelimoottorina, ja projektille piti valita oikeanlainen tekstieditori mobiilipelien kehittämiseen ja palvelin, jonka avulla pelejä pystytään testaamaan mobiililaitteissa. Työkalut, jotka valikoituivat insinöörityön tekemiseen, olivat

- Phaser 2.6.2
- Brackets
- Browsersync.

Phaser 2.6.2

Sovelluksien toteuttamista varten valittiin teknologiaksi Phaser-pelimoottori. Tämän jälkeen piti projektille valita Phaseristä oikea versio, jota mobiilipelit käyttäisivät pelimoottorinaan. Pelimoottoriksi valikoitui Phaser 2.6.2 ”Kore Springs” -versio. Tämä oli Phaser 2:n viimeisin virallinen versio, jonka Photom Storm oli julkaissut, ennen kuin se aloitti Phaser 3:n kehittämistä. Vaihtoehtona oli myös Phaser CE, joka on Phaser-yhteisön ylläpitämä versio pelimoottorista. Phaser 2.6.2 valittiin tämän projektin tekoon, koska siitä oli kattavin ohjelmointirajapintadokumentaatio saatavilla tätä insinöörityötä tehtäessä.

Brackets

Phaser-pelien ohjelmointia varten piti valita oikeanlainen kehittämisympäristö. Tekstieditoriksi valikoitui Brackets. Se on Adobe Systemin kehittämä ilmainen, avoimen lähdekoodin tekstieditori web-kehittämistä varten. Brackets on järjestelmäriippumaton, mikä tarkoittaa, että sitä voi käyttää Windows-, Linux- ja macOS-käyttöjärjestelmistä. Bracketsin tärkein ominaisuus on sen mahdollistama reaaliaikainen koodin muokkaus ja testaaminen, ilman erillistä palvelinta. [32.]

Browsersync

Insinöörityöhön kehitettyjen mobiilipelien testaamista varten tarvittiin palvelin. Palvelimen luomista varten valittiin Browsersync-automaatiotyökalu, joka tekee web-kehittämisestä nopeampaa. Browsersyncin avulla projektille voidaan luoda pieni palvelin lokaalisti, mikä mahdollistaa Phaser-pelien testaamisen mobiilialustoilla. [33.]

4.3 Prototyyppi-suunnittelumalli

Insinöörityötä varten kehitetyt mobiilipelit käyttävät prototyyppi-suunnittelumallia. Tämä suunnittelumalli perustuu JavaScriptin prototyyppiseen perintään. Tässä suunnittelumallissa kaikki objektit, jotka luodaan, ovat vain yksinkertaisempia kopioita alkuperäisestä objektista. Prototyyppi-suunnittelumallista puuttuu kokonaan luokkamääritys. [34.]

Uuden objektin luomiseksi kopio-objekti sisältää rakentajafunktion, joka toimii instanssina alkuperäiselle objektille. Esimerkiksi, tässä insinöörityössä Phaser luo peliobjektin, josta prototyyppi-suunnittelumallin mukaan luodaan kopio-objekteja, joita pelimoottori käyttää pelin tiloina ja kutsuu niitä tarvittaessa.

5 Sovelluksien toteutus

Insinöörityötä varten kehitettyjen mobiilipelien pelityypeiksi valikoituivat endless runner ja tap to kill.

Ensimmäinen insinöörityötä varten kehitetyistä peleistä oli endless runner -peli, jonka ideana oli lentää pienen avaruusaluksen kanssa mahdollisimman pitkälle väistellen eteen tulevia asteroidisokkeloita ja välttää osumasta niihin. Pelaaja ohjaa avaruusalusta näpäyttämällä mobiililaitteen näyttöä, mikä saa avaruusaluksen hyppäämään ilmaan jatkamaan lentorataa. Peli päättyy, kun pelaaja joko osuu asteroidisokkelon seinämiin tai avaruusalus putoaa liian matalalle. Peli sai inspiraationsa suositusta Flappy bird -mobiilipelistä. Peli sai nimekseen Asteroid dodge.

Toinen insinöörityötä varten kehitetyistä peleistä oli tap to kill -peli. Pelin idea on yksinkertainen: pelaajan pitää tuhota näytölle ilmestyvät jalkapallot koskettamalla niitä. Jokaisella kerralla, kun pelaaja tuhoaa yhden pallon, hän saa yhden pisteen. Pelin luomat jalkapallot ilmestyvät näytön oikeaan reunaan ja kulkeutuvat kohti vasenta reunaa. Jos pelaaja epäonnistuu jalkapallon tuhoamisessa ja pallo onnistuu osumaan pelin vasempaan reunaan, pelaaja häviää pelin. Peli myös vaikeutuu lineaarisesti: jokainen uusi pallo, jonka peli luo, saa hieman nopeamman vauhdin kuin aikaisempi pallo, mikä kasvattaa näin pelin tempoa. Tämän pelin nimeksi annettiin Ball fondlers.

5.1 Asteroid dodge -peli

Pelin alustus

Phaser alustetaan luomalla skripti HTML-sivulle. Phaser luo HTML-sivulle Canvas-elementin, jota se tulee käyttämään peliobjektin alustana. Skriptin sisään luodaan ensiksi konfiguraatio-objekti, jolle annetaan parametreiksi pelille oleellisia tietoja, kuten sen käyttämä rendreröintimoottori, pelin leveys, korkeus ja fysiikkamoottori. Tämän jälkeen luodaan Phaser-peliobjekti, jolle annetaan parametriksi aikaisemmin luotu konfiguraatio-objekti. Tämän jälkeen peliobjektiin lisätään pelissä tarvittavat pelitilat, joista jokaiselle annetaan tarvittavat parametrit. Lopuksi peliobjektille annetaan käynnistymiskäsky, joka käynnistää ensimmäisen pelitilan nimeltä ”preload” (esimerkkikoodi 1).

```
//Creating a config file with the proper parameters for the game
var config = {
     type: Phaser.AUTO,
     width: 600,
     height: 800,
     pixelArt: true,
     physics: {
          default: "arcade",
          arcade: {
               gravity: {y: 0},
               debug: false
               }
          }
};

//Initializing phaser object
var game = new Phaser.Game(config);

//Adding all the states to the object
game.state.add('preload', preloadState);
game.state.add('gameTitle', gameTitleState);
game.state.add('main', mainState);
game.state.add('gameOver', gameOverState);


//Starting the first state for the game
game.state.start('preload');
```

Esimerkkikoodi 1. Phaserin alustus ja peliobjektin luominen.

Pelitilojen toimintasykli

Ennen pelitilojen luomista piti tiloille suunnitella toimintasykli, jonka mukaan pelitilat luotaisiin ja suoritettaisiin pelissä (kuva 5). Pelitilojen toimintasykliksi muodostui seuraava:

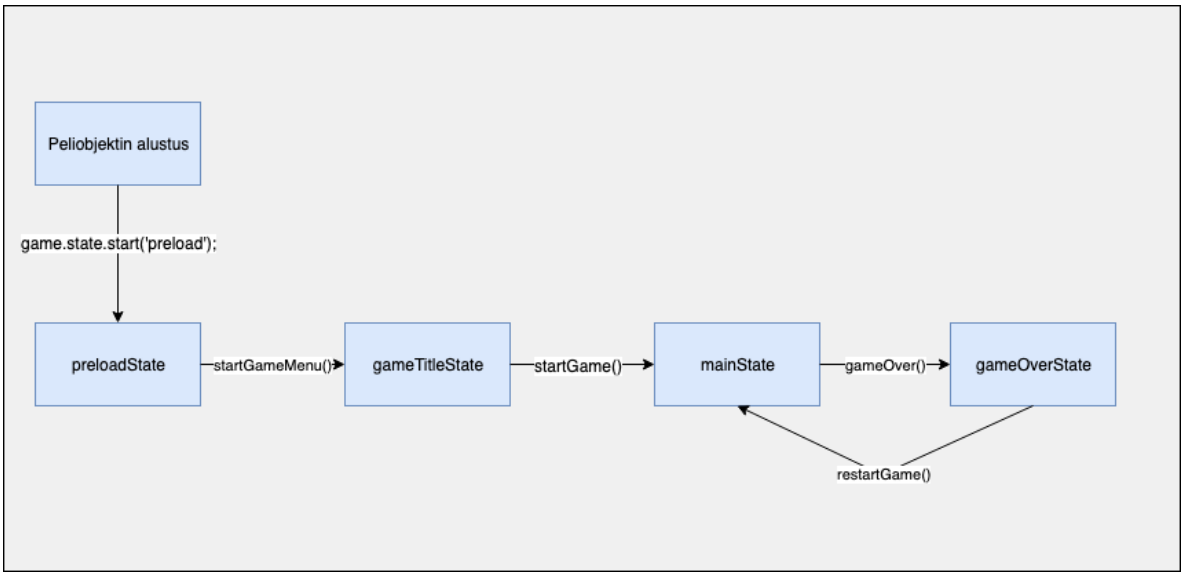
- **preload**-pelitilaa kutsutaan ensimmäisenä peliobjektin alustamisen jälkeen. Tässä pelitilassa ladataan pelin käyttämät ääni- ja kuvatiedostot valmiiksi ja huolehditaan pelin skaalautumisesta erikokoisten mobiililaitteiden näytöille. Tämän jälkeen prelaod-pelitila kutsuu gameTitle-tilaa.

- **gameTitle**-pelitila toimii pelin aloitusvalikkona. Tässä pelitilassa pelaajalle ilmoitetaan pelin nimi ja toimintaohjeet, jolla peliä pelataan.

- **main**-pelitila sisältää kaiken pelille oleellisen pelilogiikan. Pelitilassa luodaan avaruusalus, joka toimii pelin pelattavana hahmona ja luo sille liikkeet ja fysiikan. Tilassa luodaan myös asteroidisokkelo ja siihen sisältyvä liikerata.

Tässä pelitilassa huolehditaan myös törmäyksen havaitseminen avaruusaluksen ja asteroidien välillä. Pelihahmon osuessa esteisiin peliobjekti kutsuu gameOver-pelitilaa.

- **gameOver**-pelitilassa pelaaja saa tiedon pelin päättymisestä ja siitä, kuinka monen asteroidisokkelon läpi pystyi hän lentämään. Pelitila kertoo myös pelin uudelleenpelaamismahdollisuudesta. Tämän jälkeen pelitila kutsuu uudestaan main-pelitilaa ja aloittaa uuden pelin.



Kuva 5. Pelitiolen toimintasykli Asteroid Dodge -pelissä.

Pelin skaalaaminen ja ääni- ja kuvatiedostojen lataaminen

Pelin tulee skaalautua erikokoisille näyttöpäätteille mobiililaitteista riippumatta. Tämä saavutetaan käyttämällä Phaserin Scale Manager -objektia. Sen avulla pystytään peliobjektille kertomaan, miten pelin pitää linjata Canvas-elementti horisontaalisesti ja vertikaalisesti. Tämän lisäksi peliobjektille asetetaan skaalautumistyyppi, jonka avulla peli täyttää näytölle varatun tilan säilyttäen samalla sen kuvasuhteen oikeana (esimerkkikoodi 2).

Preload-pelitilassa pitää pelin käyttämät kuvatiedostot ladata valmiiksi, jotta peliobjekti pystyy käyttämään niitä tarvittaessa. Tämä tehdään kutsumalla peliobjektin

load.image()-funktiota, joka ottaa parametreiksi kuvalle annettavan nimen sekä polun, jossa kuvatiedosto sijaitsee (esimerkkikoodi 2).

```
//Canvas will be horizontally and vertically-aligned
game.scale.pageAlignHorizontally = true;
game.scale.pageAlignVertically = true;

//Scale to fill the screen while preserving the ratio.
game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
game.stage.disableVisibilityChange = true;

//Loading the assets for the game
game.load.image('ufo', 'assets/ufo.png');
game.load.image('asteroid', 'assets/smallAsteroid.png');
game.load.image('starfield', 'assets/starfield.png');

//Loading sounds for the game
game.load.audio('jump', 'assets/jump.wav');
game.load.audio('main_game','assets/main_music.wav');
```

Esimerkkikoodi 2. Peliobjektin skaalaaminen sekä ääni- ja kuvatiedostojen lataus pelin käyttöä varten.

Esimerkkikoodissa 2 nähdään myös äänitiedostojen lataaminen peliobjektin käyttöä varten. Tapa, jolla äänitiedostot ladataan, on hyvin samankaltainen kuin kuvatiedostojen lataaminen. Peliobjekti käyttää load.audio()-funktiota, jolla annetaan parametreiksi äänitiedostolle haluttu nimi sekä polku, jossa äänitiedosto sijaitsee.

Päävalikko

Pelin päävalikko sijaitsee mainTitle-pelitilassa, jossa pelaajalle kerrotaan pelin nimi ja annetaan ohjeet pelihahmon liikuttamista varten. Pystyäkseen siirtymään seuraavaan pelitilaan pitää pelaajan koskettaa näyttöä. Tämä toteutetaan pelitilassa kutsumalla peliobjektia ja käyttämällä sen input.onTap.add()-metodia, joka ottaa parametrikseen startGame()-funktion, jonka avulla se käynnistää ”main”-pelitilan (esimerkkikoodi 3).

```
//Starting the main game when user taps the screen
game.input.onTap.add(this.startGame, this);
```

Esimerkkikoodi 3. Käyttäjä aktivoi seuraavan pelitilan käyttämällä onTap-metodia.

Main-pelitila ja pelin logiikka

Pelin tärkein pelilogiikka sijaitsee main-pelitilassa. Tämän tilan sisällä luodaan pelihahmo ja annetaan sille liikkumiskomennot, luodaan asteroidisokkelo ja hallitaan törmäyksen havaitseminen pelihahmon ja asteroidien välillä.

Tätä peliä varten luotu pelihahmo oli avaruusalus, jonka tehtävänä on pysyä mahdollisimman kauan lennossa, samalla väistellen päin tulevia asteroidisokkeloita. Avaruusalus luodaan createUfo()-funktiossa, jossa avaruusalukselle määritetään ensimmäiseksi sen käyttämä kuvatiedosto peliobjektin käyttämällä add.sprite()-metodilla. Metodi ottaa parametreikseen näytöllä määritellyt x- ja y-koordinaatit sekä aiemmin ”preload”-pelitilassa kuvatiedostolle annetun nimen. Tämän jälkeen skaalataan avaruusalus oikeankokoiseksi scale.set()-metodia käyttäen. Seuraavaksi avaruusalukselle otetaan käyttöön peliobjektin käyttämä arcade-fysiikkamoottori. Lopuksi pelihahmolle annetaan vetovoima y-akselin suuntaisesti vetämään sitä alaspäin (esimerkkikoodi 4).

```
createUfo() {

      //Showing the Ufo in the correct position x=100 and y=245
      this.ufo = game.add.sprite(100, 245, 'ufo');

      //Giving the ufo a bit larger scale
      this.ufo.scale.set(2.5);

      //Adding physics to the ufo
      game.physics.arcade.enable(this.ufo);

      //Giving gravity to the ufo
      this.ufo.body.gravity.y = 1000;
}
```

Esimerkkikoodi 4. Pelattava pelihahmo luodaan createUfo()-funktiossa.

Ennen kuin peliin luodaan asteroidisokkelo, pitää ensimmäiseksi luoda yksittäinen asteroidi. Yksittäinen asteroidi luodaan samalla tavalla kuin pelihahmokin. Ensin asteroidille annetaan oikea kuvatiedosto, minkä jälkeen se ottaa arcade-fysiikkamoottorin käyttöönsä. Asteroidille lisätään myös sen x-akselin myötäinen kiihtyvyys, jolla se kulkeutuu kohti avaruusalusta.

Asteroidille annetaan myös tween-siirtymä. Esimerkkikoodissa 5 nähdään, miten asteroidille luodaan muuttuja asteroidAnimation, joka käyttää peliobjektin add.tween()-metodia. Muuttujalle annettaan tween-metodin käyttämät parametrit. Tween-siirtymä asettaa asteroidin pyörimään 60 astetta 3000 millisekunnin ajan.

```
//Adding animation to the asteroid using tween, this rotates the asteroid 60
//degrees over the period of 3000 milliseconds or 3 seconds
var asteroidAnimation = game.add.tween(asteroid);
asteroidAnimation.to(
      { angle: 60 },
      3000,
      Phaser.Easing.Linear.None,
      true
);
```

Esimerkkikoodi 5. Asteroidille lisättävä tween-siirtymä.

Asteroidille pitää myös kertoa, missä menevät pelinäkymän rajat, ja että sen pitää ”kuolla” ylittäessään nämä rajat. Esimerkkikoodissa 6 nähdään, kuinka asteroidille asetetaan parametrit checkWorldBounds ja outOfBoundsKill. Molempien parametrien tilaksi tulee tosi. Tämä tarkoittaa, että kun asteroidi ylittää pelin rajat, peliobjekti poistaa sen automaattisesti.

```
//Killing the asteroid when it is no longer visible in the screen
asteroid.checkWorldBounds = true;
asteroid.outOfBoundsKill = true;
```

Esimerkkikoodi 6. Asteroidi tuhoutuu ylittäessään pelin rajat.

Koska asteroideja tarvitaan useampi sokkelon luomista varten, soveltuu tähän hyvin peliobjektin käyttämä add.group()-metodin käyttö. Pelitilan create-elinkaarifunktiossa luodaan asteroids-niminen muuttuja, jolle annetaan add.group()-metodi. Tämän jälkeen aikaisemmin luotu yksittäinen asteroidi lisätään tähän luotuun ryhmämuuttujaan.

Asteroidisarakkeen luominen ja tyhjän aukon laskeminen

Kun yksittäinen asteroidi on saatu luotua, pitää luoda sarake, joka koostuu monesta asteroidista, ja laskea sille tyhjä aukko, jonka läpi pelihahmo pystyy lentämään. Tämä onnistuu luomalla addRowOfAsteroids()-funktio. Tässä funktiossa luodaan asteroidsNeeded-muuttuja, jonka sisällä lasketaan, kuinka monta asteroidia mahtuu sarakkeeseen, jakamalla pelikentän korkeus yksittäisen asteroidin korkeudella (esimerkkikoodi 7).

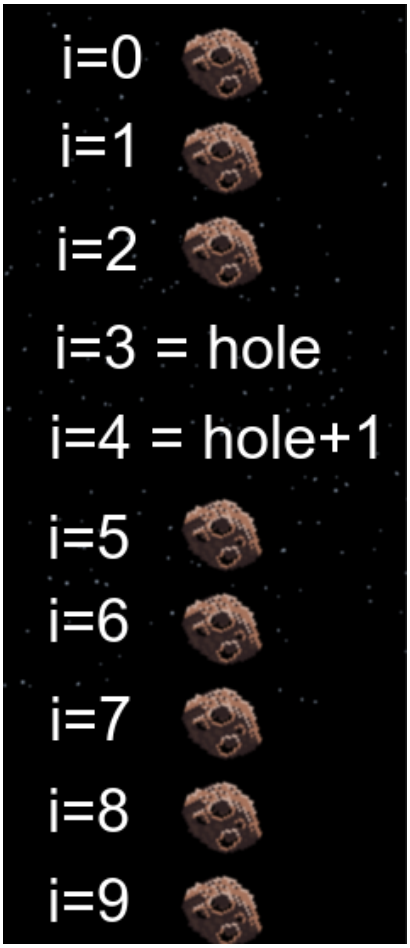
Avaruusalusta varten luodun tyhjän aukon laskenta toteutuu luomalla hole-muuttuja. Tämän muuttujan sisällä käytetään Math.floor()-metodia, joka ottaa parametreikseen Math.random() * (asteroidsNeeded - 3). Metodi palauttaa kokonaisluvun, joka on satunnaisesti luotu näytölle mahtuvien asteroidien määrästä, niin että sarakkeen ylä- ja alapäähän jää aina vähintään yksi asteroidi. Tämä kokonaisluku on hole-muuttujan arvo, ja se edustaa tyhjälle aukolle luotua paikkaa asteroidisarakkeessa (esimerkkikoodi 7).

Asteroidisarakkeen lopullinen luonti tehdään käyttämällä for-silmukkaa, jonka sisällä kerrotaan pelille, että jätetään kahden asteroidin kokoinen tyhjä aukko silloin, kun indeksi luku on sama kuin hole ja hole +1 (esimerkkikoodi 7).

```
addRowOfAsteroids() {

//Calculating how many asteroids the row needs
var asteroidsNeeded = Math.ceil(this.game.world.height /this.asteroidHeight);

//Calculating the position for the hole
var hole = Math.floor(Math.random() * (asteroidsNeeded - 3)) + 1;

//Adding the asteroids with one big hole at the position of 'hole'and 'hole+1'
for (var i = 0; i < asteroidsNeeded; i++)
     if (i != hole && i != hole + 1)
          this.addOneAsteroid(this.game.world.width, i *
          this.asteroidHeight );

//Incrasing the score with 1 when a row is created
this.score += 1;
this.labelScore.text = this.score;
}
```

Esimerkkikoodi 7. Asteroidisarakkeen luominen ja tyhjän aukon laskeminen.

Esimerkkikoodissa 7 nähdään myös, että kun uusi asteroidisarake luodaan, se päivittää pelin pistesaldoa yhdellä (kuva 6).



Kuva 6. Esimerkki addRowOfAsteroids()-funktion luomasta asteroidisarakkeesta esimerkkikoodi 7:n mukaan.

Asteroidisarakkeen luomisen jälkeen pitää pelin pystyä luomaan uusi sarake tietyin aikavälein. Useiden sarakkeiden luonnin avulla luodaan peliin asteroidisokkelo, jonka läpi pelihahmon pitää yrittää selviytyä. Tämä toteutetaan luomalla timer-muuttuja main-pelitilan create-elinkaarimetodiin. Muuttujalle annetaan käyttöön peliobjektin time.events.loop()-metodi, jolle annetaan parametreiksi aikaväli, jolla se kutsuu uutta saraketta, sekä addRowOfAsteroids-funktio (esimerkkikoodi 8).

```
this.timer = game.time.events.loop(2000, this.addRowOfAsteroids, this);
```

Esimerkkikoodi 8. Ajastin, joka kutsuu uutta saraketta 2000 millisekunnin välein.

Yksi pelin tärkeimmistä ominaisuuksista on törmäyksen havaitseminen. Se tapahtuu main-pelitilan update-elinkaarimetodissa kutsumalla peliobjektin overlap()-metodia. Metodille asetetaan parametreiksi avaruusaluksen ja asteroidin peliobjektit sekä hitAsteroid-funktio (esimerkkikoodi 9).

Overlap-metodi tarkkailee, ilmeneekö asteroidin ja avaruusaluksen kuvatiedostoilla päällekkäisyyttä. Päällekkäisyyden havaitessaan metodi kutsuu hitAsteroid-funktiota (esimerkkikoodi 9).

```
game.physics.arcade.overlap(this.ufo, this.asteroids, this.hitAsteroid, null,
this);
```

Esimerkkikoodi 9. Overlap-metodi, jolla havaitaan pelin kuvatiedostojen päällekkäisyys.

Esimerkkikoodissa 9 käytetty hitAsteroids-funktio asettaa avaruusaluksen ”elävä”-ominaisuuden epätodeksi, pysäyttää aikaisemmin luodun ajastimen, jotta peli ei enää luo uusia asteroidisarakkeita, ja tulostaa näytölle ”Game Over!”-tekstin.

Avaruusaluksen ominaisuudet

Avaruusaluksen tärkein ominaisuus on sen kyky pystyä lentämään. Avaruusalusta liikutellaan näpäyttämällä näyttöä, mikä saa aluksen ”hyppäämään” ja näin ollen pysymään lennossa.

Avaruusaluksen hyppyominaisuutta varten luotiin jump-funktio. Funktio sisältää avaruusaluksen hyppyominaisuuden, joka toteutuu nostamalla avaruusaluksen korkeutta. Funktio sisältää myös tween-siirtymän, jonka avulla pystytään avaruusaluksen kulmaa muuttamaan 20 asteen verran 100 millisekunnin aikana. Tämä tween-siirtymä elävöittää avaruusaluksen hyppyominaisuuden (esimerkkikoodi 10).

```
jump() {

      //If the ufo is already dead, don’t do nothing
      if (this.ufo.alive == false)
            return;

      //Raising the ufo vertically
      this.ufo.body.velocity.y = -350;

      //Giving animation to the jump function using tween, this changes the an-
gle of the bird 20 degrees in 100 milliseconds
```

```
        var jumpAnimation = game.add.tween(this.ufo);
        jumpAnimation.to({angle: -20}, 100);
        jumpAnimation.start();

        this.jumpSound.play();
}
```

Esimerkkikoodi 10. Avaruusalusta varten luotu jump-funktio.

Pelin loppuminen ja gameOver-pelitila

Avaruusaluksen osuessa asteroidiin kutsuu peliobjekti gameOver-pelitilaa. Tämän pelitilan sisällä pelaajalle ilmoitetaan, että peli on päättynyt, ja kerrotaan ohjeet pelin uudelleen pelaamista varten. Pelaaja pystyy aloittamaan uuden pelin näpäyttämällä näyttöä, mikä kutsuu restartGame-funktiota, joka käynnistää main-pelitilan uudestaan.

Pelin lopputulos

Aikaa Asteroid dodge -pelin kehittämiseen meni kaksi työpäivää. Insinööriyön ensimmäisen pelin tavoitteena oli luoda helppo ja yksinkertaisesti toteutettava endless runner -tyyppinen mobiilipeli. Tämä peli tulee toimimaan osana toiseen asteen opiskelijoille pidettävää mobiilikehityksen alkeiskurssia. Pelin piti olla luonteeltaan yksinkertainen, jotta opiskelijat, joilla ei ole mahdollisesti yhtään aikaisempaa ohjelmointikokemusta, pystyvät luomaan insinöörityön kaltaiset mobiilipelit kurssin tarjoamalla rajoitetulla opetusmäärällä.

Pelin päätavoite oli luoda toimiva endless runner -henkinen arcade-peli. Tämän tavoitteen toteuttamisessa onnistuttiin hyvin. Peli sisältää kaikki suunnitteluvaiheessa halutut ominaisuudet, mikä tekee pelistä yksinkertaisen, intuitiivisen ja helposti kehitettävän. Peli sisältää Phaserille tärkeimmät ominaisuudet, kuten peliobjektin luominen ja pelitilojen käyttäminen. Peli myös skaalautuu erikokoisille mobiililaitteille ja sisältää sopivan yksinkertaisen pelilogiikan. Projektia varten kehitetty peli ylsi sille asetettuihin tavoitteisiin, ja peli palvelee hyvin sille ajateltua käyttötarkoitusta.

5.2 Ball fondlers -peli

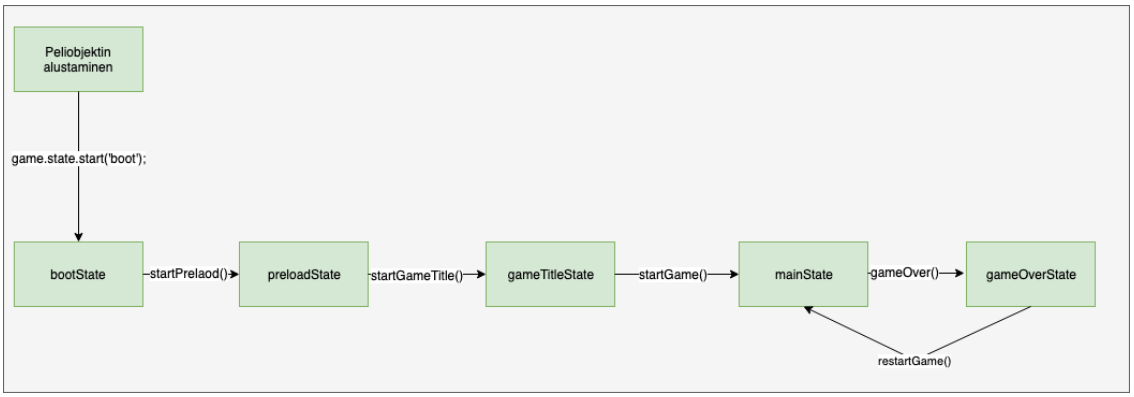
Alustus

Pelin alustetaan samalla tavalla kuin aiemmin luotu Asteroid dodge -peli. Phaser alustetaan luomalla skritpti HTML-sivulle. Skriptin sisään luodaan peliobjekti, jolle annetaan parametreiksi konfiguraatio-objekti, joka sisältää pelille oleelliset tiedot, kuten pelin tarvitsemat mitat, fysiikkamoottorin ja mitä renderöintimoottoria se tulee käyttämään. Peliobjektille lisätään myös pelitilat ja annetaan käsky käynnistää näistä ensimmäinen nimeltä ”boot”.

Pelin toimintasykli

Ennen pelitilojen luomista piti pelin käyttämille tiloille suunnitella toimintasykli, jonka mukaan pelitilat luodaan ja suoritetaan pelin edetessä (kuva 7). Pelin käyttämien tilojen toimintasykli muodostui hyvin samankaltaiseksi kuin aiemmin luodussa Asteroid dodge -pelissä.

Isoimpana erona ensimmäiseen peliin oli boot-pelitilan lisääminen. Toisin kuin Asteroid dodge -pelissä, ensimmäinen pelitila, joka kutsutaan peliobjektin alustamisen jälkeen, on boot-pelitila. Tämän tilan sisällä peliobjekti huolehtii, että peli skaalautuu oikein erikokoisille mobiililaitteille. Boot-pelitilan suorittamisen jälkeen peliobjekti kutsuu preload-tilaa, joka huolehtii kuvatiedostojen alustamisen peliä varten.



Kuva 7. Ball fondlers -pelin pelitilojen toimintasykli.

Loput pelitilat ja niiden toiminnallisuudet ovat samankaltaiset kuin ensinmäisessä pelissä. GametTitle-pelitila sisältää pelin päävalikon, ja main-pelitilan sisällä on pelilogiikka ja sille kuuluvat toiminnallisuudet. GameOver-tila sisältää tiedon pelin päättymisestä ja antaa ohjeet uudelleen pelaamista varten.

Pelin skaalautuminen ja ääni- ja kuvatiedostojen lataaminen

Ball fondlers -pelin skaalautuminen hoidetaan samanlaisella menetelmällä kuin Asteroid dodge -pelissä. Peli skaalautuu erikokoisille näytöille käyttäen Phaserin Scale Manager -objektia. Sitä käyttäen peliobjektille kerrotaan, miten sen pitää linjata Canvas-elementti näytön vaatimusten mukaan. Peliobjektille asetetaan myös skaalautumistyyppi, jonka avulla peli säilyttää aina oikean kuvasuhteen mobiililaitteen näytön koosta riippumatta.

Ainoana erona ensimmäiseen peliin on se, että siinä missä Asteroid dodge -pelissä pelin skaalautuminen suoritettiin preload-pelitilassa, Ball fondlers tekee tämän boot-pelitilassa. Tämä tehtiin siksi, että toisen esimerkkipelin pelitilojen toimintasykli olisi erilainen kuin ensimmäisessä pelissä. Pelin käyttämien ääni- ja kuvatiedostojen lataaminen suoritetaan preload-pelitilassa, mikä on sama menetelmä kuin Asteroid dodge -pelissä käytettiin aikaisemmin.

Päävalikko

Pelin päävalikko sijaitsee gameTitle-pelitilassa. Pelitila kertoo pelaajalle pelin nimen ja antaa yksinkertaiset ohjeet, jolla se kertoo pelin pelimekanismeista. Pelitila on käytännössä samanlainen kuin Asteroid dodge -pelin käyttämä päävalikko, joka esiteltiin luvussa 5.1.

Main-pelitila ja pelilogiikka

Main-pelitila sisältää pelin logiikan. Tämän pelitilan sisällä peli luo jalkapallot, havaitsee, koska niitä on kosketettu, ja huolehtii törmäyksen havaitsemisesta.

Ball fondlers -pelin ydinidea on tuhota näytölle ilmestyvät jalkapallot niihin koskettamalla ennen kuin ne ehtivät liikkua näytön vasempaan reunaan. Jalkapallot ilmestyvät näytön

oikeaan reunaan satunnaisesti generoituun kohtaan. Jokaisen tuhotun pallon jälkeen pienenee pallojen uudelleensyntymisaika ja samalla niiden liikkumisnopeus nousee.

Pelin käyttämät jalkapallot luodaan addOneFootball()-funktion sisällä. Funktio ottaa parametreikseen pelin sille asettamat x- ja y-koordinaatit. Funktion sisällä luodaan jalkapallolle satunnainen y-arvo, joka muuttuu aina, kun uutta jalkapalloa kutsutaan, ja näin luodaan uusi pallo aina uuteen sijaintiin y-akselilla. Jalkapallolle märitetään aiemmin peliin ladattu kuvatiedosto ja pallolle asetetaan myös fysiikkamoottori sekä nopeus, jolla se liikkuu pelikentän halki (esimerkkikoodi 11).

```
//givng a random y value to the football
var randomY = Math.floor(Math.random() * y) + 1;

//creating the football sprite to the given coordinations
var football = game.add.sprite(x, randomY, 'football');

//enabling physics to the football
game.physics.arcade.enable(football);

//adding velocity to the football
football.body.velocity.x = this.ballSpeed;
```

Esimerkkikoodi 11. Jalkapallon luominen ja sen y-koordinaatille luotu satunnaisarvo. Jalkapallolle otetaan käyttöön peliobjektin käyttämä fysiikkamoottori, ja sille lisätään x-akselin suuntainen kiihtyvyys.

Jalkapallon tarvitsee tietää, missä menevät pelinäkymän rajat ja että ylittäessään nämä rajat sen pitää kutsua gameOver-pelitilaa, joka kertoo pelin päättyneen. Esimerkkikoodissa 12 jalkapallolle annetaan parametri checkWorldBounds ja sille asetetaan tilaksi tosi. Seuraavaksi jalkapallolle annetaan tapahtuma onOutOfBounds, joka ottaa parametrikseen footBallIsOut-funktion, joka kutsuu gameOver-pelitilan, kun pallo ylittää pelin rajat. Näin peli loppuu pelaajan epäonnistuessa pysäyttämään jalkapallon liike, ennen kuin se osuu vastakkaiseen reunaan.

```
//creating bounds for the football and if crossed the game ends
football.checkWorldBounds = true;
football.events.onOutOfBounds.add(this.footballIsOut, this);
```

Esimerkkikoodi 12. Jalkapallolle asetetaan pelinäkymän rajat, ja ylittäessään ne se kutsuu gameOver-pelitilaa.

Kun pelaaja koskettaa jalkapalloa, sen pitää tuhoutua. Tämä toteutetaan sallimalla kosketus jalkapallo-objektille. Objektille annetaan onInputDown-tapahtuma, joka ottaa parametreikseen destroyFootball-funktion.

Esimerkkikoodissa 13 nähdään, miten destroyFootball-funktion sisällä pelaajan tuhoamien jalkapallojen pistesaldoa kasvatetaan yhdellä. Samalla pallon kiihtyvyyttä kasvatetaan ja ajastimen käyttämä aika, jolla se kutsuu uutta jalkapalloa luotavaksi, pienenee. Lopuksi pelaajan koskettama jalkapallo kutsuu desrtoy-metodia, joka tuhoaa pallon.

```
//on every destroy player will get one point
this.score ++;
this.labelScore.text = this.score;

//increasing the ball speed and reducing the timer delay
this.ballSpeed -= 10;
this.timer.delay -= 20;

//destroy the touched football
footballToBeDestroyed.destroy();
```

Esimerkkikoodi 13. Jalkapallon tuhoaminen destroyFootball-funktio sisällä.

Pelin loppuminen ja gameOver-pelitila

Pelin loppuminen ja gameOver-pelitila ovat toteutettu samalla tavalla kuin luvussa 5.1 esitellyssä Asteroid dodge -pelissä. Jalkapallon ylittäessä pelinäkymän rajat kutsuu peliobjekti gameOver-pelitilaa. Pelitila ilmoittaa pelaajalle tämän hävinneen pelin ja tulostaa pistesaldon. Pelaaja pystyy aloittamaan uuden pelin näpäyttämällä pelinäkymän ruutua, jolloin peliobjekti kutsuu uudestaan main-pelitilaa ja aloittaa uuden pelisession.

Pelin lopputulos

Ball fondlers -peli saavutti sille asetetut tavoitteet. Peli suunniteltiin yksinkertaiseksi tap to kill -peliksi, jonka kehittäminen on mahdollista toisen asteen opiskelijoille, jotka toimivat tämän projektin kohderyhmänä. Aika, joka kului tämän pelin toteuttamiseen, oli yksi työpäivä. Pelin yksinkertaisempi luonne verrattuna Asteroid dodge -peliin teki pelin kehittämisestä nopeampaa. Myös Phaserin ominaisuudet olivat tekijälle tutummat kuin ensimmäisen pelin kehitysvaiheessa.

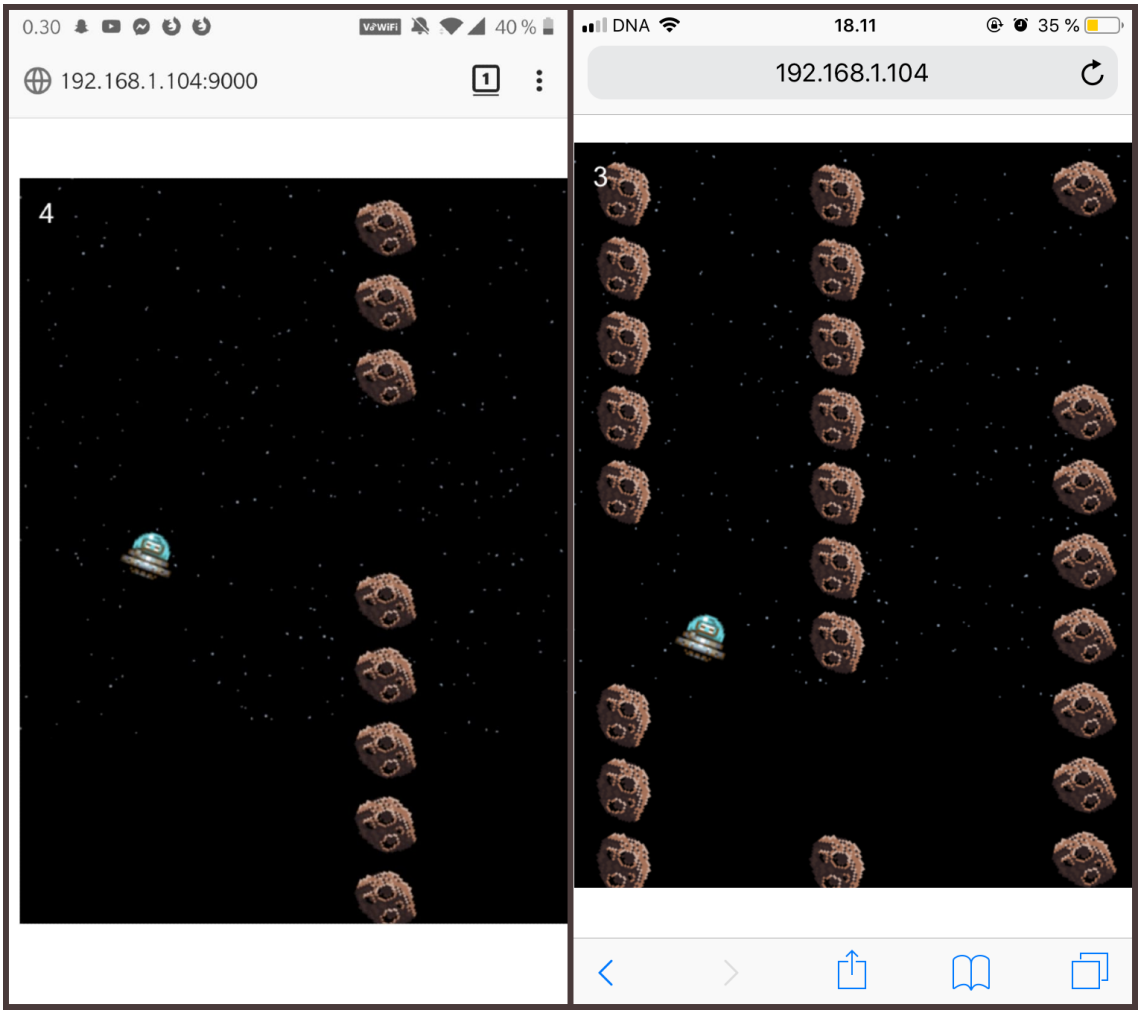
5.3 Pelien testaus

Insinöörityötä varten luotujen mobiilipelien ensimmäinen testausvaihe suoritettiin projektin kehitysympäristössä käyttämällä Brackets-tekstieditorin live-lataustoimintoa, joka salli pelien testaamisen selaimessa reaaliaikaisesti. Tämä toiminto auttoi kehitysvaiheessa huomattujen ongelmien ratkomisessa ja teki kehittämisestä paljon nopeampaa. Pelejä testattiin Google Chrome -selaimella, koska sen kehittäjätyökalujen avulla pystyttiin selaimella jäljittelemään erikokoisten mobiililaitteiden näyttöjä ja testaamaan näin pelien skaalautumista.

Mobiilipelien luomisen jälkeen oli aika suorittaa projektin toinen testausvaihe. Tavoitteena oli testata insinöörityötä varten luotujen mobiilipelien toimivuutta oikeissa mobiililaitteissa käyttäen erilaisia puhelinselaimia. Tätä varten piti peleille luoda palvelin käyttämällä Browsersync-automaatiotyökalua. Sen avulla projektille luodaan palvelin, joka sallii pelien testaamisen fyysisillä mobiililaitteilla.

Mobiililaitteet, joissa testattiin projektin pelejä, olivat iPhone 6s, joka käytti iOS-käyttöjärjestelmää ja selaimena Safaria, sekä OnePlus 5, joka käytti Android-käyttöjärjestelmää ja selaimina Mozilla Firefoxia sekä Google Chromea. iPhone 6s-puhelimessa on 4,7 tuuman näyttö, kun taas OnePlus 5 -puhelimessa on 5,5 tuuman näyttö. [35; 36.]

Ensimmäisenä pelinä testattiin Asteroid dodge -peliä (kuva 8). iPhonella peli skaalautui moitteettomasti, mutta peliominaisuuksissa oli havaittavissa pientä viivettä. Kuvataajuus selaimessa oli hieman matala, mikä sai pelin vaikuttamaan hitaalta. Myös kosketuksen havaitsemisessa oli pientä viivettä. OnePlus-puhelimella ensin Firefox-selaimessa testattaessa peli skaalautui hyvin, ja myös pelin kuvataajuus oli korkea, mikä sai pelin toimimaan jouhevasti. Kosketuksen havaitseminen toimi hyvin ja loi nopean liikkumisominaisuuden pelin hahmolle. Samankaltaiset tulokset saavutettiin myös Chrome-selaimella.



Kuva 8. Asteroid dodge -pelin skaalautuminen eri mobiililaitteissa. Vasemmalla kuvassa OnePlus 5 ja 5,5 tuuman näyttö, oikealla iPhone 6s ja 4,7 tuuman näyttö.

Seuraavaksi testattiin Ball fondlers -peli (kuva 9). Ensin peliä testattiin iPhone 6s:llä, jossa se suoriutui hyvin. Peli skaalautui oikein näytön koon mukaan, ja kuvataajuus oli korkea, mikä teki pelin toimivuudesta nopean. OnePlus 5 -puhelimessa peli skaalautui oikein ja kuvataajuus oli korkea molemmissa, Firefox- ja Chrome-selaimessa. Molemmissa mobiililaitteissa kosketuksen havaitseminen jalkapalloa koskettaessa toimi tarkasti ja loi mukavan ja sopivan haastavan pelikokemuksen.

Kuva 9. Ball fondlers -pelin skaalautuminen eri mobiililaitteissa. Vasemmalla kuvassa OnePlus 5 ja 5,5 tuuman näyttö, oikealla iPhone 6s ja 4,7 tuuman näyttö.

Projektin toisen testivaiheen jälkeen todettiin, että molemmat pelit skaalautuivat hyvin erikokoisilla näytöillä ja erilaisilla selaimilla. Tämä osoittaa peleissä käytettävän Scale Manager -objektin hyvät ominaisuudet ja yksinkertaisen implementoinnin.

Asteroid dodge -pelin suorituskyvyssä ilmeni eroja eri mobiililaitteilla. OnePlus 5 -puhelimella peli toimi moitteettomasti, mutta iPhone 6s:llä huomattiin suoristustehossa ongelmia, mikä teki pelin pelaamisesta hankalaa. Syy heikompaan pelattavuuteen iPhone-puhelimella on puhelimen suoritustehon rajallisuus.

Ball fondlers -peli suoriutui molemmilla mobiililaitteilla hyvin. Pelissä ei ilmennyt ollenkaan suoritustehosta riippuvia ongelmia, ja kosketuksen havaitseminen toimi tarkasti.

6 Yhteenveto

Insinöörityössä tutkittiin ja kartoitettiin erilaisia pelikehyksiä ja löydettiin niistä sopiva, jonka avulla projektissa kehitettiin kaksi yksinkertaista mobiilipeliä. Asiakkaan tavoitteena oli käyttää insinöörityötä osana toisen asteen opiskelijoille pidettävää web-kehityksen alkeiskurssia, minkä vuoksi projektia varten kehitettyjen mobiilipelien piti olla yksinkertaisia, mutta samalla tarpeeksi houkuttavia opiskelijoiden mielenkiinnon ylläpitämiseksi. Insinöörityön haasteita olivat sen rajattu työaika sekä sopivien työkalujen löytäminen niin, että sen toteutus pysyi mahdollisimman yksinkertaisena.

Insinöörityön mobiilipelien kehittämiseen valittiin Phaser-sovelluskehys. Se toimi suunnitellusti, ja sillä pystyttiin toteuttamaan kaikki pelien suunnitteluvaiheessa asetetut tavoitteet helposti. Peleistä tuli yksinkertaisia, skaalautuvia ja ne toimivat hyvin monien eri mobiililaitteiden selaimissa. Phaserin käyttö osoittautui hyväksi valinnaksi sen sijaan, että työssä olisi käytetty muita, yksinkertaisempia JavaScript-kirjastoja. Phaser sisälsi paljon hyviä toiminnollisuuksia, mikä teki mobiilipelien kehittämisestä nopeaa ja helppoa. Projektia varten kehitettyjen mobiilipelien jälkeen on helppo todeta, että Phaser sopii hyvin selainpohjaisten mobiilipelien kehittämiseen.

Insinöörityön jatkokehityksenä voisi projektia varten luotujen Phaser-pelien toteuttamista yrittää pelkällä JavaScriptillä. Näin pystyttäisiin havainnollistamaan, mitkä ovat Phaser-sovelluskehyksen konkreettiset hyödyt verrattuna pelkkään JavaScriptiin.

Insinöörityön tekijällä ei ollut aikaisempaa pelikehityskokemusta Phaserillä. Sovelluskehys toimi moitteettomasti, ja lopputuloksena peleistä tuli toimivia, alustariippumattomia ja mielenkiintoisia kehittää. Kaikki suunnitteluvaiheessa halutut toiminnallisuudet saatiin tehtyä, ja Phaserin käyttö projektissa toimi halutulla tavalla.

Lähteet

1 Van der Spuy, Rex. 2015. Learn Pixi.js. E-kirja. Apress.

2 EaselJS. Verkkoaineisto. CreativeJS. <https://www.createjs.com/easeljs>. Luettu 10.10.2018.

3 Three.js. Verkkoaineisto. ThreeJS. <https://threejs.org/>. Luettu 10.10.2018.

4 PlayCanvas. Verkkoaineisto. PlayCanvas. <https://playcanvas.com/features>. Luettu 10.10.2018.

5 Davey, Richard. 2013. How to learn Phaser HTML5 game engine. Verkkoaineisto. Envantotuts. <https://gamedevelopment.tutsplus.com/articles/how-to-learn-the-phaser-html5-game-engine--gamedev-13643>. Luettu 12.10.2018.

6 Davey, Richard. 2013. Phaser 1.0 and the journey we took to get there. Verkkoaineisto. Photom Storm Ltd. <http://www.photonstorm.com/phaser/phaser-1-0-and-the-journey-we-took-to-get-there>. Luettu 12.10.2018.

7 Phaser CE. Verkkoaineisto. Photom Storm Ltd. <https://phaser.io/download/phaserce>. Luettu 12.10.2018.

8 Phaser 3. Verkkoaineisto. Photom Storm Ltd. <https://phaser.io/phaser3>. Luettu 12.10.2018.

9 Canvas API. Verkkoaineisto. Mozilla. <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API>. Luettu 14.1.2019.

10 Basic usage of canvas. Verkkoaineisto. Mozilla. <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_usage>. Luettu 14.1.2019.

11 Deveria, Alexis & Schoors, Lennart. Can I use. Verkkoaineisto. <https://caniuse.com/>. Luettu 14.1.2019.

12 Getting started with WebGL. Verkkoaineisto. Mozilla. <https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Getting_started_with_WebGL>. Luettu 15.1.2019.

13 Making your first Phaser 3 game. Verkkoaineisto. Photom Storm Ltd. <https://phaser.io/tutorials/making-your-first-phaser-3-game>. Luettu 15.1.2019.

14 Houwens, Byron. 2018. An introduction to TypeScript: Static typing for the web. Verkkoaineisto. SitePoint Pty. <https://www.sitepoint.com/introduction-to-typescript/>. Luettu 16.1.2019.

15 Dubey, Kumar. 2018. Difference between TypeScript and JavaScript. Verkkoaineisto. Geeksforgeeks. <https://www.geeksforgeeks.org/difference-between-typescript-and-javascript>. Luettu 16.1.2019.

16 De Kinder, Bob. 2016. How to create fun and interactive JavaScript games using Phaser.io. Verkkoaineisto. Intracto. <https://blog.intracto.com/create-fun-and-interactive-games-with-javascript-using-phaser.io>. Luettu 9.11.2018.

17 Impact physics. Verkkoaineisto. Photom Storm Ltd. <https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Impact.html>. Luettu 25.2.2019.

18 Hadley, Michael. 2018. Modular game worlds in Phaser 3 (Tilemaps #4) – Meet Matter.js. Verkkoaineisto. Medium. <https://itnext.io/modular-game-worlds-in-phaser-3-tilemaps-4-meet-matter-js-abf4dfa65ca1>. Luettu 25.2.2019.

19 Löw, Andreas. How to create physics shapes for Phaser 3 and Matter JS. Verkkoaineisto. CodeAndWeb GmbH. <https://www.codeandweb.com/physicseditor/tutorials/how-to-create-physics-shapes-for-phaser-3-and-matterjs>. Luettu 25.2.2019.

20 Monroy, Josh. 2017. Phaser fundamentals: Using states in Phaser. Verkkoaineisto. Mobirony. <https://www.joshmorony.com/phaser-fundamentals-using-states-in-phaser/>. Luettu 13.12.2018.

21 Feronata, Emanuele. 2014. Phaser tutorial: Understanding Phaser states. Verkkoaineisto. Feronata. <https://www.emanueleferonato.com/2014/08/28/phaser-tutorial-understanding-phaser-states/>. Luettu 13.12.2018.

22 Using states in Phaser.js JavaScript game developement. 2015. Verkkoaineisto. Perplexing Technology. <http://perplexingtech.weebly.com/game-dev-blog/using-states-in-phaserjs-javascript-game-developement>. Luettu 13.12.2018.

23 Davey, Richard. 2018. How scenes work. Verkkoaineisto. Photom Storm Ltd. <https://phaser.io/phaser3/contributing/part5>. Luettu 21.2.2019.

24 Phaser state. Verkkoaineisto. Photom Storm Ltd. <http://phaser.io/docs/2.6.2/Phaser.State.html>. Luettu 13.12.2018.

25 Sprite. Verkkoaineisto. Photom Storm Ltd. <https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Sprite.html>. Luettu 13.12.2018.

26 Haynes, Jerra. 2018. A real person's guide to Phaser 3: Or, how I learned to stop worrying and love the gun, part 1. Verkkoaineisto. Medium. <https://medium.com/@jerra.haynes/a-real-persons-guide-to-phaser-3-or-how-i-learned-to-stop-worrying-and-love-the-gun-part-1-9cc6361f377c>. Luettu 20.1.2019.

27 Tween. Verkkoaineisto. Photom Storm Ltd. <https://photonstorm.github.io/phaser3-docs/Phaser.Tweens.Tween.html>. Luettu 13.12.2018.

28 Phaser Input. Verkkoaineisto. Photom Storm Ltd. <https://phaser.io/docs/2.6.2/Phaser.Input.html>. Luettu 13.12.2018.

29 Cantuni, Rubens. 2018. Designing a mobile arcade game: a case study. Verkkoaineisto. UX Collective. <https://uxdesign.cc/designing-a-mobile-arcade-game-a-case-study-253c59d60f75>. Luettu 3.3.2019.

30 Overview of 10 most popular mobile game genres. 2017. Verkkoaineisto. Weebly. <http://r-stylelab.weebly.com/blog/overview-of-10-most-popular-mobile-game-genres>. Luettu 4.3.2019.

31 Momoda, Jerry. 2013. Endless runner games: Evolution and future. Verkkoaineisto. Momoda. <http://jerrymomoda.com/analysis-endless-runners/>. Luettu 4.3.2019.

32 Brackets. Verkkoaineisto. Adobe. <http://brackets.io/>. Luettu 18.2.2019.

33 Browsersync. Verkkoaineisto. Jh. <https://www.browsersync.io/>. Luettu 18.2.2019.

34 Patel, Devan. 2016. 4 JavaScript design patterns you should know. Verkkoaineisto. Scotch.io. <https://scotch.io/bar-talk/4-javascript-design-patterns-you-should-know>. 18.2.2019.

35 iPhone 6s. Verkkoaineisto. Gsmarena. <https://www.gsmarena.com/apple_iphone_6s-7242.php>. 25.3.2019.

36 OnePlus 5. Verkkoaineisto. OnePlus. <https://www.oneplus.com/fi/5/specs>. 25.3.2019.