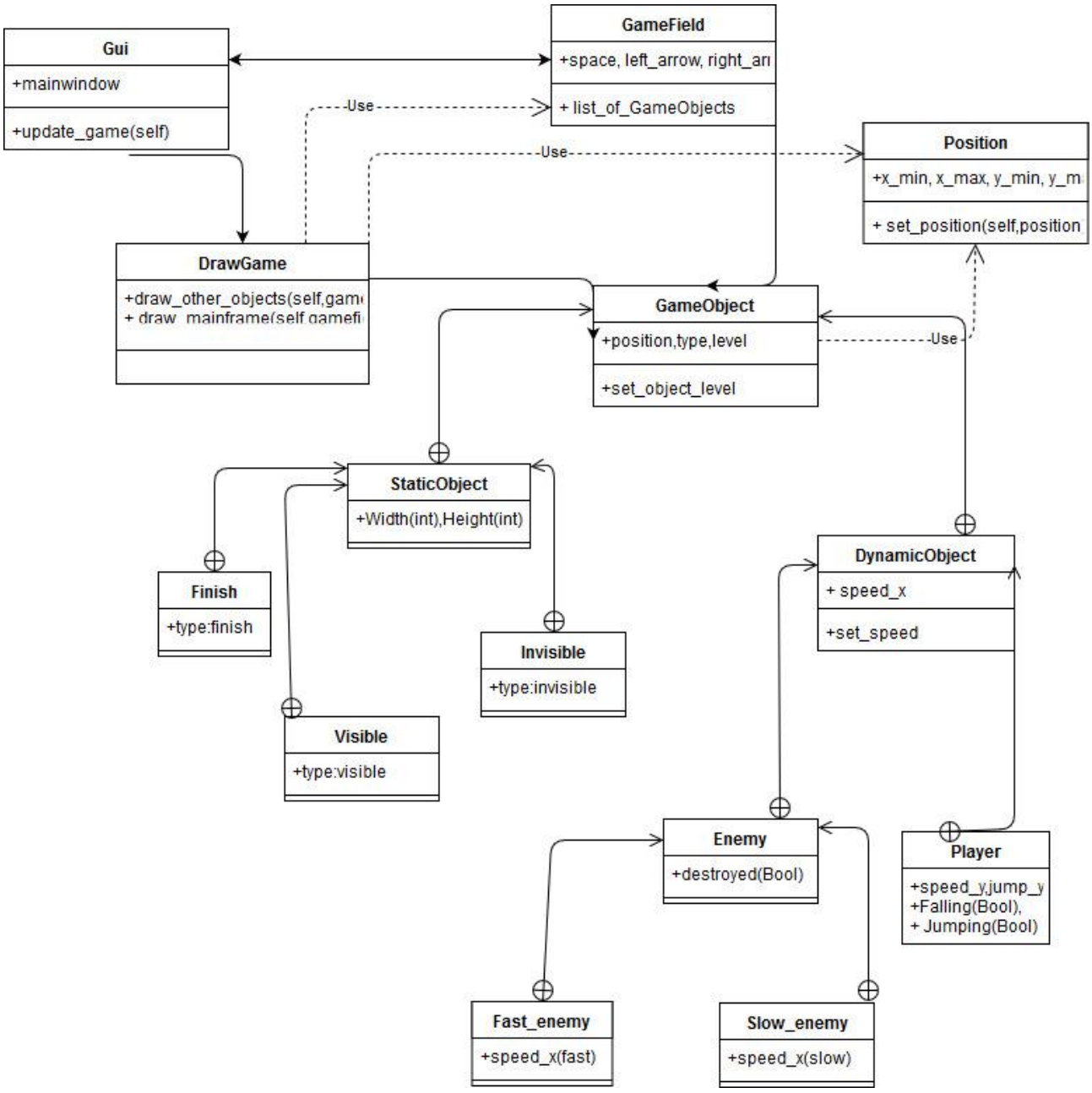
Tekninen suunnitelma

Lauri Westerholm, 530868

1. **Ohjelman rakennesuunnitelma**



Ohjelman voi jakaa selkeästi kolmeen osaan: Ensimmäinen on Gui eli käyttöliittymä, jonka kautta peli ja pelaaja kommunikoivat. Pelaaja siis näkee pelihahmon paikan, esteiden ja vihollisten sijainnin, joiden perusteella pelaaja liikuttaa pelaajaa pelialueella sivusuunnassa ja korkeussuunnassa. Toinen osa on laskentaosuus (tässä GameField-luokka), joka sisältää

muuttujia sekä metodeja pelihahmon paikan päivittämiseen pelaajan inputin perusteella. Laskentaan vaikuttavat vahvasti muiden objektien sijainnit ja niiden tyypit. Näin ollen GameField-oliolla on listamuodossa erilaiset pelikentässä esiintyvät oliot. Gamefield-luokka sisältää myös työkalut pelikentän muodostamiseen (lähtötilanne tiedostosta).

Kolmas osuus on grafiikan piirtäminen eli DrawGame, jota kutsutaan Guista käsin. Tämän luokan on tarkoitus päivittämään pelaajalle näkyvä pelinäyttö vastaamaan pelihahmon sekä muiden objektien laskettuja paikkoja sekä toisaalta myös piirtää pelin lähtötilanne ruudulle.

Luokista Gui kommunikoi suoraan Gamefieldin kanssa (käyttäjän antama keyinput) ja toisaalta huolehtii, että DrawGamen avulla päivitetään ruutua tarpeeksi usein. DrawGame puolestaan piirtää eri peliobjektit ruudulle niiden sijainnin sekä tyypin perusteella käyttäen hyödyksi GameFieldin tiedostosta parsimaa pelikenttää. GameField sisältää listauksen (eli osoittimet) kaikista objekteista ja sieltä käsin päivitetään objekteja tarvittaessa.

GameObject on pelikentällä olevien objektien yläluokka. Sen perivät alaluokat StaticObject ja DynamicObject, joista vain DynamicObject-luokan oliot kykenevät liikkumaan. DynamicObjectin alla ovat edelleen Enemy ja Player -luokat, joista pelaajaa ohjaa yhtä Player-luokan oliota. Viholliset jakautuvat vielä edelleen nopeisiin ja hitaisiin, ja niiden liikettä on rajoitettu esim. hyppyjen puuttumisen osalta. Vihollisia ohjataan GameField-luokasta käsin pelilaudalla omalla metodillaan. StaticObject-oliot puolestaan pysyvät niiden alkuperäisessä paikassa ja niillä tietty vakiokoko. Niiden eroavat tyypit määritellään alaluokista käsin sekä eri alaluokat piirretään DrawGamessa erilailla. Lisäksi niiden toiminnallisuudessa on eroja esimerkiksi pelin lopputuloksen kannalta (GameField luokan laskennassa huomiodaan staattisen objektin tyyppi). Position luokka taas sisältää GameField ja DrawGame luokkaa varten tarkat koordinaatit (pikselit) sen omistavan GameObject olion sijainnista. Sekä DrawGame että GameField ovat vahvasti riippuvaisia position-luokan toteutuksesta.

Tällaisen pelin voisi tietysti toteuttaa hyvin monilla tavoilla. Pohdinkin paljonkin sitä, pitäisikö laskentametodeista olla osa GameObject alaluokkien sisällä. Päädyin siihen, että laskennassa tarvitaan välttämättä tietoa käytännössä kaikista muista objekteista. Tuntuu siis luonnolliselta sisällyttää tällaiseen laskentaa erikoistuneet metodit suoraan luokkaan, joka sisältää tämän informaation eli GameFieldiin.

Toisaalta pohdin myös sitä, kuinka monta GameObject alaluokkaa tarvitsen. Päädyin kuitenkin käyttämään ratkaisua, jossa perintää olisi mahdollisimman paljon. Tämä taas johtaa pienempään määrään toistettavaa koodia, mitä muuten ehkä tulisi enemmän.

Mielestäni ratkaisu jakaa laskenta ja piirto erilleen tuntuu järkevältä. Muuten yhdistelmä luokasta tulisi hyvin sekava ja raskas: se sisältäisi paljon erilaisia metodeja, joiden käyttökohteet poikkeisivat laasti. Tälläkin hetkellä GameField vaikuttaa raskaalta luokalta. Yritin myös minimoida luokkien määrän: ei ylimääräisiä piirto- tai laskentaluokkia. Position luokka taas vaikuttaa kätevältä tavalta ilmoittaa objektien tarkat sijainnit. Se on myös paremmin tarvittaessa laajennettavissa kuin koordinaattien implementoiminen suoraan GameObject -olioon. Esimerkiksi, jos koordinaattien suunniteltua laskentaa on tarkoitus muuttaa, se onnistunee omasta luokasta käsin helpommin. Tosin silti pitää huomioida GameFieldin ja DrawGamen riippuvuus positiosta.

Gui taas käytännössä vaatii melkein pakollakin oman luokkansa, koska se perii pyqt5 käyttöliittymään ja piirtämään erikoistuneita metodeja ja objekteja. Se on myös tämän takia luonteeltaan hyvin erilainen kuin muut luokat. Gui hoitaakin eventhandling-osuuden pelissä.

Kokonaisuudessaan tärkeimmät metodit (lähes kaikki ja muutenkin luokat tarkemmin) on esitetty luokka_suunnittelua -dokumentissa.

Poimintoja tärkeimmistä metodeista:

1. class Gui

__init__(self):

- alustaa pääikkunan koon
- määrittelee yhteyden DrawGame ja GameField olioon

start_game(self):

- valmistelee varsinaisen pelin piirtämisen aloittamisen
- aktivoi pelin piirtämisen

update_game(self):

- varsinainen pelin piirtäminen DrawGame luokan avulla

keyPressEvent(self,event):

- keyPressEventin määrittely (luokasta QtCore QEvent)
- signaalin emittaus näppäintä painaessa

2. class GameField

parse_gamefield(self,file):

- asettaa gamefieldin leveyden
- asettaa pelaajan
- asettaa listaan staattiset objektit ja viholliset filen perusteella
- luettavan filen esimerkkimuoto löytyy mm. yleissuunnitelmasta

update_objects_positions(self):

- päivittää kaikkien vihollisten ja pelihahmon paikkaa
- kutsutaan DrawGamesta käsin

move(self,object,direction):

- liikuttaa objektia haluttuun suuntaan tietyllä nopeudella

jump(self,player,direction):

- hyppäyttää pelaajaa, jos mahdollista

fall(self,player,direction):

- yritetään tiputtaa pelaajaa tietyn vakion verran alaspäin

check_position(self,object,position):

- Tarkistaa, onko sijainti mahdollinen ennen palaajan tai vihollisen liikuttamista sijaintiin (jos ei mahdollinen, aiheuttaa tiettyjä jatkotoimenpiteitä)

3. class DrawGame

draw_mainframe(self):

- piirtää gamefieldin stattiset objektit ja ”pelikentän alustan”

draw_other_objects(self):

- piirtää gamefieldin pelihahmon ja viholliset
- päivittää ensin kaikkien liikkuvien olioiden paikat kutsumalla gamefield update_objects_positions

4. class GameObject

set_object_level(self,level):

- muutetaan level arvo vastaamaan pelikentän sisältävässä filessä olevaa / muutetaan leveliä (hyppy tai pudotus)

set_position_object(self,position):

- asetetaan gameobject-oliolle positio-olio (kts.positio)

5. class Position

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

- kutsutaan kun parsitaan filestä pelikenttää ja luodaan objektit
- luodaan filen sijainnin perusteella peliobjektille positio

2. **Käyttötapauskuvaus**

1.Pelaaja alkaa suorittaa ohjelmaa, jolloin pelin pääikkuna käynnistyy.

Peli tekee alkuvalmistelut eli lataa kentän tiedostosta, asettaa pelilaudan aloitustilanteeseen ja käynnistää eventhandling-osuuden

2. Pelaaja painaa Start Game -painikketta, jolloin varsinainen peli alkaa.

Pelaajan toimet aikaansaavat eventin, joka käynnistää varsinaisen pelin. Tästä eteenpäin Gui kutsuu DrawGame -oliota päivittämään grafiikat säännöllisesti

3.Pelaaja näkee ruudulla alkutilanteen ja alkaa liikuttamaan pelihahmoa oikealle edetäkseen pelikentässä.

Gui välittää pelaajan näppäinpainallukset eteenpäin, joita Gamefield-luokasta käsin käytetään pelitilanteen päivittämiseen. DrawGame piirtää muutokset.

4.Vastaan tulee este, jonka yli pelaaja hyppää

Sama kuin edellä: GameFieldin avulla päivitetään pelaajan sijaintia, joka sitten piirretään näytölle.

5. Pelaaja jatkaa etenemistä välillä hyppien esteiden yli

Sama kuin edellä

6. Pelaaja kohtaa vihollisen ja tuhoaa sen hyppäämällä sen päälle

Kyseisen vihollisen tietoja muutetaan siten, että sitä ei enää piirretä eikä siihen voi vaikuttaa. Tiedonvälitys luokkien välillä samoin kuin aiemmissa kohdissa.

7. Pelaaja näkee maalin ja ohjaa pelihahmon sen päälle

Nyt Gui saa tiedon, että varsinainen peli on päättynyt. Guista käsin piirretään lopetusruutu ja valmistellaan seuraavan pelin aloittamisen resetoimalla aiemman pelin tiedot

8. Pelaaja huomaa loppuruudun ja etenee alkuruutuun painamalla Continue-näppäintä

Gui palauttaa pelin käynnistyksen jälkeiseen alkutilaan ja piirtää aloitusruudun

9. Sykli jatkuu, kunnes pelaaja lopettaa painamalla Esc-näppäintä

Esc painaminen aikaansaa eventin, joka terminoi koko ohjelman pysäyttäen eventhandling-osuuden

3. **Algoritmit**

Tässä tapauksessa monimutkaisimmat algoritmit sisältyvät pelaajan liikkumiseen.

Ensinäkin ennen liikkumista tarkastetaan, että tuleva sijainti on vapaana (collosion detection). Tämä algoritmin toiminta pohjautuu siihen, että on olemassa ajantasaiset (eli siis laskettavissa) 2D sijainnit kaikista pelikentällä olevista objekteista. Voidaan siis käydä kaikki objektit läpi laskien niiden tarkka sijainti ja tarkastaa että haluttu sijainti on vapaa.

Objektien tarkan 2D sijainnin määrittäminen on mahdollista, sillä pelikenttätiedostosta luetaan objektien alkupaikat. Alkupaikoista saadaan objektin (suorakulmio) reunojen sijainnit 2D – tasossa. Näytä sijainteja käytetään törmäyksen tunnistamiseen ja niitä päivitetään liikuttamiseen erikoistuneiden metodien kautta. (Jos tiedetään suorakulmion kaikki neljä reunaa, tiedetään koko alue, eli pikselit, jotka jäävät suorakulmion sisälle)

Muuten liikkuminen on suhteellisen suoraviivaista. Huomioitavaa on kuitenkin se, että pelihahmo ei saa pystyä liikkumaan ilmassa eli hyppyjen yhteydessä on huolehdittava pelaajan levelistä suhteessa kiinteisiin objekteihin. Voidaan taas kerran objektien tarkat sijainnit tuntemalla määrittää, millä tasolla pelaajan alapuolella oleva objekti on. Jos hypyn seurauksena pelaajan leveli ei ole yhtä suurempi kuin alapuolisen objektin, pudotetaan pelaajaa, kunnes pelaajan suorakulmio koskettaa alapuolella olevaa suorakulmiota.

Vihollisilla ei ole tässä pelissä varsinaista tekoälyä. Ne liikkuvat ensin vasemmalle ja kun törmäävät kiinteään objektiin (törmäyksen tunnistus kuvattu yllä), kääntyvät takaisin oikealle jne.

Tapa, jolla törmäyksen tunnistus on tarkoitus toteuttaa (absoluuttisten koordinaattien vertailu), on luonteeltaan varma, mikä on ohjannut sen valintaan. Haittapuolena on kuitenkin suhteellisen suuri laskentamäärään tarve. Ongelmia törmäyksen tunnistuksen toteuttamiseen aiheuttaa tässä tapauksessa se, että pelaajan sijaintia tietyllä ajanhetkellä on vaikea ennakoida. Tämä taas johtaa siihen, että törmäyksen tunnistuksen on toimittava erinäisissä tilanteissa sekä hyvin luotettavasti.

4. **Tietorakenteet**

Ohjelmassa tarvitaan käytännössä välttämättömästi dynaamista rakennetta siinä vaiheessa, kun pelikenttätiedostosta tehdään pelikenttää. Muuten objektien lukumäärän pitäisi olla kiinteä, mikä ei ole vaihtoehto pelin laajennettavuuden/ erilaisten kenttien suunnittelun kannalta. Tässä tapauksessa listat eri peliobjekteista vaikuttaa järkevimmältä ratkaisulta. Myös sanakirjan käyttöä olen pohtinut, mutta se ei kuitenkaan vaikuta tuovan suurta lisäarvoa (ei esim. merkittävästi helpota collosion detectionia).

Lisäksi ohjemassa on tarkoitus käyttää muutamassa luokassa luokkamuuttujia, jotta luokan sisäisiä asioita saataisiin paremmin kontrolloitua. Luokka_suunnittelua- dokumentista löytyy tarkemmin se, missä luokkamuuttujia oli tarkoitus hyödyntää.

5. **Aikataulu**

1. GameObjektien ja Positionin alustava muoto. Kenttätiedoston lukeminen toimivaksi. Se muodostaa alkutilanteen, jonka pohjalta esim. piirtometodeja voi testata. Aikataulu: 1. ohjelmointiviikko, n. 10h

2. Gui siihen kuntoon, että peli-ikkunan voi piirtää. Keypress tomivaksi. Testaus ei-kiinteällä piirrolla (ei DrawGame). Aikataulu: 2.ohjelmointiviikko, n. 6h.

3. DrawGamen tekeminen mahdollisimman lähelle lopullista muotoa sekä objektien tekeminen piirron mahdollistaviksi. Kun toimii, pitäisi pystyä piirtämään alkutilanne pelistä. Aikataulu: (2. ja) 3. ohjelmointiviikko, n. 10h

4. Varsinaisen pelin ohjelmointi eli GameField- luokan tärkeimmät metodit toimiviksi (fileread pitäisi jo toimia). Aikataulu 4. ohjelmointiviikko, n. 10h. Tässä vaiheessa tärkeää saada pelaajan liikkuminen toimimaan, vihollisen liikkuminen ei niin olennaista

5. GameFieldin liittäminen Guihin ja DrawGameen ja varhaisen mainin tekeminen. (Tässä vaiheessa luultavasti ilmenee uusia ongelmia) Aikataulu: 5. ja 6. ohjelmointiviikko, n. 17h (aikaa varattu myös kiinniottamiseen, jos aikataulusta jääty)

6. Guin muokkaaminen lopulliseen muotoon. Muidenkin luokkien ja mainin lopullinen muoto. aikataulu 7. ohjelmointiviikko, n. 10h

7. Ohjelman lopputestaus ja bugien korjaus 8. ohjelmointiviikko, n. 8h

6. **Yksikkötestaussuunnitelma**

Yksi ensimmäisenä toteutettavista asioista on lähtötiedoston lukemisen hoitavan parse_gamefield toteutus. Tämä metodi on suhteellisen helppo yksikkötestata: Sitä tulee kokeilla parilla erilaisella filellä ja katsoa, että oikeat objektit ovat pelikentällä oikeissa paikoissa. Virhetilanteita ei tarvitse juurikaan testata (yksittäinen testi), koska sellaisia ei pitäisi tulla koskaan, sillä syötefilet ovat itsetehtyjä.

Keypress eventtien välittäminen on hyvin tärkeää, joten niitä käsitteleviä metodeja on ehdottomasti yksikkötestattava. Tarkastetaan, että näppäinpainallukset välittyvät oikealla tavalla ja muiden näppäiminen painallukset eivät vaikuta siihen.

Kun pelikentän alkutilanne on saatu muodostettua oikein, pystyy alkutilannetta hyödyntämään DrawGame -luokan metodien (esim. draw_mainframe) testaukseen. Alkutilannetta muokkaamalla saadaan aikaan muita hyviä testicaseja.

GameFiled -luokan muita metodeja onkin sitten huomattavasti vaikeampaa yksikkötestata, vaan ne käytännössä vaativat, että DrawGame on toteutettu. Jonkin verran kuitenkin niitäkin

pystyy testaamaan, esimerkiksi printtaamalla pelaajan paikkaa ennen hyppymetodin kutsumista sekä sen suorituksen jälkeen.

Guilla luodulla mainscreenillä on taas taipumus freezata, jos jokin on pahasti vialla. Sitä pystyy kuitenkin alustavasti testaamaan käyttämällä DrawGamen tilalla jotain muuta syklisesti vaihtuvaa piirtoa.

7. **Kirjallisuusviitteet ja linkit**

Jo käytetyt sivustot:

- 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
- http://pyqt.sourceforge.net/Docs/PyQt5/ (lukuisia dokumentaatiosivuja)
- https://www.qt.io/ (lukuisia dokumentaatiosivuja)
- https://pythonspot.com/pyqt5/
- http://zetcode.com/gui/pyqt5/eventssignals/
- http://zetcode.com/gui/pyqt5/customwidgets/
- https://martinfitzpatrick.name/article/multithreading-pyqt-applications-with-qthreadpool/

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

Jatkossa huomattavasti lisää pyqt5 liittyviä perusluokkakuvauksia, jotka löytyvät qt.io -sivulta. Ongelmatilanteissa tulee luettua stackoverflowta.

8. **Liitteet**

Luokka_suunnittelua -dokumentissa tarkemmat selostukset metodeista sekä luokista