Y2 Projekti dokumentaatio

Lauri Westerholm 530868

1. **Henkilötiedot**

   Tasohyppelypeli

   Lauri Westerholm 530868

   Automaatio 2. vuosikurssi Raportti laadittu 22.4.2018

2. **Yleiskuvaus**

   Yleissuunnitelmassa oltiin suunniteltu, että peli toteutettaisiin alla olevien speksien mukaisesti:

   Peli: Tasohyppely

   Toteutetaan tasohyppelypeli, jossa pelaaja ohjaa hahmoa kentän läpi hyppimällä erilaisten tasojen päälle.

   - Graafinen käyttöliittymä (pelaaja ohjaa pelihahmoa mm. nuolinäppäimillä).
     - Aloitusnäyttö, josta pääsee varsinaiseen peliin painamalla Start Game -nappulaa
     - Lopetusnäyttö, joka kertoo pelin tuloksen (voitto vai häviö). Siirtyminen alkuun painamalla Continue-nappia
   - Toimiva törmäyksen tunnistus. Ohjelman täytyy tunnistaa, kun kaksi tai useampi objekti törmäävät. Esimerkiksi, jos pelaajan hahmo juoksee estettä päin, niin ohjelman tulisi huomata tämä ja estää pelihahmon eteneminen esteen läpi.
   - Hyppymekaniikka, objektien päälle pystyy hyppäämään
   - Vihollisia (ei varsinaista tekoälyä, liikkuvat edes takaisin vasemmalta oikealle)
   - Pelaaja voittaa, kun läpäisee kentän (hyppää/osuu maaliobjektiin). Häviäminen: pelaaja osuu objektiin, joka ns. tiputtaa pelihahmon pelikentältä tai vihollinen osuu pelaajaan
   - Valmis kenttä(t). Kentät etenevät lähtökohtaisesti vasemmalta oikealle.
   - Pelihahmon ja vihollisten grafiikkana yksinkertaisia kuvia (esim. jpeg muodossa)
   - Kentät tiedostomuodossa (tekstitiedosto), josta sitten luetaan pelikentäksi

   Vaikeustaso: vaikea (keskivaikea)

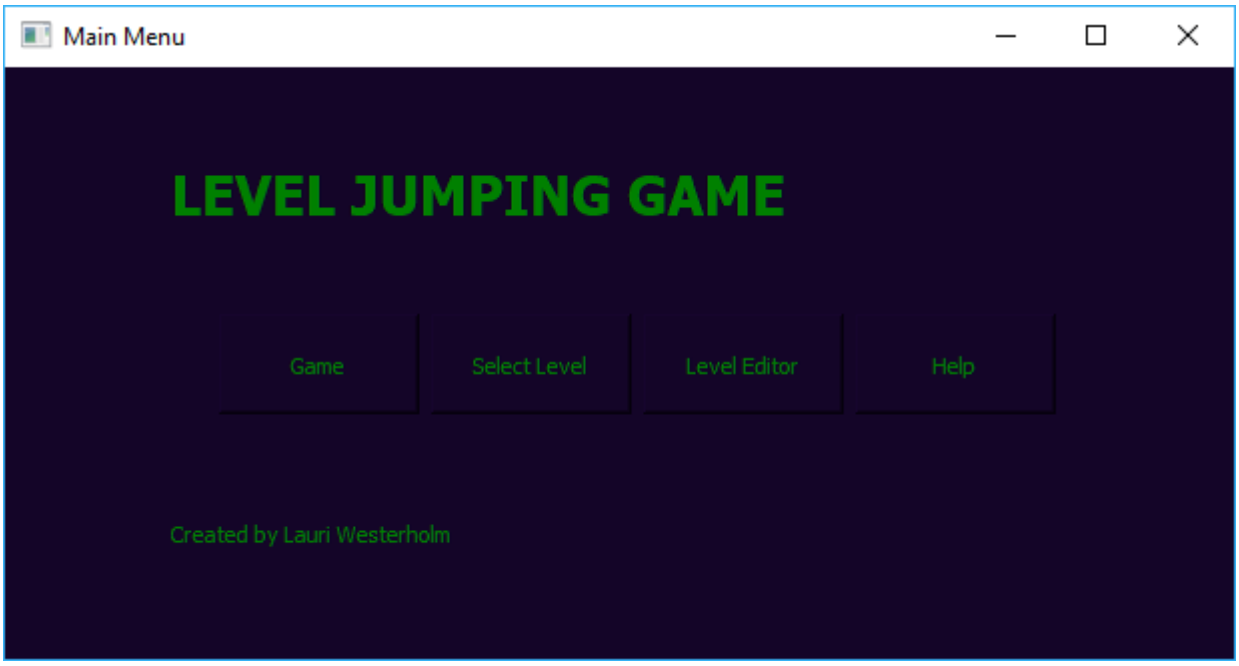
   Varsinainen toteutus poikkeaa suunnitelmasta siinä määrin, että käyttäjälle on toteutettu mahdollisuus muuttaa pelin parametriarvoja erillisen tiedoston avulla. Lisäksi on toteutettu kenttäeditori sekä kentän valintatoiminto, jotka myös vaativat suunnitelmasta poikkeavan valikkorakenteen.

   Näillä muutoksilla toteutetun projektin vaikeusaste on vaikea.

3. **Käyttöohje**

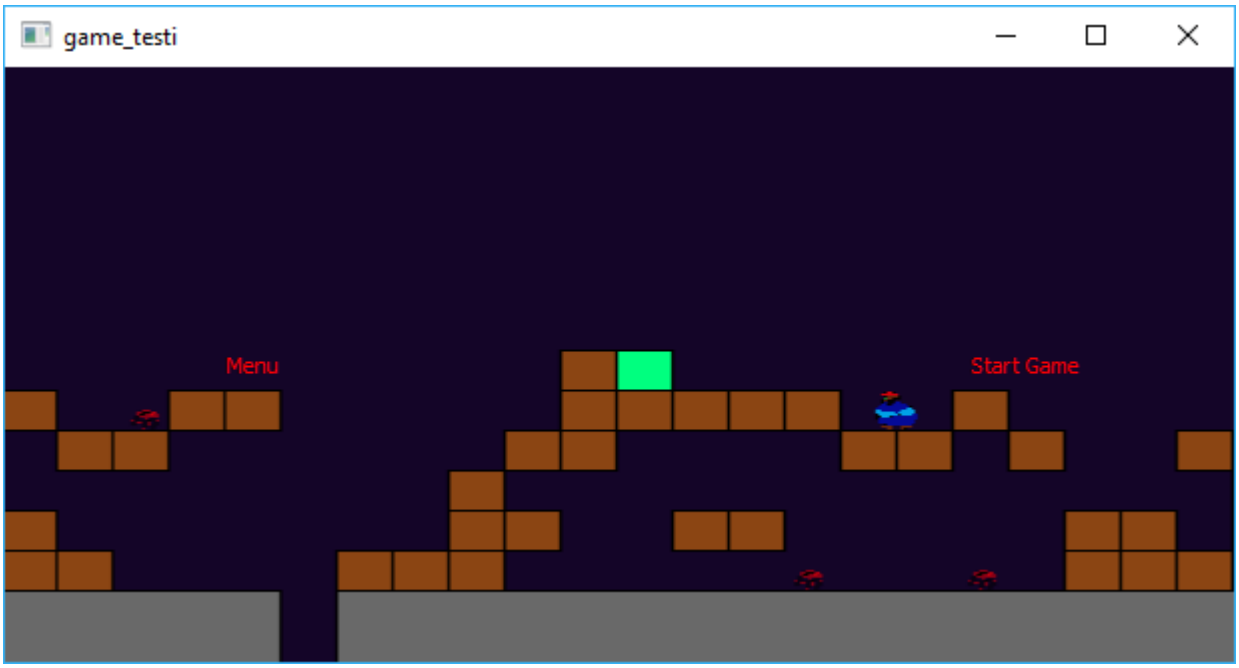
Ohjelman kansiorakenne on selitetty laajemmin README-tiedostossa. Lyhyesti ilmaistuna pelin kannalta olennaiset tiedostot löytyvät game_files -hakemistosta irrallisina (eivät ole alihakemistoissa). Peli käynnistetään suorittamalla start_the_game.py -niminen tiedosto, joka sisältää pääohjelman.

Kun ohjelma on käynnistetty, pitäisi graafisen käyttöliittymän ikkuna avautua, jos asetusarvotiedosto (kts. 8. Tiedostot) on speksien mukainen. Tämän jälkeen pelin käyttö tapahtuu graafisen käyttöliittymän kautta: tietyissä tilanteissa, yleensä ongelmatilanteissa, kuitenkin ohjelma tulostaa informaatiota komentoriville (tai sitten IDE:n vastaavaan ikkunaan eli aina stdout). Graafisen käyttöliittymän saa aina suljettua ESC-näppäimillä lukuun ottamatta pääpeliä (kts. pääpeliosio).



Kuva 1: Päävalikko

Päävalikosta (Kuva 1) käsin voi alkaa pelata pääpeliä painikkeella Game, vaihtaa pelattavaa tasoa painamalla Select Level ja luoda uusia kenttiä Level Editor:lla. Game ei kuitenkaan suoraan aloita pääpeliä valitussa kentässä, vaan avaa uuden valikon (Kuva 2), josta pääpelin voi aloittaa Start game -painikkeella ja palata päävalikkoon Menu-painikkeella (Menu johtaa aina päävalikkoon). Lisäksi päävalikossa on tarjolla Help-painike, joka avaa uuden ikkunan, jossa on selitetty perusteet pelin toiminnasta.



Kuva 2: Pääpelin valikko, kenttänä game_testi

Pelin käyttöliittymä on toteutettu siten, että se tukee dynaamista koon muuttamista tietyin rajoituksin (joillekin näkymille minimikoot). Ruudunpäivittäminen vaakasuunnassa on myös automatisoitu pelihahmon/kursorin sijainnin perusteella pääpelissä sekä kenttäeditorissa. Peli on suunniteltu lähtökohtaisesti vasemmalta oikealla pelattavaksi, joten ruutua ei päivitetä korkeussuunnassa ollenkaan.

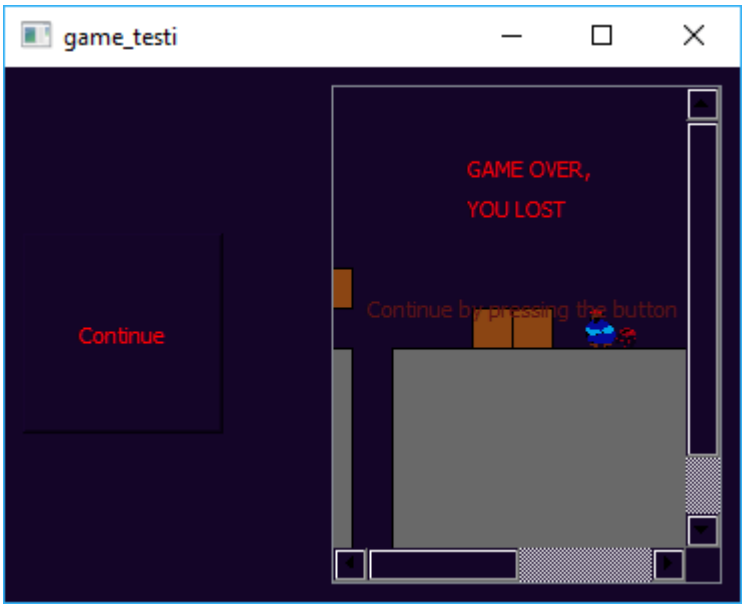
1. Pääpeli:

   Kun käyttäjä on aloittanut pääpelin painalla Start game -näppäintä, pelihahmo näkyy sinertävänä objektina, viholliset punertavina, esteet ruskeina ja maaliobjekti vihreänä (kuiluja ei luonnollisesti piirretä näkyviin). Pelin tavoitteena on koskettaa jotakin maaliobjektia, jolloin pelin voittaa ja ruudulle ilmestyy Continue-näppäin, joka palauttaa pääpelin aloittamisvalikkoon.

   Jos pelaaja putoaa (hyppää) vihollisen päälle, vihollinen tuhoutuu. Muuten viholliseen osuminen johtaa pelin häviämiseen, jolloin taas ruudulle ilmestyy Continue-näppäin (Kuva 3). Myös putoaminen kuiluun (riittää, että osuu kuilun reunaan) johtaa pelin häviämiseen.

   Pääpelissä pelihahmon liikuttaminen tapahtuu näppäimillä A (vasemmalle), D (oikealle), Space (hyppy suoraan ylös), Q (hyppy vasemmalle) ja E (hyppy oikealle). Periaatteessa myös nuolinäppäimiä (vasen ja oikea nuolinäppäin) voi käyttää, mutta niiden käyttö on varattu myös ikkunan päivittämiseen, jolloin ne eivät toimi hyvin pelihahmon liikuttamisessa. Suositeltavaa on, että nuolinäppäimiä käytetään vasta, kun Continue-näppäin on ilmestynyt ruutuun, koska niiden käyttämiselle ei ole muuten tarvetta, sillä peli päivittää ruutua automaattisesti niin kauan kuin pääpeli on käynnissä.

Pääpelin ollessa käynnissä ESC-näppäimellä pääse takaisin pääpelin aloittamisvalikkoon (jos ESC painaa uudelleen, graafinen ikkuna sulkeutuu). Pelihahmo on tällöin seuraavan kerran pääpelin käynnistettäessä takaisin aloitussijainnissa kuten pelin voittamisen ja häviämisenkin jälkeen.



Kuva 3: Pääpeli päättynyt häviöön

2. Tason valinta (Select Level)

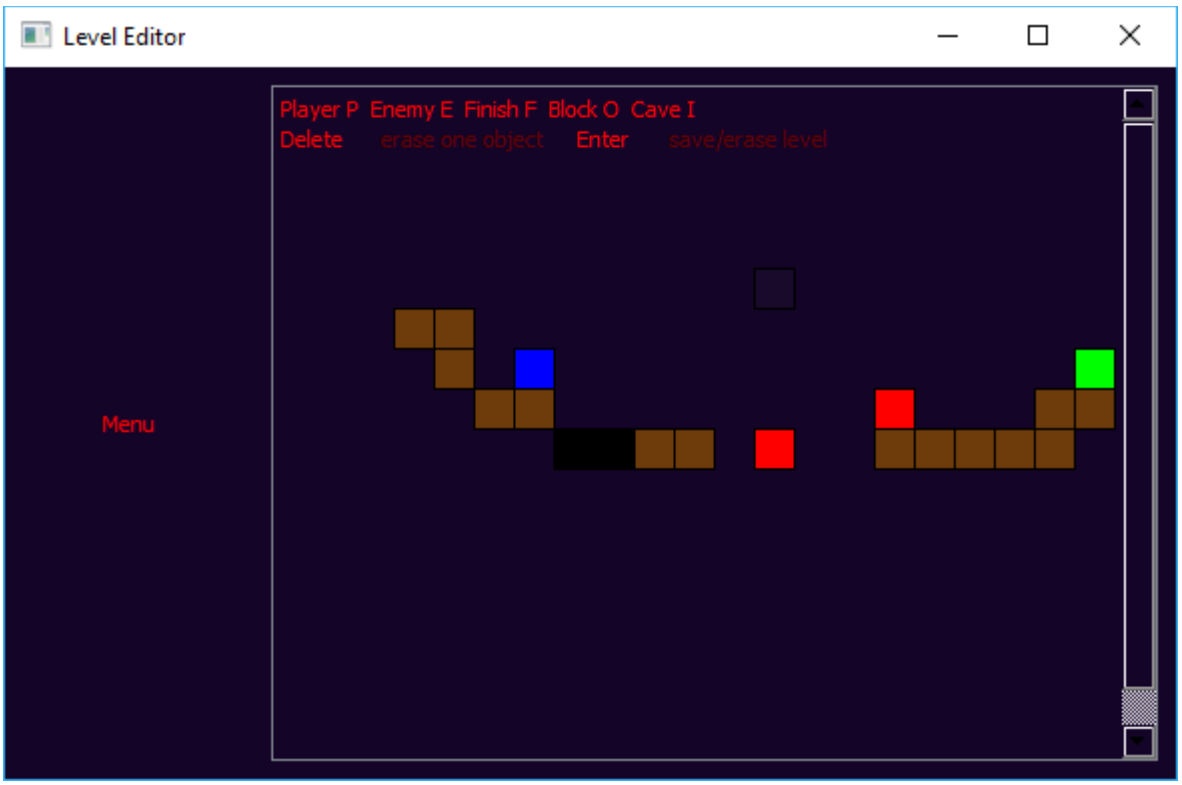
Tason valintaan pääsee päävalikosta painamalla Select Level -näppäintä. Tällöin keskelle ikkunaa tulevat näkyviin kaikki kentät, jotka on tallennettu game_levels -hakemistoon (Kuva 4). Kentän saa vaihdettua kirjoittamalla kentän nimen oheiseen tekstikenttään ja painamalla Enter (Return) - näppäintä. Tämän jälkeen peli kertoo käyttäjälle, onnistuiko kentän vaihtaminen. Jos vaihto onnistui, uusi kenttä on pelattavissa pääpelissä ja jos vaihto epäonnistui, vanha kenttä on pelattavissa.

Tason valinta on toteutettu siten, että kenttäeditorilla tehty kenttä on välittömästi pelattavissa. Periaatteessa kenttiä voi myös manuaalisesti lisätä game_files-kansioon (työlästä verrattuna kenttäeditorin käyttöön), ja ne ovat heti valittavissa, kunhan ei ole kentän lisäyksen aikana tason valinnassa. Tällöin pitää käydä päävalikossa ja käynnistää tason valinta uudelleen.

Kuva 4: Tason valinta

3. Kenttäeditori (Level Editor)

Kenttäeditorin avulla voi tehdä uusia kenttiä, jotka ovat onnistuneen tallennuksen jälkeen valittavissa tason valinnasta.

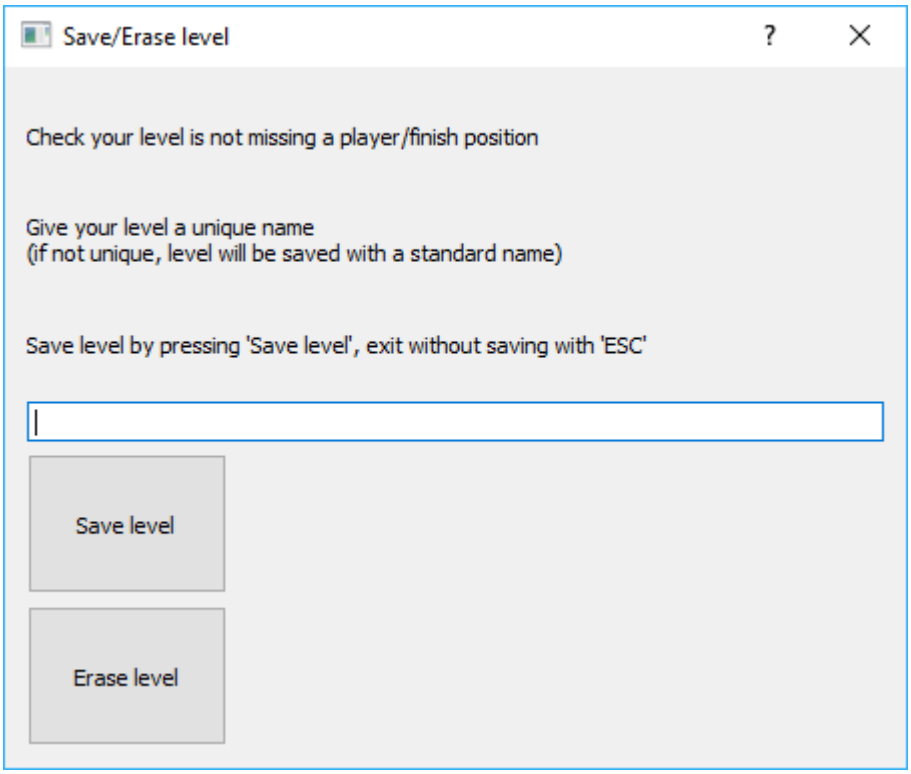


Kuva 5: Kenttäeditori

Kenttäeditorissa (Kuva 5) objektit ovat neliön mallisia ja nykyistä kursorin paikkaa merkitsee osittain läpinäkyvä neliö. Kursorin sijaintia muutetaan näppäimillä: A (vasemmalle), D (oikealle), W (ylös) ja S (alas). Nykyiseen kursorin sijaintiin saa

lisättyä haluamansa objektin näppäinkomennoilla: P (pelaaja), E (vihollinen), O (este), F (maali) ja I (kuilu). Yksittäisen objektin poistaminen tapahtuu painalla Delete-näppäintä. Kenttäeditorissa pelaaja on sininen, viholliset punaisia, esteet ruskeita, maalit vihreitä ja kuilut mustia neliöitä.

Valmiin kentän voi tallentaa painalla Enter- näppäintä, jolloin aukeaa uusi ikkuna (Kuva 6), jossa on painike kentän tallentamiseen, Save Level. Käyttöliittymä informoi käyttäjää siitä, onnistuiko kentän tallentaminen.



Kuva 6: Kentän tallentaminen/tyhjentäminen

Kenttäeditori on toteutettu siten, että se lähtökohtaisesti säilyttää kehitteillä olevan kentän niin kauan kuin graafinen käyttöliittymä on olemassa. Kenttäeditori ei tue valmiin kentän avaamista (tiedostosta) ja muokkaamista uudelleen. Tämän takia käyttäjän pitää erikseen tyhjentää kenttäeditori Enterin painalluksesta aukeavasta valikosta (Kuva 6) painamalla Erase Level ja hyväksymällä kentän tyhjennys. Tämän tyhjennyksen jälkeen kenttäeditorista ei siis enää pysty esimerkiksi tallentamaan luotua kenttä parannettuna uudella nimellä.

Muita huomion arvoista on se, että jokaisen tallennettavan kentän tulee sisältää tasan yksi paikka pelihahmolle sekä vähintään yksi maaliobjekti. Muuten kentän tallentaminen epäonnistuu. Kenttäeditori estää pelaaja lisäämästä ylimääräisiä pelaajasijainteja. Kuiluja taas ei saa sijaita kentän alimman tason yläpuolella (kenttäeditori estää myös sopimattomien kuilujen lisäyksen).

Tallennettaessa kentälle tulee antaa tekstilaatikkoon uniikki nimi, jos haluaa kentän tallennettavan määrittelemällään nimellä. Tämä tarkoittaa siis sitä, että kenttäeditori ei anna kirjoittaa olemassa olevien kenttien päälle, vaan tarvittaessa nimeää kentät käyttäen

standardinimeämiskäytäntöä. Tällöin kentät siis tallennetaan nimellä my_level ja loppuun juokseva numero luvusta 1 alkaen.

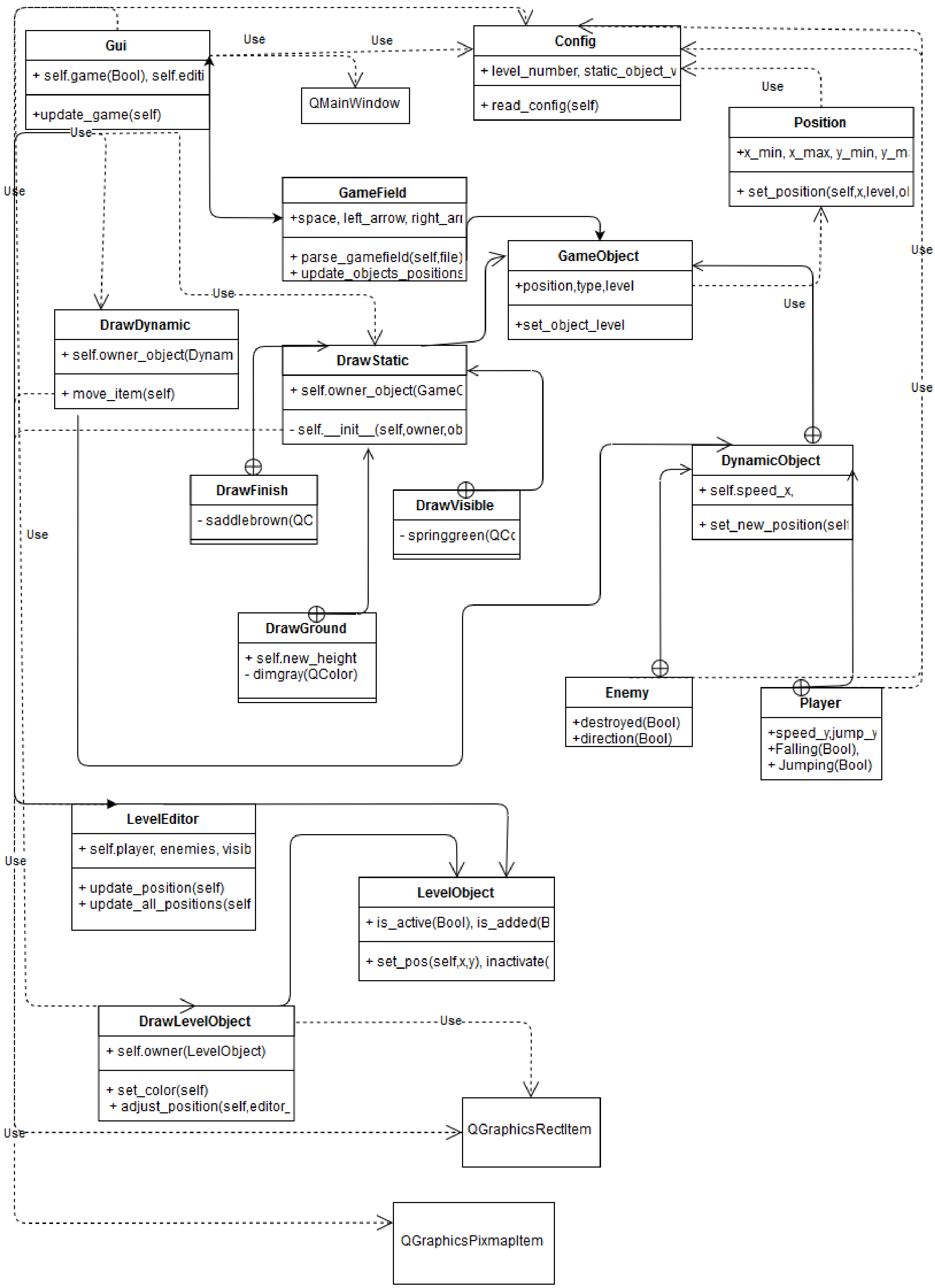
4. Ulkoiset kirjastot

Projektissa on käytetty Pythonin peruskirjastojen lisäksi ainoastaan PyQt5-kirjastoa. Tätä kirjastoa on käytetty pelin graafisen käyttöliittymän luomiseen ja ylläpitämiseen eli se on hyvin keskeisessä asemassa, eikä peli toimisi ollenkaan ilman sitä.

5. Ohjelman rakenne

Ensinäkin, kuten UML-kaaviosta (Kuva 7) nähdään, ohjelma rakentuu vahvasti Gui-luokan ympärille, joka huolehtii käyttöliittymän luomista ja ylläpitämisestä, minkä takia se on myös suhteellisen iso luokka. Guista käsin päivitetään esimerkiksi säännöllisesti pääpeliä GameField-luokan kautta, kun pääpeli on käynnissä. Vastaavasti taas, kun käyttäjä on kenttäeditorissa, päivitetään peliä LevelEditorin kautta. Tähän liittyen olennaisia metodeja ovat esimerkiksi update_game ja level_editor. Gui sisältää kuitenkin monia todella olennaisia metodeja, joita ilman peli ei toimisi ollenkaan.

Gui-luokka perii valmiista PyQt5 luokista QMainWindow-luokan. QMainwindow valmiita valikkorakenteita ei käytetty projektissa, joten periaatteessa samanlaisen käyttöliittymän olisi voinut toteuttaa myös QDialog kautta (QMainWindow perii QWidgetin ja tarjoaa työkalut pääikkunan käsittelyyn). Vaikka QMainWindowin valmiita lisäominaisuuksia suhteessa QDialogiin ei päädyttykään hyödyntämään, vaikutti loogiselta valinnalta käyttää ensisijaisen graafisen käyttöliittymän luomiseen siihen tarkoitukseen erikoistunutta luokkaa eli QMainWindowia.



Kuva 7: Ohjelman toteutunut luokkajako

Alkuperäistä suunnitelmaa noudatettiin graafisen käyttöliittymän toteutuksessa, joten graafisen käyttöliittymän ylläpitäminen on yhdessä luokassa eli Gui:ssa. Jälkikäteen ajateltuna Guista olisi voinut irrottaa joitakin omia luokkia, jotta se ei olisi niin raskas. Esimerkiksi irralliset QDialog-ikkunat, joita luodaan ainakin Help-ruutua ja tason tallennusta varten, olisi voinut irrottaa omiksi luokikseen.

Config-luokka huolehtii asetusarvotiedoston lukemisesta ja sen lukemia asetusarvoja käytetään monissa muissa luokissa (näkyy UML-kaaviossa riippuvuuksina). Tämän takia Config-olio luodaan pääohjelmassa heti alussa ja kaikkien sitä käyttävien luokkien asetusarvot päivitetään. Jos asetusarvojen lukemisessa on vakava virhe, muita oliota ei edes luoda. Config-luokka on siis selkeä kokonaisuutensa, minkä vuoksi se ehdottomasti tarvitsee oman luokkansa.

GameField-luokka on tehty pääpelin käsittelyä varten. Sen keskeisimpänä tehtävänä on muodostaa pelattava kenttä tekstitiedostosta ja sen jälkeen Guin välityksellä laskea objektien sijainteja sekä pitää huolta pelitilanteesta. GameField siis sisältää listat GameObject-oliosta (ja sen alaluokista) ja päivittää näiden objektien tietoja.

Pääpelin yksittäistä estettä, vihollista ja pelaajaa kuvaavien olioiden yläluokka on GameObject. Staattiset objektit muodostetaan suoraan GameObject-luokasta. Liikkuville objekteille on alaluokka DynamicObject ja edelleen Player ja Enemy. Näin päädyttiin tekemään, koska näillä objekteilla on uniikkeja ominaisuuksia ja toisaalta toistettavan koodin määrä haluttiin minimoida (perinnän käyttäminen). Suunnitelmassa oli tehty eri staattisille objekteillekin omat alaluokkansa, mutta tämä vaikutti turhalta (luokkiin olisi tullut yksi rivi koodia).

Peli päädyttiin siis toteuttamaan siten, että laskenta ja piirtoluokat ovat erillään. DrawDynamic ja DrawStatic (ja sen alaluokat) ovat piirtoluokkia tietyntyyppisille objekteille. DrawDynamic perii QGraphicsPixmapItem-luokan, jotta sen avulla pystytään helposti piirtämään pelihahmo ja viholliset png-kuvista. DrawStatic taas perii valmiin QGraphicsRectItem-luokan, sillä staattiset objektit ovat suorakulmiota. DrawStaticin alaluokat DrawFinish, DrawVisible sekä DrawGround ovat melko turhia: ne sisältävät vain muutaman rivin koodia ja ne olisi helppo toteuttaa osaksi DrawStaticia. Niiden muuttaminen vaatisi kuitenkin myös Guin muuttamista, joten virheiden välttämiseksi uudelleenimplementointia ei tehty projektin loppuvaiheessa. Positiivinen puoli näistä alaluokista on se, että staattisten objektien ulkoasun muuttaminen on tällä hetkellä hyvin helppoa (värin ja tyylin muokkaaminen), eikä se vaadi laajaa perehtymistä muihin toteutuksiin.

Eri piirtoluokkien oliolla on osoitin GameFieldissä olevaan objektiin eli GameObject (DynamicObject ja edelleen Enemy sekä Player) -luokan olioon. Ideana pelin toteutuksessa oli päivittää GameObject olioiden sijaintia absoluuttisena arvona, mikä mahdollisti mahdollisen suoraviivaisen piirron toteutuksen.

GameObject-oliolla on siis sijaintina Position-luokan olio. Näin kaikilla GameFieldistä käsin käsiteltävillä oliolla on vakiintunut tapa ilmasta sijainnit, mikä eliminoi mahdollisia virheitä. Myös sillä, että piirto ja laskenta ovat pitkälti erillään, tavoitellaan vähäisempiä virhetilanteita.

Yksi virheherkimpiä asioita toteuttaa oli automaattinen ruudunpäivitys pääpelin ollessa käynnissä. Tämä toteutettiin Guista käsin adjust_moving_items_coordinates- ja adjust_static_objects_positions -metodeilla. Tässä toteutuksessa virhetilanteita vähennettiin sillä, että GameObjektien sijainteihin (joita käsitellään GameFieldin laskennassa) ei kosketa ollenkaan, vaan muutetaan vain piirtoluokkien muodostamien itemien paikkaa piirtoikkunassa.

Sen jälkeen, kun pääpeli oli toteutettu, tehtiin kenttäeditori, joka muodostaa oman kokonaisuutensa. LevelEditor-olio luodaan Guista käsin, ja sen tehtävänä on pitää kirjaa, mitä objekteja käyttäjä on asettanut kehitteillä olevaan kenttään. Se myös huolehtii, että ns. kursoria päivitetään kenttäeditorissa ja siitä, että käyttäjä ei aseta objekteja paikkoihin, joihin niitä ei saa asettaa. Lisäksi LevelEditor poistaa turhat objektit. Kun käyttäjä on saanut kentän valmiiksi, Guista käsin kutsutaan LevelEditoria kirjoittamaan kenttä tiedostoon.

LevelEditor käyttää objektien sijaintien tallentamiseen listoja, joissa käytetään LevelObject-oliota. LevelObject-olioilla on sijainti sekä tieto siitä, onko kyseinen olio aktiivinen sekä onko olio lisätty Guin piirtoikkunaan. DrawLevelObject toimii analogisesti DrawStaticiin nähden eli se tarjoaa rajapinnan Guin kenttäeditorin kentän piirtämisen sekä yksittäisen LevelObjectin välille. Myös tässä tapauksessa tällaista ratkaisua käytettiin virheiden minimointiin ja se on myös aika selkeä sekä suoraviivainen ratkaisu.

Tärkeimmät metodit

Gui:

select_level(self):

- asettaa graafisen käyttöliittymän kentän valintatilaan
- listaa game_levels-kansiossa olevat kentät
- käynnistää tekstiruudun kentän valintaa varten

set_level(self,reset):

- yrittää asettaa käyttäjän valitseman kentän pääpelin pelattavaksi kentäksi
- informoi käyttäjää, onnistuiko kentän vaihtaminen (eli löytyikö kenttää, ja oliko kenttä rakenteeltaan ok)

cancel_level_select(self), cancel_level_editor(self) ja cancel_game(self):

- palataan päävalikon näkymään
- cancel_game tekee lähinnä tyylimuutoksia, kaksi muuta poistavat tarpeettomat grafiikan piirtoikkunat näkyvistä

game_(self):

- muuttaa ikkunan pääpelin käynnistysvalikkoon päävalikosta

- huom. metodin nimi on huono, eikä edusta muun pelin nimimaailmaa

start_level_editor(self):

- käynnistää kenttäeditorin
- luo grafiikan piirtoikkunan kenttää varten

level_editor(self):

- päivittää graafista käyttöliittymää
- kutsuu LevelEditorin metodeja päivittämään objektien paikkoja kenttäeditorissa
- käynnistää ikkunan kentän tallentamista varten

save_editor_level(self):

- yrittää LevelEditorin kautta kirjoittaa kentän tiedostoon
- informoi käyttäjää siitä, onnistuiko kentän tallentaminen

update_level_editor_screen(self):

- päivittää kenttäeditorin piirtoikkunan ulkoasua
- yrittää pitää ns. kursorin näkyvissä kenttäeditorin ikkunassa

add_level_draw_objects(self):

- lisää piirto-objektit, LevelDrawObject, kaikille LevelEditor-olion LevelObjecteille

determine_key(self,event):

- tunnistaa, jos käyttäjä on painanut jotain pelin kannalta olennaista näppäintä
- välittää näppäinpainalluksen eteenpäin

update_game(self):

- päivittää pääpeliä
- kutsuu GameField-luokan metodeja objektien sijainnin päivittämiseen
- kutsuu tarvittaessa lopetusnäytön (voitto tai häviö) muodostavia metodeja

reset_gui(self):

- alustaa pääpelin, kun pääpeli on päättynyt tai käyttäjä poistunut pääpelistä
- poistaa piirtoikkunan näkyvistä sekä tyhjentää sen

add_DrawGame_objects(self):

- lisää GameFieldin perusteella listoihin DrawDynamic ja DrawStatic tyyppiset oliot, jotka kuvaavat, kuinka kenttä tulee piirtää

adjust_moving_items_coordinates(self):

- yrittää pitää pelihahmon pääpelin piirtoikkunassa
- päivittää liikkuvien objektien piirto-olioiden paikkoja ruudun koon ja pelihahmon nykyisen sijainnin perustella

adjust_static_objects_positions(self):

- päivittää tarvittaessa staattisten objektien piirto-olioden paikkoja samaan tyyliin kuin adjust_moving_items_coordinates päivittää liikkuvia oliota vastaavia piirto-oliota

GameField:

parse_gamefield(self,file):

- muodostaa tiedostosta pelattavan pääpelin kentän
- palauttaa paluuarvona, onnistuiko kentän tekeminen

update_objects_positions(self):

- tarjoaa rajapinnan, jota käyttäen Guista käsin päivitetään kaikkien pääpelin objektien paikat

update_player_pos(self):

- päivittää pelihahmon paikkaa
- hyödyntää käyttäjän näppäimistöpainalluksia

update_enemy_positions(self):

- päivittää kaikkien vihollisolioiden paikkoja

move(self,dynamic_object,direction):

- liikuttaa liikkuvaa objektia haluttuun suuntaan vaakatasossa
- tarkistaa, että objekti ei törmää mihinkään ja estää tarvittaessa objektin liikkeen
- jos vihollinen törmää johonkin muuhun kuin kuiluun tai pelaajaan, kääntää vihollisen suunnan
- jos vihollinen törmää pelaajaan, peli päättyy
- jos pelaaja törmää kuiluun, peli päättyy
- jos vihollinen törmää kuiluun, vihollinen tuhoutuu
- jos pelaaja törmää maaliin, peli päättyy (voittoon)

jump(self):

- käytetään pelihahmon hyppäyttämiseen
- hyppy joko vasemmalle, ylös tai oikealle
- tunnistaa hypyn ylimmän kohdan, ja alkaa tarvittaessa pudottaa pelaajaa
- samat uuden sijainnin tarkistukset kuin move:ssa, mutta nyt viholliseen osuminen tuhoaa vihollisen

fall(self,dynamic_object):

- pudottaa liikkuvaa objektia staattisten objektien tasolle (painovoimametodi)
- vihollinen voi pudota vain suoraan alaspäin
- pelaaja voi pudota lisäksi vasemmalle tai oikealle

- tarkistukset uudesta sijainnista samaan tyyliin kuin jump-metodilla

check_position(self,dynamic_object,x_min,x_max,y_min,y_max)/
check_position_jump(self,dynamic_object,x_min,x_max,y_min,y_max, jump_x)/
check_position_fall(self,dynamic_object,x_min,x_max,y_min,y_max):

- tarkistavat, että sijainti on vapaa
- palauttavat sijainnissa olevan objektin tyypin tai tyhjä, jos sijainnissa ei ollut objektia
- hyvin samanlaisia, mutta pieniä eroja (toteutettu huonosti)

check_pos(self,pos,x_min,x_max,y_min,y_max,speed) / check_pos_jump(self, pos, x_min, x_max, y_min, y_max, jump_x, jump_y) /
check_pos_fall(self,pos,x_min,x_max,y_min,y_max,fall_y,object_type):

- tarkistavat, menevätkö pos määrittelemä paikka ja x_min, x_max, y_min, y_max (eli liikkuvan objektin tuleva sijainti) päällekkäin
- palauttavat False, jos sijainnit päällekkäin, muuten True
- kutsutaan check_position -metodeista ja jatkavat samaa huonoa toteutusta

get_enemy_object_at_position(self,x_min,x_max,y_min,y_max, comparison):

- palauttaa vihollisobjektin annetusta sijainnista, jos vihollista ei ole vielä tuhottu

get_enemy_below(self,x_min,x_max,y_min,comparison, y_value) / get_player (self,x_min,x_max,y_min, y_value) :

- palauttavat vihollisen tai pelihahmon, joka sijaitsee suoraan sijainnin alapuolella
- kuitenkin niin, että palautettava objekti on korkeammalla kuin y_value

set_object_to_ground_level(self,dynamic_object,level):

- käytetään tiputtamaan liikkuva objekti suoran staattisen objektin päälle
- tuhoaa mahdollisen liikkuvan objektin, joka sijaitsee staattisen objektin päällä
- jos vihollinen tippuu pelihahmon päälle, peli päättyy
- jos pelihahmo tai toinen vihollinen tippuu vihollisen päälle, vihollinen tuhoutuu
- täydentää fall-metodia
- olennainen silloin, kun asetusarvot tippumisen ja hypyn suhteen eivät sovi täysin yhteen objektien koon kanssa

Config:

read_config(self):

- yritetään avata config.txt -tiedosto kansiosta game_config
- luetaan kaikki asetusarvot tiedostosta
- jos asetusarvot eivät ole speksien mukaiset, käytetään osittain oletusarvoja
- palautetaan 1, jos kaikki onnistui, 2 jos jouduttiin käyttämään oletusarvoja ja 0, jos tapahtui fataali virhe (tiedostoa ei esim. löytynyt)

check_values(self,value,max_value, min_value, value_type):

- suoritetaan tyyppimuunnos merkkijonosta kokonais- tai liukuluvuksi
- tarkastetaan, että arvo on annetulla välillä
- palautetaan True, jos arvo oli ok, ja muulloin False

DrawDynamic:

move_item(self):

- liikutetaan piirto-olio vastaavan DynamicObject-olion sijaintiin

DrawStatic:

- ei mitään muita metodeja kuin init, joka alustaa QGraphicsRectItemin oikeaan sijaintiin
- lisäksi init muodostaa rajapinnan GameObject-olioon
- alaluokissa ei mitään kovin olennaista, pitkälti olioiden värien asettamista. Poikkeuksena DrawGround, jossa myös määritellään piirto-olion korkeus

GameObject:

set_position_object(self, pos_object):

- apumetodi, joka asettaa Position-olion GameObject-oliolle

DynamicObject:

set_new_position(self,x_min,x_max,y_min,y_max):

- liikuttaa olion uuteen sijaintiin
- käytetään GameFieldistä käsin olioiden sijainnin päivittämiseen

Enemy:

turn_direction(self):

- kääntää vihollisen suunnan

Player:

set_jump_direction(self,direction):

- asettaa pelaajan hypyn suunnan
- kutsutaan GameFieldistä

Position:

set_position(self,x,level,object_type, Distance_y):

- laskee staattisten objektien lukumäärän ja kentän tyhjien rivien perusteella objektin sijainnin
- ottaa huomioon objektin tyypin

LevelEditor:

update_position(self):

- päivittää ns. kursorin (nykyisen objektin) paikkaa käyttäjän näppäinpainallusten perusteella

update_all_positions(self):

- Guista käsin kutsuttuna päivittää kaikkia LevelObjekteja sekä luo uusia LevelObject-oliota

add_new_objects(self):

- lisää käyttäjän näppäinpainallusten perusteella oikeanlaisen LevelObject-olion
- asettaa olion oikeaan paikkaan
- tarkastaa, että paikassa ei ole ennestään oliota
- jos vanha olio löytyi, tekee oliosta epäaktiivisen

get_object_at_pos(self,x,y):

- palauttaa parametrina annetussa sijainnissa olevan olion
- käy kaikki LevelObject-oliot läpi

create_file(self,file_name):

- kokeilee, onko parametrina annettu kentän nimi uniikki
- jos on uniikki, avaa sen nimisen tekstitiedoston ja kirjoittaa siihen apumetodin avulla käyttäjän kenttäeditoriin luoman tason
- jos nimi ei ole uniikki, kirjoittaa tiedot standardinimiseen tiedostoon, joka on nimetty my_level + juokseva numero (joka saadaan confix.txt)

write_new_level_number(self):

- kirjoittaa config.txt perään päivitetyn level_number-arvon

LevelObject

set_pos(self,x,y):

- asettaa olion parametrina annettuun sijaintiin

DrawLevelObject

set_color(self):

- asettaa QGraphicsRectItemille oikean värin pohjautuen LevelObject-olion tyyppiin

adjust_positon(self,editor_screen,win_width,x):

- asettaa olion oikeaan paikkaan ottaen huomioon kursorin sijainnin kenttäeditorissa

6. Algoritmit

Tässä projektissa ei käytetty mitään tunnettuja algoritmeja, eivätkä pelissä toteutetut algoritmit ole perusperiaatteeltaan kovin monimutkaisia.

Tärkeimmät algoritmit liittyvät tässä projektissa törmäyksen tunnistukseen, minkä olisi toimittava mahdollisimman virheettömästi tai muuten pelihahmot pystyvät liikkumaan objektien läpi. Tässä tapauksessa suuri ongelma oli se, että käyttäjälle annetaan mahdollisuus muuttaa liikkumiseen liittyviä arvoja, joten algoritmin pitää pystyä mukautumaan erilaiseen liikenopeuteen.

Törmäyksen tunnistusalgoritmit (GameField check_pos -algoritmit) toteutettiin, kuten teknisessä suunnitelmassa oli suunniteltukin. Jokaisella pelikentän oliolla on oma sijainti, joka on ilmastu objektin vasemman reunan, oikean reunan, yläreunan ja alareunan avulla. Nämä neljä pistettä yksikäsitteisesti määrittelevät suorakulmion mallisen objektin. Törmäyksen tunnistus perustuukin siihen, että tarkastetaan, onko objektia rajaavan suorakulmion reunat toisen objektin reunasuorakulmion sisällä. Törmäykseen tässä tapauksessa siis riittää, että yksikin reuna on toisen objektin sisällä. Asetusarvojen suhteellisen vapaata määrittelyä ajatellen tulee vielä huomioida se, että pelihahmon liike voi olla niin suurta, että se menee yhdellä liikesyklillä toisen objektin läpi. Näin ollen tulee myös käyttää tietoa liikenopeudesta ja tarkastaa, että nyt esimerkiksi staattisen objektin oikealla puolella oleva pelihahmo ei ollut ennen liikesykliä staattisen objektin vasemmalla puolella.

Varsinainen törmäyksen tunnistustoteutus ei ole kovin hienostunut, vaan se pohjautuu siihen, että käydään kaikki pelikentän objektit läpi ja tarkastetaan objektien reunat. Tästä seuraa ehtotarkastelu ja iteraatio kaikkien objektien yli. Toteutustavan heikkous onkin se, että siinä ei ole mitään alkukarsintaa objekteista, vaan kaikki objektit tarkastetaan, kunnes löydetään kohdassa oleva objekti tai todetaan, että sijainti on vapaa. Tämä on siis hyvin hidas toteutustapa, jos kenttä on iso. Toteutusta saisikin tehostettua lisäämällä alkukartoituksen liikkuvan objektin sijainnista ja tarkistamalla vain lähellä olevia objekteja.

Nykyisen toteutuksen hyvä puoli on kuitenkin se, että se on toimintavarma (tarkastaa kaikki objektit) ehdolla, että tunnistus todella toimii. Toinen toteutuksen ongelma olikin siinä, että kaikkia erilaisia tapoja sille, miten kaksi suhteellisen vapaasti liikkuvaa suorakulmiota voivat olla osittain päällekkäin, oli vaikea keksiä.

Törmäyksen tunnistuksen olisi voinut toteuttaa myös esimerkiksi laskemalla riittävällä näytteenottovälillä kaikki suorakulmion sisään jäänet pikselit ja vertaamalla niitä muiden objektien vastaaviin. Tämä olisi kuitenkin ollut erittäin hidasta, minkä takia tällaiseen toteutukseen ei päädytty. Myöskään mikään ennalta määrättyihin sijainteihin perustuva algoritmi ei toimisi pääpelin törmäyksen tunnistuksessa, sillä liikkumisvakioita voi muuttaa niin vapaasti. Tällainen algoritmi, jossa vain tarkastetaan, että esimerkiksi kohta (2,5) on vapaa, on kuitenkin käytössä LevelEditorissa. Kenttäeditorissa kaikki objektit ovat samanlaisia ja joiden sijoittaminen on vakiintunutta (voi sijoittaa vain tiettyihin tasavälisiin kohtiin), joten erittäin yksinkertainen algoritmi toimii hyvin. LevelEditorissa siis tarkastetaan, että sijainnissa, johon käyttäjä haluaa laittaa objektin ei ole ennestään objektia (ja jos on, niin vanha poistetaan).

Pelin kannalta olennainen algoritmi on myös se, minkä avulla päivitetään ruutua pääpelissä (eli adjust_moving_items_coordinates ja vastaava toinen metodi staattisille objekteille). Algoritmi siis yrittää pitää pelaajahahmon näkyvissä silloin, kun hahmo liikkuu ruudun piirtoikkunan reunalle tai, kun peliruudun kokoa muutetaan. Tämän algoritmin toiminta perustuu siihen, että tarkastetaan jokaisella pääpelin päivityssyklillä, onko pelaaja ruudun oikeassa reunassa. Jos on, lisätään niin sanottua ruutulukemaa yhdellä. Jos puolestaan pelaaja on ruudun vasemmassa reunassa ja ruutulukema ei ole nolla, niin vähennetään ruutulukemaa yhdellä. Tämän jälkeen, jotta pelaaja saadaan näkymään ruudulla, vähennetään kaikkien piirto-olioiden sijainneista ruudun koko painotettuna ruutulukemalla. Nyt ruudussa näkyy pelaajan lisäksi ne objektit joiden kuulukin näkyä. Kuten jo Ohjelman rakenne -kohdassa mainittiin, tässä algoritmitoteutuksessa olennaista oli virheiden minimointi, minkä vuoksi käytännön toteutuksessa päivitetään vain piirto-olioiden sijainteja eikä laskennan suorittavien GameObject-olioiden sijaintia.

Ruudun päivittämiseen toinen ratkaisu olisi ollut pitää pelihahmoa koko ajan ruudun keskellä ja vain muuttaa sitä, mitkä objektit näkyvät. Pohdittiin edellä mainittua ratkaisua, jota näkee myös joissakin tasohyppelypeleissä käytettävän, mutta tultiin siihen tulokseen, että valitussa ratkaisussa peli näyttää paremmalle ja pelattavuuskin on luultavasti parempi mahdollistaen monimutkaisemmat kentät.

7. **Tietorakenteet**

Projektissa lähtökohtana oli se, että kentät voivat sisältää vaihtelevan märän erilasia objekteja. Näin ollen kentän käsittelyyn aika lailla ainoa järkevä vaihtoehto oli käyttää dynaamisesti muuttuvia tietorakenteita.

Kentän objektien tallentamiseen päädyttiin käyttämään listoja. Myös sanakirja olisi ollut hyvä valinta ja, jos törmäyksen tunnistusta haluaisi tehostaa, siirtyminen osittain sanakirjan käyttöön voisi auttaa (pystyy nopeasti etsimään tietyllä avaimella objekteja). Lista valittiin objektien päätietotypiksi alun perin pitkälti sen takia, että sen käyttö on yksinkertaista ja useimmiten riittävän tehokasta.

LevelEditor-luokassa kentän tallentamisen apuna käytettiin lisäksi kaksiulotteista listaa (sort_objects -metodi tekee tämän listan). Käyttäjän kenttäeditorissa kenttään sijoittamat objektit siis asetettiin omilla symboleillaan riveittäin ja sarakkeittain listaan, josta ne olivat sitten helppo suoraan indekseittäin kirjoittaa tiedostoon. 2-ulotteinen lista muodostettiin laskemalla, mikä on suurin rivi ja mikä suurin sarake kenttäeditorissa, joissa objekteja sijaitsee. Tehtiin tyhjä lista ja tämän jälkeen lisättiin tyhjään listaan sarakemäärän mukaiset uudet ”listat” jokaiselle riville erikseen, kunnes rivien määrä vastasi kenttäeditorin rivimäärää. Ei siis käytetty vain kahta listaa ja lisätty toista listaa ensimmäisen listan n:lle riville, koska tämä olisi muuttanut jokaista riviä, kun vain yhtä riviä olisi haluttu muuttaa (jokaisen rivin ns. lista olisi osoittanut samaan muistiosoitteeseen).

8. Tiedostot

Pelissä on ensinäkin asetusarvotiedosto (tekstitiedosto), jonka tulee sijaita nimellä config.txt kansiossa game_config (Tasohyppely_peli/game_files/game_config/config.txt). Koska tiedosto on niin olennainen, game_config-kansiossa on lisäksi backup-tiedosto nimeltä config_backup.txt. Eli jos alkuperäiselle config-tiedostolle tapahtuu jotakin, pitää config_backup.txt uudelleennimetä config.txt ja pelin saa jälleen käynnistymään.

Ehdot asetusarvotiedoston rakenteelle (kts. config.txt, jossa kommenteissa kattavammat speksit):

- Kommentit alkavat %-merkillä heti rivin alussa (ohitetaan)
- Whitespace ohitetaan lähtökohtaisesti kaikilta riveiltä
- Asetusarvot annetaan muodossa asetusarvo_nimi : arvo (jokaiselle arvolle oma rivi, eikä rivillä saa olla muuta, pois lukien whitespace)
- level_number pitää olla aina määritetty (muuten peli ei käynnisty ollenkaan) ja tätä arvoa ei pitäisi muuttaa käsin (oletusarvona 1 ja peli kirjoittaa tiedoston perään uusia arvoja)
- Suurin osa muutettavista arvoista vaativat kokonaisluvun, muutamassa desimaaliluku
- Jos parametreja ei anneta tai parametreissa vikaa, peli käyttää oletusarvoja (huom. arvojen pelattavuutta ei testata)
- Parametrit annetaan vain kertaalleen (pois lukien level_number, kaikkien muiden parametrien osalta tätä ei myöskään vaadita)
- Muutettavat parametrit
    - distance_x: paljonko tilaa pikseleinä ruudun vas. reunassa
    - distance_right_limit: tämä vaikuttaa pääpelin ruudunpäivitykseen (ei luultavasti kannata muuttaa, ellei halua jotain erikoista ruudunpäivitystä)
    - empty_line_height: kuinka monta pikseliä tyhjät rivit kentässä vaativat (ei kannata muuttaa ainakaan paljoa)
    - static_object_height: staattisten objektien korkeus pääpelissä pikseleinä. Pitää määrittää, jos aikoo määrittää hyppy- tai pudotusarvoja tai dynaamisten objektien korkeutta

- static_object_width: staattisten objektien leveys pääpelissä pikseleinä. Pitää määrittää, jos aikoo määrittää dynaamisten objektien leveyttä
- enemy_height: vihollisen korkeus pääpelissä pikseleinä (< static_object_height)
- enemy_width: vihollisen leveys pääpelissä pikseleinä (<= static_object_width)
- player_width ja player_height (samat vaatimukset kuin vihollisen vastaavilla arvoilla)
- player_speed: kuinka paljon pelaaja liikkuu yhdellä näppäinpainalluksella pikseleinä (isot arvot eivät ole pelattavia)
- player_jump_x: kuinka paljon pelaaja liikkuu yhden sivuttaishyppysyklin aikana pikseleinä
- player_jump_y: kuinka paljon pelaaja liikkuu ylös yhden hyppysyklin aikana (<= static_object_height)
- player_jump_max_height: kuinka korkealle pelaaja hyppää pikseleinä. Hyppykorkeus riippuu myös jump_y-arvosta. (isot arvot -> pelaaja hyppää pois näkyvistä, <= 1000)
- player_fall_y: painovoimavakio pelaajalle pikseleinä (1 – static_object_height)
- player_fall_x: kuinka paljon pelaaja liikkuu sivuttaispudotuksessa (jos haluaa paraabelisen hypyn lentoradan, käytä samaa arvoa kuin player_jump_x)
- enemy_speed: vihollisen nopeus pikseleinä, vrt. player_speed
- enemy_fall_y: kuinka nopeasti viholliset putoavat, samat ehdot kuin player_fall_y
- invisible_object_height: kuinka korkealle kuilut tulevat maaobjekteista, ei pitäisi olla suurta tarvetta muuttaa

Toinen tiedostotyyppi, jota peli käsittelee ovat kenttätiedostot, jotka ovat myös tekstitiedostoja. Pelattavissa olevat kenttätiedostot ovat kansiossa game_levels (Tasohyppely_peli/game_files/game_levels). Lisäksi testikenttiä löytyy test_levels-hakemistosta (game_files alla) ja periaatteessa näitäkin kenttiä pystyy pelaamaan, jos ne siirtää game_levels-kansioon. Huomioitavaa on kuitenkin se, että osa näistä kentistä on

tarkoituksella rikkinäisiä (ja lisäksi osassa on ääkkösiä kommenteissa, joita ei ole koodattu Unicodella -> saattaa kaataa koko pelin).

Kenttätiedostoissa kommenttirivi merkitään %-merkillä heti rivin alussa, ne rivit ohitetaan parsimisessa.

Tiedoston alussa ovat tyhjät rivit (ei kommentoidut) kertovat ylimpien objektien etäisyyden piirtoalueen yläreunasta (jos liian suuri, niin ei enää kasva).

Symbolit kenttätiedostoissa:

- - on tyhjä ruutu
- O tai o normaali este
- E tai e on vihollinen
- P tai p on pelaaja (oltava ainoastaan yksi kappale)
- W tai w on maaliobjekti (oltava vähintään yksi)
- F tai f on kuilu, jonka saa asettaa vain alimmalle riville, jolla on objekteja

Jokaisella rivillä tulee olla sama määrä merkkejä (pois lukien whitespace), jotta kenttä voidaan muodostaa tiedostosta.

Huom. jos pelin suorituksen aikana muuttaa valittua kenttää toimivasta ei-toimivaksi (ajonaikainen muokkaus ei muutenkaan ole suotavaa), kenttää ei enää luonnollisesti pysty pelaamaan, vaikka se olisikin ennestään valittuna.

Lisähuomiona, kenttäeditori on puolestaan toteutettu siten, että se suostuu tallentamaan vain kelvollisen kentän.

Esimerkki kenttätiedostosta:

```
- - - - - - - - - - - - - - - - - - - - - - O O - E - O O - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - O O - - O O O - - O O O - - - - - - - - - - - - - - - - - - - - - - -
- - - - - O O - - - - - - - - - - O O - - - - - - - - - - - - - O O - - - - - - - - - - - - - O O - - - - - -
P - - O O - - O O - E - - O O O - - - - - - - - - - - - - - - - - - F - - O O E - - - - O O - - O O - F - W
```

9. **Testaus**

Ohjelman testauta yritettiin tehdä aina sen jälkeen, kun yksittäinen kokonaisuus oli tehty. Kuitenkin etenkin GameField-luokan testaus ajoittui projektin loppupuolelle. Syynä tähän

oli muun muassa se, että toimivan kenttäeditorin avulla testikenttiä oli helppo ja nopea tehdä verrattuna käsin tekemiseen.

Suunnitelman mukaisesti ohjelman tekeminen aloitettiin kentän parsimismetodista. Tätä metodia oli suunniteltu testattavan vain muutamalla kentällä, mutta koska idea vain valmiiksi tehdyistä kentistä hyllytettiin, vaadittiin laajempaa testausta. test_files-kansiosta löytyy kyseisen metodin yksikkötesti parse_gamefield_testing.py. Kuitenkin tämä yksikkötestaus osoittautui hieman puutteelliseksi, sillä vielä lopputestauksessa metodin toiminnasta virhetilanteiden käsittelyssä löytyi muutama bugi.

Kun pelikentän parsiminen oli saatu kohtalaisen toimivaksi (yksikkötestistä nähtiin, että objektien paikat saadaan luettua tiedostosta oikein), toteutettiin Guihin suunnitelman mukaisesti alkutilanteen piirto. Sitten tätä piirtoa testattiin ja kehitettiin sillä oletuksella, että GameFieldin GameObject-oliolla on oikeat sijainnit. Syntaksivirheillä oli kuitenkin taipumus jäädyttää koko ruutu ilman virheilmoitusta. Tätä varten Guista toteutettiinkin testiversiota (osa löytyy test_files-hakemistosta), joissa kaikki olennaiset ja kehitteillä olevat metodit kutsuttiin suoraan initistä. Näin saatiin selvät virheilmoitukset syntaksivirheistä, jolloin niiden korjaaminen oli suhteellisen helppoa.

Projektin aikana logiikkavirheiden korjaamiseen käytettiin pitkälti erilaisia printtejä, koska testiohjelman, joka liikuttaisi pelaaja läpi kenttää ja monella eri tavalla, tekeminen olisi melko työlästä. Näitä printtejä käytettiin muun muassa näppäinpainallusten välittymisen testaukseen, kuten oli suunniteltukin.

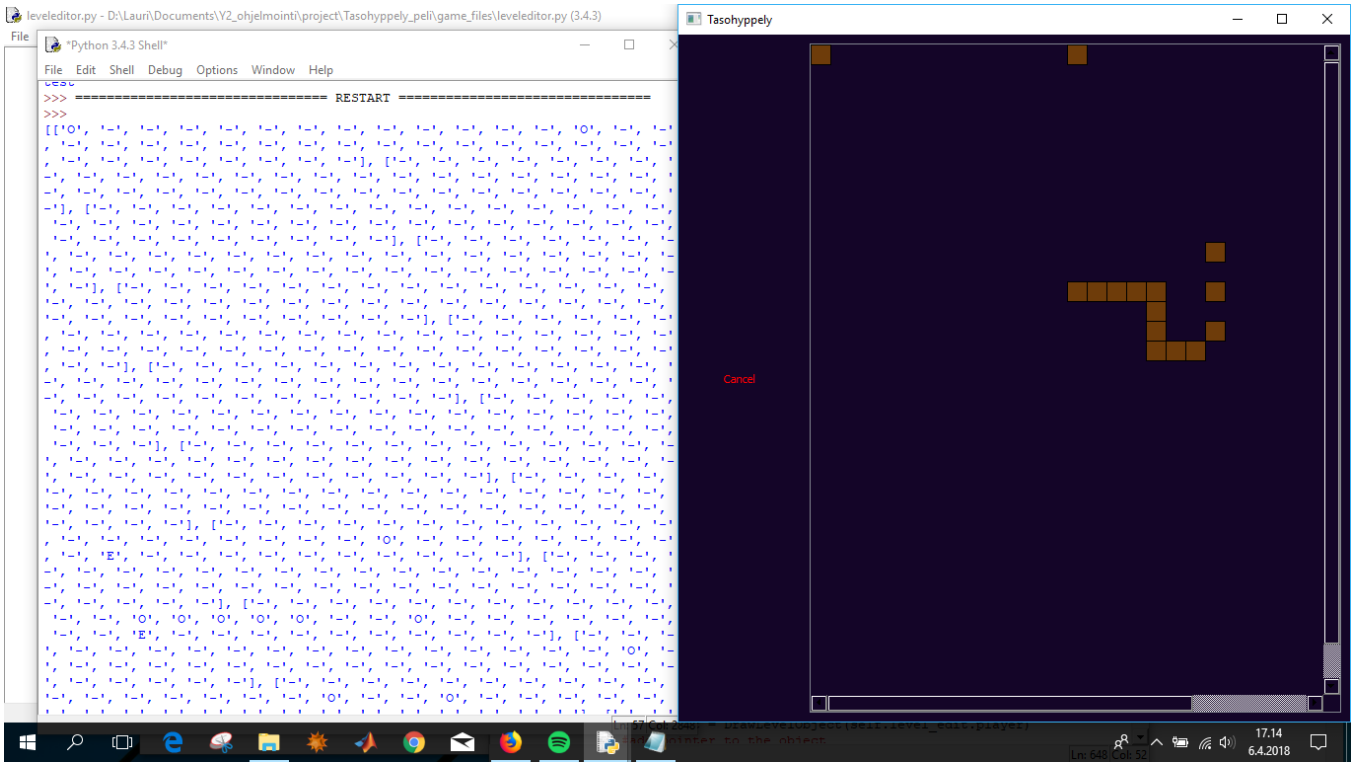
Kenttäeditorin toteutuksen testaamista varten tehtiin jälleen yksikkötesti (test_files/test_level_saving.py), jonka avulla saatiin kenttäeditorin kentän tallennus toimivaksi. Kuitenkin kenttäeditorinkin testaus oli pitkälti kokeilua: tehdään paljon erilaisia kenttiä ja katsotaan, tapahtuuko piirto-ikkunassa mitään ihmeellistä (Kuva 8). Jos ja kun virheitä löytyi, yritettiin printtien avulla paikantaa virheen aiheuttaja.

Lopputestauksessa testattiin laajalti erilaisia kenttiä, jotka tehtiin pääasiassa kenttäeditorilla, ja yritettiin löytää pääpelistä virheitä muuttamalla asetusarvotiedoston arvoja. Virheitä löytyikin lopulta melko järkyttävä määrä, ja ne on yritetty jälkikäteen paikata, mikä valitettavasti näkyy GameField-luokan koodin laadussa.

Lisäksi erilaisten virhetilanteiden eston toimintaa on testattu suhteellisen laajasti. On esimerkiksi yritetty kaataa peliä tuhoamalla tiedostoja ajon aikana, yritetty valita kentän valinnassa kentäksi aivan vääränlaista tiedostoa, yritetty sekoittaa pääpelin ruudun päivitystä muuttamalla ikkunan kokoa koko ajan ja niin edelleen.

Peli on kirjoitettu Windows-koneella, mutta sitä on lisäksi testattu kahdella Linux-koneella. Näin testaamalla ja korjaamalla on päästy eroon esimerkiksi ongelmista hakemistossa liikkumisessa: ei käytetä enää kenoviiva (\) tai kauttamerkkiä (/) hakemistossa liikkumiseen, vaan sama asia on toteutettu muuten.

Ohjelma on loppuen lopuksi läpäissyt sille asetetut testit, eikä sen pitäisi kaatua. Kaikilla asetusarvoyhdistelmillä ei ole kuitenkaan testattu ja sieltä saattaa hyvinkin löytyä yhdistelmiä, jotka rikkovat pääpelin. Kokonaisuudessaan testaus on kuitenkin ollut melko laajaa, laajempaakin kuin suunnitelmassa.



Kuva 8: LevelEditorin testausta, jossa tarkastetaan, että objektit tulevat listassa oikeisiin kohtiin

10. Ohjelman tunnetut puutteet ja viat

GameField-luokka on toteutettu paikoin surkeasti. Tästä johtuen on mahdollista, että pääpelistä löytyy vielä bugeja, joita ei ole aiemmin havaittu.

GameFieldin toteutus on vaikeaselkoisuuden (useita hyvin samanlaisia metodeja) lisäksi hyvin epäoptimoitu. Tämä puolestaan johtaa siihen, että suuret kentät ovat kelvottoman hitaita. Syynä tähän on toistuvat iteraatiot kaikkien objektien yli jokaista liikkuvaa objektia kohden. Viholliset liikkuvat jatkuvasti, joten jos niitä on paljon ja staattisia objekteja on paljon, ruudunpäivitysnopeus putoaa muutamaan ruutuun sekunnissa (pelikelvoton).

Toteutusta saisi optimoitua siten, että staattisista objekteista tarkastettaisiin vain lähellä olevat. Tähän taas voitaisiin käyttää esim. 2-ulotteista listaa tai sanakirjaa staattisista objekteista. Sitten laskettaisiin liikkuvan objektin suhteellinen sijainti (absoluuttinen sijainti / koko kentän leveys) ja testattaisiin lähellä olevien staattisten objektien (voisi poimia 2-ulotteisesta listasta suoraan ilman iteraatiota) absoluuttiset sijainnit suhteessa liikkuvaan objektiin. Vihollisten osalta testatusta on vaikeampi optimoida, jos viholliset liikkuvat jatkuvasti. Kuitenkin tällä hetkellä aivan turhaan esimerkiksi pudotetaan tuhoutuneita vihollisia, vaikka ne ovat jo törmänneet staattiseen objektiin (eli käydään siis fall-metodissa laskemassa arvoja ja todetaan, että objekteja ei voi enää pudottaa). Yleisesti ottaen siis looppeja on aivan liikaa, ja niistä osan voisi suhteellisen helpostikin korvata.

11. **3 parasta ja 3 heikointa kohtaa**

Ohjelma on kokonaisuudessaan aika hyvä. Se tarjoaa pääpelin, kenttäeditorin ja kentän valinnan. Kun vielä huomio asetusarvotiedoston, käyttäjällä on aika paljon mahdollisuuksia pelata erilaisia kenttiä ja siten kuin haluaa. Peli on myös graafisen käyttöliittymän tyylin suhteen melko yhtenäinen.

Kenttäeditori on toteutettu hyvin: tekee uusien kenttien tekemisestä suhteellisen helppoa. Lisäksi kenttäeditori on tietyssä mielessä älykäs ja estää esimerkiksi käyttäjää luomasta virheellisiä kenttiä.

Huonona puolena on etenkin GameField-luokan toteutus (käsiteltiin jo pitkälti osiossa 10). GameField-luokan koodin laatu on heikkoa, mikä johtuu siitä, että sitä jälkikäteen paikattiin laajasti. Lisäksi tiedossa oli jo silloin, että kyseinen luokka pitäisi tehdä pitkälti kokonaan uusiksi, jos siitä haluaisi oikeasti hyvän.

DrawVisible, DrawGround ja DrawFinish ovat myös aika huonoja luokkia ja ne pystyisi helposti suoraan sisällyttämään DrawStatic-yläluokkaan.

12. **Poikkeamat suunnitelmasta**

Työ toteutettiin huomattavasti laajempana kuin suunnitelmassa oli alun perin tarkoitus tehdä. Tämä luonnollisesti aiheutti suuriakin muutoksia. Esimerkiksi kenttäeditoria tai asetusarvotiedostosta lukemista ei ollut alkuperäisessä suunnitelmassa ollenkaan. Kenttäeditori tosin suunniteltiin kuitenkin melko perusteellisesti (suunnitelma löytyy projektin päiväkirjan, joka on liitteenä, lopusta).

Siltä osin, mitä oli suunniteltu, muutoksia tuli piirtoluokkiin ja peliobjekti luokkiin. Suunnitelmassa oli yksi piirtoluokka, DrawGame, joka ei oikein ollut toteuttamiskelpoinen ratkaisu, koska pelissä käytetään sekä kuva- että suorakulmio-objekteja. Sitten StaticObject ja alaluokat päätettiin karsia, koska niihin olisi tullut vain muutama rivi koodia. Lisäksi kenttäeditoria ja yleisestikin ottaen helpompaa kentän luomista varten luovuttiin kahdesta eri vihollistyypistä (nopeat ja hitaat viholliset).

Toteuttamisjärjestys ja ajankäyttöarviot menivät hyvinkin oikein siltä osin, mitä peli oltiin suunniteltu tehdä. Suunnitelmaan tulleet lisäykset tietysti vaativat huomattavastikin lisää aikaa ja niiden toteutus sijoittui projektin loppupuolelle.

13. **Toteutunut työjärjestys ja aikataulu**

Tarkka projektin toteutusjärjestys löytyy projektin päiväkirjasta, joka on liitteenä. Koska peli tehtiin aiottua laajempana, toteutettiin peli suunnitelmaa nopeampaan tahtiin. Alla lyhyesti:

23.2.2018: Monien luokkien alustavat muodot

24.3.2018: Guin alustus ja parse_gamefield-metodin alustava toteutus

25.2.2018: Guhin lisää metodeja, parse_gamefield korjausta

26.2.2018: Suunnittelua, DrawGamen hylkäys

1.3.2018: Guihin pelin alkutilanteen piirto

2–3.3.2018: GameField toteuttamista

4.3.2018: Dynaaminen ikkunan skaalaus Guihin

5.3.2018: Bugikorjauksia

7.3.2018: Tuki sivuttaissiirtymiselle pudotukseen

9.3.2018: Bugikorjauksia GameFieldiin

10–11.3.2018: Config-luokan tekeminen

16.3.2018: Pelilooppi eheäksi

17.3.2018: Help-ruudun lisäys ja messsageboxeja tärkeistä ilmoituksista

18.3.2018: Edellisen päivän toteutusten bugien korjaus

19.3.2018: Testaus Linux ja muutoksia hakemistoliikkumiseen

23.3.2018: Kenttäeditorin suunnittelu

24.3.2018: Kenttäeditorin luokkien toteutusta

25.3.2018: Tason tallennusta varten Guihin uusia metodeja

6.4.2018: LevelEditoriin kentän tallennus

7–9.4.2018: Testausta ja paljon korjauksia, osa hyvin pieniä

12.4.2018: Kenttäeditoriin ruudunpäivitys

13.4.2018: Pääpelistä poistuminen esc:llä. Paljon bugeja korjattu GameFieldistä

14.4.2018: GameFieldin paikkaaminen jatkuu

15.4.2018: Bugin korjaus ruudunpäivityksestä pääpelissä

16.4.2018: Repon siistimistä, muutettu README ja korjattu config.txt selityksiä

17.4.2018: Luettu filet läpi ja tehty pienehköjä korjauksia sekä muutettu kommentteja

18–20.4.2018: Pieniä bugikorjauksia parse_gamefield, ulkoasukorjauksia ja dokumentaation kirjoittamista

14. **Arvio lopputuloksesta**

Ensinäkin ohjelma on laaja tehtävänantoon suhteutettuna. Suurimmaksi osaksi ohjelma toimii hyvin, esimerkiksi erilaiset virhetilanteet on huomioitu perusteellisesti. Pääpelin pelattavuus on suuria kenttiä lukuun ottamatta hyvä ja peli tarjoaa käyttäjälle melko paljon valinnaisuutta kenttien luomisen ja asetusarvojen muuttamisen kautta. Käyttöliittymä on myös ihan hyvä informatiivisuudeltaan sekä ulkoasultaan.

Luokkajakoa on jo pohdittu aika laajalti aiemminkin. Yleisesti voi sanoa, että peli on jaettu suhteellisen hyvin osakokonaisuuksiin. Kuitenkin muutama turha piirtoluokka löytyy ja toisaalta taas Guista olisi voinut irrottaa osan omaksi luokakseen.

Tulevaisuudessa ohjelmaa pitäisi parantaa GameField-luokan osalta. Tästä luokasta pitäisi tehdä huomattavasti parempi niin koodin laadun kuin tehokkuuden suhteenkin. Tehokkuuden kannalta listojen osittainen muuttaminen toisiin tietotyyppeihin ja tarpeettomien toistorakenteiden karsiminen auttaisi paljon.

Ohjelmaan olisi mahdollista lisätä suhteellisen pienellä vaivalla uusia toiminnallisuuksia, esimerkiksi jo olemassa olevan kentän muokkaus kenttäeditorilla. Toisaalta kuitenkin esimerkiksi ohjelman ulkoasun perusteellinen muuttaminen olisi vaikeaa, vaikka päävalikon ulkoasua muutettiinkin aivan projektin loppuvaiheessa. Graafisen käyttöliittymän toiminta on nimittäin pitkälti sidottu valmiisiin layoutteihin, jotka eivät anna kovin paljoa kustomointimahdollisuuksia.

Uusien staattisten objektien (esim. kentältä kerättävät aarteet yms.) lisääminen ei olisi kovin vaikeaa. Myös aarteiden keräämiseen tai aikaan perustuvan ennätyslistauksen pystyisi toteuttamaan, mutta muutettavat asetusarvot osaltaan rikkoisivat tämän ennätyskirjanpidon.

15. **Viitteet**

- https://en.wikipedia.org/wiki/Collision_detection
- https://en.wikipedia.org/wiki/Platform_game
- www.stackoverflow.com
    - https://stackoverflow.com/questions/38507011/implementing-keypressevent-in-qwidget
    - https://stackoverflow.com/questions/41184719/pyqt5-how-to-emit-signal-from-worker-tread-to-call-event-by-gui-thread
    - https://stackoverflow.com/questions/39819700/replacing-the-existing-mainwindow-with-a-new-window-with-python-pyqt-qt-design
    - https://stackoverflow.com/questions/16997729/adding-qpixmap-item-to-qgraphicsscene-using-qgraphicssceneadditem-crashes-pysi
    - https://stackoverflow.com/questions/24659239/how-to-change-qpushbutton-text-and-background-color/24671124
    - https://stackoverflow.com/questions/21802868/python-how-to-resize-raster-image-with-pyqt
    - https://stackoverflow.com/questions/24226792/removing-the-last-character-from-a-string-in-python
    - https://stackoverflow.com/questions/3377439/how-to-change-the-color-of-a-qgraphicstextitem
    - https://stackoverflow.com/questions/21328030/how-to-set-text-color-in-qlineedit-when-background-image-is-set-for-qlineedit
    - https://stackoverflow.com/questions/40004672/how-to-go-the-start-of-file-in-python
    - https://stackoverflow.com/questions/3207219/how-do-i-list-all-files-of-a-directory
    - https://stackoverflow.com/questions/27320268/how-to-display-text-in-main-window
    - https://stackoverflow.com/questions/42013674/qt-remove-layout-from-other-layout
    - https://stackoverflow.com/questions/16105349/remove-scroll-functionality-on-mouse-wheel-qgraphics-view
    - https://stackoverflow.com/questions/13111669/yes-no-message-box-using-qmessagebox
    - https://stackoverflow.com/questions/6667201/how-to-define-a-two-dimensional-array-in-python
    - https://stackoverflow.com/questions/696209/non-resizeable-qdialog-with-fixed-size-in-qt
    - https://stackoverflow.com/questions/5143388/why-do-images-appear-at-incorrect-positions-in-my-qgraphicsview-derived-class
    - https://stackoverflow.com/questions/18316710/frameless-and-transparent-window-qt5
- http://pyqt.sourceforge.net/Docs/PyQt5/ (lukuisia dokumentaatiosivuja)
- http://doc.qt.io/qt-5/ (lukuisia dokumentaatiosivuja, päälähde)
- http://www.qtcentre.org/threads/27419-QPushbutton-style
- http://www.qtcentre.org/threads/20611-setAlignment-in-Layout

- https://forum.qt.io/topic/52622/solved-how-to-position-text-in-a-qgraphicsscene/2
- https://forum.qt.io/topic/71152/window-transparency-in-linux/2
- https://docs.python.org/3/library/os.path.html
- https://programminghistorian.org/lessons/working-with-text-files
- https://pythonspot.com/pyqt5/
- http://zetcode.com/gui/pyqt5/eventssignals/
- http://zetcode.com/gui/pyqt5/customwidgets/
- https://www.tutorialspoint.com/pyqt/pyqt_qmessagebox.htm
- https://www.programcreek.com/python/example/56213/PyQt4.QtGui.QDialog
- https://martinfitzpatrick.name/article/multithreading-pyqt-applications-with-qthreadpool/
- https://www.colorspire.com/rgb-color-wheel/
- https://www.rapidtables.com/web/color/gray-color.html
- https://www.rapidtables.com/web/color/green-color.html
- https://www.rapidtables.com/web/color/brown-color.html

Näiden lisäksi Y2-kurssin oma materiaali, joka löytyy aalto a+:sta.

16. **Liitteet**

Projektin päiväkirja (löytyy doc-kansiosta)

Ohjelmakoodi (löytyy game_files-kansiosta)