Ohjelmoinnin peruskurssi Y2 CS-A1121

6.5.2021

1. Tasohyppelypeli

Aaro Reijola, 731065 (AUT2019)

2. Yleiskuvaus ja vaikeustaso

Projektissa on luotu tehtävänannon 9.2 mukainen, vaativan tason vaatimusten mukaan laadittu tasohyppelypeli, jossa pelaajalla on tarkoituksenaan kerätä kaikki kolikot ja selvitä maaliin hengissä. Yksinkertaisilla grafiikoilla varustetussa pelissä liikutaan wad-tai nuolinäppäimillä, minkä lisäksi peli sisältää interaktiivisen, graafisen kenttäeditorin, jolla omia tasoja voidaan luoda sekä tallentaa. Peli sisältää viisi valmista tasoa.

3. Käyttöohje

Peli käynnistetään ajamalla main.py-tiedosto projektin src-kansiosta. Valikosta valitaan joko hiirellä tai nuolinäppämillä haluttu toiminto. Itse pelaaminen on yksinkertaista nuolinäppäinten painelua, mutta peli on nopeatempoinen. Kolikoiden keräämistä vaikeuttavat tasoihin aseteltavat piikit sekä liikkuvat viholliset. Lisäksi pitää varoa maailman laidalta putoamista.

Kenttäeditorin puolella kamerapalikkaa liikutetaan WASD-näppäimillä ja muokattavaan tasoon lisätään eri objekteja joko yhdellä klikkauksella tai kahden klikkauksen yhdistelmänä. Jokaisen lisätyn palikan tai objektin voi poistaa Z-näppäimellä, tai pyyhekumi-valinnalla klikkaamalla. Muita valintoja voi tehdä joko valikosta tai palkeissa kerrotuilla pikanäppäimillä. Tason muokkaamisen jälkeen valikosta napsautetaan Poistu ja Tallenna, ja syötetään tasolle sopiva nimi. Tiedostonnimien käsittelyn helpottamiseksi nimi tarkistetaan erikoismerkkien sekä ääkkösten varalta ja tallennetaan vain aakkoset ja numerot hyväksyen.

Tason nimeäminen olemassaolevan tason nimiseksi korvaa vanhan tason ilman erillistä varmistusta, joten nimen kanssa kannattaa olla tarkkana. Vain Default-nimi on suojattu. Lisäksi levels-kansioon luotuja tiedostoja ei voi poistaa ohjelmasta, mikä olisi tietysti erinomainen jatkokehityskohde projektille.

4. Ulkoiset kirjastot

Projektissa on käytetty vain PyQt5-kirjastoa, joka hoitaa Qapplicationin pyörittämisen sekä tarjoaa välineet graafisen käyttöliittymän luontiin. Kaikki projektin graafiset elementit pohjatuvat jotenkin Qt:n Qwidget-luokkaan, ja kirjaston dokumentaatio löytyy täältä.

5. Ohjelman rakenne

Ohjelmassa on GUI(QMainWindow)-luokka, joka toimii ikkunan ylimmän tason tirehtöörinä. Se siis huolehtii eri näkymien tuomisesta näytölle. GUI-luokan alle on näkymästä riippuen lisätty erilaisia Qwidget-luokan jälkeläisiä, joita on useita luotu itsekin menu.py-moduulissa. Ainakin kaikki ohjelman valikot sekä painikkeet ovat omina luokkinaan tiedostossa ja niitä voi pienellä kustomoinnilla käyttää nopeasti uusien näkymien luomiseen. Projektin tehtävänannossa kehotettiin keskittymään laajennettavuuteen, ja valmiiden valikoiden ja grafiikkaelementtien avulla se käy mutkattomasti.
Jälkikäteen ajateltuna olisi ollut viisaampaa jakaa kaikki näkymät omiin luokkiinsa ja tehdä ne itse QgraphicsView-luokan pohjalta (kuten LevelEditor) ja lähettää GUI-luokalle vain tieto näkymän

tehtävän valmistumisesta. Nyt GUI-luokka hoitaa kaikkien valikoiden pyörittelyn ja painikkeiden toimintojen yhdistelyn.

Käyttöliittymän lisäksi ohjelma sisältää platformerGame-luokan, joka pitää kirjaa käynnissä olevan pelin tilanteesta, sekä huolehtii kaikkien näytöllä olevien objektien päivittämisestä jokaisella gametickillä. Päivityssykli saadaan pelin ollessa käynnissä GUI-luokan Qtimerista. PlatformerGame-luokka saa pelissä käytetyn QgraphicsScenen tiedostonlukulogiikalta ja huolehtii, että peli ei kaadu virheellisiäkään tiedostoja avatessa.

Syvemmällä pelilogiikan syövereistä löytyvät erilaiset luokat pelihahmoille sekä kentän objekteille. Näyttöön piirrettävät objektit on kaikki perivät QgraphicsItem-luokan tai sen jälkeläisen. Pelissä on luokka Character(QGraphicsRectItem), joka toimii normaalin QgraphicsItemin tavoin scenellä, mutta sisältää lisäksi yhtenäisiä tietoja hahmojen grafiikoista, tilasta, sekä liikkumisen hoitavasta CharacterPhysics-luokasta.

Koska QgraphicsRectItem-luokka tarjoaa valmiit välineet törmäysten tunnistukseen scenellä, on törmäysten tunnistaminen sekä paikan muutosten validoiminen luontevaa hoitaa Character-luokassa. Tämä tapahtuu luokan yleisessä move-metodissa. Luokan jälkeläiset Player, sekä Enemy, lisäävät omat toiminnallisuutensa, eli lähinnä liikkumislogiikkansa, tähän, minkä jälkeen ne kutsuvat yläluokan metodia.

Character-luokkien lisäksi characters.py-tiedostossa on määritelty pelin eri objektit, sillä niihinkin liittyy olennaisesti törmäyksentunnistus. Näitä ovat Spike, Coin sekä Goal, ja kaikille on määritelty omat metodinsa, joita pelaajaneliö kutsuu niihin törmätessään. Uusia törmäyksen tunnistavia objekteja olisikin luontevaa lisätä, kun törmäyksentunnistuksen osalta niille pitää lisätä vain collide-metodi ja jokin grafiikka.

Itse liikkumisen logiikka hoidetaan CharacterPhysics-luokassa. Jokaisella Player- tai Enemy-hahmolla on oma tämän luokan phys-attribuutti, joka valvoo nopeuksia eri vapausasteiden suhteen, putoamisaikaa, sekä sitä, onko hahmo maan pinnalla, jolloin se voi mahdollisesti hypätä. Laskenta on järkevää eriyttää omaan luokkaansa, sillä fysiikan lait vaikuttavat pelissä kaikkiin hahmoihin yhtäläisesti. Nopeuksiin ja kiihtyvyyteen sekä putoamisaikaan perustuva ratkaisu mahdollisti realistisen näköisen (ja tuntuisen) lopputuloksen. Laskennassa käytetyt ”luonnon”vakiot ovat helposti säädeltävissä luokkamäärittelyn alussa.

GameLevel-luokka on tavallaan tietorakenne, joka pitää kirjaa kunkin tason esteistä, kolikoista, vastustajista, piikeistä sekä maalista. Luokka myös kertoo pystyykö tason avaamaan pelattavaksi, ja jos ei, niin tasoa ei anneta pelata. GameLevel-luokan olioita pyöritetään ohjelmassa parissa eri tilanteessa. Kun uusi peli alkaa, GUI kertoo platformerGame:lle haluavansa scenen näytölle. Valitun tason nimen perusteella haetaan levels-kansiosta tason tiedosto, josta Loader-moduulissa luodaan LoadLevel-funktiolla GameLevel-olio. GameLevelin generateScene-metodilla tasosta saadaan luotua varsinainen näytöllä näkyvä QgraphicsView:n kanssa toimiva QgraphicsScene. Vastaavanlaisesti LevelEditor-luokka tuottaa käyttäjän syötteen perusteella GameLevel-tyyppisiä rakenteita, jotka voidaan Loaderilla tallentaa tiedostoon myöhemmin pelattavaksi.

LevelEditor-luokka perii QgraphicsView:n, ja näin sen voi suoraan lisätä GUI:n näkymään widgettinä. Tämä ratkaisutapa olisi varmasti ollut parempi myös pelin ja valikoidenkin kannalta. LevelEditor hoitaa käyttäjän syötteen perusteella luotavaa QgraphicsSceneä. Se lisää ja poistaa objektit näkymästä sekä luotavasta GameLevel-rakenteesta, ja hoitaa niihin liittyvät tarvittavat tarkistukset, jotta vältytään virhetilanteilta. LevelEditorissa on pitkä lista erilaisia pikanäppäinmäärittelyitä ja käyttäjän valinnoista riippuvia toimintoja, jotka olisi ehkä kannattanut

vielä jakaa omaan luokkaansa. Näin olisi todennäköisesti saatu selvempi raja editorin luoman scenen ja käyttäjän syötteen prosessoimisen välille.

6. Algoritmit

Pelissä hahmojen ohjaus perustuu CharacterPhysics-luokan laskentaan, joka laskee hahmojen nopeuksien ja putoamisaikojen perusteella seuraavat kohdat. Mekaniikan kaava tälle on $x(t) = 0.5a*t^2+v*t+x_0$, missä x on paikka, v nopeus, a kiihtyvyys ja t tietenkin aika. Peli toimii vakioaikaisten diskreettien muutosten avulla, joten CharacterPhysics antaa paikan muutoksia tiettyä aikayksikköä kohti. Ihmissilmälle miellyttävännäköinen hyppyliike saadaan aikaan kiihtyvyyksiä hyödyntämällä.

Suunnitelmassa mainittu etäisyyksien laskeminen on projektissa jäänyt lähes kokonaan Qt5-kirjaston hoidettavaksi, vaikkei laskentatapa siitä mihinkään muutukaan. Lähinnä painopiste on siirtynyt koordinaatistomuunnosten ratkaisemiseen objektien tekstuurien säilyttämiseksi. Tämä tapahtuu luomalla ensin objekti omaan koordinaatistoonsa yläkulma origoon, ja sen jälkeen siirtämällä sitä omassa koordinaatistossaan mittojen puitteessa siten, että origo saadaan haluttuun kohtaan.

Törmäyksentunnistusalgoritmi ohjelmassa menee seuraavasti:
1. Haetaan mekaniikan perusteella seuraava sijainti
2. Siirretään kappale uuteen paikkaan
3. Jos syntyi törmäys, siirretään se takaisin ja kokeillaan uudelleen lähempänä lähtöpaikkaa
4. …
5. Kappale on jonkin tarkkuuden puitteissa kiinni törmäävässä pinnassa.

Tämän olisi tietysti voinut hoitaa hienostuneemminkin, mutta kyseinen ratkaisu oli todellakin yksinkertaisin saatavilla olevista vaihtoehdoista toteuttaa käytännössä. Esimerkiksi binäärihaulla olisi voinut tehdä saman muutamaa toistoa nopeammin. Ongelman laskentatehovaativuuden kannalta valinnalla ei ole suurtakaan merkitystä, sillä laskutoimituksia tulee tästä tarkistuksesta yhtä hahmoa kohden sekunnissa joitain kymmeniä, häviävän pieni määrä nykypäivän tietokoneissa. Jos lähdettäisiin tekemään pelimoottoria, niin valintaa pitäisi tietysti miettiä enemmän.

7. Tietorakenteet

Ohjelmassa käsitellään pelin aikana GameLevel-rakenteita, jotka ovat käytännössä kokoelma eri listoja. Eri listoja on objekteille (palikat, piikit), hahmoille (pelaaja, viholliset) sekä kerättäville esineille (kolikot). Lisäksi rakenne sisältää tiedon tason nimestä, aloituskohdasta sekä maalista. Tietorakenteeksi lista oli luonnollinen valinta, sillä tasoja käsitellään syklisesti iteroimalla tason jokaista objektia. Muuttumattomien rakenteiden käyttö ei olisi tasohyppelypeliin soveltunut oikein mitenkään, korkeintaan ehkä grafiikkatietojen säilytykseen.

8. Tiedostot

Ohjelma käsittelee txt-tiedostoja tasojen tallentamisen yhteydessä. Tasojen tiedot tallennetaan yhteen tiedostoon tasoa kohden, ja tiedoston nimi on *nimi*.txt. Formaatin pääpiirteet ovat seuraavanlaiset: Jokainen rivi kertoo yhden objektin ominaisuuden. Rivin viisi ensimmäistä merkkiä kertovat minkälaista tietoa loppurivi sisältää, esim. NAME: kertoo vielä tason nimen, OBJCT palikan sijainnin sekä koon ja ENEMY vastustajan sijainnin sekä liikeradan päätepisteen. Kaikki vieraat viisimerkkiset rivinalut jätetään tiedostoa lukiessa huomiotta. Esimerkkinä 1. Tason tallennustiedot:

```
NAME:1. Taso
SPWN:0,0
OBJCT0.0,100.0,500.0,50.0
OBJCT300.0,-300.0,50.0,300.0
ENEMY100.0,20;435.0,20;
####ENEMY200.0,34.0;100.0,34.0;                    #Tämä rivi jää huomiotta
COIN:-100,-200
COIN:600, 50
GOAL:500,34
```

Tallennetut tasot sijaitsevat src/levels/ kansiossa. Valitettavasti tiedostojen poisto ei ole ohjelmasta käsin vielä mahdollista.

9. Testaus

Ohjelmaa testattiin projektin elinkaaren aikana jatkuvasti, sillä käyttöliittymän rakentaminen perustuu suurilta osin vanhan sisällön päälle lisäämiseen. Testaus painottui painikkeiden ja toimintojen käyttöön itse, sillä graafisen käyttöliittymän automaattisen testauksen suhteen osaaminen loppui kesken. Graafisten elementtien osalta ohjelmaa testattiin jokaisen valikon ja painikkeen osalta tiheästi elementtien kirjoitusvaiheessakin, mikä helpotti tietysti paljon ulkoasun saamista halutunlaiseksi. Projektisuunnitelmassa mainitut nappien placeholder-toiminnot olivat suuressa roolissa ulkoasun testauksessa.

Itse peli kehitettiin ensin siihen kuntoon, että käyttäjä saattoi liikkua kovakoodatussa tasossa ja tutkiskella esimerkiksi törmäyksiin liittyviä bugeja rauhassa. Riittävän pelattavuuden saavuttamisen jälkeen kehitettiin kenttäeditori, jolla lopuksi luotiin pelin tasot. Järjestys poikkesi alkuperäisestä suunnitelmasta hieman, sillä alun perin oli tarkoitus kehittää ensin kenttäeditori jatkuvaa testausta helpottamaan. Tämä järjestys tuntui kuitenkin myös järkevältä, sillä kenttäeditorin luominen ilman toimivaa testikenttää olisi ollut todella haastavaa.

Kenttäeditorin hiominen kuntoon paljasti monia bugeja varsinkin kenttien pelattavuuden validoinnin osalta. Esimerkiksi pelin kaatuminen sen takia, että spawnpoint oli laatikon sisällä saatiin korjattua kenttäeditorin ansiosta. Alkuperäisessä testaussuunnitelmassa tavoitteeksi oli määritelty projektisuunnitelman kohdan 3 mukainen käyttötapaus, jossa sovellus avataan, taso läpäistään, ja käyttäjä ohjataan editoriin tai pois sovelluksesta. Tällaisen tilanteen sovellus hoitaa mallikkaasti, ja kenttäeditorilla pystyy luomaan monimutkaisiakin tasoja. Normaalikäytössä sovellus on pelattava ja pienellä mielikuvituksella oikein viihdyttävä.

Testaussuunnitelmassa mainittu black-box-testaus jäi ajanpuutteesta (vapusta) johtuen vähälle, mutta sitäkin on suoritettu parin henkilön toimesta. Tästäkin oli suurta hyötyä puutteiden löytämisessä, sillä uusi käyttäjä ei todellakaan ole kehittäjän kanssa samalla viivalla sanan ’intuitiivinen’ merkityksen kanssa.

Yksikkötestejä tehtiin Loader-moduulin osalta. Testeissä varmistettiin, että oikein luodut GameLevel-pelirakenteet tallentuvat tiedostoihin oikein ja että saadaan sama rakenne, kun kenttä ladataan tiedostosta LoadLevel-funktiolla. Lisäksi testattiin, että täysin vääränlaiset tiedostot eivät mene tallennetusta tasosta. Ohjelma läpäisee kaikki kirjoitushetkellä olemassaolevat yksikkötestit. Testit luovat testitiedostoja levels/-kansioon, ja nämä rikkinäisetkin tiedostot näkyvät ohjelmaa suoritettaessa tasovalikossa. Rikkinäistä kenttää klikattaessa valikko ohjaa käyttäjän takaisin päävalikkoon. Koska tiedostoja poistavaa ominaisuutta ei ole peliin lisätty, niin testitiedostojen poistaminen jää valitettavasti käyttäjän itsensä vastuulle.

10. Tunnetut puutteet ja viat

Kuten mainittu, luotuja kenttiä ei tällä hetkellä voi poistaa ja tämä olisi loogisesti seuraava kehitysaskel.

Toinen puute on se, että ohjelmassa ei liiemmin ole varauduttu tarkoitukselliseen ohjelman rikkomiseen. Normaalikäytön se kestää hyvin, eikä se omissa testeissäni ja pelailuissani ole kaatunut pitkään aikaan mutta kaikkeen on vaikea varautua jos eri osista ei tee vedenpitäviä heti kehitysvaiheessa.

11. 3 parasta ja 3 heikointa kohtaa

- GUI-luokan laajentaminen on tällä hetkellä lähes mahdotonta ja se on koko projektin suurin häpeäpilkku. Koodaaminen ei aluksi ollut päämäärätietoista, vaan pikemminkin eri asioiden kokeilemista ja se mikä toimi, jäi lopulliseen työhön. Alussa olisi pitänyt tutustua vielä paremmin Qt:n widgettien muodostamaan hierarkiaan sekä signaaleihin, joita eri widgetit toisilleen lähettävät. Nyt GUI sisältää montakymmentä hieman epäselvää metodia, jotka on kirjoitettu vastaamaan kunkin napin painallusta, kun paljon helpommalla olisi päässyt miettimällä toiminnot huolella läpi ja kirjoittamalla yhden tai pari metodia, jotka nollaavat näkymän. Tämän jälkeen olisi ollut paljon helpompi rakentaa käyttöliittymää pala palalta, jolloin eri näkymätkin olisi voinut jakaa eri luokkiin paljon järkevämmin.

- Toinen heikkous on, että lopputuloksessa ei tosiaan neuvota käyttäjää kovin paljoa. Tämä on kenttäeditorin osalta hieman pienempi ongelma, sillä sen kehitysvaiheessa palautetta oli jo tullut testaajilta sen verran, että ohjeistus ehti mukaan. Alkuperäisessä suunnitelmassa ensimmäisen tason piti olla tutoriaali, jossa pelaajalle opetetaan pelin mekaniikat, liikkuminen, tavoitteet sekä erilaiset kentällä näkyvät objektit. Valitettavasti tutoriaali jäi ajanhallinnan takia pois ja lopputulos on tylsä ykköstaso, jossa pelaaja voi vapaasti kokeilla löytää nuolinäppäimet näppäimistöltään.

- Lisäksi projektin toteutus oli ehkä liian kunnianhimoinen siinä mielessä, että eri osien toteutus olisi voinut olla paljon yksinkertaisempikin. Nyt eri osia ja samalla mahdollisia bugeja on todella paljon. Kun käyttäjä voi antaa ohjelman näkökulmasta rajattoman määrän erilaisia inputteja, ohjelma pitäisi kehittää tämä mielessäpitäen. Virhetilanteisiin varautuminen on jäänyt liian vähälle, ja vaikkei ohjelma enää normaalikäytössä kaatuilekaan, niin silti olisi hyvä kertoa käyttäjälle mahdollisesta virheestä edes jotain tietoa.

+ Paras kohta on pelin pelattavuus. Olen itse todella tyytyväinen varsinaiseen pelikokemukseen, josta tuli kenttäeditorin mahdollistaman kenttien luomisen myötä nopeatempoista ja jopa viihdyttävää.

+ Kenttäeditorista tuli yllättävänkin hyvä. Erityisen ylpeä olin laatikoiden piirtämiseen liittyvän bugin korjaamisesta, sillä editorin elämän alkuvaiheessa hiirellä palikoiden piirtely näytti aivan liian kunnianhimoiselta tavoitteelta ja harkitsinkin koko idean vaihtamista vakiokokoisiin laatikoihin. Bugi liittyi hiiren koordinaatteihin ja Qt:n tapaan luoda laatikot. Laatikoilla on valmis konstruktori, jolle annetaan vain laatikon kaksi nurkkaa, joiden välille laatikko luodaan. Täydellinen kenttäeditorille! Joskus klikkien järjestyksestä riippuen tämä menettely loi kuitenkin laatikoita negatiivisilla mitoilla, mikä kyllä piirsi laatikon näytölle oikein, mutta törmäyksentunnistus ei toiminut alkuunkaan. Viheliäisen bugin korjaamisen jälkeen lopputulos on oikeasti käyttökelpoinen!

+ Lisäksi mainitsen laajennettavuuden erilaisten pelielementtien näkökulmasta. Olen leikitellyt ajatuksella kerättävistä powerupeista, kuten muissakin klassisissa tasohyppelypeleissä. Nykyisen luokkajaon perusteella sellaisten tekeminen olisi hyvinkin tehtävissä, ja muutenkin erilaisia

hahmoja pystyy kehittämään suhteellisen vaivattomasti. Jatkokehityksen kannalta olisi kuitenkin suotavaa kehittää vielä yhtenäisempi tapa luoda hahmoja ja esineitä, sillä nyt eri luokat ottavat hieman eri parametrejä, minkä takia niiden luominen vaatii pelin koodiin jokaiselle erilaisen käsittelytavan.

12. Poikkeamat suunnitelmasta

Suunnitelmassa mainittu tutoriaali jäi valitettavasti pois osittain ajankäytön ongelmien takia, osittain kenttien suunnittelun takia. Kenttiin ei lisätty riittävän varhaisessa vaiheessa järkevää tapaa luoda tekstielementtejä tai muita vastaavia, joten ominaisuus jäi lopullisesta versiosta uupumaan. Lisäksi suunnitelmassa mainittiin ääniefektit, jotka myös puuttuvat nykyisestä pelistä kokonaan. Niitä ei vaadittu millään tavalla, mutta ne olisivat kyllä kruunanneet kokonaisuuden.

Myös luokkajako poikkesi hieman suunnitelman UML-kaaviosta, vaikkakin merkittäviltä muutoksilta säästyttiin. Suurin muutos on GUI-luokan siirtäminen hierarkian huipulle pyörittämään koko MainWindowia, eikä tämä selvästi ollut jälkeenpäin katsottuna kaikista järkevin valinta. En kuitenkaan osaa kertoa muuta perustelua ratkaisulle, kuin että olisi pitänyt osata projektin alkuvaiheilla erittäin laajan kirjaston käyttö paremmin.

Ajankäytön kannalta tuntimäärässä osuttiin arviolta lähelle suunnitelman tuntimäärää. Työmäärä ei kuitenkaan jakautunut yhtä tasaisesti, kuin suunnitelmassa oli sinisilmäisesti ajateltu. Kymmenen tunnin viikottainen työmäärä oli oikeasti lähempänä viittä nelosperiodin tenttiviikkoon asti. ELEC valitettavasti tykkää venyttää kurssit siten, että jokaisessa periodissa opiskelijalla on 5-6 puolikasta kurssia, joten projektille jäi korkeintaan yksi päivä viikossa. Jälkikäteen katsottuna olisin paljon mieluummin tehnyt projektin vaikka puolessa tästä ajasta ja tuplannut viikottaisen työmäärän. Viikon tauon jälkeen oli vaikea päästä kärryille edellisen työkerran tekemisistä ja tuhottoman paljon aikaa kului unohdetun uudelleenopetteluun. Koodaaminen oli todella paljon mukavampaa ja sujuvampaa loppuvaiheessa, kun projektiin pääsi pureutumaan toden teolla.

13. Toteutunut työjärjestys

Alunperin suunnitelmana oli luoda ensin ikkuna, sitten menuelementit, sitten kenttäeditori, ja lopulta itse pelin pelattava osuus. Editori jäi kuitenkin periaatteessa viimeiseksi, vaikka sillä nyt varsinaiset tasot luotiinkin. Järjestys tuntui suunniteltua järkevämmältä, sillä kenttäeditoria olisi ollut todella vaikea luoda ilman, että pelattavuuttta olisi testattu mitenkään. Editorin luomisen jälkeen suurimmat bugit osuivat yllättäen tasojen lataamislogiikoihin ja kenttien pelattavuuden validoimiseen. Editori toi mukanaan tilanteita, joissa spawnpointin päälle oli laitettu esineitä, jolloin pelaaja lisättiin kenttään palikan sisällä ja törmäyksentunnistuslogiikka jumittui, koska edellista sopivaa paikkaa johon pelaaja siirretään ei ollut.

Suunnitelmassa mainittiin, että 15.3-26.3 checkpointissa olisi tavoitteena olla pelattava runko, jolla voi kokeilla liikkumista. Tämä toteutuikin, ja pelissä oli lisäksi vastustajia, vaikkakin ne olivat todella bugisia. Suurimmat edistysaskeleet otettiin kolmen viimeisen viikon aikana, kun nelosperiodissa loppuvat kurssit hellittivät. Näiden kolmen viikon aikana luotiin tallennusjärjestelmä, valikoiden toiminnallisuudet, vastustajien liikkumislogiikka, kolikoiden kerääminen, maali, pelissä voittaminen sekä lopulta kenttäeditori, tässä järjestyksessä. Lopuksi keskityttiin pahimpien ohjelman kaatavien bugien paikkaamiseen sekä klikkausten koordinaattien tunnistamisen optimointiin.

14. Arvio lopputuloksesta

Kokonaisuudessaan olen tyytyväinen lopputulokseen, mutta en omaan työskentelyyni. Työmäärän kasaantuminen loppuvaiheille oli täysin omaa syytäni, mutta onneksi lopputuloksesta tuli näinkin hyvä. Pelattavuuden kannalta olen erittäin tyytyväinen, sillä pienellä mielikuvituksella kenttäeditorin puolella saa luotua oikein kätevästi haastaviakin kenttiä, joita on mielenkiintoista pelata. Kaikki pelin mukana tulevat tasot ovat läpäistävissä, vaikkakin neljäs taso on jo aika haastava. Käyttöliittymässä on vielä paljon hiottavaa, mutta se ei ainakaan itselläni projektin pääpaino missään vaiheessa ollutkaan. Tulevaisuudessa vastaava projekti alkaisi omalta osaltani perusteellisemmalla tarpeiden pohtimisella ja paremman luokkajaon laatimisella.

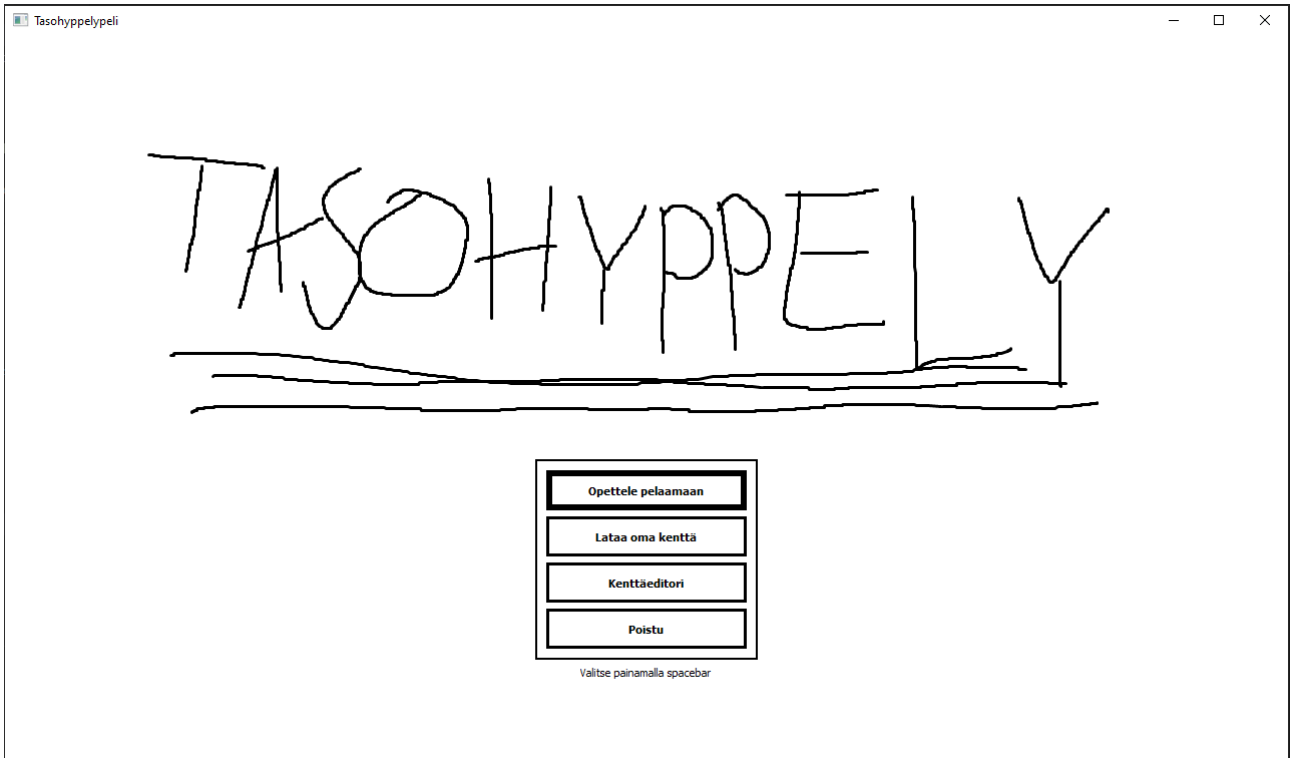
Koodin osalta lopputulos voisi olla parempi. Paikoin koodi on varmasti vaikealukuista ulkopuoliselle ja valitut ratkaisut hieman ontuvia. Kenttäeditorinkin tiedosto venyi 400 riviä pitkäksi, kun sen olisi voinut jakaa pariinkin pienempään kokonaisuuteen. Ongelman ratkaisu olisi ollut maltti koodausvaiheessa. Nyt into kokeilla aina jotain uutta johti siihen, että eri osien välisissä rajapinnoissa on toivomisen varaa ja kaikki osat tavallaan riippuvat liiankin suurilta osin toisistaan. Toisaalta olisi myös ollut mahdollisuus, että koodi ei edes toimi palautuspäivänä, joten ei lopputulosta murehtiakaan voi.

Jos projekti jatkuisi yllättäen toiset pari kuukautta, suurimmat korjaukset kohdistuisivat GUI-luokan täydelliseen remontointiin sekä käytöliittymän käyttäjäystävällisyyden parantamiseen. Lisäksi tutoriaalikenttä lisättäisiin peliin, jottei käyttäjää sysättäisi saman tien peliin ilman ohjeistusta. Todellisuudessa kaikki nämä olisivat olleet paremmalla ajanhallinnalla toteutettavissa tähänkin palautukseen.

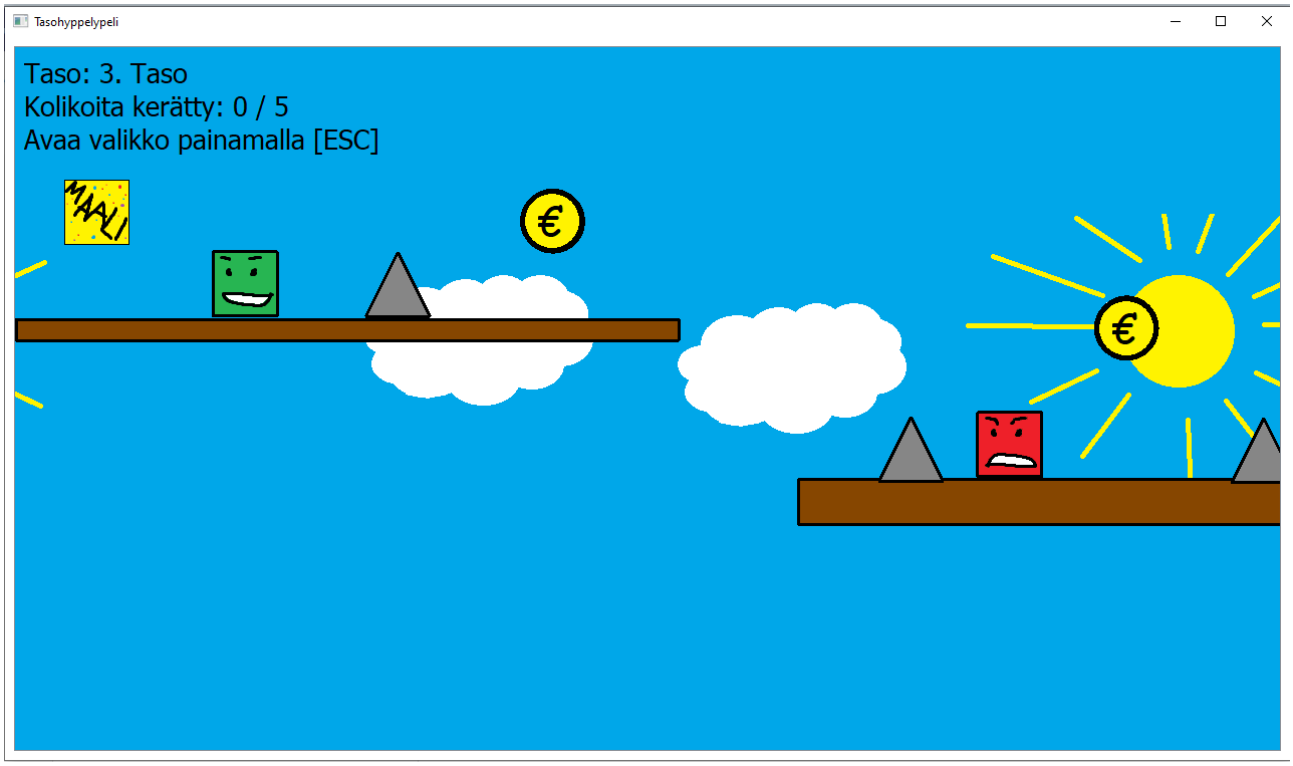
15. Viitteet

Qt:n dokumentaatio: https://doc.qt.io/qt-5/index.html
Pythonin dokumentaatio: https://docs.python.org/3/
Stackoverflow: https://stackoverflow.com/

16. Liitteet
1. Päävalikko



2. Kolmannen tason alku



3. Kenttäeditorin käyttöä: