perjantai 7. toukokuu 2021

Projektisuunnitelma – Golden Retriever

Henkilötiedot

Tekijä:

Olli Suoniemi 657 330

Insinööritieteiden kandidaattiohjelma, vuosikurssi 2018

Yleiskuvaus

Tämän projektin tarkoituksena oli harjoitella ohjelmointiprojektin tekemistä ja sen hallintaa tekemällä 2d-videopeli Python kielellä käyttäen ulkoista PyQt5 -kirjastoa. Tähän kuului itsenäisen ohjelman suunnittelu ja toteutus. Suunnittelussa käytin apunani UML-luokkadiagrammia havainnollistamaan ohjelmarakennetta ja luokkien välisiä yhteyksiä. Tavoitteenani oli tehdä vaativampi versio ohjelmastani. Projekti on laajuudeltaan melko laaja, ja siinä on paljon erilaisia ominaisuuksia. Laajuudeltaan se menee mielestäni keskitason ja vaativan väliin, sillä se täyttää keskitason vaatimukset ja osan vaativan projektin vaatimuksista.

Pelin päähahmo on koira, joka etsii kultaista luuta. Tavoitteena on löytää kultainen luu tekemällä tunneleita ja kaivamalla maata. Maata ja mineraaleja kaivamalla, pelaaja kerryttää pisteitään. Jos pelaaja löytää kultaisen luun, peli päättyy ja pelaaja voittaa. Jos pelaaja menettää kaikki energiansa, peli päättyy ja pelaaja häviää.

Käyttöohje

Tarvitset jonkin pythonia ajavan ohjelmointiohjelman. Lisäksi sinun tulee asentaa Python 3.5 >=, pip ja PyQt5. Asenna pip samalla kun lataat Python 3.5:n PyQt5:n saa asennettua ajamalla komentorivissä seuraavat komennot:

pip install pyqt5

Ulkoiset kirjastot ja muut työkalut

Projektissa on käytetty PyQt5 -kirjastoa. Tämän avulla piirsin pelin grafiikat, tarkistin objektien yhteentörmäykset sekä linkitin ääniefektit ja musiikit.

Tekstuurien piirtämiseen käytän Piskel- ja Gimp-ohjelmia, jotka ovat ilmaisia tekstuurin- ja kuvien muokkausohjelmia.

Pelin tunnusmusiikki on tehty Loudlyn Music Maker JAM -sovelluksella.

Ohjelman rakenne

Ohjelma koostuu main-luokasta, joka kutsuu ohjelman käynnistyessä peliluokkaa Game. Game-luokan lisäksi pelissä on Player, Tile ja World -luokat. Game luokka kutsuu käynnistyessään World-luokkaa. World-luokassa generoidaan satunnainen kartta halutuilla parametreilla. Kun kartta on generoitu, Game-luokka tarkistaa, että se on tarpeeksi syvä. Tämän jälkeen Game-luokan metodilla

”draw_world” piirrettään pelin kenttä ruutuun. Draw_world -metodi toimii siten, että se käy generoitua karttaa rivi kerrallaan läpi, ja piirtää kenttään kirjain koodattua vastaavan tiilen. Se kutsuu jokaisen tiilen kohdalla Tile-luokkaa, jossa ladataan kentän tekstuurit, ja luodaan ja asetetaan QPixmap-objektit kenttään.

Pelin maailma koostuu useista yksittäisistä Tile-luokan objekteista. Tile-objektin tärkeimmät muuttujat ovat: ”type” (tyyppi, str), ”level” (taso, int), ”value” (arvo, int), ”dig_time” (kaivuuaika, int), ”dug” (kaivettu, boolean). ”Type” kertoo tiilin tyypin, ja se on siis jokaiselle erilaiselle mineraalille erilainen. Sen perusteella metodi ”draw_world” tunnistaa kentän erilaiset mineraalit ja osaa luoda ja piirtää ne kenttään oikeanlaisina. ”Level” kertoo sen millä tasolla kyseinen mineraali on. Ensimmäinen rivi, josta mineraalit alkavat on 0-taso. ”Sand”-tyypin tiileillä syvyys vaikuttaa sen arvoon. Tämän vuoksi tarvitsemme syvyystietoa. Jokaisella mineraalilla on eri arvo, osa on yleisempiä ja toiset taas ovat harvinaisempia, kuin toiset. Tämän vuoksi tarvitsemme ”value” muuttujaa. ”Dig_time” kertoo kyseessä olevan tiilen kaivuuajan. Harvinaisemmilla tiileillä kaivuuaika on pidempi.

```
# Yleisimmästä harvinaisimpaan
    # name = type = value

       # sand = s = 1-9

       # bronze = b = 9
      # silver = l = 15
       # gold = g = 20

       # ruby = r = 50
     # diamond = d = 75
       # opal = o = 120

   # golden bone = w = N/A
```

Player-luokassa käsitellään pelin aikaiset tapahtumat kartan luomisen jälkeen. Luokka lukee ja käsittelee pelaajan antaman syötteen. Luokalle annetaan parametreina ”world” ja ”view”, jotka ovat siis Game-luokassa luodut QGraphicsScene-objekti ja QGraphicsView-objekti. Näin pääsemme käsiksi kätevästi edellä mainittuihin luokkiin. Player luokassa käynnistetään QTimer, joka kutsuu metodia timerEvent joka 10. millisekunti. TimerEvent-metodissa käsitellään paljon erilaisia ehto ja toistokäskyjä, joiden avulla pelin tapahtumia kontrolloidaan. Se toimii siis ikään kuin pelin moottorina. Ensimmäisenä se aina tarkistaa, onko muuttuja ”game_over” tosi. Tämän boolean arvon ollessa tosi, peli päättyy, ja pelaajan pisteet näytetään, ja kerrottaan voittiko vai hävisikö pelaaja pelin.

TimerEvent-metodissa on paljon erilaisten ehtojen tarkistuksia, ja en niitä kaikkia erittele, vaan avaan pääpuolin kyseisen metodin logiikkaa. Tässä on eriteltynä tärkeimmät kyseisen metodin tehtävistä:

- TimerEvent-metodi tarkistaa onko pelaaja törmäämässä HUD-tekstien (Heads up display) kanssa, jotka näyttävät siis pelaajan kerätyt pisteet ja energiamäärän.
- TimerEvent-metodi tarkistaa, onko pelaaja ilmassa, siten että alapuolella ei ole maata. Tällöin se aiheuttaa painovoiman pelaajaan.

- TimerEvent-metodi tarkistaa, onko pelaaja kävelemässä eri suuntiin. Jos on, niin se liikuttaa pelajaa ”speed” muuttujan verran, ja kutsuu metodia ”updateSprite”, joka animoi pelaajan liikeen.
- TimerEvent-metodi tarkistaa, onko pelaaja kaivamassa johonkin suuntaan. Jos on, niin se kutsuu metodia ”updateSprite” ja ”updateDiggingSprite”. Metodi ”updateDiggingSprite” animoi mineraalien ”rikkoutumisen” niitä kaivettaessa nopeudella, joka on muuttujassa ”drill”.
- TimerEvent-metodi tarkistaa, onko pelaaja lentämässä. Jos on, niin se liikuttaa pelaajaa ylöspäin muuttujan ”fan_speed” verran.
- TimerEvent-metodi päivittää HUDin.

Algoritmit

Kartanluomisgeneraattori luo rivit eräänlaisina lohkoina siten, että yksi rivi koostuu kolmesta erilaisesta lohkosta. Aivan aluksi se generoi kymmenen saman levyistä a-riviä. Tämän jälkeen se alkaa luomaan satunnaisesti ”x” -ja ”s”-tyypin tiilejä. Ensimmäinen lohko alkaa aina ”x”-tyypillä, joka vastaa siis läpäisemätöntä seinää. Lohkon keskellä on jokin mielivaltainen pätkä 4:n ja lohkon leveyden – 1:n verran läpäistävää seinää. Läpäisemätöntä seinää generoidaan sen verran, mitä lohkon leveydestä jää jäljelle. Keskimmäisellä lohkolla ei tarvitse olla ”x”-tyyppiä reunoilla. Täten sen generoituminen on hieman vapaampaa, kuin reunalohkojen. Viimeisen lohkon generoituminen on taas vastaavanlainen kuin ensimmäisen. Nyt on vain huolehdittava, että oikeaan reunaan generoituu aina ”x”-tyypin tiili.

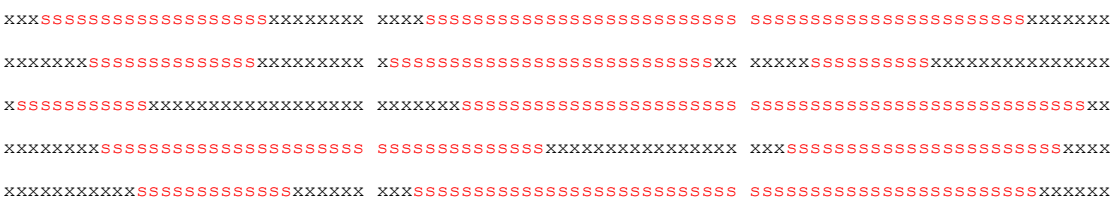
Kun rivi on saatu generoitua, tarkistetaan, että viimeksi luodun rivin aukoista on mahdollista kulkea läpi seuraavaan rivin aukkoihin. Sellaiset rivit, jotka eivät täytä tätä ehtoa hylätään. Tällöin generaattori yrittää löytää uutta ratkaisua niin kauan, että se löytää sellaisen rivin, joka täyttää tämän ehdon. Lopulta kun generaattori on saanut tehtyä peruskartaston pelkillä ”x” -ja ”s”-tyypeillä, se alkaa lisäämään karttaan satunnaisesti muitakin mineraaleja, ja voittoon vaadittavan kultaisen luun.

Generaattori alkaa käymään luotua karttaa uudestaan läpi rivi riviltä, ja lisää sinne mineraaleja satunnaisuuteen ja syvyyteen perustuen. Ensimmäisellä tasolla, joka vastaa kartassa 0-tasoa, ei esiinny vielä muita kuin ”s”-tyypin mineraaleja. Tämän rivin jälkeen listaan ”pool”, aletaan lisäämään erilaisia tyyppejä perustuen syvyyteen, jolla ollaan. Ensimmäisellä rivillä listaan lisätään ”b”-tyypin mineraali, joka vastaa siis pronssia. Tämä mineraali sijoitetaan satunnaiseen paikkaan kyseisellä rivillä. Jokaisella rivillä siis lisätään kenttään yksi mineraali ”pool”-listasta. Säännöllisin väliajoin listaan lisätään lisää muita tyyppejä, ja täten kenttään tulee lisää satunnaisuutta. Harvinaisempia ja arvokkaampia mineraaleja lisätään listaan vasta alemmilla tasoilla. Täten niitä ei voi siis esiintyä heti alussa. Voiton tuova mineraali ”w”, lisätään karttaan viimeistään rivillä 130, ja aikaisintaan rivillä 100.

Tässä on esimerkki kartan rakenteesta ilman a-rivejä tai muita kuin s-mineraaleja. Lohkot on jaoteltu erikseen havainnollistamisen parantamiseksi. Tästä pystyy huomaaman hyvin, kuinka ”aukot”, jotka on väritetty punaisella, generoituvat kenttään.

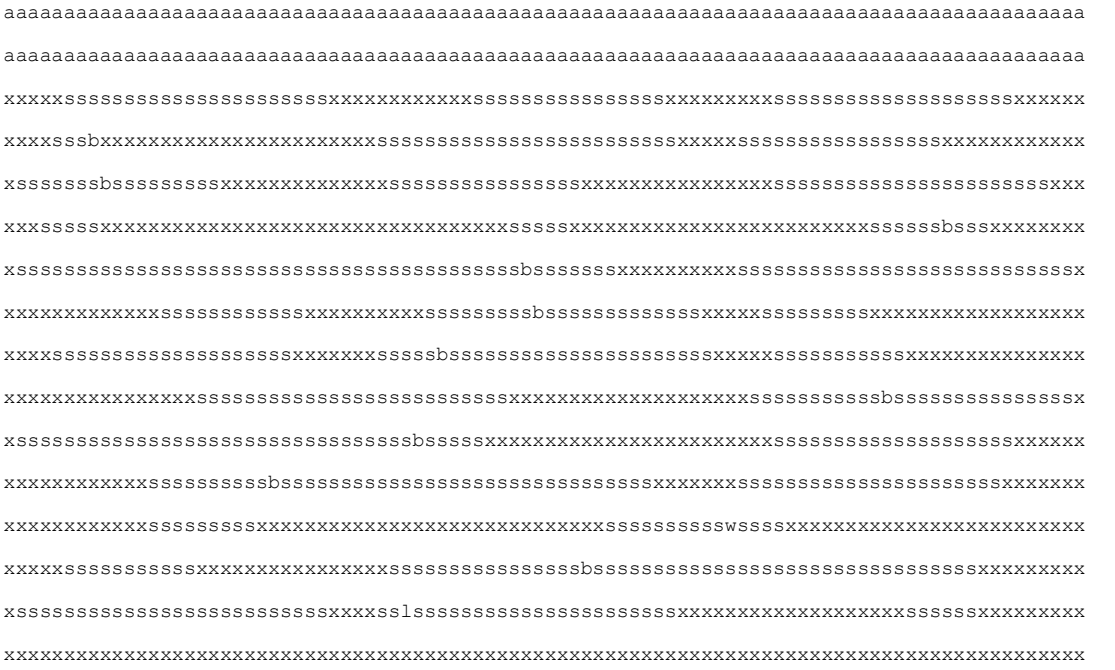
1. lohko 2. lohko 3. lohko

```
xxxxxxxxxxxxxxxxxxxxxxxxssssx xxxxssssssssssxxxxxxxxxxxxxxx xxsssssssssssssssssssxxxxxx
xsssssssssssssssssssssssxxx xsssssssssssssssssssssxxx xxxxxxxxxxxxxxssssssssxxxxxxxx
xsssssssssssssssssssssssss sssssssssssssssssssssxxx xxxsssssssssssssssssxxxxx
xxxsssssssssxxxxxxxxxxxxxxx xsssssssssssssssssssssss sssssssssssssssxxxxxxxxxxx
xsssssssssssssssssssssssss sssssssssssssssxxxxxxxxx xxxxxxxxxxssssssssssssssssx
```



Tietorakenteet

Pelin kenttä on tallennettu tekstitiedostona, jossa jokaisella merkillä on merkityksensä. Jokainen kirjain vastaa jotain tiettyä Tile-luokan instanssia, jolla on jokin tietty tekstuuri.



```
sand = s
bronze = b
silver = l
gold = g
ruby = r
diamond = d
opal = o
```

Pythonin omista tietorakenteista, käytin projektissa pääosin list-rakennetta. List on muuttuva rakenne, eli sen alkioita voi muokata tai korvata, ja listan järjestystä voidaan muuttaa. List on myös dynaaminen rakenne sillä siihen voi lisätä ja poistaa alkioita mielivaltaisesti, jolloin list suurenee ja pienenee automaattisesti.

Tiedostot

Pelin animaatiot perustuvat ”sprite”-kuviin, joita näytetään nopeasti peräkkäin, jolloin saadaan animoitua elävää liikettä. Pelin liikkumiset ja kaivamiset on tehty kyseisellä tavalla. Piirsin itse peliin kaikki tekstuurit ja ”sprite”-kuvat. Kuvat ovat tallennettuna jpg-tiedostoina. Koko pelin liikkuminen perustuu tiilijärjestelmään, jossa pelaaja voi kaivaa aina vain yhden kokonaisen tiilen kerrallaan.

Tämän vuoksi, näin järkevimpänä tehdä kaikki kuvat samassa koossa. Kaikki kuvat ovat siis kooltaan 20x20 px.

Kuvien lisäksi pelissä on musiikkia ja äänitiedostoja. Lyhyet ääniefektit ovat tallennettuna wav-muodossa. Musiikit taas ovat mp3-muodossa. Ääniefektit ja osan musiikeista olen ladannut Mixkit ja Zapsplat palveluista. Ne tarjoavat ilmaisia rojaltivapaita musiikki- ja äänitiedostoja ei-kaupallisiin projekteihin.

Testaus

Projektissani tärkeä testauskohde on pääkentän oikeellisuus. Ohjelman täytyy pystyä generoimaan validi kenttä, jotta pelaaminen on mahdollista. Yksikkötestejä en kerennyt tekemään kentälle, vaikka tarkoituksenani olikin. Tarkoituksenani oli testata, toteuttaako luotu kenttä sille asetetut vaatimukset. Näihin kuuluu muun muassa:

- Maan generointi alkaa 0-tasosta
- Aukon on oltava vähintään 4 tiiltä leveä
- Yhdellä rivillä voi olla maksimissaan 3 aukkoa, vähintään 2
- Seuraavan rivin generoiminen tapahtuu sellaiseen kohtaan, josta on pääsy ylemmän rivin aukkoon
- Yhdelle riville generoituu vain yksi harvinaisempi mineraali
- Kultainen luu on rivien 100 ja 130 välissä
- Seinillä ja lattialla on aina ”x”-tyypin tiili

Lisäksi tarkoituksenani oli testata, lataako peli tarvittavat tekstuurit ja äänitiedostot oikein.

Ohjelman tunnetut puutteet ja viat

Ohjelma on jokseenkin sekava ja siinä on hyvin paljon ehtolauseita ja toistoja. Olen varma, että huolellisella tarkastelulla, ohjelmaa saisi varmasti pienenettyä korjaamalla tarkistamista. Tämän lisäksi ohjelmassa on hyvin paljon asioita yhdessä luokassa, eikä niitä ole jaoteltu. Tämän voisi korjata myös huolellisella tarkastelulla, katsomalla esiintykö toistoja, ja korvaamalla niitä funktioilla, ja jakamalla omat isommat kokonaisuudet omiksi luokiksi.

Ohjelmaa ei myöskään ole yksikkötestattu. Tosin projektin aikana on jatkuvasti tarkistettu, että lisätyt ominaisuudet eivät ole ristiriidassa aiempien ominaisuuksien kanssa, eivätkä esimerkiksi aiheuta ohjelman kaatumista.

Kaivamisessa esiintyy jonkunlainen bugi, jota en saanut korjattua. Hahmo pysäyttää kaivamisen satunnaisesti mineraaleja kaivaessa. Tähän en löytänyt korjausta, sillä en löytänyt mistä tuo satunnaisuus johtuu.

Ohjelma on myös verrattain raskas ajaa. Esimerkiksi HUDin toteuttaminen suuremmilla fonteilla saa aikaan ohjelman pätkimisen. Lisäksi ääniefektien toistaminen ja äkillinen pysäyttäminen saa aikaan pätkimistä ohjelmassa. En ole varma, johtuuko tämä Qt5:n ja Pythonin metodien hitaudesta, vai omasta hitaasta ja resursseja tuhlaavasta toteutuksesta. Optimoimalla ohjelmaani tämä asia selvenisi.

Ohjelmassa ei hyödynnetä PyQt:n omia mekanismeja hirveästi. Esimerkiksi hahmon animointi tehdään yksinkertaisesti sen kuvaa vaihtamalla hyvin nopeasti. Qt tarjoaa kuitenkin animoimiseen tarkoitettuja työkaluja, mutta en saanut niitä itse toimimaan. Lisäksi Qt tarjoaa myös objektien törmäysten tunnistamiseen työkaluja, joita käytinkin osassa tilanteista, mutta esimerkiksi hahmon törmäyksen tunnistamisessa seiniin, tätä työkalua ei käytetty.

Monet asiat on toteutettu ensimmäisellä mieleen tulleella metodilla, kuten esimerkiksi juuri törmäysten tunnistaminen. Näihin olisi varmasti voinut keksiä nerokkaampia ratkaisuja, mutta ajasta johtuen jätin nämä ratkaisut lopulliseen työhön.

Hahmon liikkumisen animointi on resursseja kuluttavaa, sillä se lataa ne tiedostoista, aina kun hahmo liikkuu. Tämän voisi helposti optimoida lataamaan kuvat vain kerran pelin alussa, ja käyttää niitä sitten ohjelman sisällä aina uudelleen, tiedostosta uudelleen lataamisen sijasta.

Parhaimmat ja heikoimmat kohdat

Kenttägeneraattori, oma hauska toteutus, jonka avulla kenttää voi muokata helposti vaihtamalla vain parametreja.

Grafiikat ja ulkoasu. Vaikuttaa paljon pelin ulkoasuun ja pelattavuuteen. Itse tehdyt kuvat.

Epämääräinen jaottelu. Luokkien jako epäselvää, ja Player-luokalla liikaa vastuuta pelin organisoimisesta

Toisteisuus, raskas, turhaa toistoa. Paljon ehtolauseita, joista osa varmasti turhia.

Monet asiat toteutettu hölmösti, kamera, törmäyksen tunnistus, HUD. Kamera keskittää kuvan pelaajaan aina, vaikka pelaaja ei liikkuisi pikseliä. Kamera ei ole kiinteästi yhteydessä pelaajaan vaan joutuu jatkuvasti hakemaan pelaajan sijainnin ja päivittymään sen perusteella. Törmäyksentunnistuksessa suurimmaksi osaksi ei käytetä hyväksi jo valmiita metodeja. HUD on raskaasti tehty, ja aiheuttaa hidastumista pelissä, jos fonttia kasvattaa. Lisäksi se ei ole kovakoodattu näkymään, vaan joutuu koko ajan vaihtamaan sijaintiaan. Tämän vuoksi joudun koko ajan poistamaan vanhan HUDin ja lisäämän uuden HUDin peliin. Tämä on se syy, miksi varmasti fontin kasvattaminen aiheuttaa pätkimistä.

Painovoiman toteutus on muuten hyvä, mutta se ei vaikuta pelaajan, kun se lentää.

Liikkumisen animointi. Ei hyödynnetä PyQt:n omia metodeja.

Liikkumisen toteutus. Peliin olisi voinut toteuttaa jonkinlainen kiihtyvyys liikkeelle lähdettäessä ja pysähtyessä. Nyt liikkuminen tapahtuu siten, että pelaaja lähtee liikkumaan aina vakionopeudella, ja pysähtyy välittömästä liikkumisen loputtua. Tämän pienen lisän toteuttaminen olisi varmasti vaikuttanut pelaajan liikkuvuuteen ja sen sulavuuteen paljon, mikä olisi tehnyt kokonaisuudessaan pelin pelaamisesta hauskempaa.

Poikkeamat suunnitelmasta

Yllättävän kiireisestä loppukeväästä johtuen, oli pitkä jakso, jolloin en kerennyt tätä projektia tekemään. Onneksi aloin heti aikaisin keväällä tekemään tätä projektia, jolloin olin saanut jo tehtyä osan suunnitelluista asioista heti kättelyssä. Kuitenkin projektin edetessä pelimekaanikoiden tekemisessä meni yllättävän kauan toteuttaa, jolloin minun oli pakko jättää osa suunnitelluista ominaisuuksista toteuttamatta. Näihin lukeutuu pelin tallentaminen, päävalikko ja jonkinlaisen kaupan toteuttaminen.

Ajankäyttöarvioni siitä, kuinka paljon aikaa kukin ominaisuus vie aikaa, ei mennyt ihan nappiin, ja sen lisäksi asiat kasaantuivat ulkoisista syistä projektin alku- ja loppupäihin.

Toteutunut työjärjestys ja aikataulu

Heti projektin alussa aloin suunnittelun lisäksi etsimään tietoa PyQt5-kirjastosta. Projektisuunnitelman jälkeen, loin kenttägeneraattorin. Tämän jälkeen toteutin pelin grafiikat hyvin yksinkertaisilla kuvilla perustuen kenttägeneraattoriin. Tämän lisäksi toteutin ohjelmalle mahdollisuuden lukea käyttäjältä syötettä, ja ohjelman toimimaan käyttäjän syötteen perusteella. Nämä asiat toteutin heti helmikuun lopussa ja maaliskuun alussa. Tämän jälkeen projektin toteuttamisessa tuli tauko. Jatkoin projektia taas huhtikuun lopussa.

Huhtikuussa muokkasin ohjelman generoimisen rakennetta siten, että loin Tile-luokan, joka luo jokaiselle tiilelle oman olionsa. Tämän jälkeen aloin tekemään pelaajan törmäyksen tunnistamista seiniin käyttäen apunaan Tile-luokan objekteja. Tein paljon yleisiä muokkauksia pelin pelattavuuden kannalta. Toteutin kameran, joka seuraa pelaajaa ja keskittää näkymän aina pelaajaan, pelaajan liikkuessa. Sen lisäksi toteutin pelaajan kaivamisen ja yleisen painovoiman, joka vaikuttaa karkeasti aina silloin kun pelaajan alla ei ole maata. Lisäsin peliin myös hahmon hyppäämisen, mutta poistin sen sen jälkeen kun, olin toteuttanut lentämisen. Näin, että ei ollut tarvetta sisällyttää kahta erilaista tapaa liikkua ylöspäin. Näiden lisäksi muokkasin kenttägeneraattoria siten, että se generoi kenttään myös harvinaisempia mineraaleja.

Toukokuussa aloin luomaan liikkumisille ja kaivamisille animaatioita. Piirsin tekstuurit näille ja lisäsin ne peliin. Tein ja latasin äänitiedostot, ja linkitin ne myös peliin. Tein myös paljon viime hetken muokkailuja ja siivoamista luokkiin. Lopuksi tein projektin dokumentaation.

SUUNITELMA	TOTEUTUNUT	
Projektisuunnitelma, 10 h	Projektisuunnitelma, kenttägeneraattori 14 h	vko 8
Kenttägeneraattori, 10 h	Käyttäjän syötteen lukeminen ja reagoiminen (liikkuminen), 18 h	vko 9
Peliluokka, 20 h		vko 10
Muut luokat, käyttöliittymä, 10 h		vko 11
Muut luokat, käyttöliittymä, 10 h		vko 12
Tallennus, 15 h		vko 13
Tekstuurit, 7.5 h		vko 14
Tekstuurit, äänitiedostot, 7.5 h		vko 15
Äänitiedostot, testaamista, 5 h	Tile-luokan tekeminen, törmäyksentunnistaminen, kameran seuraaminen pelaajaa, kaivaminen, 35 h	vko 16
Testaamista, 5 h	Törmäyksentunnistaminen, liikkumisen tekeminen uudelleen, kaivaminen, painovoima, kenttägeneraattorin muokkaaminen, tekstuurien piirtämistä ja animaatioiden tekemistä, 50 h	vko 17
Projekti valmis, kok. aika = 100 h	Tekstuurien piirtämistä, animaatioiden tekemistä, äänien tekeminen ja lisäys, voitto/häviö -ehtojen tekeminen, luokkien siivous ja yleinen parantelu, dokumentaation tekeminen, 45 h Projektin kokonaisaika = 162 h	vko 18

Käytin siis projektiin huomattavasti enemmän aikaa, kuin olin alun perin suunnittelut. Tästä huolimatta, en edes saanut kaikki ominaisuuksia toteutettua. Kuitenkin luulisin, että tallennusominaisuuden tekeminen ei veisi hirveästi aikaa. Myöskään päävalikon tekeminen ei olisi kovin aikaa vievää. Kauppaominaisuuden tekemisessä ei kovin kauaa menisi, jos sen tekisi vain tarpeeksi yksinkertaisesti. Pelin mekaanikoiden tekeminen oli yllättävän aikaa vievää, ja olisin halunnut tehdä liikkumisesta vieläkin sulavampaa. Ajanpuutteen vuoksi jouduin kuitenkin

luopumaan edellä mainituista ominaisuuksista. Suunnitelmanani on, että jatkan projektin tekemistä vapaa-ajallani, ja teen yllä mainitut ominaisuudet, jotta saan pelin hyvään päätökseen.

Arvio lopputuloksesta

Mielestäni ohjelman lopputulos on hyvä, vaikka siinä paljon puutteita onkin. Opin paljon Qt5 kirjastosta, ja samalla jouduin palauttelemaan mieliin C++ kieltä, sillä luin pääosin Qt5:n dokumentaatiota, joka on C++ kielellä kirjoitettu. Lisäksi ohjelmointitaitoni kehittyivät, ja huomasin, että en enää tarvitse jatkuvasti tietolähteitä saataville, jotta voin ohjelmoida. Tuntuu, että olen ylittänyt jonkinlaisen kriittisen rajan, jonka johdosta osaan soveltaa ja käyttää perusasioita monipuolisesti, jolloin pystyn yhtäjaksoisesti keskittymään pelkästään ohjelmoimiseen ja tekemiseen, tiedonhaun sijaan. Toki jouduin paljon etsimään jatkuvasti tietoa PyQt5:stä, mutta muuten, jouduin hyvin vähän etsimään tietoa itse Pythonin ominaisuuksista. Tämä oli todella mukava huomata, ja se motivoi paljon.

Parasta projektin tekemisessä oli se, että sain itse päättää kaikesta, ja toteuttaa asiat omalla tavalla. Vaikka monet kohdat on melko suoraviivaisesti toteutettu, on niissä kuitenkin joutunut miettimään etukäteen, kuinka jonkin asian toteuttaa. Esimerkiksi kenttägeneraattorin tekeminen oli hauskaa, kun huomasi, että voin parametreilla helposti esimerkiksi muuttamaan kentän syvyyttä, luoden vaikka 1000 riviä syvän kentän. Myös painovoiman ideoiminen oli mukavaa, ja sen näkeminen toiminnassa sai minussa aikaan saavan tunteen.

Luokkajaon tekisin uudelleen, ja miettisin, miten saisin suuren osan toiminnoista siirrettyä Player-luokasta Game-luokkaan. Tällä hetkellä Game-luokka vain kutsuu kenttägeneraattori ja piirtää kentän, ja Player-luokka tekee kaiken pelin aikaisen toiminnon.

Suurimmat haasteet koin PyQt5:n kirjaston soveltamisesta pelien tekemiseen. Grafiikkakirjastolla, joka on tarkoitettu lähinnä käyttöliittyminen tekemiseen, oli haastava toteuttaa graafinen tasohyppelypeli, jossa on paljon toteutettavaa. Vaikeinta oli se, että en löytänyt lähes ollenkaan oppaita tai minkäänlaisia ohjeita moniin asioihin, mitä yleensä peleissä on. Esimerkiksi animaatioiden ja pelaajan liikuttamisen tekemiseen, en löytänyt juurikaan esimerkkejä. Tiedonhaku vei todella paljon aikaa, ja oli turhauttavaa, kun hakukoneet antavat vastaavia esimerkkejä ongelmiini toisilla kirjastoilla, jotka ovat pelien tekemiseen tarkoitettujakin. Mietinkin monesti projektin aikana, että olisi pitänyt tutustua huolellisemmin Qt5 kirjastoon, ja siihen minkälaisiin peleihin se voisi soveltua. Jos nyt tekisin uudelleen tämän projektin, valitsisin jonkinlaisen kevyemmän pelin, jossa kamera pysyy paikallaan, tai jonkunlaisen graafisen järjestelmän toteuttamisen tasohyppelypelin sijaan.

Kirjallisuuslähteet ja linkit

https://docs.python.org/3/tutorial/datastructures.html Käytetty tarkistamaan Pythonin valmiiden tietorakenteiden ominaisuuksia

https://www.riverbankcomputing.com/static/Docs/PyQt5/ PyQt:n luojan RiverBank Computing Limitedin oma virallinen ohjekirja PyQt5:n käyttöön

https://wiki.python.org/moin/PyQt/Tutorials PyQt tutoriaaleja

https://www.piskelapp.com/ Piskel, ilmainen tekstuurinteko-ohjelma

https://www.gimp.org/ Gimp, ilmainen kuvankäsittelyohjelma

https://doc.qt.io/qt-5/ Qt5 dokumentaatio

https://stackoverflow.com/ Ohjelmointiin liittyvä keskustelupalsta

https://forum.qt.io/ Qt-ohjelmointiin liittyvä keskustelupalsta

https://build-system.fman.io/pyqt5-book PyQt5 Book 2021, Python and Qt, the best parts, Michael Herrmann, 2021

https://mixkit.co/ Ilmaisia äänitiedostoja

https://www.zapsplat.com/ Ilmaisia äänitiedostoja

Liitteet