Objektorienteeritud programmeerimine

Rühmatöö nr 1 – tekstipõhine rollimäng

Robin Pau, Marilin Kuusk

1. Projekti üldkirjeldus ja eesmärk

Projekti eesmärk oli luua tekstipõhine rollimäng. Kasutaja seikleb koobastes, kohtub seal erinevate tegelastega ning peab ellu jääma. Koopa enda ülesehitust on lihtne muuta, kuna see on salvestatud 2D massiivi, mille põhjal meetod genereerib mängija jaoks koopa.

Mängu alustades peab kasutaja sisestama oma karakteri nime. Koopas on mitu taset ning igas tasemes saab mängija valida 1-4 toa vahel, kuhu minna. Valides 0, on võimalik mäng enneaegselt lõpetada. Igas toas ootab teda mingisugune tegelane (koll, võlur, abistaja, lõks, boss), kellega mängija interakteerub ning sõltuvalt toatüübist, on interaktsioon kas automaatne (koll, abistaja, lõks, boss) või saab mängija midagi ellujäämiseks ise ära teha (võlur). Kollide tugevus on muidu juhuslik, v.a boss, kes on samuti loodud klassi Koll põhjal, ent kes on palju tugevam tavakollist.

2. Klassid

Main

Selles klassis luuakse mängijale talle määratud kaardi põhjal koopad (meetod koopaLoomine) ning jooksutatakse põhiklass. Põhiklassis küsitakse mängija nime, väljastatakse tema algandmed ning küsitakse, millisesse ruumi ta soovib edasi minna. Põhiklassis oleva tsükli abil kutsutakse välja vastava toa tegevus ning kontrollitakse, kas mängija on veel elus.

Koll

Isendi loomisel konstruktori abil genereeritakse kollile suvaline kirjeldus antud listist. Klass sisaldab endas meetodid ruumiTegevus, kus viiakse läbi võitlus.

Võitluse meetodi üldidee: võitluse iga tsükli alguses genereeritakse nii kollile kui ka mängijale tugevus ja kaitse vahemikust 1...nendeMaxKaitse/Tugevus. Ehk kui mängija tugevus on näiteks 10, siis tema löögitugevus võib olla suvaline väärtus 1...10. Sellest lahutatakse Kollile samamoodi suvaliselt genereeritud kaitse (jällegi lubatud vahemikust) ning see vahe määrab ära, kui palju koll päriselt saab kannatada. Kui mängija ründetugevus on 7, aga kolli kaitse on 6, siis tegelikult teeb mängija viga ainult 1 eluühikut. See põhimõte käib mõlematpidi - nii kollile kui ka mängijale. Kolli maxTugevus ja maxKaitse genereeritakse suvaliselt koopa loomisel. Võitlus kestab alati senikaua, kuni koll või mängija surma saab. Klassis on igaks juhuks ka Getterid ja Setterid.

Mängija

Mängija klass sisaldab endas ainult konstruktorid ja erinevaid Gettereid ja Settereid. Üheks isendiväljaks on kaart, mille põhjal luuakse mängijale koobas.

ToaTegevus

ToaTegevus on Liides. See sisaldab kahte meetodit:

public boolean ruumiTegevus(Mängija mängija);

public void getKirjeldus();

Võlur

Võlurile genereeritakse samuti suvaline kirjeldus, kuid lisaks ka suvaline mõistatus, mida ta mängijalt küsib.

Meetod ruumiTegevus hõlmab endas just sellesama küsimuse küsimist ning õigsuse kontrollimist. Mängija saab vastata 3 korda, suur- ja väiketähti erinevateks ei loeta, küll aga on ainult üks õige vastus. Kui mängija vastab õigesti esimese korraga, saab ta 1 lisaelu, kui aga 3 korda valesti, kaotab 2 elu.

Abistaja

Abistaja klass on loodud, et mängijale lisatuge koopa läbikäimisel pakkuda. Abistaja ruum on olemas peaaegu igas tasemes (v.a Boss ja Lõpp). Abivahendeid on neli erinevat: mõõk, kilp, kaart ja ravijook

Mõõk lisab mängijale 2 tugevust, kilp 2 kaitset, ravijook 1 elu ning kaart väljastab mängijale, mis ruumid järgmises tasemes on, et ta saaks teha teadliku valiku. Abivahendi generatsioon on juhuslik.

Lõks

Lõksule genereeritakse suvaline kirjeldus. Meetod ruumiTegevus kontrollib, kas koopaLoomisel genereeritud suvaline vigatamistugevus vahemikust 0-2, tekitab mängijale ka päriselt kahju ning kas mängija jäi sellest ellu. Mängija enda kaitsetugevus siinkohal teda ei aita.

3. Projekti protsessi kirjeldus ning iga rühmaliikme panus

Esmalt mõeldi erinevalt, kuidas mingeid tegevusi või üldstruktuuri realiseerida ning mis tegevused või karakterid mängus üldse olemas peaksid olema. Seejärel saadi mitu korda kokku ning kogu programmeerimine tehti koos ühes või teises arvutis, koos samal ajal arutades. Leidsime, et selliselt töötamine on meie jaoks efektiivsem, kuna nii on mõlemal osapoolel hea ettekujutus sellest, kaugel protsess on, mida mingi meetod teeb ning üleüldine koodipilt oleks ühtsem. Kui mängu funktsionaalsus oli olemas, oli viimane samm mäng värvikamaks muuta läbi erinevate kirjelduste ja *„flavor text“*-de lisamisega. Üldine rühmaliikmete panus oligi võrdne, kuna kõik etapid tehti koos.

Kuna kaks pead on ikka kaks pead, siis sai projekt üpris hõlpsalt valmis ning üldajakulu oli umbes 10 tundi projekti peale, mis hõlmab endas ideede mõtlemist ja arutamist, programmeerimist ennast ning kirjelduste väljamõtlemist ning programmi lisamist.

4. Tegemise mured

Peamiselt oli kahju sellest, et sellist rollimängu on võimalik lõpmatuseni huvitavamaks ja komplekssemaks teha, kuid antud projekti raames tuli kuskile piir tõmmata. Paljud tegevused tuli teha n-ö automaatseks, kuna ei tulnud häid ideid, kuidas mängijat kaasata ning ilmselt oleks see ka juba liiga keeruliseks läinud. Samuti oli protsess natuke kaootiline, kuna meil polnud kindlat plaani, mida järgida, vaid loome toimus pigem nii, kuidas ideed järjest tulid. Testima saime hakata siis, kui peameetod lõpuks valmis sai. Testimiseks kasutasime korduvat läbimängimist, kui mingi koht oli kahtlane, siis kasutasime print meetodit, et kontrollida, kus kohas täpsemalt segadus tekib.

5. Hinnang lõpptulemusele

Arvame, et üldpildis tuli meie projekt välja huvitav ja funktsionaalne, milline peabki üks projekt olema. Oskustest puudu ei jäänud ning suutsime kasutada kursuse raames õpitud teadmisi. Arendada võiks projektijuhtimist – kuidas panna paika mingi kindel tegevuskava selle asemel, et lihtsalt tegema hakata.