TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Indro Kottise 175858IDDR

Reklaammängu loomine Phaser 3 raamistikuga Ekspress Meedia näitel

Diplomitöö

Juhendaja: Toomas Lepikult
Ph. D.
Ivar Krustok
Rakenduskõrgharidus

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Indro Kottise

07.01.2021

Annotatsioon

Käesoleva töö peamiseks eesmärgiks oli luua kahemõõtmeline reklaammäng *Phaser* 3 raamistikuga, milles kasutatakse reaalajas uueneva edetabeli loomiseks Google *Firebase* reaalaja andmebaasi. Lisaks loodi taaskasutatavaid programmifragmente, mida on võimalik kasutada edaspidi uute mängude loomisel.

Töös analüüsiti ning võrreldi omavahel erinevaid mänguraamistike ja reaalaja andmebaase, valiti sobivad vahendid reklaammängu loomiseks, uuriti mängutaseme ja sellega seotud objektide ning mehhanismide loomist. Tulemuse registreerimiseks ning kuvamiseks uuriti reaalaja andmebaasi kasutamist.

Tulemusena valmis veebimäng, mida on võimalik mängida mobiilis ning mille alglaadimise maht vastab nõuetele. Mängus loodud funktsioonid, näiteks vormi kasutamine, andmebaasiga ühendamine, suhtlemine ja struktuuri reegleid saab kasutada tulevastes väiksemamahulistes projektides, säästmaks aega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 28 leheküljel, 3 peatükki, 12 joonist, 4 tabelit.

Abstract

Creation of Advertisement Game with Phaser 3 Framework: the Case of Ekspress Meedia

The purpose of the thesis was to create a two-dimensional advertisement game for Ekspress Meedia. The game uses real-time database provided by Google Firebase which allows for scoreboard data to be updated instantly when a new entry is posted and meets the requirement of being in the top 7. Different functions were created to speed up the development of advertisement games and they can be used as examples or reusable methods in future games with little modifications.

The analysis included comparisons of different frameworks for game creation and real-time databases that had the option that the third-party website hosted it. A suitable framework and real-time database were chosen to meet the demands of Ekspress Meedia. The creation of the game level and objects relating to it were examined and different mechanics for the gameplay were added. For submitting the result to the scoreboard and updating it instantly, real-time database usage and hosting prices were researched.

As a result, an online game that can be displayed on an advertisement placement was created. The requirements of file size limit were met and the game is resizeable so it can be shown in full width on mobile devices.

Functions created in the advertisement game, for example, usage of register form, connecting with the database on demand, posting and getting data from it, and database rules created for this game are set as an example for future games. Database functions can also be used in future projects that are small and demand fast setup to save development time, for example different quizes or polls, where users are allowed to react with different objects.

The thesis is in Estonian and contains 28 pages of text, 3 chapters, 12 figures, 4 tables.

Lühendite ja mõistete sõnastik

API	*Application programming interface*, rakenduse programmeerimise liides
arcade	arkaad
bitmap	Bittraster, kahemõõtmeline massiiv näitamaks atribuute [1].
canvas	Lõuend, element, mille abil saab joonistada graafikat ja animatsioone [2].
CLI	*Command-line interface*, käsurea liides
DOM	*Document Object Model*, programmeerimise liides, mille abil saab manipuleerida veebidokumendiga [3].
event emitter	sündmuse väljasaatja
frame rate	kaadrisagedus
JSON	*JavaScript Object Notation*, süntaks serialiseerimaks andmeid [4].
listener	kuulaja
loop	silmus
MMOG	*Massively Multiplayer Online Game*, massiivne mitme mängijaga veebimäng [12]
MUD	*Multiuser dungeon*, mitme mängijaga vangikoobas, mängu nimetus [12].
SDK	*Software development kit*, tarkvara arenduse komplekt
sprite	Sprait, visuaalne olem, mida liigutades tekib animatsioon [5].
spritesheet	Spraidi kogum ühel pildil [6].
SQL	*Structured Query Language*, struktureeritud päringu keel, kasutatakse andmebaasiga suhtlemisel [7].
touch	puudutus
tween	Protsess, mille abil luuakse vahepealseid kaadreid piltide vahel, loomaks kujutuse liikumisest [8].
WebGL	*Web Graphics Library*, teek, mida kasutatakse kahe- ja kolmemõõtmeliste graafikate joonistamiseks [9].
XML	*Extensible Markup Language*, laiendatav märgistuskeel, kasutatakse andmete hoiustamiseks [10].

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Veebimänge on võimalik olnud mängida veebibrauseris juba eelmisest sajandist. Mängude üks eesmärk on pakkuda lõbusat ajaviidet ning tänapäeval on raske leida noort inimest, kes ei ole nendega kokku puutunud. Reklaammäng on üks veebimängu vormidest, mille ülesanne on teha reklaami millelegi ja samal ajal pakkuda ajaviidet.

Reklaammängu eesmärk on propageerida toodet, teenust või ettevõtet ning mõjutada selle kaudu tarbijat. Ekspress Meedial puudub standardiseeritud lahendusviis reklaammängude ja nendes kasutatavate andmebaaside loomiseks, mis võimaldaks reaalajas andmete muutmist, et mängu loodav edetabel uueneks igal ajahetkel, mitte ainult edetabelit avades. Töö eesmärgiks on leida lahendus, mille abil kiirendada mängu loomise protsessi, luua taaskasutatavaid elemente, mida saaks kasutada ka teistes mängudes, ja leida sobiv reaalaja andmebaas, mida haldab kolmas osapool. Töös antakse ülevaade veebimängude ajaloost ning *Flash Player*-ist, analüüsitakse ja võrreldakse omavahel mänguraamistike *Phaser* 3, *Pixi.js*, *ImpactJS* funktsionaalsust ja võimalusi ning reaalaja andmebaase Google *Firebase*, *Parse*, *Kuzzle*.

Töö esimeses osas kirjeldatakse veebimängude ajalugu ning *Flash Player*-it. Teises osas võrreldakse omavahel kolme mänguraamistikku, mida kasutatakse JavaScript programmeerimiskeeles ja kolme reaalaja andmebaasi. Kolmandas osas analüüsitakse reklaammängu loomisprotsessi ja tulemusi.

2 Taust

2.1 Veebimäng

Veebimänguks nimetatakse mängu. mida saab mängida veebis kas üksi või mitmekesi. Veebimängude juured ulatuvad 1970-ndatesse, kui Ameerika Ühendriikides ühendati ülikoolid ARPANET-iga, mida omakorda peetakse eelkäijaks tänasele Internetile [11]. 1980-ndatel ühendati ARPANET Essexi Ülikooliga, kus sealsed kaks tudengit lõid teksti baasil seiklusmängu, mida nad kutsusid nimetusega „multiuser dungeon" ehk MUD-ks. *Online* mängurluse alguseks nimetatakse aga seda hetke, kui esimene mängu loomisega mitteseotud inimene ühendus ARPANET-i kaudu. MUD-i hakati kiiresti edasi arendama, lisati graafikat, vestlusvõimalus ja mängijate grupid. Need omadused kandusid edasi MMOG-sse (*Massively Multiplayer Online Game*). Seda kõige paremini ilmestavaks näiteks on 2004. aastal ilmunud *World of Warcraft*, mille fenomen tõi kokku mängima miljoneid tavamängureid. Paljudel neist polnud kusjuures üldse varasemat kokkupuudet MMOG-ga.[12]

Esimesed veebibrauseri mängud said alguse *FutureSplash Animator*-ist, mis loodi 1996. aastal Macromedia poolt. Enne selle loomist oli ainukeseks viisiks veebibrauseris animatsiooni mängida Java programmeerimiskeele abil, kuid antud lahendus oli aeglane. Kui Netscape tuli välja API pistakuga, siis suurenes ka kiirus. 1996 detsembris vahetus *FutureSplash Animator* omanik ning selle nimeks sai *Macromedia Flash* [13]. Esimesed *Flash*-i mängud olid sageli klassikute, näiteks *Pac-Man* ja *Frogger* taasloomised ning Interneti kiiruse tõustes said kuulsaks nimed nagu *Bowman ja Max Dirt Bike* [14]. *Flash*-mängude populaarseimaks veebileheks peetakse peamiselt Miniclipi, mis asutati 2001. aastal 40 000 naelase eelarvega [15]. *Flash*-mänge pakkuvate veebilehtede nimistusse kuuluvad ka näiteks Newgrounds ning AddictingGames [16].

2.1.1 Flash Player

Flash Player avaldati 1. jaanuaril 1996 firma Macromedia poolt tasuta jagatava pistikprogrammina, mis aitas neil kiiresti turuosa suurendada. Adobe omandas Macromedia aastal 2005 [17]. *Flash Player*-i tööpõhimõte seisneb SWF failide käitamises, ühtlasi toetab see vektorgraafikat, 3D graafikat, audiot ja videot. Antud tarkvara oli kunagi populaarne formaat veebimängude, animatsioonide ning graafiliste kasutajaliideste loomiseks, kuid veebibrauserite uute standardite tõttu (näiteks HTML5 ja WebGL) on kadunud vajadus kolmanda osapoole pistikprogrammide järele [18]. Adobe teatas 2017 juulis, et lõpetab 31. detsembrist 2020 *Flash Player*-i levitamise ning uuendamise. Otsus avaldati koostöös Apple, Facebook, Google, Microsoft ja Mozillaga, lisaks eemaldatakse kõik ametlikud allalaadimise võimalused ning *Flash*-i töötamine blokeeritakse kõigil seotud veebisaitidel [19].

3 Mänguloomise raamistike ning reaalaja andmebaaside analüüs ning võrdlus

Kahemõõtmeliste mängude arendamiseks leidub tänapäeval mitmeid tasuta kättesaadavaid spetsiaalseid mänguloomise raamistikke, millega saab üles ehitada HTML5-le baseeruvaid mänge. Järgnevalt analüüsitakse *Phaser* 3, *Pixi.js* ja *ImpactJS* raamistike, mille kõigi ühiseks jooneks on programmeerimiskeel JavaScript. Lisaks sisaldavad nad kõik füüsikamootorit ning nendega saab luua 2D mänge. Uuritakse ka reaalaja andmebaaside Google *Firebase*, *Parse* ja *Kuzzle* abil edetabelite loomist.

3.1 Phaser 3

Phaser avalikustati blogipostituses 2013 aprillis ning esimene versioon lasti välja sama aasta septembris [20]. Graafika kuvamiseks kasutati *Pixi.js* teeki [21]. *Phaser* 3 avalikustati 13. veebruaril 2018 ning sealsed muutused olid suuremahulised. Iga element ehitati nullist ümber ning hakati kasutama muudetud WebGL renderdajat, täitmaks moodsate kahemõõtmeliste mängude nõudmisi. Parandati dokumentatsiooni ning loodi mitmeid näidiseid kuvamaks võimalusi, kuidas antud raamistiku kasutada saab [22]. *Phaser* 3 mänge saab luua kas JavaScripti või TypeScriptiga. Renderdamiseks kasutatakse WebGL-i, kui veebibrauser seda toetab, vastasel juhul taandutakse *canvas* renderdamisele. Mänge saab kolmanda osapoole tööriistadega kompileerida nii iOS kui ka Android operatsioonisüsteemidele. Raamistik toetab staatilisi ja dünaamilisi pilte, nende animeerimist, kasutaja sisendite kuulamist ning 3 erinevat füüsikamootorit [23].

3.2 Pixi.js

PixiJS eesmärk on pakkuda kiiret kahedimensioonilist teeki, mis töötaks kõikidel seadmetel nii, et kasutaja ei peaks süvenema seadmete ühildatavusega seotud nüanssidesse. *Pixi* kasutab samuti WebGL renderdamist nagu *Phaser* ning taandub

HTML5 *canvase* renderamisele, seda juhul, kui esimest varianti ei toetata. *PixiJS* soovitatakse kasutada, kui eesmärk on luua graafiliselt rikast lahendust või HTML5 mänge [24]. *Pixi.js* sai alguse veebruaris 2013 Matt Groves poolt ning seda kasutavad ka suured firmad nagu Disney, BBC, McDonalds ja teised [25]. Viimane suurim versioon 5 tuli välja 2019. aasta aprillis [26]. *Pixi.js* raamistiku kohta on ka palju erinevaid näiteid ja õpetusi, nii mängu loomise, kui ka erinevate veebi meedialahenduste kohta [27],[28].

3.3 ImpactJS

ImpactJS nimetatakse üheks testitumaks HTML5 mängumootoriks, mille esialgne väljalase toimus aastal 2010. See on disainitud töötama kõikidel seadmetel ning kasutab *Ejecta* raamistikku, et mänge iOS rakendustepoes avaldada. *ImpactJS* sisaldab endas taseme redigeerijat ning silumistööriistu [29],[30]. Alustamiseks leidub antud mängumootori kodulehel videoõpetusi ning viidet kahele inglise keelsele raamatule, mis sisaldavad detailseid õpetusi *ImpactJS*-ga mängude arenduse, silumise, analüütika, edetabeli loomise, arenduskeskkonna ülesseadmise ning mängu publitseerimise, ka Facebook-i, kohta [31]. Viie ja poole aastaga ei ole suuremaid ImpactJS versioone ilmunud, viimane versioon 1.24 ilmus 28. juuli 2014 [32].

3.4 Reaalaja andmebaasid

Veebimängudega on alatiselt kaasas käinud edetabelid, kuhu inimesed võimalusel enda tulemused kirja saavad panna, ning seeläbi end teiste mängijatega võrrelda või konkureerida. Reaalaja andmebaasiks nimetatakse andmebaasi süsteemi, mis kasutab reaalajas töötlemist pidevalt muutuvate töökoormuste käsitlemiseks [33]. Järgnevalt analüüsitakse kolme reaalaja andmebaasi ning nende sobivust edetabeli loomiseks.

3.4.1 Google Firebase

Firebase sai alguse firmast nimega Envolve, mis asutati aastal 2011. Nende eesmärk oli pakkuda sidusvestluse integratsiooni, mis toimiks reaalajas. Varsti märgati, et nende toodet kasutati andmete saatmiseks, mis ei olnudki sageli vestlusega seotud. Nimelt olid mängude arendajad hakanud ära kasutama vestluse API-t, et saata teistele kasutajatele infot mängija "eludest". Peale juhtunut otsustati vestluserakendus ja reaalaja arhitektuur

üksteisest eraldada [34]. *Firebase Realtime Database* eesmärgiks ongi sünkroniseerida andmeid erinevate seadmetega ning salvestada need *Firebase*-i enda pilve [35]. *Firebase* omandati Google poolt 2014. aastal ning liitumise eesmärgiks oli teenuse skaleerimine [36].

3.4.2 Parse

Parse oli mobiili tagaraamistiku platvorm, mille omandas Facebook 2013. aastal [37]. Firma sai alguse nelja asutaja poolt ning nende esmaseks eesmärgiks oli pakkuda tagaraamistiku tööriistu mobiiliprogrammide arendajatele, näiteks autentimiseks ja andmete salvestamiseks [38]. Mainitud tööriistu kasutas 2012. aastal 20 000 arendajat, ning antud number kasvas 40% igas kuus [39]. *Parse Server* kasutab MongoDB andmebaasi ning seda peab majutama mujal [40]. 2016. aastal teatati *Parse* sulgemisest, ning platvormi lähtekood muudeti avatuks, et kasutajad saaksid migreerida rakendused muule majutusele või platvormile [41].

3.4.3 Kuzzle

Kuzzle on avatud lähtekoodiga lahendus, millel on skaleeruv server, kolme protokolliga API ja administreerimist võimaldav konsool [42]. Andmete hoidmine, pärimine ning otsinguga seotud toimingud on lahendatud *Elasticsearch* abiga. Tagaraamistiku saab muuta vastavalt vajadusele, näiteks käivitada andmete muutumisega seotud toiminguid, luua meetodeid avaliku API jaoks või lisada erinevaid autentimise viise. *Kuzzle* toetab programmeerimiskeeli nagu näiteks JavaScript, PHP, Java, C# ja Dart [43].

3.5 Mänguloomise raamistike ja reaalaja andmebaaside võrdlus

Eelnevates peatükkides uuriti raamistike, mis võimaldavad luua 2D mänge ning sisaldavad füüsikamootorit, nende ajalugu, sobivust erinevate platvormide ja seadmetega. Analüüsiti, milliseid erinevaid tööriistu need sisaldavad ning millised on visuaali renderdamise võimalused. Tabelis (Tabel 1) esitatakse mänguraamistikega seotud detaile, näiteks võrreldakse teekide kaalu, viimast suuremat väljalaset, populaarsust ja ka kasutajate tagasisidet.

Tabel 1.Mänguraamistike andmete võrdlus

	Phaser 3	**Pixi.js**	**ImpactJS**
Esmane väljalase	2013 september	2013 veebruar	2010
Võrreldav versioon	3.24.1	v5	1.24
Programmeerimiskeel	JavaScript, TypeScript	JavaScript	JavaScript
Viimane väljalase	3. aprill 2018 [44]	3. aprill 2018 [44]	28. juuli 2014 [44]
Populaarsus	64% [44]	58% [44]	85% [44]
Reiting	4.5 / 5 [44]	5 / 5 [44]	3.5 / 5 [44]
Teeki kaal	200 kilobaiti [45]	77.9 kilobaiti [45]	125 kilobaiti [32]

Järgnevalt võrreldakse erinevaid mänguraamistike funktsionaalseid omadusi viimaste versioonide põhjal, nagu näiteks füüsikamootoreid, visuaalsete elementide loomiste võimalust ning heli (Tabel 2).

Tabel 2. Mänguraamistike võimaluste, tööriistade, füüsikamootorite, loogika ja heli võrdlus

	Phaser 3	**Pixi.js**	**ImpactJS**
Visuaalsed võimalused	Staatilised pildid, *Sprite*'d, animatsioonid (*tweens)*, osakesed (*particles)*, kaamerad, *fonts, bitmap fonts, tilemap*'id, sisseehitatud visuaalne skaleeritavus [46]	Staatilised pildid, *Sprite*'d, animatsioonid, video mängimise võimalus, tekst*, bitmap text, webfont, masks,* filtrid, *mesh,* filtrid [28]	Staatilised pildid, *Sprite*'d, animatsioonid, *Animation sheets*, *onts*, *bitmap fonts,* 2D kaardid [31]
Raamistikuga seotud mängutaseme loomise tööriistad	Puudub [46]	Pixi-tilemap [47]	Weltmeister [31]
Füüsikamootor	Arcade, Impact, Matter [46]	Pixi-poly, pixi-tiled [28]	Impact [31]
Loogika	Eellaadimine, grupid, sisendi kuulamine, *collisions,* taimer, sündmused,	Eellaadimine, grupid, sisendite kuulamine, *collisions,* taimer, sündmused,	Eellaadimine, grupid, sisendi kuulamine, *coillisions,* taimer

	Phaser 3	Pixi.js	ImpactJS
	pistikprogrammid, stseenid [46]	pistikprogrammid [28]	[31]
Heli	Ogg, MP3, SID, audio sprites [46]	Ogg ja MP3 [48]	Ogg ja MP3, võimalus luua esitusloendeid

Eelnevalt analüüsiti ka kolme reaalaja andmebaasi ajalugu, järgnevalt esitatakse nende teenuste hinnakiri. *Parse*-l endal ei ole majutuse võimalust, vaid see vajab kolmanda osapoole serverit. Näiteks on võetud Zone'i virtuaalserveri Pakett II, mis vastab *Parse* nõuetele (Tabel 3) [40],[49].

Tabel 3. Reaalaja andmebaaside pakutavate teenuste võrdlus ja hinnakiri

	Google Firebase	Parse	Kuzzle
Tasuta kasutamise võimalus ja majutusliik	Jah, Google pakub majutust [50]	Jah, majutamisega peab tegelema kasutaja [40]	Jah, majutamisega peab tegelema kasutaja [51]
Tasuta majutuse limiidid	100 lubatud samaaegset ühendust, 1 GB hoiustatavat mahtu, 10 GB andmemahtu kuus [50]	-	-
Tasuline kasutamise võimalus ja majutusliik	Jah, Google pakub majutust [50]	Jah, majutus Zone poolt [49]	Jah, Kuzzle pakub majutust [51]
Makseliik	Kuumakse, summa oleneb andmete suurusest [50]	Fikseeritud kuumakse 14,40 € [49]	Kuumakse alates $1200 [51]
	5$ 1 GB hoiustatava mahu kohta, $1 1 GB andmemahu kohta [50]		
Andmebaasi skaleeritavus	Jah [50]	Ei, paketiga piiritletud. [49]	Jah [51]

4 Reklaammängu arendus

Eelnevas peatükis analüüsitud vahendeid kasutatakse järgnevalt uue reklaammängu loomiseks. Mängul on võimalik kujutada reklaamitava firma sümboolikat, koguda e-posti aadresse ning luua mängijatele edetabel. Mängu loomisel arvestatakse tingimustega, et mäng peab olema mängitav mobiilis ning mängija jaoks kergesti arusaadav, alglaadimise andmemaht ei tohi ületada 100 kilobaiti ja mäng peab sisaldama edetabelit, mis uueneb reaalajas.

4.1 Mängu olemus

Mäng on mõeldud mobiilis mängimiseks ning mängija eesmärgiks on võimalikult kaua „kõksides" jalgpalli üleval hoida ning koguda võimalikult suur punktisumma. Iga maha kukkunud pall võtab maha ühe elu. Kui mängu on loodud mitu palli, siis mängija kaotab elu ainult viimase palli maha kukkumisel. Teiste pallide maha kukkumisel pallid ainult haihtuvad ning muid tagajärgi ei ole. Elude otsa saades kuvatakse mängijale menüü, kus on näidatud tema tulemus, võimalusega seda sisestada edetabelisse, vaadelda teiste mängijate tulemusi ning uuesti mängida. Edetabelit jälgides on võimalik näha järjestuse muutumist, kui keegi saavutab punktisumma seitsme parima hulka.

4.2 Mängu loomiseks valitud vahendid

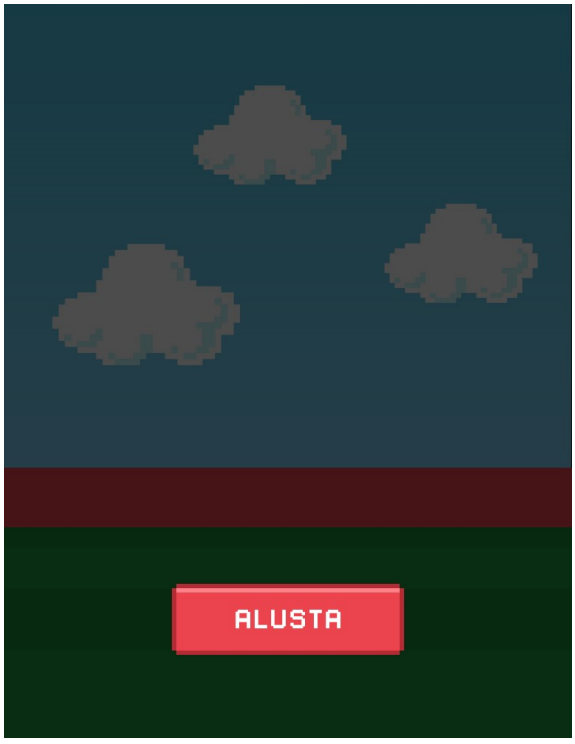
Reklaammängu loomiseks sai valitud *Phaser* 3 raamistik. Võrreldavatest variantidest osutus *Phaser* valituks seepärast, et sellel on kõige realistlikumat objektide liikumist võimaldav füüsikamootor *MatterJS*, stseenide baasil mängu ehitamine, sisseehitatud mängu loomist oluliselt kiirendav visuaalne skaleerimine ning Ekspress Meedia eelnev kogemus *Phaser* 2 raamistikuga.

Reaalaja andmebaaside seast osutus valituks Google *Firebase*, sest reklaammäng on üldjuhul lühiajaline projekt ning *Firebase*-i üles seadmine toimub kiiremini kui teistel võrreldud andmebaasidel. Liikluse ja mängijate koguse ennustamine on

reklaammängude puhul keerukas ning kulud tuleb hoida madalad. Ressursi puudujäägil tuleks seda juurde muretseda võimalikult väheste vahenditega, et suurendada teenitavat kasumit. Lisaks pakub Google võimalust maksta ainult kasutatud ressursi eest ning vajadusel kõvaketta ruumi või lubatud andmemahu suurendamine käib automaatselt.

4.3 Mänguraamistiku laadimine ja üles sättimine

Mängu alustuseks lisati pilt, mis oleks atraktiivne ning tõmbaks värvide ja kompositsiooni poolest potentsiaalse mängija tähelepanu. Joonisel (Joonis 1) on kujutatud avakuva, millele on jäetud reklaamkliendi logo jaoks ruumi „ALUSTA" nupu kohale. Avakuva tegelik eesmärk on olla *listener,* mille peale vajutades laaditakse mänguraamistiku teek. *Phaser* 3 raamistiku teek kaalub ligikaudu 200 kilobaiti ning halb tava on raisata iga möödakerija andmemahtu, kes ei mängi mängu. Selle jaoks loodi funktsioon, mis lisab raamistiku teeki HTML-i päisesse alles avapildile vajutades.



Joonis 1. Mängu avakuva

Raamistiku töötamiseks peab seadistama mängu konfiguratsiooni. Määrata tuli renderdamise liik, soovitatav on jätta see automaatse peale, et WebGL mittetöötamise korral oleks *Phaser*-il võimalus taanduda c*anvas* renderdamisele. *Canvas* HTML5 element lisatakse dokumendile siis kui skript välja kutsutakse, ent soovi korral on

võimalik defineerida ka konteiner mängu sisestamiseks [52]. Antud mängu puhul loodi *canvas* samasse konteinerisse kuhu avakuva asetati.

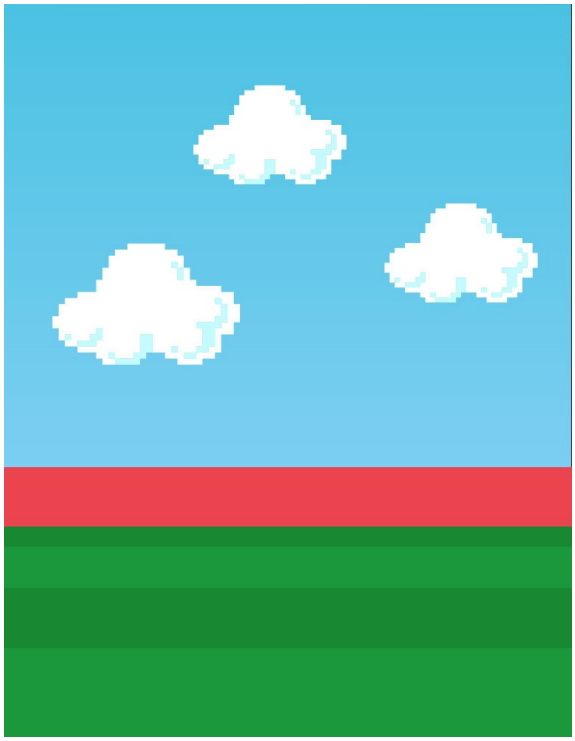
Phaser 3 stseen sisaldab kolme funktsiooni: *preload*, *create* ja *update*. *Preload* kasutatakse enne mängu loomist elementide laadimiseks. C*reate* kutsutakse välja ühe korra mängu loomisel ning *update* käivitatakse igas kaadris [53]. Mängu vajadus mitmete stseenide järele nõuab klasside defineerimist, mis sisaldavad eelnevalt mainitud funktsioone.

Mängu skaala defineerimine on vajalik teadmaks, kui suure ala sisse mäng luuakse ning kui suur on mängu enda ala [52]. Muutuva suurusega mängu loomiseks defineeriti skaleerumise režiim ning joondati see keskele. Laius ja kõrgus määrati vastavalt 768 ja 1024 pikslit.

Konfiguratsioonis on võimalik määrata füüsikalisi parameetreid ning selle sätteid, võimalikult realistliku mängu loomiseks kasutati *MatterJS* füüsikamootorit. Säilitamaks *arcade* laadset tunnetust vasakule-paremale liikudes ning mitte vajades gravitatsiooni tõmmet antud teljel kummaski suunas, sätiti x-telje väärtuseks 0. Palli ja mängija maa poole tõmbamiseks ning vältimaks palli liiga kiiret maha kukkumist, määrati y-telje gravitatsiooni väärtuseks 3.

4.4 Mängutaseme loomine

Mängijale maailma kuvamiseks alustati esimese stseeni loomisega. Kahe stseeni loomiseks on vajalik mõlemad defineerida kui eraldi klassid. Taustapildi laadimiseks raamistikku kasutati *preload* funktsiooni, kus esimese parameetrina täpsustati faili nimi sõnena, mida kasutatakse ülejäänud raamistikus ning teise parameetrina asukoht failisüsteemis. Asetamaks pilt mängu, on vajalik *create* funktsioonis määrata lisatava staatilise objekti asukoht. Esimesed kaks parameetrit defineerivad, kuhu liigutatakse objekti keskpunkt. *Phaser* raamistik sätib vaikimisi kõik elemendid paika keskpunkti abil ning seda on võimalik muuta *setOrigin* meetodi abil. Siiski peaks taust alati olema keskel ja mängu mõõtmeid muutes peaks toimuma nihkumine [53]. Joonisel (Joonis 2) on kujutatud *Phaser* raamistikuga loodud mängu taustapildiga.

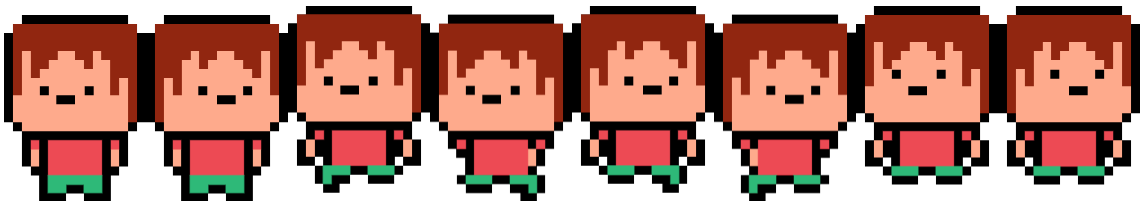


Joonis 2. *Phaser* raamistikuga loodud kuvand

4.5 Mängija loomine

Karakteri eesmärk on olla objekt, mida kontrollitakse mängija poolt ning mille abil hoitakse palli õhus. Karakteril on määratletud piirid, mille pihta pall põrkab. Liikumine on võimaldatud nii horisontaalselt kui ka vertikaalselt.

Mängija *sprite* sisaldab kõiki kaadreid, mida plaanitakse kasutada, antud mängu puhul seismist, liikumissuuna poolse jala tõstmist, vastasjala tõstmist, hüppamist ning kõikidest tegevustest leidub vasakule kui ka paremale poole liikumise variant. Joonisel (Joonis 3) näidatakse mängus kasutatavat karakterit ning kõiki kaadreid, mida kasutatakse.



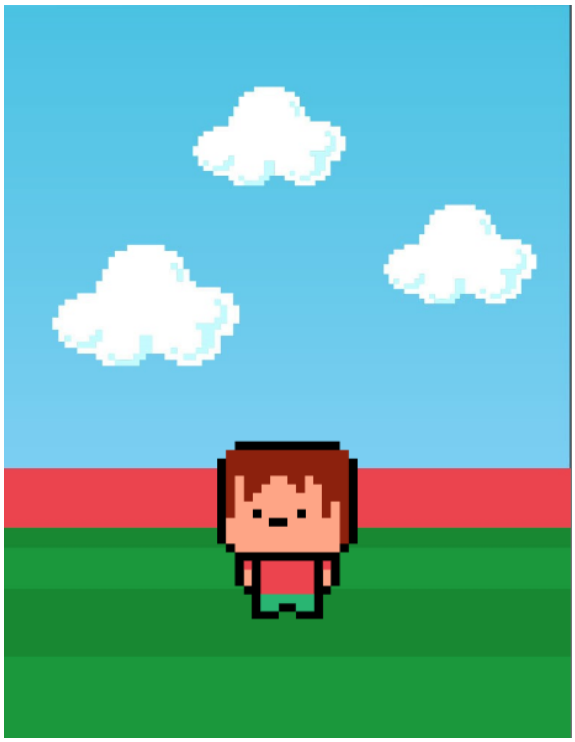
Joonis 3. Karakteri *sprite*

Karakteri kuvamiseks laaditi *preload* funktsioonis kasutaja *sprite* kui *spritesheet'i*, et võimaldada objekti liikumist ning hiljem oleks võimalik kaadreid vahetada mängija liikudes. Mängumaailma asetamiseks lisati tegelane läbi *MatterJS* füüsikamootori, et määrata füüsikalisi omadusi, näiteks mass, kiirus, põrke- ja hõõrdetegur. Vältimaks mängija põrkamist, sätiti põrkamine *setBounce* meetodi abil nulliks ning hõõrdetegur eemaldati *setFriction* meetodiga, jäljendamaks *arcade* mängu füüsikat [54].

4.5.1 Mängija jaoks platvormi loomine

Hetkel karakterit mängu asetades jääb mängija igavesti y-teljel kukkuma. Põhjus peitub peamiselt mängu raamide puudulikus definitsioonis, ent *MatterJS* füüsikamootoril on võimalik piire määratleda *setBounds* meetodiga, mis defineerib mängumaailma piirideks eelnevalt konfiguratsioonis paika pandud suurused [55].

Mugavdamaks mängija olukorda mängu jälgimisel ning tõstmaks selle visuaalset kvaliteeti, liigutati karakterit ülespoole ja loodi murule peidetud platvorm. Platvormiks määrati pilt, mis lisati füüsikamootorisse ning takistamaks platvormi liikumist, määrati see teiste objektide suhtes staatiliseks [56]. Joonisel (Joonis 4) näidatakse mängu asetatud karakterit, mis seisab platvormi peal.



Joonis 4. Karakter loodud mängumaailmas

4.5.2 Mängija animeerimine

Karakteri liigutamiseks on *Phaser* raamistikus loodud eraldi animatsioonid. Mängija liigutamiseks tuleb luua 4 erinevat animatsiooni ja valida sobivad mängitavad kaadrid: liikumine ning hüppamine vasakule ja paremale.

Igal animatsioonil peab olema unikaalne võti, mille abil hiljem animatsioonile refereerida. *Frame rate* abil saab määrata mitu kaadrit sekundis näidatakse, kaadrite valimiseks on sealjuures mitu varianti: defineerida algus- ja lõppkaader või kõik näidatavad kaadrid massiivis [57]. Karakteri peatamiseks valiti massiiviga lahendus, sest see võimaldas animatsiooni lõppedes karakteri visuaalselt seisma jätta. Algus- ja lõppkaadrit defineerides jäi viimases kaadris mängija jalg õhku. *Phaser*-is on animatsioonisüsteem igalt poolt kättesaadav ning seeläbi on võimalik kasutada ühte animatsiooni mitmel objektil [58].

Animatsioonid käivitatakse, kui kasutaja vajutab mängides arvutil nooli või mobiilis ekraanil liikumise suuna poole. Esmalt tuleb defineerida mobiilis *touch* ning klaviatuuril nooled. Mobiilis saab mitut kursorit määrata, kuid antud mängu puhul on lubatud ainult üks.

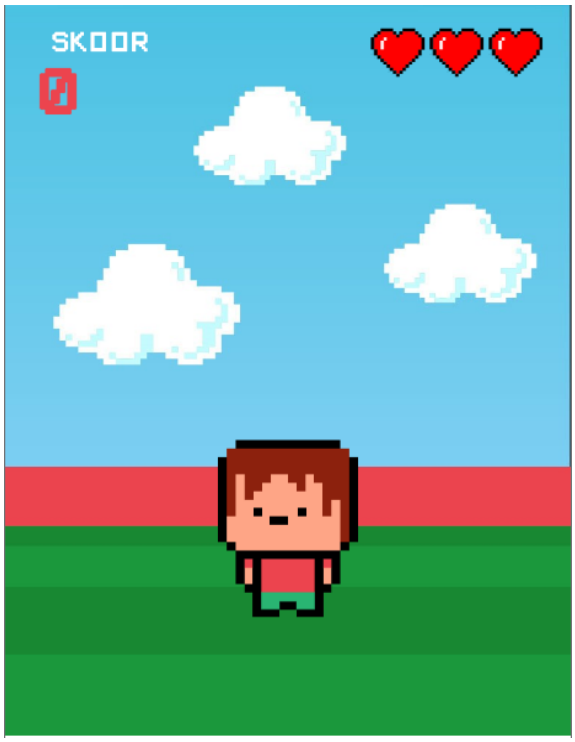
Liikumise väljakutsumist kuulatakse *update* funktsioonis, kus on erinevad klauslid. Nende abil käivitatakse meetodid, mis panevad tööle vastavad animatsioonid ning liigutavad mängutegelast soovitud suunas. Karakterisse on jäetud tühi tsoon, kuhu vajutades mängutegelane ei liigu. Tühja tsooni eesmärgiks on vältida mobiilis tekkivat vasakule ja paremale liikumise animatsiooni vahetusel ilmnevat viga, kus mängija liigutab kursorit paari piksli võrra ning animatsioone jäädakse omavahel kiiresti vahetama.

Vasakule ning paremale ekraanile vajutades kontrollitakse, kummal pool kursor on karakterist ning kas vajutatakse mängija kõrvale, mitte üles. Üles hüppamiseks tuleb vajutada taeva peale, aga vältimaks ebaloogilist hüppamist kontrollitakse õhus viibimise tõesust. Suunaliselt hüppamise jaoks loodi eraldi klauslid, mis kontrollivad õhus olekut ning tegelase suunda, mängimaks vastavat animatsiooni. Kõikidesse klauslitesse kirjutati sisse ka võimalus nooltega liikuda.

4.6 Skoori ja elude kuvamine

Mängu eesmärk on koguda võimalikult suur tulemus ning selle kuvamiseks kasutati *bitmap* fonti. Kirja lisamiseks kasutati *preload* funktsioonis *Phaser* raamistiku meetodit, kus tuleb defineerida sõnana fondi nimi, *bitmap* fonti pilt ning pildile vastav XML. Teksti lisamiseks kasutati samuti selleks loodud spetsiaalset raamistiku meetodit [59]. Tulemus sätiti vasakule, sest skoori suurenedes hakkavad numbrid pikenema paremale poole. Ankurpunktiga tulemuse keskele joondamisel tekiks probleem, kus number hakkaks mängu raami serva taha kaduma.

Südamete lisamiseks eellaaditi nii täis kui tühi süda ning defineeriti nende asukohad massiivis. Mängu alguses luuakse 3 südant ning iga kord kui viimane pall maha kukub, joonistatakse paremalt poolt üks süda üle tühjaks. Seda soodustab *Phaser* raamistiku pildi lisamise loogika, sest ainult lisamine nõuab vähem ressurssi kui südame kustutamine ja lisamine [53].



Joonis 5. Skoor ja elude arvu kujutavad südamed

4.7 Stardiloendus

Mängu alustamisel antakse mängijale stardiloendus mängu alguseni. See annab aega ette valmistuda, et mängu lisatav pall kohe alla kukkuma ei hakkaks ning et jõutaks ka mängu olemusega tutvuda. *Phaser* mänguraamistiku allalaadimise aeg oleneb internetiühenduse kiirusest ning loenduri eesmärk on parandada kasutajakogemust.

Stardiloendur käivitatakse viimasena *create* funktsioonis, kui kõik muu on valmis ning lisatud. Mängu füüsika pannakse loenduri käivitudes seisma, sest stseeni peatades ei ole võimalik enam *canvasele* joonistada. Teksti vahetamiseks loodi *tween* [60]. Iga silmus vähendatakse loenduri arvu ühe võrra, kuniks väärtus nulli jõuab ning näidatakse „*GO!*". *Tween*'i lõppedes kasutatakse meetodit *onComplete*, mis taaskäivitab füüsika.

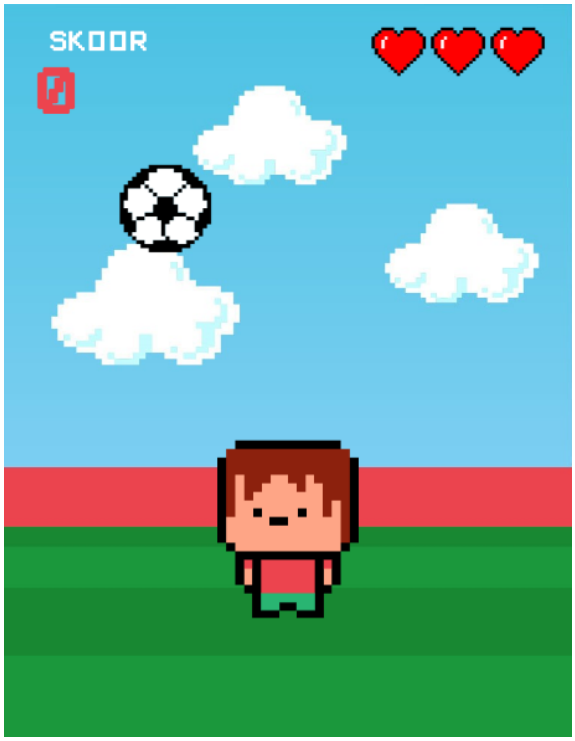
4.8 Palli loomine

Karakterile lisatakse mängu pall, millega saab skoori suurendada. Palli lisamiseks loodi taaskasutatavad meetodid, mida hiljem rakendatakse kõikide pallide jaoks uuesti. Rohkemate pallide tekitamiseks kontrollitakse, kas tulemus jaguneb 25-ga või kas mängus on üldse mõni pall alles. Lisaks luuakse kokkupõrke tuvastamiseks funktsioonid, et palli maha kukkudes kaotaks mängija elusid või suurendaks punktisummat, seda juhul kui suudeti edukalt palli üles põrgatada.

4.8.1 Palli lisamise meetod

Meetodi alguses suurendatakse mängus olevate pallide koguse muutujat, et vältida liiga paljude pallide teket. Funktsioon tagastab *image* elemendi, mis ilmub mängu üleval. Takistamaks palli ilmumist ekraani servas, kasutatakse *Phaser* raamistikus sisseehitatud meetodit, mis tagastab täisarvu määratud vahemikus. Horisontaalteljel määrati ilmumisvahemik keskpunktist veerand mängu laiust mõlemas suunas.

Võimaldamaks ideaalset põrget ning hoidmaks palli kauem õhus, määrati palli keha elastsuseks 1. Elastsuse koefitsent võib olla 0 ja 1 vahel: 0 tähendab, et põrge võib olla täielikult mitte-elastne ning 1 tähendab, et keha võib põrgata kogu kineetilist energiat säilitades [61]. Pidurdamaks esialgset kukkumist, antakse pallile kukkumise vastassuunas kiirusvektor. Joonisel (Joonis 6) on kujutatud mängu lisatud palli kukkumist.



Joonis 6. Mängus kukkuv pall

4.8.2 Kokkupõrgete kontrollimine

Mängu mängides hakkab nii pall kui ka mängija erinevate objektidega kokku põrkama. *MatterJS* füüsikal on võimalik kontrollida kontakte objektide vahel, kontakt on alati defineeritud kahe elemendi vahel [62]. Identifitseerimaks kokkupõrkavaid objekte, tuleb kasutada nende tekstuuri võtmeid. Kontrollima peab ka karakteri ja maa vahelist kontakti, palli ja mängija ning maa ja palli kokkupuudet.

Maa ja karakteri vahelise puute kontrollimine takistab karakteril õhus hüppamist. Palli ning mängija põrkel suurendatakse skoori ning selle jaoks loodi eraldi funktsioon, kus mängijale lisatakse palli põrke võimendamiseks vertikaalteljel kiirendust. Raskendamaks skoori suurendamist ning palli ühe koha peal põrkamist, suunatakse pall kas vasakule või paremale. Iga puutega käivitatakse skoori suurendamiseks funktsioon, kus kontrollitakse skoori uuendamist aegumise meetodi abil. Lahendus laseb poole sekundi jooksul ühe punkti saada ning takistab mängija tahet proovimast palli enda ja seina vahel põrgatada.

Maa ja palli vahelise kokkupõrke eesmärk on hävitada pall maha kukkumisel. Elude olemasolul käivitatakse pallide hävitamise meetod, kus luuakse uus pall juhul kui neid antud hetkel rohkem mängus pole. Elude lõppemisel ei looda uusi palle.

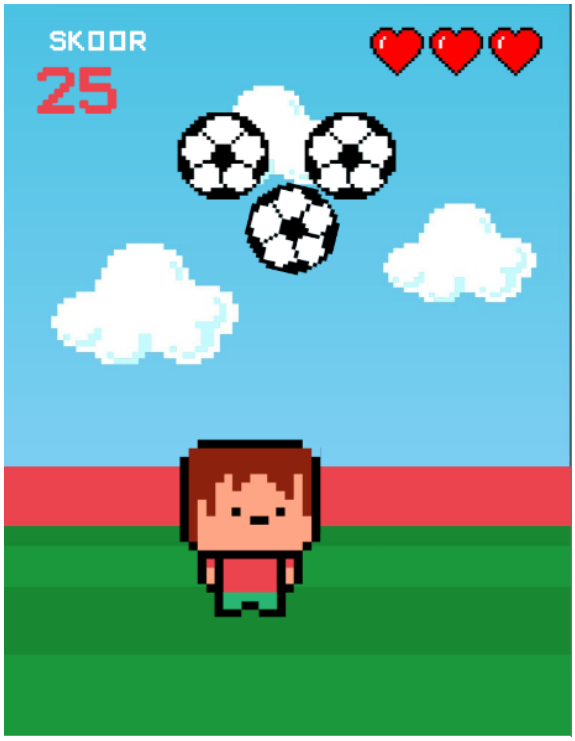
4.8.3 Pallide hävitamine

Pallide hävitamiseks loodud funktsioon kontrollib kõigepealt, kas mängus maha kukkunud pall on loodud lisana või kannab see põhipalli eesmärki. Lisapalli maha kukkumise kontroll teostatakse enne, sest pallid on erinevalt defineeritud ning samasugust loogikat ei saa mõlemal juhul kasutada. Järgmiseks eemaldatakse pallide hulka loetlevast muutujast kukkunud pall.

Kui kõik pallid on mängust eemaldatud, kaotab mängija ühe elu. Vähendamaks palli loomisel tekkivat võimaliku viga, kus ühtegi palli ei looda, loodi funktsioon, mis palli esialgsesse olekusse viib. Kontrollitakse, kas palli juba luuakse, nende kogust mängus ning elude jääki. Tingimuste tõeks osutumisel taastatakse pall.

4.8.4 Uute pallide lisamine

Update funktsioonis jälgitakse skoori järgnevalt: kui skoor jagub 25-ga tekib täisarv ning uusi palle juba ei looda, siis käivitatakse suvaliste pallide tekitamiseks vastav meetod. Esmalt valitakse juhuslikult pallide kogus, mis võib olla 1 või 2. Valitud arvu korda käivitatakse pallide loomiseks varasemalt loodud funktsioon ning peale pallide loomist lubatakse uusi palle mängu alles peale 5 sekundi möödumist. Ajalise piirangu põhjus on anda mängijale aega taastumiseks, kui kõik pallid maha kukuvad. *Update* funktsioon käivitatakse 30 korda sekundis ning antud lahendus aitab vältida funktsiooni mitmekordset käivitamist lühikese aja vältel. Joonisel (Joonis 7) on näidatud esialgset palli ning kahte lisaks loodud palli, kui skoor jagub 25-ga.



Joonis 7. Esialgne ja kaks lisaks loodud palli mängus

4.8.5 Palli loogika silumine

Testides tekkis mitu juhust, kus pall hävinedes uuesti ei „sündinud" või siis haihtus, kui pall jäi karakteri ja seina vahele. Logides juhtunut, selgus, et pall läheb maailma piiridest välja. Selle lahendamiseks loodi klausel, mis kontrollib kas pall on mängus nähtav. Esimene osa klauslist sisaldab võrdlust, kas pall on madalamal kui karakteri keskpunkt maa peal olekus ning teise osa põhimõte on vaadata, kas palli koordinaadid on mängu piiride sees. Kolmandas osas kontrollitakse palli y-telje koordinaati, et see oleks samal platvormil kus mängija seisab ning mängu ülemise serva vahel. Kui üks neist võrdlustest ei kehti, siis pall „tapetakse".

4.9 Menüü loomine

Elude otsa saamisel kuvatakse kasutajale menüü, kust on võimalik näha tema tulemust. Menüü loodi kui uus stseen, et struktureerida programmikoodi. Menüü esimeseks kihiks asetati graafikapealistus, mille läbipaistvuse koefitsendiks määrati 0.5 ning juurde lisati ka nupud mängu uuesti alustamiseks, edetabeli vaatamiseks ja registreerimise vorm tulemuse salvestamiseks edetabelisse. Kasutades *bitmap* fonti, mis on olnud eelnevalt kasutuses erinevate tekstide loomiseks, paigutati menüüsse skoori tekst ja skoor

numbrina. Nupud lisati piltidena menüü alumisesse osasse. Nuppude töötamiseks on vaja nendel kasutada meetodit *setInteractive*, mis lubab nupu peal vajutada. Nuppudele lisati funktsionaalsused: uuesti mängides peidetakse nupp *setAlpha* meetodi abil ära, sest muidu saab kasutaja selle peale vajutada esimeses stseenis.

Sündmuste saatmiseks on *Phaser* raamistikus olemas e*vent emitter*, mis aitab suhelda stseenide vahel ning saata kaasa skoope [63]. Selle jaoks loodi eraldi klass, mis defineeriks mängu kontekstis *event emitter*i, et seda oleks võimalik kasutada teistes klassides. Mängu uuesti alustamise nupuga saadetakse kaasa sündmus, mis esimeses stseenis sätib vajalikud muutujad, nagu näiteks elude hulk ja skoor, paika. Nupu vajutusega taastatakse mängu esimene stseen esialgsesse olekusse ning tuuakse esimeseks kihiks [64]. Joonisel (Joonis 8) on kujutatud menüü tulemuse ja uuesti mängimise võimalusega.

Joonis 8. Skoor ja uuesti mängimise võimalus menüüs

4.9.1 Menüü vorm

Registreerimisvormi lisamiseks mängu on *Phaser* raamistikul loodud funktsioon, mille abil saab HTML faile *canvase* peale kuvada. Kuna *canvas* elemendis joonistatakse graafikat, siis ei ole võimalik raamistiku siseselt vorme luua [65]. Registreerimiseks

loodi eelnevalt valmis vorm, mis laaditi raamistiku *preload* funktsioonis raamistikus defineeritud meetodiga.

Esmalt kontrollitakse, kas mängija on juba tulemuse registreerinud, et vältida mitu korda sama skoori postitamist andmebaasi. Kui eelnevalt pole tulemust postitatud, luuakse vahemälusse varasemalt meetodi *createFromCache* abil laetud vorm DOM-i [66]. Positsioneerides määrati keskpunkt vormi keskele, kuna vaikimisi sätitakse HTML vasaku ülemise nurga järgi. Joonisel (Joonis 9) on näidatud vorm tulemuse registreerimiseks

Joonis 9. Vorm tulemuse registreerimiseks

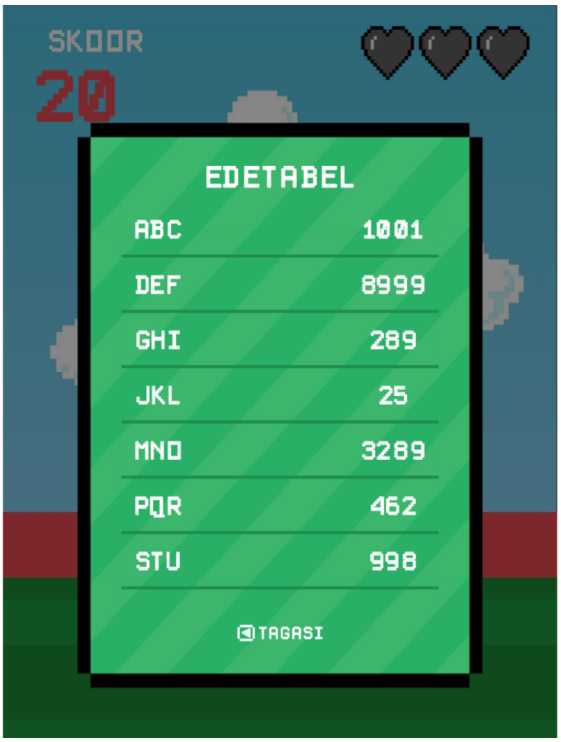
Vormi kasutamise loogika kirjutati stseeni sisse hoidmaks asju ühes skoobis. Vormi enda valideerimine toimub läbi *Phaser* raamistiku. Kontrollitakse nii e-posti aadressi valideerumist, seda kas nime lahter on tühi, lubatud sümbolitele ning kolmele tähemärgile vastavust ja tingimustega nõustumist. Mittevastavale väljale tehakse ümber punane kast, mis viitab tehtud veale. Kui vorm on õigesti täidetud, eemaldatakse vormilt kliki kuulamine ning tähistatakse registreerimisõigus. Viimase asjana avatakse edetabel. Joonisel (Joonis 10) on kuvatud mittevalideeruv vormi.

Joonis 10. Vorm tulemuse registreerimiseks

4.10 Edetabeli loomine

Edetabeli loomiseks kasutati esialgu stseeni loogikas testi jaoks defineeritud massiivi. Antud viisi eesmärgiks oli luua funktsioonid, mis joonistaksid edetabelisse tulemused ja neid eraldavad jooned vastavalt massiivi pikkusele. Testandmete kasutamise põhjuseks oli kiirendada töös eelnevalt kirjeldatud osade valmimist, et saaks hiljem keskenduda implementeeritavale andmebaasile. Edetabeli pealkiri ja menüüsse naasmist võimaldav nupp koos funktsionaalsusega lisati *create* meetodis, kuid nende läbipaistvus seatakse nulliks vähendamaks andmete laadimisega tekkivat viivitust ning luua võimalikult palju elemente stseeni loomisel.

Vajutades edetabeli peal või registreerides tulemuse esimest korda, seatakse ebavajalike tekstide ja registreerimisvormi läbipaistvus nulli ning joonistatakse funktsioonide abil edetabel. Kõik paigutatavad elemendid on lisatud gruppi, lihtsustamaks nende eemaldamist ja uuesti loomist andmete muutumisel. Gruppides olevatele elementidele lisatakse animatsioon, et kasutajale kuvada edetabelis parima seitsme tulemuse uuenemist. Joonisel (Joonis 11) näidatakse testandmetega loodud edetabelit, mis ei ole sorteeritud alates suurimast skoorist.



Joonis 11. Edetabel testandmetega

4.11 Mängu ühendamine reaalaja andmebaasiga

Andmebaasi kasutamiseks loodi konsoolis uus projekt, millele määrati esmalt nimi „Phaser 3 reklaammang". Pakuti välja ka Google *Analytics*-i kasutamine, kuid sellest loobuti. Nimelt oleks selle kasutamine nõudnud lisateeke, mis suurendaksid mängu suurust. Minnes projekti ülevaatesse, valiti *Realtime Database* ning loodi uus andmebaas, kus seejärel pidi määrama projekti reeglite režiimi. *Firebase* laseb valida kahe variandi vahel: lukus režiim, mis keelab igasuguse andmete postitamise ja lugemise, ning testimise režiim, kus lubatakse kõike lugeda ning kirjutada.

4.11.1 Andmebaasi struktuur

Reaalaja andmebaasis hoitakse andmeid kui JSON objektidena, puuduvad tabelid nagu on SQL andmebaasides. Andmete lisamisel tekivad sõlmed, mida ühendatakse võtmetega. Võimalik on kasutada kas andmebaasi poolt automaatselt loodud võtmeid või lisada neid ise, ent sel juhul peavad need vastama UTF-8 kodeeringule ning olema kaalult maksimaalselt 768 baiti [67].

Dokumentatsioonis soovitatakse andmete pesastamist vältida ning hoida struktuur võimalikult siledana. Lisaks kui pärida vanemelemendi kohta, saadakse kaasa ka kõik lapssõlmed [67]. Selle jaoks loodi suurkogu *players*, kuhu postitatakse kaks eraldi alamkogumit: personaalne info, mis sisaldab emaili ja tingimustega nõustumist, ning skoor, kus hoitakse tulemust. Suurkogu loomise eesmärk on vältida andmete kirjutamist kõige kõrgemale tasemele.

Mõlemad kogud sisaldavad tulemusega registreeritud nime, et oleks kergem luua seoseid tulemuste ja emaili vahel. Personaalse info ja skoori eraldi hoidmise põhjuseks on sensitiivse info kohta tehtavate päringute takistamine. Postitades andmebaasi sõlme nii emaili kui ka skoori sisaldava objekti, ei ole võimalik reeglitega defineerida osa sõlme mittelugemisest, sest päring tagastab objektid võtmetega ja kogu sisuga [67].

4.11.2 Andmebaasi reeglite loomine

Reeglitega määratleti lugemis- ja kirjutamisõigused kogu andmebaasi projektile (Lisa 1 – Google *Firebase* reaalaja andmebaasi reeglid). Esialgu keelati kõik lugemise ja kirjutamise päringud terves andmebaasis. Reeglid kirjutatakse JSON formaadis. Esimeseks lisati objekt *players*, mille sisse omakorda kaks objekti nimedega *personaldata* ja *scores* [68].

Personaldata lugemisõigus keelustati enne järgnevaid objekte ära, kuid kirjutamisõigus jäeti alles, et lapssõlmedes oleks võimalik andmeid valideerida. Objekt *$uid* defineerib aga automaatselt loodavat võtit. Postitavate objektide jaoks loodi erinevad validatsioonid, mis kattuvad vormi omadega, takistamaks ebasobivate andmete postitamist. Objekti *$other* eesmärk on takistada muude väärtuste postitamist, mida ei ole eelnevalt andmebaasis reeglitega defineeritud [68],[69].

Scores objekti lugemis- ja kirjutamisõigused on sätitud tõeseks. Objekti *indexOn* eesmärk on kiirendada päringu kiirust ning indeksi väärtust teades saab selle reeglites kiiremini kindlaks määrata [70]. Ülejäänud struktuur sarnanes *personaldata* reeglitega: lisati väärtuste valideerimine ning keelati muude andmete postitamine ära.

4.11.3 Andmebaasi kasutus

Võimaldamaks kasutada mängus Google *Firebase* andmebaasi päringuid, lisati HTML-i päisesse teekid. Esimeseks lisati *Firebase app* teek, mis peab alati esimesena olema, kuna see on tuum SDK. Järgmisteks on vaja valida, mis tooteid soovitakse kasutada. Antud mängu jaoks vajati veel *Firebase Realtime Database SDK*'d [71].

Avamaks andmebaasi ühendus, käivitatakse funktsioon kontrollimaks, kas ühendus *Firebase* andmebaasiga on juba initsialiseeritud [72]. Menüü stseeni esialgsel ehitamisel luuakse ühendus, aga kui on toimunud stseenide vahetus ning ühendus eelnevalt loodud, siis ei hakata mitut ühendust tegema.

Hoidmaks andmebaasi ühenduste arv võimalikult madal, implementeeriti ühenduse loomise juurde lisaloogika. Menüü avanedes kasutab *Firebase* meetodit *goOnline*, et uuesti avada ühendus. Menüüst lahkudes käivitatakse *goOffline* funktsiooni, vältimaks andmebaasi päringute toimumist, kui mäng käib, et säästa andmemahtu ning seadme ressurssi [73].

4.11.4 Andmete kasutamine

Andmete postitamiseks reaalaja andmebaasi kasutatati *push* meetodit. Saatmaks kirjeid kahte erinevasse sõlme, vajasid sõlmed eraldi lahti defineerimist. *Set* meetodit kasutades kirjutatakse kirjed antud sõlmes üle, kuid p*ush* funktsiooniga antakse kaasa ka unikaalne võti, mis välistab selle juhtumise [74].

Andmete kättesaamiseks ning andmebaasis toimuvate muutuste kuulamiseks, kasutati *once* funktsiooni asemel *on* funktsiooni, sest see jääb väärtuste muutusi kuulama. Nii näeb edetabelit jälgides muutusi ka mängija [74]. Sorteerimaks infot olid abiks *Firebase* andmebaasi sisse ehitatud *orderByChild* ja *limitToLast* funktsioonid. *OrderByChild* ülesanne oli järjestada kõik kirjed skoori põhjal ning *limitToLast* võttis viimased 7 kirjet, kuna edetabelis kuvatakse ainult 7 parimat tulemust [75]. Iga kord kui jälgitav väärtus muutub, kustutatakse vana edetabel ja luuakse animatsiooniga uus.

Google *Firebase* pakub võimalust kirjutada funktsioone, mida käitatakse tagaraamistikus. Kood hoitakse pilves ning muutused rakenduvad kõigile. *Cloud Functions* aitab kiiresti viia sisse muutusi päringutes ning kasutajad ei pea enda poolt

midagi uuendama. Lisaks aitab see korrastada andmebaasi struktuuri [76]. *Cloud Functions* pakub ka võimalust kuulata andmete lisandumist, uuendamist või kustutamist [77]. Antud projektile see lahendus ei sobinud, sest sellel leidus kaks negatiivset külge. Esimeseks negatiivseks omaduseks on *Firebase CLI*, mille ülesse seadmine võtab rohkem aega kui kasutaja pool päringute defineerimine [78]. Teiseks on *Cloud Functions* tasuline, mis vähendab mängult saadavat tulu [50]. Võimalus on kasutada ka tasuta paketti, kuid see hõlmab *Node.js* 8, mille kasutus lõppeb 15. veebruar 2021 ning kõik eelnevad funktsioonid peatatakse 15. märts 2021 [79].

4.12 Mängu lisamine reklaamisüsteemi

Ekspress Meedia AS kasutab Adform-i reklaamisüsteemi. Lubamaks mängu eetrisse, peab see vastama mitmetele nõuetele. Reklaammäng peab olema ühes ZIP failis ning mitte tegema kolmanda osapoolte lehtedele päringuid, kui pole just muud kokkulepet. Antud mängu puhul on olemas luba *Phaser* raamistik alla laadida mujalt.

Kaal peab olema limiidi sees. Esialgu näidatakse ainult avakuva ning koos *Firebase* teekidega kaalub mäng 89.1 kilobaiti, mis jääb piiridesse. Mäng vastab ka muutuva suuruse nõudele, et mobiilis oleks mängu laius vastav seadme laiusele [80].

Kasutamaks Adform-i sündmuste registreerimist, et saada statistikat kasutajate käitumise kohta, lisati HTML-i päisesse Adform-i teek. Sündmuste saatmiseks kasutati staatiliste väärtuse jaoks mõeldud meetodit, kus *event_id* väärtus jäi 1 ja 20 vahele [81]. Tabelis (Tabel 4) on kirjeldatud kasutatavaid sündmuseid nimetustega.

Tabel 4. Reklaamisüsteemi saadetavad sündmused ja nende väärtused

Sündmus	ID	Sündmuse nimetus
Mängu alustamine	1	Game start
Mängu lõpetamine	2	Game end
Tulemuse registreerimine	3	Register
Karakteri hüppamine	4	Player jump
Edetabeli vaatamine nupust	5	Highscore from button
Uuesti mängimine	6	Replay

Sündmus	ID	Sündmuse nimetus
Palli põrge vastu mängijat	7	Bounce against player

5 Kokkuvõte

Käesolevas lõputöös uuriti veebimängude teket, *Flash Player*-it, selle edu põhjust ning langust. Uuriti kolme erinevat mänguraamistiku, millega saab JavaScript programmeerimiskeelega mänge luua. Võrreldi nende ajalugu, viimase suurema väljalaske aega, populaarust, reitingut kui ka vanust. Lisaks uuriti mänguraamistike funktsionaalsuse ja võimaluste erinevust viies kategoorias: visuaalsed võimalused, raamistikuga seotud mängutaseme loomise tööriistad, füüsikamootor, loogika ja heli. Kiirendamaks ja standardiseerimaks reklaammängu arendust ning aitamaks luua taaskasutavaid mängu osasid, osutus väljavalituks *Phaser*, mille füüsikamootor lubab konkurentidega võrreldes kõige realistlikumat liikumist. Igal ajahetkel uueneva edetabeli loomiseks valiti reaalaja andmebaaside võrdlusest välja Google *Firebase*, mille kasuks eeliseks konkurentide ees oli kiire seadistamise aeg ja hinnakiri.

Töö tulemusena valmis reklaammäng, mis loodi *Phaser* 3 mänguraamistiku ja Google *Firebase Realtime Database* abil, on mängitav mobiilis ning lihtsa põhimõttega, alglaadimise maht jäi alla lubatud piiri ning mängus on edetabel, milles toimuvad muutused reaalajas. Taaskasutatavate elementidena loodi näiteks andmebaasiga suhtlemine ja selle jaoks määratud reeglid, mida saab kasutada ka muudes reklaami erilahendustes või veel suuremates projektides, kus on vaja reaalajas jooksvat infot. Taaskasutusväärtus on ka liikumise, stardiloenduse, stseenivahetuse ning kokkupõrgete kuulamise funktsioonidel, mida saab lihtsalt implementeerida järgnevatesse reklaammängudesse, hoides aega kokku.

Phaser 3 raamistiku tundma õppimine oli eeldatust ajakulukam ning Google *Firebase* andmete struktureerimise võimalused olid uudsed. Ühtlasi illustreeris väiksemahulise mängu loomine hästi arendusprotsessiga seotud tööressursside kulu, andes aimu kui suur töö on suuremate mängude loomise taga.

Kasutatud kirjandus

[1] IT terministandardi sõnastik, http://www.eki.ee/dict/its/index.cgi?Q=bitmap&F=M&C06=et&C10=1 (25.11.2020)

[2] <canvas>: The Graphics Canvas element, https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas (25.11.2020)

[3] Introduction to the DOM, https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction (25.11.2020)

[4] JSON, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON (25.11.2020)

[5] IT terministandardi sõnastik, http://www.eki.ee/dict/its/index.cgi?Q=sprite&F=M&C06=et&C10=1 (25.11.2020)

[6] Lambert, S. An Introduction to Spritesheet Animation, https://gamedevelopment.tutsplus.com/tutorials/an-introduction-to-spritesheet-animation--gamedev-13099 (25.11.2020)

[7] Structured Query Language (SQL), https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql?redirectedfrom=MSDN&view=sql-server-ver15 (25.11.2020)

[8] Inbetweening, https://en.wikipedia.org/wiki/Inbetweening (25.11.2020)

[9] WebGL, https://developer.mozilla.org/en-US/docs/Glossary/WebGL (25.11.2020)

[10] XML introduction, https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction (25.11.2020)

[11] Featherly, K. ARPANET, https://www.britannica.com/topic/ARPANET (11.11.2020)

[12] Ray, M., Online gaming, https://www.britannica.com/technology/online-gaming (11.11.2020)

[13] Adobe, https://web.archive.org/web/20060717071243/http://www.adobe.com/macromedia/events/john_gay/page04.html (11.11.2020)

[14] Benson, T. The best Flash games, https://www.digitaltrends.com/gaming/best-flash-games/ (11.11.2020)

[15] Weber, R. From MiniClip to Mega Brand https://www.gamesindustry.biz/articles/2013-05-17-miniclip-interview (11.11.2020)

[16] Mouli, M. In loving memory of flash games, https://www.thedailystar.net/bytes/news/loving-memory-flash-games-1653250 (11.11.2020)

[17] Macromedia, https://en.wikipedia.org/wiki/Macromedia (09.11.2020)

[18] Adobe Flash Player, https://en.wikipedia.org/wiki/Adobe_Flash_Player (09.11.2020

[19] Adobe, https://www.adobe.com/ee/products/flashplayer/end-of-life.html (09.11.2020)

[20] Announcing Phaser (Flixel HTML5) and our Adobe Max session, http://www.photonstorm.com/phaser/announcing-phaser-flixel-html5-and-our-adobe-max-session (09.11.2020)

[21] Phaser 1.0 and the journey we took to get there, http://www.photonstorm.com/phaser/phaser-1-0-and-the-journey-we-took-to-get-there (09.11.2020)

[22] Welcome to Phaser 3, https://phaser.io/phaser3 (09.11.2020)

[23] Phaser – HTML5 Game Framework, https://github.com/photonstorm/phaser (09.11.2020)

[24] PixiJS — The HTML5 Creation Engine, http://pixijs.download/release/docs/index.html (09.11.2020)

[25] Tracey, E. What is Pixi.js and why is it so popular in Germany? https://blog.honeypot.io/pixi.js-is-the-german-gaming-industrys-newest-friend/ (09.11.2020)

[26] Github, https://github.com/pixijs/pixi.js/releases (09.11.2020)

[27] PixiJS, https://www.pixijs.com/tutorials (10.11.2020)

[28] Examples for PixiJS, https://pixijs.io/examples/ (10.11.2020)

[29] Impact, https://impactjs.com/ (10.11.2020)

[30] ImpactJS Engine, https://html5gameengine.com/details/2/impactjs-engine (10.11.2020)

[31] Impact, https://impactjs.com/documentation (10.11.2020)

[32] Impact, https://impactjs.com/download (10.11.2020)

[33] Real-time database, https://en.wikipedia.org/wiki/Real-time_database (11.11.2020)

[34] Melendez, S. Sometimes You're Just One Hop From Something Huge, https://www.fastcompany.com/3031109/sometimes-youre-just-one-hop-from-something-huge (11.11.2020)

[35] Firebase Realtime Database, https://firebase.google.com/products/realtime-database (11.11.2020)

[36] Lardinois, F. Google Acquires Firebase To Help Developers Build Better Real-Time Apps, https://techcrunch.com/2014/10/21/google-acquires-firebase-to-help-developers-build-better-realtime-apps/ (11.11.2020)

[37] Hickey, M. Facebook Buys Mobile App Platform Parse, https://www.forbes.com/sites/matthickey/2013/04/25/facebook-buys-mobile-app-platform-parse/ (11.11.2020)

[38] Kincaid, J. YC-Funded Parse: A Heroku For Mobile Apps, https://techcrunch.com/2011/08/04/yc-funded-parse-a-heroku-for-mobile-apps/ (11.11.2020)

[39] Lylney, M. This Startup Could Literally Change The Way The Entire App Industry Works, https://www.businessinsider.com/meet-the-most-important-startup-of-2012-2012-7 (11.11.2020)

[40] Parse Server Guide, https://docs.parseplatform.org/parse-server/guide/ (11.11.2020)

[41] Tepper, F. Facebook's Parse developer platform is shutting down today, https://techcrunch.com/2017/01/30/facebooks-parse-developer-platform-is-shutting-down-today/ (11.11.2020)

[42] Kuzzle, https://kuzzle.io/company/about-us/kuzzle-in-5-minutes/ (11.11.2020)

[43] Kuzzle, https://docs.kuzzle.io/core/2/guides/essentials/introduction/ (11.11.2020)

[44] HTML5 Game Engines, https://html5gameengine.com/tag/2d (10.11.2020)

[45] cdnjs, https://cdnjs.com/libraries (10.11.2020)

[46] Phaser 3 examples, https://phaser.io/examples (12.11.2020)

[47] pixi-tilemap, https://github.com/pixijs/pixi-tilemap (12.11.2020)

[48] PixiJS Sound Basics, https://pixijs.io/pixi-sound/examples/#section-filetypes (12.11.2020)

[49] Virtuaalserverite detailne võrdlus, https://www.zone.ee/et/virtuaalserver/vordlus/ (13.11.2020)

[50] Pricing plans, https://firebase.google.com/pricing (13.11.2020)

[51] Kuzzle, https://kuzzle.io/pricing/ (13.11.2020)

[52] Davey, R. Part 1 - Introduction, http://phaser.io/tutorials/making-your-first-phaser-3-game/part1 (14.11.2020)

[53] Davey, R. Part 2 - Loading Assets, http://phaser.io/tutorials/making-your-first-phaser-3-game/part2 (14.11.2020)

[54] Photonstorm, https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Matter.Sprite.html (15.11.2020)

[55] Photonstorm, https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Matter.World.html (15.11.2020)

[56] Photonstorm, https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Matter.Components.Static.html (15.11.2020)

[57] Spritesheet Animation feedback, https://www.html5gamedevs.com/topic/32088-spritesheet-animation-feedback/ (15.11.2020)

[58] Davey, R. Part 5 - Ready Player One, https://phaser.io/tutorials/making-your-first-phaser-3-game/part5 (15.11.2020)

[59] Load Bitmap Font, https://phaser.io/examples/v2/loader/load-bitmap-font (15.11.2020)

[60] Davey, R, Tween Task, https://rexrainbow.github.io/phaser3-rex-notes/docs/site/tween/ (15.11.2020)

[61] Photonstorm, https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Matter.Components.Bounce.html (16.11.2020)

[62] Collision Event, https://phaser.io/examples/v3/view/physics/matterjs/collision-event (16.11.2020)

[63] Event emitter, https://rexrainbow.github.io/phaser3-rex-notes/docs/site/eventemitter3/ (16.11.2020)

[64] Davey, R. Phaser 3 Dev Log #121, https://phaser.io/phaser3/devlog/121 (16.11.2020)

[65] Canvas API, https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API (16.11.2020)

[66] Photonstorm, https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.DOMElement.html (16.11.2020)

[67] Structure Your Database, https://firebase.google.com/docs/database/web/structure-data?authuser=0 (17.11.2020)

[68] Learn the core syntax of the Realtime Database Rules language, https://firebase.google.com/docs/database/security/core-syntax?authuser=0 (17.11.2020)

[69] Use conditions in Realtime Database Rules, https://firebase.google.com/docs/database/security/rules-conditions?authuser=0, (17.11.2020)

[70] Index Your Data, https://firebase.google.com/docs/database/security/indexing-data (17.11.2020)

[71] Add Firebase to your JavaScript project, https://firebase.google.com/docs/web/setup (17.11.2020)

[72] Installation & Setup in JavaScript, https://firebase.google.com/docs/database/web/start (17.11.2020)

[73] Documentation, https://firebase.google.com/docs/reference/js/firebase.database.Database (17.11.2020)

[74] Read and Write Data on the Web, https://firebase.google.com/docs/database/web/read-and-write (17.11.2020)

[75] Work with Lists of Data on the Web, https://firebase.google.com/docs/database/web/lists-of-data (17.11.2020)

[76] Cloud Functions for Firebase, https://firebase.google.com/docs/functions (17.11.2020)

[77] Database Triggers, https://firebase.google.com/docs/functions/database-events (17.11.2020)

[78] Get started: write, test, and deploy your first functions, https://firebase.google.com/docs/functions/get-started (17.11.2020)

[79] Manage functions deployment and runtime options, https://firebase.google.com/docs/functions/manage-functions#set_nodejs_version (17.11.2020)

[80] Mis on HTML5? https://www.ekspressmeedia.ee/reklaam/html5/#nouded (18.11.2020)

[81] Event Tracking, https://support.adform.com/documentation/build-html5-banners/html5-banner-specifications/event-tracking/ (18.11.2020)

Lisa 1– Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Indro Kottise

1 Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Reklaammängu loomine Phaser 3 raamistikuga Ekspress Meedia näitel" mille juhendaja on Toomas Lepikult

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2 Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3 Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

07.01.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Google Firebase reaalaja andmebaasi reeglid

```
{
   "rules": {
     ".read": false,
     ".write": false,
       "players": {
         "personaldata": {
           ".read": false,
           ".write": true,
             "$uid": {
               "name": {
                 ".validate": "newData.val().length < 4",
               },
               "email": {
                 ".validate": "newData.val().matches(/^[A-Z0-9._%+-]+@[A-Z0-
                          9.-]+\\.[A-Z]{2,4}$/i)",
               },
               "agree1": {
                 ".validate": "newData.isBoolean()",
               },
               "agree2": {
                 ".validate": "newData.isBoolean()",
               },
               "$other": { ".validate": false },
             }
         },
         "scores": {
           ".read": true,
           ".write": true,
           ".indexOn": ["score"],
             "$uid": {
               "score": {
                 ".validate": "newData.isNumber()",
               },
               "name": {
                 ".validate": "newData.val().length < 4",
               },
               "$other": { ".validate": false },
             }
         }
       }
   }
}
```

Joonis 12. Reaalaja andmebaasi reeglid