Tallinna Ülikool

Informaatika Instituut

3D mängude loomine XNA keskkonnas. Õppematerjal

Bakalaureusetöö

Autor: Tambet Paljasma

Juhendaja: Jaagup Kippar

Autor: ....."…….”2011

Juhendaja: ....."…….”2011

Instituudi direktor: ....."…….”2011

Tallinn 2011

Autorideklaratsioon

Kinnitan, et käesolev lõputöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud.

....

(kuupäev) (allkiri)

Sisukord

Sissejuhatus

3D mängud on muutumas järjest populaarsemaks. Ka Eestis leidub palju huvilisi, kes sooviksid mänge ise luua. Mängude loomisel tuleb abiks Microsoft XNA. Kahjuks aga ei leidu Microsoft XNA kohta kuigi palju eestikeelset materjali.

Käesoleva bakalaureusetöö eesmärkideks on ülevaate andmine Microsoft XNA kohta, Microsoft XNA 3D algajatele mänguloojatele mõeldud eestikeelse õppematerjali koostamine ning selle kasutatavuse testimine.

Bakalaureusetöö teema valik tulenes reaalsest vajadusest eestikeelse XNA õppematerjali järele, mida saaks kasutada Microsoft Eneta Veebistuudiumi koolitusprogrammis. Teema valikut mõjutas ka autori isiklik huvi Microsoft XNA vastu, mis väljendus ka 2010. aastal loodud seminaritöös „XNA Game Studio õppematerjal“. Nimetatud töö keskendus 2D mängu loomise juhendile.

Bakalaureusetöö esimestes peatükkides antakse ülevaade Microsoft XNA-st ning selle kõige uudsemast versioonist Game Studio 4.0 ning tehakse intervjuu Eesti Infotehnoloogia Kolledži Microsofti õppejõuga. Edasi keskendutakse õppematerjali kirjeldamisele, koostamise põhimõtetele ja võrdlemisele Microsofti poolt loodud „Going Beyond: XNA Game Studio in 3D“ juhendiga.

Nii ajaliselt kui ka mahuliselt küllaltki suure osa tööst moodustab õppematerjali testimise protsess, mis viidi läbi kolmes etapis. Testimist kirjeldatakse lähemalt töö kuuendas peatükis.

Koostatud XNA 3D õppematerjal on esitatud käesoleva bakalaureusetöö lisana.

1. Intervjuu Heiki Tähisega

Eestikeelse XNA õppematerjali vastu on huvi tundnud Microsoft Eneta Veebistuudiumi koolitusprogramm.

Saamaks kinnitust Microsoft XNA kasulikkuse ja selle kasutatavuse kohta Eestis, tegi autor intervjuu Heiki Tähisega.

Heiki Tähis on Eesti Infotehnoloogia Kolledži Microsofti õppejõud, kes kuulub ka Eneta Veebistuudiumi lektorite hulka. Muuhulgas on talle omistatud Microsoft Certified Technology Specialist, Microsoft Certified Professional Developer, Microsoft Certified Trainer, Microsoft Most Valuable Professional tiitlid.

XNA tööriistu on intervjueeritav kasutanud õppetöös "Programmeerimine C# keeles" aine läbiviimisel ning näeb nende peamise kasutegurina just head platvormi objektorienteeritud lähenemise õpetamisel ja mõistmisel.

Heiki Tähis ütles: “XNA raamistiku piirid struktuurprogrammeerimisele iseloomulike töövahendite kasutamisel selguvad üsna kiiresti ning vähegi suuremahulisemate mängude loomisel tuleb hakata objektorienteeritud lähenemist kasutama. Omamoodi tuleb programmeerimise õpetamisel kasuks ka see, et XNAs pole võimalik kasutada .NET raamistikule muidu nii omast „võta ja lohista“ töövõtteid”.

Intervjueeritav lisas veel et, loomulikult ei ole vähetähtis ka asjaolu, et hilises teismeliseeas on mängude loomise võimalus noortele atraktiivne ja nii muutub programmeerimise õppimine lõbusaks väljakutseks.

Eestikeelse õppematerjali vajalikkuse kohta ütles Heiki Tähis järgmist: “Eestikeelse õppematerjali olemasolu on oluline just iseõppijatele, kelle keeleoskus ei võimalda võõrkeelsete allikatega vabalt töötada. Samas tuleb siinjuures mainida, et ka tavatudengitele on emakeelne materjal abiks, sest õppida tuleb nii kui nii ühte võõrkeelt (.NET raamistiku puhul C# või Visual Basic), kui seda teha veel võõrkeelsete materjalidega, siis seda keerulisem see on”.

Lisaks leiab ta veel et, eestikeelne lihtsakoeline XNA õppematerjal on kindlasti kasulik ka noorematele huvilistele. Kui täna põhikoolis õppijad saavad kasutada emakeelset õppematerjali ja teha mänge, mis töötavad tema arvutis, Xboxis ja telefonis, siis ta ka seda teeb. Tulemuseks on loodetavasti see, et gümnaasiumilõpetajana valib ta IT erialal kõrgkooliõpingud

2. Microsoft XNA

Microsoft XNA on Microsofti poolt loodud vajalike virtuaalsete tööriistade kogum, mis on mõeldud mängude loomiseks ja arendamiseks, püüdes mängu loojat säästa liigsest koodi kirjutamisest.

Lühend XNA tähendas algselt "*Xbox New Architecture*" hiljem muudeti lühendi tähendust humoorikalt: "*XNA is Not an Acronym*", kuna Microsofti „maailm“ on täis lühendeid.

Micrsoft XNA võimaldab esimest korda maailmas mitteprofessionaalsel mänguarendajal luua virtuaalseid mänge, mida saab mängida kas üksi või mitmekesi. Mitmekesi on võimalik mänge mängida nii ühe arvuti taga kui ka kasutada arvutite vahelist võrku. Suurim edusamm XNA puhul oli võimaldada tavakasutajal luua mänge Xbox 360 mängukonsoolile. Microsoft on suutnud luua väga aktiivse mänguarendajate kogukonna ning teinud võimalikuks mängude loomise akadeemilisel tasemel. (Lobão jt 2009: 16).

2.1 XNA raamistik

XNA raamistik koos esimese Game Studio versiooniga sai avalikkusele kättesaadavaks aastal 2004. XNA raamistikku kasutatakse erinevates Game Studio tarkvara versioonides. Raamistik võimaldab mänge luua Xbox 360-le ja Windowsi operatsioonisüsteemil töötavatele arvutitele, kasutades Microsofti poolt toodetud programmeerimiskeelt C#. XNA raamistik on peamiselt mõeldud aitamaks õpilastel ja programmeerimishuvilistel arvutimängude loojatel lihtsustada koodi kirjutamist. XNA raamistik on .NET „tööriistade“ kogum, mida on vaja mängu loomiseks.

2.1.1 Peamised eesmärgid

XNA Game Studiot arendaval meeskonnal oli kaks peamist eesmärki:

- **Võimaldada mängu kasutamist erinevatel platvormidel**.

Üks peamiseid eesmärke, mille XNA Game Studio meeskond endale seadis oli see, et kasutajal oleks võimalus luua mänge nii Xbox 360-le kui arvutile kasutades sama koodi. Microsofti eesmärk oli muuta programmeerijale, kes loob arvutimängu Windowsile, sama mängu Xbox 360 versiooni loomise võimalikult lihtsaks. Lisaks oli oluline võimaldada 95% ulatuses sama koodi kasutamist erinevatel platvormidel. XNA kasutamisel erinevatel platvormidel võib esineda juhtumeid, kus kõik ühel platvormil kasutatav ei sobi teisele nagu näiteks arvutimängu juures hiire kasutamine ei sobi Xboxile.

- **Lihtsustada mängu loomist.**

Mängude loomine on keerukas tegevus. Lisaks õpilastele ja hobikorras mänguarendajatele, kellele XNA raamistik on peamiselt mõeldud, tekib takistusi ka professionaalsel mänguloojal. Tihti on vaja ekraanile mõne objekti tekitamiseks väga palju koodi kirjutada. See on aeganõudev protsess, sest arvestada tuleb ka kergesti tekkivate vigade avastamiseks ja parandamiseks kuluva ajaga. Seepärast oligi XNA Game Studio meeskonna eesmärgiks muuta mängude loomine palju kergemaks ja kiiremaks kui varem. Nüüdseks on palju kasutaja eest eelnevalt ära tehtud. XNA arendajate eesmärgiks oli võimaldada kasutajal hakata mänguks vajalikku koodi kirjutama kohe pärast uue projekti loomist. See on teostatav, kuna kasutaja ei pea enam muretsema näiteks akna loomise pärast, kus hiljem mängu näha saab, samuti graafika adapterite või muude kuvamisel vajaminevate osade pärast. Kasutaja ei pea looma *Direct3D 9* seadet, mida on vaja 3D objektide näitamiseks, või muretsema viimase töökorras hoidmise eest, kui mängu ekraani suurust muudetakse. XNA raamistik hoolitseb kõigi nende asjade eest ise.

Iga mängu loomise juures on väga oluliseks aspektiks vajaliku sisu lisamine, ehk siis kõik objektid, kujundid, pildid ja muu selline, mis on mängu lahutamatu osa. Sisu mängule lisamine on küllaltki keerukas ja komplitseeritud protsess, kuid XNA raamistikus aitab *Content Pipeline* kõik selle väga lihtsaks teha. *Content Pipeline* hoolitseb sisu lisamise, kompileerimise ja selle mängu laadimise eest.

XNA Game Studio meeskond lisas raamistikku ka täiesti valmis ja töötava mängu, mida saab väga hästi kasutada uue mängu loomisel alusmaterjalina. Lisatud mängu on võimalik kasutada, kui uue projekti loomisel valida *Starting Kit*. Selle mängu peaeesmärgiks on anda algajatele mänguloojatele võimalus näha valmis mängu koodi ning seda muutes luua võibolla oma esimene mäng. XNA Game Studio meeskonna poolt loodud näidismänguga on kaasa ka loomulikult dokumentatsioon ja mitmed mängu muutmise juhised.

2.1.2 XNA kihid

Kui rääkida XNA raamistiku ülesehitusest, siis on seda kõige lihtsam seletada nelja peamise kihi abil.

- **Platvorm.**

Platvorm on kõige alumine kiht, mille peale hakatakse XNA raamistiku abil mängu looma. Platvorm koosneb osadest nagu näiteks *Direct3D 9*, *XACT*, *Xinput* ja *Xcontent*.

- **Tuumraamistik.**

Tuumraamistiku kihti võib lugeda esimeseks XNA raamistikku puudutavaks kihiks ehk tuumaks või südameks. See kiht koosneb peamistest funktsioonidest, mida järgnevad kihid kasutavad. Need funktsioonid hoolitsevad graafika, heli, matemaatiliste tehete, mälu ja muu sellise eest.

- **Pikendatud raamistik.**

Pikendatud raamistiku kiht on välja ehitatud kasutades eelnevat kihti. Kihi peamine ülesanne on teha koodi kirjutamine kasutaja jaoks võimalikult lihtsaks.

- **Mäng**

Viimane ehk mängu kiht koosneb kasutaja poolt kirjutatud koodist. Selles kihis on mängu sisu ja ka *Starting Kit* ehk tarkvara loojate poolt süsteemi lisatud näidismäng. (Cookiecups 2006).

2.2 Programmeerimiskeel C#

C# on programmeerimiskeel, mille tootja ja haldaja on Microsoft. Keelt kasutatakse ka XNA raamistiku abil mängude loomiseks. Programmeerimiskeel C# sai üldsusele kättesaadavaks

juunis 2000. C# näol on püütud teha võimalikult lihtne, uudne ja objektorienteeritud programmeerimiskeel. C# programmeerimiskeel on paljus sarnane C, C++ ja Java keeltega. C# programmeerimiskeel ei ole mõeldud ainult mängude kirjutamiseks, seda saab kasutada peaaegu ükskõik millise programmi kirjutamiseks. (Jones 2002: 7).

2.3 XNA Game Studio 4.0

XNA Game Studio 4.0 on programmeerimiskeskkond, mis lubab kasutada Microsoft Visual Studiot mängude loomiseks Xbox 360 mängukonsoolile, Windows Phone'ile või Windowsi kasutavatele arvutitele. Mängude loomine Windows Phone'ile on võimalik ainult XNA Game Studio 4.0 kasutades. XNA Game Studio sisaldab XNA raamistikku, mis baseerub Microsoft .NET 2.0 raamistikul. XNA Game Studio 4.0 ja XNA raamisik lubavad kasutajal sama koodi kirjutades luua mäng erinevatele platvormidele. Tänu sellisele võimalusele ei pea kasutaja kirjutama koodi eraldi iga seadme jaoks. Võimalik on ka muuta rakenduse olemust olenevalt seadmest, mille peal rakendust kasutatakse, kuna seadmetel on erinevad võimsused ja võimalused. Nimetatud asjaolu võimaldab mitmekesisemaid kasutusvõimalusi, kuna sama koodi saab kasutada erinevates seadmetes. XNA Game Studio 4.0 sai avalikkusele kättesaadavaks 16. septembril 2010 ja on seni uusim XNA versioon. XNA Game Studio 3.0-st räägitakse lähemalt autori poolt seminaritööks loodud õppematerjalis.

2.3.1 Nõuded XNA Game Studio 4.0 kasutamiseks

XNA Game Studio 4.0 saab kasutada järgnevate operatsioonisüsteemidega:

- Windows XP
- Windows Vista
- Windows 7

XNA Game Studio 4.0 töötab kõigi ülaltoodud operatsioonisüsteemide erinevatel versioonidel.

Märkused operatsioonisüsteemide kohta.

- Microsoft Windows Phone Developer Tool ehk tööriistade kogum, millega on võimalik luua rakendusi Windows Phone-le, ei tööta Windows XP peal.
- XNA Game Studio 4.0 vajab Windows Vista operatsioonisüsteemis töötamiseks vähemalt Service Pack 2.
- XNA Game Studio käivitamiseks on vaja arvuti administraatori luba välja arvatud Windows XP kasutajal.

XNA Game Studio 4.0 kasutamiseks peab arvuti sisaldama graafikakaarti, mis toetaks vähemalt DirectX 9.0c. Soovituslikud on graafikakaardid, mis toetaksid DirectX 10.

XNA Game Studio kasutamiseks peab arvutisse olema installeeritud vähemalt üks järgnevatest programmidest:

- Microsoft Visual Studio 2010 Express for Windows Phone
- Microsoft Visual Studio 2010 Express Edition
- Microsoft Visual Studio 2010 Standard Edition
- Microsoft Visual Studio 2010 Professional Edition

Vanemate Microsoft Visual Studio versioonide peal XNA Game Studio 4.0 ei tööta. Microsoft Viusal Studio 2010 Express for Windows Phone'i ja XNA Game Studio 4.0 kasutamiseks peab arvutisse olema installeeritud ka Microsoft Windows Phone Developer Tool. XNA Game Studio 4.0 vajab arvutis töötamiseks ka Microsoft .NET Framework 4.0 olemasolu. (Microsoft).

3. Õppematerjali võrdlev analüüs ja kirjeldus

Käesolevas bakalaureusetöös vaadeldav õppematerjal on loodud Microsoft Visual C# 2010 peal kasutamiseks. Lisaks on kasutajal vajalik installeerida oma arvutisse Microsoft Game Studio 4.0.

Õppematerjal on mõeldud peamiselt IT huvilistele, näiteks IT tudengitele, kellel on esmased teadmised programmeerimise kohta. Kindlasti võivad õppematerjali kasutada ka edasijõudnud programmeerijad, kes tahavad tutvuda Microsoft XNA Game Studio poolt pakutavate 3D võimalustega. Kuna õppematerjal on püütud teha võimalikult lihtne ning kõiki vajalikke tegevusi 3D mängu loomiseks on püütud kirjeldada kõigile arusaadavalt, usub autor, et materjali kasutamine on jõukohane ka neile, kellel varasem kokkupuude programmeerimisega praktiliselt puudub. Õppematerjali võib kasutada ka lihtsalt Microsoft XNA Game Studio tutvustamiseks ilma praktilise tegevuseta.

Teadaolevalt kavatsetakse ühe võimalusena Microsoft Eneta Veebistuudiumi koolitusprogrammi raames järgneval aastal pakkuda programmeerimisoskuste algõpet XNA kaudu. Tulenevalt sellest pidas autor vajalikuks lähtuda oma õppematerjali koostamisel mõnest Microsofti poolt loodud XNA-d tutvustavast juhendist. Õppematerjali loomisel on paljus võetud eeskujuks autori arvates hetke parim inglisekeelne materjal nimega „*Going Beyond : XNA Game Studio in 3D*“.

3.1 Õppematerjali analüüs

Microsofti „Going Beyond“ juhendi järgi valmiv mäng on mõeldud eelkõige Xbox 360 peal kasutamiseks. Käesoleva õppematerjali koostamisel on püütud vältida puudusi, mis autori arvates esinesid „*Going Beyond*“ juhendis. Puuduste all peab autor silmas mitte põhimõttelisi koodivigu, mis takistaksid mängu loomist, vaid puudujääke protsesside kirjeldamisel ja selgitamisel. „*Going Beyond*“ materjal võib olla algajale mänguprogrammeerijale raskesti mõistetav ja mõningate toimingute sisu võib jääda arusaamatuks. See väide leiab kinnitust ka „*Going Beyond*“ internetilehekülje

kasutajapoolsetes kommentaarides. Mitme kasutaja väitel on see juhend algajale programmeerijale liiga raske.

Autori poolt loodud õppematerjalis kirjeldatud tegevuste tulemuseks on lihtne 3D mäng, milles on võimalik ringi lennata kosmosesüstikuga, hävitada asteroide neid tulistades ning asteroidiga kokkupuutel hävitada süstik. Mäng ei saa õppematerjali järgides kindlasti lõplikult valmis vaid ootab kasutajapoolset täiendamist ja edasiarendamist. Kõige selle loomiseks on kasutatud elementaarseid C# ja XNA Game Studio osi, mida saab hiljem tarvitada ka teiste mängude loomisel.

Õppematerjal on püütud koostada tegevusi võimalikult lihtsalt ja detailselt lahti seletades, et kasutajal oleks mugav materjali jälgida ja mõista. Programmeerimise juures ei ole tähtis mitte ainult see, et kood oleks väga täpselt kirjutatud, vaid ka arusaamine, mida kirjutatud kood teeb, miks ta on just selline.

Paljud programmeerimiskeskkonnad kasutavad koodi osade eristamiseks erinevaid värve. Käesolevas töös on koodi kirjutamisel kasutatud samu värve, mida kasutab Microsoft XNA Game Studio.

Näide: `Vector3 kaameraPositsioon = new Vector3(0.0f, 50.0f, 5000.0f);`

Samade värvide kasutamisega on püütud eristada kirjutatavat koodi ning ära hoida pisivigade tekkimist koodi kirjutamisel. Värvide jälgimine on väga oluline. Kui õppematerjalis esineva koodi ja kasutaja poolt kirjutatud koodi värvid ei ühti, on kusagil tekkinud viga ning kasutajal tuleks kirjutatud kood õppematerjalis antud juhiseid täpselt järgides uuesti üle kontrollida. Kui kasutaja on juba pisut kogenenum koodikirjutaja, pakuvad alati abi ka Microsoft Visual Studio C# programmi poolt näidatavad veateated.

Autor on õppematerjali koostamisel jälginud, et materjalis kirjeldatud tegevused (peatükid) oleksid mängu loomise seisukohalt loogilises järjekorras. Õige järjekord on oluline parema arusaadavuse ja jälgitavuse huvides. „*Going Beyond*“ materjalis võivad mõningad mängu loomiseks tehtavate tegevuste järjekorrad tunduda ebaloogilised.

Näide: Enne 3D objekti liigutamise õpetamist räägitakse sellest, kuidas viia kaamera asendisse, kust võib objekti liikumist paremini jälgida. Vastupidises järjekorras toimides,

nagu see on kirjeldatud „*Going Beyond*“ juhendis, võib tekkida olukord, kus kaamera ebasoodsa asendi tõttu kaob objekt seda liigutades vaateväljast. See võib viia mängulooja asjatult segadusse.

Ka õppematerjalis sisalduva koodi kirjutamise järjekord on püütud hoida võimalikult lihtne. Täidetakse üks fail, meetod, klass korraga, mitte ei lisata vajalikke osi järjest, nagu teevad kogenud programmeerijad. See on vajalik hoidmaks ära võimalikke mittemõistmisi ja vältimaks vigu, mis võivad tekkida, kui kasutaja peab koodi kirjutama pidevalt erinevatesse failidesse, meetoditesse või klassidesse. „*Going Beyond*“ õppematerjalis sellele asjaolule tähelepanu ei pööratud.

Õppematerjalis sisalduvate kirjelduste juures on kasutatud erinevaid fonte ja kirjasuuruseid. Seda on tehtud eesmärgil muuta kasutaja jaoks lihtsamaks õpetuse jälgimine ning tähtsamate osade märkamine. Kõik koodis sisalduvad sõnad on ka tavatekstis suurendatud ning font muudetud. Lisaks kannavad kõik meetodid nime järgi tähist „()“, mille abil on kasutajal kergem kindlaks teha, et räägitakse just meetodist, mitte millestki muust. „*Going Beyond*“ õppematerjalis sellist eristamist ei kasutatud.

Õppematerjalis olev tekst, mis kirjeldab eelnevalt kirjutatud koodi või on muul põhjusel oluline, on ümbritsetud kastiga. See on vajalik materjali parema jälgitavuse huvides ning kasutaja tähelepanu püüdmiseks.

Õppematerjal sisaldab palju pilte, mis teenib sama eesmärki - kasutajale materjali arusaadavamaks muutmine ja selle jälgimise lihtsustamine. Lisaks aitavad pildid võrrelda visuaalselt juhendis kirjeldatut kasutaja poolt loodud tulemustega.

Kokkuvõtlikult võib märkida, et võrreldes „*Going Beyond*“ materjaliga, on käesolev juhend märksa lihtsam ja kasutajasõbralikum, sest ta on koostatud silmas pidades eelkõige väheste kogemustega mänguprogrammeerijaid.

3.2 Õppematerjali kirjeldus

Õppematerjal hõlmab peamiseid Microsoft XNA Game Studio 4.0 abil 3D mängu programmeerimise meetodeid ning võtteid.

Juhendmaterjali kaheksa peatükki võib mõtteliselt jaotada kahte erinevasse ossa.

Esimene osa, mis haarab nelja esimest peatükki, käsitleb tegevusi, mis on vajalikud mängu loomiseks: uue projekti loomine ning esimese 3D objekti ekraanile tekitamine, 3D objekti tekitamine kasutades uut faili, vajalikkude muutuste tegemine mitme objekti ekraanile tekitamiseks ning kaamera asukoha ja vaatevälje kirjeldus.

Teine osa, viimased neli peatükki, annab kokkuvõtliku ja lihtsa ülevaate Microsoft XNA raamistiku võimalustest mängu täiustamiseks: 3D objekti liigutamine klaviatuuri nuppude abil, suuremas hulgas uute objektide ekraanile tekitamine, erinevate 3D objektide kokkupuute kontrollimine, 3D objektide automaatne liigutamine kasutades kas olemasolevat teist objekti või etteantud suuruseid.

1. 3D objekti tekitamine

Esimeses peatükis on kirjeldatud, uue projekt loomine ning olemasolevale projektile 3D objektide lisamine. Lisaks kirjeldab esimene peatükk ka 3D objekti ekraanile kuvamiseks vajalikke koodiridu.

2. 3D objekti tekitamine kasutades uut faili

Teises peatükis on kirjeldatud, kuidas toimida, et tekitada mängu uus objekt, kasutades selleks uut faili mitte olemasolevat. Suuremate mängude või projektide juures on alati kasulik hoida mängu või projekti osad eraldi failides. See teeb koodi lugemise palju lihtsamaks ning võimaldab kasutajal teha kiirelt ja lihtsalt vajalikke muudatusi, kuna üks fail ei sisalda enam nii palju koodiridu.

3. Vajalikud muutused mitme objekti kuvamiseks

Kolmandas peatükis on näidatud, mida on vaja teha, et efektiivselt laadida mängu rohkem kui üks 3D objekt. Nende õpetuste abil pääseb kasutaja liigsest koodi kirjutamisest. Mida vähem on koodi, seda lihtsam on tehtut jälgida.

4. Kaamera ja projektsioon

Neljas peatükk räägib lähemalt kaamerast ning selle seadistamisest. Kaamera vaateväli ehk ruumipilt, mida kasutaja näeb, on üks tähtsamaid mängu osi.

5. Laeva asukoha muutmine nuppude abil

Viiendas peatükis kirjeldatakse lähemalt seda, kuidas panna 3D objekt mängus nuppude abil soovitud suunas liikuma.

6. Teiste objektide tekitamine ekraanile

Kuuendas peatükis näidatakse, kuidas tekitada ekraanile suuremas koguses uusi objekte. Lisaks käsitleb kuues peatükk ka kõigi nende uute objektide haldamist, liigutamist, uuendamist ja muud vajalikku.

7. Asteroidi ja laeva kokkupuute kontrollimine

Seitsmes peatükk hõlmab tegevusi, mida on vaja kahe erineva objekti kokkupuute kontrollimiseks ning kirjeldab, kuidas panna mäng kokkupuutele reageerima.

8. Laevale kuulide lisamine

Kaheksandas peatükis näidatakse, kuidas uutele lisatud objektidele anda liikumissuund kasutades teist objekti.

4. Õpiteed

Käesolev õppematerjal on mõeldud peamiselt programmeerimishuvilistele, kes omavad algteadmisi koodi kirjutamise kohta, nagu näiteks:

- koodi astendamine, mis teeb kirjutatud koodi paremini jälgitavaks, lihtsustab vigade avastamist ja parandamist ning uue koodi lisamist olemasoleva sisse;
- koodi kirjutamise järjekord - koodirida, mille abil luuakse uus muutuja peab olema kirjutatud muutujat kasutavast reast ülespoole;
- arusaam klasside ja meetodite alguse ja lõpu kohta, ehk kus täpselt klass või meetod algab ning kus see lõppeb, et vältida vigu, mis võivad tuleneda koodi valesse kohta kirjutamisest.

Õppematerjalis kirjeldatu on soovi korral kindlasti võimalik läbi teha ka isikutel, kes ei ole kokku puutunud programmeerimisega, kuid soovivad proovida, kuidas toimub lihtsa 3D arvutimängu loomine. Nendel isikutel soovitab autor tutvuda enne tema poolt seminaritööks koostatud XNA 2D juhendiga, kus on mõningaid asju täpsemalt selgitatud kui XNA 3D materjalis.

4.1 Soovitused õppematerjali läbimiseks

Õppematerjali edukaks läbimiseks on soovitatav:

- Kirjutatud koodist aru saada. Autori arvates on koodist arusaamine isegi tähtsam, kui tegeliku koodi kirjutamine. Seetõttu soovitab autor õppematerjali läbimisega mitte enne edasi minna, kui on arusaadav, mida kirjutatud koodirida täpselt teeb. Kirjutatud koodist arusaamine võimaldab kasutajal oma mängus lihtsamini uuendusi ja muudatusi sisse viia.
- Aru saada, mida teevad erinevad meetodid. Mängu kood koosneb paljudest erinevatest meetoditest, millel kõigil on oma tähtis roll. Autor soovitab kasutajal enne koodi kirjutamist aru saada, mida teevad erinevad meetodid. See võimaldab kergemat koodi

mõistmist ning ka hilisemate muudatuste ja uuenduste tegemist. Meetodeid tutvustab autor ka oma varasemas seminaritöös.

- Jälgida õppematerjalis oleva koodi värve. Õppematerjalis oleva koodi värvid kattuvad värvidega, mis reaalselt esinevad mängu koodi kirjutamisel Microsoft XNA-d kasutades. Kui õppematerjalis toodud värv ja mängu looja poolt kirjutatud koodi värvid ei ühti, on viimane kusagil vea teinud ja autor soovitab uuesti läbi lugeda selgitavad laused koodi kohta.

- Enne koodi kirjutamist lugeda hoolikalt selgitavat juhendit. Õppematerjalis kirjeldatud koodi kirjutamist on võimalik läbi teha ka ainult kopeerimise ja asetamise teel, tuleb vaid hoolikalt jälgida, kus kirjeldatav koodirida peab täpselt asuma. Autor soovitab kindlasti läbi lugeda kõik koodi selgitavad laused, mis aitavad kasutajal aru saada kirjutatavast koodist ning hiljem teha muudatusi ja uuendusi.

4.2 Iseharjutamise võimalused.

Õppematerjal ei sisalda täiendavaid harjutusi, mida kasutajal oleks võimalik täita. Kuna õppematerjali peamiseks sihtgrupiks ei ole professionaalsed koodikirjutajad, võib harjutuste täitmine muutuda raskeks ja frustreerivaks. Küll aga sisaldab õppematerjal palju näiteid ja suurel hulgal koodi seletusi. Nende abil on võimalik katseliselt koodi muuta, et täpsemalt aru saada, mida konkreetne koodirida teeb. Autor soovitab õppematerjali kasutajatel katsetamise eesmärgil toodud näidete abil proovida koodi muuta. Samuti võiks koodi muuta kohtades, kus koodi toimimine on üheselt arusaadav, nagu näiteks objekti liikumiskiiruse muutmine.

5. Õppematerjali testimine.

Õppematerjali testimine on üks selle bakalaureusetöö olulisemaid osi. Õppematerjali testimine viidi läbi kolmes etapis. Kõiki etappe kirjeldatakse allpool lähemalt. Testimise üheks eesmärgiks oli võimalike vigade ja ebatäpsuste avastamine juhendis. Veel olulisemaks pidas autor testimisega välja selgitada, kas koostatud õppematerjal on sobiv eeldatud kasutajagrupile ning kas seda on võimelised kasutama ja sellest aru saama ka vähem kogenud kasutajad.

Esimeses testimise etapis oli õppematerjali kasutajaks isik, kellel puudus eelnev programmeerimisalane kogemus. Kasutaja oli vaid varem läbi teinud autori poolt seminaritööks koostatud õppematerjali XNA 2D kohta. Esimene etapp erines järgmistest eelkõige selle poolest, et töö autor viibis testimise juures ja jälgis materjali kasutaja toiminguid ja reaktsioone vahetult. Lisaks sellele, kas õppematerjal on arusaadav ja sobiks kasutamiseks ka eelneva programmeerimiskogemuseta isikutele, soovis autor veenduda, kas ja mil määral tuleb kasuks, kui XNA 3D õppematerjali kasutaja on eelnevalt tutvunud eelpool mainitud XNA 2D õppematerjaliga. Seega kas võib väita, et XNA 3D õppematerjal on loogiliseks ja harmooniliseks jätkuks varem koostatud XNA 2D materjalile.

Testimise teises etapis saadeti õppematerjal koos tagasisideks mõeldud küsimustega laiali kuuele IT tudengile. Testijad pidi materjalis kirjeldatu praktiliselt läbi tegema ning seejärel vastama kaheteistkümnele küsimusele. Enamik küsimusi eeldas põhjalikumat vastust kui ainult jah või ei. Saadud tagasiside abil püüdis autor kindlaks teha, kas õppematerjal on sobiv ja atraktiivne IT alal kogenenumatele kasutajatele ja kas midagi tuleks muuta või lisada.

Testimise kahe esimese etapi käigus kogutud informatsiooni põhjal töötas autor koostatud õppematerjali veelkord läbi, viis sisse mitmeid parandusi ja täpsustusi ning viimistles materjali visuaalset külge.

Testimise kolmandas etapis saadeti täiendatud ja parandatud õppematerjal veelkord mõnele teise etapi testijale uueks läbitöötamiseks, veendumaks, et autori püüdlused õppematerjali paremaks muuta on ennast õigustanud ja õppematerjalis tehtud muudatused on asjakohased ning selle lõplik variant kasutajatele hästi mõistetav.

5.1 Testimise esimene etapp

Koostatud õppematerjali esmane testimine viidi läbi kohe peale materjali esimese versiooni valmimist. Eesmärgiks oli välja selgitada juhendis esineda võivad vead ning probleemsed ja raskesti- või mitmetimõistetavad kohad, et neid parandada, paremini jälgitavaks ja mõistetavaks muuta. Testija ülesandeks oli iseseisvalt juhendmaterjali alusel kõik kirjeldatud toimingud läbi teha. Probleemi või küsimuse tekkimisel luges testija lahenduse leidmiseks veelkordselt juhendit. Kui ta vastust ei leidnud, küsis ta abi autorilt. Kommunikatsiooni hõlbustamiseks ja parema ülevaate saamiseks testimise käigus ilmneda võivatest kitsaskohtadest, viibis autor testimise juures ning jälgis pidevalt testija toiminguid ja reaktsioone.

Materjali testijaks valiti humanitaarteaduste tudeng, kes aasta tagasi aitas testida ka autori poolt seminaritööna koostatud õppematerjali XNA 2D kohta. Igasugune muu varasem kokkupuude programmeerimisega tal puudus. Enne töö algust olid testijal teatavad eelarvamused oma võimete kohta 3D mängu loomiseks, ta suhtus asjasse skeptiliselt, kuna teadis, et testitav juhend on koostatud siiski programmeerimishuvilisi isikuid silmas pidades.

Järgnevalt vaatleb töö õppematerjali peatükkide kaupa, kuidas kulges testimine ja milliseid tähelepanekuid võis teha.

1. 3D objekti tekitamine ekraanile

Testija leidis lihtsalt ja kiiresti üles veebileheküljed, kust on võimalik alla laadida Microsoft Visual C# 2010 Express-i ning Game Studio 4.0-i, kasutades Google-i otsingumootorit. Kuna tegu pole programmeerimise vallas kogenud isikuga, tekitas temas edukogemuse asjaolu, et enne juhendi lugemist mäletas ta ka vajaminevate programmide nimesid, kuigi neid vanemaid versioone, mida kasutati XNA 2D juhendi puhul.

Projekti loomisega sai testija hästi hakkama. Tal ei tekkinud mingeid küsimusi ega arusaamatusi ning ta liikus kiirelt edasi uute kataloogide tekitamise juurde. Ka uute kataloogide loomisel ei tekkinud mingeid tõrkeid. Kataloogid said loodud kiiresti ja õigetesse kohtadesse. Kataloogidesse uute failide lisamine kulges samuti probleemideta.

Koodi kirjutamise juurde jõudnud, pidas testija vajalikuks, et juhendisse lisataks selgitus, et Game1.cs fail, kus toimub peamine koodi lisamine ja muutmine, avaneb automaatselt programmi peaaknasse, kui uus projekt on loodud ning seda ei pea eraldi üles otsima ja avama. Kasutajal endal ei tekkinud küll probleeme Game1.cs faili leidmisega või mõistmisega, et see on juba avatud, kuna tema väitel mäletas ta seda eelnevast juhendist, kuid tundis muret inimeste pärast, kes sooviksid alustada XNA 3D juhendist mitte 2D omast.

Testija ei saanud juhendi järgi päris täpselt aru, kuhu peab esimese koodirea lisama ning soovitas seda osa paremini seletada. Ta leidis, et juhendis võiks olla selgitatud, kust üks klass või meetod algab ning kus lõppeb. Testija mäletas seda küll XNA 2D juhendist ja koodirida sai lisatud õigesse kohta, kuid esmakordne kasutaja ei pruugi seda osata.

Testijal tekkis küsimus, mida tähendab see, kui kirjutatud muutja nime alla tekib roheline laineline joon. Lisaks ei teadnud ta, mida näitab vasakule ekraani akna äärde tekkiv vertikaalne kollane joon.

Edasine esimeses peatükis käsitletav oli arusaadav ja probleemideta teostatav.

Esimese peatüki läbitöötamise järel, muutis testija katsetamise eesmärgil 3D objekti pöörlemiskiirust ning tundis suur rõõmu selle üle, et leidis iseseisvalt õige koodirea ja teadis, mida muuta vaja. Kasutajal tekkis ka kohe muid mõtteid, kuidas loodavat mängu muuta ja täiendada.

Arvestades testimisel ilmnenud asjaolusid tegi autor juhendi 1. peatükki järgmised täiendused:

- lisatud on selgitavad laused Game1.cs faili iseenesliku avanemise kohta;
- lisatud on selgitus klassi alguse ja lõpu kohta;
- täpsemalt on kirjeldatud, kuhu tuleb lisada esimesed koodiread;
- on ära selgitatud, mida tähendavad roheline laineline joon koodis oleva muutuja nime alla ja kollane vertikaalne joon programmi akna vasakul serval.

2. 3D objekti tekitamine kasutades uut faili

Teise peatüki läbis testija väga kiiresti ning ilma probleemide või küsimusteta. Kasutaja arvates ei vajanud ükski selle peatüki osa täiendavat selgitamist, kõik oli mõistetav.

3. Vajalikud muutused mitme objekti kuvamiseks

Kolmandas peatükis tekkis testijal probleem koodireaga, mis tuli lisada *Draw* meetodi alla. Probleem oli põhjustatud juhendis olevast ebatäpsusest. Juhendis ei olnud märgitud, et koodirida tuleb lisada eelnevate koodiridade alla, mitte üles. Autorile tundub koodiridade lisamine olemasolevate alla iseenesest mõistetavana, kuid algajatele mitte. Edasine kolmanda peatüki läbimine kulges ilma probleemide või küsimusteta.

Vältimaks tekkida võivaid arusaamatusi täiendas autor testimise tulemusena 3. peatükki selgitustega, et lisatav kood kirjutatakse olemasoleva koodirea alla.

4. Kaamera ja projektsioon

Neljanda peatüki alguses tekkis testijal arusaamatus, kas koodiread, millest räägitakse, peab lisama või on need juba olemas. Kasutaja soovitas, et võiks olla välja toodud ka see, kus need koodiread, millest parasjagu räägitakse, asuvad, millises failis ning millises klassis või meetodis.

Kõigest muust neljandas peatükis seletatust sai testija enda väitel hästi aru ning tal ei tekkinud mingisuguseid küsimusi. Testija mainis ka asjaolu, et talle meeldisid õppematerjalis olevad „katsetamiseks“ antud juhised selle kohta, kuidas teatavat koodirida muuta ja mis on selle tulemuseks. Ta leidis, et tänu koodi muutmise võimaluste kirjeldamisele on hästi mõistetav, mida konkreetne kood teeb. Ka autori arvates ei ole tähtis mitte ainult koodi kirjutamine, vaid väga oluline on aru saada kirjutatava koodi sisust. Testija sõnul muudavad sellised näited juhendi palju elavamaks ning kergemini mõistetavaks.

Juhendisse on testimise tulemusena lisatud selgitused, mis annavad kasutajale teada, et kirjeldatavad koodiread on juba olemas ning täpsustatud on nende paiknemist.

5. Laeva asukoha muutmine nuppude abil

Viienda peatüki läbimine kulges ilma probleemide ja takistusteta. Testijal õnnestus kõik vanad koodiread kustutada või muuta ja uued lisada ning laevaga ringilendamine pakkus suurt elevust. Testijal tekkis mitmeid mõtteid, mida oma mängu juures muuta ja ta sai analoogselt juhendis kirjeldatuga iseseisvalt koodi muutes soovitud toimingutega edukalt hakkama. See tekitas rahulolu tunde ja soovi mängu veelgi edasi arendada. Ta proovis näiteks oma 3D objekti liikumiskiiruse ja pööramise kiiruse muutmist.

6. Teiste objektide tekitamine ekraanile

Testija soovitas selgitada terminit *Garbage Collection*, kuna algajad programmeerijad ei pruugi teada selle tähendust ning seetõttu võib jääda arusaamatuks selle funktsioon ja eesmärk. Samadel kaalutlustel soovitas ta selgitada ka mõistet “massiiv”. Rohkem soovitusi tal kuuenda peatüki kohta ei olnud ja selle läbitöötamisel probleeme ei tekkinud. Testija oli rahul, kui nägi ekraanile uusi objekte lendamas. Siit tekkis tal koheselt ka loogiline mõte, et erinevate 3D objektide kokkupuutumist peaks vältima ning põrkumise puhul peaksid mõlemad ära kaduma. Seda aga käsitleb juba juhendi seitsmes alapunkt.

Testimise tulemusena on parema mõistetavuse huvides juhendisse lisatud selgitused kõigi segadust tekitanud terminite kohta.

7. Asteroidi ja laeva kokkupuute kontrollimine

Seitsmenda peatüki läbis testija kiirelt ning ilma probleemideta. Kõik vajalikud koodiread said lisatud või muudetud. Testijal tekkis küll üks väike probleem, mis oli tingitud tema enda veast, kuid selle sai ta kiirelt ja ilma kõrvalise abita parandatud, kui luges uuesti juhendis olevat seletust.

8. Laevale kuulide lisamine

Testijal tekkis probleem ühe lisatava *for* tsükliga. Analoogselt kolmanda peatüki toimingute juures kirjeldatuga lisas ta *for* tsükli koodireast ülespoole, mitte alla nagu oleks olnud õige. Tulemuseks oli see, et *for* tsükli sees olevat muutujat, millele anti väärtus, ei leitud, kuna see muutuja oli kirjeldatud koodis, mis asus allapool. Kasutaja soovitas ka sinna juhendi ossa lisada täpsem selgitus, kuhu vajatav *for* tsükkel lisada, kuna muidu võib teistel kasutajat esineda sama probleem, mis temal.

Edasine kaheksanda peatüki läbimine kulges kiirelt ja ilma probleemideta.

Testimise tulemusena on juhendisse arusaamatuse vältimiseks lisatud täiendavad selgitused, kuhu koodiread täpsemalt lisama peab.

Esimese etapi kokkuvõte

Juhendi esmane testime möödus autori arvates väga edukalt. Avastati mõned juhendis olnud vead ja lisati selgitavaid lauseid kohtadesse, mis testijal probleemseks kujunesid. Testimise tulemusena muutus juhend kindlasti märkimisväärselt konkreetsemaks ja ka algajale paremini mõistetavaks. Testija jäi juhendiga rahule ja oli rõõmus oma tehtud töö üle. Ta tõdes, et kõik kirjeldatu on ka algajale mõistetav ja teostatav.

Autor veendus, et enne XNA 3D õppematerjali juurde asumist on algajatel soovitav tutvuda XNA 2D materjaliga, mille autor koostas 2010. aastal seminaritööna. See annab teatud enesekindluse ja julguse ka 3D mängu loomist proovida. Mitmed toimingud, mis on vajalikud mõlema mängu loomise puhul, on 2D mängu juhendis pikemalt ja põhjalikumalt lahti kirjutatud. Kuna mõlemad juhendid on küllaltki sarnase struktuuriga ja koostatud samu põhimõtteid järgides, on 2D mängu loomise läbiteinud algajal lihtsam ja mugavam ka 3D mängu juurde asuda.

5.2 Testimise teine etapp

Teises testimise etapis saadeti esimese etapi käigus kontrollitud ja selle tulemusena parandatud õppematerjal koos kaheteistkümne tagasiside saamiseks koostatud küsimusega kuuele IT tudengile, kes omasid mõningast eelnevat programmeerimiskogemust. Testijad pidid õppematerjalis kirjeldatu läbi tegema ja vastama esitatud küsimustele.

Küsimused ja neile saadud vastused:

Kas olete varem kokku puutunud C# programmeerimiskeelega?

Neli kuuest kasutajast oli varem kokku puutunud C# programmeerimiskeelega.

Milliste teiste programmeerimiskeeltega olete kokku puutunud?

Kõik kasutajad olid varem kokku puutunud vähemalt kahe erineva programmeerimiskeelega. Erinevate kasutatud programmeerimiskeelte kogum oli küllaltki suur: C, C++, Java, Javascript, PHP, Python, Visual Basic, ActionScript, Ruby, Pascal.

Kas kõik juhendis kajastatu on arusaadav või peaks mingi osa olema paremini ja põhjalikumalt seletatud? Kui, siis milline osa?

Probleeme tekkis esimesest peatükist arusaamisega, kuhu täpselt esimesed koodiread kirjutada tuleb. Ühel testijal kuuest tekkis probleem ka *Content* kataloogi leidmisega.

Kõigile testijatele tekitas mõningast arusaamatust õppematerjali teine peatükk. Probleem oli tingitud autoripoolsest veast, mis oli jäänud peale esimest testimist parandamata. Selle küsimuse vastustest lähtuvalt:

- lisati juhendi esimesse peatükki täpsustavaid lauseid ja pilte;
- teises peatükis olnud viga parandati.

Kas teie arvates aitavad pildid muuta juhendit arusaadavamaks ning jälgitavamaks? Miks?

Kõik testijad pidasid pilte vajalikuks. Kasutajate meelest aitavad pildid ära hoida materjali muutumist liialt üksluiseks. Pildid illustreerivad ning võimaldavad paremat arusaamist visuaalsel teel. Mõlemad C# programmeerimiskeelt kasutanud mainisid, et neile tulid küll pildid kasuks, kuid arvasid, et võibolla ei peaks piltidega uuesti näitama läbitehtud asju, nagu näiteks failide loomise juures. Nad möönsid, et selline mõte võis tuleneda ka nende eelnevast kogemusest Microsoft Visual Studio tarkvaraga.

Kas teie arvates erineva fondi kasutamine ja suurema kirjaga tekstisisesed koodist võetud osad lihtsustavad juhendi jälgitavust ? Miks?

Kõik kuus testijat väitsid, et erinevate fontide ja kirjasuuruste kasutamine tuleb kasuks. Nende sõnul oleks materjali jälgimine väga raske, kui kõik tekst oleks ühte moodi kirjutatud. Üks testijatest juhtis tähelepanu asjaolule, et kuna ühesugust stiili ja samu värve on kasutatud nii koodis kui ka seletavas tekstis koodi kirjeldamisel, siis on esmapilgul mõnikord raske eristada koodi, mida peab kirjutama, koodist, mida materjalis seletatakse. Sellest tulenevalt tehti juhendisse parandus:

- kõik eelnevalt kirjutatud koodi seletused ümbritseti kastiga, et parandada materjali loetavust.

Kas juhendi läbitöötamine andis teile uusi mõtteid selle kohta, kuidas loodud mängu saaks edasi arendada? Milliseid?

Autori heameeleks tekkis kasutajatel väga palju ning erinevaid mõtteid mängu edasiarendamiseks. Testijatel tekkis mõtteid skoori tabeli lisamiseks, uute objektide tekitamiseks, mis mängijat tagasi tulistaks, erinevate relvade kasutamiseks kuni selleni välja, et mängu oleks omavahel võimalik mängida kasutades internetti.

Kas juhendis kirjeldatud teemad on teie arvates mängu loomise seisukohalt loogilises järjekorras või tuleks osade järjestust muuta?

Ükski testija ei arvanud, et osade järjekorda tuleks muuta. Nelja testija sõnul olevat õppematerjal üles ehitatud sarnaselt sellele, kuidas neile eelnevalt programmeerimist on õpetatud ning nad võisid juba eeldada, mida materjali alusel järgmiseks teha tuleb.

Kas koodi kirjutamise järjekord on loogiliselt üles ehitatud või tuleks teie arvates midagi muuta? Miks?

Ükski testijatest, ei arvanud, et koodi kirjutamise järjekorda peaks muutma. Autori mõningaseks üllatuseks ütles üks testija, et talle väga meeldis asjaolu, et korraga täideti ühte faili, klassi või meetodit, kuna teda mõnikord häirib koolitunnis õpetatava koodi kirjutamise järjekord.

Kas värvilised koodiread aitavad teid koodi kontrollimise juures? Kui jah, siis kuidas?

Kõik testijad pidasid värvilisi koodiridu väga vajalikuks. Testijad väitsid, et värvilise koodi kasutamine õppematerjalis aitab teha materjali paremini loetavaks, arusaadavaks ning lihtsustab vigade leidmist. Viiel kuuest testijast esines vähemalt ühel korral, nende enda kirjavea tõttu koodis viga, mille parandamise tegi värvide kasutamine palju lihtsamaks, kui see muidu oleks võinud olla.

Kas teie arvates on juhend piisav andmaks lihtsat ülevaadet XNA 3D kohta.

Kõik testijad pidasid materjali piisavaks lihtsa ülevaate andmiseks XNA 3D kohta. Leidus ka kaks testijat, kes oleks soovinud veel mängu õppematerjali järgi edasi arendada. Selle

bakalaureusetöö raames ei ole see kahjuks võimalik. Autor kavatseb aga materjali hiljem täiustada ning lisada uusi punkte mängu edasiarendamiseks.

Kas teie arvates on juhendis sisalduvad näited, mis puudutavad olemasoleva koodi muutmise võimalusi, kasulikud (näiteks: laeva pöörlemise telje muutmine või kaamera asukoha muutmine)? Miks?

Kõik kasutajad pidasid sellised näited väga kasulikuks. Sellised näited aitavat kasutajal paremini aru saada, mida koodiread teevad ning anda mõtteid mängu täiendamiseks. Paljud testijatest lisasid, et nad jätsid paljud näidete abil enda poolt tehtud muutused materjali lõpuni kasutusse.

Mida peate veel oluliseks lisada omapoolse kommentaari või soovitusena?

Testijad, kellel tekkis probleeme materjaliga, lisasid soovitusi, kuidas võiks nendel tekkinud probleemi parandada. Kõiki neid soovitusi on autor arvesse võtnud ning teinud õppematerjali nendest tulenevalt täiendusi. Testijate meelest oli materjal küllalt sisukas ja huvitav. Keegi ei väitnud, et materjal oleks muutunud ebahuvitavaks või igavaks. Testijad, kes oleksid tahtnud materjali abil mängu edasi arendada, soovitasid autoril tulevikus materjali koostamist jätkata.

Teise etapi kokkuvõte

Autor peab testimise teist etapi väga edukaks, kuna leiti vigu, mis esimeses etapis ei ilmnenud. Samuti saadi mitmeid mõtteid ja kasulikke soovitusi õpematerjali täiustamiseks. Kuna küsimuste vastused olid paljus väga sarnased ja üldised hinnangud positiivsed, julgeb autor loota, et õppematerjal on hästi koostatud.

5.3 Testimise kolmas etapp

Testimise kolmandas etapis saatis autor teise etapi põhjal parandatud ja täiustatud, õppematerjali veelkordselt mõnele teise etappi testijale. Eelkõige neile, kellel oli tekkinud mingeid probleeme ja küsimusi. Sellega püüdis autor kindlaks teha, kas tehtud parandused ja uuendused ennast ka õigustavad. Autor andis testijatele eelnevalt ülevaate tehtud parandustest

ja täiendustest koos selgitusega, mis eesmärki need teenivad. Kolmanda etapi testijatele esitati kaks küsimust:

Kas tehtud parandused rahuldavad teid ja lahendavad tekkinud probleemi?

Kõik testijad jäi rahule tehtud parandustega.

Kas teil on veel midagi omapoolselt lisada paranduste või kogu materjali kohta?

Ükski testija ei soovinud midagi muud omalt poolt lisada.

Kolmanda etapi kokkuvõte

Autori arvates läks kolmas testimise etapp väga hästi. Kõik testijad, kellel tekkis eelnevalt õppematerjaliga probleeme jäid autori poolt tehtud paranduste ning täiustustega rahule.

Kokkuvõte

Käesoleva bakalaureusetöö põhitulemuseks on Microsoft XNA 3D õppematerjal, mis on mõeldud IT huvilistele algajatele programmeerijatele. Õppematerjali sobivust ja kasutamiskõlblikkust on testitud kolmes etapis ning on jõutud autorit rahuldava tulemuseni. Õppematerjali on analüüsitud võrrelduna Microsofti poolt loodud *„Going Beyond: XNA Game Studio in 3D“* juhendiga. Analüüsi tulemusena on püütud selle töö raames loodud õppematerjalis ära hoida probleeme, mis võivad tekkida Microsofti poolt loodud õppematerjali läbides. Autor on andnud ka omapoolseid soovitusi õppematerjali kasutamiseks ning selgitusi eelduste kohta, mida võiks kasutamisel silmas pidada. Autor tegi intervjuu Eesti Infotehnoloogia Kolledži Microsofti õppejõu Heiki Tähisega, mille põhjal on võimalik veenduda eestikeelse XNA õppematerjali vajalikkuses ning selle kasutamise perspektiivikuses.

Bakalaureusetöös on antud ka eestikeelne ülevaade Microsoft XNA-st ning selle kõige uuemast versioonist Game Studio 4.0-st.

Summary

The aim of this Bachelor's thesis titled „Development of 3D Games in XNA Environment. Tutorial“ is to give a overview of Microsoft XNA and to create a Microsoft XNA 3D tutorial for beginner level programmers in Estonian and to test it. This tutorial is aimed to provide simple instructions of making a simple 3D game using Microsoft XNA. The need for such tutorial derives from Estonian Microsoft Eneta Web Studio training program that gives instructions of using various Microsoft products.

In the first chapter the author interviews a teacher of Microsoft systems of the Estonian Information Technology College in order to determine if such tutorial is needed and usable in class.

In the second chapter there is a overview of Microsoft XNA and requirement for run it on a PC.

The aim of third chapter is to give a closer view of the tutorial and comparison to „Going Beyond: XNA Game Studio in 3D” tutorial by Microsoft.

In the fourth chapter the author gives advices for more efficient use of the tutorial and touches upon recommended prior knowledge.

The last chapter describes the process and methods of testing the tutorial. The testing was conducted in three phases to determine possible errors and to ascertain the compatibility.

The tutorial is provided in the appendix of this bachelor's thesis. The tutorial contains of eight chapters.

Kasutatud kirjandus

Cookiecups. (2006). *What is the XNA Framework*. MSDN Blogs 25. august. http://blogs.msdn.com/b/xna/archive/2006/08/25/724607.aspx (16.04.2011)

Jones, Bradley L. (2002). *Sams teach yourself C# in 21 days.* Indianapolis: SAMS

Lobão, Alexandre Santos. Evangelista, Bruno Pereira. Leal de Farias, José Antonio. Grootjans, Riemer. (2009). *Beginning XNA 3.0 game programming: from novice to professional*. New York: Apress

Microsoft. *XNA Game Studio 4.0*. MSDN. http://msdn.microsoft.com/en-us/library/bb200104.aspx (16.04.2011)

Heiki Tähis: *Autori intervjuu*. Kirjalik. Tallinn 5. aprill 2011.

Konsulteeritud kirjandus

Microsoft. *Going Beyond: XNA Game Studio in 3D*. MSDN. http://msdn.microsoft.com/en-us/library/bb203897%28XNAGameStudio.31%29.aspx (16.04.2011)

Microsoft. *Content Catalog*. App Hub. http://create.msdn.com/en-US/education/catalog/?lc=1033&p=1 (16.04.2011)

Miles, Rob.(2008). *Getting Started Making Games with C# and Microsoft XNA Game Studio.* Microsoft Faculty Connection 30. aprill. https://www.facultyresourcecenter.com/curriculum/pfv.aspx?ID=7992 (16.04.2011)

L I S A

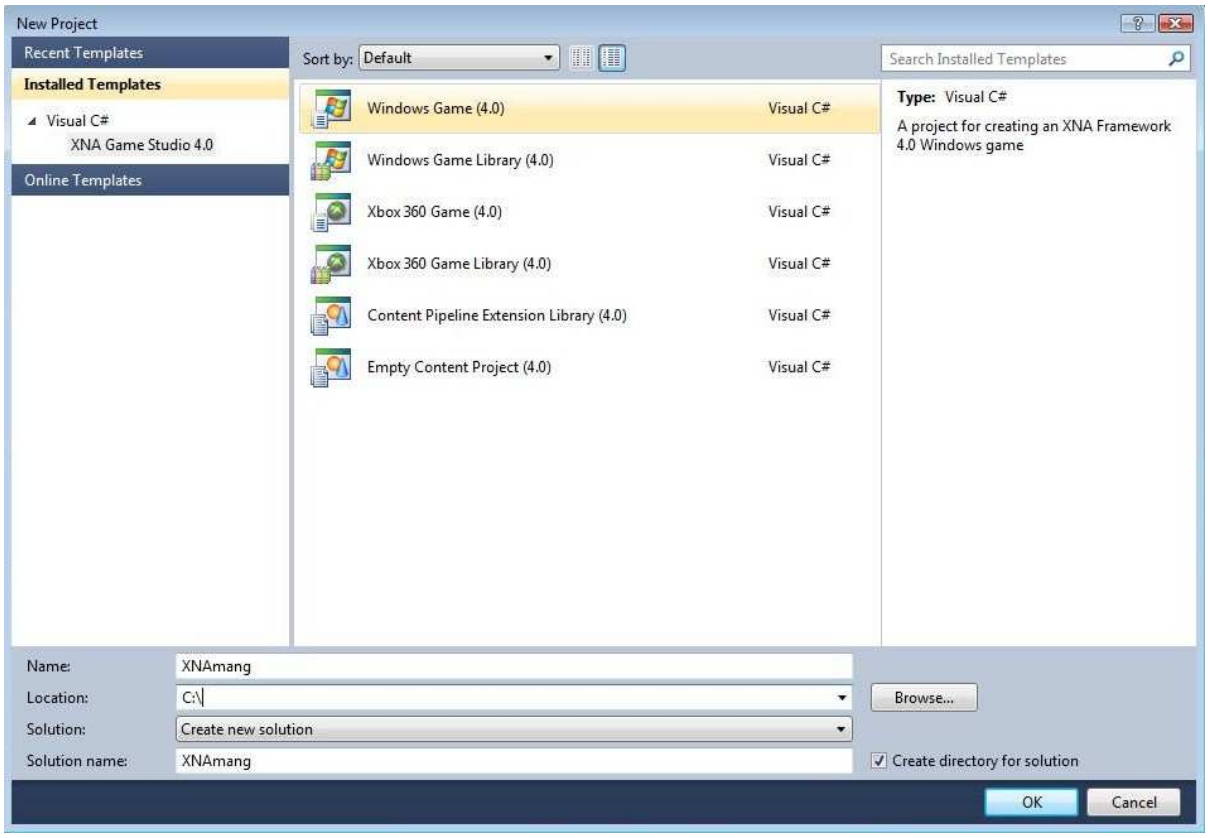
Lisa 1. Microsoft XNA 3D õppematerjal

1. 3D objekti tekitamine ekraanile

Õppematerjali kasutamise eelduseks on Microsoft Visual C# 2010 Expressi ja XNA Game Studio 4.0 olemasolu arvutis.

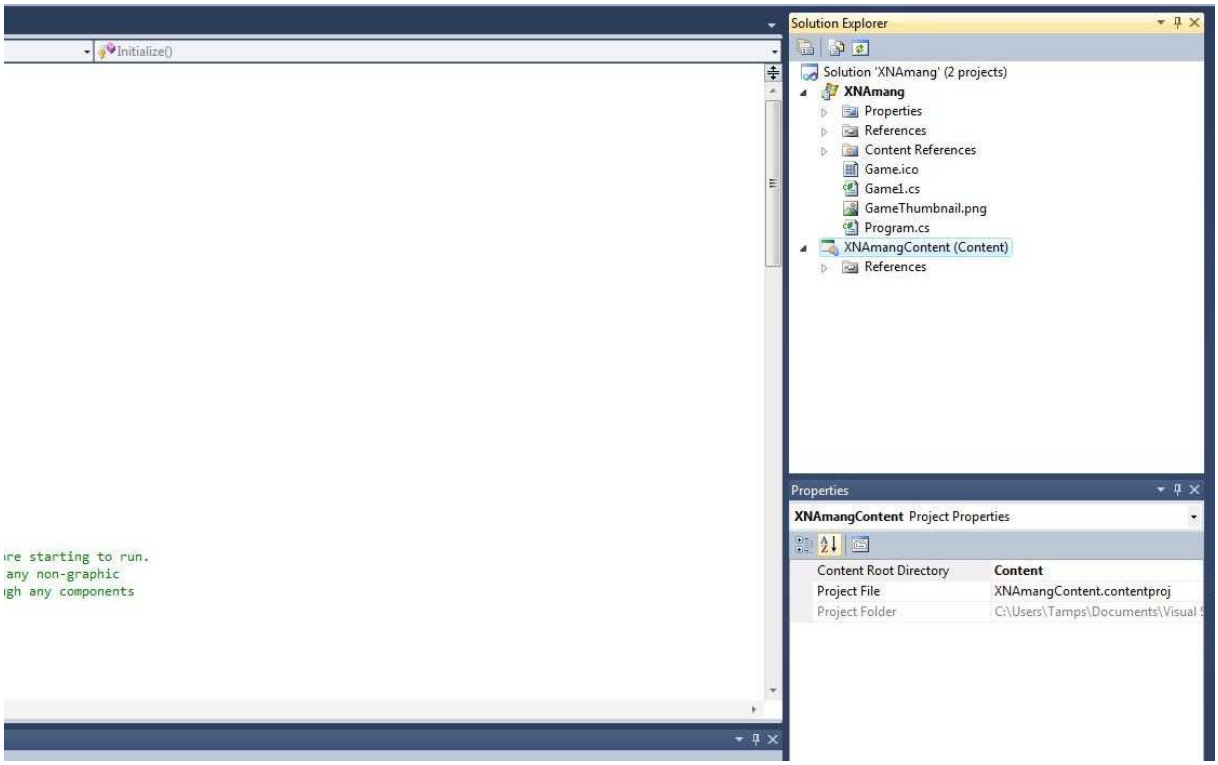
Käesolev peatükk hõlmab käske, mida on vaja 3D objekti ekraanile näitamiseks. Selle juhendi alusel loodavas mängus on objektiks kosmoselaev.

Esiteks on vaja teha uus projekt. Kõigepealt tuleb avada Microsoft Visual C# 2010 ning vajutada vasakul pool nupule New Project. Projekti loomisel peab silmas pidama, et valitakse XNA Game Studio 4.0 alt Windows Game(4.0). Kasulik on ka panna projektile nimi mille järgi on teda hiljem teistest kergem eristada.



Joonis 1. Projekti loomine

Kui projekt on loodud, tuleb sinna lisada 3D objekt, mida kasutama hakatakse. XNA Game Studio 4.0 Content kataloog, kuhu tuleb kõik visuaalset osa puudutavad failid lisada, on eraldi osa, mitte enam projekti alamkataloog.



Joonis 2. Content kataloog

Esmalt tuleb lisada kaks uut kataloogi: esimene nimega Models ja teine Textures. Kataloogi lisamiseks tuleb teha parem hiire klõps Content kataloogil ning valida Add ja New Folder.

Nüüd tuleb lisada 3D objekti fail ning tema tekstuuri fail. Faili lisamine käib ikka vana moodi ehk siis parem hiireklõps Content kataloogi tekitatud alamkataloogil Add ja edasi Existing Item. Models kataloogi tuleb lisada fail nimega p1_wedge.fbx ja Textures kataloogi fail nimega wedge_p1_diff_v1.tga.

Edasi tuleb hakata koodi kirjutama, mille abil liidetakse objekt mängule ja kuvatakse ekraanile. Lisaks tuleb määrata veel ka kaamera asukoht, mis edastab mängupilti.

Kui projekt on loodud avaneb kasutajal automaatselt Game1.cs fail.

Nüüd tuleb Game1.cs failis kõige ülemisse klassi, mille nimi on public class Game1 lisada kaks koodirida. Klasside ja meetodite alla jääv kood peab alati olema loogeliste sulgude vahel. Loogelised sulud näitavad kust klass või meetod algab ning kus ta lõppeb. Game1 klassi tuleb lisada järgnevad koodiread;

```
Model laev;
float aspectRatio;
```

Model laev annab lisatud 3D objektile nimi. Nime on vaja, et XNA raamistik saaks aru, et koodi read puudutavad just seda kindlat objekti.

Float näitab, et järgneva muutuja väärtused on komakohaga arvud.

aspectRatio-t kasutades tehakse kindlaks, kui suurelt näidatakse 3D objekti projektsiooni ekraanil. Koodirea alla peaks tekkima roheline loogeline joon, mis näitab, et seda muutujat ei ole veel kusagil kasutatud.

Iga uue koodirea kirjutamisel, mida programm ei ole veel „üle vaadanud“, tekib vasakule ekraani serva vertikaalne kollane joon.

Game1.cs falis asuvasse LoadContent() meetodi alla tuleb lisada järgnevad koodiread:

```
laev = Content.Load<Model>("Models//p1_wedge");
aspectRatio = graphics.GraphicsDevice.Viewport.AspectRatio;
```

Nüüd tuleb anda 3D objektile nurk, mis määrab ära, laeva ruumis paiknemise asendi. Selleks tuleb Game1 klassi lisada koodirida:

```
float laevNurk;
```

Peale seda tuleb lisada ka kaamera asukoht ruumis. Selleks on vaja Game1 klassi lisada koodirida:

```
Vector3 kaameraPositsioon = new Vector3(0.0f, 50.0f, 5000.0f);
```

Vector3 näitab, et tegu on 3D objektiga ja et kasutada on vaja X,Y,Z koordinaate.

Nüüd tuleb liikuda Game1.cs failis asuva Draw() meetodi juurde. Silmas peab pidama, et järgnevad koodiread tuleb lisada peale koodirida GraphicsDevice.Clear(Color.CornflowerBlue); , kuna see koodirida tekitab mängule tausta ja vastasel juhul tekiks laev tasuta taha ning kasutaja näeks ainult tausta.

Sinna tuleb lisada järgnevad koodiread:

```
Matrix[] muutused = new Matrix[laev.Meshes.Count];
laev.CopyAbsoluteBoneTransformsTo(muutused);
    foreach (ModelMesh mesh in laev.Meshes)
    {
      foreach (BasicEffect effect in mesh.Effects)
      {
          effect.EnableDefaultLighting();
        effect.World = muutused[mesh.ParentBone.Index] *
                Matrix.CreateRotationY(laevNurk);
        effect.View = Matrix.CreateLookAt(kaameraPositsioon,
          Vector3.Zero, Vector3.Up);
        effect.Projection = Matrix.CreatePerspectiveFieldOfView(
          MathHelper.ToRadians(45.0f), aspectRatio,
          1.0f, 10000.0f);
      }

      mesh.Draw();
    }
```

Koodi on küllalt palju, kuid kõik on vajalik. Kuna XNA raamistik teeb ise kõik vajalikud matemaatilised arvutused 3D objekti kuvamiseks, pääseb kasutaja niigi paljust koodi kirjutamisest.

ModelMesh on võrgustik, mis ümbritseb 3D objekti. Kuna ühel objektil võib olla mitu võrgustikku, tuleb kirjutatud koodi read ümbritseda foreach tsükliga, mis rakendab koodi igale 3D objekti võrgustiku osale.

Matrix[] muutused on maatriks, kus hoitakse 3D objektiga toimuvaid muutusi.

laev.CopyBoneTransformsTo kopeerib objektiga tehtud muudatused muutused maatriksisse.

Effect.World-i kasutatakse objektiga toimuvate muutuste näitamiseks. Objektiga toimuvate muutuste arvutamisel kasutatakse ette antud telge. Testimiseks võib CreateRotationY asendada näiteks CreateRotationX. Kohest muutust pole küll näha, kuid kui juhendis natuke edasi minna ja panna laev pöörlema on näha, et laev pöörleb teist telge pidi. Kui objekti ei taheta liigutada, pole seda ega ühtegi Transforms rida vaja.

Effect.View abil antakse teada kuhu kaamera on suunatud ehk kuidas ja millist ruumi punkti kasutaja näeb.

Effect.Projection hoolitseb selle eest, kuidas 3D ruum muuta arvuti ekraanile 2D pildiks, kuna ekraanipilt on siiski kahemõõtmeline. Lisaks määratakse sellega veel ka pildi sügavus ehk see, kui kaugele kasutaja näeb. Selles räägitakse täpsemalt ka neljandas juhendi punktis.

Kogu 3D objekti nägemiseks on vaja teda ruumis pöörata. Selleks tuleb minna Update() meetodisse, mis hoolitseb ekraanile kuvatava pildi uuendamise eest, ja kirjutada koodirida:

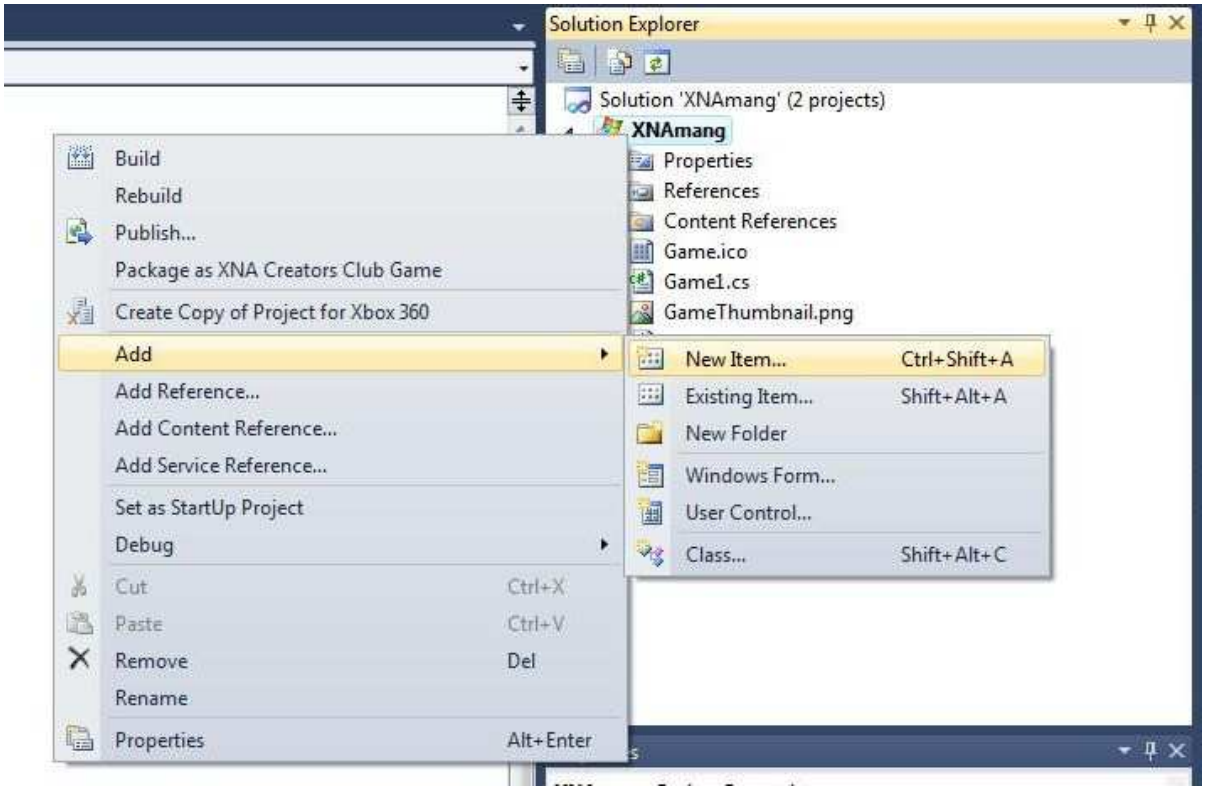
```
laevNurk+= 0.02f;
```

Kui kõik see tehtud tuleb vajutada rohelise Start Debugging nupule, mis asetseb üleval tööriista ribal või vajutada klahvile F5.

2. 3D objekti tekitamine kasutades uut faili

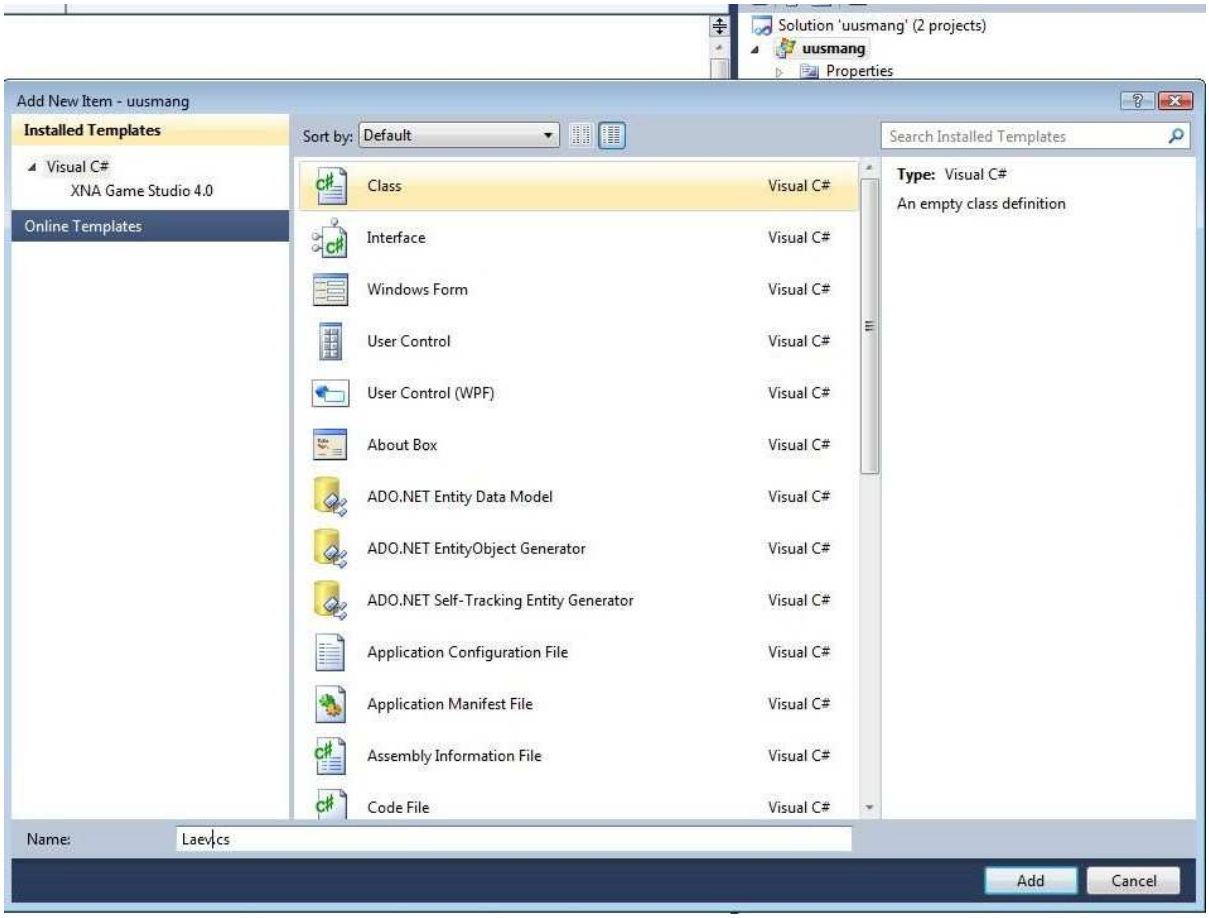
Tihti on vaja ekraanile kuvada mitut sarnast objekti. Kuna sellisel juhul läheks Game1.cs fail liiga pikaks ja raskelt järgitavaks, on alati kasulik objektidele luua oma enda fail, mille abil neid ekraanile kuvatakse.

Kõigepealt tuleb luua uus fail, mille nimeks võib panna näiteks laev. Uue faili lisamiseks tuleb teha hiire parem klõps projekt peakataloogil valida Add ning New Item.



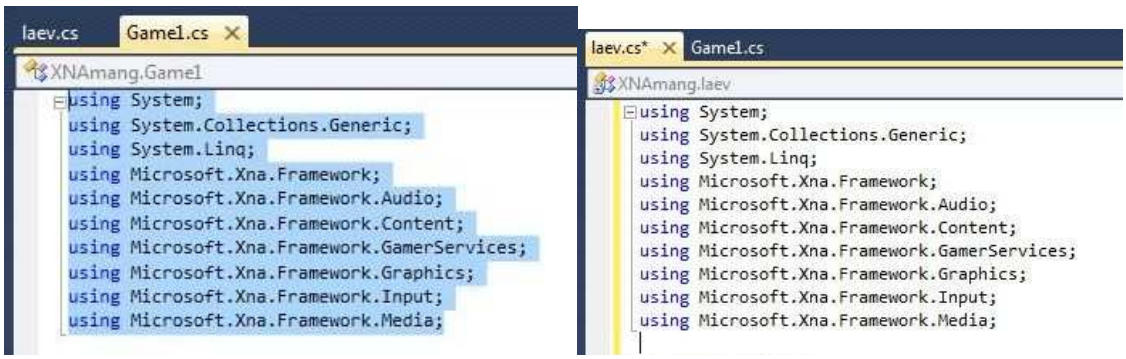
Joonis 3. Uue faili lisamine

Edasi tuleks valida faili tüübiks Class ja panna failile nimi ning vajutda Add.



Joonis 4. Faili tüüp

Esiteks tuleb Game1.cs failist kopeerida using laused ning asendada need Laev.cs olevatega.



Joonis 5. Using laused

Nüüd võib hakata Laev.cs faili sisu muutma. Laev.cs faili Laev klassi alla tuleb lisada järgmised koodiread:

```
public Model Mudel;
public Vector3 Positsioon = Vector3.Zero;
```

```
public float Nurk;
```

Fail Laev.cs hoiab informatsiooni objekti mudeli, positsiooni ja nurga kohta.

Kuna nüüd pole eelnevaid muutujaid Game1.cs failis vaja, tuleb need sealt ka kustutada ja faili sisu ka veel natuke täiendada. Game1 klassi alt tuleb eemaldad järgnevad read:

```
Model laev;
float laevNurk;
```

Nende asemel on vaja lisada koodirida:

```
Laev laev = new Laev();
```

See rida aitab Laev.cs failist võtta vajalikku informatsiooni.

Edasi tuleb muuta Game1.cs failis olevat koodi.

Esmalt peab muutma LoadContent() meetodit. Olemasolev koodirida

```
laev = Content.Load<Model>("Models//p1_wedge");
```

tuleb muuta ja kirjutada nii, et objekti mudel laetakse Laev.cs faili kaudu:

```
laev.Mudel = Content.Load<Model>("Models//p1_wedge");
```

Kui soovitakse, et laev jätkaks kohapeal keerlemist, tuleb muuta Update() meetodit,. Selleks on vajalik muuta olemasolev koodirida:

```
laevNurk+= 0.02f;
```

nii, et nurga suurus võetakse Laev.cs failist:

```
laev.Nurk += 0.02f;
```

Kui pole soovi, et laev tiirleks, võib selle algse koodirea lihtsalt kustutada.

Edasi tuleb muuta Draw() meetodit. Meetodi sisu tuleb muuta nii, et see näeks välja järgmiselt:

```
Matrix[] muutused = new Matrix[laev.Mudel.Meshes.Count];
        laev.Mudel.CopyAbsoluteBoneTransformsTo(muutused);

        foreach (ModelMesh mesh in laev.Mudel.Meshes)
      {
        foreach (BasicEffect effect in mesh.Effects)
        {
                    effect.EnableDefaultLighting();
          effect.World = muutused[mesh.ParentBone.Index] *
            Matrix.CreateRotationY(laev.Nurk);

          effect.View = Matrix.CreateLookAt(kaameraPositsioon,
            Vector3.Zero, Vector3.Up);
          effect.Projection = Matrix.CreatePerspectiveFieldOfView(
            MathHelper.ToRadians(45.0f), aspectRatio,
            1.0f, 10000.0f);
        }

        mesh.Draw();
      }
```

Nagu näha, tuleb kõigis koodiridades, mis varem sisaldasid objekti nime laev, asendada laev.Mudel -iga. Nii tehes võetakse laeva mudeli andmed Laev.cs failist.

3. Vajalikud muutused mitme objekti kuvamiseks

Esmalt tuleb lisada Laev.cs faili Laev klassi alla üks uus rida:

```
public Matrix[] Muutmine;
```

public Matrix[] Muutmine hoiab informatsiooni laeva tekstuuri muutuste kohta.

Järgmiseks tuleb lisada kaks uut koodirida Game1.cs faili Game1 klassi alla.

```
Matrix projektsiooniMatrix;
Matrix vaateMatrix;
```

projektsiooniMatrix ja vaateMatrix hoolitsevad samade asjade eest, nagu esimeses punktis Draw() meetodi all olevad effect.View ja effect.Projection.

Nüüd, kus kõik vajalik olemas, tuleb hakata eelnevalt Game1.cs faili kirjutatud koodi muutma ja uusi ridu lisama.

Kõigepealt tuleb Game1.cs failis asuvasse Initialize() meetodisse lisada mõned uued koodiread. Initialize() meetodit kasutatakse mitte graafiliste vahendite laadimiseks. Lisada tuleb järgnevad koodiread:

```
projektsiooniMatrix = Matrix.CreatePerspectiveFieldOfView(
MathHelper.ToRadians(45.0f),GraphicsDevice.DisplayMode.AspectRatio, 1.0f,
10000.0f);

vaateMatrix = Matrix.CreateLookAt(kaameraPositsioon,
Vector3.Zero, Vector3.Up);
```

Mida vaateMatrix ja projektsiooniMatrix täpsemalt teevad ja millest koosnevad, räägitakse lähemalt järgnevas juhendi punktis.

Järgmiseks tuleb lisada uus klass. Klassi võiks luua kohe pärast Initialize() meetodit, mis asub Game1.cs failis. Lisada tuleb järgnevad koodiread:

```
private Matrix[] Objekt(Model mudel)
    {
Matrix[] ObjektiMuutused = new Matrix[mudel.Bones.Count];
        mudel.CopyBoneTransformsTo(ObjektiMuutused);

        foreach (ModelMesh mesh in mudel.Meshes)
        {
           foreach (BasicEffect effect in mesh.Effects)
           {
              effect.EnableDefaultLighting();
              effect.Projection = projektsiooniMatrix;
              effect.View = vaateMatrix;
           }
        }
        return ObjektiMuutused;
    }
```

Seda klassi on vaja, kui soovitakse lisada mängu rohkem objekte. Selle klassi puudumisel tuleks kõigi objektide jaoks kirjutada esimeses peatükis Draw() meetodisse kirjutatud koodiread, mis hoolitseks objekti tekstuuri ja muutuste eest. Seda klassi saab kasutada kõigi objektide jaoks. See säästab kasutajat paljust koodi kirjutamisest.

Järgmine muutus tuleb teha Game1.cs failis asuvas LoadContent() meetodis. Tuleb lisada ka mudelile antav tekstuur. See koodirida tuleb lisada peale koodirida laev.Mudel = Content.Load<Model>("Models//p1_wedge"); kuna muidu ei ole veel olemas 3D objekti millele tekstuur lisatakse. Tuleb lisada järgneb koodirida:

```
laev.Muutmine = Objekt(laev.Mudel);
```

Tekstuuri fail on esimeses punktis juba mängu projektile lisatud ning eelnevates punktides kasutati seda automaatselt.

Järgmised muudatused peab tegema Game1.cs failis asuvas Draw() meetodis. Meetodi koodi tuleb muuta nii, et ta näeks välja järgmiselt (ehk vana kood tuleb kustutada ning selle asemele lisada järgnev:

```
protected override void Draw(GameTime gameTime)
    {
      GraphicsDevice.Clear(Color.CornflowerBlue);

Matrix laevMuutusteMatrix = Matrix.CreateRotationY(laev.Nurk)
      * Matrix.CreateTranslation(laev.Positsioon);
      JoonistaMudel(laev.Mudel, laevMuutusteMatrix, laev.Muutmine);

      // TODO: Add your drawing code here

      base.Draw(gameTime);
    }
```

Matrix laevMuutusteMatrix hoiab endiselt informatsiooni laeva nurga ja positsiooni kohta. Erinevus tekib aga JoonistaMudel klassi esile kutsumisel. Seda klassi ei ole veel olemas, aga see luuakse järgmisel sammul. JoonistaMudel klass võtab joonistamisel arvesse kõik selle objektiga seotud vajalikud muutujad. Antud juhul siis laev.Mudel, laevMuutusteMatrix ja laev.Muutmine .

Edasi tuleb luua uus klass. See klass tuleks luua otse peale Draw() meetodit, mis asub Game1.cs failis. Üles tuleb leida Draw() meetodi lõpu loogeline sulg. Lõpetavat loogelist sulgu saab kergesti üles leida vajutades alustava loogelise sulu peale. Alustavale ja lõpetavale loogelisele sulule peaks ümber tekkima hall kast. Tuleb lisada järgnevad koodiread:

```
public static void JoonistaMudel(Model mudel, Matrix mudelMuutused,
```

```
    Matrix[] ObjektiMuutmine)
{
    foreach (ModelMesh mesh in mudel.Meshes)
    {
     foreach (BasicEffect effect in mesh.Effects)
      {
         effect.World =
            ObjektiMuutmine[mesh.ParentBone.Index] *
            mudelMuutused;
      }
                mesh.Draw();
    }
}
```

Nagu varem öeldud, hoolitseb see klass selle eest, et kõik objektiga tehtud muutused jõuaksid õigesti ekraanile.

Kui kõik see tehtud, tuleb vajutada rohelise Start Debugging nupule, mis asetseb üleval tööriista ribal või vajutada klahvile F5. Koheseid muutusi pole küll näha, aga nüüd on võimalik kergemini kuvada ekraanile mitut objekti, mida läheb vaja hilisemate punktide täitmiseks.

4. Kaamera ja projektsioon

Kaamera seadistamine käib läbi kolme käsurea. Kõik need koodiread on juba olemas.

Esiteks:

```
Vector3 kaameraPositsioon = new Vector3(0.0f, 50.0f, 5000.0f);
```

Selle koodirea abil antakse kaamerale koht, kus ta ruumis asub. Koha määramine käib läbi X,Y ja Z koordinaatide.

Teiseks:

```
vaateMatrix = Matrix.CreateLookAt(kaameraPositsioon,
Vector3.Zero, Vector3.Up);
```

Mis asub Game1.cs failis olevas Initialize() meetodis.

Selle koodirea abil antakse teada, kuhu kaamera on suunatud ja millist ruumi punkti kasutaja näeb.

CreateLookAt koosneb kolmest osast. Esimese osaga kaameraPositsioon pannakse kaamera ruumis kohta, mis varem kaamerale oli ette antud. Teine osa Vector3.Zero näitab, kuhu kaamera ruumis on suunatud - siinkohal Vector3.Zero ehk ruumi keskpunkti. Kolmas osa Vector3.Up näitab, mis pidi on kaamera pilt pööratud. Testimiseks võib koodis Vector3.Up asendada Vector3.Down ja laev peaks olema tagurpidi.

Joonis 6. Tagurpidi laev

Kolmandaks:

```
projektsiooniMatrix = Matrix.CreatePerspectiveFieldOfView(
MathHelper.ToRadians(45.0f), GraphicsDevice.DisplayMode.AspectRatio, 1.0f,
10000.0f);
```

Mis asub Game1.cs failis olevas Initialize() meetodis.

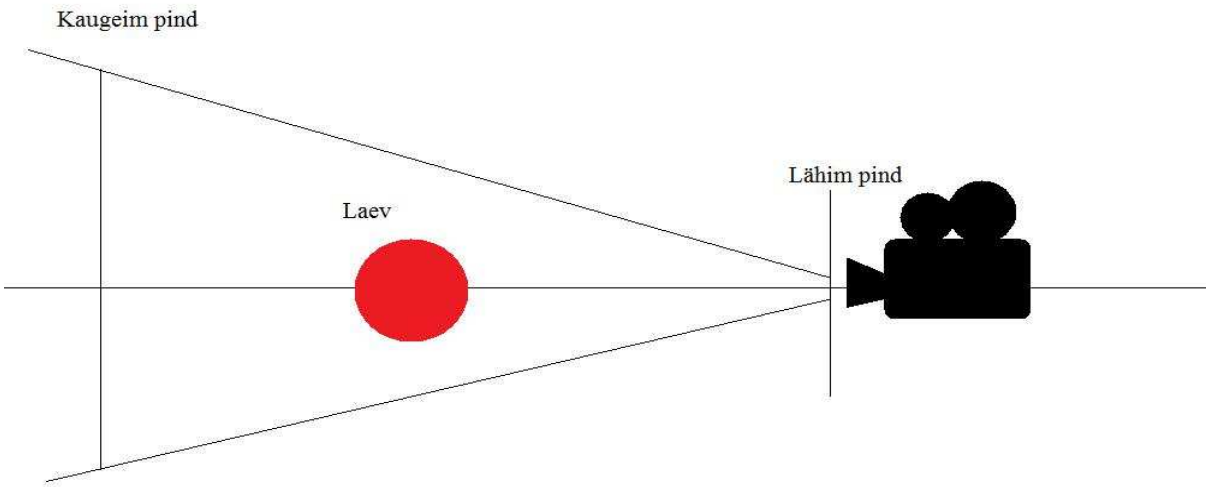
CreatePrespectiveFieldOfView koosneb neljast osast:

Esimese osaga MathHelper.ToRadians(45.0f) arvutatakse vaateväli, mida kasutaja näeb.

Teise osaga GraphicsDevice.DisplayMode.AspectRatio arvutatakse vaatevälja laiuse ja kõrguse suhe ehk laius jagatud kõrgusega.

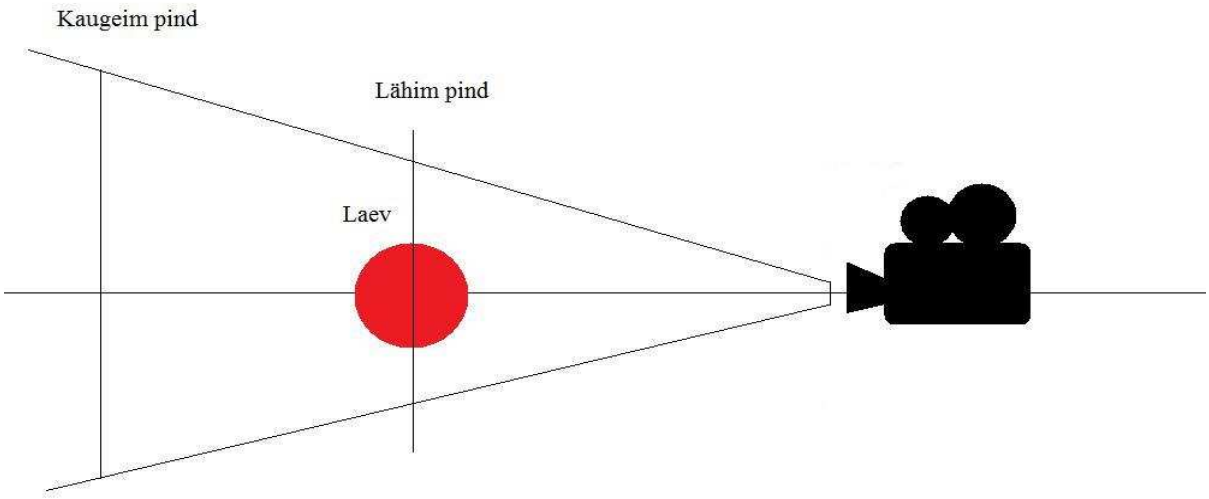
Kolmanda osaga 1.0f antakse kaamera lähima pinna kaugus ehk kui kaugel asub kaamera silm.

Neljanda osaga 10000.0f antakse kaamerale kaugeim pind ehk kui kaugele kaamera näeb. Neid nelja komponenti kasutades tekitakse pilt, mis kuvatakse kasutaja ekraanile.



Joonis 7. Algne kaamera

Hetkel on kõik hästi, kuid kui kaamera lähim pind liigutada suurusele 5000.0f, on näha, kuidas pool laeva vahepeal ära kaob. See on tingitud sellest, et pool laeva on kaamera lähimast pinnast möödas ja kaob vahepeal vaateväljast.



Joonis 8. Muudetud kaamera

Edasise materjali läbimiseks tuleb kaamerat natuke liigutada.

Kõigepealt on vaja anda kaamerale uus positsioon. Selleks tuleb minna Game1.cs faili ja üles leida Game1 klassis olev koodirida :

```
Vector3 kaameraPositsioon = new Vector3(0.0f, 50.0f, 5000.0f);
```

Koodirida tuleb muuta nii , et see näeks välja järgmiselt:

```
Vector3 kaameraPositsioon = new Vector3(0.0f, 10000.0f,30000.0f);
```

Jägmiseks tuleb muuta Initialize() meetodis olevat koodirida:

```
projektsiooniMatrix = Matrix.CreatePerspectiveFieldOfView(
MathHelper.ToRadians(45.0f),GraphicsDevice.DisplayMode.AspectRatio, 1.0f,
10000.0f);
```

Koodirida tuleb muuta nii, et see näeks välja järgmiselt:

```
projektsiooniMatrix = Matrix.CreatePerspectiveFieldOfView(
MathHelper.ToRadians(45.0f),GraphicsDevice.DisplayMode.AspectRatio,
100.0f, 40000.0f);
```

Nüüd tuleb vajutada Start Debugging nuppu või klavituurilt F5. Laev peaks olema kaamerast palju kaugemal.

Joonis 9. Uus vaade

5. Laeva asukoha muutmine nuppude abil

Kui senini keerles laev iseenesest ekraanil, siis selles peatükis näidatakse, kuidas saab laeva ise liigutada.

Esiteks tuleb minna Laev.cs faili ja Laev klassi lisada kolm uut koodirida:

```
public Matrix NurkMatrix;
public Vector3 Kiirus = Vector3.Zero;
private static KeyboardState nupp;
```

NurkMatrix hakkab hoidma informatsiooni laeva positsiooni ja Kiirus laeva kiiruse muutuste kohta. KeyboardState nupp; koodirida on vaja hiljem laeva liigutamiseks. nupp hakkab hoidma informatsiooni klaviatuuri seisundi kohta.

Edasi tuleb luua Laev.cs fails asuva Laev klassi sisse uus meetod:

```
public void Update()
{
}
```

Sinna meetodi sisse tuleb kirjutada vajalikud koodiread, et XNA raamistik saaks aru, kui teatud nuppu on vajutatud ja mida selle korral tegema peab.

Kõigepealt tuleb sinna meetodisse lisada järgnev koodirida:

```
nupp = Keyboard.GetState();
```

See koodirida kontrollib klaviatuuri staatust ehk siis, kas mõnda nuppu vajutatakse või mitte.

Järgmiseks tuleb sinna meetodi alla lisada koodirida NurkMatrix-i arvutamiseks järgmiselt:

```
NurkMatrix = Matrix.CreateRotationX(MathHelper.PiOver2) *
Matrix.CreateRotationZ(Nurk);
```

Kuna nüüd on olemas käsklus, mis kontrollib klaviatuuri staatust ja arvutatava NurkMatrix-i väärtust, võib edasi liikuda nupu vajutuste juurde.

Laev.cs failis eelnevalt lisatud Update() meetodisse tuleb lisada järgnevad koodiread:

```
if (nupp.IsKeyDown(Keys.Left))
{
   Nurk += 0.05f;

}
if (nupp.IsKeyDown(Keys.Right))
{
   Nurk -= 0.05f;
}

if (nupp.IsKeyDown(Keys.Up))
{
   Kiirus += NurkMatrix.Forward * 6.0f;

}
```

Esimene if lause tegeleb sellega, et kui klaviatuuril on vasaku noole klahv alla vajutatud, lahutab ta laeva nurgast 0.05 ühikut. Järgmine if lause käitub analoogselt, ainult et parema noole klahvi vajutades liidab laeva nurgale 0.05 ühikut. Kolmas if lause liidab laeva kiirusele NurkMatrix-ist saadud väärtuse, mis on omakorda korrutatud 6.0 ühikuga.

NurkMatrix.Forward arvutatakse XNA raamistiku poolt automaatselt, kasutades objekti nurka ehk siinkohal nurga suurust, kuhu poole laev on pööratud, et teada saada, kuhu poole peab laeva liigutama.

Nüüd, kus kõik vajalik on Laev.cs faili lisatud, tuleb pöörduda Game1.cs faili juurde ja seda natuke täiustada.

Kõigeapealt tuleb täiustada Game1.cs failis asuvat Update() meetodit:

Esiteks tuleb kustutada järgnev koodirida, kuna enam pole vaja, et laev omapäi pöörleks:

```
laev.Nurk += 0.02f;
```

Järgmiseks tuleb lisada Game1.cs failis asuvasse Update() meetodisse koodirida, mis arvutaks laeva uue asukoha, kui laeva kiirust on muudetud. Selleks tuleb lisada järgnev rida:

```
laev.Positsioon += laev.Kiirus;
```

Selle koodirea juures on väike viga. Laeva positsioonile liidetakse pidevalt saadud kiirust, seega ühekordsel üles noole klahvile vajutamisel lendab laev ekraanist välja. Selleks, et seda ei juhtuks on vaja ühte uut koodirida:

```
laev.Kiirus *= 0.95f;
```

See koodirida korrutab laeva kiirust pidevalt 0.95 ühikuga läbi. Seega, kui kiiruse lisamise nuppu enam ei vajutata aeglustub laeva liikumine, kuni jääb täielikult seisma. Testimiseks võib selle koodirea kustutada, et näha mis juhtub.

Järgmiseks tuleb lisada koodirida, mis pidevalt kutsuks esile Laev.cs failis asuvas Update() meetodis tehtavaid muudatusi. Selleks tuleb lisada Game1.cs failis asuvasse Update() meetodisse koodirida:

```
laev.Update();
```

Nüüd tuleb minna Game1.cs failis asuva Draw() meetodi juurde ja muuta seal üks koodirida:

```
Matrix laevMuutusteMatrix = Matrix.CreateRotationY(laev.Nurk)
* Matrix.CreateTranslation(laev.Positsioon);
```

Koodi tuleb muuta nii, et see näeks välja järgmiselt:

```
Matrix laevMuutusteMatrix = laev.NurkMatrix *
Matrix.CreateTranslation(laev.Positsioon);
```

Nüüd arvutatakse laeva positsiooniga toimuvad muutused kasutades Laev.cs failis asuvat NurkMatrix-it.

Viimase koodireaga muutus ka natuke kasutajale kuvatav pilt, mis on tingitud sellest, et NurkMatrix-i arvutamisel kasutakse kahte koordinaattelge.
Siinkohal võib vajutada Start Debugging nupule või klaviatuurilt F5.

Võib märgata, et laev lendab nüüd mööda kumerat pinda. Mida rohkem pildi alumise serva äärde laev jõuab, seda kaugemale ta pildist läheb. Natuke mõnusamaks saab olukorda muuta, kui minna Game1.cs failis asuvasse Initialize() meetodisse ning muuta koodirida:

```
vaateMatrix = Matrix.CreateLookAt(kaameraPositsioon,
Vector3.Zero, Vector3.Up);
```

ja asendada see koodreaga:

```
vaateMatrix = Matrix.CreateLookAt(kaameraPositsioon,
Vector3.Zero, Vector3.Down);
```

Sellega keeratakse kasutajale kuvatavat pilti ja nüüd liigub laev kuvatavast seda kaugemale, mida rohkem ta pildi ülemisele servale läheneb.

Viimane muutus on maitse asi ja seda ei pea tingimata tegema.

Kuna nüüd on võimalik kasutajal ise laevaga ringi sõita, on ka võimalus, et laev võib pildist välja lennata ja ära kaduda. Selleks, et laev tagasi ekraanile saada, tuleb lisada Laev.cs failis olevasse Update() meetodisse üks uus if lause:

```
if(nupp.IsKeyDown(Keys.X))
{
   Nurk = 0.0f;
   Positsioon = Vector3.Zero;
   Kiirus = Vector3.Zero;

}
```

Nupule X vajutades viiakse laev tagasi oma algasendisse.

Laeva väljumist ekraanilt saab ka takistada. Selleks tuleb minna Laev.cs failis asuva Update() meetodisse ja lisada sinna järgmised koodiread:

```
 if (Positsioon.X > 23000.0f)
{
   Positsioon.X -= 2 * 23000.0f;
}
if (Positsioon.X < -23000.0f)
{

   Positsioon.X += 2 * 23000.0f;
}
if (Positsioon.Y > 16000.0f)
{
   Positsioon.Y -= 2 * 16000.0f;
}
if (Positsioon.Y < -16000.0f)
{
   Positsioon.Y += 2 * 16000.0f;
}
```

Need if laused hoolitsevad selle eest, et kui laev peaks jõudma antud ühiku kaugusele ehk siinkohal pildi servani, siis tekitatakse ta tagasi pildi vastasserva. Nii tehes jääb tunne, et kui laev lendab pildist välja, tuleb ta ekraanipildi vastasserva alt jälle välja, kuid tegelikult on tegu vaid lihtsate matemaatiliste tehetega.

Eelnevalt tehtud if lauset, mis X nupu vajutamise korral viib laeva tagasi algasendisse, on hiljem vaja, seega seda ära kustutada pole mõtet.

Kui kõik see tehtud, tuleb vajutada rohelise Start Debugging nupule, mis asetseb üleval tööriistaribal, või vajutada klahvile F5. Nüüd peaks olema kasutajal võimalik laevaga ise ringi lennata.

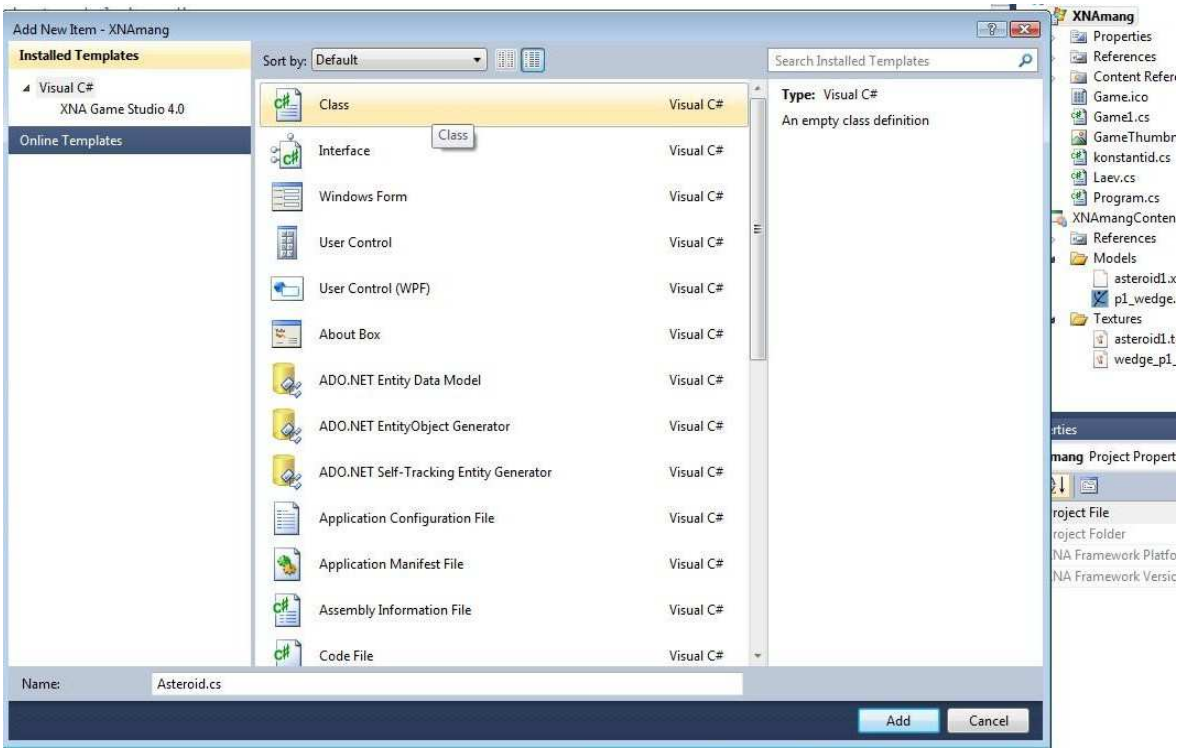
6. Teiste objektide tekitamine ekraanile

Selles osas näidatakse, kuidas tekitada mitut uut objekti ekraanile ja kuidas need liikuma panna. Meie mängus on uuteks objektideks asteroidid.

Nagu ka kõige esimese objekti puhul, milleks oli laev, on ka nüüd vaja objekti mudeli ja tekstuuri faili.

Objekti mudeli fail tuleb jälle lisada Content kataloogis asuvasse Models kataloogi ja tekstuuri fail Textures kataloogi. Models kataloogi tuleb lisada fail nimega asteroid1.x ja Textures kataloogi fail nimega asteroid1.tga.

Kui vajalikud failid on lisatud, tuleb luua uus fail, mis hoiaks asteroidi informatsiooni. Selleks tuleb, nagu ka varem, vajutada projekti peakataloogil ja valida Add ning New Item. Faili tüübiks tuleb valida Class ja nimeks võiks panna Asteroid.cs.



Joonis 10. Faili tüüp

Kui uus fail on loodud tuleb sinna ka seekord kopeerida Game1.cs failist using laused. Samamoodi nagu ennem sai tehtud Laev.cs faili puhul.

Nüüd tuleb, aga teha väike muudatus Asteroid klassis. Kuna asteroide tekitatakse ekraanile palju, tuleb asteroidi klass muuta struct-iks. Selleks tuleb lihtsalt Asteroid-ile eelnev sõna class asendada sõnaga struct.

```
struct Asteroid
{
```

Klassi struct-iks muutmine võimaldab XNA raamistikul kergemat *garbage collection*-it ja objekti eest hoolitsemist. *Garbage collection* püüab vabastada arvuti mälu prügist ehk siis objektidest, mida programm tegelikult enam ei kasuta. Struct optimeerib elus objektide arvu ehk kõiki objekte ei kontrollita korraga. Selles näites struct-i mitte kasutamine tooks mitmeid tõrkeid, mis on just nimelt seotud ülekoormusega.

Edasi tuleb Asteroid.cs failis asuva Asteroid klassi alla lisada mõned uued koodiread:

```
public Vector3 Positsioon;
public Vector3 Suund;
public float Kiirus;
public Model Mudel;
public Matrix[] Muutmine;
```

Nagu ka laeva puhul hoiab Positsioon informatsiooni asteroidi asukoha ja Kiirus asteroidi kiiruse kohta. Suund hoiab informatsiooni selle kohta, millises suunas asteroid liigub.

Mudel hoiab informatsiooni asteroidi mudeli ehk objekti kohta ja Muutmine on maatriks, kus hoitakse informatsiooni iga asteroidi tekstuuri muutmise kohta.

Nüüd on vaja ka Asteroid.cs faili luua uus Update() meetod, mis tegeleks asteroidi asukoha muutustega. Jällegi tuleb jälgida, et Update() meetod tekitatakse struct Asteroid sisse. Kood on sama nagu ka laeva faili puhul:

```
public void Update(float aeg)
{
}
```

Silmas peab pidama seda, et seekord on meetodile järgnevates sulgudes ka muutuja float aeg. Mida float aeg täpsemalt teeb, selgitatakse natuke hiljem.

Kui uus meetod loodud, tuleb sinna sisse kirjutada mõned koodiread.
Esiteks tuleb kirjutada järgnev koodirida:

```
Positsioon += Suund * Kiirus * 5.0f * aeg;
```

See rida on vajalik, et arvutada asteroidi positsiooni.

Edasi tuleb Update() meetodisse lisada read, mis takistaks asteroidil ekraanilt kaduda:

```
if (Positsioon.X > 23000.0f)
{
   Positsioon.X -= 2 * 23000.0f;
}
if (Positsioon.X < -23000.0f)
{

   Positsioon.X += 2 * 23000.0f;
}
if (Positsioon.Y > 16000.0f)
{
```

```
        Positsioon.Y -= 2 * 16000.0f;
    }
    if (Positsioon.Y < -16000.0f)
    {
        Positsioon.Y += 2 * 16000.0f;
    }
```

Need if laused võib ka kopeerida Laev.cs-is asuvast Update() meetodist, kuna nad teevad täpselt sama asja ning ka kood on sama.

Kui kõik see tehtud tuleb minna Game1.cs faili juurde.

Esiteks tuleb Game1 klassi, mis asub Game1.cs failis, lisada järgnevad read:

```
Asteroid asteroid = new Asteroid();
Asteroid[] Asteroidid = new Asteroid[10];
Random juhuslik = new Random();
```

Asteroidid on massiiv, kus hoitakse kõigi mängus olevate asteroidide kohta informatsiooni. Massiiv on andmestruktuur, mis lubab samatüübilisi andmeid koondada ühise nime alla ning teha andmeelementidel vahet järjekorra numbri ehk indeksi järgi. Asteroid[] ja new Asteroid[10] näitavadki, et massiivis nimega Asteroidid hoitakse kümmet asteroidi, millel on kõigil ühised omadused, mis tulenevad Asteroid.cs failist. Random juhuslik = new Random(); aitab luua juhuslikke arve, mida hiljem tarvis läheb.

Edasi tuleb lisada üks uus koodirida Initialize() meetodisse Game1.cs failis. Järgnev koodirida tuleks lisada base.Initialize(); rea kohale:

```
UuendaAsteroidid();
```

UuendaAsteroidid() meetodit ei ole veel olemas. Järgmise sammuna tulebki see meetod luua. Meetodi võiks luua peale Initialize() meetodit. Lisada tuleb järgnevad koodiread:

```
private void UuendaAsteroidid()
{
   float xAlgus = 1000;
   float yAlgus = 4000;
   for (int i = 0; i < 10; i++)
   {
      if (juhuslik.Next(2) == 0)
      {
         xAlgus = (float)- 23000.0f;
      }
      else
      {
         xAlgus = (float)+ 23000.0f;
      }
      yAlgus =
      (float)juhuslik.NextDouble() * 16000.0f;
      Asteroidid[i].Positsioon = new Vector3(xAlgus, yAlgus, 0.0f);
      double Nurk = juhuslik.NextDouble() * 2 * Math.PI;
      Asteroidid[i].Suund.X = -(float)Math.Sin(Nurk);
      Asteroidid[i].Suund.Y = (float)Math.Cos(Nurk);
      Asteroidid[i].Kiirus = 100 +
        (float)juhuslik.NextDouble() * 300;
   }
}
```

Koodi on päris palju, kuid kõik see on matemaatiliste arvutuste kogum.

`float xAlgus = 1000;` `float yAlgus = 4000;`

on asteroidi alguspunkti, mille järgi hiljem arvutatakse koha, kuhu asteroid täpselt tekib, X ja Y koordinaadid. Edasi hakatakse arvutama igale kümnele asteroidile enda kiirust ning täpset alguskohta. Iga asteroidi X ja Y koordinaadi arvutamiseks kasutatakse juhuslikke arve ja

ruumile antud suurust, mida kasutatakse ka asteroidide ja laeva tagasitoomiseks ekraanile, kui nad pildist väljuma peaksid. Asteroidi Kiirus ja Suund arvutamisel kasutatakse jällegi juhuslike arve, kusjuures asteroidi suuna arvutamiseks kasutatakse ka muutujat Nurk.

Järgmiseks tuleb liikuda Game1.cs failis asuva LoadContent() meetodi juurde, et mängu laadida asteroidi mudel ja tekstuur. Selleks tuleb LoadContent() meetodisse lisada järgnevad koodiread:

```
asteroid.Mudel = Content.Load<Model>("Models//asteroid1");
asteroid.Muutmine = Objekt(asteroid.Mudel);
```

Edasi tuleb minna Game1.cs failis asuva Update() meetodi juurde. Sinna tuleb lisada järgnevad koodiread:

```
float aeg = (float)gameTime.ElapsedGameTime.TotalSeconds;

for (int i = 0; i < 10; i++)
{
   Asteroidid[i].Update(aeg);
}
```

Asteroid.cs faili alla Update() meetodi loomist kirjeldades sai märgitud, et muutuja aeg vajalikkusest räägitakse hiljem. Nüüd on õige koht rääkida muutujast aeg lähemalt.

Muutuja aeg laseb asteroide puudutavat informatsiooni uuendada teatud aja tagant mitte iga Update() meetodiga. Muutujat aeg mitte kasutades lendaksid asteroidid ekraanil ringi väga suurel kiirusel.

Kui see kõik on tehtud, tuleb asteroidid vaid ekraanile joonistada. Selleks on vaja minna Game1.cs failis asuvasse Draw() meetodisse ja lisada sinna järgnevad koodiread:

```
for (int i = 0; i < 10; i++)
   {
```

```
Matrix asteroidiMuutmiseMatrix =
Matrix.CreateTranslation(Asteroidid[i].Positsioon);
JoonistaMudel(asteroid.Mudel,
 asteroidiMuutmiseMatrix,asteroid.Muutmine);

 }
```

for tsükli abil joonistatakse ekraanile kõik kümme asteroidi, mitte ainult üks.

Nüüd võib vajutada Start Debugging nupule või klaviatuurilt F5.

Kui ekraanile tekkinud asteroid peaksid olema mustad, siis tuleb minna ja mängu projekti alt üles leida fail nimega asteroid1.x, mis peaks asuma Models kataloogis. Sellele failile ühekordselt vajutades peaks ekraani alla paremasse nurka tekkima uus aken erinevate muutujatega. Juhul kui seda akent ei ole, tuleb asteroid1.x failil teha klikk hiire parema klahviga ning valida tekkinud rippmenüüst Properties. Sealt tuleb üles leida muutuja nimega Content Processor ning vajutada talle eelnevale noolele. Selle peale peaks kasutajale avanema mitu uut muutujat. Sealt tuleb üles leida muutuja nimega Premultiply Texture Alpha ning selle väärtus muuta False-ks. Seda saab teha eelneva väärtuse True peale vajutades.

Nüüd mängu uuesti käivitades peaksid asteroidid omama õiget tekstuuri.

7. Asteroidi ja Laeva kokkupuute kontrollimine

Selles osas näidatakse, kuidas kontrollida, millal kaks objekti kokku puutuvad.

XNA raamistik teeb peaaegu kõik selleks vajalikud matemaatilised arvutused ise kasutaja eest ära.

Kõigepealt tuleb nii laevale kui asteroididele luua üks uus muutuja, millega määratakse, kas objekt on „elus“ või mitte. See tähendab, et kui näiteks laev ei ole „elus“, siis teda lihtsalt ekraanile ei joonistata.

Esiteks tuleb minna Laev.cs faili Laev klassi ja lisada järgnev koodirida:

```
public bool elus = true;
```

Selle koodirea abil näidatakse, kas laev on elus või ei. Kõigepealt on laev elus seega ka

```
elus = true;
```

Järgmiseks tuleb lisada koodirida ka Asteroid.cs failis asuvasse Asteroid structi. Kuna asteroidi puhul on tegemist struct-iga, tuleb lisada järgnev rida:

```
public bool elus;
```

Kuna asteroide on palju, tuleb neile elus väärtus anda Game1.cs failis, muidu käiks see muutuja ühiselt kõigi asteroidide kohta.

Nüüd, kus laeval ja asteroidil on elus muutuja olemas, võib edasi liikuda Game1.cs failis asuva UuendaAsteroidid() meetodi juurde. Selles meetodis tuleb anda igale asteroidile eraldi elus muutuja väärtus. Koodirida tuleb lisada UuendaAsteroidid() meetodis asuva `for (int i = 0; i < 10; i++)` tsükli viimase reana ehk kohe pärast `Asteroidid[i].Kiirus = 100 + (float)random.NextDouble() * 300;` koodirida. Sinna tuleb lisada järgnev koodirida:

```
Asteroidid[i].elus = true;
```

Selle koodireaga kindlustatakse asjaolu, et elus muutujale antakse väärtus just selle kindla asteroidi jaoks, mida hetk tagasi töödeldi for tsüklis.

Seejärel tuleb minna Game1.cs failis asuva Update() meetodi juurde. Sinna meetodisse tuleb lisada järgnevad koodiread:

```
if (laev.elus)
{
 BoundingSphere laevKera = new BoundingSphere(
 laev.Positsioon, laev.Mudel.Meshes[0].BoundingSphere.Radius * 0.5f);
  for (int i = 0; i < Asteroidid.Length; i++)
  {
     if (Asteroidid[i].elus)
     {
   BoundingSphere asteroidKera = new
   BoundingSphere(Asteroidid[i].Positsioon,
    asteroid.Mudel.Meshes[0].BoundingSphere.Radius * 0.95f);
     if (asteroidKera.Intersects(laevKera))
        {
              laev.elus = false;
              Asteroidid[i].elus = false;

              break;
        }
      }
  }
}
```

Nagu varem öeldud, teeb XNA raamistik kasutaja eest peaaegu kõik kokkupuute kontrollimiseks vajalikud matemaatilised arvutused ise ära. Kokkupuudet saab kasutaja kontrollida luues BoundingSphere, mis on nähtamatu kera, oma objektide ümber ning hiljem if lause ja Intersects käsu abil kontrollida, kas objektid puutuvad kokku või mitte. BoundingSphere-le, mis kannab nime laevKera, koosneb kahest komponendist. Esimene on kera positsioon, mis siinkohal peab olema sama, mis laeva oma (laev.Positsioon). Teine on kera suurus, mis selle näite põhjal peab olema poole väiksem, kui laeva enda oma (laev.Mudel.Meshes[0].BoundingSphere.Radius * 0.5f). Kuna laev ei ole ümmargune, tuleb laeva ümbritsev kera natuke väiksem teha ehk korrutada läbi 0.5f-ga. Kui kera väiksemaks muuta, võib juhtuda, et laev ja asteroid „puutuvad kokku“ ka siis, kui visuaalset puudet ei toimu. Sarnane kera tuleb luua ka asteroidide ümber. Kui mõlema objekti ümber kerad tehtud, tuleb vaid kontrollida, millal nad kokku puutuvad ja mida selle peale tegema peab. Selle eest hoolitseb koodiosa:

```
if (asteroidKera.Intersects(laevKera))
        {
            laev.elus = false;
            Asteroidid[i].elus = false;

            break;
        }
```

Kui laev ja asteroid kokku puutuvad, muudetakse laeva ja temaga kokkupuutunud asteroidi elus väärtus false-ks.

Edasi tuleb liikuda Game1.cs failis asuva Draw() meetodi juurde. Draw() meetodis on vaja laeva ja asteroidide joonistamist muuta nii, et see oleneks muutuja elus väärtusest. Selleks tuleb teha kaks muudatust:

Esiteks tuleb koodirida

```
JoonistaMudel(laev.Mudel, laevMuutusteMatrix, laev.Muutmine);
```

asendada

```
   if (laev.elus)
   {
JoonistaMudel(laev.Mudel, laevMuutusteMatrix,laev.Muutmine);

    }
```

Ehk enne, kui kutsutakse välja JoonistaMudel() meetod, kontrollitakse, kas laeva muutuja elus väärtus on true. Sama põhimõtet tuleb rakendada ka asteroidide korral.

Teiseks tuleb koodirida:

```
JoonistaMudel(asteroid.Mudel,asteroidiMuutmiseMatrix,
asteroid.Muutmine);
```

asendada

```
        if(Asteroidid[i].elus)
        {
   JoonistaMudel(asteroid.Mudel,asteroidiMuutmiseMatrix,

   asteroid.Muutmine);

      }
```

Järgmiseks tuleb minna Laev.cs failis asuvasse Update() meetodisse. On vaja üles leida if(nupp.IsKeyDown(Keys.X)) ehk if lause, mis tegeleb peale nupu X vajutamist vajalike muutustega. Sinna tuleb lisada üks uus koodirida:

```
elus = true;
```

Kui nüüd nupule X vajutada, tekib ka laev ekraanile tagasi, kui ta on eelnevalt asteroidiga kokku põrkunud.

Kui klikkida nüüd Start Debugging nupule või vajutada klaviatuurilt F5 klahvi, näeb mäng välja nagu varem, kuid kui laevaga vastu asteroidi sõita, kaovad asteroid ja laev ekraanilt.

8. Laevale kuulide lisamine

Kõigepealt tuleb lisada projektile uue objekti mudel ja tekstuur. Kuuli mudeliks kasutatakse faili nimega *pea_proj.x*, mis on vaja lisada Models kataloogi ja tekstuuri failiks kasutatakse faili *pea_proj.tga*, mis lisataks Textures kataloogi. Järgmiseks tuleb luua uus fail. Faili tüüp peaks ka seekord olema Class ja nimeks võiks panna Kuul.cs.

Kuuli fail hakkab olema väga sarnane asteroidi failiga, kuna nad omavad samu osi. Koguni nii sarnane, et võib võtta kogu Asteroid.cs sisu ja kopeerida selle Kuul.cs faili. Kopeerimise juures peab silmas pidama seda, et kui Asteroid.cs failis sisu on paigutatud Kuul.cs faili, on vaja koodirida:

```
struct Asteroid
```

asendada koodireaga:

```
struct Kuul
```

Edasi tuleb muuta Kuul.cs failis asuvad Update() meetodit. Asteroid.cs failis asuv Update() meetod hoolitseb kahe asja eest:

Esiteks: positsiooni muutmise eest, mida on vaja ka kuuli korral.

Teiseks: objekti tagasi ekraanile tekitamiseks, juhul kui see peaks pildist väljuma. Seda pole kuulide juures vaja, vaid kuulid, mis väljuvad pildist tuleks ära kaotada.
Kuulide ärakaotamiseks tuleb Kuul.cs failis asuvat Update() meetodit natuke muuta. Meetodi sisu peab välja nägema järgmiselt:

```
Positsioon += Suund * Kiirus * 5.0f * aeg;

  if (Positsioon.X > 23000.0f ||
  Positsioon.X < -23000.0f ||
```

```
Positsioon.Y > 16000.0f ||
Positsioon.Y < -16000.0f)
{
   elus = false;
}
```

See if lause muudab kuuli elus väärtust juhul, kui kuul väljub mängupildist.

Edasi tuleb minna Game1.cs failis asuva Game1 klassi juurde. Game1 klassi alla tuleb lisada kaks uut koodirida:

```
Kuul kuul = new Kuul();
Kuul[] Kuulid = new Kuul[10];
```

Need koodiread teevad kuuli puhul sama, mida asteroidi omad asteroidi puhul.

Järgmiseks tuleb liikuda Game1.cs failis LoadContet() meetodi juurde. Sinna meetodi alla on vaja lisada järgnevad koodiread:

```
kuul.Mudel = Content.Load<Model>("Models//pea_proj");
kuul.Muutmine = Objekt(kuul.Mudel);
```

Edasi tuleb liikuda Game1.cs failis asuvasse Update() meetodisse. Nende koodiridade lisamisel tuleb jälgida teha, et need read paikneksid pärast koodirida float aeg = (float)gameTime.ElapsedGameTime.TotalSeconds; kuna muidu ei leita muutujat aeg.

Sinna meetodisse tuleb lisada järgnevad koodiread:

```
for (int i = 0; i < 10; i++)
{
   if (Kuulid[i].elus)
   {
      Kuulid[i].Update(aeg);
   }
```

```
}
```

See for tsükkel hoolitseb iga kuuli uuendamise eest eraldi.

Edasi tuleb luua ka igale kuulile BoundingSphere, ehk siis nähtamatu kera nende ümber, mille abil saaks hiljem kokkupuuteid kontrollida.

Selleks tuleb minna tsüklisse `for (int i = 0; i < Asteroidid.Length; i++).` Selles tsüklis tuleb üles leida koodrida:

`BoundingSphere asteroidKera = new BoundingSphere(Asteroidid[i].Positsioon, asteroid.Mudel.Meshes[0].BoundingSphere.Radius * 0.95f);`

ning kohe sinna alla lisada järgnevad koodiread:

```
    for (int j = 0; j < Kuulid.Length; j++)
{
    if (Kuulid[j].elus)
    {
        BoundingSphere kuulKera = new BoundingSphere(
         Kuulid[j].Positsioon,
         kuul.Mudel.Meshes[0].BoundingSphere.Radius);
        if (asteroidKera.Intersects(kuulKera))
        {
            Asteroidid[i].elus = false;
            Kuulid[j].elus = false;
            break;
        }
    }
}
```

Need koodiread peab lisama just sinna sellepärast, et eelnevalt on kõigile asteroididele ümber loodud kera, mille abil saab kontrollida kuulKera ja asteroidKera kokkupuuteid.

Järgmiseks tuleb anda kuulidele suund, kiirus ja positsioon. Selleks tuleb liikuda Game1.cs failis asuva Update() meetodi lõppu ning enne käsklust base.Update(gameTime); lisada järgnevad koodiread:

```
if (laev.elus && laev.laseb)
{
  for (int i = 0; i < 10; i++)
  {
    if (!Kuulid[i].elus)
    {
      Kuulid[i].Suund = laev.NurkMatrix.Forward;
      Kuulid[i].Kiirus = 2000.0f;
Kuulid[i].Positsioon = laev.Positsioon + (200 * Kuulid[i].Suund);
      Kuulid[i].elus = true;

      break;
    }
  }
}
```

Laev.laseb on muutuja, mida hetkel veel ei ole, kuid mis tuleb hiljem Laev.cs faili lisada. Laseb on väga sarnane elus muutujale ning omab ka samu väärtusi ehk false või true. Lisatud koodiridade esimesel real asuv if lause kontrollib, kas laev.elus ja laev.laseb

väärtused on true, kui jah liigutakse edasi järgnevate käskluste juurde. Järgneva for tsükliga käiakse läbi kõik kuulid ning antakse neile suund, kiirus ja positsioon. Siinkohal antakse kuulile positsioon laeva järgi kasutades laeva NurkMatrix-it.

Kindlasti peab tähele panema koodirida if (!Kuulid[i].elus), see koodirida kindlustab, et muudetakse vaid kuule, mille elus muutuja väärtus on false, kuna muidu muutuks kuuli asukoht kogu aeg sõltuvalt laevast. Seda aitab teha ! märk enne Kuulid[i].elus, mis annab XNA raamistikule teada, et Kuulid[i].elus on vastupidise väärtusega ehk siis false. Kuna Kuulid[i].elus võetakse algväärtuseks true.

Nüüd tuleb hoolitseda kuulide ekraanile joonistamise eest. Selleks tuleb minna Game1.cs failis asuvasse Draw() meetodisse. Sinna meetodisse tuleb lisada järgnevad koodiread:

```
 for (int i = 0; i < 10; i++)
{
   if (Kuulid[i].elus)
   {
      Matrix KuulMuutmiseMatrix =
       Matrix.CreateTranslation(Kuulid[i].Positsioon);

        JoonistaMudel(kuul.Mudel, KuulMuutmiseMatrix,
       kuul.Muutmine);
   }
}
```

Siin saadetakse kõigi kuulide, mille elus väärtus on true, informatsioon JoonistaMudel() meetodile, mis nad ekraanile joonistab.

Järgmiseks tuleb minna Laev.cs faili alla ja luua nupp, mille puhul laev kuule tulistama hakkaks. Esiteks tuleb Laev.cs failis asuvasse Laev() klassi lisada üks uus koodirida:

```
private static KeyboardState eelnevnupp;
public bool laseb;
```

Seda koodirida on vaja selleks, et kasutaja saaks lasta kuule ühekaupa, kuna muidu lastakse kõik kuulid korraga välja.

Edasi tuleb liikuda Laev.cs failis asuvasse Update() meetodisse. Sinna meetodisse tuleb lisada jägnevad koodiread:

```
        laseb = false;
    if (nupp.IsKeyDown(Keys.Space) && eelnevnupp.IsKeyUp(Keys.Space))
    {
       laseb = true;
     }
    eelnevnupp = nupp;
```

Kõigepealt antakse muutujale laseb väärtuse false. Siis kontrollitakse, kas nupp, mida vajutatakse, on Space ning kas nupp, mis lahti lasti, on ka Space. Kui kasutaja vajutab nupule Space ning laseb ka seejärel Space lahti, muudetakse muutuja laseb väärtus true-ks, mille peale kuulile antakse vajalikud väärtused ning joonistatakse see ekraanile. Lõpuks antakse muutujale eelnevnupp sama väärtus, mis on muutujal nupp. Sellega kindlustatakse, et välja lastakse vaid üks kuul. Vastasel juhul jääks nupp ja eelnevnupp muutujate väärtused sellisteks, mida if lause rakendamiseks vaja, ja muutuja laseb väärtus jääks senikaua true-ks, kuni see enne if lauset ümber muudetakse.

Kui kõik see on tehtud, tuleb klikkida rohelisele Start Debugging nupule, mis asetseb üleval tööriista ribal või vajutada klaviatuuril klahvile F5. Nüüd peaksid asteroidid pildilt kaduma, kui nad kuuliga pihta on saanud.

Head mängu edasiarendamist !