Y2-Projektityön Dokumentti

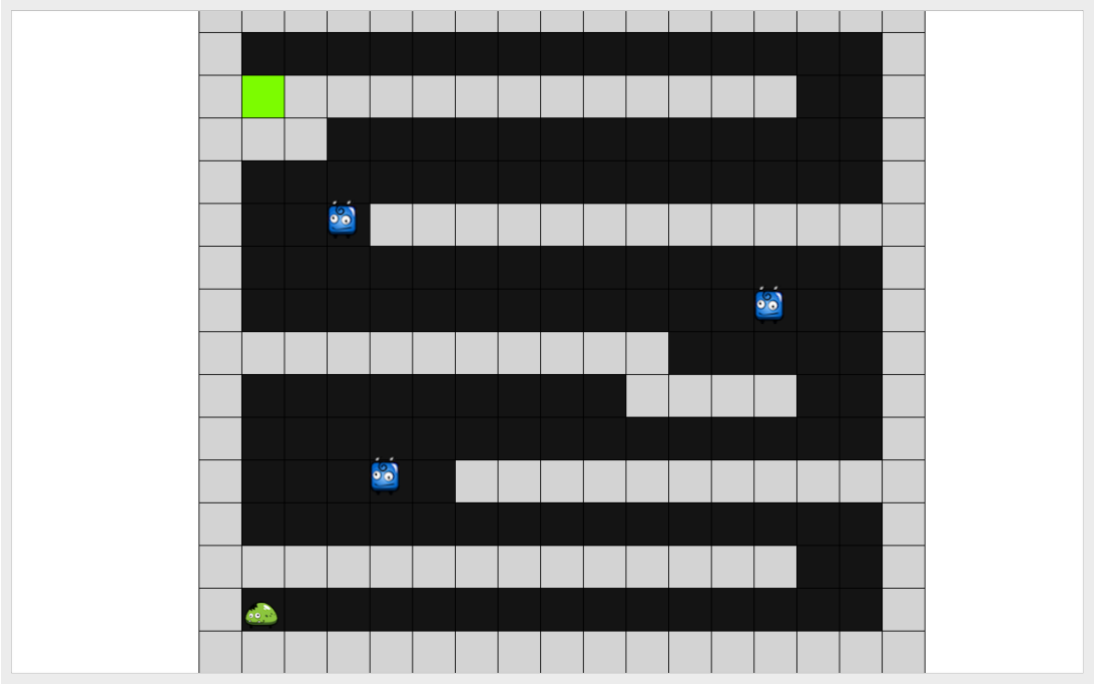
Aapo Linjama, 587895, Elektroniikka ja sähkötekniikka, 21.4.2019

1.Yleiskuvaus

Tasohyppelypeli, jossa pelaaja ohjaa hahmoa kentän läpi hyppimällä tasojen päälle. Peli on toteutettu niin, että koko kenttä näkyy alusta asti näytöllä ja tavoitteena on päästä ylös vihreään maaliruutuun. Jos pelaaja osuu liikkuviin sinisiin hahmoihin hän häviää eplin ja joutuu takaisin alkuun. Tavoitteena oli toteuttaa peli vaativana, siten että lisäominaisuuksina olisivat olleet esim. kenttäeditori ja highscore-taulukko, mutta niiden implementoimiseen ei aika riittänyt. Pelissä on kuitenkin liikkuvia vihollisia ja perusvaatimukset täyttyvät. Kokonaisuutena pelin toteutuksen vaikeusaste on mielestäni keskivaikea.

2.Käyttöohje

Ohjelman voi ajaa main.py-tiedostosta käsin. Eli ajetaan normaalisti main-tiedosto, jolloin ikkunan pitäisi aueta, jossa on näkyvillä pelikenttä, liikkuvat viholliset ja pelihahmo. Pelihahmoa voi alkaa liikuttelemaan heti ikkunan auettua. Kontrollinäppäimet ovat: a, d ja w. A-näppäimestä hahmo liikkuu vasemmalle päin, mikäli mahdollista, d-näppäimestä hahmo liikkuu vuorostaan oikealle ja w-näppäimestä hahmo hyppää.



3.Ulkoiset kirjastot

PyQt5-kirjasto on ainut ulkoinen kirjasto mitä projektissa on käytetty.

4.Ohjelman rakenne

Gui.py

Ohjelman graafisen käyttöliittymän toteuttamisesta vastaa GUI-luokka. Kun main-tiedosto ajetaan luo se GUI-olion, joka aloittaa peli-loopin. GUI-luokassa on muun muassa määritelty widget pääikkunalla ja layout, jolla eri widgetit asettuvat pääikkunaan. Lisäksi GUI:ssa luodaan scene pääikkunaan, jotta graafisia objekteja voidaan piirtää näkyviin. Pelihahmon ja -kentän sekä vihollisten olioiden luonti tapahtuu myös tässä luokassa. Kaikki olennainen pelin alustaminen tapahtuu oikeastaan tässä luokassa. Tässä luokassa on myös useita metodeja, jotka huolehtivat mm. pelin etenemisestä ja käyttäjän kontrollien vastaanottamisesta. Esimerkiksi keyPressEvent-metodi huolehtii siitä, että käyttäjän antamat kontrollit tallentuvat. TimerEvent-metodia taas kutsutaan joka kerta kun ohjelmaa päivitetään (kolmen millisekunnin välein), eli timerEvent kutsuu metodia Player ja Enemy -luokista, jotka päivittävät graafisen näkymän vastaamaan kutakin tilannetta. GUI-luokasta on assosiaatio lähes kaikkiin muihin luokkiin.

Player.py

Player-luokassa alustetaan pelaajalle olio, jota hän voi liikutella pelikentällä. Päädyin käyttämään PyQt-kirjaston GraphicsPixmapItem-luokkaa pelihahmon graafisen ilmentymän toteuttamiseen, sillä tällä valinalla pystyin lataamaan png-formaatissa olevan kuvan pelihahmoksi. Player-luokassa on myös apumuuttujiksi tarkoitettuja attribuutteja, joiden avulla peliä saadaan päivitettyä aina oikealla tavalla. Apumuuttujiin tallennetaan esim. tieto siitä onko peli voitettu tai hävitty, tai pystyykö pelaaja hyppäämään. Luokan metodi game_update on hyvin olennainen ohjelman kannalta sillä, siellä tapahtuu suurin osa pelin kulkuun vaikuttavista prosesseista kuten selvitys siitä onko pelaaja voittanut tai hävinnyt pelin, lisäksi tarkastellaan käyttäjän antamia näppäinkomentoja. Loput luokan metodeista ovat apumetodeja game_updatelle, niissä esimerkiksi määritetään voiko pelaaja kulkea johonkin suuntaan, eli törmäyksen havaitseminen ja hahmon liikuttaminen on toteutettu kokonaisuudessaan tässä luokassa. Lisäksi tekstien ’’You Won!” ja ”You Lost!” ilmestyminen määritellään tässä luokassa, vaikka niiden esille tuominen tapahtuukin GUI-luokassa. Luokasta on assosiaatiot Map-luokkaan, jotta pelikentästä saadaan informaatio törmäyksen tunnistamiseen ja liikkumiseen. Mitään koordinaattiluokkaa ei lopulta tarvittu sijainnin tutkimiseen, sillä pelihahmon koordinaatit oli melko helppo muuntaa neliöiden koordinaateiksi kaksiulotteiseen listaan.

Map.py

Tässä luokassa luetaan tekstitiedostosta dataa kaksiulotteiseen listaan, jonka avulla määritellään pelikenttä ja luodaan jokaiselle koordinaatille oma Square-olio. Pelikentän lukeminen tiedostosta on mielestäni kätevä tapa toteuttaa pelikentän määritteleminen, lisäksi kenttä ei ole näin ennalta määrätty vaan olisi melko helppo toteuttaa erilaisia kenttiä, kunhan tekstitiedostoja toteuttaisi lisää. Lisäksi kenttäeditorin olisi voinut toteuttaa siten, että kun käyttäjä on toteuttanut haluamansa kentän, se olisi tallennettu tekstitiedostoon ja pelikentän luonti vaiheessa tästä tiedostosta olisi luettu pelikenttä. Ikävä kyllä näiden ominaisuuksien toteuttamiseen ei jäänyt aikaa. Luokassa olevat metodit ovat tarkoitettu siihen, että muut luokat saavat niiden avulla tietoa esim. kentän leveydestä tai korkeudesta. Metodin get_map avulla saadaan koko kaksiulotteinen lista muiden luokkien käyttöön. Graafinen toteutus pelikentästä on SquareGraphs-luokassa. Tästä luokasta on assosiaatio Square-luokkaan.

Square.py

Square-oliot ovat pelikentän neliöitä, jotka muodostavat kokonaisuuden. Tässä luokassa ei toteuteta graafisia oliota vain osien äly, eli ovatko ne esteitä vai vapaasti kuljettavaa aluetta. Is_obstacle-metodista saadaan tieto onko jokin tietty pelikentän osa este vai ei. Käytännössä tätä metodia kutsutaan, kun käydään Map-luokassa luotua kaksiulotteista listaa läpi Square-olioista ja halutaan tietää minkä väriseksi neliöt määritellään, tai kun tutkitaan törmäyksiä Player-luokassa.

Square_graphics.py

SquareGraphs-luokassa luodaan create_square_graphs-metodissa kaksiulotteinen lista samaan tapaan kuin Map-luokassa, mutta tällä kertaa listan instanssit ovat QGraphicsRectItem-olioita. Nämä oliot ovat siis PyQt-kirjaston luokasta toteutettuja olioita, jotka saadaan lisättyä GUI:ssa sceneen ja näin näkyviin peliruudulle. Map-luokan toteuttamasta listasta saadaan Square-olioista tieto pitääkö neliö lisätä mustana vai harmaana graphSquare-listaan. Create_square_graphs-metodi palauttaa listan QGraphicsRectItem-olioista, eli neliöistä, GUI:hin ja näin pelikentän ''piirtäminen'' ruudulle voidaan toteuttaa siellä. Tästä luokasta on siis assosiaatio Map-luokkaan.

Enemy.py

Tässä luokassa luodaan oliot vihollisille. Käytin jälleen GraphicsPixMapItem-oliota graafisen olion toteuttamisessa, sillä kuten aiemmin sain näin kätevästi ladattua png-formaatissa olevan kuvan vihollisille. Metodi enemy_update suorittaa olion sijaiinnin siirtämisen, ja GUI-luokassa se piirretään aina uuteen sijaintiin. Enemy-oliot liikkuvat vakionopeudella ylös ja alas.

Enemy_container.py

Enemy_container-olio on nimensä mukaisesti varasto Enemy-olioille. Tämä luokka tarvittiin siksi, että joka peliloopin kierroksella voitaisiin tutkia osuuko pelaaja viholliseen, eli häviääkö hän pelin. Säiliönä vihollisille toimii siis tavallinen lista ja add_enemy-metodilla tähän listaan voidaan lisätä uusi instanssi. Get_enemies-metodi taas palauttaa listan. Enemy_containerista on luonnollisesti assosiaatio Enemy-luokkaan.

5.Algoritmit

Törmäyksien havaitseminen, pelaajan liikkumisen rajoittaminen ja ''gravitaatio'' on toteutettu tutkimalla viereisiä neliöitä ja niiden sisältöä, eli ovatko ne esteitä vai vapaasti kuljettavaa aluetta. Esimerkiksi kun käyttäjä liikuttaa pelihahmoa oikealle tutkii Player-luokan metodi can_moveright ensin onko pelihahmon yläosa ja alaosa eri indekseissä pelikenttää mallintavassa matriisissa. Jos käy niin, että yläosa on eri indeksissä, käytetään tutkimisen referenssinä alaosan indeksiä. Yläosan

ollessa kuitenkin esteneliön rajalla käytetään yläosan indeksiä referenssinä. Tämän jälkeen tutkitaan onko referenssitason oikealla puolella oleva neliö este, jos on can_moveright palauttaa Falsen ja hahmoa ei voi liikuttaa oikealle.

Can_moveleft toimii täysin samalla logiikalla paitsi nyt tutkitaan referenssitason vasemmalla puolella olevaa neliötä.

Putoaminen eli ns. gravitaatio on toteutettu metodissa fall siten että jokaisella peliloopin kierroksella tutkitaan pelihahmon alapuolella olevat neliöt, eli hahmon vasemmassa ja oikeassa reunassa alapuolella olevat neliöt. Jos niistä kumpikaan ei ole este lähtee hahmo putoamaan. Kuitenkin jos hyppy on kesken ei tätä tarkastelua tehdä vaan tehdään tarkastelu onko ylöspäin meneminen mahdollista, mikäli se ei ole alkaa hahmo putoamaan.

Ylöspäin liikkumisen tutkiminen tehdään jump-metodissa, siellä yksinkertaisesti tutkitaan pelihahmon yläpuolella olevat neliöt oikeasta ja vasemmasta reunasta, jos jompikumpi on esteneliö ei ylös päin liikkuminen ole mahdollista.

Törmäykset vihollisten kanssa tutkitaan joka kierroksella siten että vihollisten säiliön toteuttava Enemy_container olio iteroidaan läpi ja PyQt-kirjaston CollidesWithItem-metodilla tutkitaan osuvatko pelaaja ja jokin säiliön vihollinen toisiinsa. Jos pelaaja ja vihollinen osuvat toisiinsa ilmestyy ruudulle sadan peliloopin kierroksen ajaksi teksti ”You Won!” ja pelaaja asetetaan välittömästi takaisin aloitusruutuun.

Algoritmi, joka tutkii onko peli voitettu toimii siten, että se muuttaa pelaajan koordinaatit neliöiden koordinaateiksi eli kaksiulotteisen listan indekseiksi ja tutkii onko pelihahmo vihreässä ruudussa. Mikäli pelaaja on päässyt maaliin, ilmestyy ruudulle teksti ”You Won!”, teksti on ruudulla näkyvissä 100 peliloopin kierrosta. Pelihahmo asetetaan takaisin alkuun viidenkymmenen peliloopin kierroksen jälkeen.

Pelikenttä luetaan siis int-tietotyyppisinä alkioina tekstitiedostosta kaksiulotteiseen listaan Map-luokassa, jonka jälkeen jokainen listan alkio käydään läpi, jos listan alkio on 0 tarkoittaa se pelikentällä vapaasti kuljettavaa Square-oliota, jos taas alkio on 1 tarkoittaa se este Square-oliota pelikentällä. Kuten aikaisemmin mainittu tämä muodostettu map-lista iteroidaan läpi SquareGraphs-luokan metodissa create_square_graphs ja luodaan uusi kaksiulotteinen lista, muttä tällä kertaa graafisille QGraphicsRectItem-olioille. Viimeisessä vaiheessa tämä graafisille instansseille tehty lista käydään läpi GUI:ssa ja lisätään sceneen, jotta kenttä saadaan näkyviin näytölle.

6.Tietorakenteet

Tavalliset python-kirjaston listat olivat hyvä tapa tallentaa dataa mielestäni. Kuten edellä mainittua, koko pelikenttä on käytännössä tallennettuna kaksiulotteiseen listaan, viholliset ovat myös omassa listassaan. Painettut näppäimet tallentuvat ohjelmassa set-tietotyyppiseen muuttujaan. Tietorakenteet, joita käytän ohjelmassa ovat oikeastaan kaikki muuttuvatilaisia.

7.Tiedostot

Pelikenttä luetaan kentta1.txt-tiedostosta, joka on siis tekstitiedosto. Tieto on esitetty siinä ykkösinä ja nollina, välilyönnillä erotettuna matriisina. Jokainen 1 ja 0 vastaa muodostettavan pelialueen indeksiä kaksiulotteisessa listassa. Jos aikaa olisi jäänyt enemmän olisi kenttiä voinut luoda lisää helposti. Toki silloin pelihahmon, maaliruudun ja vihollisten paikkoja olisi pitänyt mahdollisesti muokata.

8.Testaus

Ohjelman testaus suoritettiin lisäämällä print-komentoja paikkoihin ja haaroihin joihin oletettiin peliloopin menevän. Lisäksi ohjelman käännyttyä testasin toteuttamiani ominaisuuksia pelissä ja pyrin testaamaan ne läpi mahdollisimman tarkasti. Jos jokin ominaisuus ei toiminut sai sen syyn selville melko nopeasti, kun lisäili print-komentoja ja tutki niiden muuttujien arvoja, jotka olivat vastuussa prosessin etenemisestä. Lisäksi kun ohjelman ajoi ja kokeili toteuttamiaan ominaisuuksia eri tavoilla huomasi usein, jos niissä oli puutteita. Suunnitteluvaiheessa oli vielä tarkoituksena toteuttaa yksikkötestejä, mutta niille ei jäänyt aikaa, mutta kaikki suunnitelmassa mainitut ominaisuudet testattiin kyllä muilla tavoin.

9.Ohjelman tunnetut puutteet ja viat

Suurimmat puutteet ohjelmassa ovat melko varmasti törmäyksenhavaitsemisessa. Kun esimerkiksi hahmolla hyppää tai putoaa sivusuunnassa kohti esteneliöitä voi niihin jäädä kiinni eli leijumaan ilmaan. Tämän bugin voisi poistaa lisäämällä tarkastelumetodeihin haaroja erikoistapauksille, joissa hahmo on kosketuksissa vain toisesta reunasta esteneliöihin. Bugi johtuu tällä hetkellä esimerkiksi pudotessa siitä, että tarkastelu tehdään hahmon alareunassa oleville neliöille, jos toinen niistä on edes este niin hahmo ei pysty putoamaan. Ehtoa voisi muuttaa niin että alareunoissa olevien neliöiden ei tarvitsisi molempien olla vapaasti kuljettavaa aluetta, mutta tällöin hahmo saattaisi pudota esteneliöiden sisään joissain tapauksissa. Lisää tarkasteltavia tapauksia tarvittaisiin siis useita, jotta bugi saataisiin täysin pois.

Player-luokasta löytyy myös metodeja, jotka tutkivat tapauksia, joissa pelihahmo olisi menossa ulos pelialueelta, mutta lopulta päätin lisätä pelialueen ympärille esteneliöt, sillä se helpotti ja lyhensi implementoitavien tarkastelujen määrää. Nämä metodit ovat siis toteuttamassani pelikentässä turhat.

Välillä ohjelma myös kaatuu, kun sen ajaa ja heti ikkunan auettua painaa kontrollinappia. En saanut selville mistä tämä johtuu.

10. 3 parasta ja 3 heikointa kohtaa

Mielestäni esteiden ja vihollisten kanssa törmäämisen havaitseminen toimii melko hyvin. Lisäksi kentän lukeminen tekstitiedostosta on mielestäni toteutettu sujuvasti ja siten että kentän voisi halutessaan lukea toisesta tiedostosta. Heikkouksia ovat esimerkiksi se että koodi saattaa olla ylipäätänsä melko vaikeaselkoista välillä muun muassa siksi, että aikaa kattavalle kommentoinnille ei riittänyt.

11.Poikkeamat suunnitelmasta

Suunnitelmasta poikettiin jonkin verran, sillä joitain ominaisuuksia jäi uupumaan mitä alun perin ohjelmaan piti tulla. Esimerkiksi aloitusnäyttö, josta käyttäjä voisi aloittaa pelin jäi implementoimatta. Alkuperäisessä suunnitelmassa oli myös tarkoituksena saada ohjelmaan näkyviin käytetty aika pelin suorittamiseen.

Tarkoitus oli siis toteuttaa projekti vaativana, jolloin lisäominaisuuksina olisivat olleet esim. kenttäeditori tai highscore-taulukko, ne jäivät kuitenkin myös pois. En alun perin ollut varma suunnitteluvaiheessa toteutanko ohjelmaan vihollisia, lopulta päädyin toteuttamaan liikkuvat viholliset.

Ajankäyttöarvio meni melko pieleen, sillä suurin osa työstä jäi viimeiselle viikolle, joka varmasti näkyy gittiin tehdyista pushauksista. Toisaalta olin melko varma, että saan suhteellisen lyhyessä ajassa pelin toteutettua hyvään muotoon, jos vain käytän ajan tehokkaasti. Toteutuksen pakkautuminen viimeiselle viikolle johtui muiden kurssien ja työpaikan aiheuttamasta työmäärästä.

12.Toteutunut työjärjestys ja aikataulu

Aloitin toteutuksen siitä, että sain ikkunan auki ja jaoteltua scenen ruutuihin, jotka olivat eri värisiä. Seuraavaksi etsin pelihahmolle hyvän graafisen png-kuvan ja loin sille PixMapItem-objektin sceneen ja näkyviin näytölle. Hyödynsin myös GIMP-ohjelmaa muokatessani hahmon halutun laiseksi. Tämän jälkeen aloin toteuttamaan sitä, että ohjelma tunnistaisi käyttäjän antamat syötteet kuten kontrollit ja liikkuisi niiden mukaan ruudulla. Kun sain tämän ominaisuuden alkuun jatkoin suunnitelemalla Map-luokan ja tiedostosta pelikentän lukemisen. Samaan aikaan aloin jo toteuttamaan törmäyksen havaitsemista. Kun törmäyksen havaitseminen, Square-luokka, Map-luokka, ja SquareGraphs-luokka olivat melko valmiit loin vihollisille oman luokan ja säiliön, johon ne tallennetaan. Lopuksi toteutin törmäyksen havaitsemisen vihollisten kanssa ja määrittelin milloin peli voitetaan tai hävitään, jolloin tekstit ilmestyvät näytölle. Ohjelman toteuttaminen meni pitkälti suunnitelman mukaisesti.

13.Arvio lopputuloksesta

Kokonaisuutena olen tyytyväinen kehittämääni peliin. Edellä on jo mainittu kohdat jotka on toteutettu mielestäni hyvin, mutta törmäyksen havaitsemisesta tuli suhteellisen hyvä, vaikka puutteita toki on. Lisäksi graafinen käyttöliittymä on hyvä. Puutteet johtuvat suurimmaksi osaksi siitä että olisi tarvittu enemmän tarkasteltavia tapauksia, jotta kaikki ominaisuudet olisi saatu hiottua hyviksi. Ohjelmaa voisi siis parantaa tulevaisuudessa lisäämällä niitä. Lisäksi kenttiä voisi tehdä lisää ja erilaisia vihollisia. Luokkajakoa olisi tietysti voinut laajentaa ja saada joitakin toimenpiteitä ’’piiloon’’ GUI:sta ja Player-luokasta, mutta muuten luokkajako on mielestäni toimiva. Ohjelman rakenne soveltuu melko hyvin laajennettavuuteen ja muutoksiin. Mutta jos esimerkiksi kenttiä tehdään lisää täytyy huolehtia, että pelaaja ja viholliset ilmestyvät uudessa kentässä järkeviin paikkoihin. Tiedoston lukeminen on toteutettu siten että tiedoston sisältöä voi helposti muuttaa ja ohjelma toimii silti halutulla tavalla.

14.Viitteet

Qt-kirjaston omat dokumentaatiot:

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

PyQt:n omat kirjastot

https://pypi.org/project/PyQt5/

https://stackoverflow.com/

https://www.tutorialspoint.com

https://www.w3schools.com