Projektityön dokumentti

1. **Henkilötiedot**

   Tasohyppely Kario Game

   Henri-Matias Tuomaala 609265

   Elektroniikka ja sähkötekniikka 21.4.2019

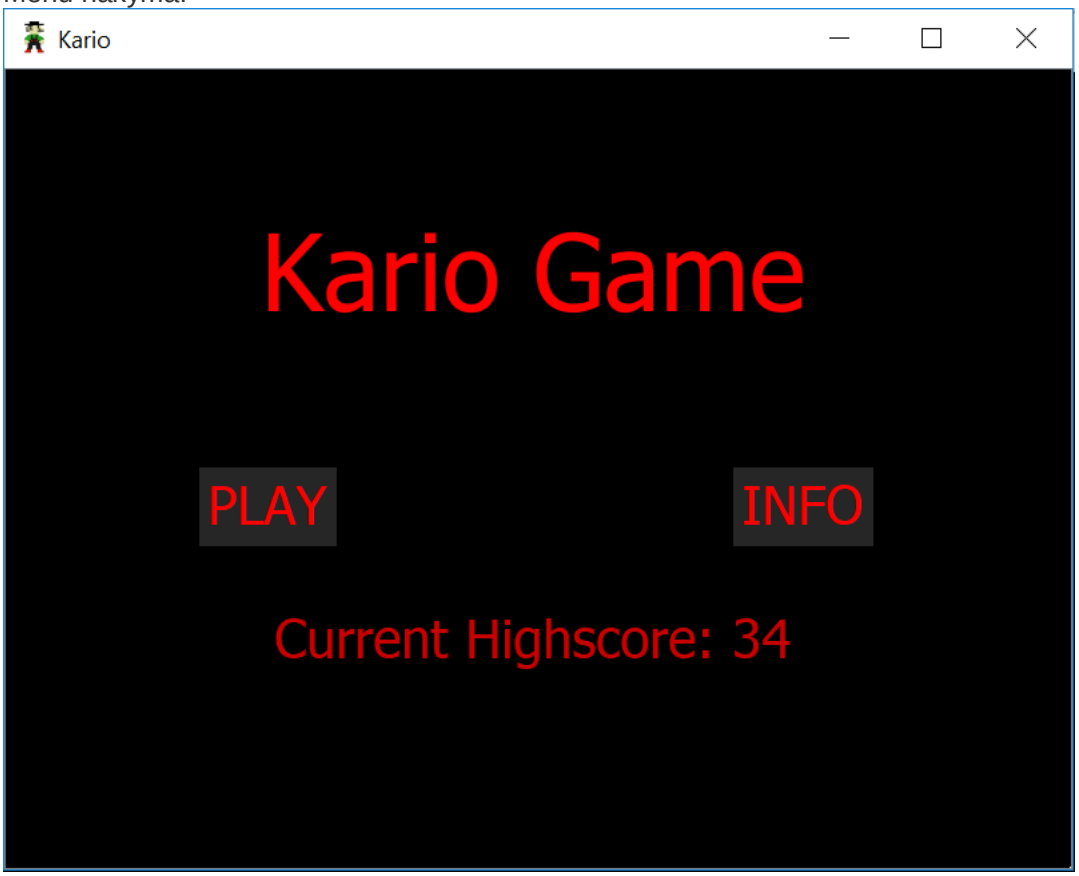
2. **Yleiskuvaus**

   Toteutin tasohyppelypelin, jossa pelaaja liikkuu pelin sisällä, keräten palkintoja ja yrittäen päästä maaliin. Tarkoituksenani oli toteuttaa projektin keskivaikea toteutus, joskin pelini sisältää muun muassa valikon, liikkuvan vihollisen ja graafisia tehosteita itse pelin sisällä, esimerkiksi pelaajan liikkumisen animointia, joten toteutus on hieman keskivaikeaa haastavampi.

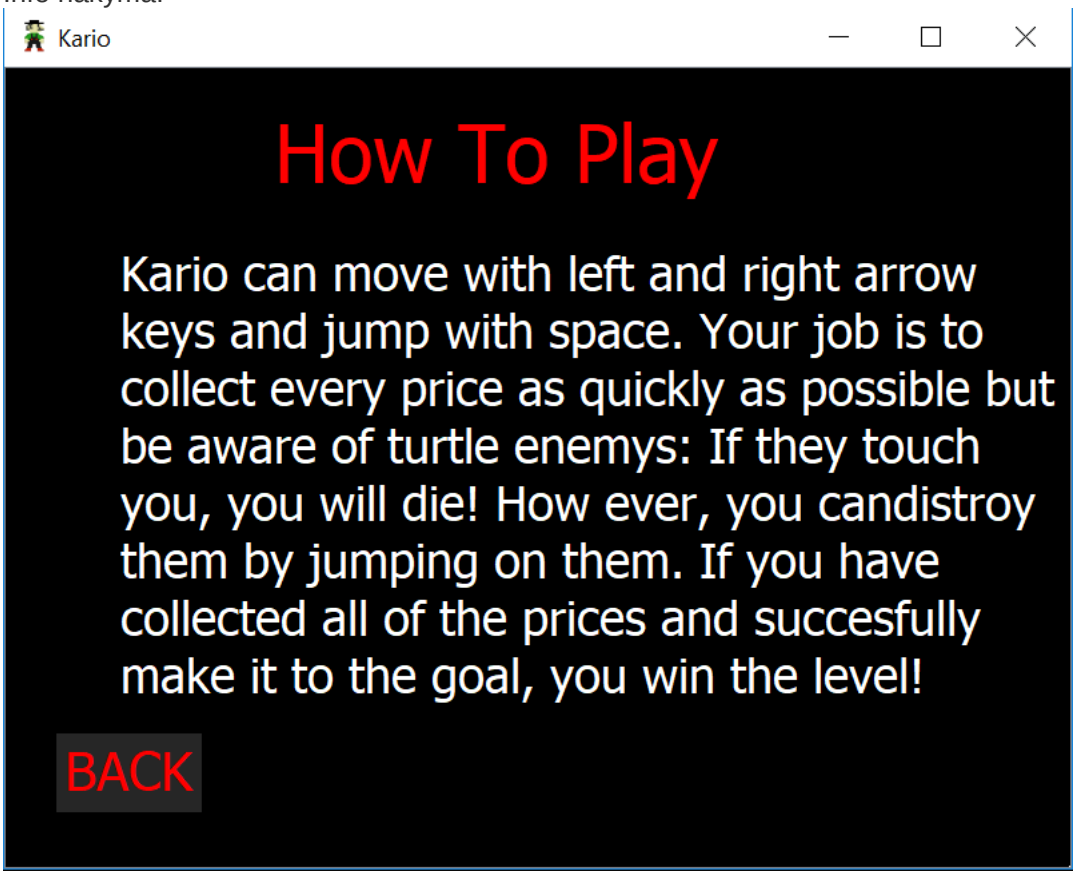
3. **Käyttöohje**

   Peliä pääsee pelaamaan ajamalla src tiedostossa sijaitsevan main.py tiedoston. Pelin alkaessa pelaaja voi valita infon, jossa kerrotaan tarkemmin, miten peliä pelataan, tai suoraan itse pelin pelaamisen hiirellä valitsemalla. Alla erilaisia näkymiä itse pelistä.

   Menu näkymä:



Info näkymä:



Peli näkymä, jossa pelaaja kuollut:

Pelinäkymä, jossa pelaaja tehnyt uuden highscoren:

4. **Ulkoiset kirjastot**

   PyQt:n lisäksi en ole käyttänyt muita ulkoisia kirjastoja

5. **Ohjelman rakenne**

   Ohjemani jakautuu karkeasti jaoteltuna näkymiä ja niiden toimintaa määritteleviin luokkiin, itse näkymien sisällä oleviin peliaiheisiin luokkiin ja niin sanottuihin apuluokkiin. Menu, Scene ja Infoscene ovat kaikki PyQt:n QGraphicsScenen alaluokkia ja ovat vastuussa pelaajalle näytettävästä materiaalista. Camera luokka on QGraphicsView:n alaluokka ja se määrittelee, mikä edellisistä QGraphicSceneistä näytetään milloinkin. Se on myös vastuussa siitä, että pelihahmo näkyy koko ajan ruudussa. Nämä neljä luokkaa ovat ohjelman näkymän määritteleviä luokkia. Peliaiheisia luokkia ovat Enemy, Goal, Platform, Player ja Price. Jokainen näistä peliaiheisista luokista on QGraphicsPixmapItem:in alaluokka ja määrittelee yksittäisen peliin liittyvän kokonaisuuden toimintaa. Nimiensä mukaan Enemy määrittelee, miten vihollinen liikkuu ja miltä se näyttää, Goal on vastuussa maalin toiminnasta, Platform taas määrittelee erilaisten tasojen ulkonäköä, Player määrittelee pelaajaan liittyvää toiminnallisuutta ja Price palkintojen ulkonäköä ja sijaintia. Viimeisenä kategoriana on apuluokat, joita ohjelmassani ovat Globals ja Map. Globals ei varsinaisesti ole luokka vaan ennemminkin aputiedosto, jossa on määritelty peliin liittyviä globaaleita muuttujia helpottamaan kenttien rakennusta. Map luokka taas määrittelee kentän rakenteen ja esteiden paikat 2D lista tyyppisenä tietorakenteena.

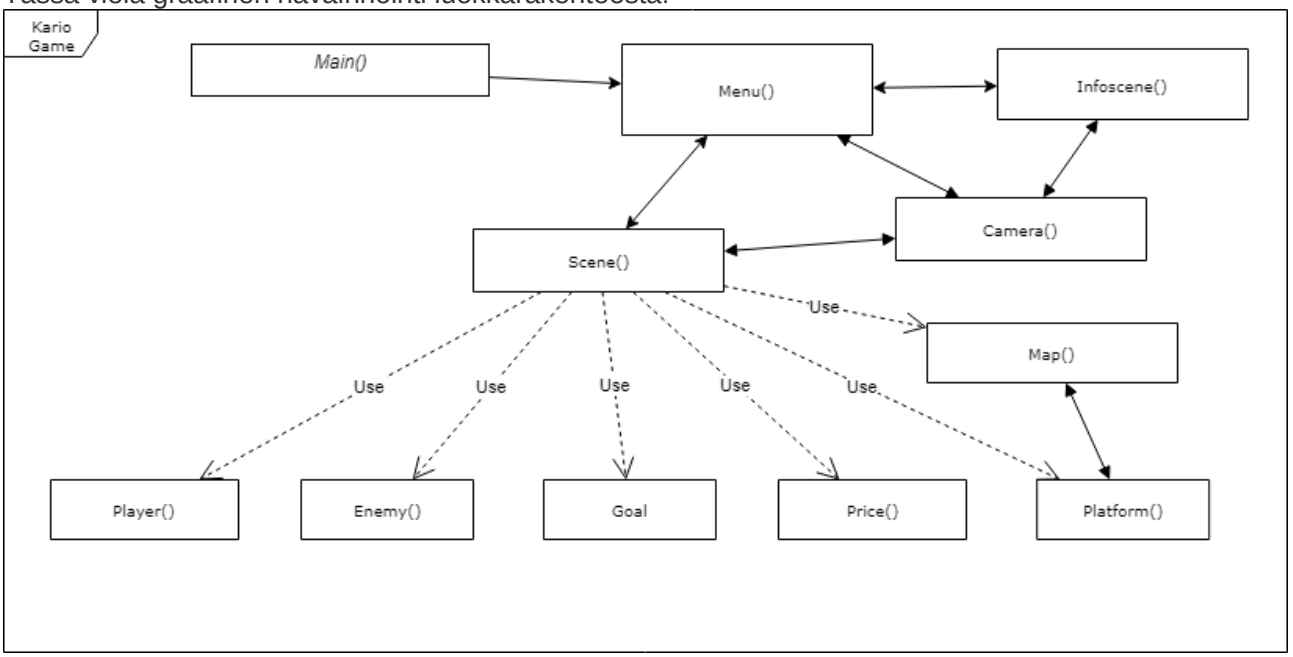
   Kaikki peliaiheiset luokat ovat Scene luokan alaisuudessa, kuten myös Map luokka, joiden avulla se luo pelimaailman ja ottaa pelaajalta vastaan käskyjä hahmon liikuttelua varten.

   Scene luokan tärkeimpiin metodeihin kuuluvat game_update, game_win ja game_over. Game_update metodia kutsutaan jokaisella timer_eventillä ja sen tehtävä on tehdä peliin liittyvät päivitykset. Game_win ja game_over ovat vastuussa tapahtumista, joissa pelaaja joko voittaa tai häviää pelin.

Camera luokan ainoa metodi on update_scene joka saa parametrina scnenen, johon kameran tulisi sillä hetkellä näyttää.

Enemy ja Player luokkien tärkeimmät metodit ovat enemy_update ja player_update, jotka nimiensä mukaisesti päivittävät vihollisen ja pelaajan sijainteja, ympäröivän maailman ja näppäimistöltä annettujen ohjeiden mukaisesti.

Tässä vielä graafinen havainnointi luokkarakenteesta:



6. **Algoritmit**

Yksi vaikeimmista projektissani tarvittavista algoritmeista oli eittämättä toimiva pelaajan ja pelikentän välinen törmäyksen tunnistus ja siitä johtuvat toimenpiteet. Ratkaisin ongelman sellaisella algoritmilla, joka tarkistaa jokaisen piirretyn kuvan välissä pelihahmon nopeusvektorin perusteella ensin x-suunnassa mahdolliset törmäykset, ja sitten y-suunnassa. Käytännössä algoritmini olettaa, että pelaaja ei tule tekemään mitään muutoksia hahmon liikkeeseen ja tarkastaa, sitten tulisiko seuraavan piirretyn kuvan aikana törmäys. Jos törmäys tulisi, liikutetaan pelihahmoa ainoastaan sen verran, ettei se pääse laattojen sisälle. Käytin samaa törmäyksen tunnistusta myös vihollisen liikkeen toteutuksessa, mutta aina x-suuntaisen törmäyksen sattuessa vihollinen vaihtaa kulkusuuntaa. Kyseinen Algoritmi on ehdottomasti paras minun projektiini sillä se yksinkertaistaa törmäyksen tarkastelun aina yhdelle akselille, mutta silti on tarpeeksi tehokas toimimaan pelin sisällä.

Toinen hankalamman puoleinen toteuttamani algoritmi on pelihahmon liikkeen animointi. Tätä ongelmaa lähdin ratkaisemaan siten, että laitoin hahmon eri liikkeitä edustavat kuvat kahteen eri listaan sen perusteella, kuvaavatko ne hahmon liikettä oikealle vai vasemmalle. Sitten, kun pelaaja painaa esimerkiksi vasenta nuolinäppäintä, ruvetaan kasvattamaan muuttujan suuruutta tietyn verran jokaisen piirretyn kuvan välissä. Tämän muuttujan kasvu käytetään animoinnissa hyväksi siten, että piirretään aina muuttujan kuvaaman kokonaisluvun nimeämä listan alkio. Että saataisiin jatkuvaa animointia, syötetään listalle muuttuja modulo-operaattorin läpi ja tällöin animaatio käy läpi kaikki listassa olevat kuvat jatkuvasti, kunnes pelaaja päästää nuolinäppäimestä irti ja muuttuja nollataan.

7. **Tietorakenteet**

Käytin suurimaksi osaksi pythonin omia, muuttuvatilaisia ja muuttumattomia, tietorakenteita ohjelmassani. Kuitenkin yksi tietorakenne eli map on osittain minun suunnittelemani. Map:in ideana on tallentaa kenttä 2D listaan siten, että kentän piirtäminen ja törmäysten tunnistaminen helpottuu. Käytännössä siis kentän jokaista 40*40 pikselin kokoista laattaa kuvaa yksi luku kaksiulotteisessa listassa. Tätä tietorakennetta käytän hyväkseni törmäyksentunnistus

algoritmissanikin. Pelihahmon paikka maailmassa on helppo selvittää syöttämällä map tietorakenteelle hahmon koordinaatit, jolloin saadaan tietää, sijaitseeko tietyssä pisteessä kenties tiililaatta, vettä vai ehkä ilmaa ja täten törmäyksentunnistus algoritmi voi päätellä mitä kyseisessä koordinaatissa pitäisi tehdä. Myös itse maailman suunnittelu on paljon helpompaa map tietorakenteen johdosta. Kyseinen tietorakenne on Map luokan ainoa muuttuja.

8. **Tiedostot**

   Ohjelmani käsittelee ainoastaan yhtä ulkoista tiedostoa, jonka tarkoituksena on pitää kirjaa tämän hetkisestä highscoresta. Tiedosto on formaatiltaan json tyyppinen ja ohjelmani muokaa sitä aina kun siihen on tarvetta, eli silloin kun tulee uusi ennätys. Tiedosto sijaitsee src kansion static kansiossa.

9. **Testaus**

   Ohjelmaa rakentaessani, testasin sitä lähinnä laittamalla print-komentoja sellaisiin paikkoihin, että pystyin varmentamaan ohjelman oikeanlaisen toiminnan. Lisäksi lisäsin ohjelmaani yksikkötestauksen, mikä testaa ohjelmani kannalta kriittisiä vaatimuksia, joiden puuttuminen johtaisi mahdollisesti ohjelman kaatumiseen. Suunnitelmassa kuvattuja yksikkötestejä en pitänyt enää järkevänä toteuttaa, sillä projektin edetessä, keksin tapoja suorittaa mainitsemani ongelmat tavoilla, jotka ovat huomattavasti vähemmän virhealttiita.

10. **Ohjelman tunnetut puutteet ja viat** Tällä hetkellä ainoa löytämäni selvä vika/puute on pelaajan niin sanotun hitboxin koko verrattuna itse piirrettyyn pelaajaan. Toisin sanoen, jos pelaaja seisoo aivan laatan reunalla, näyttää se siltä, että hahmo leijuisi ilmassa.

11. **3 parasta ja 3 heikointa kohtaa** Mielestäni kolme parasta kohtaa pelissäni ovat: Yleinen graafinen ilme, pelaajan ja maailman välinen erittäin toimiva törmäyksentunnistus ja kameran liikkuminen pelaajan mukana, johon keksin tehokkaan, mutta yksinkertaisen ratkaisun. Heikkona kohtana voidaan pitää toteutustani, jossa menun kanssa voi vuorovaikuttaa ainoastaan hiiren painalluksilla, eikä näppäimistöllä olenkaan. Tämä ja myös edellisessä kohdassa mainitsemani puute olisi kuitenkin melko helppo korjata, jos haluaisin jatkaa projektin työstämistä.

12. **Poikkeamat suunnitelmasta**

    Yksikkötestaussuunnitelmaa lukuun ottamatta projektini ei poikkea hirveän paljoa suunnitelmasta. Aikataulun osalta toteutettujen asioiden järjestys oli hieman erilainen, kuin suunnitelmassa ja ajankäytöllisesti käytin noin 10-20 tuntia yli sen, mitä olin suunnitellut.

13. **Toteutunut työjärjestys ja aikataulu** Ensimmäiset tiedostot (15.3), Ensimmäinen toimiva pelityyppinen toteutus (16.3), edellisen päivän tuotoksien hiomista (17.3), menun tekemisen aloitus (18.3), menun ja pelin välinen vuorovaikutus (19.3), info-näkymänrakentelua (20.3), Törmäyksen tunnistamisen varsinainen toteutus (4.4), graafisia lisäyksiä peliin (5.4), Parannuksia pelaajan liikkeeseen (8.4), Maalin toteutus (9.4), kokonaisuuden viilaamista (18.4), viimeiset lisäykset (22.4). Kokonaisuudessa toteutus vastaa melko hyvin suunniteltua. Ainoastaan pieniä muutoksia toteutus järjestyksessä.

14. **Arvio lopputuloksesta**

    Mielestäni projektini lopputulos on erinomainen. Laatu on mielestäni hyvä niin peliteknisessä mielessä, kuin myös graafisesti ja ulkonäöllisesti. Luokka jako toimii mielestäni hyvin, mutta jos tekisin ohjelma, nyt uudestaan, vetäisin ehkä vieläkin selvemmät rajat sille, mitkä asiat ovat minkäkin luokan vastuulla. Laajennettavuuteen ja sen miettimiseen käytin kuitenkin paljon aikaa ja mielestäni uusien tasojen lisääminen peliin olisi todella yksinkertaista. Kaiken kaikkiaan mielestäni tekemäni työn laatu ja käytetty aika näkyvät hyvin projektin laadussa ja lopputuloksessa.

15. **Viitteet**

    https://gist.github.com/rogerallen/f06ba704ce3befb5459239e3fdf842c7

    https://github.com/OneLoneCoder/videos/blob/master/OneLoneCoder_PlatformGame1.cpp

https://doc.qt.io/qt-5/index.html

https://docs.python.org/3/library/index.html

https://www.riverbankcomputing.com/static/Docs/PyQt4/index.html