

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	4
2.5	Graafinen käyttöliittymä	4
2.6	Pelitilanteen tallennus ja lataus	5
3	Ohjelman ja sen osien kuvaaminen	5
3.1	combat-paketti	5
3.1.1	Creature-luokka	5
3.1.2	Player-luokka	7
3.1.3	Monster-luokka	7
3.1.4	MonsterGenerator-luokka	8
3.1.5	Attack-luokka	9
3.1.6	Consumable-luokka	9
3.1.7	AttackIDList-, SpellIDList- ja ItemGenerator-luokat	9
3.2	world-paketti	10
3.2.1	Tile-luokka	11
3.2.2	world-luokka	11
3.3	gui-paketti	12
3.3.1	Game-luokka	12
3.3.2	Handler-luokka	14
3.3.3	GameObject-luokka	15
3.3.4	Block-luokka	15
3.3.5	Goal-luokka	16
3.3.6	GuiMonster-luokka	16
3.3.7	GuiPlayer-luokka	17
3.3.8	GameCamera-luokka	18
3.3.9	InitCombat-luokka	18
3.3.10	Loot-luokka	20
3.3.11	Spells-luokka	21
3.3.12	Physicals-luokka	21
3.3.13	UseItem-luokka	21
3.3.14	CombatOutPutStream-luokka	22
4	Testausjärjestely	25
4.1	combat-paketti	25
4.1.1	CombatEngine- ja CombatTest-luokat	25
5	Liitteet	26
5.a	Alkuperäinen tehtävänanto	26
5.b	Ohjelmalistaus	26
5.c	Käyttöohje	26

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 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 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 karttaruudusta satunnaislukuja generoimalla, ja kohtaamisen tullessa vastaan käynnistetään pelin kohtaamisnäkyvä.

2.1 Pelaajan ja vihollisten luokat

Taistelujärjestelmän näkökulmasta pelaaja ja vihollinen ovat vain yliluokan Creature ilmentymiä, joilla on jaettuja attribuutteja(esimerkiksi nimi, osu-mapisteet, 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 kokemustaso saavuttaa(sekä myös itse kokemustason ja ilmentymien statistikkojen nosto)

Pelihahmon luokalla on myös yliluokkansa sisältämien tietojen lisäksi tietomissä kohtaa pelihahmo sijaitsee karttaruudusta, sekä tälle asetus- ja havainnointimetodit. Näiden lisäksi pelihahmole 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 kokemustason 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 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 kokemustaso on, ja nostaa sitä ennen vihollisolon palauttamista.

2.3 Taistelumoottori

Itse taistelumoottori järjestää taistelun pelaajan ja vihollisolon 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 nolleen tai sen alle, ja jos pelaaja selvi elossa annetaan pelaajalle vihollisen kestävyyspisteiden verran kokemuspisteistä. Tämän lisäksi on pelaajalla mahdollisuus saada satunnaisesti generoituja esineitä, joiden määrä perustuu vihollisen kokemustasoon.

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 64×64 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 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 esinelistaus saattaa vaihdella, luodaan käyttöliittymäelementit siihen lennosta

2.6 Pelitilanteen tallennus ja lataus

Pelitilanne on mahdollista tallentaa ja ladata käyttämällä hyväksi pelimootorin 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-luokka

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

int maxHP Olion suurimmat mahdolliset kestävyyspisteet

int hp Olion tämänhetkiset kestävyyspisteet

int strength Olion voima

int magic Olion taikavoima

int defense Olion puolustus

int mana Olion taikapisteet

int maxMana Olion suurimmat mahdolliset taikapisteet

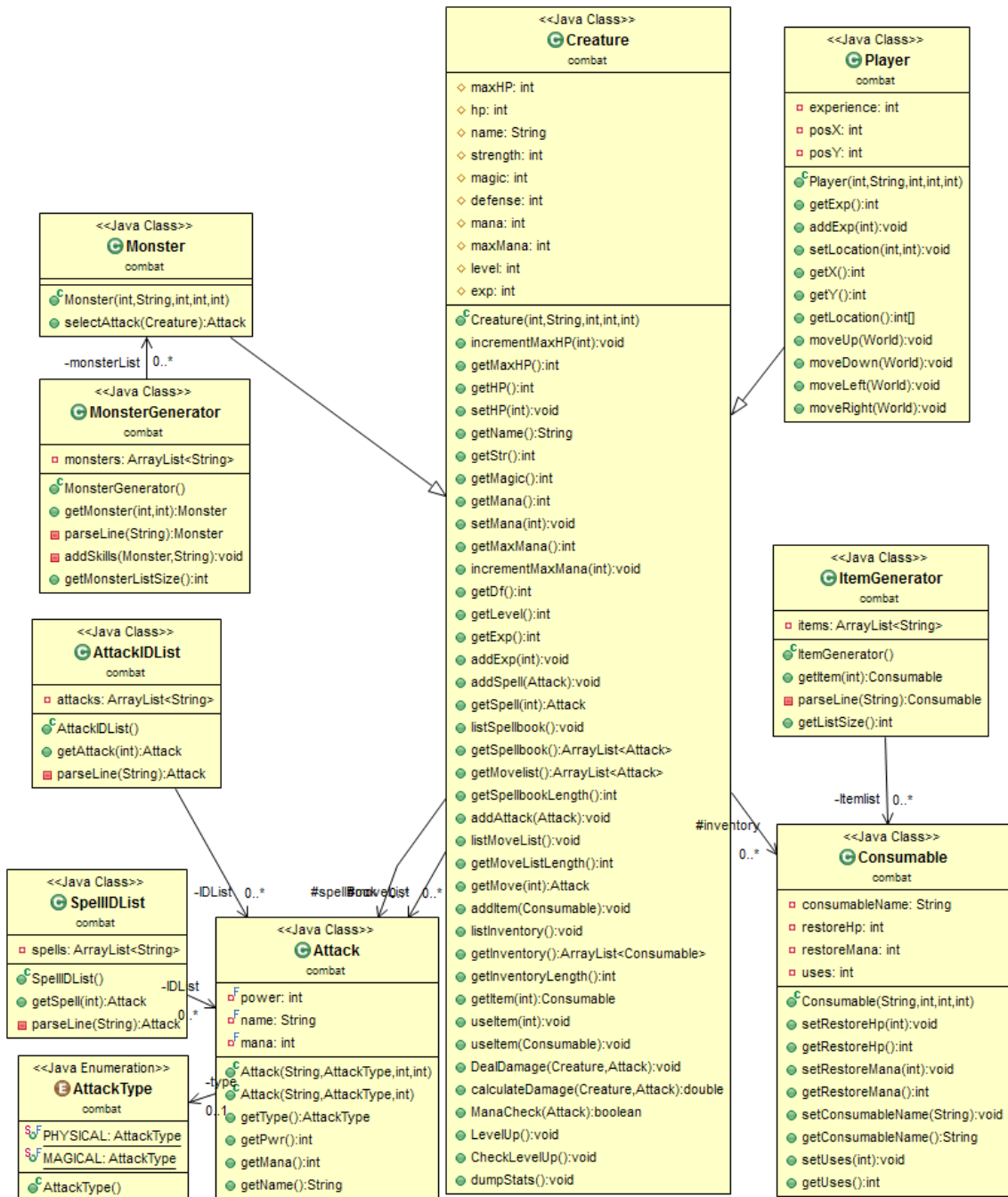
int level Olion kokemustaso

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-moduuli 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.



Kuva 1: Luokkakaavio combat-paketista

`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 parametrikseen Creature-luokan ilmentymän, sekä Attack-luokan ilmentymän a, ja antaa kutsuu näillä parametreilla `calculateDamage`-metodia. `calculateDamage` paluuarvo perusteella vähennetään defenderin elämäpisteistä a:n tekemä vahinko, ja Creaturelta, jonka `DealDamage`-metodia kutsutaan vähennetään hyökkäyksen 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 kokemustasolle. Jos tämä toteutuu, kutsuu se `LevelUp`-metodia

`public void LevelUp()`-niminen metodi kutsuu ei palauta mitään, vaan se suorittaa seuraavalle kokemustasolle 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 argumentina 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 kokemustaso asetetaan tasolle 1.

public Attack selectAttack(Creature target-metodi laskee käyttäen hyväksi Creature-luokan CalculateDamage-metodia hyökkäyksen, jolla saadaan suurin vahinko aikaiseksi target-Creatureem. 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 itse 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 tiedämät 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 argumentina parseLine (joka lisää hirviön tiedettyjen hirviöiden listaan) tai addSkills (joka lisää hirviölle sen hyökkäykset) -metodeille, riippuen siitä onko loitsujen ja hyökkäysten erottimena käytetty merkki ''-länä 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.

public Monster getMonster(int i, int lvl)-metodi hoitaa vihollisen palautuksen. Se ottaa argumentteinaan kokonaisluvun i, joka on vihollisen indeksinumero listalla, sekä kokonaisluvun lvl, joka on kokemustaso, jolla vihollista halutaan kutsua. Funktio palauttaa Monster-luokan ilmentymän, jonka kokemustasoa nostatetaan kunnes se saavuttaa satunnaisesti arvotun tason suhteessa kokemustasoon jolla funktiota kutsutaan. Vihollishirviön kokemustaso arvotaan kokonaisluvuiksi pyöristettynä välille:

$$\left[lvl - \frac{lvl}{10}, lvl + \frac{lvl}{10} \right]$$

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 palauttavat elämäpisteet

`int restoreMana` Esineen palauttavat taikapisteet

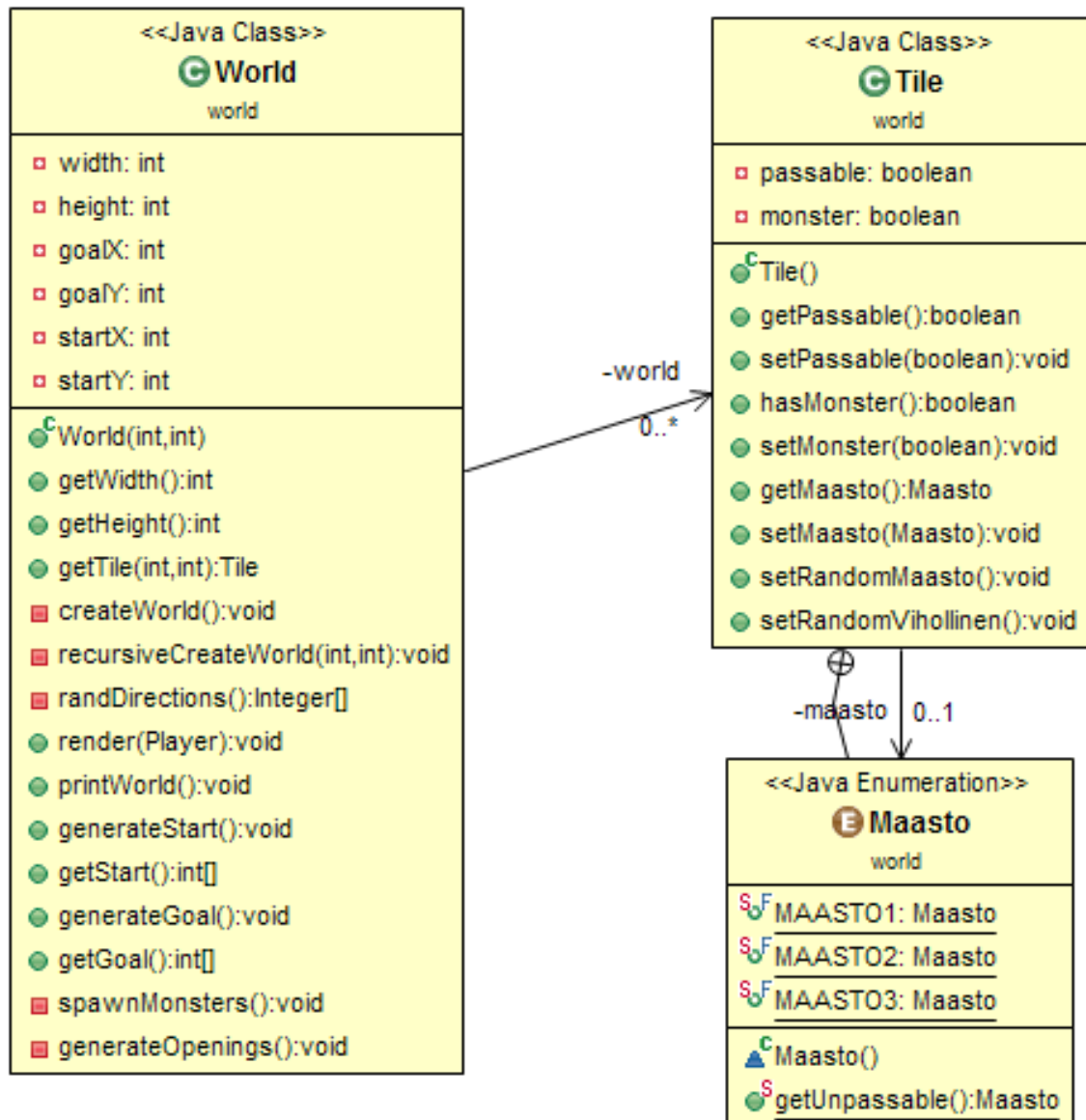
`int uses` Esineen käyttökerrat

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 `FileNotFoundException` jos tiedostoa ei löydetä. Kaikki nämä luokat sisältävät samat metodit.



Kuva 2: Luokkakaavio world-paketista

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.

3.2 world-paketti

Tämä paketti pitää sisällään kaikki luokat, joita tarvitaan pelimaailmaa luodessa.

3.2.1 Tile-luokka

Koska kartta koostuu 64×64 -kokoisista tileistä, joilla on omia ominaisuuksiaan toteutettiin nämä tilet omana luokkanaan. Tileillä on olemassa seuraavat attribuutit:

`boolean passable` Tieto siitä, pystyykö maaston lävitse kävelemään

`boolean monster` Tieto siitä, onko maastoruudussa ”piilossa” kohtaaminen vihollisen kanssa

Läpipääsemättömillä maastoilla on enum `Maasto`, joka voi saada arvot `MAASTO1`, `MAASTO2` ja `MAASTO3`, joille jokaiselle ladataan erikseen omat grafiikkansa käyttöliittymän puolella. `public static Maasto getUnpassable()`-metodi arpoo jokaiselle näistä maastoista omat enum-tyypinsä. Tämä päädyttiin toteuttamaan sen takia, että pelkkiä kahdenlaisia Tilen grafiikkoja sisältämä karttanäkymä on turhan yksikertainen.

Havainnointi, ja asennusmetodiensa lisäksi luokalla on omia metodejaan.

`public void setRandomVihollinen()` käyttää javan satunnaislukugenerointia karttaruudun kohdalla ja jos maasto on läpi käveltävää muotoa, arvotaan siihen totuusarvo sisältääkö ruutu taistelun vai ei.

`public void setRandomMaasto()`-metodi asettaa Maaston tyyppin karttaruudun perusteella. Jos maastosta pystytään kävelemään lävitse, asetetaan sille `Maasto.MAASTO1`, ja muussa tapauksessa arvotaan sille toisenlainen `Maasto`.

3.2.2 world-luokka

Tämä luokka sisältää metodit ja attribuutit joita tarvitaan itse pelikartan luomisessa. Pelikartta on yksinkertaisesti kaksiulotteinen taulukko, joista jokainen taulukon solu pitää sisällään yhden Tile-luokan ilmentymän. Luokalla on seuraavat attribuutit:

`int width` Kartan koko leveyssuunnassa

`int height` Kartan koko pystysuunnassa

`int goalX` Maalipisteen x-koordinaatti

`int goalY` Maalipisteen y-koordinaatti

`int startX` Alkupisteen x-koordinaatti

`int startY` Alkupisteen y-koordinaatti

`Tile[][] world` Kaksiulotteinen taulukko kartan sisältämistä karttaruuduista

`world`-luokkaa kutsuttaessa sen signatuuri on muotoa `public World(int width, int`

`height)`, jossa argumentteina asetetaan kartan koko. Tämän jälkeen konstruktori alustaa uuden taulukon, joka on `width × height`-kokoinen, ja luo taulukon soluihin uuden instanssin `Tile`-oliosta. Tämän jälkeen konstruktori kutsuu muita luokkansa metodeja, jotka luovat itse kartan, asettavat siihen alku-, ja lopetuspisteensä sekä luo karttaan aukkoja ja viholliskohtaamisia.

`private void recursiveCreateWorld(int r, int c)`-metodi on rekursiivinen metodi, jota käytetään maailman luomisessa. Sen tehtävä on luoda *depth first*-hakumetodilla ”täydellinen”, eli ratkaistavissa oleva labyrintti. Tämä tapahtuu siten, että metodi ottaa parametrinsa `r:n` ja `c:n` aloituspisteidensä koordinaateiksi, ja lähtee `randDirections`-metodin antamiin suuntiin kiemurtelevia kulkuväyliä asettamalla naapurikarttaruudut läpikuljettaviksi. Metodi jatkaa kulkuun rekursiivisesti, kunnes se törmää kartan ulkoseinään.

`private void createWorld()` on metodi, joka kutsuu `recursiveCreateWorld`-metodia. Tämä arpoo satunnaislukuina aloituspisteen, jonka `x-` ja `y-`koordinaatit ovat kahdella jaollisia(jotta seinän tunnistava aritmetiikka toimisi `recursiveCreateWorld`-metodilla), asettaa aloituspisteensä läpikuljettavaksi ja kutsuu `recursiveCreateWorld`-metodia.

`private Integer[] randDirections()`-metodi joka luo ja palauttaa kokonaislukutaulukon, johon tallennetaan satunnaisessa järjestyksessä neljä suuntaa ilmaistuna kokonaisluvuilla.

`public void generateStart ()`-metodi luo satunnaisen aloituspisteen karttaruutuun, jonka lävitse pystyy kävelemään.

`public void generateGoal ()`-metodi puolestaan luo kartan satunnaiseen kulmaan lopetuspisteen.

`private void spawnMonsters()` on metodi, joka käy karttaa karttaruutu kerrallaan lävitse, ja satunnaislukugeneroinnilla arpoo pelaajan läpikäveltäviin ruutuihin satunnaiskohtaamisen asettamalla karttaruudulle `setMonster(true)`.

`private void generateOpenings()`-metodi käy karttaa lävitse karttaruutu kerrallaan, ja luo satunnaisenkokoisia nelikulmaisia alueita, joille asetetaan karttaruutuihin `setPassable(true)`.

3.3 gui-paketti

Tämä paketti pitää sisällään kaikkeen graafiseen käyttöliittymään sidotun toiminnallisuuden.

3.3.1 Game-luokka

`Game`-luokka pitää sisällään itse ohjelman päärunгон, ja se kutsuu tarvittaessa muita luokkia suorittamaan ohjelman toiminnallisuutta. Tällä luokalla on seuraavat attribuutit:

`FontLoader fl` Fonttien lataamiseen käytetty luokka

`boolean isRunning` Totuusarvo onko pelin prosessi pystyssä

`Thread thread` Pelin suorituksen yksi säie

`Handler handler` Luokka, joka pitää sisällään pelimaailman objektit ja huolehtii niiden päivityksestä ruudulla

`GameCamera` `cam` Luokka, joka pitää huolen karttaikkunan näkymästä ja sen muutoksista
`SpriteSheet` `blocksheet` Luokka, joka pitää sisällään tiedot mikä kuva ladataan isomasta kuvasta grafiikkoja varten

`BufferedImage` `blocksheetImg` Itse kuvan sisältävä luokka

`SpriteSheet` `playersheet` Luokka, joka pitää sisällään tiedot mikä kuva ladataan pelaajan kuvia sisältävästä kuvasta

`BufferedImage` `playersheetImg` Luokka, joka pitää sisällään pelaajan kuvan

`BufferedImage` `road` Luokka, joka pitää sisällään kuvan tiestä

`BufferedImage` `bus` Luokka, joka pitää sisällään kuvan loppuruudusta

`STATE` `state` Tieto missä tilassa peli on, `enum`

`Menu` `menu` Luokka, jossa toteutetaan pelin päävalikko

`PauseMenu` `pmenu` Luokka, jossa toteutetaan pelin taukovalikko

`AboutMenu` `amenu` Luokka, jossa toteutetaan pelin informoitu

`StartScreen` `startsscreen` Luokka jolla toteutetaan pelin aloitusnäky

`Font` `font1` Pelin fontteja

`Font` `font2` Pelin fontteja

Game-luokan konstruktori on muotoa `public Game(int x, int y)`, jossa kokonaislukuparametrit ovat ikkunan leveys ja pituus pikseleinä. Tämän jälkeen konstruktori luo pelille uuden ikkunan, luo kaikki tarvittamansa elementit (ja tarvittaessa kutsuu muiden luokkien metodeja lukeakseen näitä tiedostoista) ja soittaa pelimusiikkia. Konstruktori myös luo uudet kuunteluluokat näppäimistö- ja hiirisyötteelle ja liittää ne itse pelilogiikkaan. Tämän lisäksi luokka myös aloittaa pelisäikeen suorituksen.

`private void start()`- ja `private void stop()`-metodit aloittavat ja lopettavat pelin prosessin säikeen.

`public void run()`-prosessi hoitaa itse pelin pyörittämistä, ja se huolehtii pelin ajanhallinnasta mikrosekunteinä kuvaruudun piirtämisestä varten, ja kutsuu pelin piirtometodia. Nostaa `IllegalStateException`in, jos peli ei voi piirtää uutta ruutua.

`public void tick()`-metodi pitää huolen ruudunpäivityksestä, ja käy yksi kerrallaan läpi listaa peliin luoduista objekteista, ja päivittää näiden sijaintia pelinäköymässä, jos pelin tila on `GAME`.

`public void tick()`-metodi piirtää pelinäköymän ruudulle kaksiulotteisena kuvana. Jos pelinäköymä on jossain valikossa, se piirtää valikon. Muussa tapauksessa se piirtää pelin eri elementit karttanäköymästä.

`public void loadLevel()`-metodi hoitaa pelikartan luomisen. Se aloittaa luomalla uuden kartan käyttämällä `World.world`-luokan metodeja, ja käy kartan kaksiulotteisen taulukon yksi solu kerrallaan läpi. Jokaiselle solulle luodaan oma peliobjekti `handler`-luokkaan, jolle asetetaan oikeat grafiikat ja peliominaisuudet pelikartan arvojen mukaan.

`public STATE getState()` ja `public STATE setState()` ovat asetus, ja havainnointimetodeja ohjelman tilasta, jota käytetään päättämään mitä näkymiä piirretään.

`public void playerStats(Graphics2D g)`-metodi on apumetodi, jota käytetään piirtämään graafiset elementit, jotka välittävät tietoa pelaajan tilasta pelajalle.

3.3.2 Handler-luokka

Tämä luokka pitää kirjaa kaikista peliin piirrettävistä objekteista ja kutsuu piirtorutiineja niille. Luokalla on seuraavat attribuutit:

`ArrayList<GameObjects> objects` Lista kaikista pelin objekteista

`boolean up` Tieto siitä onko annettu syöte liikkua ylös

`boolean down` Tieto siitä onko annettu syöte liikkua alas

`boolean right` Tieto siitä onko annettu syöte liikkua oikealle

`boolean left` Tieto siitä onko annettu syöte liikkua vasemmalle

`JFrame frame` Peli-ikkunan sisältö, ns *frame*

Luokan konstruktorin signatuuri on muotoa: `public Handler(JFrame frame)`, jossa `frame` on se ikkunan sisältämä *frame*, johon tämän luokan tekemät muutokset kohdistuvat. Tälle *framelle* on myös havainnointimethodi `public JFrame getFrame`, joka palauttaa kyseisen *framen*.

`public void tick()`-metodi käy läpi `objects`-listaa, ja yksi kerrallaan kutsu jokaisen objektin omaa päivitysmetodia.

`public void render(Graphics2D g)`-metodi käy vastaavasti läpi listaa jokaisesta peliobjektista, ja kutsuu näille niiden omia piirtometodeja.

`public void addObject(GameObject temp)` ja `public void removeObject(GameObject temp)`-nimiset metodit lisäävät ja poistavat parametrina saadun peliobjektin `temp` `objects`-listalta

`public boolean isUp()`, `public boolean isDown()`, `public boolean isRight()` ja `public boolean isLeft()` ovat havainnointimetodeja syötteen suunnille. Näille on myös vastaavat asetusmenodit `public void setUp(boolean up)`, `public void setDown(boolean down)`, `public void setRight(boolean right)` ja `public void setLeft(boolean left)`.

`public void releaseKeys()`-metodi ”vapauttaa” jokaisen annetun suunnan syötteen asettamalla näiden muuttujien arvoiksi `false`.

3.3.3 GameObject-luokka

Tämä luokka on abstrakti luokka, jolla kuvaillaan muille luokille eri menet-
toja niiden tulee toteuttaa. GameObject-luokan perivillä luokilla kuuluu olla
seuraavat attribuutit:

`int x` Ilmentymän sijainti x-akselin suhteen

`int y` Ilmentymän sijainti y-akselin suhteen

`float velX` Ilmentymän nopeus x-akselilla

`float velY` Ilmentymän nopeus y-akselilla

`ID id` Ilmentymän ID, enum

GameObject-luokan konstruktorin signatuuri on muotoa `public GameObject(int x, int y, ID id)`, jossa sille annetaan kokonaislukuparametreina sen sijainti x- ja y-akselien suhteen. Konstruktori myös olettaa lähtönopeudet nolllaksi (ts. objekti on liikkumaton), ja antaa sille objektiin sidotun ID:n

Asetus- ja havainnointimetodiensa lisäksi luokalla tulee olla määriteltynä myös
muutama muu metodi. `public void tick()`-metodilla määritellään peliobjek-
tien päivittäminen. `public void render(Graphics2D g)` ottaa parametrikseen
grafiikkaobjektin, johon GameObjectin alaluokan tulee piirtää itsensä. Lopuk-
si vielä luokan alaluokilla tulee olla määriteltynä `public abstract Rectangle`
`getBounds()`-niminen metodi, joka palauttaa neliökappaleena peliobjektin ul-
korajat törmäyksien tarkistamista varten.

3.3.4 Block-luokka

Tämä luokka pitää sisällään kaikki seininä toimivien objektien määrittelyt, ja
se on GameObject-luokan alaluokka. Sillä on seuraavat attribuutit:

`BufferedImage bushimg` Eräs seinämien grafiikoista

`BufferedImage bushtreeimg` Eräs seinämien grafiikoista

`BufferedImage houseimg` Eräs seinämien grafiikoista

`BufferedImage house2img` Eräs seinämien grafiikoista

`BufferedImage parklotimg` Eräs seinämien grafiikoista

`BufferedImage studentsimg` Eräs seinämien grafiikoista

`Random rand` Satunnaislukugeneraattori

`int r` Kokonaislukumuuttuja satunnaisluville

`Rectangle bounds` Neliökappaleena annetut peliobjektin rajat

Luokan konstruktori on muotoa `public Block(int x, int y, ID id, SpriteSheet ss)`, ja se ottaa parametreinaan sijaintinsa ja ID:n. `ss` on ns. *spritesheet*, josta puolestaan ladataan tekstuurit kaikille `BufferedImage`-grafiikoille. Kokonaislukumuuttujaan `r` asetetaan kokonaisluku, jonka mukaan päätetään mikä grafiikka piirretään ruudun kohdalle.

`public void tick()`-metodi lisää sijaintiin `x`- ja `y`-akseleilla niiden akselien mukaisen nopeuden

`public void render(Graphics2D g)`-metodi piirtää satunnaisluvun määräämän grafiikan parametrina saadulla grafiikkaobjektilla olion koordinaatteihin.

`public Rectangle getBounds()`-metodilla asetetaan tämän objektin määräämät rajat sen määräämän keskipisteen mukaan, ja palauttaa tämän neliökappaleena.

3.3.5 Goal-luokka

Goal-luokka mallintaa pelin lopetusruutua, ja se on `GameObject`-luokan alaluokka. Lisäksi on huomioitava, että vaikka luokan pitääkin toteuttaa metodi `public void tick()`, on tämä metodi tyhjä. Luokalla on seuraavat attribuutit:

`BufferedImage bus` Maaliruudun grafiikka

`public void render(Graphics2D g)`-metodi piirtää maaliruudun grafiikan maaliruudun koordinaatteihin kartalla käyttäen parametrina saamansa piirto-objektin metodia.

`public Rectangle getBounds`-metodi palauttaa uuden neliö-objektin, joka mallintaa maaliruudun reunoja.

3.3.6 GuiMonster-luokka

Tämä luokka mallintaa ruutua, jossa on satunnaiskohtaaminen vihollisen kanssa pelin objektina. `GuiMonster` on luokan `GameObject` alaluokka.

Luokan konstruktori on muotoa `public GuiMonster(int x, int y, ID id)`, ja se ottaa parametreikseen sijaintinsa ja ID:nsä, ja antaa nämä parametrit ylliluokan konstruktorille.

`public void tick()` päivittää objektin sijaintia pelimaailmassa sen nopeuden mukaan

`public void render(Graphics2D g)` ei piirrä mitään, satunnaiskohtaamisten on pysyttävä yllätyksenä pelaajalle ja niiden piirtäminen kartalle paljastaisi ne (Testausvaiheessa tämä metodi piirsi kartalle karttaruudun kokoisen sinisen neliön).

`public Rectangle getBounds()`-metodi palauttaa karttaruudun rajat neliöobjektina.

3.3.7 GuiPlayer-luokka

Tämä luokka mallintaa pelaajan avataria pelin karttanäkymällä, ja se on myös luokan `GameObject` alaluokka. Tällä luokalla on seuraavat omat attribuutit:

`Handler handler` `Handler`-luokan ilmentymä joka pitää kirjaa syötteestä ja ruudunpäivityksestä

`SpriteSheet ss` *Spritesheet*, josta ladataan pelaajan grafiikat

`BufferedImage playerimg` pelaajan grafiikka alaspäin liikkeessä

`BufferedImage playerimgL` pelaajan grafiikka vasempaan liikkeessä

`BufferedImage playerimgR` pelaajan grafiikka oikeaan liikkeessä

`BufferedImage playerimgB` pelaajan grafiikka ylöspäin liikkeessä

`int tempX` väliaikainen sijainti x-akselin suhteen

`int tempY` väliaikainen sijainti y-akselin suhteen

`Player p` `combat`-paketin `Player`-luokan ilmentymä jota käytetään taisteluissa

`Rectangle bounds` pelaajan rajat neliöobjektina

Luokan konstruktori on muotoa `public GuiPlayer(int x, int y, ID id, Handler handler, SpriteSheet ss)`, jossa `x` ja `y` ovat pelaajan sijainti kartalla, ja `ID` kertoo pelimoottorille että kyseessä on juurikin pelaajan hahmosta kyse. Konstruktori antaa nämä eteenpäin ylikuokan konstruktiorille, ja lataa pelaajan grafiikat *spritesheetistä*. Tämän lisäksi luodaan pelaajan hahmon ympärille neliönä rajat törmäyksen tarkistamista varten, luodaan uusi pelaaja ja annetaan pelaajalle pelaajan tietämät loitsut ja hyökkäykset.

`public void tick()`-metodi käsittelee käyttäjän syötteen kysymällä sitä `Handler`-luokalta, ja muuttaa pelaajan nopeuksia sen mukaan. Tämän jälkeen kutsutaan `updatePos()`-metodia päivittääkseen pelaajan paikkaa kartalla.

`public void render(Graphics2D g)`-metodi piirtää pelaajan hahmoa uuteen sijaintiin kartalla sen nopeuksien mukaan käyttäen luokan `Graphics2D` metodia.

`public Rectangle getBounds()` palauttaa pelihahmon ympärille piirretyn neliön rajat törmäyksien tarkistamista varten.

`public void updatePos(int newY, int newX)`-metodi hoitaa pelihahmon siirtymisen pelilogiikan. Se saa parametreinaan *newY* ja *newX* -kokonaisluvut, jotka ovat hahmon uusi sijainti kartalla. Tämän jälkeen tarkistetaan tuleeko törmäys seinän kanssa, jolloin pelaaja ei liiku, tai vihollisen kanssa jolloin alustetaan uusi taistelu. Metodi myös hoitaa kaikki tarvittavat toimenpiteet jotka seuraavat taistelua.

3.3.8 GameCamera-luokka

Tämä luokka pitää huolta, että pelaajan hahmo sijaitsee keskellä ruutua, ja että maailma näkyy pelaajan ympärillä. Sillä on seuraavat attribuutit:

double x Kameran sijainti x-akselilla

double y Kameran sijainti y-akselilla

double cx Ikkunan leveys

double cy Ikkunan korkeus

Luokalla on asetus, ja havainnointimetodit x:lle ja y:lle. Näiden lisäksi luokalla on metodi public void tick(GameObject player), joka saa parametrinaan peliobjektin. Kameran uudet x- ja y-koordinaatit lasketaan pelaajan koordinaatin ja ruudun koon perusteella, kuitenkin siten että kamera ei näytä kartan ulkopuolelle.

3.3.9 InitCombat-luokka

Tämä luokka pitää huolen pelaajan ja vihollisen välisestä taistelusta, ja se toteuttaa ActionListener- ja Runnable-rajapintoja. Luokka avaa uuden ikkunan taistelunäkymään, samalla kun maailmanäkymän suoritussäie jää tauolle. Luokalla on seuraavat attribuutit:

Monster m Taistelun vihollinen

Player p Pelaajan ilmentymä

JDialog jd Taistelun dialogi-ikkuna

boolean gameOver Tila siitä päätyikö peli taisteluun

boolean running Pyöriikö pelin säie

JFrame jf Ikkunaan piirrettävä sisältö

JTextArea playerText Pelaajan tietojen näyttämiseen tarvittava tekstialue

JTextArea enemyText Vihollisen tietojen näyttämiseen tarvittava tekstialue

ArrayList<Consumable> loot Lista palkinnoksi saatavista esineistä

JTextArea jt Tekstialue jossa taistelun viestit näytetään

PrintStream ps Tulostusvirta josta taistelun viestit tulevat

JScrollPane jsp Ylös- ja alaspäin rullattava käyttöliittymäelementti taistelun viesteille

JPanel selectAction Käyttöliittymäalue toimintaa valitseville elementeille

JPanel information Informaation näyttävä käyttöliittymäalue

JPanel selectConsumable Esineiden käyttämisen käyttöliittymäalue

JPanel selectAttackType Hyökkäyksen tyyppin valitsemisen käyttöliittymäalue

Jpanel scrollBox Tekstin vierityslaatikko

JButton mag Loitsut valitseva käyttöliittymäpainike

JButton phy Hyökkäykset valitseva käyttöliittymäpainike

FlowLayout layout Taisteluikkunan elementtien sommittelu

Dimension textbox Tekstialueen mitat Dimension-elementtinä

DefaultCaret caret Taistelun tekstialueen alavierittävä olio

Luokan konstruktori on muotoa `public InitCombat(Player p, Monster m, JFrame jf)`, ja se saa parametrinaan pelaajan `p` ja hirviön `m`, joiden välillä taistelu tapahtuu. `jf` on *frame*, johon ikkunan sisältö piirretään. Lisäksi taistelun tilalle asetetaan arvo `true`, ja konstruktori vielä alustaa käyttöliittymäelementit.

`protected void createMain()`-metodi luo uuden taisteluikkunan, ja piirtää siihen kaikki sen tarvitsemat käyttöliittymäelementit. Metodi myös kiinnittää nappeihin niiden tarvitsemat `ActionListenerit`.

`protected void refresh()`-metodi poistaa kaikki piirretyt käyttöliittymän elementit paitsi taistelun tulosteen, että ne voidaan piirtää uudestaan (näin vältetään monen päällekkäisen elementin ongelma).

`private void createConsumablesMenu()`-metodi luo lennosta pelaajan `p` inventaarion perusteella käyttöliittymäpainikkeet jokaiselle käytettävissä olevalle esineelle.

`private void createAttacksMenu()`-metodi luo käyttöliittymäpainikkeet loitsujen tai fyysisten hyökkäysten valintaan

`private void createPhysicalsMenu()` luo lennosta pelaajan tietämistä hyökkäyksistä käyttöliittymäpainikkeet.

`private void createMagicalsMenu()` luo jokaiselle loitsulle oman käyttöliittymäpainikkeen lennosta.

`private void createLootMenu()` luo jokaiselle palkinnoksi saadulle tavaralle oman käyttöliittymäpainikkeen, jos niitä ylipäänsä on.

`private ArrayList<JButton> createButtons(ArrayList<String> names, ActionListener al)` saa parametreinaan listan painikkeiden nimistä ja niihin liittyvästä `ActionListeneristä`. Metodi luo nimistä painikkeet ja liittää niihin `ActionListenerin`, ja lisää painikkeen listaan. Tämän jälkeen listaan lisätään painike, joka palaa valikkohierarkiassa ylöspäin. Tämän jälkeen metodi palauttaa listan painikkeista.

`public void stillAlive()`-metodia käytetään tarkistamaan jokaisen tapahtuman jälkeen onko pelaaja ja vihollinen vielä elossa. Jos pelaaja kuolee, ohjelma näyttää oikean dialogin, ja jos vihollinen kuolee pelaaja palkitaan kokemuspisteillä ja mahdollisesti esineillä jos niitä löytyy tästä taistelusta palkintona.

`public void kill()`-metodi asettaa ohjelman ajamista ilmentävän muuttujan arvoksi `false`, ja sulkee ikkunan.

`private ArrayList<Consumable> generateLoot()`-metodi luo uuden listan palkinnoksi saatavista tavaroista taistelulle perustuen vihollisen kokemustasoon, ja palauttaa listan.

`private ArrayList<String> getLootNames()`-metodi lukee palkinnoksi saatavien tavaroiden listan ja lukee tavaroiden nimet sieltä uuteen merkkijonolistaan, joka palautetaan.

`private ArrayList<String> getSpellNames()`-metodi lukee listan pelaajan tietämistä loitsuista, ja lisää jokaisen loitsun palautettavaan merkkijonolistaan.

`private ArrayList<String> getAttackNames()`-metodi lukee listan pelaajan tietämistä hyökkäyksistä, joista jokainen lisätään merkkijonolistaan. Metodi palauttaa tämän merkkijonolistan.

`private ArrayList<String> getInventoryItemNames()`-metodi lukee listan pelaajan esineistä, ja lisää esineiden nimet merkkijonolistaan. Lopuksi metodi palauttaa merkkijonolistan.

`public void actionPerformed(ActionEvent e)` ottaa parametrinaan suoritettua toiminnon `ActionEvent`in. Metodi tyhjentää nykyisen ikkunan, ja kutsuu metodeja, joilla luodaan toiminnosta seuraavat painikkeet käyttöliittymään.

`public void run()`-metodi kutsuu metodia, joka tyhjentää näytön ja kutsuu toista metodia joka piirtää taistelun päänäkökuvan uudestaan.

3.3.10 Loot-luokka

Tämä luokka toteuttaa `ActionListener`-rajapintaa, ja se odottaa kunnes pelaaja painaa palkinnoksi saatavien tavaroiden painikkeita. Sillä on seuraavat attribuutit:

`InitCombat ic` Se `InitCombat`-luokan ilmentymä jonka toimintoja odotetaan

`Player player` Pelaaja, johonka toiminnot kohdistetaan

Luokan konstruktorin signatuuri on muotoa `public Loot(InitCombat ic)`, jossa se saa syötteenään `InitCombat`-olion, johon kuuntelija halutaan liittää. Konstruktori asettaa sen jälkeen attribuuttiensa arvot parametrinaan saamasta oliosta.

`public void actionPerformed(ActionEvent e)`-metodi ottaa parametrikseen `ActionEventin`, jonka mukaan kutsutaan metodeita sulkemaan ikkuna tai lisäätään painikkeella valittu esine pelaajan listaan esineistä, päivitetään ikkuna ja kutsutaan `stillAlive`-metodia tarkistamaan tilanne uudestaan

3.3.11 Spells-luokka

Tämä luokka toteuttaa `ActionListener`-rajapinnan, ja tämän tarkoituksena on odottaa loitsujen painikkeisiin sidottuja tapahtumia ja toimia niiden mukaan. Luokalla on seuraavat attribuutit:

`InitCombat ic` Se taistelun instanssi, jonka tapahtumia kuunnellaan

`Player player` Pelaajan ilmentymä, jota tapahtumat koskevat

`Monster monster` Se vihollisen ilmentymä, jota tapahtumat koskevat

Luokan konstruktori on muotoa `public Spells(InitCombat ic)`, ja se sitoo parametrina saadun `InitCombatin` attribuutit luokan ominaisuuksiin.

`public void actionPerformed(ActionEvent e)`-metodi saa parametrikseen `ActionEventin`, ja kutsuu `InitCombatin` metodeja toteuttamaan `ActionEventin` ilmentämän loitsujen kuvauksen.

3.3.12 Physicals-luokka

Tämä luokka toteuttaa `ActionListener`-rajapinnan, ja tämän tarkoituksena on odottaa hyökkäysten painikkeisiin sidottuja tapahtumia ja toimia niiden mukaan. Luokalla on seuraavat attribuutit:

`InitCombat ic` Se taistelun instanssi, jonka tapahtumia kuunnellaan

`Player player` Pelaajan ilmentymä, jota tapahtumat koskevat

`Monster monster` Se vihollisen ilmentymä, jota tapahtumat koskevat

Luokan konstruktori on muotoa `public Physicals(InitCombat ic)`, ja se sitoo parametrina saadun `InitCombatin` attribuutit luokan ominaisuuksiin.

`public void actionPerformed(ActionEvent e)`-metodi saa parametrikseen `ActionEventin`, ja kutsuu `InitCombatin` metodeja toteuttamaan `ActionEventin` ilmentämän hyökkäysten kuvauksen.

3.3.13 UseItem-luokka

Tämä luokka toteuttaa `ActionListener`-rajapinnan, ja tämän tarkoituksena on odottaa esineiden painikkeisiin sidottuja tapahtumia ja toimia niiden mukaan. Luokalla on seuraavat attribuutit:

`InitCombat ic` Se taistelun instanssi, jonka tapahtumia kuunnellaan

`Player player` Pelaajan ilmentymä, jota tapahtumat koskevat

`Monster monster` Se vihollisen ilmentymä, jota tapahtumat koskevat

Luokan konstruktori on muotoa `public UseItem(InitCombat ic)`, ja se sitoo parametrina saadun `InitCombat`in attribuutit luokan ominaisuuksiin.

`public void actionPerformed(ActionEvent e)`-metodi saa parametrikseen `ActionEvent`in, ja kutsuu `InitCombat`in metodeja toteuttamaan `ActionEvent`in ilmentämän esineiden toiminnan kuvauksen.

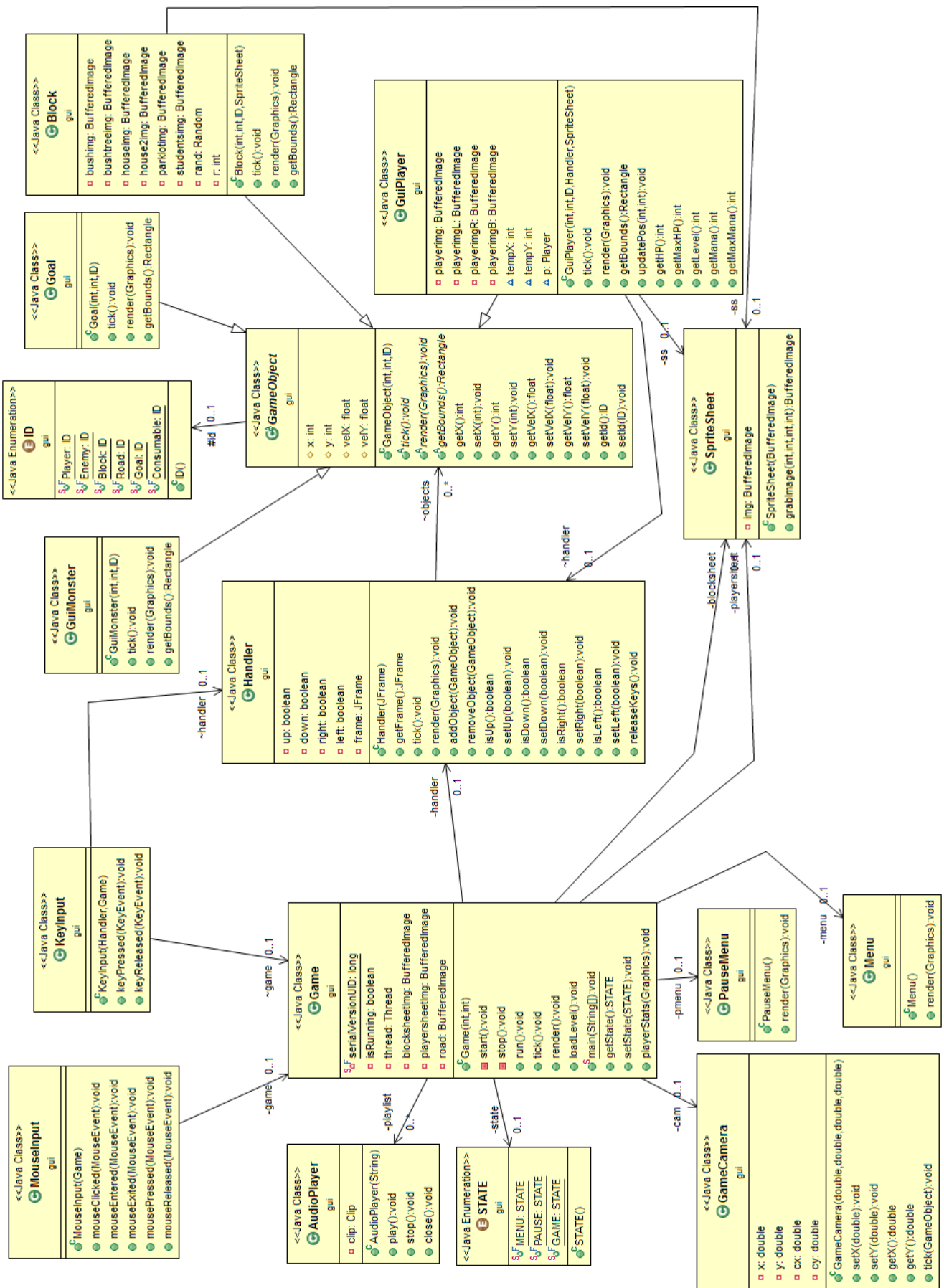
3.3.14 `CombatOutputStream`-luokka

Tämän on luokan `OutputStream` aliluokka, ja sen tarkoituksena on ohjata tuloste taisteluikkunan tekstilaatikkoon. Luokalla on seuraava attribuutti:

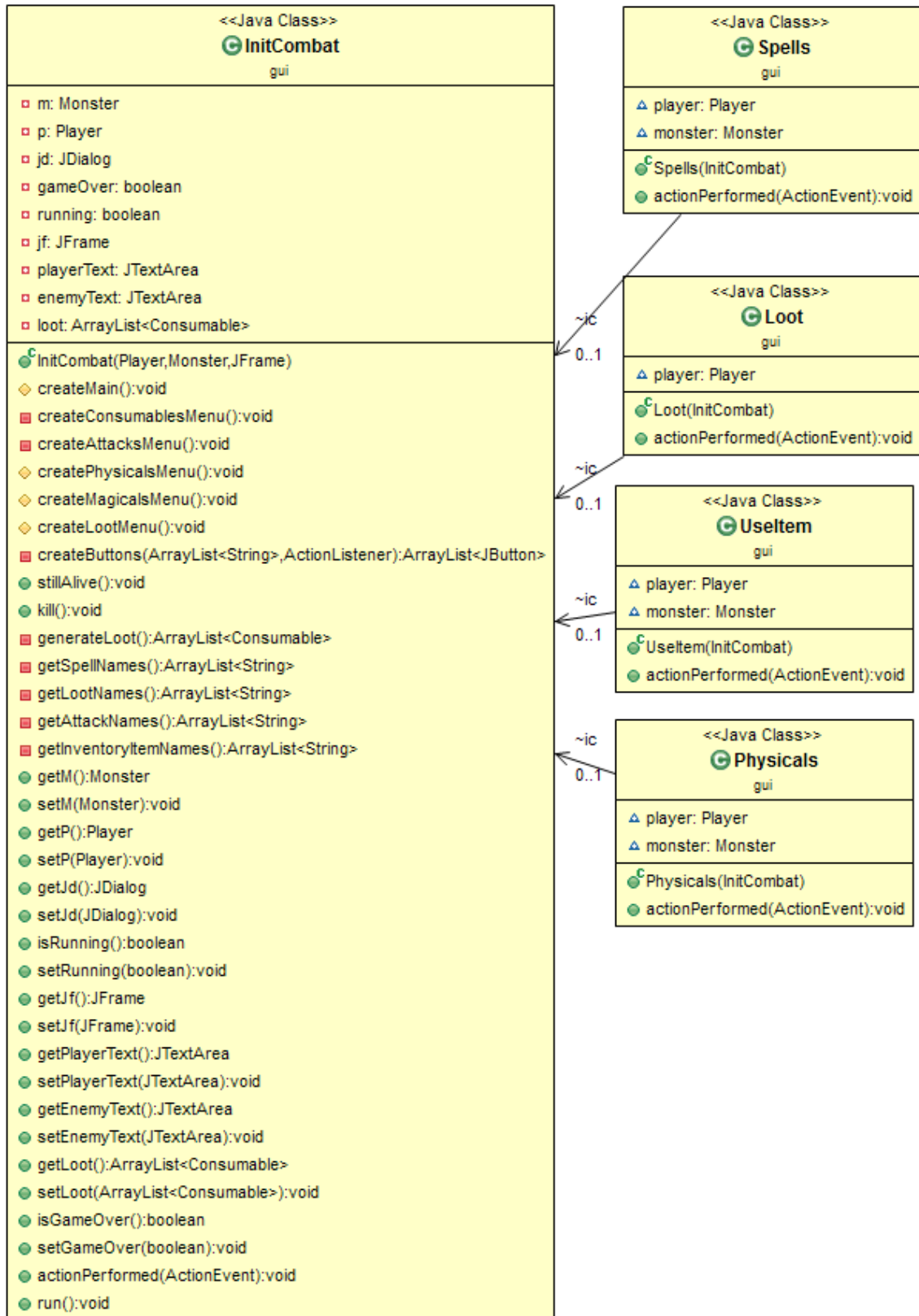
`JTextArea jsp` Tekstialue johon tuloste ohjataan

Luokan konstruktori on muotoa `public CombatOutputStream(JTextArea scrollpane)`, joka asettaa scrollpanen tekstialueeksi johon tuloste halutaan ohjata.

`public void write(int i)`-metodi ylikuormittaa ylliluokan vastaavanimisen metodin, ja se lisää syötteenä saadun parametrin `char`-tyyppisenä luokan tekstialueeseen.



Kuva 3: Luokkakaavio gui-paketin keskeisistä elementeistä



Kuva 4: Luokkakaavio InitCombat-luokasta

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

4.1 combat-paketti

Koska graafinen käyttöliittymän toteutus ei alkanut kuin vasta myöhemmin kehityksen aikana, päädyimme testaamaan kirjoittamiamme combat-paketin luokkia ja niiden metodeja seuraavilla testluokilla:

4.1.1 CombatEngine- ja CombatTest-luokat

Nämä kaksi luokkaa toimivat combat-paketin luokkien ja metodien testaukseen. CombatTest alustaa ilmentymät Player- ja Monster-olioista, ja lisää Player-luokalle Attack-luokasta hyökkäyksiä. Tämän jälkeen CombatTest kutsuu CombatEngine-luokkaa antamalla sille parametreinä luodut oliot.

CombatEngine-luokka koostuu pääsilmukasta, joka pitää kirjaa parametreina saamiensa Playerin ja Monsterin elämä- ja taikapisteistä. Luokan päämetodi lukee jokaisella suorituskierroksella käyttäjältään saaman syötteen, ja validin syötteen saatuaan tulostaa ruudulle uuden listan toiminnoista joita käyttäjä voi tehdä. Nämä kaikki listat luetaan Player-luokan listoista, ja niiden pohjalta koodi generoi tekstipohjaisen listauksen joka tulostetaan näytölle. Kun annetaan valinta, joka johtaa hyökkäyksen tekemiseen, koodi kutsuu Creature-luokan metodia tekemään vahinkoa Monster- ja Player-luokkien olioihin. Vihollisen hyökkäys päätetään kutsumalla Monster-luokan selectAttack-metodia. Silmukan suoritus päättyy kunnes joko vihollisen tai pelaajan elämäpisteet ovat ≤ 0 .

Kun pääsilmukan suoritus lopetetaan, tarkistetaan kumman elämäpisteet ovat loppuneet. Jos pelaaja hävisi (eli elämpisteet loppuivat), tulostetaan yksinkertainen viesti ruudulle. Jos puolestaan pelaaja voitti, annetaan pelaajalle vihollisen elämäpisteiden verran kokemuspisteitä ja tarkistetaan riittävätkö ne uuteen kokemustasoon kutsumalla CheckLevelUp-metodia. Tämän jälkeen luodaan lista mahdollisista palkinnoksi saatavista esineistä kutsumalla generateLoot-metodia, joka puolestaan poimii satunnaisia elementtejä ItemGenerator-luokan palauttamasta esinelistauksesta.

CombatEnginen toiminnalla kokeiltiin ja testattiin myöhemmin toteutettavan graafisen käyttöliittymän taistelunäkymän logiikkaa ja toiminnallisuutta, ja peruslogiikka kummankin takana on samankaltaista.

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