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

Kuvat

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 sovikaan puhtaasti kumpaankaan aiheeseen, päädyimme rajaamaan projektin toteutetulla 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 taisteluruudulla, 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 kohtaamisnäkymä.

2.1 Pelaajan ja vihollisten luokat

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

Pelihahmon luokalla on myös yliluokkansa sisältämien tietojen lisäksi tieto missä kohtaa pelihahmo sijaitsee karttaruudulla, sekä tälle asetus- ja havainnointimetodit. Näiden lisäksi pelihahmolle on myös metodit, joilla liikutaan pelimaailmassa eri suuntiin(kuitenkin graafisella käyttöliittymällä on tähän omat metodinsa). Näitä metodeja on käytetty koodin testaamiseen ja prototyyppaukseen 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:

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

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

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

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

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

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 erikoistyökaluja, 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.

Maailma 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 tietyin 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 esinelistaus saattaa vaihdella, luodaan käyttöliittymäelementit siihen lennosta

2.6 Pelitilanteen tallennus ja lataus

Pelitilanne 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

3.1 `combat`-paketti

`combat`-paketti sisältää kaikki pelin taistelumekaaniikkojen kannalta olennaiset luokat. Oheisessa kuvauksessa on jätetty testikoodit listaamatta, sillä nämä käsitellään myöhemmin kohdassa 4. Testausjärjestely.

3.1.1 `Creature`-moduuli

`Creature`-moduuli on työssä käytetyistä moduleista laajin. Tämä moduuli sisältää seuraavat attribuutit, joille on myöls olemassa omat asetus- ja havainnointimetodinsta:

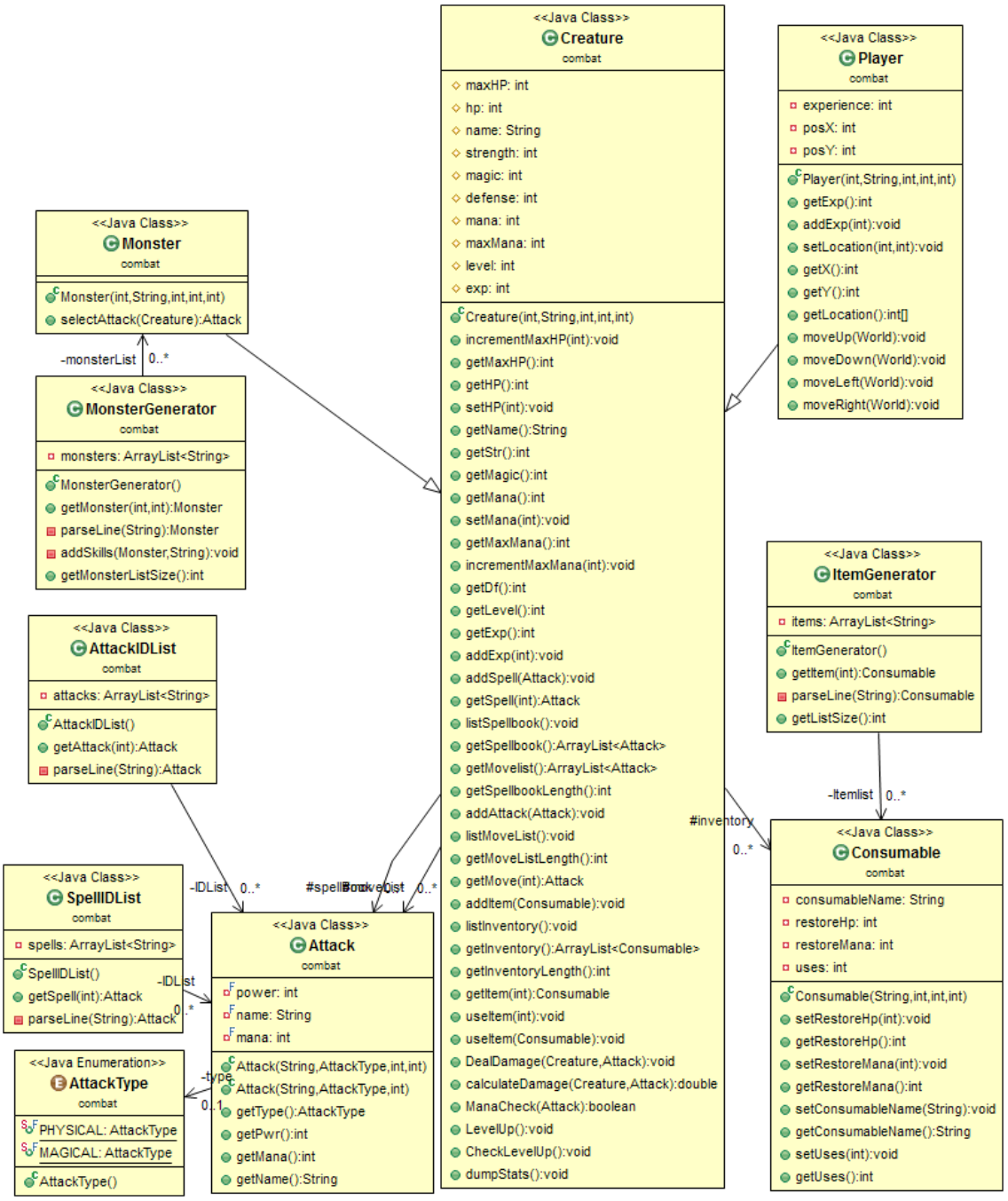
`int maxHP` Olion suurimmat mahdolliset kestävyyspisteet

`int hp` Olion tämänhetkiset kestävyyspisteet

`int strength` Olion voima

`int magic` Olion taikavoima

`int defense` Olion puolustus



Kuva 1: Luokkakaavio `combat`-paketista

`int mana` Olion taikapisteet

`int maxMana` Olion suurimmat mahdolliset taikapisteet

`int level` Olion kehitystaso

`int exp` Olion kokemuspisteet

`String name` Olion nimi

`Creature` **alustetaan** `public Creature(int hp, String name, int strength, int defense, int magic)`**-muotoisella metodilla, jossa sille annetaan sen nimi merkkijonomuotoisena, sekä kokonaislukuina sen sisältämät statistiikat.**

`Creature`**-moduli myös pitää sisällään** `ArrayList<Spell>`, `ArrayList<Attack>` **sekä** `ArrayList<Consumable>` **-muotoiset tietorakenteet joissa on tieto olion tietämistä loitsuista, hyökkäyksistä sekä käytettävissä olevista esineistä. Näille listoille on myös omat metodinsa, joiden avulla listoja voidaan muokata lisäämällä ja poistamalla elementtejä listoista. Koska nämä listat ovat asetettu yksityisiksi attribuuteiksi, on näillä myös metodi jolla voidaan havainnoida listan pituus. Lisäksi listamuotoisilla tietorakenteilla on omat metodinsa niiden listaamiseksi lennosta.**

`public void useItem(int index)`**-metodi ottaa metodikutsussaan listan indeksin argumentiksi. Itse metodin runko hakee** `Creaturen` **inventaarioista** `Consumablen` **parametrina annetun indeksinumeron mukaan ja lukee sen sisältämät tiedot paljonko elämä- ja taikapisteitä palautetaan.**

`public double calculateDamage(Creature defender, Attack a)`**-metodi laskee paljonko vahinkoa** `Creature`**-saa aikaiseksi** `defender`**-olioon, käyttäen hyökkäystä** `a`**. Tämä metodi palauttaa liukulukuna lasketun vahingon määrän.**

`public void DealDamage(Creature defender, Attack a)`**-metodi ottaa parametreikseen** `Creature`**-luokan ilmentymän, sekä** `Attack`**-luokan ilmentymän** `a`**, ja antaa kutsuu nällä parametreillä** `calculateDamage`**-metodia.** `calculateDamagen` **paluuarvo perusteella vähennetään** `defenderin` **elämäpisteistä** `a`**:n tekemä vahinko, ja** `Creaturelta`**, jonka** `DealDamage`**-metodia kutsutaan vähennetään hyökkäykseen kuluneet taikapisteet**

`public boolean ManaCheck(Attack a)`**-metodi ottaa parametrikseen hyökkäyksen** `a`**. Tämä metodi yksinkertaisesti palauttaa joko** `true`**, jos taikapisteet riittävät hyökkäyksen tekemiseen, tai** `false` **jos taikapisteet eivät riitä.**

`public void CheckLevelUp()`**-metodi tarkistaa rekursiivisesti riittävätkö** `Creaturen` **kokemuspisteet seuraavalle kehitystasolle. Jos tämä toteutuu, kutsuu se** `LevelUp`**-metodia**

`public void LevelUp()`**-niminen metodi kutsuu ei palauta mitään, vaan se suorittaa seuraavalle kehitystasolle nousemisen**

`public void dumpStats()`**-metodi tulostaa** `Creaturen` **tiedot järjestelmän ulosteeseen.**

3.1.2 `Player`-luokka

Tämä luokka on `Creaturen` **aliluokka, ja se perii kaikki sen sisältämät attribuutit ja metodit. Tämän lisäksi luokka pitää sisällään tiedot sen sijainnista x- ja y-akseleilla**

Pelaajalla on myös liikkumafunktiot `public void moveUp(World world)`, `public void moveDown(World world)`, `public void moveLeft(World world)`, `public void moveRight (World world)`. Nämä liikkumafunktiot muuttavat pelaajan positiota argumenttina annetussa maailmassa `world`, sen tarkistaessa kyetäänkö seuraavaan ruutuun liikkumaan.

3.1.3 `Monster`-luokka

Aivan samalla tavalla kuin `Player`, on myös tämä luokka `Creaturen` alaluokka, ja se perii kaikki tämän ominaisuudet. Oliota alustaessa sen kehitystaso asetetaan tasolle 1.

`public Attack selectAttack(Creature target`-metodi laskee käyttäen hyväksi `Creature`-luokan `CalculateDamage`-metodia hyökkäyksen, jolla saadaan suurin vahinko aikaiseksi `target`-`Creatireem`. Tämä tapahtuu lukemalla olion hyökkäykset ja loitsut yksi kerrallaan listalta, ja vertaamalla niitä tiedettyä suurinta arvoa vasten. Lisäksi tässä metodissa on pelin mielenkiintoisammaksi tekemisen takia mahdollisuus, että 15% ajasta tämä metodi palauttaa satunnaisen hyökkäyksen.

3.1.4 `MonsterGenerator`-luokka

Tämä luokka on tarkoitettu vihollisten lukemiseen tiedostosta, jottei niitä tarvitse kovakoodata ise pelilogiikkaan. Tällä luokalla on vain pari omaa attribuuttia:

`ArrayList<Monster> monsterList` Lista, joka sisältää tiedot kaikista tiedostosta luetuista hirviöistä

`ArrayList<String> monster` Lista, jossa on tekstimuodossa aina yksi rivi tiedostosta

Tämä luokka lukee alustaessaan tiedoston `monsterlist`, jossa on kaikki pelin tietämät vihollisoliot ja näiden tiemätä hyökkäykset koodattu pilkuilla erotettuina arvoina riveittäin. Itse lukeminen tapahtuu hyödyntämällä `BufferedReader`-luokkaa, joka lukee resurssin läpi riveittäin ja antaa tämän rivin argumenttina `parseLine` (joka lisää hirviön tiedettyjen hirviöiden listaan) tai `addSkills` (joka lisää hirviölle sen hyökkäykset) -metodeille, riippuen siitä onko loitsujen ja hyökkäysten erottimena käytetty merkki `''`-läsnä luetulla rivillä. Tämä lukurutiini nostaa `IOException`-poikkeuksen, jos tiedostoa ei voida lukea. Tämän lisäksi luokalla on metodi, joka palauttaa hirviölistan koon.

`private Monster parseLine(String line)`-metodi parsii sille annetun pilkuilla erotetun merkkirivin, lukee sen taulukkoon ja palauttaa näillä arvoilla uuden `Monster`-luokan ilmentymän.

`private void addSkills(Monster m, String inputline)` ottaa argumenteikseen hirviön `m`, ja merkkijonon `inputline`, ja parsii `inputlineltä` pilkuilla erotettuna hyökkäysten ja loitsujen indeksien arvot. Tämän jälkeen hirviölle `m` lisätään yksi kerrallaan nämä hyökkäykset ja loitsut.

3.1.5 `Attack`-luokka

Vaikka pelissä on kahdenlaisia hyökkäyksiä, loitsuja ja fyysisiä hyökkäyksiä, ovat ne kummatkin saman `Attack`-luokan ilmentymiä. Näitä kahta eri hyökkäystyyppiä erottaa `enum AttackType`, joka voi saada arvot `PHYSICAL` tai `MAGICAL`. Tämä luokka sisältää seuraavat attribuutit:

`AttackType type` Hyökkäyksen tyyppi

`int power` Hyökkäyksen oma voimakkuus

`String name` Hyökkäyksen nimi

`int mana` Hyökkäykseen tarvittavat taikapisteet

Tämän luokan alustusmetodin kutsu on `public Attack(String name, AttackType type, int power, int mana)`, joka asettaa luokan attribuuteille omat arvot. Tarvittaessa `int manan` voi jättää tyhjäksi, jolloin ylikuormitettu alustusmetodi asettaa sen arvoksi nollan. Luokan metodeille on olemassa vain havainnointimetodit, sillä tämän luokan kaikki ilmentymät luodaan muuttumattomiksi tekstitiedostosta.

3.1.6 `Consumable`-luokka

Tämä luokka pitää sisällään tiedot käytettävien esineiden piirteistä. Luokalla on seuraavat attribuutit havainnointi, ja asetusmetodeineen:

`String consumableName` Käytettävissä olevan esineen nimi

`int restoreHP` Esineen palauttamat elämäpisteet

`int restoreMana` Esineen palauttamat taikapisteet

`int uses` Esineen käyttökerrat

Luokan alustusmetodin signatuuri on muotoa `public Consumable(String name, int hp, int mana, int uses)`

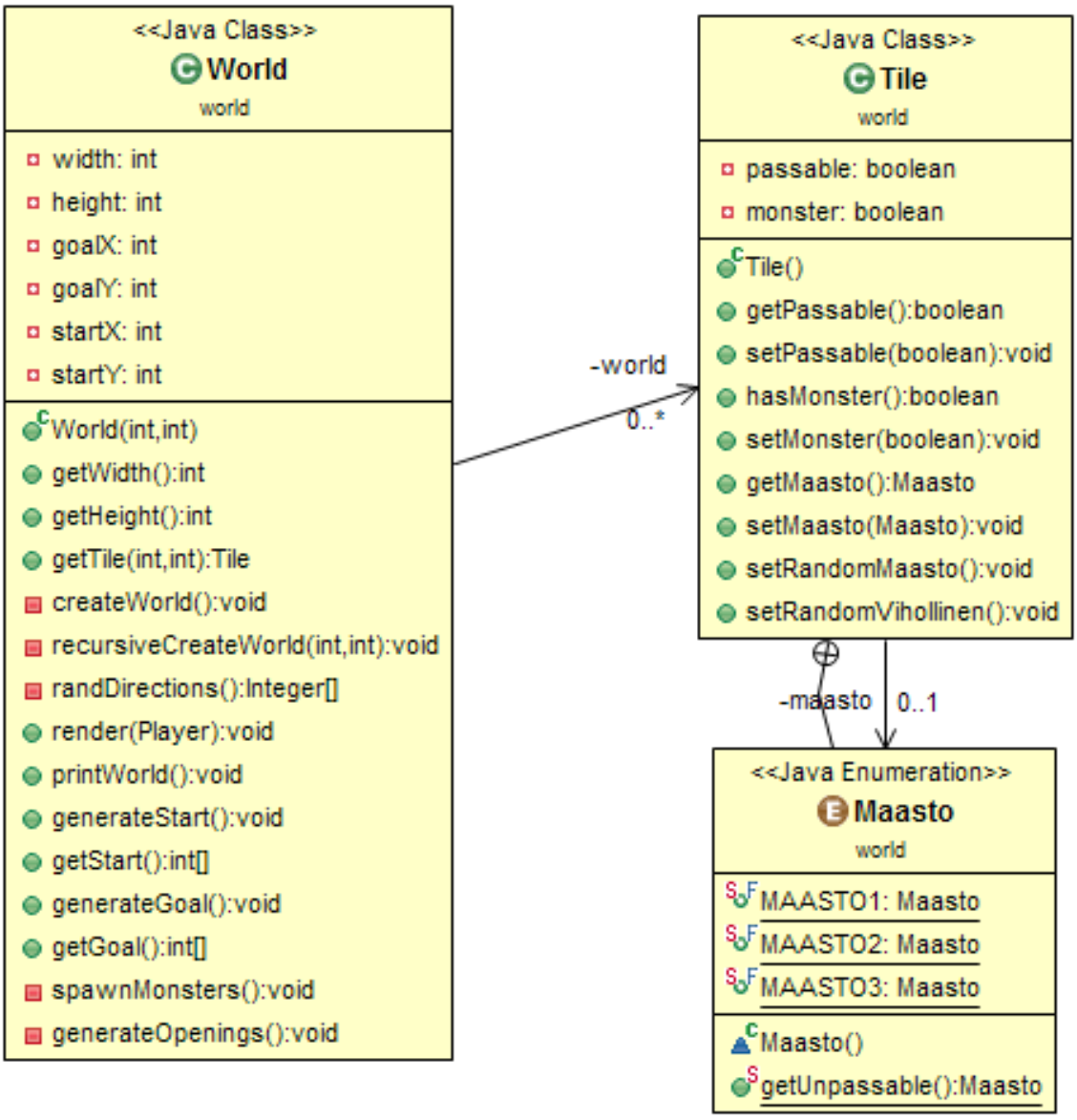
3.1.7 `AttackIDList`-, `SpellIDList`- ja `ItemGenerator`-luokat

Nämä luokat ovat vastuussa hyökkäysten, loitsujen ja käytettävien esineiden lukemisesta tiedostosta, ja ovat toiminnaltaan riittävän samankaltaisia että kuvaamme niitä tässä yhdessä kappaleessa. Nämä luokat muistuttavat toiminnallisuudeltaan `MonsterGenerator`-luokkaa, mutta parsintalogiikassa on pieniä poikkeuksia. Kaikki nämä luokat hyödyntävät kahta listaa, ensimmäistä johon tallennetaan kaikki hyökkäysten tai esineiden ilmentymät.

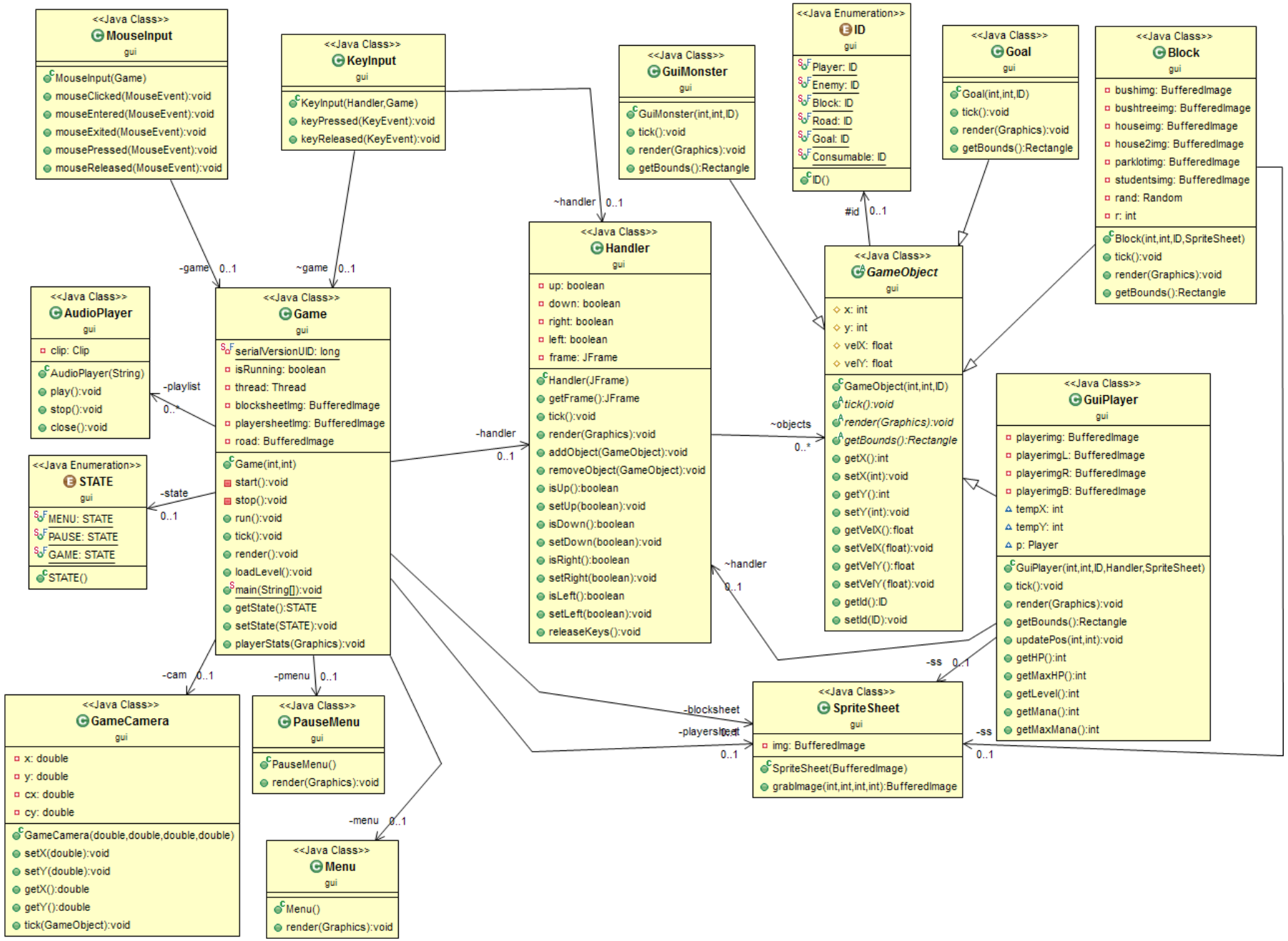
Kaikki nämä luokat lukevat oman objektinsa utf-8 -muotoisesta tekstitiedostosta riveittäin pilkulla erotettuina tietueina. Toiminnallisuudeltaan nämä luokat noutavat resurssin jota halutaan lukea, lukevat sen riveittäin `BufferedReader`-oliota hyväksikäyttäen ja tallentavat tämän rivin toiseen tekstimuotoiseen listaan. Lukurutiini voi nostaa `IOException`-poikkeuksen jos tiedostoa ei voida lukea, tai `FileNotFoundExceptionin` jos tiedostoa ei löydetä. Kaikki nämä luokat sisältävät samat metodit.

`private parseLine(String line`-metodi palauttaa aina `Attack`- tai `Consumable`-muotoisen olion, joka luetaan sille parametrina annetusta merkkijonosta `line`. Merkkijono erotetaan osiin pilkkujen kohdalta taulukkoon, ja taulukon sisältämien tietojen pohjalta luodaan uusi ilmentymä oliosta(joka riippuu minkä luokan `parseLine`-metodia kutsutaan.

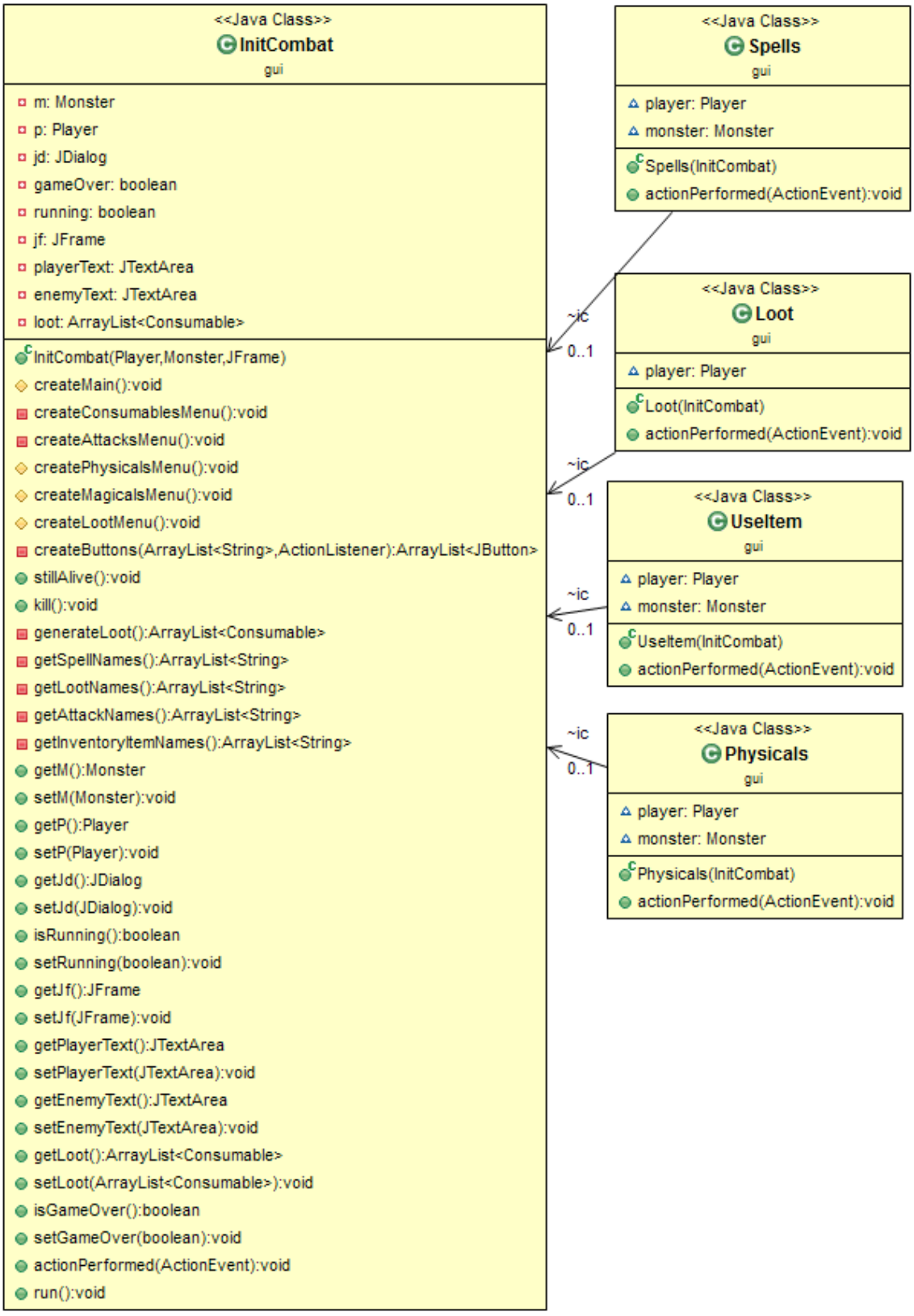
Kaikki nämä luokat myös sisältävät havainnointimetodit, jotka palauttavat tunnettujen olion ilmentymien listasta joko pituuden tai olion parametrina annetun indeksin `i` kohdalta.



Kuva 2: Luokkakaavio `world`-paketista



Kuva 3: Luokkakaavio `gui`-paketin keskeisistä elementeistä



Kuva 4: Luokkakaavio `InitCombat`-metodista

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 löydettiinkin ennakkoimattomia virheitä ja poikkeustilanteita, jotka saatiin korjattua melko ripeästi. Pelissä ei nykytietämyksen mukaan ole mahdollisia virhetilanteita.

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