VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Mikko Kallio

Verkkopelin kehitys Web-ohjelmointikieliä ja työkaluja käyttäen

Tekniikka
2024

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Mikko Kallio
Opinnäytetyön nimi	Verkkopelin kehitys Web-ohjelmointikieliä ja työkaluja käyttäen
Vuosi	2024
Kieli	suomi
Sivumäärä	61
Ohjaaja	Tommi Rintala

Tämän opinnäytetyön tarkoitus on luoda verkkopeli Web-ohjelmointikieliä ja työkaluja käyttäen. Työn tarkoituksena on tutkia, kuinka kattavan ja realistisen pelin Web-ohjelmointiin soveltuvilla kielillä ja työkaluilla on mahdollista luoda. Tarkoituksena on samalla perehdyttää opinnäytetyön tekijää niin Web-ohjelmoinnin kuin pelikehityksen aloihin.

Työssä käytetiin ohjelmointikielinä HTML:ää, CSS:ää sekä JavaScriptiä ja kaikki työn vaiheet ja muutokset tallennettiin GitHubiin Git versionhallintatyökalun avulla. Tietokantana käytettiin MySQL:a ja palvelimen hallintaan käytettiin PHP-komentokieltä. Koodieditorina käytettiin Visual Studio Codea ja pelin testaaminen paikallisessa palvelimessa toteutettiin MAMP:n avulla. Realistilla fysiikkamalleina käytettiin Marco Monsterin luomia fysiikkakaavoja. Työ toteutettiin kolmessa eri vaiheessa: ensimmäisenä toteutettiin pelin perusmekaniikat, toisessa vaiheessa peli siirrettiin ajettavaksi palvelimeen ja viimeisessä vaiheessa luotiin ja korjattiin muutamia pienempiä toimintoja.

Työn tuloksena luotiin varsin onnistunut ajopeli. Pelin tärkeimmät ominaisuudet saatiin luotua ja toimimaan riittävän onnistuneesti. Ajoneuvot pelissä saatiin myös paikoin käyttäytymään fysikaalisesti realistisella tavalla. Projektissa ja pelissä on joitakin pieniä virheitä ja puutteita, jotka jäävät hyviksi jatkokehittämiskohteiksi tulevaisuudessa. Työ kuitenkin todistaa, että pelkillä Web-ohjelmoinnin kielillä ja työkaluilla voi luoda kattavan ja laaja-alaisen pelin.

Avainsanat web-ohjelmointi, peliohjelmointi, PHP, MySQL, JavaScript

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Mikko Kallio
Title	Web Game Development using Web Development Languages and Tools
Year	2024
Language	Finnish
Pages	61
Name of Supervisor	Tommi Rintala

The purpose of this thesis was to create an online game with Web development languages and tools. The aim was to investigate how comprehensive and realistic a game it is possible to create by languages and tools suitable for Web development. The purpose was also to develop the thesis author for fields of both Web and game development.

HTML, CSS and JavaScript were used as programming languages and all phases and changes were saved in GitHub by the Git version control tool. MySQL was used as a database and the PHP script language for controlling server. Visual Studio Code was used as a code editor and testing the game in the local server was executed by MAMP. Physics formulas created by Marco Monster were used as realistic physics models. The thesis was implemented in three different periods, first the basic mechanics of the game were implemented, in the second phase, the game was transferred to be executed in the server and in the final phase, a few small functions were created and fixed.

A fairly successful game was created as a result of the thesis. The most crucial functions of the game were created and they work well enough. Vehicles work in a physically realistic way in some parts of. There are some minor errors and deficiencies in the project and in the game, which a good target for further development in the future. Overall, the thesis proves that a comprehensive and wide-ranging game can be created just by Web development languages and tools.

Keywords Web development, game development, PHP, MySQL, JavaScript

SISÄLLYS

KUVA- JA TAULUKKOLUETTELO

1 JOHDANTO

1.1 Yleiskuvaus

Tämän opinnäytetyön tavoitteena on luoda interaktiivinen verkkopeli eli peli, jossa pelaajan tekemät päätökset/toiminnot ohjaavat peliä eteenpäin käyttäen web-ohjelmoinnissa käytettäviä ohjelmointikieliä ja työkaluja. Tarkoitus on näin soveltaa sekä pelikehityksen ja Web-ohjelmoinnin periaatteita yhteen ja perehdyttää täten opinnäytetyön tekijää molempiin ohjelmoinnin aloihin. Pelinkehityksen lisäksi olisi tarkoitus lisätä osaamista myös web-ohjelmoinnista ja niihin soveltuviin aloihin, kuten web-suunnittelu, frontend- ja backend-ohjelmointi sekä UI-suunnittelu.

Web-ohjelmoinnilla viitataan verkkosivujen kehittämiseen, rakentamiseen ja ylläpitämiseen. Web-ohjelmointiin kuuluu olennaisina osina web-suunnittelu, web-julkaisuun ja tietokantojen hallinta (GeeksForGeeks, 2023e). Niistä kukin tulee olemaan olennaisia työn kehittelyssä.

Pelikehityksellä viitataan pelin luomiseen ja kuvataan pelin suunnittelua, kehitystä ja itse pelin julkaisua. Pelikehittäjä voi olla itse ohjelmoija, mutta myös esimerkiksi äänisuunnittelija, taiteilija ja suunnittelija. Pelikehitys voi olla yksittäisten ihmisten tai suurien pelistudioiden aloittama prosessi ja se voi olla laajuudeltaan niin pieni kuin iso. Monet kehittäjät käyttävät peliä kehittäessään pelimoottoreita, kuten esimerkiksi Unityä, Unreal Engineä ja CryEngineä, mikä tekee pelin kehittämisestä helpompaa. Se myös mahdollistaa lukuisten toimintojen, kuten 2D- ja 3D-grafiikoiden ja fysiikan mallinnuksen, törmäyksen havaitsemisen, äänten ja monen muun, käyttämistä. (freeCodeCamp, 2019.)

1.2 Pelin kuvaus ja tavoitteet

Tavoitteena on luoda realistista ajosimulaatiota mukaileva 2D-ajopeli (verkkopeli) web-ohjelmointityökaluja, -kieliä ja metodeja käyttäen. Kaksiulotteisuudessa (2D)

on kaksi ulottuvuutta eli pituus ja leveys. Tässä pelissä 2D viittaa siihen, että pelin kuvakulma on ylhäältä alas eli pelikenttä ja sen objektit näytetään ylhäältä käsin.

Pelissä olisi mahdollisuus pelata useita eri ajotyyppejä/kisamuotoja muita pelaajia vastaan samalla kun ajoneuvojen ominaisuudet muuttuvat ajon aikana (mm. polttoaineen, renkaan ja moottorin kuluminen). Pelissä olisi myös mahdollisuus muuttaa ja säätää auton säätöjä sekä ominaisuuksia mieleisekseen ja kustomoida itselleen sopivan näköinen auto.

Pelaajilla olisi mahdollisuus pelata sekä rekisteröityneenä että ilman. Pelaajien ajamat kierrosajat tallentuisivat tietokantaan (ei-rekisteröityneillä vain väliaikaisesti) ja sieltä tulostauluihin muiden nähtäväksi. Näin pelaajat voisivat kilpailla toisiaan vastaan aika-ajo menetelmällä, mahdollisesti myös moninpelissa keskenään toisiaan vastaan kilpailumuodossa. Rekisteröityneillä pelaajilla olisi mahdollisuus tallentaa heidän autojensa säädöt ja värit tietokantaan, josta he voisivat myöhemmin ladata ne seuraavaan pelikertaan.

Tarkoituksena olisi kehittää myös muita pienempiä ominaisuuksia peliin, kuten esimerkiksi käsijarru, turbo ja animoidut grafiikat. Peliin halutaan useita eri ominaisuuksia/toimintoja, joten selvennetään vaatimuslistan avulla, mitä pelin pitäisi sisältää ja kuinka tarpeellisia ominaisuudet pelissä ovat. Taulukossa 1 on tarkennettu lista vaatimuksista, joita pelissä pitäisi olla. Myös prioriteetit on ilmoitettu taulukossa.

Taulukko 1. Projektin vaatimuslista.

Viittaus	**Kuvaus**	**Prioriteetti**
F1	Ajan otto	1
F2	Renkaiden/polttoaineen/moottorin kuluminen	1

F3	Auton säätöjen muokkaus	1
F4	Auton kustomointi	1
F5	Eri kilpamuotojen valitseminen (Formula, Rally, Gymkhana)	1
F6	Kiihtyvyys ja jarrutus grafiikat	2
F7	Ohjauspyörän liikkeen grafiikka	2
F8	Sekundääriset ominaisuudet (turbo, käsijarru, kers, hybrid)	2
F9	Dynaaminen/mukautuva pelikartta	2
F10	Kiihtyvyyden, jarrutuksen ja ohjauksen herkkyyden säätö	3
F11	Automaatin tai manuaalin valinta	3
F12	Graafiset animaatiot	3
F13	Delta-ajat kierroksilta	3
F14	Tekoälypelaajia	3
F15	Pelidatan tallennus tietokantaan (kierrosajat, säädöt, auton värit)	1
F16	Rekisteröinti/kirjautumisominaisuus	1
F17	Tulostaulu	1
F18	Dynaaminen valikko	2

F19	Online/monipelimoodi	3
F20	Realistisen näköinen auto (muoto, renkaat, yksityiskohdat)	1
F21	Realistiset ajo ominaisuudet (kiihdytys, jarrutus, sivuluisu)	2
F22	Realistinen radanpinta (kuiva tai märkä asfaltti, sora)	2
F23	Eri ratojen valinta	3

1.3 Tutkimuskysymykset

Web-kehitys ja peliohjelmointi ovat kaksi eri ohjelmointialaa, joissa käytetään pitkälti eri kieliä ja työkaluja eri tarkoituksiin, web-kehityksessä nettisivujen luontiin ja peliohjelmoinnissa pelien luontiin.

Tutkimuskysymykset ovatkin, miten realistisien, laaja-alaisen ja kattavan verkkopelin voi luoda web-kehitykseen tarkoitetuilla kielillä/työkaluilla ja miten hyvin web- ja pelikehitys kulkevat rinnakkain?

Monet suositut pelimoottorit, kuten Unity ja Unreal Engine, sisältävät fysiikkamoottorin, jonka avulla voidaan simuloida erilaisten fyysisten esineiden fysikaalisia ominaisuuksia, kuten painoa tai kappaleen kovuutta. Esitetään tämän perusteella vielä tutkimuskysymys: Miten ajoneuvot saadaan pelissä käyttäytymään realistisella tavalla?

Tavoitteena on siis selvittää, kuinka ajoneuvot käyttäytyvät esimerkiksi kiihdyttäessä, jarruttaessa ja sivuluisussa, kun kullekin ajoneuvoille määritellään sekä teho että paino. Esimerkiksi miten nopeasti auto kiihtyy haluttuun nopeuteen, kuinka

nopeasti auto pysähtyy, kun auton jarrua painetaan tai kun kaasupoljinta nostetaan ja miten pitkän sivuluisun auto voi tehdä tietyssä nopeudessa ja millaisessa kulmassa? Näihin kysymyksiin yritetään löytää vastaus käyttäen hyväksi fysiikan kaavoja ja funktioita, joita siten sovelletaan lopullisessa pelissä.

Lisäksi voidaan esittää vielä seuraavat kysymykset: Kuinka voidaan mitata pelin/ohjelman laatua/onnistumista, mitkä tekijät määrittelevät, onko peli realistinen ja mitkä ovat käytännöllisiä tapoja mitata/vertailla pelin/ohjelman toimintoja/käyttöystävällisyyttä ja kokonaistulosta.

2 TEORIATAUSTA

2.1 Web-kehityksen ja pelikehityksen erot

Vaikka molemmat ovat ohjelmistokehityksen aloja, web-kehitys ja peliohjelmointi eroavat toisistaan monella eri alueella: Web-kehityksessä keskitytään pääasiallisesti (joko staattisten tai dynaamisten) nettisivujen ja web-sovellusten kehittämiseen, kun taas peliohjelmointi keskittyy interaktiivisien pelien kehittämiseen ja rakentamiseen erilaisilla alustoilla, kuten esimerkiksi konsoleilla, PC:llä ja mobiilina. (Rafath, 2023; Tripathi, 2023.)

Web-kehityksessä työkaluina ja teknologioina käytetään ohjelmointikieliä, muun muassa HTML:ää, CSS:ää, JavaScriptiä ja laajan kattauksen web-kehyksiä, kuten esimerkiksi jQueryä ja Bootstrapiä. Pelikehityksissä käytettävät teknologiat ovat peleihin erikoistuneita pelimoottoreita ja kieliä, kuten C++, C#, UnityScript, Unity, Unreal Engine ja GameMaker Studio. (Rafath, 2023; Tripathi, 2023.)

Web-kehityksessä visuaalisten grafiikoiden ja animaatioiden tekeminen on mahdollista, mutta niiden taso on yleensä rajallinen verrattuna pelikehitykseen, jossa näyttävien ja yksityiskohtaisten grafiikoiden, efektien, animaatioiden ja simulaatioiden kehittäminen on yleisempää. Myös fysiikkamallien kehittäminen, kuten esimerkiksi painovoima, törmäykset, objektien käytös, vaatii pelialalla enemmän kuin web-kehityksessä. (Rafath, 2023.)

Lisäksi web-kehityksessä kohdeyleisö on laajempi kuin pelikehityksessä, jossa pelejä kehitetään yleensä aina tietyille ikäryhmille (Tripathi, 2023). Web-kehityksessä on laaja mahdollisuus erilaisiin uramahdollisuuksiin frontendistä backendiin, kun taas pelikehityksessä on enemmän tiettyyn alaan keskittyviä uramahdollisuuksia. Web-kehityksen projektit ovat myös suoraviivaisempia, nopeampia ja pienemmillä tiimeillä tehtyjä kuin pelikehityksen projektit, jotka tapaavat olla enemmän aikaa vieviä ja monimutkaisempia isommilla ja monipuolisimmilla projektitiimeillä. (Rafath, 2023.)

Molempien alojen eri aspekteista huolimatta on hyvä huomata, että molemmilla aloilla voi olla päällekkäisyyksiä, kuten esimerkiksi selainpohjaiset pelit tai pelinkaltaiset verkkosovellukset. Jotkut kehittäjät saattavat työskennellä molemmilla aloilla, hyödyntäen taitojaan niissä kummassakin. (Rafath, 2023.)

2.2 Fysikaalisia taustoja

Tähän opinnäytetyöhön tarvitaan autoihin soveltuvia fysiikkakaavoja, joiden avulla voidaan testata ja mitata tarkasti ajoneuvojen ominaisuuksia. Tässä opinnäytetyössä käytetään Marco Monster nimisen henkilön esittämiä fysiikkakaavoja, joilla voidaan hallita ajoneuvon käyttäytymistä. Tämän lisäksi tässä työssä käytetään myös vertailun vuoksi Spacejackin luomaa omaa versiota, joka on myös taltioinut Marco Monsterin vanhan alkuperäisen demon, joka ei edellä mainitussa lähteessä enää toimi vanhentuneen verkon takia.

Tässä työssä keskitytään ajoneuvon kiihtyvyys- ja jarrutusvoimaan sekä kääntymiseen liittyviin ominaisuuksiin. Alla näkyy muutama fysiikkakaava, joita olisi tarkoitus tässä opinnäytetyössä soveltaa.

Ajoneuvon nopeus ja kiihtyvyys ominaisuudet olisi tarkoitus ratkaista seuraavaa yhdeksää eri fysiikan kaavaa hyödyksi käyttäen:

$$F_{Traction} = \boldsymbol{u} * F_e \quad (1)$$

Missä $F_{traction}$ on pitovoima, $\boldsymbol{u}$ vektoriyksikkö, mihin suuntaan ajoneuvo on menossa ja F_e on moottoriteho. Ilmanvastuksen voima F_{drag} saadaan selville kaavalla:

$$F_{drag} = -C_{drag} * v * |v| \quad (2)$$

Missä C_{drag} on vakioarvo ilmanvastukselle ja v on nopeus yksikkö. Vierintävastuksen voima F_{rr} lasketaan kaavalla:

$$F_{rr} = -C_{rr} * v \quad (3)$$

Missä v on edelleen nopeusyksikkö ja C_{rr} on vakioarvo vierintävastukselle, tässä tapauksessa C_{drag} kertaa 30. Marco Monster perustelee vakioarvon C_{rr} sillä, että ajoneuvon matalalla nopeudella vierintävastus on suurempi kuin ilmanvastus, joka tosin vaihtuu nopeuden kasvaessa. Noin 100 km/h, eli suurin piirtein 30 m/s vauhdissa vastukset ovat yhtä suuret, minkä vuoksi C_{rr} on C_{drag} kertaa 30. (Monster, 2024.)

Ajoneuvon kohtisuoraisen tehon summa F_{long} saadaan, kun lisätään pitovoimaan $F_{traction}$ ilman- ja vierintävastuksen voimat, eli F_{drag} ja F_{rr} kaavan 4 mukaisesti. Kun ajoneuvon ilman- ja vierintävastuksen yhteen laskettu voima ylittää pitovoiman, ajoneuvo on saavuttanut maksiminopeuden. (Monster, 2024.)

$$F_{long} = F_{traction} + F_{drag} + F_{rr} \quad (4)$$

Edellä mainitun kohtisuoraisen tehon avulla voidaan laskea ajoneuvon kiihtyvyys a kaavalla:

$$a = F/M \quad (5)$$

Missä F on ajoneuvon nettoteho, eli tässä tapauksessa F_{long} ja M on ajoneuvon massa. Kiihtyvyydestä päästään ajoneuvon nopeuteen v kaavalla:

$$v = v + \Delta t * a \quad (6)$$

Missä Δt on ajanlisäys sekunneissa fysiikkamoottorin kutsujen välillä. Lopulta nopeus v määrää ajoneuvon sijainnin p lopullisella kaavalla:

$$p = p + \Delta t * v \quad (7)$$

Jarrutukseen kiteytyviä ominaisuudet olisi tarkoitus saada selville vain kahdella olla olevalla kaavalla:

$$F_{long} = F_{braking} + F_{drag} + F_{rr} \quad (8)$$

Missä $F_{braking}$ on jarrutusvoima, joka määrittyy kaavalla:

$$F_{braking} = -\boldsymbol{u} * C_{braking} \quad (9)$$

Missä $C_{braking}$ on vakioarvo jarrutukselle. Yksinkertaisesti siis jarrutusvoima vähentää ilman- ja vierintävastuksen tavoin kohtisuoraa tehoa F_{long} näin vähentäen ajoneuvon nopeutta.

Lopuksi tarkoitus olisi myös keskittyä ajoneuvon kääntymiseen liittyviin fysikaalisiin ominaisuuksiin. Hitaalla nopeudella voidaan soveltaa yksinkertaista kaavaa, mutta kun ajoneuvon nopeus kasvaa, ei voida enää välttämättä odottaa ajoneuvon liikkuvan renkaiden osoittamaan suuntaan. Esimerkki kaava tästä on sivuluisto kaava, joka tarkoittaa kulmaa ajoneuvon nopeusvektorin ja suunnan väliltä. Tämä saadaan kaavalla. (Monster, 2024.)

$$\beta = arctan\left(\frac{v_y}{v_x}\right) \quad (10)$$

Missä β on ajoneuvon sivuluistokulma, v_y ajoneuvon sivuttaisnopeus ja v_x on ajoneuvon kohtisuorainen nopeus.

Nopeissa käännöksissä ajoneuvon renkaat kehittävät sivuttaista voimaa, joka on riippuvaista luistokulmasta eli kulmasta, mihin suuntaan rengas osoittaa ja minne rengas todellisuudessa matkaa. Etu ja takarenkaiden luistokulmat voidaan laskea kahdella seuraavilla kaavoilla. (Monster, 2024.)

$$a_{front} = ARCTAN\left(\frac{v_{lat} + \Omega * b}{|v_{long}|}\right) - delta * sgn(v_{long}) \quad (11)$$

$$a_{rear} = ARCTAN\left(\frac{v_{lat} - \Omega * c}{|v_{long}|}\right) \quad (12)$$

Edellä mainituissa kaavoissa a_{front} on ajoneuvon etupyörien luistokulma a_{rear} puolestaan takapyörien luistokulma, v_{lat} on ajoneuvon sivuttaisnopeus, v_{long} ajoneu-

von pitkittäisnopeus, b on pituus ajoneuvon keskipisteestä etuakselille ja c keskipisteestä taka-akselille. Molemmissa kaavoissa Ω on ajoneuvon kulmanopeus ja delta eturenkaan kulma auton pitkittäissuuntaan nähden.

Näiden edellä mainittujen fysikaalisten kaavojen avulla olisi tarkoitus luoda realismia mukaileva ajopeli, jonka myötä ajoneuvot käyttäytyvät fysikaalisesti realistisella tavalla.

On kuitenkin hyvä mainita muutama ongelma liittyen Monsterin (2024) ja Spacejackin (2024) lähteisiin. Aiemmin mainittiin, että vierintävastuksen vakioarvo C_{rr} on 30 kertaa C_{drag}. Lähteessään Monster mainitsee epäillyksensä kyseiseen asiaan, eikä hän omien sanojen mukaan kyennyt vahvistamaan teoriaa missään. Lisäksi useat fysiikkakaavat vaatisivat sellaisia tietoja ajoneuvoista, mitä ei välttämättä ole mahdollista löytää mistään. Esimerkiksi Monsterin mukaan rengasvalmistajat ovat hyvin vaiteliaita kertomaan heidän renkaihinsa liittyvistä ominaisuuksista. Myös molempien (Monsterin ja Spacejackin) demoissa on hieman eroavaisuuksia, ne eivät välttämättä sisällä Monsterin itse esittämiä fysiikkamalleja. Tämän vuoksi tässä projektissa joudutaan ajoneuvojen joitakin ominaisuuksia hienosäätään paikoin summittaisesti.

3 TEKNOLOGIA

3.1 HTML, JavaScript, CSS

HTML on oleellinen ohjelmointikieli web-kehityksessä, jolla luodaan rakenne verkkosivuille. HTML on yhdistelmä merkintäkieliä ja hypertekstiä. Hyperteksti määrittelee linkin verkkosivujen välillä ja merkintäkieli määrittelee tekstidokumentin HTML-tageilla. HTML:n ymmärtäminen on oleellista kaikille, jotka haluavat rakentaa uraa Web-kehityksessä, koska se muodostaa kulmakiven kaikille verkkosivuille. (GeeksForGeeks, 2023b.) Viimeisin versio HTML:stä on HTML5, joka sisältää uusia elementtejä, attribuutteja ja APIja. Tämän version opetteleminen on suositeltavaa, jotta pysyttäisiin ajan tasalla modernin web-kehityksen käytännöistä. (GeeksForGeeks, 2019.) Yhdessä CSS:n ja JavaScriptin kanssa HTML muodostaa ytimen frontend-web-kehitykseen mahdollistaen interaktiivisien ja visuallisesti näyttävien verkkosivujen luonnin. (GeeksForGeeks, 2023b.) HTML:n varsin yksinkertaisen rakenteen avulla ohjelmointikoodi on mahdollista pitää selkeänä tässä työssä, mikä helpottaa sekä koodien luontia että virheiden korjausta.

CSS on kieli, jolla määritellään verkkosivujen ulkoasu, mikä tehostaa verkkosivujen visuaalista ilmettä. (GeeksForGeeks, 2023a.) Ilman CSS:ää verkkosivun ulkomuoto tulisi olemaan hyvin pelkistetty, joten pelin käyttöliittymän ehostamiseksi on perusteltua käyttää CSS:ää tässä työssä.

JavaScript on maailman suosituin Web-kehityksessä käytettävä ohjelmointikieli, jota sekä käyttäjä että palvelin voivat käyttää. Kyseinen kieli tunnetaan myös komentokielenä verkkosivuille, jonka avulla voidaan luoda dynaamisia ja interaktiivisia sisältöjä verkkosivuille. JavaScriptillä on useita kirjastoja ja kehyksiä, kuten esimerkiksi jQuery, React ja Angular. Se tarjoaa monia sisäänrakennettuja funktioita ja metodeja ominaisuuksien lisäämiseen verkkosivuille helposti. (GeeksForGeeks, 2023c.) Siinä missä HTML ja CSS luovat rakenteen ja tyylin verkkosivun käyttöliittymän, JavaScript ohjaa verkkosivujen sisäisiä ominaisuuksia/funktioita, kuten

myös tässä työssä pelin erilaisia funktioita, mikä tekee JavaScriptistä merkittävän ohjelmointikielen tässä projektissa.

3.2 Git/GitHub

Git on Linus Torvaldsin vuonna 2005 luoma ilmainen ja avoimen lähdekoodin versionhallintajärjestelmä, jonka avulla voidaan hallita suurten ja pienten projektien etenemistä nopeasti ja tehokkaasti. Koska Git on helppokäyttöinen ja se tarjoaa tehokkaan suorituskyvyn pienellä ”jalanjäljellä” (Git, 2024; Kinsta, 2023), se on erinomainen versionhallintatyökalu tähän työhön. Gitin merkitystä ei voi tässä projektissa väheksyä, sillä tämän avulla voidaan pysyä selvillä projektissa tehdyistä muutoksista ja seurata, mihin suuntaan projekti on menossa.

GitHub on tuottoa tavoitteleva yhtiö, joka tarjoaa pilvipohjaisen Git-tietovarasto-verkkopalvelun. Tämä tekee Gitin käyttämisestä yksittäisille ihmisille tai tiimeille helppoa sekä versionhallinnassa että yhteistyössä. (Kinsta, 2023.) Tämän opinnäytetyön projektin materiaalit ja versiot tullaan tallentamaan yhtiön tietokantaan. GitHubin käyttäjäystävällisen käyttöliittymän ansiosta sivusto toimii loistavana apuna tähän työhön.

3.3 MySQL, PHP

MySQL on maailman suosituimpia avoimen lähdekoodin relaatiotietokantahallintajärjestelmiä. Monien yritysten, kuten Facebookin, Netflixin, Uberin ja Airbnb:n sovellukset hyödyntävät MySQL järjestelmää. Sen sijaan, että kaikki tieto tallennetaan yhteen suureen tietovarastoon, relaatiotietokannat tallentavat tiedon erillisiin taulukoihin. MySQL tukee useita suosittuja ohjelmointikieliä, kuten esimerkiksi PHP:tä, Pythonia ja Java/JDBC:tä. Avoimen lähdekoodin lisäksi MySQL on helppo asentaa/käyttää ja se on silti riittävän luotettava ja skaalautuva (Oracle, 2024) soveltuen näin tähän työhön mainiosti. Koska tässä työssä halutaan luoda kirjautumis-/rekisteröitymisjärjestelmä ja mahdollisuus tallentaa erilaisia tietoja tietokantaan, MySQL:n merkitys tässä työssä on erittäin suuri.

PHP on avoimen lähdekoodin palvelinpuolen komentokieli, jota monet kehittäjät käyttävät web-ohjelmoinnissa. PHP:tä käytetään pääsääntöisesti web-palvelimien luontiin. PHP:n etuina voidaan pitää sen alustariippumattomuutta, eli PHP:tä voidaan pyörittää kaikilla alustoilla, kuten Windowsissa, MacOS:ssä ja Linuxissa. Lisäksi PHP on avoimen lähdekoodin kieli, jonka alkuperäinen koodi on saatavilla kaikille, jotka haluavat rakentaa sen ympärille. PHP voidaan myös helposti liittää kaikkiin tietokantoihin riippumatta siitä, onko tietokanta relaatiotietokanta vai ei, kuten esimerkiksi MySQL, Postgress sekä MongoDB. Edellä mainittujen ominaisuuksien lisäksi PHP-koodia on mahdollista upottaa osaksi HTML-koodiin. (Chris, 2021.) Se tekee PHP:stä juuri sopivan komentokielen tähän projektiin. PHP:n avulla on varsin helppoa tallettaa palvelimen kautta tietokantaan tietoa suoraan HTML-koodista, jonka takia PHP:n käyttö on perusteltua tässä työssä.

3.4 Visual Studio Code

Visual Studio Code tai myös VSCode on Microsoftin Electron Frameworkin avulla kehittämä teksti-/lähdekoodieditori, joka voidaan asentaa niin Windowsille, Linuxlle ja MacOSille. VSCode tukee lukuisia määriä eri ohjelmointikieliä ja on myös ideaali niin kevyille kuin myös kohtuullisen monimutkaisille projekteille, kuten web-ohjelmoinnille. Lisäksi VSCode on kevyt ohjelmisto, joka vie vain vähän tallennustilaa koneesta. Sillä on myös nopea käynnistymisaika.

Myös työskentely Gitin kanssa on tehty helpommaksi VSCodessa, jossa Git-muokkauksia voidaan tehdä suoraan editorista käsin. (GeeksForGeeks, 2023d.) VSCodella on erittäin helppoa vertailla eri kooditiedostoja keskenään, kuten myös muutoksia, mitä eri tiedostoihin on tehty. Lisäksi HTML-koodia pystyy ajamaan vaivatta livenä paikallisesti, minkä vuoksi VSCode soveltuu mainiosti tähän projektiin.

3.5 MAMP

MAMP on ilmainen Web-ohjelmistopino, jolla voidaan ajaa PHP-koodia paikallisessa palvelinympäristössä. MAMP tukee komentokielessä PHP:tä, Pythonia, Perliä ja Rubya, tietokannassa MySQL:a ja palvelimessa sekä Apachea että Nginxiä. MAMP tarjoaa lisäksi työkaluja, joita tarvitaan WordPressin ajamiseen testaus- ja kehittämistarkoitukseen. Myös projektien testaus mobiililaitteilla on mahdollista. (MAMP-Mac, 2024; MAMP-Windows, 2024.)

Tässä projektissa palvelimena käytetään Apachea, tietokantana ja komentokielenä aiemmissa luvuissa mainittua MySQL:a ja PHP:ta, minkä vuoksi MAMP on juuri sopiva tähän opinnäytetyöhön. Siinä missä VSCodella pystyy ajamaan helposti HTML-koodia, MAMP:n avulla on kätevämpää ajaa PHP-koodia. Myös tietokantojen ja niiden kyselyiden testaaminen on helpompaa phpMyAdminin avulla. On mahdollista, että on olemassa parempiakin ohjelmistopinoja kuin MAMP, kuten esimerkiksi XAMPP ja WAMP. Aiemmissa projekteissa MAMP-ohjelma on itse asiassa muutaman kerran kaatunut sitä käytettäessä. Ongelma on tosin korjautunut uudelleen asentamisen jälkeen. Yleisesti MAMP:n uskotaan toimivan tässä projektissa riittävän hyvin.

4 TYÖN SUUNNITTELU

Opinnäytetyön tekeminen jaetaan kolmeen eri vaiheeseen: ensimmäisessä vaiheessa luodaan itse peli ja sen funktiot toimimaan pelin kehittäjän ja yleisen fysikaalisen periaatteiden vaatimalla tavalla. Tässä vaiheessa aloitetaan myös etsimään aineistoja, jolla vastataan tutkimuskysymykseen, kuinka realistisen oloisen pelin voi luoda käyttäen pelkästään web-ohjelmointiin käytetyillä kielillä ja työkaluilla.

Työn toisessa vaiheessa perusmekaniikaltaan valmis peli siirretään ajettavaksi MAMP-nimiseen ohjelmaan, jossa voidaan kehittää ja suorittaa paikallisessa palvelimessa. Tässä vaiheessa peli kytketään kiinni tietokantaan, jossa pelissä tapahtuvat ja/tai tallennetut voidaan varastoida tietokantaan ja ladata sieltä uudelleen peliin halutulla tavalla.

Viimeisessä vaiheessa hiotaan pelissä valmiiksi vielä pienet tai suuremmat ensimmäisessä vaiheessa kesken jääneet ja puutteelliset asiat valmiiksi ja luodaan mahdollisuus moninpeliin.

Kussakin vaiheessa luodaan ja testataan erillisessä tiedostossa peliin tulevia komponentteja ja funktioita, jotka valmiiksi saatuaan liitetään mukaan varsinaiseen projektiin.

4.1 Ensimmäinen vaihe

Projektin ensimmäisessä vaiheessa testattiin ja luotiin vaatimuslistan F1-, F2-, F3-, F4- ja F5-ominaisuudet eli kaikki ensimmäisen prioriteetin kohdat projektin ensimmäisessä vaiheessa. F1:ssä vaadittiin ajanottoa, joka toimii, kun pelin ajoneuvo ylittää tarkastuspisteeksi (checkpoint) määriteltyyn kohtaan. Kun pelaaja on ylittänyt riittävän monen tarkastuspisteen ja lopulta maalilinjaan (finish line), aika tulostuu näytölle.

Kuvissa 1, 2 ja 3 näkyy ajanoton ja siihen liittyen hitObject-funktion toiminta koodissa. Kuvassa 1 on hitObject-funktion toiminta, joka tarkistaa, tapahtuuko funktion kahdella valitulla esineiden välillä osuma return-funktion palauttaessa vastauksen.

```
475   function hitObject(main, other) {
476     var mainleft = main.x;
477     var mainright = main.x;
478     var maintop = main.y;
479     var mainbottom = main.y;
480     var otherleft = other.x;
481     var otherright = other.x + (other.width);
482     var othertop = other.y;
483     var otherbottom = other.y + (other.height);
484     var crash = true;
485     if ((mainbottom < othertop) || (maintop > otherbottom) || (mainright < otherleft) || (mainleft > otherright)) {
486       crash = false;
487     }
488     return crash;
489   }
```

Kuva 1. hitObject-funktio.

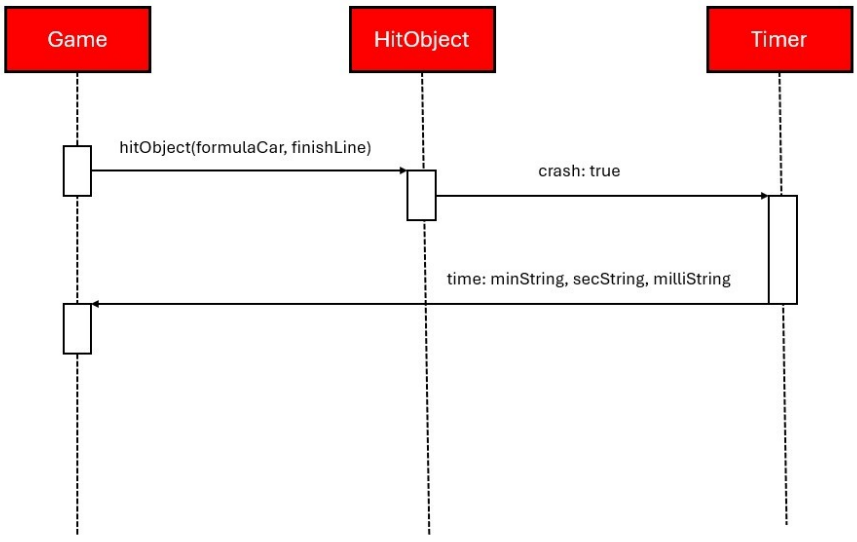
Kuvassa 2 nähdään esimerkkejä hitObject-funktion toiminnasta, kun ajoneuvo osuu kuhunkin objektiin (checkPoint, checkPoint2 ja finishLine). Kuvassa 3 puolestaan on Timer-funktio eli ajanoton toiminta. Kuvassa 4 on UML-sekvenssikaavio formulaCar-ajoneuvon ja finishLine-objektin toiminnasta.

```
498       if(hitObject(formulaCar, checkPoint)){
499         ch1 = 1;
500         console.log("You hit!")
501       } else if (hitObject(formulaCar, checkPoint2) && ch1 == 1) {
502           ch2 = 1;
503           console.log("second")
504       } else if (hitObject(formulaCar, finishLine) && ch1 == 1 && ch2 == 1) {
505           console.log("Final")
506
507           ch1 = 0, ch2 = 0;
508           clearInterval(interval);
509           minString = min, secString = sec, milliString = mil;
510           document.getElementById('lastTime').innerHTML = minString + ":" + secString + ":" + milliString;
511           min = 0, sec = 0, mil = 0;
512           minString = min, secString = sec, milliString = mil;
513           document.getElementById('time').innerHTML = minString + ":" + secString + ":" + milliString;
514           interval = setInterval(Timer, 10);
515
516       } else if (hitObject(formulaCar, fence1)) {
517         clearInterval(interval);
518         stopSession();
519         //game = window.cancelAnimationFrame(draw);
520         console.log("Hit");
521       }
```

Kuva 2. Esimerkki hitObject-funktion toiminnasta.

```
667     function Timer () {
668         mil++
669
670         if(mil <= 9){
671           milliString = "0" + mil;
672         }
673
674         if (mil > 9){
675           milliString = mil;
676
677         }
678
679         if (mil > 99) {
680           sec++;
681           secString = "0" + sec;
682           mil = 0;
683           milliString = "0" + 0;
684         }
685
686         if (sec > 9){
687           secString = sec;
688         }
689
690         /*if (sec < 60){
691             minString = "0";
692         }*/
693
694         if (sec > 59){
695             min++;
696             minString = min;
697             sec = 0;
698             secString = "0" + sec;
699         }
700
701         document.getElementById('time').innerHTML = minString + ":" + secString + ":" + milliString;
702     }
```

Kuva 3. Timer-funktio.



Kuva 4. UML-sekvenssikaavio hitObject-funktion toiminnasta.

F2:ssa vaadittiin kulumisfunktiota eli toimintoa, jonka aikana auton renkaat, polttoaine yms. kuluvat ajon aikana. Tämäkin saatiin onnistumaan perustasolla, mutta ei vielä fysikaalisesti realistisella tasolla. Kuvassa 5 on esimerkki polttoaineen kulumisesta, kun ajoneuvon vauhti on enemmän kuin nolla yksikköä, ajoneuvon polttoaine kuluu vähitellen. Kun fuel-yksikkö on nolla tai vähemmän, ajoneuvon speed-yksikkö vaihtuu nollaksi, eikä ajoneuvo pitäisi enää tämän jälkeen liikkua.

```
if(formulaCar.speed == speed || formulaCar.speed == -speed) {fuel = fuel - speed*0.01;}
if (fuel <= 0) {formulaCar.speed = 0; console.log(speed)}

document.getElementById("fuel").innerHTML = "Fuel: " + fuel;
```

Kuva 5. Esimerkki polttoaineen kulumisesta.

F3-, F4- ja F5-vaatimuksen ominaisuudet (auton säätöjen muokkaus, värin valinta/kustomointi, kilpamuodon valitseminen) saatiin niin ikään luotua, jokseenkin vajavaisesti ja perustasolla. Tulokset näkyvät seuraavissa kuvissa. Kuvassa 6 näkyy koodi ajoneuvon säätö- ja valikko-ominaisuuksista. Polttoaineen lisääminen tapahtuu numero syötteestä, downforce-yksikkö range-syötteestä ja kisamuodon sekä värin valinta pudotusvalikosta.

```
<div id="menuDIV">
  <label for="refuel">Add fuel: </label>
  <input type="number" id="fuelAdd" name="refuel" min="1" max="50"><br>

  <input id="range_input" type="range" min="0" max="5"/>
  <p>Downforce: <output id="range_value"></output></p>

  <label>
    <select id="carForm" onchange="selectForm()">
        <option value="formula">Formula</option>
        <option value="rally">Rally</option>
        <option value="gymkhana">Gymkhana</option>
    </select>
</label>

<label>
  <select id="carColor" onchange="selectColor()">
      <option value="white">White</option>
      <option value="black">Black</option>
      <option value="red">Red</option>
      <option value="yellow">Yellow</option>
      <option value="green">Green</option>
      <option value="blue">Blue</option>
      <option value="orange">Orange</option>
  </select>
</label>
```

Kuva 6. Koodiesimerkki polttoaineen syötöstä, säätövalikosta, värin ja kisamuodon valinnasta.

Kuvassa 7 on koodia startFunction-funktiosta, joka joko käynnistää tai keskeyttää pelin ja päinvastoin näyttää säätövalikon tilanteen mukaan sekä hieman koodia range-valikon toiminnasta. Kuvassa 8 puolestaan näkyy funktiot selectForm ja selectColor, jotka määrittävät valitun kisamuodon ja ajoneuvon värin pudotusvalikon valinnan mukaan.

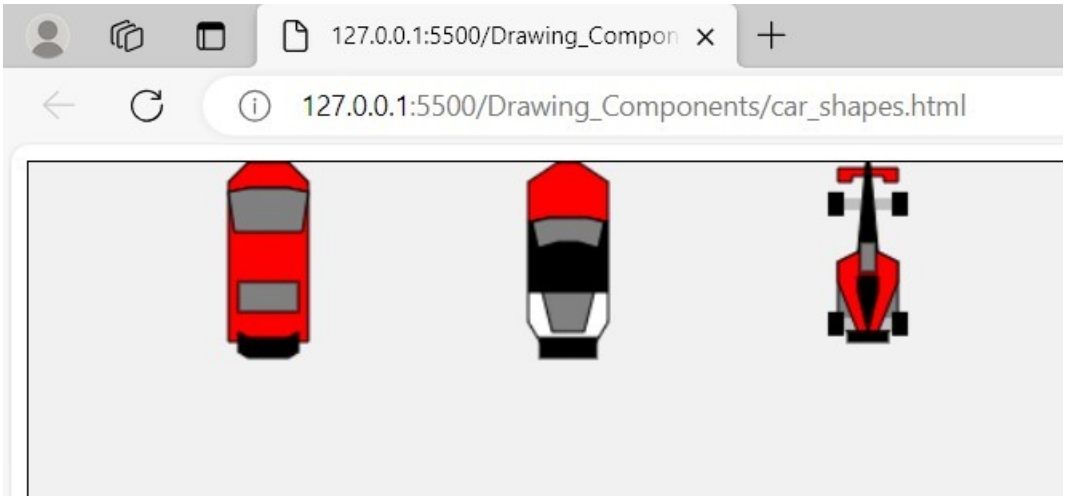
```
87    const rangeValue = document.querySelector("#range_value");
88    const rangeInput = document.querySelector("#range_input");
89    rangeValue.textContent = rangeInput.value;
90    rangeInput.addEventListener("input", (event) => {
91      rangeValue.textContent = event.target.value;
92      //turnR = value.textContent;
93      //turnL = value.textContent;
94      turn = rangeValue.textContent;
95    });
96
97    canvas.style.display = "none";
98
99    function startFunction() {
100     if (menuDIV.style.display === "none") {
101       menuDIV.style.display = "block";
102       cancelAnimationFrame(game);
103       canvas.style.display = "none";
104     } else {
105       menuDIV.style.display = "none";
106       canvas.style.display = "block";
107       var fuelAdd = document.getElementById("fuelAdd").value;
108       fuel = Number(fuelAdd);
109       turnS = turn;
110       if(tireCompo == "soft") {turnL = 80; turnR = 80;}
111       else if(tireCompo == "hard") {turnL = 100; turnR = 100;}
112
113       game = window.requestAnimationFrame(draw);
114     }
115   }
```

Kuva 7. Koodiesimerkki ajoneuvon säätötoiminnosta ja startFunction-toiminnosta.

```
117     function selectForm() {
118         var i = document.getElementById("carForm");
119         var selectForm = i.options[i.selectedIndex].value;
120         raceForm = selectForm;
121         console.log(selectForm);
122     }
123
124     function selectColor() {
125         var i = document.getElementById("carColor");
126         var selectColor = i.options[i.selectedIndex].value;
127         carColor = selectColor;
128         console.log(selectColor);
129     }
```

Kuva 8. Koodiesimerkki kilpamuodon ja auton värin valintafunktioista.

Ylimääräisessä F20-vaatimuksessa vaadittiin realistisen näköisten ajoneuvojen luontia. Canvas Api -toiminnon avulla pystyttiin piirtämään varsin hyvät ja realistisen näköiset kolme erinäköistä ajoneuvoa, jotka näkyvät kuvassa 9.



Kuva 9. Luodut ajoneuvot vasemmalta oikealle: ralli, gymkhana ja formula.

Kuvissa 10 ja 11 näkyy esimerkki formula-auton (formulaCar) luonnista koodissa. Esimerkiksi kuvassa 10 määritellään heti pelin alkuun muun muassa formula-auton x- ja y-sijainnit, speed-arvo ja piirretään Canvas Api -toiminnolla ajoneuvon eri osat. Kuvassa 11 nähdään vielä lisää ajoneuvon eri osien piirtämistä Canvas Api -toiminnon avulla.

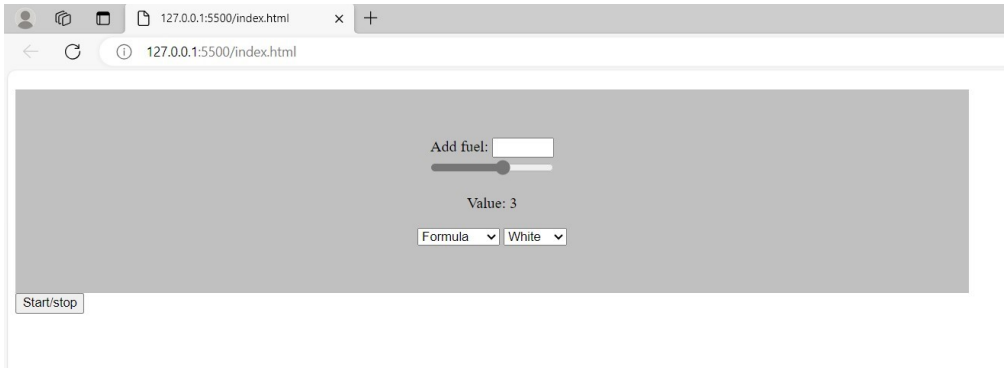
```
158     const formulaCar = {
159       x: 100,
160       y: 100,
161       angle: 0,
162       speed: 0,
163       moveAngle: 0,
164       color: "red",
165       color2: "white",
166       color3: "black",
167       color4: "gray",
168       color5: "silver",
169       draw() {
170         //Front spoiler
171         ctx.save();
172         ctx.translate(this.x, this.y);
173         ctx.rotate(this.angle);
174         ctx.beginPath();
175         ctx.moveTo(-15, -42);
176         ctx.lineTo(15, -42);
177         ctx.lineTo(15, -35);
178         ctx.lineTo(8, -35);
179         ctx.lineTo(8, -38);
180         ctx.lineTo(-8, -38);
181         ctx.lineTo(-8, -35);
182         ctx.lineTo(-15, -35);
183         ctx.closePath();
184         ctx.fillStyle = this.color;
185         ctx.fill();
186         ctx.stroke();
187
188         //Chassis
189         ctx.beginPath();
190         ctx.moveTo(-5, 0);
191         ctx.lineTo(5, 0);
192         ctx.lineTo(15, 5);
193         ctx.lineTo(15, 35);
```

Kuva 10. Koodiesimerkki formula-auton luonnista.

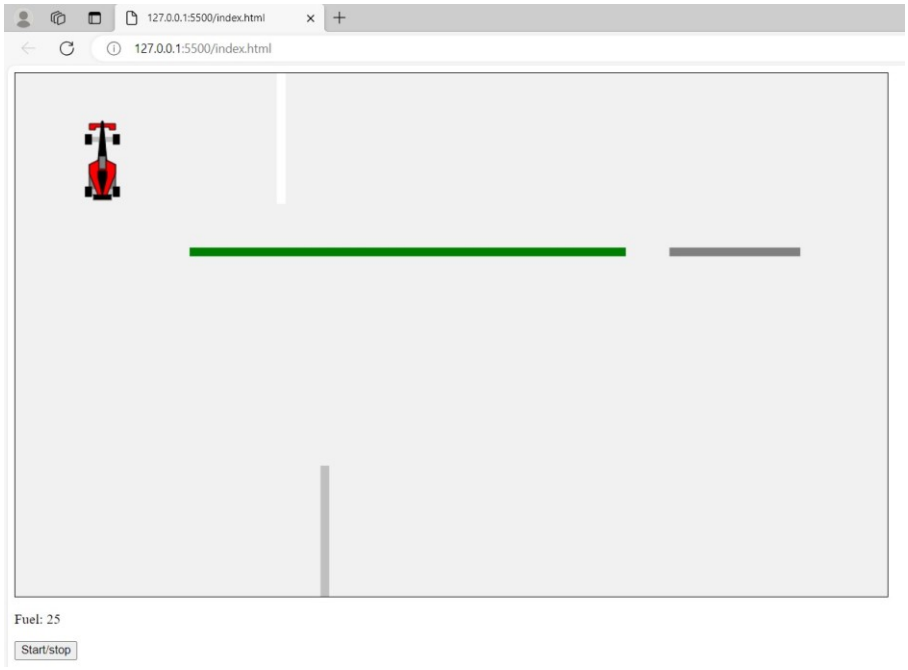
```
194         ctx.lineTo(10, 40);
195         ctx.lineTo(-10, 40);
196         ctx.lineTo(-15, 35);
197         ctx.lineTo(-15, 5);
198         ctx.closePath();
199         ctx.fillStyle = this.color4;
200         ctx.fill();
201         ctx.stroke();
202
203         //Suspensions
204         ctx.fillStyle = this.color5;
205         ctx.fillRect(-15, -27, 15, 6);
206         ctx.fillRect(0, -27, 15, 6);
207         ctx.fillRect(-15, 33, 15, 6);
208         ctx.fillRect(0, 33, 15, 6);
209
210         //Monocoque
211         ctx.beginPath();
212         ctx.moveTo(0, -45);
213         ctx.lineTo(1, -44);
214         ctx.lineTo(5, 0);
215         ctx.lineTo(-5, 0);
216         ctx.lineTo(-1, -44);
217         ctx.closePath();
218         //ctx.fillStyle = this.color3;
219         ctx.fillStyle = carColor;
220         ctx.fill();
221         ctx.stroke();
222
223         //Car body
224         ctx.beginPath();
225         ctx.moveTo(-5, 0);
226         ctx.lineTo(5, 0);
227         ctx.lineTo(15, 5);
228         ctx.lineTo(15, 15);
229         ctx.lineTo(5, 40);
```

Kuva 11. Koodiesimerkki formula-auton luonnista.

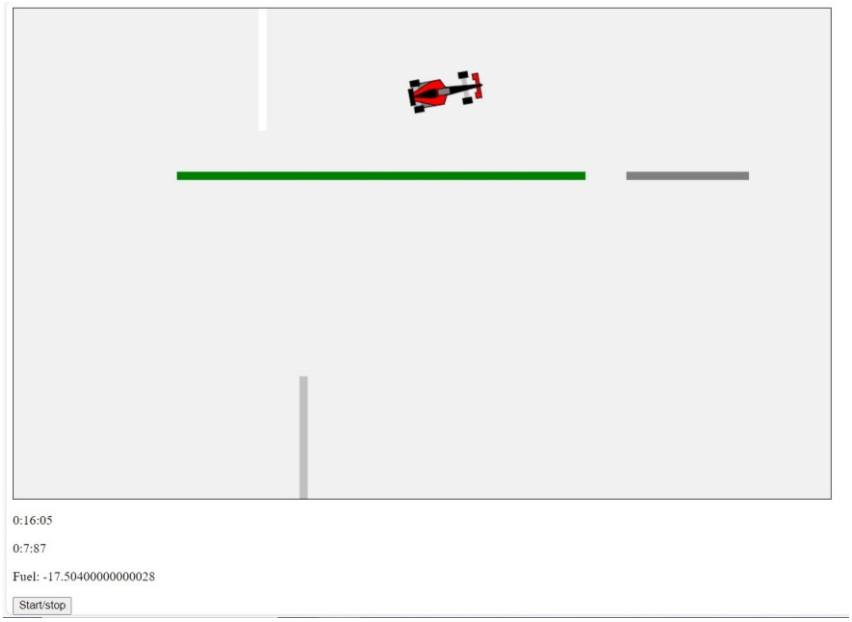
Kuvista 12, 13 ja 14 näkee, miltä peli näyttää, kun vaatimukset yhdistetään lopulliseen projektiin. Kuvassa 12 näkyy pelin alkuvalikko, jossa voi valita polttoaineen määrän, downforce-yksikön yhdestä viiteen sekä kilpamuodon ja ajoneuvon värin (osan ajoneuvosta). Kuvassa 13 puolestaan näkyy pelinäkymä heti pelin aloittamisen jälkeen (Start/Stop-napin painamisen jälkeen) ja kuvassa 14 pelinäkymä hetken pelaamisen jälkeen.



Kuva 12. Pelin alkuvalikko.



Kuva 13. Pelinäkymä pelin alussa.



Kuva 14. Pelinäkymä lyhyen pelaamisen jälkeen.

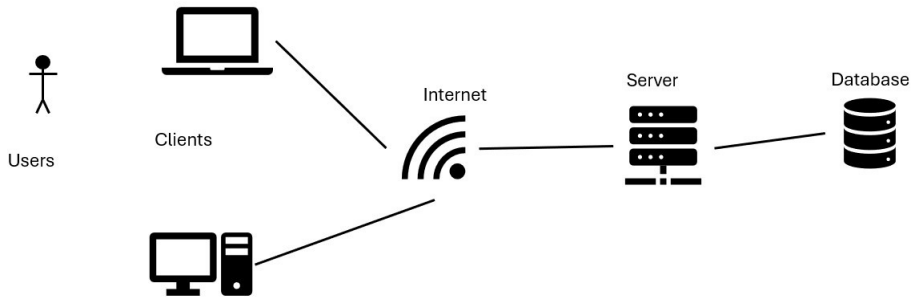
Kun kaikki prioriteetin 1 vaatimuksen (F1, F2, F3, F4, F5 ja F20) komponentit on yhdistetty, peli ja sen valikot toimivat tässä vaiheessa varsin moitteettomasti perustasolla. Muutamia virheitä/puutteita ohjelmassa on kuitenkin vielä ratkaistavana: auto ei pysähdy heti, kun polttoaine on ajoneuvosta loppu, vaan vasta sitten kun nostaa nuolinäppäimen (joka liikuttaa ajoneuvoa). Tämän jälkeen ajoneuvo ei liiku ollenkaan nuolinäppäimen painamisesta huolimatta. Lisäksi kun ajoneuvo osuu seinään (fence1 esimerkkinä), peli ei pysähdy, vaikka hitObject-funktio rekisteröi tapauksen, jonka pitäisi pysäyttää ajoneuvo. Myös ajanottoa pitää hieman parantaa (tallentaa kunkin kierroksen kierrosaika). Pientä parannusta pitää tehdä myös auton säätö/kustomointiominaisuuksiin. Lisäksi muun muassa renkaiden ja moottorin kulumisominaisuudet täytyy lisätä.

Edellä mainitut puutteet/virheet tullaan korjaamaan joko heti toisessa kehitysvaiheessa tai viimeistään kolmannessa ja viimeisessä vaiheessa.

4.2 Toinen vaihe

Toisessa vaiheessa projekti siirrettiin ajettavaksi paikalliseen palvelimeen MAMP-nimiseen ohjelmaan ja samalla kooditiedostot luodaan/vaihdetaan PHP tiedostoiksi. Tässä vaiheessa testattiin ja luotiin vaatimuslistan vaatimukset F15, F16 ja F17 eli pelin datan tallennukset tietokantaan, peliin rekisteröitymis-/kirjautumismahdollisuus ja tulostaulu, joka tulostaa tietokantaan tallennetut ajanotto datan. Samalla korjattiin myös edellisen vaiheen puutteelliseksi jääneet ominaisuudet: ajoneuvon pysähtyminen polttoaineen loppumisen jälkeen, kunkin kierrosajan tallennus Array-lukujonoon, mahdollisuus ajoneuvojen eri osien kustomointiin sekä pieniä muutoksia ajoneuvon ominaisuuksiin säätömuutoksilla ja renkaiden kuluminen.

Kuvassa 15 on arkkitehtuurikuva projektista, jossa asiakasohjelma, eli tässä tapauksessa projektin ohjelmakoodi kommunikoi Internetin kautta palvelimen kanssa. Palvelin puolestaan kommunikoi tietokannan kanssa, kun projekti on tässä vaiheessa siirretty palvelimeen ajettavaksi.



Kuva 15. Arkkitehtuurikuva projektista.

Kuvassa 16 on esimerkkikoodi ajoneuvon ominaisuuksien muutoksista. Ajoneuvon speed eli nopeus määrittyy sen perusteella, kuinka paljon ajoneuvossa on polttoainetta (fuel) ja mikä on downforcen yksikkö (koodissa tunrS). Lisäksi koodissa nähdään, miten polttoaine ja renkaat kuluvat sen mukaan, mikä on ajoneuvon nopeus

ja kääntyvyys, kun ajoneuvo ei ole pysähdyksissä. Kun polttoaineen ja kääntyvyyden arvo on nolla tai alempi, speed ja tunrS arvot muuttuvat nolliksi, eikä ajoneuvo enää liiku tai käänny tämän jälkeen.

```
611         speed = 5 - 0.2*tunrS - 0.02*fuel;
612
613         if (fuel <= 0) {speed = 0;}
614         if (turnL <= 0 || turnR <= 0) {tunrS = 0}
615
616         if(arrowButton[38]){rallyCar.speed = speed;}
617         if(arrowButton[38] && arrowButton[37]){rallyCar.speed = speed; rallyCar.moveAngle = -tunrS;}
618         if(arrowButton[38] && arrowButton[39]){rallyCar.speed = speed; rallyCar.moveAngle = tunrS;}
619         if(arrowButton[40]){rallyCar.speed = -speed;}
620         if(arrowButton[40] && arrowButton[37]){rallyCar.speed = -speed; rallyCar.moveAngle = tunrS;}
621         if(arrowButton[40] && arrowButton[39]){rallyCar.speed = -speed; rallyCar.moveAngle = -tunrS;}
622
623         if(rallyCar.speed != 0) {fuel = fuel - speed*0.01;}
624         if(rallyCar.speed != 0 && rallyCar.moveAngle > 0) {turnR = turnR - speed*0.1;}
625         if(rallyCar.speed != 0 && rallyCar.moveAngle < 0) {turnL = turnL - speed*0.1;}
```

Kuva 16. Koodiesimerkki ajoneuvon muuttuvista ominaisuuksista.

Kuvissa 17 ja 18 nähdään, miten jokainen kierrosaika tallentuu lapTimes-taulukkoon. Kun ajoneuvo ajaa maalilinjan yli, saatu aika parsitaan ensin parseTime-funktiossa millisekunniksi, jonka jälkeen aika tallennetaan lapTimes-taulukkoon.

```
533             parseTime(min, sec, mil);
534             lapTimes.push(parseTime(min, sec, mil));
535             console.log(lapTimes);
```

Kuva 17. Koodiesimerkki ajan tallennuksesta ja parsimisesta.

```
716     function parseTime(min, sec, mil) {
717         var millis = mil;
718         var seconds = sec*1000;
719         var minutes = min*60*1000;
720         var total = millis+seconds+minutes;
721         return total;
722     }
```

Kuva 18. parseTime-funktio.

Ennen toisen vaiheen alkua luotiin paikallisen palvelimen phpMyAdmin hallintatyökaluun tietokanta nimeltä Web_game_data ja sinne taulukot users, customs ja

times, joita tullaan käyttämään koko toisen vaiheen aikana. Kuvissa 19 ja 20 on MySQL-kyselyt, joilla taulukot on luotu.

```
CREATE TABLE IF NOT EXISTS `users` (
        `id` int(11) NOT NULL AUTO_INCREMENT,
        `username` varchar(50) NOT NULL,
        `password` varchar(255) NOT NULL,
        `email` varchar(100) NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

Kuva 19. MySQL-kysely users-taulukon luomiseksi.

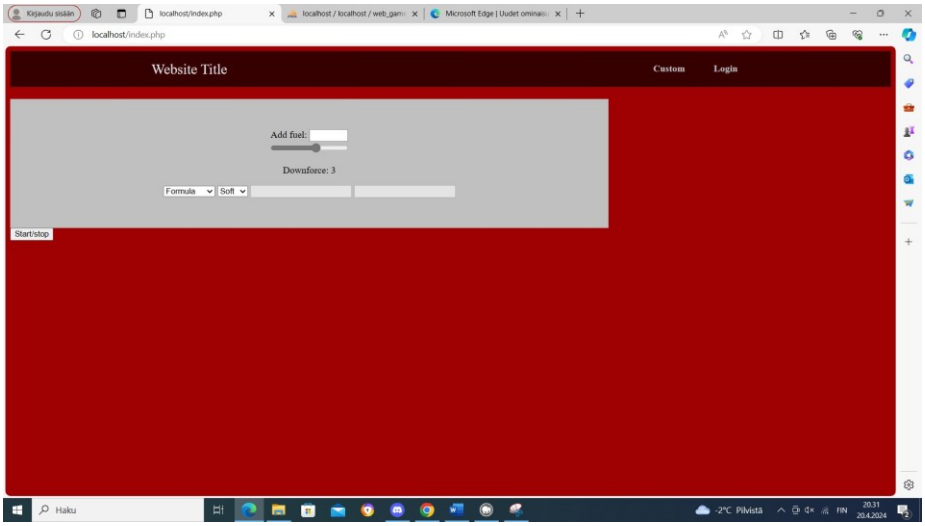
```
CREATE TABLE IF NOT EXISTS `customs` (
        `id` int(11) NOT NULL AUTO_INCREMENT,
    `playerid` int(11) NOT NULL,
        `customname` varchar(50) NOT NULL,
    `color1` varchar(50) NOT NULL,
    `color2` varchar(50) NOT NULL,
    `color3` varchar(50) NOT NULL,
    `color4` varchar(50) NOT NULL,
    `color5` varchar(50) NOT NULL,
    `raceform` varchar(50) NOT NULL,
    PRIMARY KEY (`id`),
    FOREIGN KEY (playerid) REFERENCES users(id)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `times` (
        `id` int(11) NOT NULL AUTO_INCREMENT,
    `playerid` int(11) NOT NULL,
        `time` int(20) NOT NULL,
    `raceform` varchar(50) NOT NULL,
    PRIMARY KEY (`id`),
    FOREIGN KEY (playerid) REFERENCES users(id)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

Kuva 20. MySQL-kysely customs- ja times-taulukoiden luomiseksi.

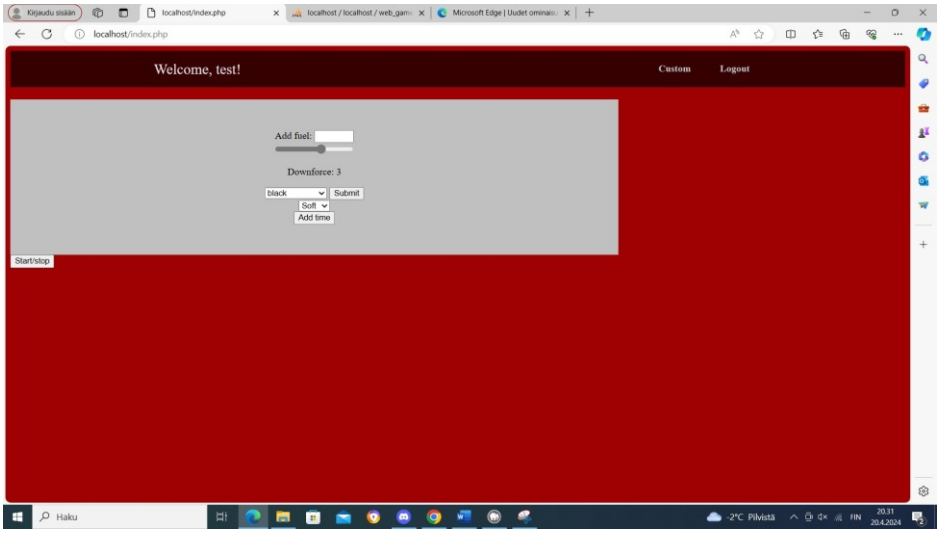
Vaatimus F16 eli kirjautumis-/rekisteröitymisjärjestelmä luotiin käyttäen hyödyksi David Adamsin esimerkkikoodia CodeShack-sivuilta. Koodin avulla voidaan luoda käyttäjätunnus tietokantaan, jota voidaan myöhemmin käyttää kirjautuessa sisään peliin (Adams, 2018a; Adams, 2018b). Kuvista 22 ja 23 nähdään, miltä ohjelma näyttää, kun käyttäjä on joko kirjautunut järjestelmään tai selailee sivustoa ilman kirjautumista. Kuvassa 21 puolestaan on koodiesimerkki, jossa luodaan kirjautumissessio, kun käyttäjä on syöttänyt tarvittavat tiedot kirjautumissivulla.

```
31          if ($stmt->num_rows > 0) {
32              $stmt->bind_result($id, $password);
33              $stmt->fetch();
34              // Account exists, now we verify the password.
35              // Note: remember to use password_hash in your registra
36              if (password_verify($_POST['password'], $password)) {
37                  // Verification success! User has logged-in!
38                  // Create sessions, so we know the user is logged i
39                  session_regenerate_id();
40                  $_SESSION['loggedin'] = TRUE;
41                  $_SESSION['name'] = $_POST['username'];
42                  $_SESSION['id'] = $id;
43                  //echo 'Welcome back, ' . htmlspecialchars($_SESSIO
44                  header('Location: index.php');
45              } else {
46                  // Incorrect password
47                  echo 'Incorrect username and/or password!';
48              }
```

Kuva 21. CodeShackin koodiesimerkki kirjautumissession luonnista.



Kuva 22. Projektin etusivu, kun käyttäjä ei ole kirjautunut sisään.



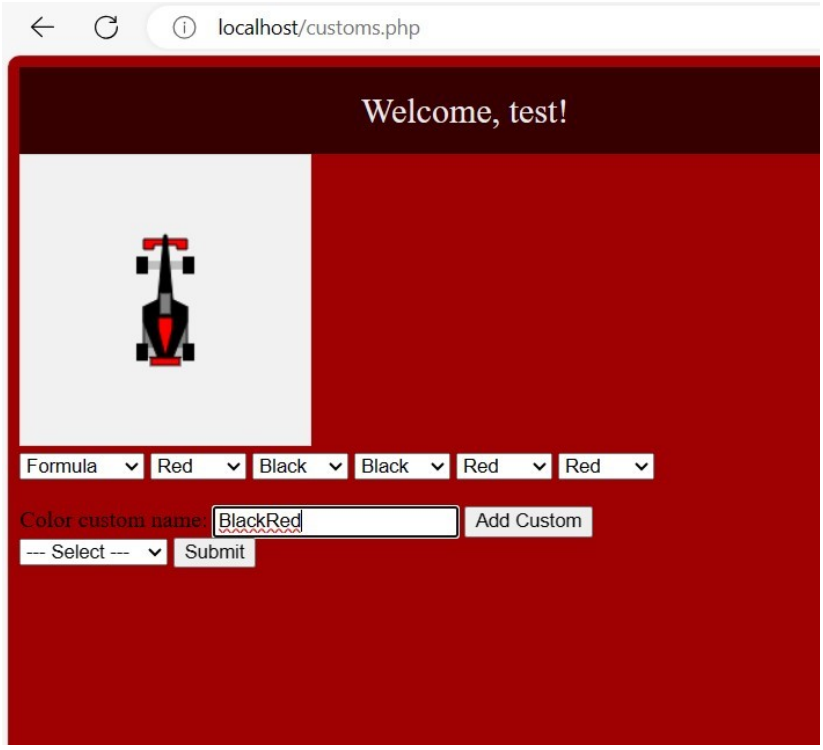
Kuva 23. Projektin etusivu, kun käyttäjä on kirjautunut sisään.

Kuvassa 24 on koodiesimerkki, miten käyttäjä ei pääse Customs-sivulle, ellei hän ole kirjautunut järjestelmään. Tällöin koodi ohjaa käyttäjän kirjautumissivulle.

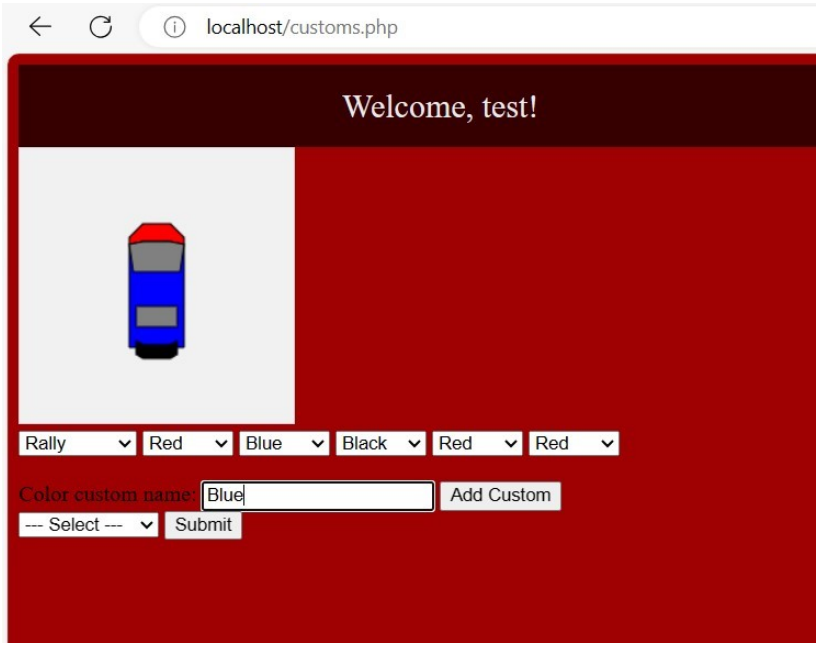
```
1    <?php
2    // We need to use sessions, so you should always start sessions using the below code.
3    session_start();
4    // If the user is not logged in redirect to the login page...
5    if (!isset($_SESSION['loggedin'])) {
6            header('Location: login.php');
7            exit;
8    }
```

Kuva 24. CodeShackin koodiesimerkki, joka ohjaa käyttäjän toiselle sivulle.

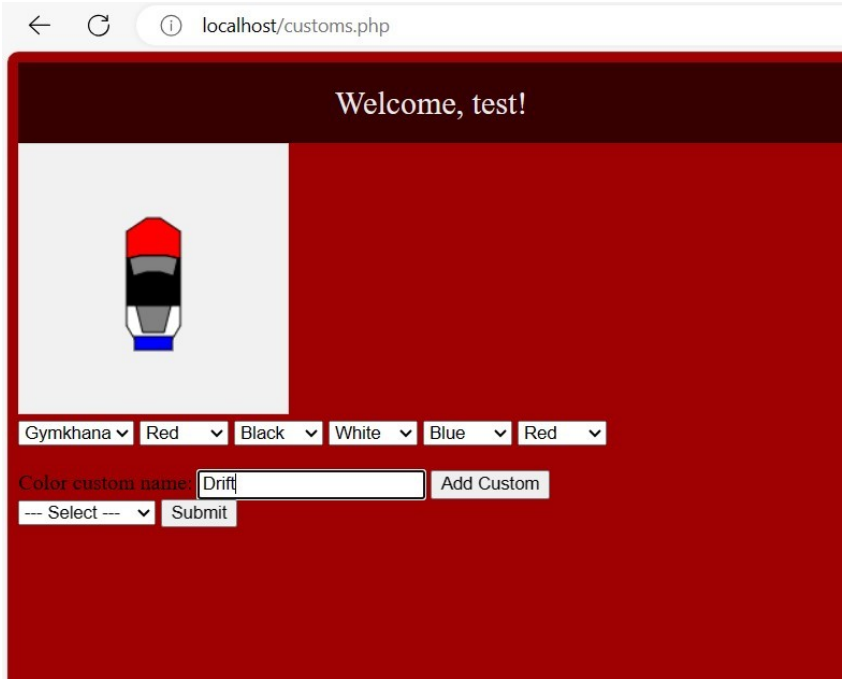
Kohdassa F15 vaadittiin pelissä kertyvän datan (ajoneuvojen värien, kierrosaikojen ja säätöjen) tallentamista tietokantaan, josta niitä voidaan hyödyntää myöhempää käyttöä varten. Tämä saatiin luotua onnistuneesti, vaikka käyttäjän kannalta käyttöliittymä on vielä tässä vaiheessa hieman epäjohdonmukainen ja sisältää pieniä virheitä. Kuvista 25, 26 ja 27 nähdään kolme esimerkkiä Customs-sivun värivalikoista eri ajoneuvojen kesken. Add Custom -painikkeesta tiedot tallennetaan tietokantaan, kun sekä värit että kilpamuoto on valittu ja kustomoinnille on annettu nimi. Alemmasta pudotusvalikosta käyttäjä voi valita aiemmin tallentamansa kustomointitiedon ja tulostaa sen pelikenttään Submit-painiketta painamalla.



Kuva 25. Customs-sivu ja esimerkki formula-auton kustomoinnista.



Kuva 26. Customs-sivu ja esimerkki ralliauton kustomoinnista.



Kuva 27. Customs-sivu ja esimerkki Gymkhana-auton kustomoinnista.

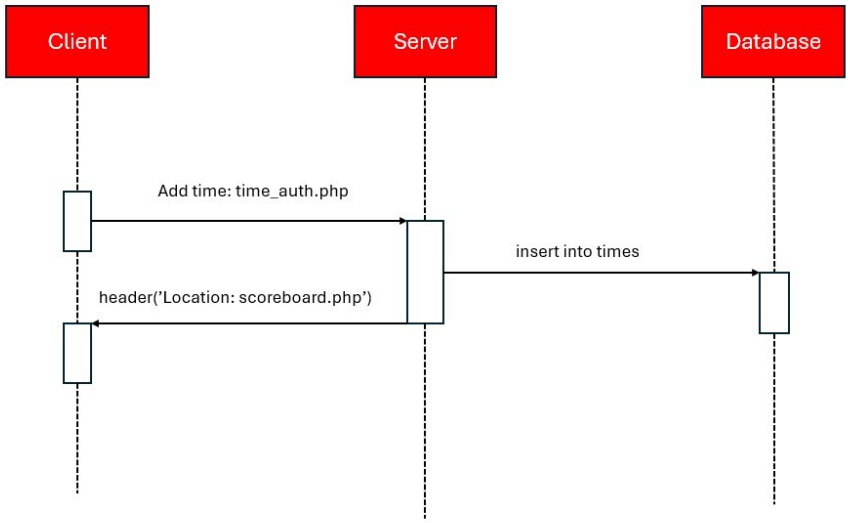
Kuvissa 28 ja 29 nähdään koodissa, miten kustomointitieto tallennetaan tietokantaan customs-taulukkoon ja kierrosaika times-taulukkoon tietokannassa. Kuvassa 30 nähdään UML-sekvenssikaavion muodossa, miten kierrosaika tallennetaan tietokantaan palvelimen kautta, minkä jälkeen palvelin ohjaa käyttäjän Scoreboard-sivulle.

```
29            if($stmt->num_rows > 0){
30                // Customname exists
31                echo 'Customname exists, please choose another!';
32            } else {
33                // Customname doesn't exists, insert new custom
34                if ($stmt = $con->prepare('INSERT INTO customs (playerid, customname, color1, color2, color3, color4, color5,
35                raceform) VALUES (?, ?, ?, ?, ?, ?, ?, ?)')) {
36                    $stmt->bind_param('issssss', $_SESSION['id'], $_POST['cname'], $_POST['carColor'], $_POST['carColor2'], $_POST['carColor3'],
37                    $_POST['carColor4'], $_POST['carColor5'], $_POST['carForm']);
38                    $stmt->execute();
39
40                    $color1 =  $_POST['carColor'];
41                    $color2 =  $_POST['carColor2'];
42                    $color3 =  $_POST['carColor3'];
43                    $color4 =  $_POST['carColor4'];
44                    $color5 =  $_POST['carColor5'];
45                    $id = $_SESSION['id'];
46                    $form = $_POST['carForm'];
47
48                    //echo 'Please check your email to activate your account!';
49                    header('Location: index.php');
```

Kuva 28. Koodiesimerkki kustomointitiedon tallentamisesta tietokantaan.

```
if ($stmt = $con->prepare('INSERT INTO times (playerid, time, raceform) VALUES (?, ?, ?)')) {
    $stmt->bind_param('iis', $_SESSION['id'], $_POST['bestSessionTime'], $_POST['sessionForm']);
    $stmt->execute();
```

Kuva 29. Koodiesimerkki parhaan kierrosajan tallentamisesta tietokantaan.

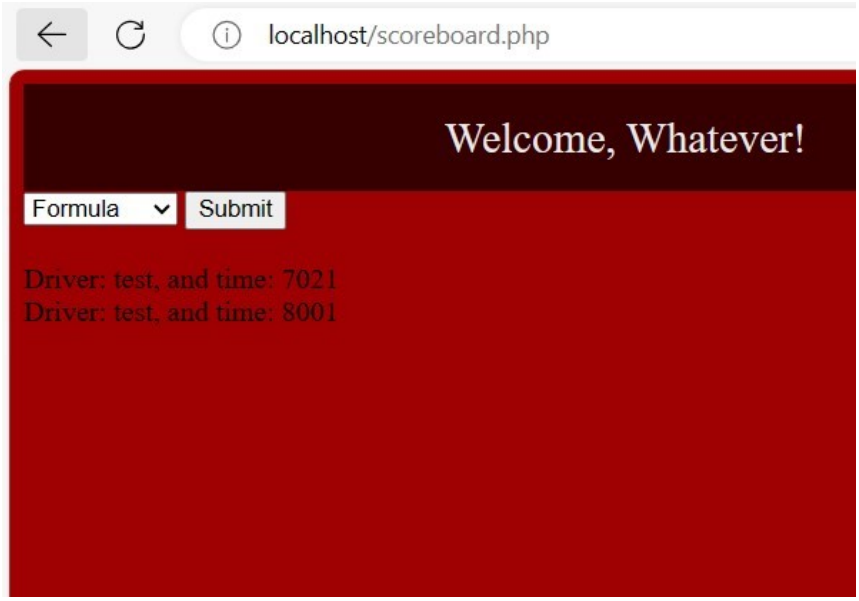


Kuva 30. UML-sekvenssikaavio kierrosajan tallentamisesta asiakas-palvelin-arkkitehtuurissa.

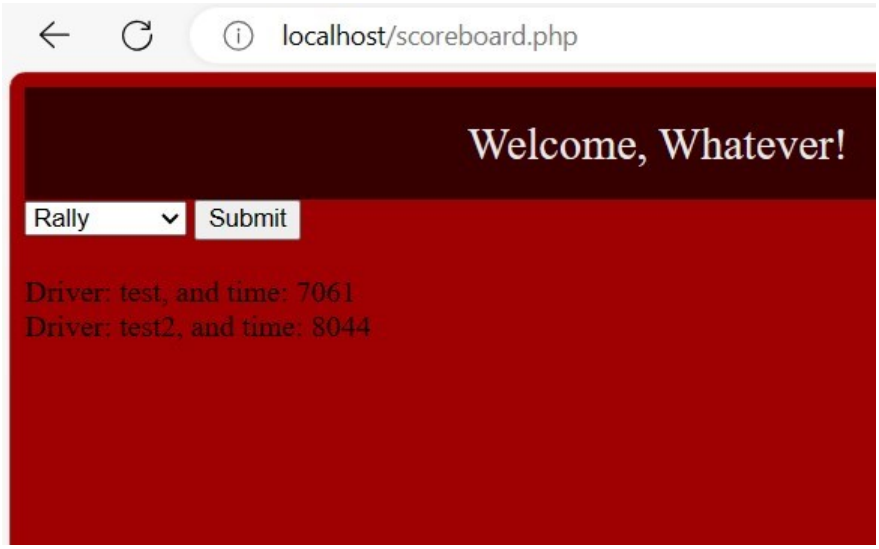
Vaatimuksessa F17 vaadittiin tulostaulun luontia, joka tulostaa kaikki tietokantaan tallennetut ajat Scoreboard-nimiseen sivustoon. Tämäkin saatiin onnistumaan, vaikka varsin alkeellisesti vielä tässä vaiheessa. Kuvissa 32 ja 33 nähdään esimerkki Scoreboard-sivulta toiminnasta. Kuva 31 puolestaan näyttää koodin, jossa kierrosaika ja sen ajanut käyttäjänimi haetaan users- ja times-taulukoista sen jälkeen, kun haluttu kilpamuoto on valittu pudotusvalikosta.

```
80              $sql2 = "SELECT username, time FROM users, times WHERE users.id = times.playerid and raceform = '".$raceForm."' order by time ASC";
81              echo "<script>";
82              echo "console.log('something: " . $sql2. "' );";
83              echo "</script>";
84              //console.log($sql2);
85              $result2 = mysqli_query($con, $sql2);
86              if($result2->num_rows > 0){
87                  while($row2 = mysqli_fetch_assoc($result2)){
88                  echo "Driver: " . $row2['username'] . ", and time: " . $row2['time'] . "<br>";
```

Kuva 31. Koodiesimerkki times- ja users-tietojen hakemisesta tietokannasta.



Kuva 32. Scoreboard-sivu ja Formula-kilpamuodon kierrosajat.



Kuva 33. Scoreboard-sivu ja Rally-kilpamuodon kierrosajat.

Toisessa vaiheessa onnistuttiin luomaan kaikki vaatimuslistan toisen vaiheen prioriteetti 1 vaatimukset (F15, F16, F17). Samalla saatiin korjattua useat ensimmäisen vaiheet puutteelliset/virheelliset ominaisuudet. Ohjelma toimii yleisesti halutulla

tavalla, mutta paikoin varsin alkeellisesti ja epäjohdonmukaisesti. Lisäksi hitObject-funktion toimintaa fence1-objektin kohdalla ei ole vielä saatu korjattua.

Edellä mainitut puutteet/virheet sekä mahdollinen säätöasetuksien tallentaminen tietokantaan ja muut vaatimuslistan alemman prioriteetin vaatimukset tullaan korjaamaan/lisäämään viimeisessä, kolmannessa vaiheessa.

4.3 Kolmas vaihe

Kolmannessa ja viimeisessä vaiheessa hienosäädettiin ja luotiin pienempiä ominaisuuksia ja korjattiin aiemmassa osassa keskeneräiseksi jääneet virheet/puutteet. Tilanteessa, jossa mikä tahansa ajoneuvo osuu sellaiseen objektiin (esimerkiksi aiemmin mainittuun fence1:een), jonka olisi tarkoitus keskeyttää peli ja pysäyttää ajoneuvo, saatiin tässä vaiheessa toimimaan oikein. Kuvissa 34, 35 ja 36 nähdään kyseisen funktion toiminta, kun ajoneuvo osuu haluttuun objektiin, funktio muuttaa hit-arvon ykköseksi, joka siten aktivoi CancelAnimationFrame() metodin, joka keskeyttää koko pelin ja sitä kautta pysäyttää myös ajoneuvon. Kun peli aloitetaan uudelleen Start/stop-painikkeesta, startFunction()-funktio palauttaa ajoneuvon takaisin oletuspaikalle eli ajoneuvon x- ja y-arvot sataan ja muuttaa hit-arvon takaisin nollaksi.

```
} else if (hitObject(rallyCar, fence1)) {
  clearInterval(interval);
  hit = 1;
  console.log("Hit");
}
```

Kuva 34. hitObject-funktio, joka muuttaa hit-arvon ykköseksi.

```
game = window.requestAnimationFrame(draw);
if(hit == 1){
  cancelAnimationFrame(game);
}
```

Kuva 35. cancelAnimationFrame()-funktio, joka aktivoituu, kun hit-arvo on 1.

```
if(hit == 1){hit = 0; formulaCar.x = 100; formulaCar.y = 100;
  rallyCar.x = 100; rallyCar.y = 100; gymkhanaCar.x = 100; gymkhanaCar.y = 100;
}
```

Kuva 36. startFunction()-funktion toiminta, joka palauttaa ajoneuvon oletuspaikalle.

Myös ajoneuvojen realistiset fysiikkaominaisuudet saatiin suurelta osin toimimaan onnistuneesti vaatimuslistan kohdan F21 mukaisesti. Ajoneuvon kiihtyvyys ja jarrutus ominaisuudet saatiin toimimaan edellä mainittujen fysiikan kaavojen vaatimalla tavalla. Kuvissa 37 ja 38 nähdään esimerkki ominaisuuksien toiminnasta. Nuolinäppäimiä painamalla moottorin teho ja jarrut aktivoituvat tiettyihin arvoihin muuttaen kokonaistehon nopeudeksi, joko kiihdyttäen tai jarruttaen.

```
if(arrowButton[38]){Tforce = 3770; console.log(Tforce)}
if(arrowButton[37]){formulaCar.moveAngle = -Fturn;}
if(arrowButton[39]){formulaCar.moveAngle = Fturn;}
if(arrowButton[40]){Fbrake = 2000; ++turboC;
  if(speed <= 0){
    Fbrake = 0;
    velo = 0;
  }
  if(turboC >= 300){
    turboC = 300;
  }
}
```

Kuva 37. Toiminnot, jotka aktivoivat ajoneuvon moottori- ja jarrutehon.

```
586         Cdrag = 0.5 * 0.30 * 1.29 * Farea;
587         Croll = 30 * Cdrag;
588
589         Tdrag = Cdrag * velo * velo;
590         Troll = Croll * velo;
591         Flong = Tforce + turbo - Fbrake - Tdrag - Troll;
592         console.log(Flong);
593         console.log(velo);
594         console.log(Cdrag);
595         //console.log(speed);
596         mass = 750+fuel;
597         //console.log(mass);
598         acce = Flong/mass;
599         //acce*0.0167
600         velo = velo + 0.06*acce;
```

Kuva 38. Ajoneuvon toiminta fysiikan kaavoja mukaillen.

Sen sijaan ohjaukseen liittyvät fysikaaliset kaavat jäivät toteuttamatta, koska Spacejackin käyttämä esimerkki ei toiminut testattaessa halutulla tavalla. Sen sijaan ajoneuvojen ohjauksessa käytettiin yksinkertaistettua esimerkkiä, jossa ohjausteho heikkenee sitä mukaan, mitä nopeammin ajoneuvo kulkee. Kuvassa 39 näkyy esimerkki, jossa Fturn-arvo pienenee nopeuden kasvaessa. Esimerkiksi jos tunrS-yksikkö on 3 ja nopeus yksikkö velo on 30, lopullinen ohjausteho Fturn on 2,1 eli noin kolmanneksen heikompi kuin normaalisti.

```
Fturn = tunrS - (tunrS*0.01)*velo;
```

Kuva 39. Ohjausteho Fturn, joka heikkenee nopeuden kasvaessa.

Vaatimuslistan kohdassa F8 haluttiin saada toimimaan niin sanotut sekundääriset ominaisuudet, joita ovat muun muassa turbo ja käsijarru. Turbo saatiin toimimaan onnistuneesti, kuten kuvissa 40 ja 41 voidaan nähdä toimintakoodissa. Kun näppäimistöstä painetaan T-näppäintä, se tuo turboarvon verran lisää tehoa kokonaistehoon. Turboa ei kuitenkaan voi käyttää, jos ajoneuvon nopeus on nolla, eli turboa ei voi yksistään käyttää moottoritehona, vaan myös kaasupoljinta täytyy käyttää. Kuitenkin samalla, kun turboa käytetään, turboC-arvo kuluu, eikä turboa voi käyttää enää, kun arvo turboC on nolla. Kyseinen arvo kuitenkin latautuu uudelleen maksimiinsa eli 300:n, kun pelaaja painaa jarrunäppäintä.

```
if(arrowButton[40]){Fbrake = 2000; ++turboC;
  if(speed <= 0){
    Fbrake = 0;
    velo = 0;
  }
  if(turboC >= 300){
    turboC = 300;
  }
}
if(arrowButton[84]){turbo = 2000; --turboC;
  if(turboC <= 0 || speed <= 0){
    turbo = 0;
    turboC = 0;
  }
}
```

Kuva 40. Turbo-näppäimen (84) ja jarrunäppäimen (40) toiminta koodissa.

```
Flong = Tforce + turbo - Fbrake - Tdrag - Troll;
```

Kuva 41. Turboarvo koodissa, joka lisää Flong-arvon tehoa 2000:lla turboa käyttäessä.

Käsijarrukin saatiin toimimaan, kuten kuvassa 42 voidaan nähdä. Ajoneuvo hidastuu tätä käytettäessä fysikaalisesti realistisella tavalla, mutta käsijarrukäännöksen toteutus jää kuitenkin realistisesti toteuttamatta johtuen siitä, ettei kääntymiseen liittyviä realistisia ominaisuuksia saatu aiemmin toteutettua. Tässäkin tapauksessa käytettiin yksinkertaista keinoa, jossa Fturn-arvo nousee, kun käsijarrua käytetään.

```
if(arrowButton[32]){eBrake = 5500; Fturn = 5.5;
  if(speed <= 0){
    eBrake = 0;
    velo = 0;
  }
}
```

Kuva 42. Käsijarrun toiminta koodissa, joka aktivoi eBrake-arvon ja nostaa Fturn arvoa korkeammalle.

Vaatimuslistan kohdan F9 haluttiin saada dynaaminen kartta eli kartta, joka liikkuu sitä mukaa, missä ajoneuvolla ajetaan tietyllä hetkellä. Hyödyntäen MDN Web

Docs-sivuston esimerkkiä tämä ominaisuus saatiin onnistuneesti luotua, kuten kuvissa 43, 44 ja 45 voidaan nähdä. Kun ajoneuvo liikkuu esimerkiksi yli 500 pikseliä x-sijaintiin, funktio muuttaa mapX-arvon negatiiviseksi carVeloX-arvoksi (nopeus pelinäkymän vaakasuunnassa) siirtäen näin karttaa translate()-funktion avulla vastakkaiseen suuntaan pitäen näin ajoneuvon näytöllä näkyvissä. Tämän funktion mapX- ja mapY-arvot vastaavat suurin piirtein MDN Web Docs-sivulla (2024) mainitun camera-objektin x- ja y-parametreja, jotka määrittelevät camera-objektin sijainnin vasemman yläkulman kautta, tässä tapauksessa ajoneuvon sijainnin mukaan.

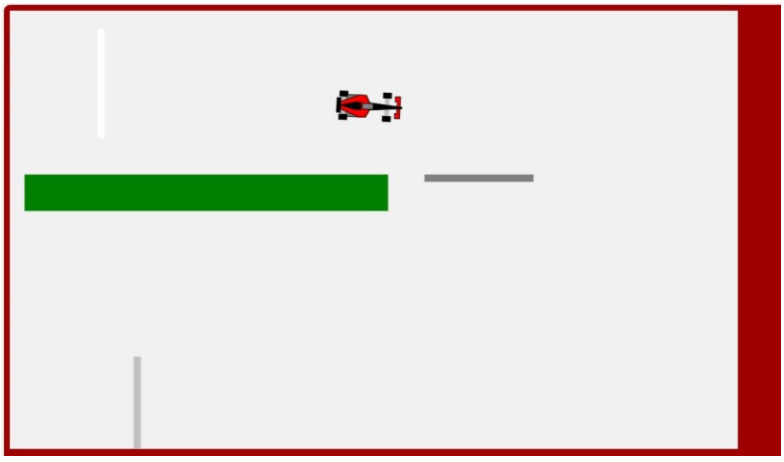
Vaikka funktio toimii yleisesti varsin hyvin, paranneltavaa vielä jäi: funktion cameraX- ja cameraY-arvot ja sitä kautta mapX- ja mapY-arvot eivät pysähdy, vaikka ajoneuvo saavuttaa pelikentän minimin ja maksimin sekä pysty- että vaakasuorassa (tässä tapauksessa 1000 ja 600 pikseliä), kuten kuvassa 46 voidaan nähdä. Lisäksi kun ajoneuvo törmää pelin keskeyttävään objektiin, pelinäkymä ei nollaudu alkuperäiseen paikkaan, mikä aiheuttaa sen, ettei näyttö seuraa ajoneuvoa oikeassa kohtaa ja saattaa jopa kadota pelaajan silmistä, kuten kuvassa 47 voidaan nähdä. Tässä vaiheessa tämän pystyy korjaamaan vain lataamalla koko sivun uudelleen.

```
495     function draw() {
496       ctx.clearRect(0, 0, canvas.width*10, canvas.height*10);
497       ctx.translate(mapX, mapY);
498
499       checkPoint.update();
500       checkPoint2.update();
501       finishLine.update();
502       fence1.update();
503       mapX = 0;
504       mapY = 0;
```

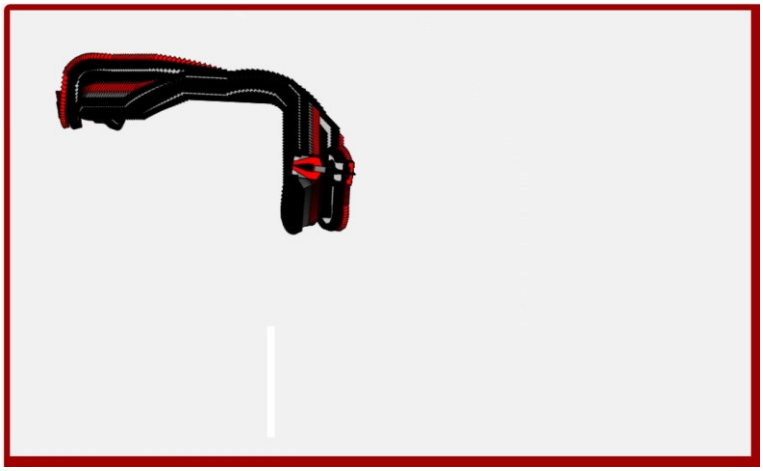
Kuva 43. mapX- ja mapY-arvot, jotka siirtävät pelikenttää translate()-funktion avulla.

```
600         carVeloX = (velo/3) * Math.sin(formulaCar.angle);
601         carVeloY = -(velo/3) * Math.cos(formulaCar.angle);
602         console.log(carVeloX);
603         console.log(carVeloY);
604         cameraX += carVeloX;
605         cameraY += carVeloY;
606         //cameraX2 += carVeloX;
607         //cameraY2 += carVeloY;
608         console.log("camera x: " + cameraX2);
609         console.log("camera y: " + cameraY2);
610
611         if(cameraX >= 500 && carVeloX > 0){cameraX = 500; mapX = -carVeloX; cameraX2 += -carVeloX;}
612             if(cameraX <= 200 && carVeloX < 0){cameraX = 200; mapX = -carVeloX; cameraX2 += -carVeloX;}
613             if(cameraY >= 500 && carVeloY > 0){cameraY = 500; mapY = -carVeloY; cameraY2 += -carVeloY;}
614             if(cameraY <= 200 && carVeloY < 0){cameraY = 200; mapY = -carVeloY; cameraY2 += -carVeloY;}
```

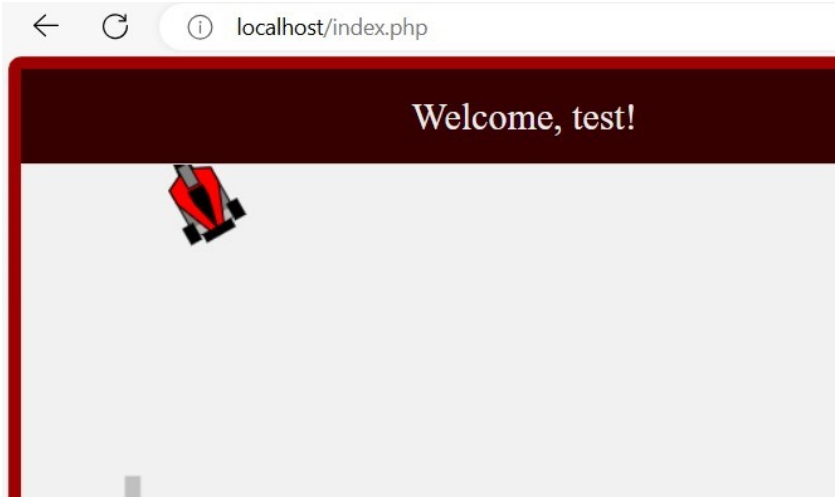
Kuva 44. Dynaamisen kartan toiminta koodissa.



Kuva 45. Pelikenttä lyhyen ajamisen jälkeen, kun pelikenttä on liikkunut hieman.

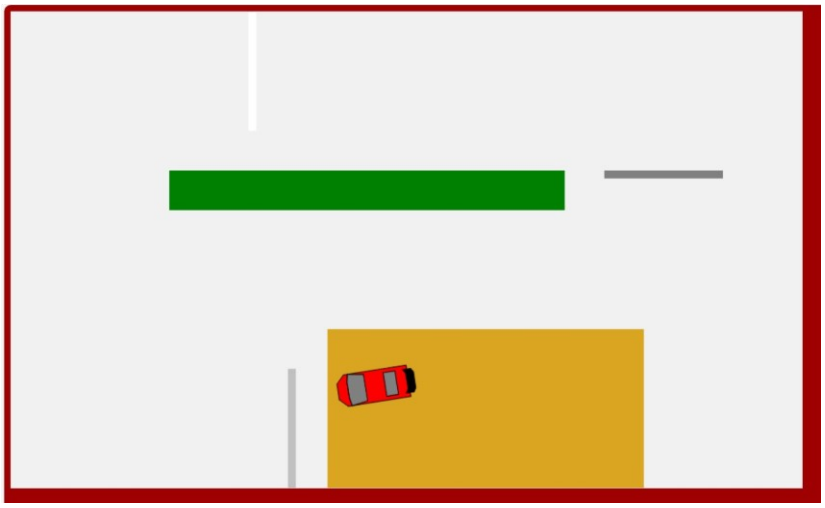


Kuva 46. Pelinäkymä, kun ajoneuvo on pelirajojen ulkopuolella.



Kuva 47. Pelinäkymä, kun ajoneuvo on osittain pelikentän ulkopuolella karttafunktion virheen takia.

Viimeisenä tässä vaiheessa luotiin vielä vaatimuslistan kohdan F22 vaatimus, jonka myötä ajoneuvon ohjattavuus muuttuu, kun ajoneuvo ajaa tietyn tiepinnan, tässä tapauksessa gravel-objektin eli soraosuuden läpi, minkä myötä ohjausteho heikkenee. Tämäkin ominaisuus saatiin luotua, mutta ei kuitenkaan fysikaalisesti realistisella tavalla edellä mainittujen puutteiden vuoksi. Kuvista 48 ja 49 kuitenkin nähdään, kun ajoneuvo ajaa soraosuuden läpi, hitGravel-arvo muuttuu ykköseksi, mikä muuttaa ajoneuvon ohjaustehon Fturn-arvon tavallista heikommaksi, mitä nopeammin ajoneuvo liikkuu. Puutteena voidaan mainita myös virhe, joka tapahtuu silloin, kun ajoneuvo ajaa liian lujaa: kuvan 49 koodista näkyy, että Fturn muuttuu negatiiviseksi, mikä tarkoittaa sitä, että ajoneuvo kääntyy vastakkaiseen suuntaan kuin mihin ajoneuvoa yritetään kääntää. Esimerkiksi kun nopeus velo nousee 30:een ja tunrS arvo on 4, Fturn-arvo muuttuu -2:ksi, minkä vuoksi ajoneuvo kääntyy vasemmalle, vaikka näppäimistössä painetaan oikeaa nuolinäppäintä.



Kuva 48. gravel-objekti pelikentällä.

```
669         if(hitObject(rallyCar, gravel)){gravelHit = 1;/*tunrS = tunrS*0.6;*/}
670         //if(gravelHit == 1){tunrS = tunrS*0.6;}
671
672         //speed = velo - 0.2*tunrS - 0.02*fuel;
673         Fturn = tunrS - (tunrS*0.01)*velo;
674         if(gravelHit == 1){Fturn = tunrS - (tunrS*0.05)*velo;}
```

Kuva 49. hitObject()-funktio, kun ajoneuvo ajaa gravel-objektin läpi.

Ajan puutteen vuoksi työssä jätettiin toteuttamatta muutamia 2- ja 3-prioriteettien vaatimuksia, esimerkiksi moninpelimuoto ja säätöjen tallennus. Tästä huolimatta kolmannessa ja viimeisessä vaiheessa saatiin luotua useita vaatimuslistan vaatimia ominaisuuksia, joista kaikki toimivat enemmän tai vähemmän halutulla tavalla. Joitakin virheistä pelissä on, mutta lähes kaikki virheet/puutteet on mahdollista korjata varsin vaivatta. Yleisesti koko peli toimii kuitenkin halutulla tavalla.

5 TYÖN KOKONAISTULOKSEN ARVIOINTI

Jotta voidaan saada selville, onko pelin kehitys onnistunut ja onko peli täyttänyt halutut vaatimukset ja kriteerit, pitää olla tiedossa, miten peliä ja sen ominaisuuksia voidaan mitata ja vertailla. Edellä mainittuihin tutkimusongelmiin esitetään vastaukset alla olevissa luvuissa, jolloin voidaan vastata, onko pelin kokonaistulos onnistunut.

5.1 Pelin laadun ja onnistumisen mittaus

Pelin onnistuneisuus voidaan ELVTR-sivuston mukaan mitata niin sanottujen pelimittareiden (game metrics) avulla. Niiden avulla voidaan huomata alueet, joita pitää vielä parantaa. Pelimittareiden seuraaminen on tärkeää pelikehittäjille, koska sen avulla voidaan selvittää, mitkä tekijät pelissä vetävät pelaajia takaisin pelin pariin. Pelimittareita on useita, esimerkiksi pelaajamäärä (player count), joka määrittää aktiivisesti peliä pelaavien määrän auttaen vertailemaan pelin suosiota ja mahdollista potentiaalia. Toinen mittari on keskimääräinen session pituus (average session length) eli kuinka pitkään pelit keskimäärin kestävät. Mitä pidempi aika on, sitä sitoutuneempia pelaajat ovat pelin pelaamisessa. Lisäksi myös päivittäiset ja kuukausittaiset aktiiviset pelaajat-mittari (daily active users DAU ja month active users MAU) ilmoittaa pelaajat, jotka ovat sitoutuneet pelaamaan peliä päivittäin tai kuukausittain. Tämä mittari kertoo pelin yleisestä tavoittavuudesta ja suosiosta. Muita myös merkittäviä mittareita ovat muun muassa säilymistaso (retention rate), tulot käyttäjää kohti (revenue per user ARPU), kaatumisraportit (crash reports) ja sosiaaliset mittarit (social metrics). (ELVTR, 2024.)

5.2 Pelin realistisuus

Videopeleissä realismi voidaan jakaa useaan eri muotoon, kuten visuaalinen realismi, pelattavuusrealismi, maailman suunnittelurealismi ja tarina/hahmorealismi. Britannican (2024) mukaan määritelmä realismille taiteessa on ”Tarkka, yksityiskohtainen, koristelematon kuvaus luonnosta tai nykyaikaisesta elämästä”. Näiden

perusteella voidaan päätellä, että pelin realistisuus määräytyy pelin useiden eri osa-alueiden realistisuudella, eli kuinka tarkasti/yksityiskohtaisesti eri alueet on luotu, kuinka lähellä todenmukaisuutta asiat pelissä on. Esimerkiksi ajopelissä ovatko visuaalisesti animaatiot, mallit, fysiikat ja tekstuurit lähellä todellisuutta. (Fazeli, 2022.)

Kun puhutaan ajopeleistä, etenkin ajosimulaatiopeleistä, merkittävin osa-alue pelin realistisuudelle on ajofysiikat, joilla voidaan parantaa pelaajien pelikokemusta. Realistisilla fysiikoilla voidaan simuloida ajoneuvojen kiihtyvyyttä, jarrutusta, kääntymistä ja muita ajo-ominaisuuksia, kuten renkaiden kulumista ja painonjakaumaa. Myös ajoneuvon säätämisen mahdollisuus, esimerkiksi jousituksen, pitovoiman ja jarrujen, lisää pelin realistisuutta. (Juego Studio, 2023; Mazuru, 2024.)

Tarkastellaan seuraavaksi ajoneuvojen kahta eri ominaisuutta, huippunopeutta ja kiihtyvyyttä. Kuten luvussa 2.2 mainittiin, ajoneuvo saavuttaa maksiminopeuden silloin kun ajoneuvon vierintä- ja ilmanvastus yhdessä kasvavat isommaksi kuin ajoneuvon pitovoima eli moottoriteho. Kun ajoneuvolle on määritelty teho ja pinta-ala kohtisuoraan katsottuna ajoneuvosta, voidaan laskea ajoneuvolle teoreettinen huippunopeus, kohtisuorainen pinta-ala A menee ilmanvastuksen vakioon C_{drag} kaavaan:

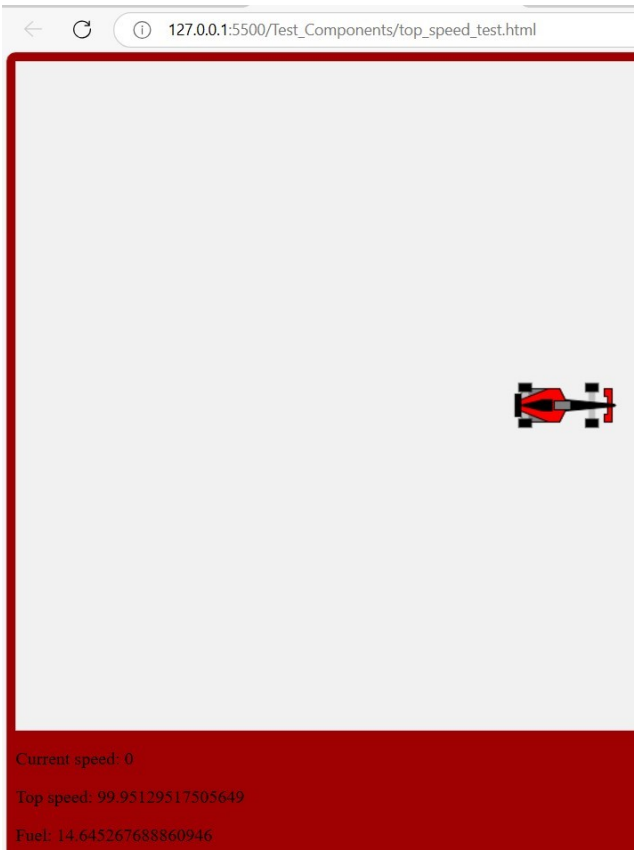
$$C_{drag} = 0{,}5 * 1{,}29 * A * 0{,}30 \quad (13)$$

missä 1,29 on ilmantiheys ja 0,30 on tässä tapauksessa summittaisesti määritelty ajoneuvon kitkakerroin C_d. Vierintävastuksen vakioarvo C_{rr} puolestaan saadaan kertomalla ilmanvastuksen vakioarvo kolmellakymmenellä. (Monster, 2024.)

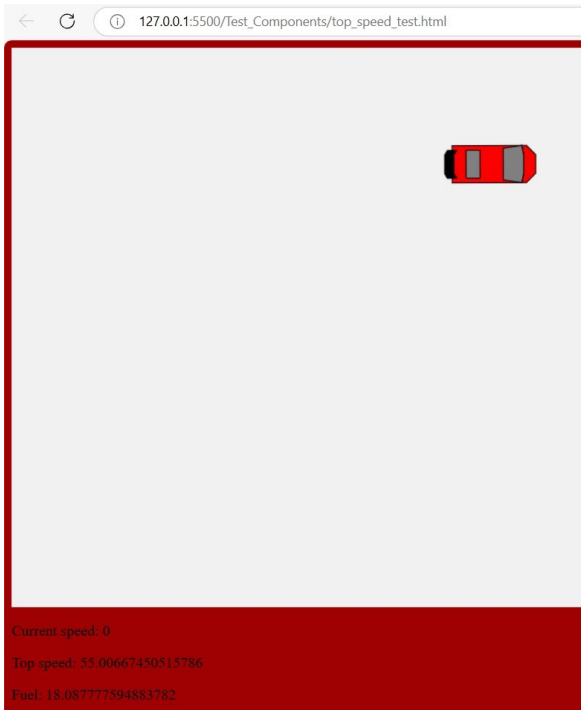
Näillä tiedoilla voidaan laskea ajoneuvon oletettu huippunopeus. Peliä testatessa ajoneuvojen huippunopeudet osoittautuivat suurin piirtein samoiksi kuin ennalta lasketut huippunopeudet, kuten taulukko 2 osoittaa. Kuvissa 50, 51 ja 52 nähdään ajoneuvojen saavuttamat huippunopeudet pelissä.

Taulukko 2. Ajoneuvojen oletetut ja testatut huippunopeudet.

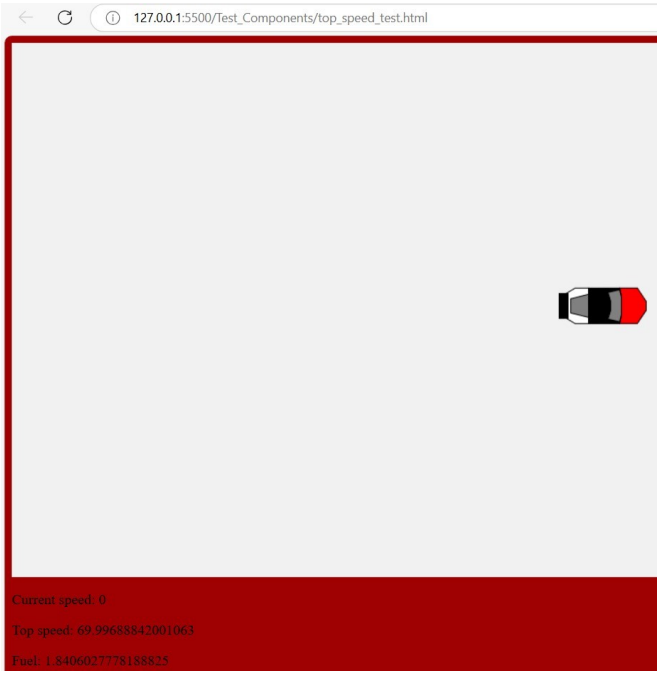
Ajoneuvo tyyppi	Ajoneuvon teho	Ajoneuvon pinta-ala	Ajoneuvon oletettu huippunopeus (m/s)	Ajoneuvon saatu huippunopeus (m/s)
Formula	3770	1,5	99,95	99,951
Rally	2262	2,5	55	55,006
Gymkhana	3386	2,5	69,99	69,996



Kuva 50. Formula-auton saavutettu huippunopeus lyhyen ajon jälkeen.



Kuva 51. Ralliauton saavutettu huippunopeus lyhyen ajon jälkeen.



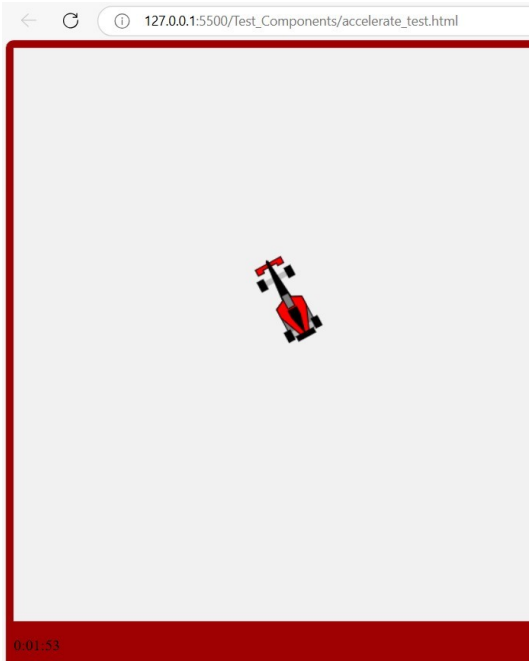
Kuva 52. Gymkhana-auton saavutettu huippunopeus lyhyen ajon jälkeen.

Kiihtyvyys saadaan selville, kun tiedetään ajoneuvon perustiedot eli teho, kohtisuora pinta-ala, massa (mukaan lukien ajoneuvon polttoaineen määrä) ja ajanlisäys Δt sekä pelin fps (Frames Per Seconds) eli kuinka monta kuvaa pelissä on sekunnin ajan. Tässä työssä käytetään pelin pyörittämiseen requestAnimationFrame()-funktiota, jonka fps on normaalisti 60, eli yhteen sekuntiin mahtuu 60 kuvaa. Kunkin ”freimin” aikana ajoneuvon tehosta vähennetään ilman- ja vierintävastuksen voimat ja tämä tulos (nettovoima) jaetaan ajoneuvon massalla, joka lopulta kerrotaan Δt -yksiköllä. Formula-auton kohdalla saadaan tulokseksi ensimmäisen ”freimissä” kiihtyvyydeksi 0,297632-yksikköä. Kiihtyvyys tulee hidastumaan sitä mukaa, mitä nopeammin ajoneuvo liikkuu. Toisaalta polttoaineen väheneminen keventää ajoneuvoa, jolloin ajoneuvon kiihtyvyys hieman paranee. Tämä tarkoittaa, että ajoneuvojen lasketut/oletetut kiihtyvyydet saattavat erota pelissä saaduista kiihtyvyyksistä jonkin verran.

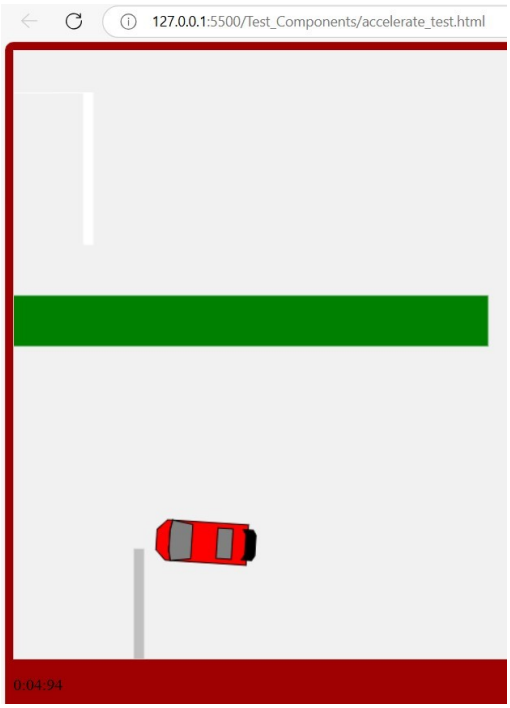
Tässä testissä halutaan ajoneuvon kiihtyvän nollasta 28 m/s (metriä sekunnissa). Kun luku 28 jaetaan ensimmäisen ”freimin” kiihtyvyydellä, eli formula-auton kohdalla 0,297632 saadaan tulokseksi noin 94,08, eli päästäkseen nopeuteen 28 tarvitaan 95 ”freimiä”, mikä tarkoittaa sekunneissa noin 1,58. Formula-auton kiihtyvyys nollasta 28:een olisi siten noin 1,5 sekuntia. Pelissä testattaessa formula-auton kiihtyvyydeksi saatiin 1,53 sekuntia. Tämän ja muiden ajoneuvon kiihtyvyyksien oletetut ja saadut tulokset voidaan nähdä taulukosta 3. Kuvissa 53, 54 ja 55 nähdään ajoneuvojen kiihtyvyyteen käytetyt ajat pelissä lyhyen testauksen jälkeen. Tässäkin testissä nähdään, että ajoneuvojen kiihtyvyydet pelissä ovat suurin piirtein samoja, kuin etukäteen lasketut oletetut kiihtyvyydet. Suurin ero löytyy ralliautossa, jossa oletetun ja saadun kiihtyvyyden ajat eroavat noin 0,6 sekuntia mikä johtuu polttoaineen vähenemisestä ja vastuksien voimistumisesta.

Taulukko 3. Ajoneuvon oletetut ja testatut kiihtyvyydet.

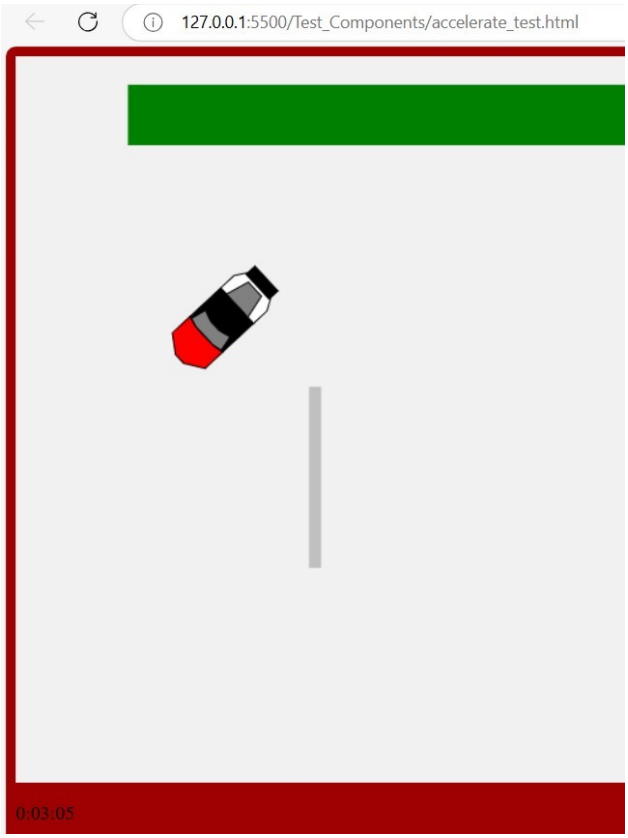
Ajoneuvo tyyppi	Ajo-neu-von teho	Ajo-neu-von pinta-ala	Ajo-neuvon massa (mu-kaan lukien poltto-aine)	Δt	Ajoneu-von kiih-tyvyys en-simmäi-sellä ”freimillä”	Ajo-neu-von oletus kiihty-vyys sekun-neissa (arvi-olta)	Ajo-neu-von saatu kiihty-vyys sekun-neissa
Formula	3770	1,5	760	0,06	0,297632	1,58	1,53
Rally	2262	2,5	1260	0,06	0,107714	4,33	4,94
Gymkhana	3386	2,5	1260	0,06	0,161238	2,9	3,05



Kuva 53. Formula-auton nollasta 28:een käytetty aika pelissä.



Kuva 54. Ralliauton nollasta 28:een käytetty aika pelissä.



Kuva 55. Gymkhana-auton nollasta 28:een käytetty aika pelissä.

Taulukoista 2 ja 3 voidaan nähdä, että kunkin ajoneuvon kiihtyvyys ja huippunopeus pelissä vastaavat suurin piirtein fysikaalisten kaavojen mukaisia tuloksia. Ajoneuvojen ominaisuudet ovat siis varsin realistisia, mikä lisää pelin realistisuutta ainakin pelaajan näkökulmasta, vaikka kunkin ajoneuvon kohdalla fysiikkakaavoja voisi vielä hienosäätää.

5.3 Käytännölliset tavat mittaukseen ja vertailuun

Useiden ohjelmistotalojen ja koulutuskeskusten, kuten ELVTR:n, GEEIQ:n ja RocketBrushin, verkkosivustoilla mainitaan pelimittareiden (game metrics) olevan pelikehittäjille erittäin tärkeitä työkaluja, joiden avulla kehittävät voivat seurata, miten hyvin peli menestyy ja missä osa-alueissa peli tarvitsee kehittämistä. Näiden mittareiden seuraamisella kehittäjät voivat tehdä viisaita ratkaisuja peliä kehitellessään ja sitä kautta parantaa pelaajien pelikokemusta ja sitoutumista jatkaa pelaamista tehden pelistä vieläkin onnistuneemman. Tämän perusteella pelimittareita voidaan pitää käytännöllisimpänä tapana mitata ja vertailla pelin toimintoja ja käyttäjäkokemusta ja sitä kautta pelin kokonaistulosta. (ELVTR, 2024; GEEIQ, 2024; RocketBrush Studios, 2023.)

6 JOHTOPÄÄTÖKSET

Tämän opinnäytetyön tarkoituksena oli tutkia, voiko web-ohjelmointiin soveltuvilla kielillä ja työkaluilla rakentaa kattavan, laaja-alaisen ja realistisen pelin sekä miten hyvin web- ja pelikehitys kulkevat rinnakkain. Tämän työn kokonaistulos todistaa, että web-ohjelmointikielillä ja työkaluilla voidaan rakentaa kattava ja laaja-alainen verkkopeli, jossa voidaan soveltaa realistisia fysiikkakaavoja lisäten näin pelin realistisuutta.

Pelissä pystyy pelaamaan aika-ajomenetelmällä toisiaan vastaan usealla eri ajoneuvoilla samalla kun ajoneuvojen polttoaine ja renkaat kuluvat. Lisäksi ajoneuvojen säätöjä on mahdollista muokata, kuten myös ajoneuvon kustomoinnin (ajoneuvon eri osien värien valinta) itselleen sopiviksi. Pelaajan rekisteröitymis-/kirjautumisominaisuus saatiin myös toimimaan onnistuneesti, minkä ansiosta pelaaja voi tallentaa ajamansa kierrosajan, jonka hän voi myöhemmin nähdä tulostaulusta muiden pelaajien aikoja ja kustomointitiedon lisäksi. Tiedot pelaaja voi myöhemmin ladata seuraavaan pelikertaan. Myös muutamat alemman prioriteetin vaatimukset saatiin toteutettua, kuten turbo, käsijarru ja dynaaminen kartta, joka mukautuu sitä mukaa, missä ajoneuvo liikkuu. Ajoneuvot saatiin myös kiihdytyksen ja huippunopeuden kohdalla toimimaan fysikaalisesti realistisella tavalla.

Osittain ajanpuutteen vuoksi muutamat toiminnot projektissa jäivät toteuttamatta tai jäivät keskeneräiseksi/alkeellisiksi. Esimerkiksi moninpeliominaisuus jäi kokonaan toteuttamatta. Lisäksi ajoneuvojen kääntymisominaisuudet jäivät fysikaalisesti realistisella tavalla toteutumatta. Muita puutteita olivat muun muassa pelin käyttöliittymän hieman epäjohdonmukainen toiminta, dynaamiseen karttaan liittyvät virheet/puutteet ja eri radanpintaan liittyvä ajoneuvojen ominaisuuksien muuttuminen, sen periaate toimi, mutta hyvin alkeellisesti.

Edellä mainitut virheet ja muut puutteelliset tai toteuttamatta jääneet ominaisuudet/toiminnot jäävät pelin erinomaisiksi jatkokehittämiskohteiksi. Kuitenkin kaikki

vaatimuslistan prioriteetin 1 vaatimukset saatiin toteutettua ja yleisesti peli saatiin toimimaan halutulla tavalla pelkästään web-ohjelmointikielillä ja työkaluilla.

LÄHTEET

Adams. (2018a, 15 maaliskuuta). Secure Login System with PHP and MySQL. CodeShack. Noudettu 21.4.2024 osoitteesta https://codeshack.io/secure-login-system-php-mysql/

Adams. (2018b, 20 maaliskuuta). Secure Registration System with PHP and MySQL. CodeShack. Noudettu 21.4.2024 osoitteesta https://codeshack.io/secure-registration-system-php-mysql/

Britannica. (2024, 30 heinäkuuta). Realism | History, Definition, & Characteristics. Noudettu 14.9.2024 osoitteesta https://www.britannica.com/art/realism-art

Chris. (2021, 30 elokuuta). What is PHP? The PHP Programming Language Meaning Explained. FreeCodeCamp. Noudettu 16.4.2024 osoitteesta https://www.freecodecamp.org/news/what-is-php-the-php-programming-language-meaning-explained/

ELVTR. (2024, 18 heinäkuuta) https://elvtr.com/blog/key-metrics-in-game-analytics-measuring-and-optimizing-game-performance

Fazeli, V. (2022, 26 elokuuta). Realism in video games: What we can learn from Red dead redemption 2. Medium. Noudettu 14.9.2024 osoitteesta https://medium.com/@fazeli.vahid/realism-in-video-games-what-we-can-learn-from-red-dead-redemption-2-8a8162083f82

FreeCodeCamp. (2019, 28 joulukuuta). What Is Game Development? Noudettu 9.2.2024 osoitteesta https://www.freecodecamp.org/news/what-is-game-development/

GEEIQ. (2024, 27 helmikuuta). How to measure success in gaming: 10 key gaming metrics you need to know. Noudettu 17.9.2024 osoitteesta https://geeiq.com/how-to-measure-success-in-gaming-10-key-gaming-metrics-you-need-to-know/

GeeksforGeeks. (2023a, 11 joulukuuta). CSS Tutorial | Learn CSS Online for Free. Noudettu 30.1.2024 osoitteesta https://www.geeksforgeeks.org/css-tutorial/

GeeksforGeeks. (2023b, 4 joulukuuta). HTML Tutorial | Learn HTML Online for Free. Noudettu 30.1.2024 osoitteesta https://www.geeksforgeeks.org/html-tutorial/

GeeksforGeeks. (2019, 27 helmikuuta). HTML5 | Introduction. Noudettu 2.2.2024 osoitteesta https://www.geeksforgeeks.org/html5-introduction/

GeeksforGeeks. (2023c, 11 joulukuuta). JavaScript Tutorial - Learn JavaScript Online for Free. Noudettu 30.1.2024 osoitteesta https://www.geeksforgeeks.org/javascript/

GeeksforGeeks. (2023d, 26 lokakuuta). Visual Studio vs Visual Studio Code - What to Choose in 2024? Noudettu 17.4.2024 osoitteesta https://www.geeksforgeeks.org/visual-studio-vs-visual-studio-code/

GeeksforGeeks. (2023e, 5 joulukuuta). Web Development. Noudettu 30.1.2024 osoitteesta https://www.geeksforgeeks.org/web-development/

Git. (2024, 2 helmikuuta). https://git-scm.com/

Juego Studio. (2023, 7 syyskuuta). Racing Game Mechanics: A Detailed Guide For Beginners. Noudettu 14.9.2024 osoitteesta https://www.juegostudio.com/blog/racing-game-mechanics

Kinsta. (2023, 17 marraskuuta). What Is GitHub? A Beginner's Introduction to GitHub. Noudettu 2.2.2024 osoitteesta https://kinsta.com/knowledge-base/what-is-github/

MAMP-Mac. (2024, 16 huhtikuuta). https://www.mamp.info/en/mamp/mac/

MAMP-Windows. (2024, 17 huhtikuuta). https://www.mamp.info/en/windows/

Mazuru, Mircea. (2024, 23 kesäkuuta). The 7 Most Realistic Car Racing Sim Games, Ranked. autoevolution. Noudettu 14.9.2024 osoitteesta https://www.autoevolution.com/news/the-7-most-realistic-car-racing-sim-games-ranked-235787.html

MDN Web Docs. (2024, 23 heinäkuuta). Square tilemaps implementation: Scrolling maps - Game development. Noudettu 14.8.2024 osoitteesta https://developer.mozilla.org/en-US/docs/Games/Techniques/Tilemaps/Square_tilemaps_implementation:_Scrolling_maps

Monster, Marco. (2024, 5 toukokuuta). https://www.asawicki.info/Mirror/Car%20Physics%20for%20Games/Car%20Physics%20for%20Games.html

Oracle. (2024, 16 huhtikuuta). https://www.oracle.com/mysql/what-is-mysql/

Rafath. (2023, 6 helmikuuta). *Difference between web development and game development*. Medium. Noudettu 9.2.2024 osoitteesta https://medium.com/@rafath_87861/difference-between-web-development-and-game-development-73ce8f4b7e50/

RocketBrush Studios. (2023, 8 marraskuuta). 11 Game Metrics All Developers Should Know. Noudettu 17.9.2024 osoitteesta https://rocketbrush.com/blog/11-game-metrics-all-developers-should-know

Spacejack. (2024, 18 kesäkuuta). https://github.com/spacejack/carphysics2d/tree/master/public

Tripathi, P. (2023, 1 tammikuuta). *7 differences you must know to decide between Web Development and Game development*. DEV Community. Noudettu 9.2.2024 osoitteesta https://dev.to/pragyanatvade/7-differences-you-must-know-to-decide-between-web-development-and-game-development-3hcl/