

Olio-ohjelmoinnin peruskurssi

Harjoitustyö

Mikko Malkavaara
mmmalk@utu.fi
517788

Vili Ahava
vsahav@utu.fi
516680

Maks Turtiainen
mjturt@utu.fi
517579

Työn ohjaaja:
Hanna Ahtosalo
hakrah@utu.fi

Sisältö

1	Tehtävän kuvaus ja analysointi	2
1.1	Tehtävänanto	2
2	Ratkaisuperiaate	2
2.1	Pelaajan ja vihollisten luokat	2
2.2	Vihollis-, hyökkäys-, esine- ja loitsulistaukset	3
2.3	Taistelumoottori	3
2.4	Maailman luominen	3
2.5	Graafinen käyttöliittymä	4
2.6	Pelitilanteen tallennus ja lataus	4
3	Ohjelman ja sen osien kuvaaminen	4
4	Testausjärjestely	4
5	Liitteet	7
5.a	Alkuperäinen tehtävänanto	7
5.b	Ohjelmalistaus	7
5.c	Käyttöohje	7

1 Tehtävän kuvaus ja analysointi

1.1 Tehtävänanto

Tehtävänantona oli toteuttaa yksinkertainen peli, jossa on komentorivipohjainen tai graafinen käyttöliittymä. Lisäksi pelin tulisi mahdollistaa tilan tallentaminen, ja muita mahdollisia ominaisuuksia mainittiin esimerkiksi ns. ”high score” -taulu. Lisäksi toinen mahdollinen aihe oli itse keksitty aihe. Vaikka tämä ratkaisu ei välttämättä sovi puhtaasti kumpaankaan aiheeseen, päädyimme rajaamaan projektin toteutettavalla tavalla rajoitetun ajan takia. Työ myös pyrittiin toteuttamaan kovakoodaamalla mahdollisimman vähän mitään, vaan kaikki pyritään luomaan lennossa tai lukemaan ulkoisista resursseista.

2 Ratkaisuperiaate

Tehtävänannon perusteella päätimme tehdä yksinkertaisen roolipelin. Pelin tarkoituksena on päästä satunnaisesti valitusta aloitusruudusta niinikään satunnaisesti valittuun maaliruutuun. Koska pelillisesti pelkkä sokkelossa navigoiminen ei olisi kovin mielenkiintoinen, tai laajuudeltaan järin sopiva harjoitustyön aiheeksi mielestämme, päädyimme toteuttamaan peliin myös satunnaisia kohtaamisia vihollisten kanssa.

Itse peli koostuu kahdesta näkymästä, karttaruudusta jolla liikutaan sekä taisteluruudusta, jossa kohtaamiset vihollisten kanssa tapahtuvat. Nämä näkymät toteutettiin graafisesti, käyttäen hyväksi javan awt- ja swing-kirjastoja. Näissä näkymissä pelaajan syöte luetaan näppäimistöltä, ja palaute tulostetaan ruudulle (joko esimerkiksi graafisena liikkeen ilmentymänä tai tekstimuotoisena syötteenä jonkin asian tapahtumisesta). Satunnaiskohtaamiset vihollisten kanssa tarkistetaan karttaruudulla satunnaislukuja generoimalla, ja kohtaamisen tullessa vastaan käynnistetään pelin kohtausnäkyvä.

2.1 Pelaajan ja vihollisten luokat

Taistelujärjestelmän näkökulmasta pelaaja ja vihollinen ovat vain yliluokan Creature ilmentymiä, joilla on jaettuja attribuutteja (esimerkiksi nimi, osumapistee, eri tilastot jne). Creature-luokka sisältää myös tiedot ilmentymiensä tietämistä hyökkäyksistä, inventaariosta ja loitsuista. Havainnointi- ja asetusmetodiensa lisäksi Creature-luokalla on myös metodit myös taistelussa hyökkäyksien aikaansaamoon vahingon laskemiseen ja tekemiseen, esineiden käyttämiselle sekä myös kehityspisteiden tarkistamiseksi voidaan seuraava kehitystaso saavuttaa (sekä myös itse kehitystason ja ilmentymien tilastotietojen nosto).

Pelihakmon luokalla on myös yliluokkansa sisältämien tietojen lisäksi tieto missä kohtaa pelihahmo sijaitsee karttaruudulla, sekä tälle asetus- ja havainnointimetodit. Näiden lisäksi pelihakmole on myös metodit, joilla liikutaan pelimaailmassa eri suuntiin (kuitenkin graafisella käyttöliittymällä on tähän omat metodinsa). Näitä metodeja on käytetty koodin testaamiseen ja prototyypaukseen ennen kuin varsinaisen graafisen käyttöliittymän kehitystä oli aloitettu.

Vihollisen luokalla on vain yksi oma metodi, jonka avulla käydään lävitse lista vihollisen tietämistä loitsuista sekä hyökkäyksistä, ja lasketaan isoin mahdollinen vahinko minkä vihollinen pystyy hyökkäyksen kohteeseen tekemään. Loitsuja läpikäyvä silmukka jättää vertailusta pois sellaiset loitsut, joiden käyttämiseen vihollisen taikapisteet eivät riitä.

Pelissä jokainen kukistettu vihollinen kerryttää omien elämäpisteidensä verran kokemuspisteitä. Pelaajan tarvitsemat kokemuspisteet lasketaan seuraavalla kaavalla:

$$\frac{5 \cdot \text{player.level}}{4}$$

Tämän lisäksi Creaturen kehitystason noustessa attribuutit voima, puolustus, taika, sekä taika- ja kestävyyspisteet nousevat noudattaen seuraavaa kaavaa, tämän lisäksi Creaturen voima- ja kestävyyspisteet asetetaan uusiin maksimeihinsa

$$\lfloor \frac{\text{creature.level} + 10}{10} \rfloor$$

Creature-luokan ilmentymän tekemä vahinko lasketaan puolestaan käyttäen hyväksi seuraavaa kaavaa:

$$\frac{\frac{\text{attacker.strength}}{\text{defender.defense}} \cdot \text{attack.strength} \cdot (\frac{\text{attacker.level} \cdot 2}{5} + 2)}{50} + 2$$

Loitsut noudattavat samaa kaavaa, paitsi että voima ja puolustus korvataan hyökkääjän ja puolustuksen taikavoimalla.

2.2 Vihollis-, hyökkäys-, esine- ja loitsulistaukset

Koska pelisuunnittelu on muutenkin tarpeeksi epäkiitollista toimintaa, päätimme helpottaa omaa elämäämme toteuttamalla metodin, joka lukee peliin luotavia olioita tiedostosta. Halusimme, että tämä data on helposti muokattavissa nopean kehityksen mahdollistamiseksi ilman erikoistyoäkaluja, joten nämä tiedot tallennetaan utf-8-muotoiltuna pilkulla erotettuna tietueena tekstitiedostossa. Nämä tiedot luetaan riveittäin tiedostosta, ja niitä käytetään uuden olion alustamisessa, ja vihollistiedostossa on tämän lisäksi myös tiedot vihollisen tietämistä loitsuista ja hyökkäyksistä.

Lisäksi kirjoitimme luokat vihollisten ja esineiden generointiin. Vihollisten generoinnin tapauksessa luetaan tiedostosta vihollisten tiedot uusiksi olioiksi, ja näihin olioihin lisätään niiden tietämät loitsut sekä hyökkäykset. Näitä vihollisia palauttava koodi myös määrittää millä tasolla vihollisen kehitystaso on, ja nostaa sitä ennen vihollisolion palauttamista.

2.3 Taistelumoottori

Itse taistelumoottori järjestää taistelun pelaajan ja vihollisolion välillä. Varsinainen silmukka käsittelee käyttäjän syötteen näppäimistöltä, ja tulostaa ruudulle pelaajan ja vihollisen kannalta oleelliset tiedot. Pelaaja ja vihollinen toimivat vuorotellen tuossa järjestyksessä, ja pelisilmukka tulostaa aina uudet tiedot joka vuoron jälkeen. Pelisilmukkaa ajetaan kunnes jommaltakummalta, pelaajalta tai viholliselta kestävyyspisteet putoavat nollaan tai sen alle, ja jos pelaaja selvisi elossa annetaan pelaajalle vihollisen kestävyyspisteiden verran kokemuspisteistä. Tämän lisäksi on pelaajalla mahdollisuus saada satunnaisesti generoituja esineitä, joiden määrä perustuu vihollisen kehitystasoon.

2.4 Maailman luominen

Peliä varten luodaan jokaiselle uudelle pelikerralle uusi maailma. Tämä pelimaailma koostuu ns *tileistä*, joka on pienempi ruutu pelimaailmaa. Yksi tile sisältää aina tiedon minkätyyppinen se on, ts. voiko pelaaja kävellä sen läpi vai ei. Graafista käyttöliittymää varten määrittelimme yhden tilen kooksi 64 x 64 pikseliä.

Maaailma luodaan tileistä koostuvana taulukkona, jolle voidaan antaa haluttu koko pysty- ja vaakasuunnassa. Algoritmi luo ensin Tile-taulukon, johon se lähtee muodostamaan eri reittejä satunnaislukugeneraation avulla muodostaen yhtenäisiä ”ratoja”. Itse metodi siis asettaa tilen tilan sellaiseksi, että pelaaja pystyy kulkemaan siitä lävitse. Tämän jälkeen toisella metodilla puhkotaan satunnaisiin väleihin sokkelossa reikiä asettamalla ruutuja sellaisiksi että pelaaja pääsee kävelemään niistä

2.5 Graafinen käyttöliittymä

Graafinen käyttöliittymä koostuu lähinnä kahdesta ikkunasta, maailmanäkymästä sekä taistelunäkymästä. Kumpikin suunniteltiin niin, että käyttöliittymä hoitaa vain pelin piirtämisen ja käyttäjän syötteen lukemisen.

Pelin karttanäkymä koostuu kolmesta pääelementistä, itse tasosta jolle kaikki piirretään, kartalle piirrettävistä maastoista sekä pelihahmosta. Koska pelimaailma ja ikkuna kumpikin voivat olla mielivaltaisen kokoisia, toteutettiin pelinäkymän piirtäminen niin, että pelihahmo pysyy keskellä ruutua ja maailma liikkuu pelihahmon alla. Pelille luodaan siis uusi ikkuna, johon luodaan uusi maailma ja pelaaja. Tämän jälkeen näille tehdään graafiset ilmentymät jotka piirretään ruudulle ja käyttäjän syöte luetaan tietyn väliajoin. Lisäksi maailman pelinäkymä käyttää erillistä handler-luokkaa hoitamaan objektien päivityksen.

Taistelunäkymä pelissä toimii piirtämällä uuden ikkunan, johon luodaan pelaajan mahdolliset toiminnot Swing-kirjastoa hyväksikäyttäen valmiita painike-luokkia. Nämä tiedot luetaan pelaaja-luokan tiedoista ja generoidaan dynaamisesti, ja ne sidotaan taistelumoottorin metodeihin. Taistelunäkymä saa fokuksen maailmanäkymältä ja se luodaan maailmanäkymän ali-ikkunana. Koska esimerkiksi pelaajan esinelistausta saattaa vaihdella, luodaan käyttöliittymäelementit siihen lennosta

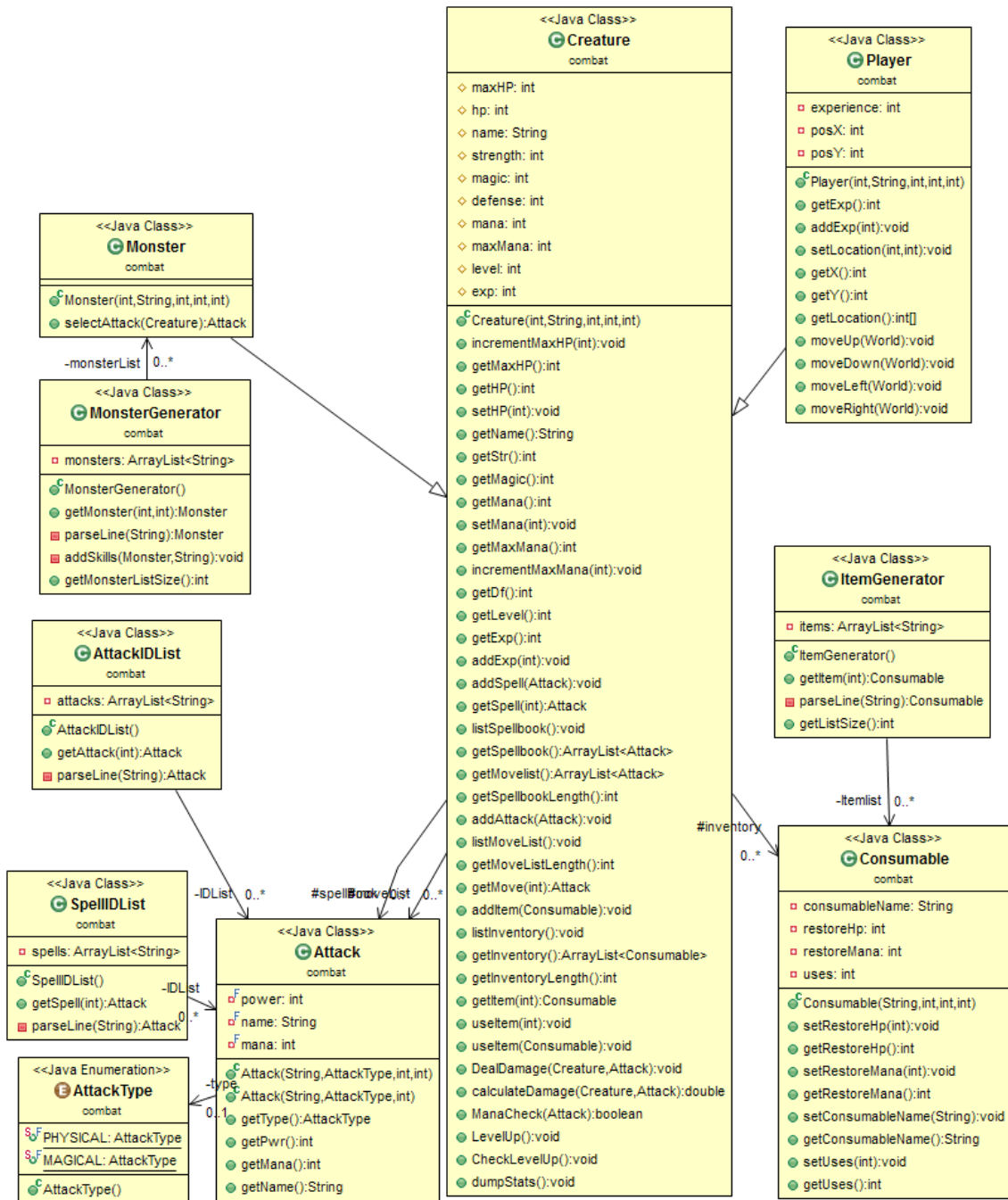
2.6 Pelitilanteen tallennus ja lataus

Pelitalanne on mahdollista tallentaa ja ladata käyttämällä hyväksi pelimoottorin handler-luokkaa. Tämä luokka tallennetaan /sav-kansion alle binääriobjektina tiedostoon, ja se voidaan lukea sieltä vapaasti käyttäen hyväksi siihen kirjoitettua metodia.

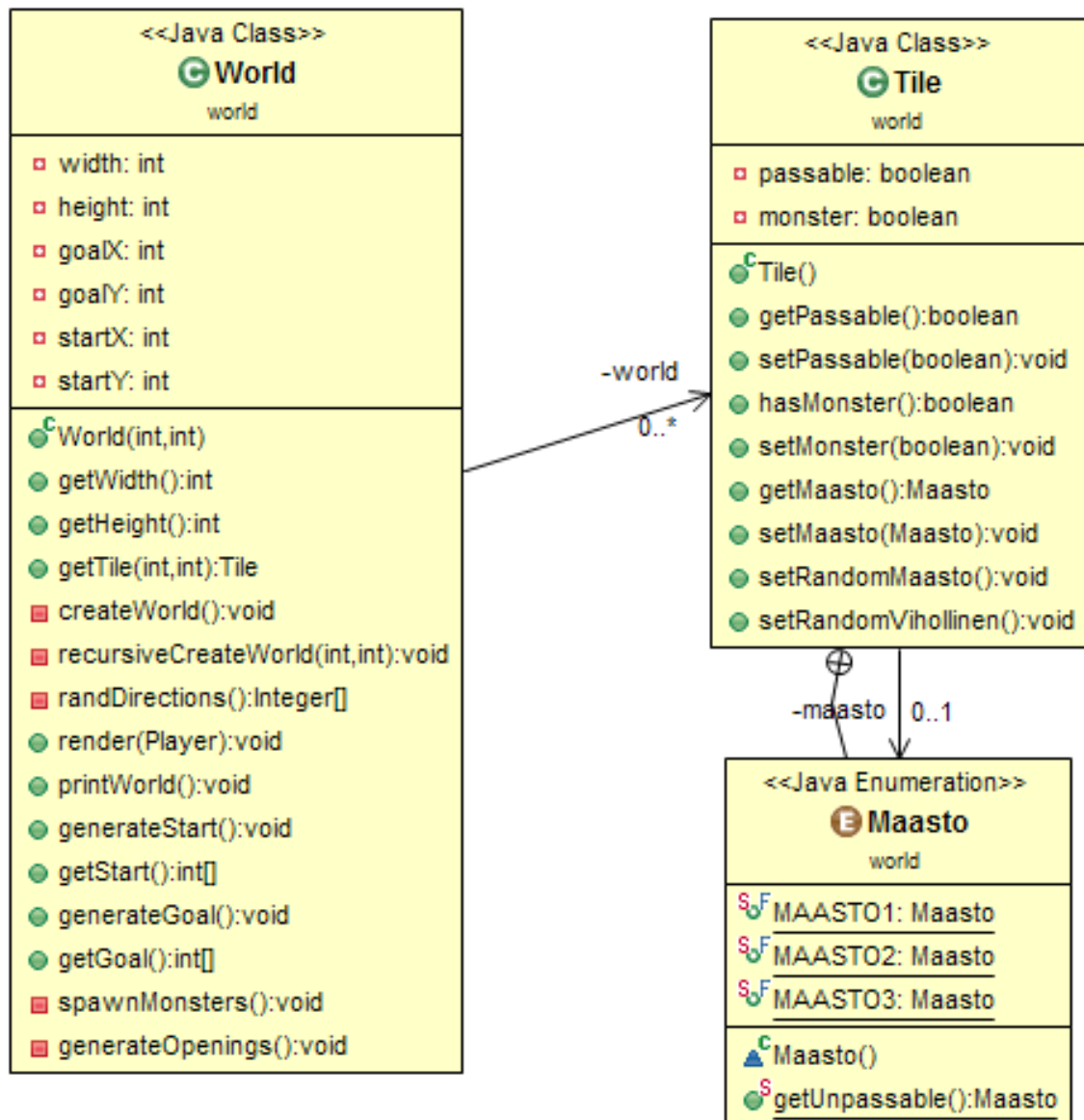
3 Ohjelman ja sen osien kuvaaminen

4 Testausjärjestely

Harjoitustyötä tehdessä käytimme hyväksemme kahdenlaista testausstrategiaa; pelitestausta sekä myöskin metodien toimintaa testaavia prototyyppejä tulevista funktioista. Nämä testikoodit eivät kuitenkaan olleet varsinaisia yksikkötestejä, vaan lähinnä komentorivipohjaisia toteutuksia asioista jotka lisättiin graafiseen käyttöliittymään myöhemmin. Varsinaisessa pelitestauksessa samoja toimintoja toistettiin useampaan otteeseen ja usein näin löydettiin ennakkoimattomia virheitä ja poikkeustilanteita, jotka saatiin korjattua melko ripeästi. Pelissä ei nykytietämyksen mukaan ole mahdollisia virhetilanteita.



Kuva 1: Luokkakaavio combat-paketista



Kuva 2: Luokkakaavio world-paketista

5 Liitteet

5.a Alkuperäinen tehtävänanto

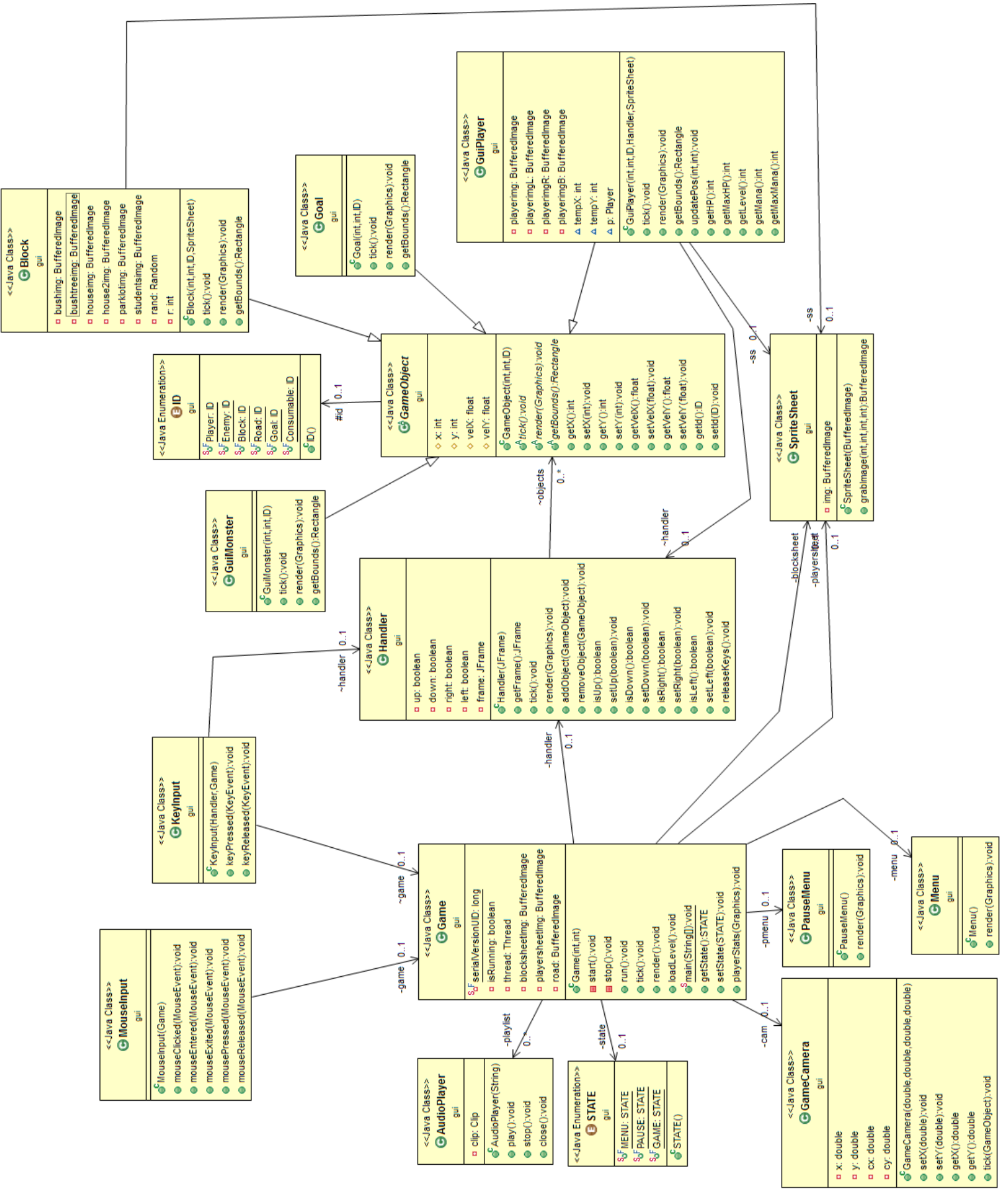
Alkuperäinen tehtävänanto liitteenä tämän hakemistorakenteen juuressa.

5.b Ohjelmalistaus

Ohjelmalistaus löytyy tämän hakemiston src-alihakemistosta. Ohjelman käyttämät resurssit taas vastaavasti assets, res, ja graphics -alihakemistoista

5.c Käyttöohje

Ohjelmiston käyttöohje löytyy hakemistorakenteen juuresta käyttöohje.pdf-nimisenä tiedostona



Kuva 3: Luokkakaavio gui-paketin keskeistä elementteistä