Projektidokumentaatio 2D-Platformer-Shooter

Valtteri Valtonen 716462

Tietotekniikan koulutusohjelma

Vuosikurssi 1

19.4.2019

Yleiskuvaus

Tässä projektissa parantelin joulun aikana aloittamaani 2D-Platformer-Shooter tyyppistä peliä. Projekti alkoi pisteestä, jossa pelaaja kykeni liikuttamaan ruudulla näkyvää animoitua hahmoa pelin maailmassa, käyttämään kahta erityiskykyä (suojakenttä ja ajan hidastaminen) ja taistelemaan yksinkertaisia vihollisia vastaan(valmiin projektin shooterEnemy). Peli sisälsi myös hieman musiikkia.

Projektini ensimmäinen osa oli esinejärjestelmän toteuttaminen peliin. Esinejärjestelmän myötä pelin maailmaan ilmestyi kahdentyyppisiä esineitä: hyötytavaroita ja aseita, joita pelaaja saattaa poimia ja käyttää halutessaan. Esineitä on pelin maailmassa vapaina, mutta myös viholliset saattavat pudottaa niitä tuhoutuessaan. Toteutin kaksi erilaista asetta(slow firing weapon ja rapid fire weapon) ääniefekteineen ja kaksi hyötytavaraa(energy pack ja health pack). Uusien esineiden toteuttaminen on helppoa: tarvitsee vain luoda luokka, joka perii joko Weapon-piirreluokan tai UtilityItem-piirreluokan.

Projektin toinen osa oli pelinaikaisen pelaajan hudin rakentaminen. Pelinäkymän vasemmassa yläkulmassa näkyy asevalikko ja hyötytavaralaatikko, johon pelaajan poimimat esineet ilmestyvät ja joiden avulla pelaaja voi valita esineiden välillä. Nämä kaksi elementtiä hyödyntävät laatimaani itemBox-luokkaa. Pelinäkymän vasemmassa reunassa ovat myös elämä- ja energiapalkit, jotka kertovat pelaajalle hahmonsa tilasta. Palkkien perustana on scalafx:n oma GameBar-luokka. Palkkien ulkonäköä on muutettu CSS:ää hyödyntäen. Pelaajan HUDiin kuuluu myös pelinäkymän oikeassa yläkulmassa oleva viestialue, joka kertoo pelaajalle pelinaikaisista tapahtumista.

Projektin kolmas osa oli pelin valikoiden luominen. Rakensin peliin päävalikon, tasovalikon, asetusvalikon, pause-valikon, kuolemavalikon, tallennusvalikon ja pelin latausvalikon. Valikot sisältävät otsikon, taustakuvan ja erilaisia nappeja. Tasovalikon erikoisuutena on tikapuuosa, jota voi hiiren rullalla liikuttaa pystysuunnassa. Asetusvalikkon kautta voi muuttaa pelin äänten volyymiä ja laittaa päälle kehittäjätilan. Pelin napit käyttävät laatimaani AnimatedButton-luokkaa. Napin toiminnot määritellään antamalla nappia painettaessa suoritettava koodi parametrina luokan instanssille.

Projektin neljäs osa oli pelin tallennusjärjestelmän laatiminen. Pelaaja voi tallennusvalikon kautta tallentaa pelin yhteen neljästä tallennuspaikasta ja ladata pelin vastaavasta paikasta latausvalikosta. Tallennustiedostojen formaatti on laatimani DWAsave. Tallennustiedostot säilyttävät pelaajan, vihollisten ja esineiden sijainnit pelin maailmassa, pelaajan ja vihollisten elämä- ja energiapisteet, pelaajan tavaraluettelon, tasojen avoimuustilanteen ja viimeisimmän tason nimen.

Projektin viides osa piti sisällään pelin kameran toimintojen parantelua. Lisäsin peliin ns. “karttatilan”. Kun peli on karttatilassa, pelaaja voi liikuttaa kameraa vapaasti, sekä zoomata sisäänpäin ja ulospäin.

Projektin kuudentena ja viimeisenä osana oli uuden pelaajaa seuraavan vihollistyypin luonti. Uusi vihollinen liikkuu maata pitkin kohti pelaajaa ja hyökkää ampumalla säännöllisin väliajoin. Vihollinen puolustautuu pelaajan ammuksilta käyttämällä vastaavanlaista suojakenttää kuin pelaaja. Jos pelaaja kiipeää tikkaita vihollisen ollessa lähellä, vihollinen kiipeää pelaajan perään. Ulkonäöltään uusi vihollinen muistuttaa pelaajahahmoa. Sillä on myös vastaavat animaatiot kuin pelaajalla.

Yllä lueteltujen varsinaisten osien lisäksi projektiin liittyi runsaasti vanhan koodin parantelua. Ehkä laajin parannus oli täydellinen pelin sijaintijärjestelmän uudistus. Ennen uudistusta pelin “loogista” koordinaatistoa ja pelin kuvan koordinaatistoa ei oltu erotettu toisistaan. Uudistuksen jälkeen nämä ovat selkeästi erillään toisistaan. Uudistus selkeytti pelin toimintaa huomattavasti, ja mahdollisti koko näytön tilan sekä karttatilan toteuttamsen.

Mielestäni projekti on toteutettu vaikealla vaikeustasolla.

Käyttöohje

Peli käynnistetään tiedoston GUI.scala kautta. Näytölle pitäisi avautua ikkuna, joka sisältää pelin päävalikon. Ikkunan kokoa voi muuttaa vapaasti ja myös koko näytön tila on mahdollinen, tosin se heikentää hieman pelin suorituskykyä.

Päävalikosta voi siirtyä muihin valikoihin painamalla vastaavaa nappia hiiren vasemmalla näppäimellä. Play game-napista peli siirtyy varsinaiseen pelinäkymään.

Pelinäkymässä kontrollit ovat seuraavat:

w	hyppää/liiku ylös tikkaissa
s	liiku alas tikkaissa
a	liiku vasemmalle
d	liiku oikealle
q	hidastetun ajan tila
e	suojakenttä
hiiren vasen näppäin	ammu
hiiren rulla / numerot 1-5	valitse ase
x	valitse edellinen hyötytavara
c	valitse seuraava hyötytavara
h	käytä hyötytavaraa
m	karttatila
esc	pause-valikko
o	suorita yksikkötestit

Karttatilassa:

i	liikuta kameraa ylöspäin
k	liikuta kameraa alaspäin

j	liikuta kameraa vasemmalle
l	liikuta kameraa oikealle
nuoli ylös	zoomaa sisään
nuoli alas	zoomaa ulos
m	poistu karttatilasta

Pelin tallentaminen onnistuu siirtymällä pause-valikosta pelin päävalikkoon ja valitsemalla sieltä tallennusvalikko.

Peli ladataan valitsemalla latausvalikosta haluttu tallennuspaikka. Lataukseen kuluu pieni hetki. Kun peli on ladattu, latausvalikosta voi siirtyä takaisin päävälikkoon ja siitä peliin.

Tason läpäisy onnistuu saavuttamalla maali, joka on merkitty keltamustaraidallisin tiilin. Tasossa numero 1 maali löytyy pelaajasta katsottuna alaoikealta. Tehtävää helpottaa asetusvalikon kautta käynnistettävä kehittäjätila, joka tarjoaa loputtomat elämä- ja energiapisteet ja hypyt. Kun pelaaja pääsee maaliin, peli siirtyy tasovalikkoon, jossa taso numero 2 on valittavissa. Tasoa 2 ei vielä voi läpäistä koska kolmatta tasoa ei ole.

Ohjelman rakenne

Pelin rakenne on jonkin verran erilainen suunnitelmaan verrattuna. Ohjelman tärkeimmät osakokonaisuudet ovat varsinainen peli, pelin käyttöliittymä ja tallennustiedostoja lukeva ja kirjoittava osa. Projektin luokkarakenne on keskeisimpien luokkien osalta seuraava:

Esinejärjestelmä:

abstract class Item Keskeiset metodit: ID, locationForSprite Metodi ID kertoo esineen tunnuksen jota hyödynnetään tallennuksessa. LocationForSprite kertoo GameSprite-luokalle mihin esineen kuva tulee sijoittaa.	**abstract class Weapon** Keskeiset metodit: ei ole Tämän luokan keskeiset metodit on toteutettu perivissä luokissa	**abstract class UtilityItem** Keskeiset metodit: isSpent Luokka perii Itemin. Metodi isSpent kertoo onko esine kulutettu loppuun.
abstract class HealthPack **abstract class EnergyPack** Keskeiset metodit: use Metodi use kasvattaa pelaajan elinvoimaa/energiaa ja vähentää omaa käyttökertojen määrää yhdellä. Luokkien ja pelaajan välillä on siis viittaussuhde.	**abstract class RapidFireWeapon** Keskeiset metodit: fire Fire-metodi laukaisee aseen. Se mitä tämä tarkoittaa käytännössä vaihtelee aseen mukaan. Aseiden toiminta pelissä riippuu projectile-luokasta. Fire-metodi antaa ammukselle suunnan käyttämällä DirectionVector-luokkaa. Perii Weaponin.	**abstract class SlowFiringWeapon** Keskeiset metodit: fire Fire-metodi toimii kuten RapidFireWeaponin tapauksessa. Perii Weaponin.

Käyttöliittymä:

Käyttöliittymä vastaa suurelta osin suunnitelmaa. Valmiissa projektissa on lisänä NotificationArea. Keskeisimmät luokat ovat seuraavat:

class GameBar Keskeiset metodit: setValue Metodi setValue käyttää valmiin ProgressBar-luokan metodia setProgress, joka säätelee palkin pituutta.	**class ItemBox** Keskeiset metodit: select, deselect, insertItem, removeItem Metodit vaikuttavat esinelaatikon tilaan. WeaponHud käyttää niitä.
object WeaponHud Keskeiset metodit: updateItems, createHud Metodi updateItems käy pelaajan tavaraluettelon läpi ja sijoittaa esineet laatikoihin. CreateHud luo asevalikon laatikot pelin käynnistyksen yhteydessä asettamalla niitä vierekkäin location-muuttujan sijainnista lähtien.	**object NotificationArea** Keskeiset metodit: announce Metodi announce näyttää viestialueella parametrina annetun viestin.

class AnimatedButton Keskeiset metodit: refreshLocation Metodi refreshLocation pitää napin paikoillaan ikkunan koon muuttuessa Napilla on ScalaFx:n mukaiset tapahtumankuuntelijat ja käsittelijät jotka pitävät huolen siitä mitä esimerkiksi napin painamisesta seuraa. Napin toiminto annetaan luokalle parametrina.

Muita käyttöliittymään liityviä luokkia ovat esimerkiksi täysin oma GameCheckBox ja scalafx:n ProgressBariin perustuva GameSlider.

Tallennustiedostoja lukeva ja kirjoittava osa

Tämä osa vastaa suunnitelmaa.

object SaveHandler
Keskeiset metodit: loadGame, saveGame Metodi loadGame lukee tietoa tallennustiedostosta scala.io:n työkalujen avulla ja luo uuden peliolion. Metodi saveGame puolestaan kirjoittaa uuden tallennustiedoston peliolion perusteella. Tallennettavaa tietoa ovat pelaajalla hallussaan olevat esineet, pelaajan ja vihollisten sijainnit ja tilat pelin maailmassa, kenttien avoimuustilanne ja viimeisimmän kentän nimi.

Muita keskeisiä luokkia:

class DirectionVector	**class GamePos**	**class GameSprite**
Keskeiset metodit: angle, opposite Metodi angle kertoo kulman, jossa vektori on. Sitä hyödynnetään esimerkiksi pelaajan käden kiertokulman määrittämiseen. DirectionVector määrää ammuksen suunnan. Metodi opposite kääntää suunnan päinvastaiseksi, kun ammus havaitsee törmäävänsä suojakenttään.	Keskeiset metodit: locationInImage, move Metodi LocationInImage muuntaa “loogisen” koordinaatiston koordinaatin kuvakoordinaatiston koordinaatiksi”. Metodi move siirtää sijaintia. Pelaaja, viholliset, ammukset ja kamera käyttävät GamePos-tyyppistä sijaintia.	Keskeiset metodit: image Image-metodi palauttaa luokan instanssin kuvan. Se valitsee peilikuvan tai tavallisen kuvan välillä. Hieman vastaava AnimatedGameSprite näyttää useita kuvia peräkkäin ja huolehtii niin animaatiosta.

Algoritmit

Peli käyttää muutamaa erilaista algoritmia, jotka ovat keskeisiä sen toimivuuden kannalta. Näitä ovat etäisyyspohjainen törmäystunnistus, koordinaattien muunnos pelin “loogisesta” koordinaatistosta kuvan koordinaatistoon ja pelaajaa seuraavan vihollisen “tekoäly”.

Etäisyyspohjainen törmäystunnistus

Törmäysten tunnistaminen on tärkeää pelin toimivuuden kannalta. Törmäyksien tunnistamisesta huolehtivat pelaajaan ja vihollisiin liitetyt neljä “puskuria”. Puskureita on yksi kullekin ilmansuunnalle ja ne koostuvat sijaintipisteistä, joiden suhteen törmäyksiä tarkkaillaan. Törmäyksien tunnistamisessa on seuraavat vaiheet:

1. Suodatetaan pelin nykyisen kentän kaikista tiilistä sellaiset, jotka ovat tietyn rajaetäisyyden sisäpuolella puskurista katsottuna. Etäisyydet puskurin ja tiilten välillä määritetään pythagoraan lauseella.
2. Tarkistetaan, löytyykö ensimmäisessä vaiheessa saadusta tiilten listasta tiiltä, jonka etäisyys puskurin mistä tahansa sijaintipisteestä on pienempi kuin törmäysetäisyys. Etäisyyksien laskenta tapahtuu tässä vaiheessa erikseen x-suunnassa ja y-suunnassa yksinkertaisten vähennyslaskujen avulla.
3. Jos toisessa vaiheessa löytyy tiili, puskuri havaitsee törmäyksen. Tiedon perusteella voidaan esim. pysäyttää pelaajahahmon liike kun se kohtaa lattian.

Algoritmin voisi myös toteuttaa siten että kentän tiilet jaettaisiin jo kentän luomisvaiheessa koordinaattiensa perusteella eri lohkoihin. Tällöin yllä kuvatun ensimmäisen vaiheen raskaan etäisyyden laskennan sijaan voitaisiin vain nopeasti tarkistaa missä lohkossa pelaaja kullakin hetkellä on ja siirtyä toiseen vaiheeseen käyttäen senhetkisen lohkon tiiliä. Toteutin lohkoja hyödyntävän järjestelmän projektin loppumetreillä, mutta se toi mukanaan ongelmia joiden korjaamiseen ei enää ollut aikaa. Valitsin projektissa käytetyn mallin siksi että se toimi varmasti ja tarpeeksi nopeasti.

Koordinaattimuunnos

Pelissä on kaksi koordinaatistoa: “looginen” koordinaatisto jossa maailma säilyy paikoillaan ja pelaaja liikkuu, sekä kuvan koordinaatisto jossa pelaaja pysyy paikoillaan

pelin kuvan keskellä ja maailma liikkuu. Pelin sisäisen logiikan toiminnot kuten esimerkiksi törmäyslaskenta tapahtuvat “loogisessa” koordinaatistossa. Kun pelin kuva piirretään näytölle, “loogiset” koordinaatit muunnetaan kuvan koordinaateiksi. Muunnoksessa on seuraavat vaiheet:

1. Tarkistetaan onko olio jolle muunnettavat koordinaatit kuuluvat tarkoitus pysyä näytön keskellä.
2. Jos olio on “keskipiste”, palautetaan kunkinhetkisen peliikkunan keskipiste jonka koordinaatit saadaan jakamalla ikkunan leveys ja korkeus kahdella.

   Jos olio ei ole keskipiste, kuvakoordinaatit lasketaan seuraavalla kaavalla:

   olion looginen koordinaatti - kameran looginen koordinaatti + kameran kuvakoordinaatti.

Peli toimisi myös vain yhden koordinaatiston avulla, mutta käyttämäni kahden koordinaatiston malli on selkeämpi ja eheämpi. Käytetty malli mahdollistaa myös paremmin esimerkiksi vapaan kameran tilan.

Pelaajaa seuraavan vihollisen tekoäly

Pelaajaa seuraavan vihollisen tekoälyyn liittyy kaksi algoritmia. Toinen selvittää pelaajan karkean suunnan viholliseen nähden ja toinen liikuttaa vihollista tähän suuntaan.

Pelaajan karkea suunta selvitetään yksinkertaisesti pelaajan ja vihollisen x- ja y-koordinaattien erotuksien merkin avulla. Suuntaan liittyy aina vaakasuuntainen ja pystysuuntainen komponentti. Mahdollisia vaakasuunnan komponentteja ovat “vasemmalla”, “oikealla” ja “lähellä”. Mahdollisia pystysuunnan komponentteja ovat “yläpuolella”, “alapuolella” ja “tasossa”. Pelaaja on “lähellä” tai “tasossa” kun koordinaattien etäisyys laskee tarpeeksi pieneksi. Vihollista liikuttava algoritmi toimii komponenttiparin perusteella. Mahdollinen pari olisi esimerkiksi (vasemmalla, yläpuolella).

Vihollinen liikkuu vaakasuunnassa joko vasemmalle tai oikealle saamansa vaakasuuntaisen komponentin mukaan. Jos vihollisen vasemmanpuoleinen tai oikeanpuoleinen “puskuri” havaitsee törmäyksen vihollinen hyppää. Jos pelaaja on vihollisen yläpuolella ja vihollinen on tikkaiden kohdalla vihollinen kiipeää tiikkaita pitkin kohti pelaajaa. Jos tikkaita ei ole saatavilla ja pelaaja on lähellä vihollinen hyppää.

Vihollinen on jumissa jos sen vasen puskuri, oikea puskuri ja jompikumpi ylä- tai alapuskureista havaitsevat törmäyksen samaan aikaan. Tällöin vihollisen sijaintia siirretään joko alaspäin tai ylöspäin siten että vihollinen selviää jumista.

Liikkuessaan kohti pelaajaa vihollinen ampuu tasaisin väliajoin kohti pelaajaa. Ampumissuunta määräytyy päivittämällä pelaajan ja vihollisen välistä suuntavektoria.

Vihollinen suojautuu pelaajan ammuksilta käyttämällä vastaavanlaista suojakenttää kuin pelaaja. Vihollinen käynnistää suojakentän jos se havaitsee pelin ammusten joukosta ammuksen, joka on tietyn rajaetäisyyden sisäpuolella.

Vihollisen tekoälyn olisi myös voinut toteuttaa esimerkiksi raycastingin avulla. Valitsin käyttämäni vaihtoehdon koska se toimi tarpeeksi tehokkaasti ja oli helpompi toteuttaa.

Tietorakenteet

Pelin käyttämät tietorakenteet ovat sekä muuttuvatilaisia että muuttumattomia scalan valmiita tietorakenteita. Muuttuvatilaisista tietorakenteista hyödynsin useimmiten ArrayBufferia ja muuttumattomista vektoria. Valitsin juuri nämä tietorakenteet, koska ne toimivat tarpeeksi tehokkaasti ja soveltuivat hyvin käyttötarkoituksiinsa. String-tyyppi oli myöshyödyllinen tietojen tallennukseen lukuisten metodiensa ansiosta.

Tiedostot ja verkossa oleva tieto

Peli käsittelee kuvatiedostoja ja DWAsave-tyyppisiä tallennustiedostoja. Kuvat antavat pelille ulkonäkönsä, ja värikoodattu kuva toimii pohjana pelin tason luonnille.

DWAsave-tiedostoformaattii on ihmisen ymmärrettävissä ja se koostuu lohkoista. Tiedosto näyttää tältä:

Player:

LOC:8350.0|505.0

STAT:1000.0|1000.0

INV:empty

#

Enemy:

STAT:750.0|450.0|200.0,1450.0|1950.0|200.0,1700.0|300.0|200.0,2250.0|300.0|200.0,3700.0|5250.0|200.0,3950.0|2200.0|200.0,4150.0|4600.0|200.0,4150.0|4850.0|200.0,4750.0|5000.0|200.0,5050.0|1350.0|200.0,5050.0|1850.0|200.0,5250.0|2100.0|200.0,5450.0|1850.0|200.0,5550.0|650.0|200.0,6550.0|1900.0|200.0,6800.0|1900.0|200.0,7200.0|2150.0|500.0,7500.0|2600.0|200.0,8000.0|904.0|200.0,8100.0|2600.0|200.0,8300.0|2050.0|200.0,8400.0|1300.0|200.0,9100.0|1900.0|200.0,9350.0|1900.0|200.0

#

Followingenemy:

STAT:500.0|500.0|7200.0|2150.0|

#

Level:

UNS:false|false

LLN:Large City

LI:SFW|100.0|450.0,HP1|250.0|2050.0,HP1|450.0|450.0,EP1|600.0|450.0,RFW|1200.0|450.0,EP1|2300.0|150.0,HP1|4150.0|5150.0,HP1|5100.0|650.0,EP1|5550.0|1350.0,EP1|6250.0|5800.0,EP1|6500.0|5800.0,SFW|7950.0|1350.0,HP1|8000.0|700.0,RFW|8500.0|500.0,EP1|9400.0|1900.0

#

FILEEND

Formaatin lohkot ovat pelaajalohko, vihollislohko, pelaajaa seuraavan vihollisen lohko, ja tasolohko. Kukin lohko sisältää alaotsikoita, jotka erottuvat varsinaisesta datasta kaksoispisteen avulla. Datan eri osat on eroteltu pystyviivoin. Jos lohkossa on useampien vihollisten dataa, eri vihollisten data on eroteltu pilkuin. Lohkoja erottaa #-merkki. Koko tiedoston päättää FILEEND-tunniste.

Peli ei lue dataa verkosta tai kirjoita dataa verkkoon.

Testaus

Testasin peliä enimmäkseen pelinaikaisen tarkkailun ja println-komennon käytön avulla. Metodeja pickup, drop, ja loadgame testasin myös yksikkötestein. Testaus vastasi pääosin suunnitelmassa esitettyä. En suunnitelmasta poiketen kirjoittanut savegame-metodille yksikkötestejä, sillä println-tuloste ja loadgame-metodin toiminta osoittavat savegame-metodin toiminnan. Käytin yksikkötestien kirjoittamiseen ScalaTest-kirjastoa. Peli läpäisee kaikki testit. Testauksen suunnittelussa ei ollut olennaisia aukkoja.

Loadgamen yksikkötesti tarkastaa latautuvatko ehjän tallennustiedoston sisällään pitämät asiat oikein peliin. Se myös varmistaa että metodi heittää oikean poikkeuksen erilaisten korruptoituneiden tiedostojen tapauksessa.

Pelin yksikkötestit eivät toimi UnitTests.scala-tiedoston kautta suoritettuina grafiikkakirjastoon liittyvän ongelman vuoksi. Testit voi suorittaa pelinäkymässä o-näppäintä painamalla. Ne on hyvä suorittaa heti uuden pelin aluksi.

Ohjelman tunnetut puutteet ja viat

Pelin päävalikon nappien kuvat eivät aina palaudu normaalitilaan, jos hiiren siirtää nopeasti nappien yli. Ongelma näyttäisi johtuvan siitä että scalafx:n tapahtumankuuntelija ei ehdi havaita hiiren kursorin poistumista napin päältä. Ongelma saattaisi ratketa esimerkiksi käyttämällä korkeampia nappeja.

Valikoiden ulkonäkö saattaa kärsiä erilaisilla näytöillä. Ongelma ratkeaisi käyttämällä tarkempia kuvia.

Pelin volyymiasetuksen vaikutus ei näy saman tien. Ongelma johtuu äänien käsittelyyn käytettävän AudioClip-luokan ominaisuuksista. Ongelman voisi ratkaista esimerkiksi siirtymällä käyttämään jotakin muuta luokkaa.

Peli hidastuu hieman vihollisten lähellä. Ongelman voisi korjata parantamalla pelin törmäysten suorituskykyä. Yksi tapa tehdä tämä olisi siirtyä etäisyyteen perustuvasta törmäyksen tunnistuksesta scalafx:n oman intersects-työkalun käyttöön ja jakaa maailma osiin. Törmäykset tapahtuisivat vain siinä osassa, missä pelaaja on.

Jos pelaajahahmo putoaa korkealta, se saattaa vajota hieman maan sisään. Ongelma johtuu siitä että pelaajan koordinaatti “hyppää” joka tickin aikana pelaajan nopeuden verran. Jos nopeus on suuri, pelaaja joutuu hieman maan sisään. Ongelmaa voisi pyrkiä ratkaisemaan esimerkiksi kasvattamalla colliderien törmäysetäisyyttä nopeuden kasvaessa, mikä kompensoisi suurta nopeutta.

Esimerkiksi kuvatiedostojen poistaminen projektin kansiosta kaataa pelin, koska catch-lohko ei jostain syystä onnistu tunnistamaan seuraavaa poikkeusta. Tavalliset scalan poikkeukset peli hoitaa näyttämällä virheilmoituksen.

3 parasta ja 3 heikointa kohtaa

Parhaat kohdat:

1. Esinejärjestelmä

Perustelu: Esinejärjestelmä toimii hyvin ja uusia esineitä on helppo lisätä. Aseet ilmestyvät pelaajan käteen suunnitellusti, mikä ei ollut aivan helppoa toteuttaa.

2. Tallennustiedostot

Perustelu: Järjestelmä, joka kasvattaa ohjelman laatua pelinä huomattavasti. Tallennuspaikkojen määrän kasvattaminen on helppoa tulevaisuudessa.

3. Karttatila

Perustelu: Kameran vapaa liikuttaminen ja zoomaaminen avaavat valtavasti mahdollisuuksia esimerkiksi uusille aseille ja esineille tulevaisuudessa.

Heikoimmat kohdat:

1. Suorituskyky
2. Koodin siisteys

Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu

Toteutin projektin pääosin suunnitelman mukaisessa järjestyksessä. Projektin päävaiheet olivat seuraavat:

1. Esinejärjestelmä	Alku 25.2	Toiminnassa 27.2
2. Pelaajan HUD	Alku 1.3	Toiminnassa 5.3
3. Pelin valikot	Alku 9.3	Toiminnassa 16.3
4. Tallennustiedostot	Alku 19.3	Toiminnassa 21.3
5. Uusi vihollinen	Alku 14.4	Toiminnassa 16.4

Projektin päävaiheet veivät yleisesti suunniteltua vähemmän aikaa. Siksi tein päävaiheiden ohella peliin monia parannuksia joita alkuperäisessä suunnitelmassa ei ollut. Näistä merkittävimpiä olivat GamePos-luokka ja pelin sijaintijärjestelmän uudistus sekä pelin karttatila.

En lopulta toteuttanut suunnitelmassa mainittua esineiden liikettä vihollisen pudottaessa esineen, koska toiminto ei ollut erityisen merkittävä pelattavuuden kannalta ja pelin suorituskyky olisi kärsinyt.

Projektiin kului lopulta yhteensä jonkin verran vähemmän aikaa kuin olin arvioinut. Arvioni oli 65 tuntia, kun todellinen ajankäyttö oli n. 60 tuntia. Suurin ajankäytön ero suunnitelmaan verrattuna tuli tallennustiedostojen kohdalla. Olin varannut niiden toteuttamiseen kolme viikkoa, kun varsinainen toteutus syntyi kolmessa päivässä.

Kokonaisarvio lopputuloksesta

Sanoisin projektin olevan lopulta onnistunut. Projekti laajensi jouluna aloittamaani peliä huomattavasti ja paransi sen laatua siten että pelin laajentaminen tulevaisuudessa on helppoa.

Projektin hyviä puolia ovat esimerkiksi yllä luetellut esinejärjestelmä, tallennustiedostot ja karttatila, mutta lisäksi sain (ainakin omasta mielestäni) pelin myös näyttämään ja kuulostamaan suhteellisen hyvältä. Kaikki pelin hyödyntämät kuvat (toisen tason maapallotaustaa lukuun ottamatta) ja äänet ovat itse luomiani.

Huonoja puolia projektissa ovat ajoittainen koodin epäsiisteys ja hieman heikohko suorituskyky. Tulevaisuudessa suorituskykyongelmaa voisi pyrkiä ratkaisemaan jakamalla pelin maailma lohkoihin ja tutkimalla törmäyksiä vain ajankohtaisen lohkon tiilien joukosta. Aivan projektin loppumetreillä toteutin lohkoja käyttävän törmäystunnistuksen. Se paransi suorituskykyä, mutta toi mukanaan uusia bugeja joiden ratkaisuun aika ei enää riittänyt. Päädyin lopulta käyttämään raskaampaa mutta tarpeeksi hyvin ja varmasti toimivaa tapaa.

Työssä ei ole juurikaan puutteita. Tulevaisuudessa projektia voisi parantaa ainakin lisäämällä uusia aseita, esineitä ja tasoja, sekä kasvattamalla grafiikan laatua.

Valitut ratkaisumenetelmät ja tietorakenteet soveltuivat tarkoituksiinsa hyvin.

Jos nyt aloittaisin projektin alusta, laatisin samantien oman sijaintiluokan jolla pitäisin pelin “loogisen” koordinaatiston ja kuvan koordinaatiston erillään. Suunnittelisin pelin rakenteen myös tarkemmin esimerkiksi UML-notaatiota hyödyntämällä.

Viitteet

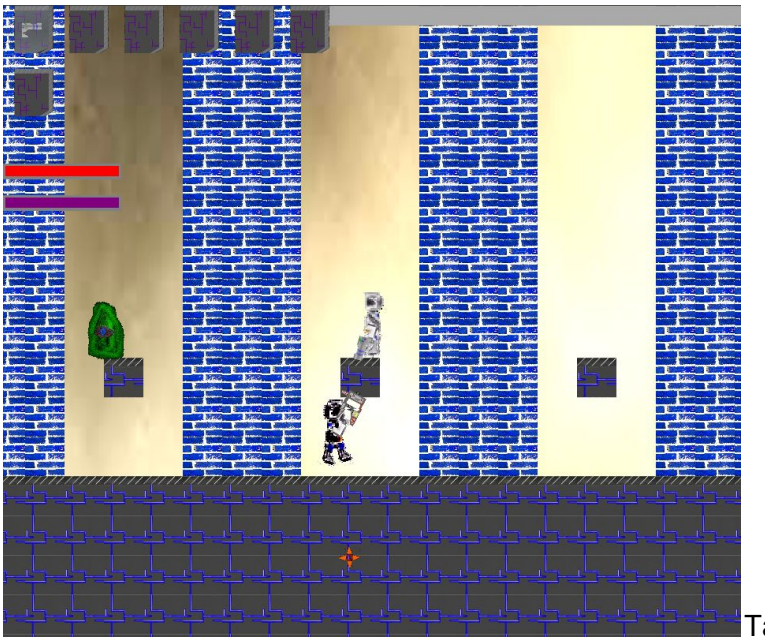
Scalafx:n dokumentaatio: http://www.scalafx.org/api/8.0/index.html#package

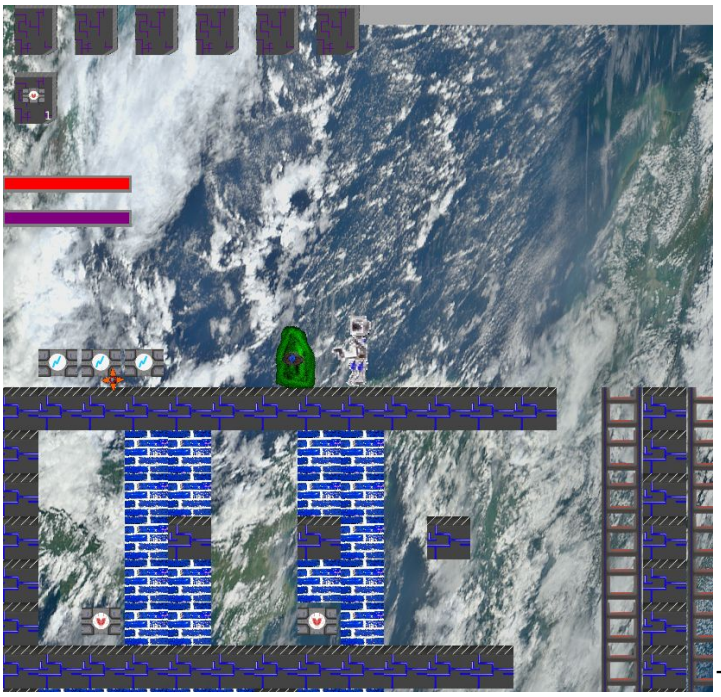
Kurssin oppimateriaali: https://plus.cs.hut.fi/studio_2/k2019/

Mark Lewisin Youtube-kanava: https://www.youtube.com/channel/UCEvjiWkK2BoIH819T-buioQ

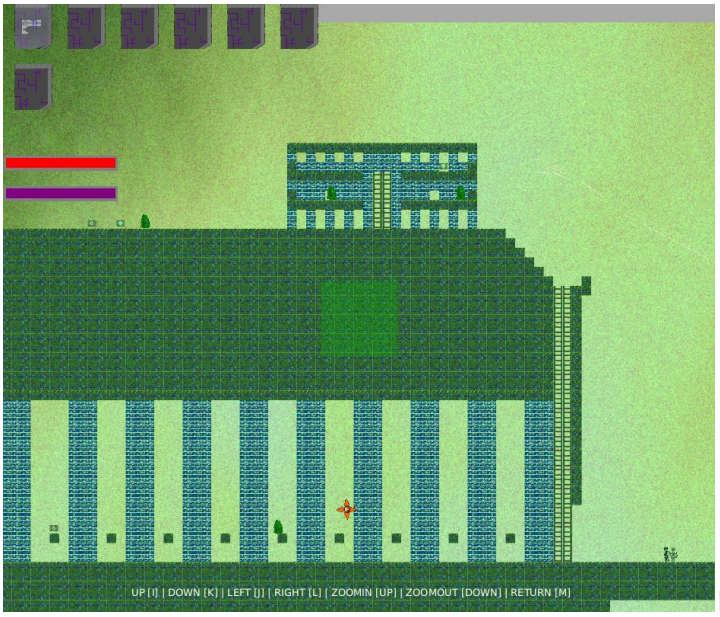
Liitteet

Päävalikko


Taso 1



Taso 2



Karttatila