Tallinna Ülikool

Digitehnoloogiate instituut

Mängumootori Unity 3D õppematerjal

Seminaritöö

Autor: Siim Suu

Juhendaja: Jaagup Kippar

Autor: .... “ .... “2016

Juhendaja: .... “ .... “2016

Instituudi direktor: .... “ .... “2016

Tallinn 2016

Autorideklaratsioon

Kinnitan, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem

kaitsmisele esitatud.

....

(kuupäev) (allkiri)

Sisukord

Sissejuhatus ... 5
1. Unity 3D üldine tutvustus ... 6
2. Kasutajaliides ... 8
2.1 Scene view ... 8
2.2 Game view ... 9
2.3 Project browser ... 10
2.4 Hierarchy ... 10
2.5 Inspector ... 11
Ülesanne 1. ... 12
3 Graafika ja heli ... 13
3.1 Mesh ... 13
3.2 Tekstuurid ... 13
3.3 Materjalid ... 14
3.4 Kaamerad ... 14
3.5 Valgus ... 15
3.6 Heli ... 16
Ülesanne 2. ... 17
4. Füüsika ... 18
4.1 Colliders ... 18
4.2 Rigidbodies ... 18
4.3 Füüsilised materjalid ... 19
4.4 Joints ... 19
4.5 Raycasting ... 20
Ülesanne 3. ... 20
5 Skriptimine ... 21
5.1 Awake ja Start ... 21
5.2 Update ja FixedUpdate ... 22
Ülesanne 4. ... 23
5.3 Komponendi lubamine ja keelustamine, mänguobjekti aktiveerimine ja mänguobjekti hävitamine ... 23
5.4 Klaviatuuri- ja hiireklahvide kasutamine ... 25
5.5 GetComponent ja klassid ... 26
5.6 Instantiate ... 28

6. Näidismängu loomine Unity 3d abil.....30
6.1 Maastiku loomine.....30
6.2 Tegelaskuju ja tema liikumine.....33
6.3 Mängu temaatika.....33
Kasutajate tagasiside.....36
Kokkuvõte.....37
Kasutatud kirjandus.....38

Sissejuhatus

Teadaolevalt pole mänge lihtne teha. Suuremate mängude valmistamiseks kulub suurtel firmadel aastaid. Muidugi ei pea kõik mängud olema nii mahukad ning inimesi võib ka lihtsa idee abil lõbustada. Seega mängu tegemise juures on alati kõige tähtsam idee. Mõningad aastad tagasi valmistati mäng nimega *Drag Race,* mis oli teostuse poolest väga lihtsakoeline mäng. Mängul polnud palju võimalusi ning ka graafika oli minimaalne, kuid siiski tõmmati seda mängu miljonite kaupa ning ka tulu oli meeletu. Too mängu idee oli aga positiivne: sõida teistega võidu ning teeni punkte ja saad paremaid autosid osta. Ideed oli varem kasutatud üksikmängija puhul, kuid nüüd sai mängida koos teistega ning autosid võrrelda ning seda kõike sai teha läbi telefoni, mis muutis mängu mängimise väga mugavaks. Järelikult piisab heast ideest ning ka parajal määral oskustest ning ajast, et valmistada konkurentsivõimeline mäng, kuid ei pea omama suurt meeskonda.

Käesoleva töö probleemiks on eestikeelse õpetuse puudumine *Unity 3d'ga* mängude valmistamiseks. *Unity* enda leheküljel on olemas väga hea õpetus, mis on inglisekeelne, kuid selle läbitöötamiseks on vaja kuid. Teha on üritatud võimalikult kompaktne ning üsnagi lihtne õpetus, mida saab eesti keeles lugeda.

Seminaritöö eesmärgiks on tutvustada *Unity 3d* mängumootorit ja tema võimalusi.

Eesmärgi saavutamiseks on seatud järgmised ülesanded:

- Tutvustada *Unity 3d* üldist ülesehitust;
- Valmistada üks lihtsam mäng *Unity 3d* programmi näitel.

Loodud õppematerjalis ei anta ülevaadet kõikidest *Unity 3d* mängumootori võimalustest ega eesmärgiks ei ole luua ka täielikult valmis rakendus, vaid pigem anda ülevaade olulisematest *Unity* võimalustest ning sellest, kuidas nende kasutamine aitab rakendusi luua. Seminaritöö lõpus luuakse ka näidisrakendus.

Koostatud õppematerjal on põhineb *Unity 3d* mängumootori 4.3 versioonil. Õppematerjali läbimise eelduseks on: installeeritud *Unity 3d* mängumootor, algsed teadmised C# programeerimiskeeles ning algsed teadmised modelleerimise valdkonnas.

1. Unity 3D üldine tutvustus

Enamik mängumootoreid pakuvad võimalust, et kergendada mängu arendamist nagu: graafika, heli, füüsika ja AI. Füüsika elementi võiks pidada kõige tähtsamaks, sest tavaprogrameerijad pole füüsikaga nii “Sina ” peal, kui näiteks graafika või siis AI valmistamises. Mängumootoreid kutsutakse ka vahel “vahevaraks”, sest äri tähenduses , pakuvad nad paindlikku ja korduvkasutatavat tarkvara platvormi, mis pakub kõiki põhifunktsioone, et arendada mängu ning samal ajal alandada maksumust, keerukust ning ajakulu. Kõik need tegurid on tänasel turul väga tähtsad tänu tihedale konkurentsile mängutööstuses.

Mängumootorite eesmärgiks on hoida kokku programmeerija aega ja ressursse. Mängumootor on põhimõtteliselt kogum koodidest, mudelitest ja muudest rakendustest, mida mängu programmeerijal võib arendamise protsessis vaja minna. On olemas ka mängumootoreid, mis on kombineeritud kokku mitmetest teistest mängumootorite komponentidest. Selliseid mootoreid kutsutakse kohandatud mängumootoriteks. Üheks selliseks on näiteks RenerWare.

Unity 3d kutsutakse ka enamasti lihtsalt Unityks. Mängumootor pole iseenesest väga uus, kuid ta on uustulnuk mängumootorite arutelus. Väga odav lahendus mängu arenedajale , sest Unity litsents on ühekordne ning ükskõik kui suureks mängu kasum ulatub või mitu mängu sa valmistad, siis rohkem juurde maksma ei pea . Nüüdseks on ta täiesti konkurentsivõimeliseks saanud AAA graafikaga ning tema koostöös Autodeskiga, võib Unityga suuremahulisi projekte valmistada. Tema suureks miinuseks on tööriistade vähesus. Unity on mitmeplatvormiline ehk siis saab mänge teha nii mobiilidele, eri operatsioonisüsteemidele jne. Unity programmeerimskeelteks on on C#, javascript/unityscript ja Boo. Hetkel on mängumootori viimane versioon 4.3.4 , mis siis toetab iOS, Android, Windows, OS X, Linux, veebi lehekülgi, Flash, PlayStation 3, Xbox 360, and Wii U (Brodkin, 2013).

Unity'ga on väga lihtne ja mugav jälgida oma mängu arendusprotsessi. Vaid ühe nupuvajutusega saab siseneda mängimise režiimi ja vaadata kuidas lõpptulemus välja näeb. Mängu saab peatada ja muuta erinevaid väärtusi, muutujaid ja skriptijuppe. Mängu saab kaaderhaaval läbi käia ja vigu otsida. Võimalik on analüüsida igat aspekti- näiteks, milline mängu osa mõjutab kõige enam jõudlust. Sellisel viisil on kiiresti võimalik leida mängus olevaid kitsaskohti ning neid varakult eemaldada või parendada.

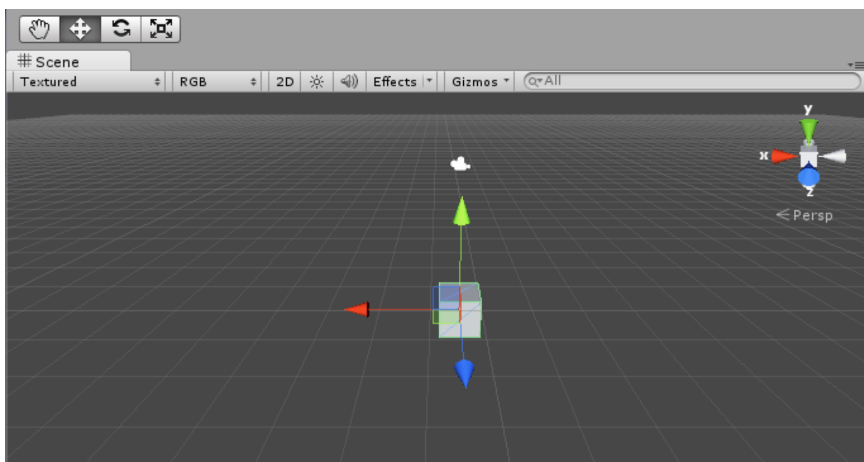
Õnneks on *Unity'l* olemas täiesti korralik tasuta versioon. Leheküljelt http://unity3d.com/get-unity/download/archive saab tõmmata *Unity 4.3* installiatsiooni faili. *Unity* installiatsiooni osa on kerge. Ei ole üleliigseid küsimusi ega pea andmebaase lisama. Alguses küsitakse ainult milliseid pakette tahad lisada. Neid võib lisada ka hiljem projektiga tegelemise käigus, kuid lihtsuse mõttes valige kõik paketid, et pärast segadust pole, kui on mingi asi puudu. Esimest korda programmi käima pannes tuleb kohe uus projekt ette, seega eraldi projekti pole vaja teha.

2. Kasutajaliides

Unity tuum koosneb viiest elemendist: *Scene view ,Hierarchy, Game view, Project panel, Inspector panel.* Lihtsama mängu valmistamiseks ei pea muud kasutama kui neid elemente ning lisaks ka skriptimise jaoks mõeldud tekstiredaktorit.

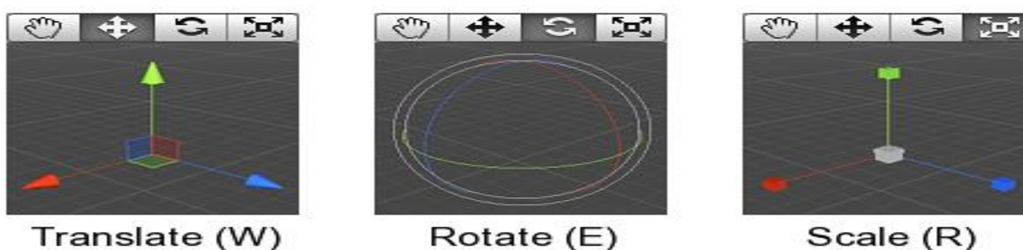
2.1 Scene view

Scene view on vaade, kus luuakse visuaalselt oma mäng. *Scene view'd* saab kasutada keskkonna, mängija, kaamera, vaenlase ning muude mänguobjektide selekteerimiseks ja positsioneerimiseks. Objektide manööverdamine ning käsitlemine on ühed tähtsamad funktsioonid *Unity's*, seega on tähtis seda teha võimalikult kiirelt. Tähtsamaid operatsioone saab teostada kasutades klahve klaviatuuril.



Joonis 1. Scene view akna vaade

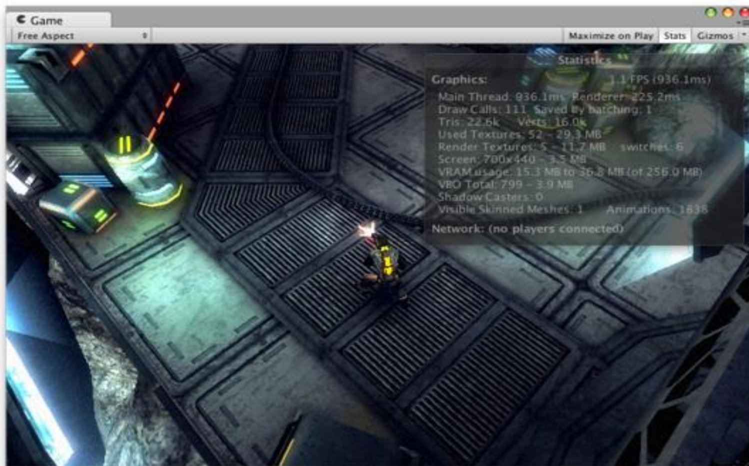
Mängu ehitamisel peab mitmeid objekte paigaldama. Paigaldamiseks saab kasutada objekti pöörlemist, suuruse muutmist ning asukoha muutmist. Objekti asukoha või asendi muutmiseks peab vasakut hiireklahvi all hoidma ning tõmbama oma valitud suunas.



Joonis 2. Erinevad positsioneerimis võimalused

2.2 Game view

Mängu vaade on esitatud kaamera või mitme kaameraga, mille asukohti saab muuta *Scene view's*. Mängu vaade näitab, milline on viimase muudatuse või lõpliku projekti tulemus ning kuidas kasutaja seda mängu näeb. Unity vaate üleval keskel on *Play Mode*, kus saad mängu käivitada ning vajadusel pausi peale panna. Uuesti vajutades, kuid need muudatused on ajutised. See on lihtsalt kiiremaks testimiseks vajalik.



Joonis 3. Game view akna vaade.

Game view üleval on juhtriba, mis aitab vajadusel muuta vaadet ning testida oma mängu erinevates olukordades.

Joonis 4. Game view juhtriba.

Kõige vasakpoolsem valik annab võimaluse muuta monitori kuvasuhet. See aitab testida, kuidas mäng näeb välja erinevate resolutsioonide puhul.

Järgmine nupp vasakult on lüliti. Kui see on sisse lülitatud ning käivitad mängu, siis mäng jookseb tervel ekraanil mitte ainult mängu vaates.

Teine nupp paremalt on samuti lüliti. Kui see on sisse lülitatud, siis näeb graafika ning võrgu jõudlust, nähtu põhjal saab testida erinevaid funktsioone koodis või muuta füüsika elemente *Inspectori´s* ning testida, millise jõudluse võidu või kaotuse see tõi.

Kõige parempoolsem valik on lüliti, mille sisse lülitades kõik *Gizmo'd* ehk füüsikalised elemendid, mis on nähtavad ka *Scene view's*, on nähtavad ka *Game view´s*. *Gizmo* nupul on ka hüpikaken, mis näitab erinevaid komponente, mida mängus kasutatakse (Unity Tehnologies, 2014).

2.3 Project browser

Selles vaates pääseb ligi varadele, mis lihtsustavad projekti tööd.

Joonis 5. Project browser.

Vasakpoolne paneel näitab projekti kaustade struktuuri. Vajutades mingi kausta peale, avaneb paremale paneeli selle kausta sisu. Kausta sisus on võimalus näha üksikuid *assets'eid* või siis uusi kaustasid. *Asset'siks* võib olla ükskõik, milline objekt, tekstuur või kujund ehk siis varad, mida saab mängu valmistamisel kasutada. Üksikuid *assets´eid* saab kasutada lohistades need *Hierarchy* paneelile. Enim kasutust leidvad *assets´id* saab asetada ka lemmikute alla. See vähendab otsimise aega. Sinna saab ka oma otsingute tulemused salvestada.

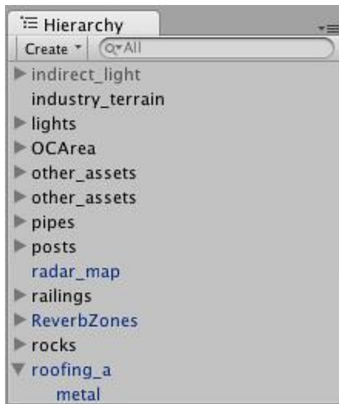
Joonis 6. Project Browser'i üleval olev juhtriba.

Kõige vasakpoolsem valik annab võimaluse lisada vastavasse kausta, mis on hetkel selekteeritud, uue *assets´i* või alamkataloogi. Sellest paremal pool on otsinguriba, mis lubab sul otsida ükskõik, mis *assets´it* projektist. Otsingut saab mugavamaks teha paremal pool olevate ikoonide abil. Tänu nendele saab konkreetselt otsida kas ainult mudeleid või skripte või midagi muud. Kõige parempoolne nupp salvestab sinu viimase otsingu lemmikute alla (Unity Tehnologies, 2014).

2.4 Hierarchy

Hierarchy sisaldab iga mänguobjekti S*cene view's*. Mõningad mänguobjektid on võetud otse, *assets´itest,* kuid enamus on tehtud kokkupandavatest objektidest. Võid valida objekti ning lohistada selle teise objekti peale, tehes esimesest objektist alamobjekti. Alamobjekt pärib oma liikumise ja pöörlemise vanemobjektist. Vajadusel vanemobjekti kolmnurga peale vajutades

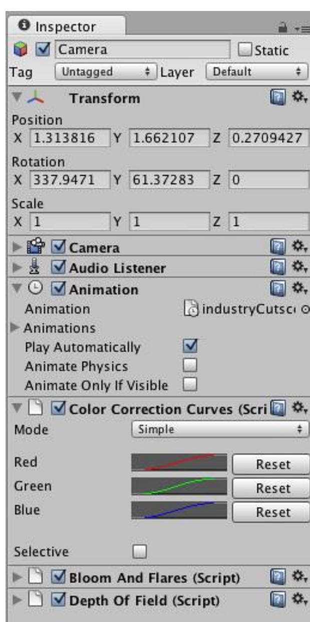
saab tema alamobjektid ära peita. Kustutades objekti *Scene view* vaates kustub objekt ka automaatselt *Hiearchy* vaatest.



Joonis 7. Hiearchy akna vaade.

2.5 Inspector

Mängud on tehtud alati mitmetest mängu objektidest, mis koosnevad omakorda skriptidest, helidest või siis graafika elementidest. *Inspector* kuvab detailse informatsiooni valitud mängu objekti kohta ning kaasab kõik komponendid ja nende omadused selle objekti kohta. Igat omadust, mis on nähtav, saab koheselt muuta. Isegi skripti enda muutujaid saab muuta ilma, et peaks koodis neid muutma. Testimisel on see väga kasulik, sest mängu jooksutamise ajal saab muutujaid muuta vastavalt soovile. Samuti kui skriptis defineerida avalik muutuja, näiteks objekti tüüp, siis saab seda lisada ja kustutada *Inspector'is* vastavalt vajadusele (Unity Tehnologies, 2014).



Joonis 8. Inspector akna vaade.

Ülesanne 1.

Luua objekt. Alustuseks sobib tavalise kasti loomine*(GameObject->Create Other->Cube*). Object ilmub loomisel ka automaatselt *Hierarchy'sse.* Alustuseks tuleks selgeks saada kuidas objekti rotatsioonid ning suuruse muutmised käivad *Scene view* abil. Kui olete mõne paketi programmi käivitamisel või siis uue projekti alustades lisanud, siis võib proovida *project browseri*'*st* neid lisada. Lisaks tuleb *Inspector* vaatest muuta objekti suurust ja rotatsiooni pärast teatud objektile vajutamist.

3 Graafika ja heli

3.1 Mesh

Scene view on enamjaolt Unitys loodud 3d *mesh'idest* ja hulknurksetest *mesh'idest*. *Mesh'i* võib kujutada, kui suvalist ruumilist kujundit. Hulknurksed *mesh'id* võivad olla väga lihtsad, kuid ka väga keerukad. Lihtsamad on uksed, milleks on tavaline ristkülik teatud mustriga ning raskemad on 3d karakterid või siis keskkonna geomeetria.

Hulknurksed *mesh'id* on tehtud väga kergetest elementidest, nimelt punktidest ja joontest.

Kahjuks ei saa keerukamaid *mesh'e* Unitys teha, vaid peab importima neid teistest programmidest, mis saavad *mesh'ide* tegemisega hakkama (Baumgart, 1975).

Vertex-Vertex Meshes (VV)

Vertex List

v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,1	v4 v5 v6 v7
v9	.5,.5,0	v0 v1 v2 v3

Joonis 9. Kuubikust mesh.

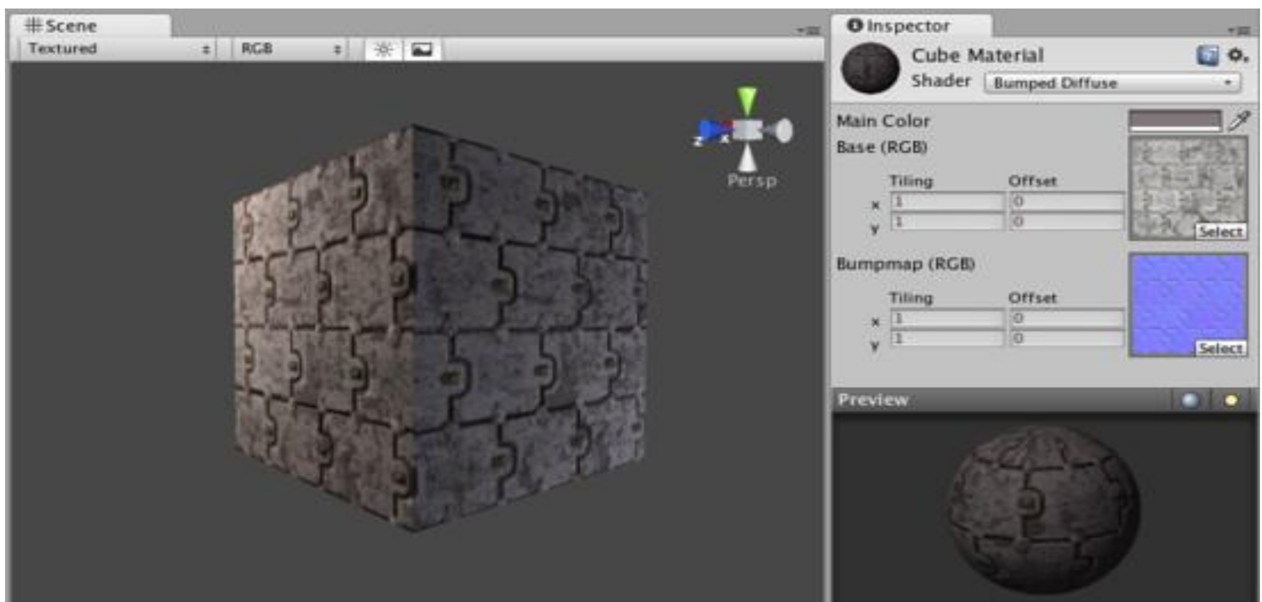
3.2 Tekstuurid

Tekstuur on tavaline pilt. Unity kasutab tekstuure erineval moel. Enamasti lisatakse tekstuur materjalile, et saavutada ilusam raamvõrk. Tekstuuridega saab genereerida ka mängu kaarte või siis lihtsalt maastikku ilusamaks teha. Üldjuhul oleks hea, kui tekstuurid on valmistatud mõne fototöötlusprogrammiga nagu seda näiteks on *Photoshop*. Seal saab pilti ehtida kihtide kaupa. *Unity's* saab pärast kihte vähemaks võtta ja vajadusel lisada uuesti, kui just mäng juba käivitatud pole. *Unity's* peaksid hoidma tekstuure *Assets* kaustas, kus on eraldi välja toodud *Textures*.

3.3 Materjalid

Materjalid on varad, mis kontrollivad mängu objekti välimust. Materjali saad luua *Projects* paneelil vajutades *create* ja seejärel *material.* Tihtipeale arvatakse, et materjal ning tekstuur on sama tähendsuega, kuid tegelikult tekstuur on lihtsalt pilt ning materjal näitab, kuidas pilti kuvatakse.

Suur tähtsus materjalil on varjude ning toonide muutmine, et genereeritaks tahetud materjali kujutlelm.



Joonis 10. Materjali näide.

3.4 Kaamerad

Kaamerad on *Unity's* üks vajalikumaid komponente*(GameObject-> Create Other -> Camera).* Kaamera saadab vaatajani *Scene view'i* toimuva sisu. Igal *Scene view'l* peab olema vähemalt üks kaamera, millega ta vaatajale pilti edastab, vastasel juhul poleks midagi näidata. Uut *Scene* tehes on üks objekt alati temaga kaasas. Selleks on põhikaamera.

Kaamera on nagu iga objekt, millel saab muuta asukohta, skripte juurde lisada ja palju muud. Kui mängu tegelane kõnnib või lendab kaardi peal ringi, siis on erinevaid võimalusi, kuidas kaameraid asetada.

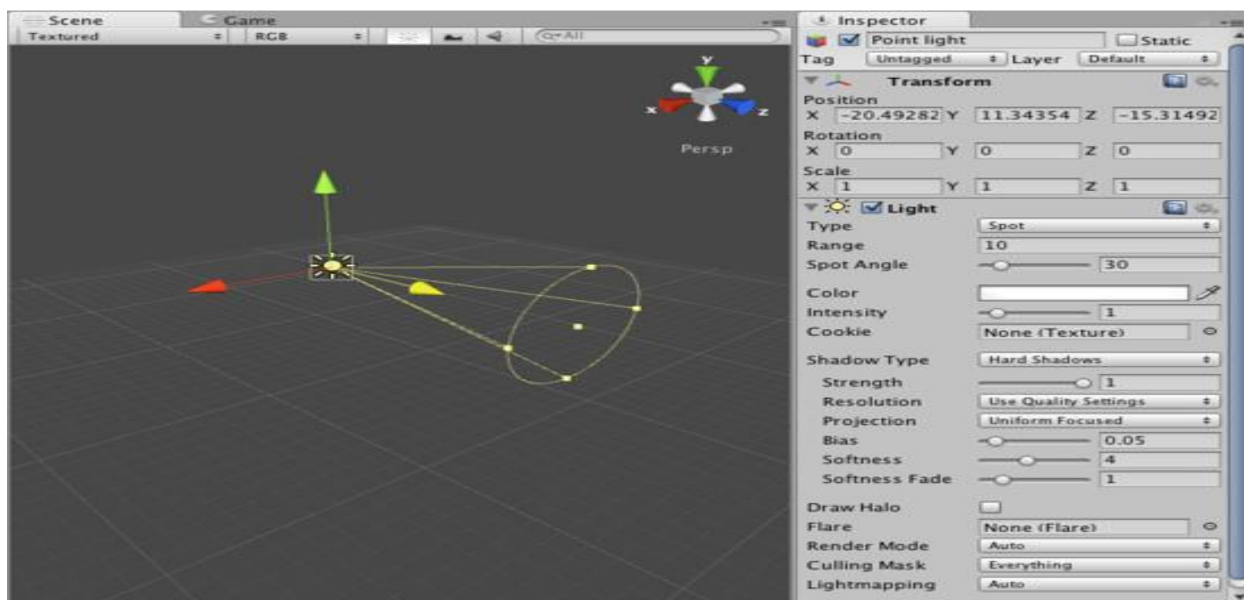
- Staatiline kaamera, mis jälgib kogu maastikku. Saab panna ka mitu kaamerat kaardile ning teatud olukordades lülitada üle vastavale kaamerale. Üldjuhul mäng ise valib kaamera kauguse järgi õige kaamera.
- Kaamera ühendatakse mänguobjektiga, üldjuhul siis mängu tegelasega ning kaamera jälgib pidevalt mängutegelast ja ta ümbrust. Kõik oleneb kui kaugel kaamera mänguobjektist on, siis seda suuremalt või väiksemalt kõik toimub.
- Kaamera ühendatakse mänguobjektiga, kuid sellel juhul kinnitatakse kaamera mänguobjekti ette ning saadakse reaalne vaade, mida mängijad näevad.
  (Unity Tehnologies, 2014)

3.5 Valgus

Unity vajab valgust, et kõik nähtav oleks*(GameObject-> Create Other -> Directional Light)*. Ilma tuledeta poleks kaameratele mitte midagi nähtav. Unitys on kahte sorti tulesid: dünaamiline tuli ning *baked* tuli. Dünaamiline tuli on meil elektripirni eest. *Baked* tuli võib kogu kaardi ühtlaselt ära valgustada. Dünaamilisi valgusobjekte saab samamoodi liigutada nagu teisigi mänguobjekte ning *Inspector'is* tema andmeid muuta.

Tulel on neli erinevat tüüpi:

- Täppvalgus, mis käitub nagu taskulamp, kus valgus on suunatud vaid ühte kohta. Valguse ulatus ja heledus sõltub tule kaugusest.
- Suunatud valgus käitub nagu päike, kus ainult valguse suund on oluline. Valguse asukoht ei määra aga midagi.
- Punktvalgus käitub nagu elektripirn, kus valguse keeramine ei määra mingit rolli, sest valgus paistab kõikjale ühtemoodi.
- Alaline valgus töötab ainult küpsetatud tule juures. Ta käitub nagu päeval aknast tulev valgus, kus on kas oksad või näiteks majad varjudena ees.
  Igal objektil peab olema ka vari, kui valgus talle peale paistab. *Unity's* on kahte sorti varjusid: tugevad ning nõrgad varjud, kus tugevamad võtavad vähem resursse ning paistavad tugevamana ning ebarealistlikumana (Unity Tehnologies, 2014).



Joonis 11. Tule sätted Inspector'is.

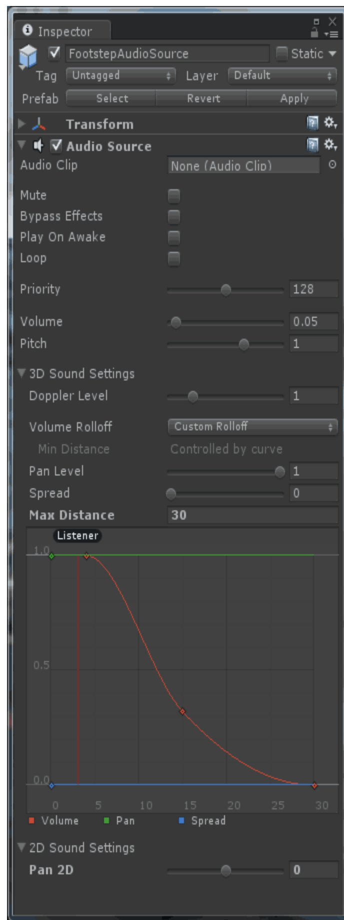
3.6 Heli

Mänguobjektile saad lisada komponendi, milleks on *audio listener* *(Component-> Audio -> Audio listener)*. See on võrdväärne inimese kõrvadega. Ta võtab vastu või kuulab audio allikaid, mis on mängumaailmas. *Audio listener* on algselt ühendatud kaamera külge, kuid võib mistahes mis mänguobjektiga ühendada. Tihti on ta ühendatud ka mängija karakteri külge.

Heli allikad on komponendid, mis mängivad heli. Nad on nagu kõlarid meie maailmas. Igale heli allikale peab ühendatud olema audio klipp. Heli allikaid saab samamoodi ühendada iga mänguobjekti juurde. Olenevalt heli kuulaja asukohast, kuuleb mängu mängija erinevalt heli allikaid. Mida kaugemal ta heliallikast on, seda vaiksemini ta seda kuuleb. Heli võib töötada pidevalt, kuid selle võib aktiveerida alles siis, kui keegi teatud mänguobjektile läheneb. Seda tehakse üldjuhul skripti abil. Helisid saab ka tähtsuse alusel järjekorrastada. Tähtsamad helid on meil pidevalt kuuldaval ning vähemtähtsamaid kuuleb vaid väga lähedale minnes. Heli allikaid saab ka suuremaks teha, nagu enamusi mänguobjekte, mida suurem see on, seda valjemalt heli kostub.

Heli klippe saab samuti Unitys muuta. On võimalus peale suruda 3D heli või mono. Saab muuta ka heli kvaliteeti halvemaks. Kui klipi kvaliteet on liiga hea, siis on see mõttetult mahukas. 3D helide jaoks on olemas ka täpsustavad sätted, näiteks auto möödasõidu korral kuuled sa häält

juba kaugelt. Saab valida kas logaritmilise või lineaarse valiku vahel. On olemas ka manuaalne variant, kus saab ise joonist juhtida ning heli kontrollida läheneva või kaugeneva objekti puhul.



Joonis 12. Heli sätted Inspector'is.

Ülesanne 2.

Lisada *Unity'sse* tekstuur lohistades see *Assets* kausta. Teha uus materjal ja samuti ka tekstuur. Lisada *scene view'sse* ka eri tüüpe valgusi nind vaadelda, mis vahe neil on. Lisada loodud materjal ning tekstuur objektile, mis on eelnevas ülesandes juba valmis tehtud. Võimaluse korral lisada ka objekti külge helifail.

4. Füüsika

4.1 Colliders

Colliders on komponendid *(Component->Physics->Box Collider*), mis lubavad mänguobjektil, mille küljes nad paiknevad, reageerida teiste *Collider'itega* tingimusel, et kummalgi mänguobjektil on *rigidbody* (vt järgmine peatükk) kinnitatud. *Colliders* tulevad väga erinevate kujunditena ning tüüpidena ja on ära märgitud *Scene view's* rohelise kontuuriga. Nad on järgnevate kujunditena: kera, kapsel ja kast. Keerukamate kujundite jaoks on kaks järgnevat võimalust. On võimalus panna detailsem kokkupõrgatav kokku lihtsamatest kujunditest või määrata mänguobjektile *mesh collider*. Viimasel variandil on *Collider,* täpselt samade mõõtmetega, mis *Mesh*. Põhjus, miks *mesh collider'it* mitte kasutada, on detailsus. Ta on liiga detailne ning mõjub kõvasti mängu jõudlusele.

Kui mängumootoris toimuvad kokkupõrked, siis üks kokkupõrgatav lööb teist kokkupõrgatavat ning sündmus nimega *OnCollisionEnter* kutsutakse välja. See kutsutakse välja skriptis. Kui pole skripti midagi kirjutatud, siis midagi ei juhtugi. On olemas ka sündmused nimega *OnCollisionExit* ning *OnCollisionStay*. Esimesel juhul funktsioon aktiveerub, kui kokkupõrget enam ei toimu ning teisel juhul funktsioon kestab nii kaua, kui kaks kokkupõrgatavat on ikka veel üksteisega kontaktis.

Colliders sätete alt saab muuta *trigger* seisundit. Kui seisund on maas, siis kõik toimib nii nagu eelnevalt, kuid kui see sisse lülitada, siis esimene *Colliders* lendab teisest *Colliders'ist* läbi, kuid kokkupõrget saab siiski määrata *OnTriggerEnter*, *OnTriggerExit* ja *OntriggerStay* abil. Enamasti kasutatakse seda nähtamatute objektidega kokkupõrke tuvastamiseks.

4.2 Rigidbodies

Valmistades mänguobjekti, mis liiguks, peab kindel olema, et sellele objektile on kinnitatud *rigidbody*. *Rigidbodies (Component->Physics->Rigidbody)* on komponendid, mis lubavad mänguobjektil olla mõjutatud füüsikast. Näiteks hakkab objektile gravitatsioon mõjuma ning tulevad mängu füüsilised muutujad nagu näiteks mass, pidurdamine ja kiirus. Pannes kasti õhku rippuma ilma *rigidbody* lisamata, jääb ta õhku rippuma.Vastasel juhul aga kukub, kuni mingi

objekt talle vastu tuleb. Kui lisada mänguobjektile *rigibody* komponent, siis kutsutakse seda mänguobjekti *rigidbody* objektiks. Kindlasti peab olema ka ühendatud *collider*, et tuvastada kokkupõrkeid.

Rigidbody sätete all on palju tähsaid komponente:

- Mass - Lisada või alandada objekti massi. See määrab kui kõvasti kokkupõrkel objektid põrkuvad.
- Drag - Kui kiirelt objekt aeglustub ilma vahele segamata.
- Use Gravity – Kas mänguobjektile mõjub gravitatsioon.
- Is Kinematic – Teised objektid ei mõjuta selle *rigidbody* objekti liikumist.
  (Unity Tehnologies, 2014)

4.3 Füüsilised materjalid

Füüsilised materjalid on materjalid, mis ei mõjuta välimust mänguobjektil,vaid mõjutab objekti reageerimist füüsikamootoriga. Võib tuua näite korvpalli ja keeglipalli näol. Tänu nende pinnale ei ole neil ainult massi erinevus, vaid ka põrkamises on erinevus. Erinevate pindade reageerimist teiste pindadega kontrollitakse friktsioonidega ning põrkamise tugevusega.

Füüsilist materjali saad luua *Project's* paneelil vajutades *create* ja seejärel *Physic material.* Sätted nagu ikka saab muuta läbi *Inspectori.*

- *Dynamic Friction* – friktsioon, mis näitab, kui suur on hõõrdumise tugevus, kui objekt liigub.
- *Static Friction* - friktsioon, mis näitab, kui palju jõudu on vaja, et objekt liikuma hakkaks.
- *Bounciness* - näitab kui põrkav on objekt.

4.4 Joints

Joint on liiges *(Component->Physics-> Fixed joint)*, mida saab ühendada mänguobjekti külge, millega saab ühendada üksteise külge teisi *joint'e.* Ta on justkui keti lülide ühenduslüli. Põhiline eesmärk ongi tal kettide, nööride, vedrude või millegi sarnase valmistamine. On kahte erinevat sorti *Joint'e: fixed joint* ja *spring joint.*

Kui lukustada *fixed joint* mänguobjekti külge, siis on see *joint* üldjuhul paigal, kui just ei määrata talle piisavalt jõudu, et ta oma positsioonilt ära lendaks. Enamasti pannakse piisavaks jõuks lõpmatus, mis tähendab, et see mänguobjekt jääb sinna püsima ükskõik mis olukorras. *Fixed joint* on seega objekt, kuhu kett kinnitatakse.

Spring joint'id on aga liikuvad joint'id, mis üritavad iga hinna eest eelmise objekti külge klammerduda. Kinnitatud ots on kindlalt siis eelmise objekti küljes, kuid vastaspoolne ots üritab liikuta sinna, kus jõud ta suunanud on ja seega tekibki keti või vedru effekt (Unity Tehnologies, 2014).

4.5 Raycasting

Raycasting on tulistamise protsess, mis laseb teatud punktist välja nähtamatu kiire valitud suunas, et kindlaks teha, kas mingisugused *collider'id* on tee peal ees või mitte. Selleks on vaja kahte sorti koordinaate. Esiteks kiire algus ning teiseks kiire suund. Kiire algus seotakse üldjuhul mingi mänguobjektiga, selleks on siis mängija ise või niisama kast. Süntaks näeb koodis välja selline: *Physics.Raycast (Vector origin, Vector3 direction, RaycastHit hitInfo, float distance, int LayerMask);* . Tänu *hitInfo'le* saame teada, milline mänguobjekt talle tee peale ette on jäänud. Sünataksis kaks viimast muutujat pole kohustuslikud. *Distance* määrab ära kiire pikkuse. Ilma kindlat pikkust panemata on kiir lõpmatult pikk. *LayerMask* on number, mis iseloomustab objektide gruppi, kuhu on valitud teatud objektid, mida ignoreeritakse kiire poolt.

Ülesanne 3.

Lisada juba olemaolevale objektile rigidbody ning minnes *Play mode*, siis on näha, kuidas gravitatsioon mõjub objektile. Võib luua ka teise objekti ja lisada see olemasoleva objektist alla poole. Lisades mõlemale objektile *collider'i*, siis nad põrkuvad omavahel, mitte ei lähe üksteisest läbi.

5 Skriptimine

Unity's väljendub skript teatud objekti käitumist. Skriptimine ehk siis koodijupi kirjutamine. Skripti saab lisada *Unity's* samamoodi nagu teisi funktsioone mänguobjektile*(Component->Scripts-> New Behaviour Script)*. Nagu eespool mainitud on skriptide kirjutamiseks kolm eri keelt olemas: *C#, Boo* ja *Javascript*. Rohkem näited leiab internetist *C#* ja *Javascripti* kohta. Meie vaatleme skripte *C#* näitel ning ei hakka lihtsamat *C#* süntaksit seletama, eeldades, et lugeja neid juba oskab.

5.1 Awake ja Start

Awake ja *start* on kaks funktsiooni, mis kutsutakse välja automaatselt peale skripti laadimist. *Awake* funktsioon kutsutakse esimesena välja, isegi kui skripti komponendid pole sisse lülitatud. *Start* funktsioon kutsutakse välja koheselt, enne esimest *Update* funktsiooni. See tähendab seda, et koheselt mängu käivitades käivitub *Awake,* ning nii kaua ei toimu edasisi korraldusi kuni hetkeni, kui see skript välja kutsutakse ning käivitatkse *Start* funtksioon ning seejärel funktsioon *Update*, kui see olemas on. Mõlemat funktsiooni saab ainult korra kasutada.

```
using UnityEngine;
using System.Collections;

public class AwakeAndStart : MonoBehaviour
{
    void Awake ()
    {
        Debug.Log("Awake funktsioon kutsutakse");
    }
    void Start ()
    {
        Debug.Log("Start funktsioon kutsutakse");
```

```
    }
}
```

Koodinäide 1. Start ja Awake funktsiooni kasutamise näide.

5.2 Update ja FixedUpdate

Update on *Unity's* kõige sagedasemini kasutatavam funktsioon. Seda kutsutakse välja igas kaadris ning seda iga erineva skripti kohta. Kõik, mida on vaja regulaarselt käivitada või muuta, juhtub selles funktsioonis. Iga kaadri pikkus on eelmisest üldjuhul natuke erinev.

FixedUpdate on *Update* funktsiooniga sarnanse, kuid on mõningad sisse impordidud erinevused. Seda funktsiooni kutsutakse välja ühtlaste ajavahedega. *FixedUpdate* kutsutakse välja iga füüsikalise sammu juures. Näiteks kasutatkse seda füüsikaliste objektide muutmisel (*Rigidbody*).

```
using UnityEngine;
using System.Collections;

public class UpdateAndFixedUpdate : MonoBehaviour
{
    void FixedUpdate ()
    {
        Debug.Log("FixedUpdate aeg :" + Time.deltaTime);
    }

    void Update ()
    {
        Debug.Log("Update aeg :" + Time.deltaTime);
```

```
    }
}
```

Koodinäide 2. FixedUpdate ja Update funktsiooni erinevuse näide.

Rakendades üleval toodud näidet on näha, kuidas funktsioonil *Update* pole pea kunagi sama protsessi aeg. *FixedUpdate'el* on aga seevastu kogu aeg kaadri vahemik 0.02 sekundit.

Ülesanne 4.
Lisada objektile mõlemad eespool olevad skriptid, et veenduda funktsioonide korrektsuses. Iga skript käib ainult selle objekti kohta kuhu see on lisatud. Logmine toimub *Console* aknas.

5.3 Komponendi lubamine ja keelustamine, mänguobjekti aktiveerimine ja mänguobjekti hävitamine

Mänguobjektil on enamasti palju komponente või siis funktsioone lisatud. Atribuuti *enabled* abil saab komponente näiteks siis keelustada või lubada. Allpool toodud näites on ära näidatud, kuidas muutujaks võetakse mänguobjektiks tuli. Ning tühikklahviga klaviatuuril saab tuld kustutada või põlema panna olenedes eelmisest muutuja seisundist.

```
using UnityEngine;
using System.Collections;

public class EnableComponents : MonoBehaviour
{
    private Light myLight;
    void Start ()
    {
        myLight = GetComponent<Light>();
    }
```

```
    void Update ()
    {
        if(Input.GetKeyUp(KeyCode.Space))
        {
            myLight.enabled = !myLight.enabled;
        }
    }
}
```

Koodinäide 3. Tule aktiveerimise ja deaktiveerimise näide kasutades Enabled aktribuuti.

Mänguobjekti saab aktiveerida ning deaktiveerida kaheti. Esiteks võid sa visuaalselt deaktiveerida mänguobjekti *Inspector'i*s mängubjektil linnukese ära võtmisega või siis koodi kaudu. Alltoodud näites on näha, kuidas see käib skripti abil. Meeles tuleb pidada, et kui töötada hierarhiaga, siis deaktiveerides ülemobjekti, alamobjektid küll kaotavad oma nähtavuse, kuid nad on siiski veel aktiveeritud. *ActiveSelf* atribuudi abil saab skriptis ka kontrollida, kas see objekt on aktiivne või mitte ning atribuudi *activeInHierarchy* abil saab kontrollida, kas kogu hierarhia on aktiveeritud. Allpool on toodud näide mänguobjekti deaktiveerimisest.

```
using UnityEngine;
using System.Collections;

public class ActiveObjects : MonoBehaviour
{
    void Start ()
    {
        gameObject.SetActive(false);
    }
}
```

Koodinäide 4. Mänguobjekti deaktiveerimise näide.

Kui teatud mänguobjekti aga täielikult tahta hävitada hiearhiast, siis tuleb kasutada *Destroy* funktsiooni. Sellest lihtne näide. Hetkelises olukorras hävineb objekt, mille külge skript lisatud on.

```
using UnityEngine;
using System.Collections;

public class DestroyOther : MonoBehaviour
{
    public GameObject other;

    void Update ()
    {
        if(Input.GetKey(KeyCode.Space))
        {
            Destroy(other);
        }
    }
}
```

Koodinäide 5. Mänguobjekti hävitamise näide.

5.4 Klaviatuuri- ja hiireklahvide kasutamine

Unity's on *GetButton* ja *GetKey* klahvivajutust äratundvad funktsioonid. Esimest funktsiooni on hea kasutada siis, kui on kindel nupp olemas ning mängusiseselt neid muuta ei saa. *GetKey* on üldjuhul tunduvalt parem, sest siis saab mängija ise valida, mis klahv mis tegevust teeb. *GetKey* väärtusi saab muuta minnes *Edit-Project settings-Input* ning seal on sätted iga mängus kasutusel oleva tegevuse jaoks. *Getbutton* ja *GetKey* funktsiooniga saab Unity igas kaadris aru, kas klahvi hoitakse all või mitte ning teostab vastavaid tegevusi. *GetButtonDown* ja *GetKeyDown* funktsioonid aktiveeritakse ühel korral ja seda siis, kui klahv alla vajutatakse. Vastupidiselt töötavad *GetButtonUp* ja *GetKeyUp*, kus funktsioonid käivitatakse klahvi vabanemisel. Skripti näiteks sobib eelmine joonis, kus on kasutatud tühikklahvi aktiveerimist.

Hiirel on olemas funktsioon *OnMouseDown*, mis on sarnane klaviatuuri funktsioonidega, kuid erinevus on selles, et see töötab ka teatud mänguobjekti peale vajutades, kus on see skript ühendatud (Unity Tehnologies, 2014).

5.5 GetComponent ja klassid

Vahel on vaja ligi pääseda sama mänguobjekti skriptile või isegi teiste mänguobjekti skriptidele. Appi tuleb *GetComponent.* Tänu sellele funktsioonile, saad ligi kogu infole, mis vaja, kui ta pole privaatne.

Iga skript koosneb automaatselt juba klassist. Neid võib lisada nii nagu tavaliste kompilaatorite puhul, kuid ei saa välja kutsuda kogu projekti ulatuses teisi klasse, kui nad on teises skriptis. Tänu sellele ongi vaja funktsiooni *GetComponent.*

Alltoodud näites võetakse komponentide väärtus kahest eri skriptist ja väljastatakse need ning samuti määratakse kastile tema suurus. Skriptist nimega *AnotherScript* saadakse mängija punktisumma, milleks on 9001 ning skriptist nimega *YetAnotherScript* saadakse mängija surmad kokku liidetuna, milleks on 3.

Klasside näiteid ei hakka siinkohal tooma, sest need võiksid olla selged.

```
using UnityEngine;
using System.Collections;

public class UsingOtherComponents : MonoBehaviour
{
    public GameObject otherGameObject;
    private AnotherScript anotherScript;
    private YetAnotherScript yetAnotherScript;
    private BoxCollider boxCol;

    void Awake ()
    {
```

```
        anotherScript = GetComponent<AnotherScript>();
        yetAnotherScript = otherGameObject.GetComponent<YetAnotherScript>();
        boxCol = otherGameObject.GetComponent<BoxCollider>();
    }

    void Start ()
    {
        boxCol.size = new Vector3(3,3,3);
        Debug.Log("Mängija skoor on " + anotherScript.playerScore);
        Debug.Log("Mängija on surnud " + yetAnotherScript.numberOfPlayerDeaths + "
korda");
    }
}
```

Koodinäide 6. Põhiklass.

```
using UnityEngine;
using System.Collections;

public class AnotherScript : MonoBehaviour
{
    public int playerScore = 9001;}
```

Koodinäide 7. Esimene alamklass.

```
using UnityEngine;
using System.Collections;

public class YetAnotherScript : MonoBehaviour
{
    public int numberOfPlayerDeaths = 3;
}
```

Koodinäide 8. Teine alamklass.

5.6 Instantiate

Instantiate funktsiooni kasutatakse originaalmänguobjekti kopeerimiseks. Tänu sellele saab tekitada lõpmatult palju kloone originaalist ilma, et peaks kulutama palju resursse. Koheselt saab määrata uue klooni asukoha ning suuna. Tähele peab panema seda, et kloonides mingit mänguobjekti, kloonitakse ka kõik selle mänguobjekti alamobjektid. Kui kloonitav mänguobjekt pole aktiivne, siis pole ka uus kloon aktiivne ning seda ei kuvata. Näiteks tuleb see funktsioon kasuks relvast kuulide laskmisel, kus iga kuul on *prefab'i* koopia. Alltoodud näites väljastab skript kuuli klooni, kui mängija on vajutanud klahvi, mis ta on ise määranud tulistamiseks, mille alguseks on relva toru positsioon ning talle määratakse *Rigidbody* ning samuti jõud 5000 ühiku osas, mis seda kuuli edasi peaks tõukama (Unity Tehnologies, 2014)

.

```
using UnityEngine;
using System.Collections;

public class UsingInstantiate : MonoBehaviour
{
    public Rigidbody rocketPrefab;
    public Transform barrelEnd;



    void Update ()
    {
        if(Input.GetButtonDown("Fire1"))
        {
            Rigidbody rocketInstance;
            rocketInstance = Instantiate(rocketPrefab,
barrelEnd.position, barrelEnd.rotation) as Rigidbody;
            rocketInstance.AddForce(barrelEnd.forward * 5000);
        }
    }
}
```

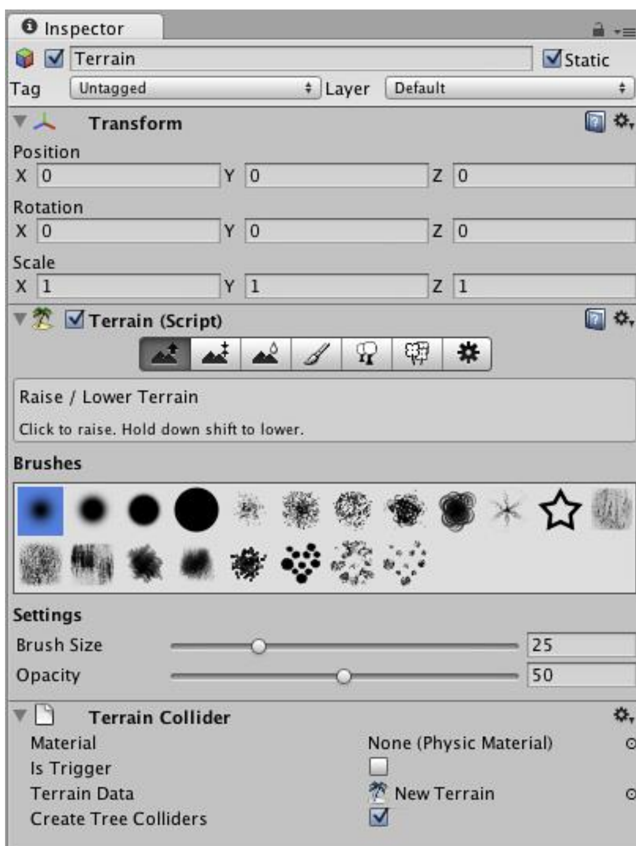
Koodinäide 9. Mänguobjekti hävitamise näide.

6. Näidismängu loomine Unity 3d abil

Järgnevas näites luuakse *Unity 3d* mängumootori võimalusi kasutades lihtne mäng, mis võimaldab kasutada eelnevat õpetust. Loodava mängu peategelaseks on mängija, kel on otsevaade metsale ning tal on võimalus seal ringi käia ning ta ülesandeks on otsida ning koguda nii palju relvi kui võimalik. Esmalt luuakse maastik ehk maailm, kus mängija hakkab tegutsema.

6.1 Maastiku loomine.

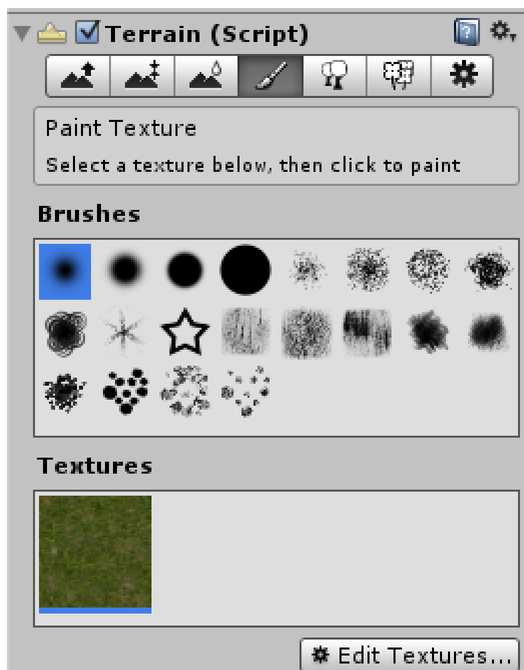
Alustuseks tuleb luua maastik, kuhu peale kõik mängukaart tuleb (*GameObject->Create Other->Terrain*). Siis ilmub suur lame ala, mida saab vastavalt muuta. *Inspector'i* alt saab muuta maaastiku andmeid ning välimust.



Joonis 13. Maastiku sätteid saab muuta Inspector'ist Terrain'i komponendi alt.

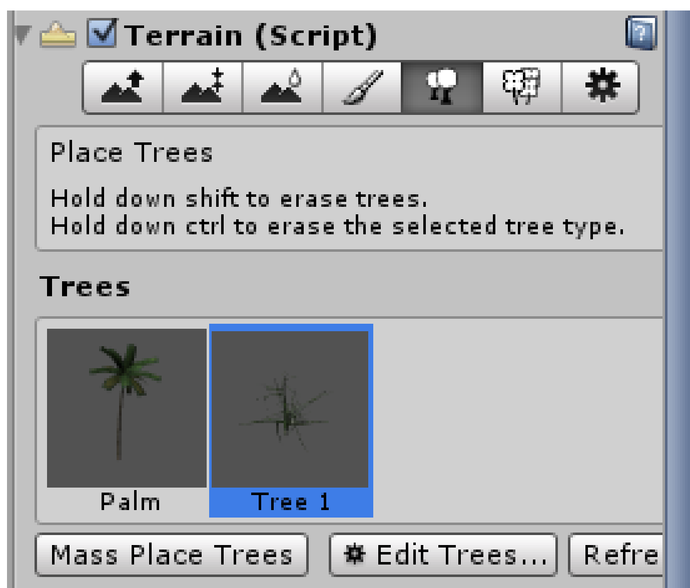
Vastavalt vajadusele saab *Terrain'i* madalamaks ning kõrgemaks teha sealt, kus soov on ning tänu sellele saab mägesid ning orgusid immiteerida. Orgudesse saab panna ka vett. Vett saab

lisada lohistades asukohaga *Assets->Standard Assets->Water(Basic)->Daylight Simple* mänguobjekti *Scene* aknas sinna, kus on näiteks org tehtud. Nüüd on olemas hallid mäed ja orud ning ka mõningane vesi, kuid on vaja kaarti realistlikumaks teha.



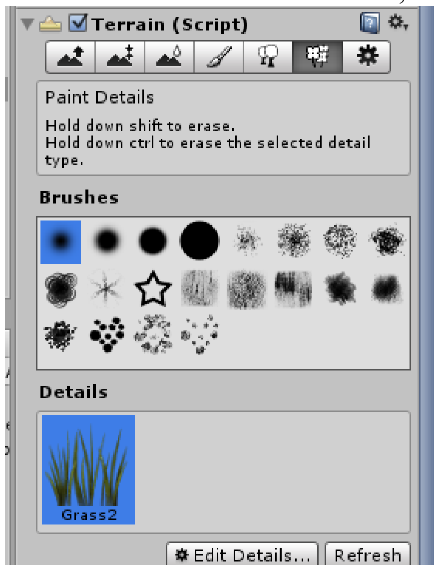
Joonis 14. Maastiku tekstuuri saab paigaldada erinevate pintslite abil.

Tähtis asi kaardi juures on maastiku tekstuur, mis võib jäljendada kalju pinda või tavalist rohtu. Praegu on valitud tavaline lihtne roheline rohtu meenutav tekstuur. Tore oleks maastikule lisada ka mõningaid puid, tänu millele, meenutab meie kaart nüüd džunglit. Puid saab ise modelleerida erinevate programmide abil, kuid *Unity's* on samuti see võimalus olemas, et puid ise teha.



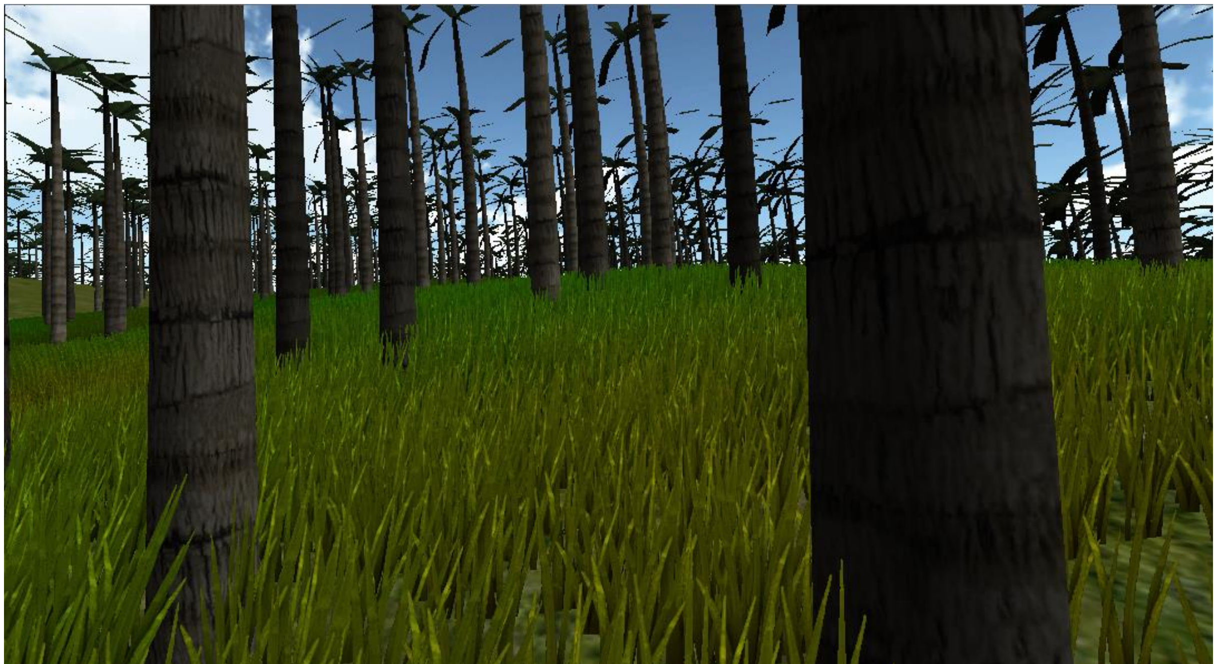
Joonis 15. Puid saab paigaldada massiliselt või ükshaaval

Nüüd on peaaegu korralik maastik olemas, kuid, et seda veel paremaks teha, võib lisada mitte ainult rohulaadse tekstuuri, vaid rohu või ükskõik millised taimed kasvama.



Joonis 16. Maastikule taimestiku lisamine.

Nüüd on maastik olemas, kuid taevasse vaadates pole seal midagi. Taevast saab lisada minnes *Edit->Render Settings* ning seal sättedes on *Skybox Material*. Saab valida ükskõik millise materjali taevasse.



Joonis 17. Valminud maastik.

Hetkel on valminud hõreda metsaga maastik, kus valdavateks puudeks on palmipuud. Maastikul on ka palju muid sätteid, mida saab vastavalt vajadusele muuta. Näiteks saab muuta tuule tugevust ja palju muud.

6.2 Tegelaskuju ja tema liikumine

Tegelaskuju valmistamine on *Unity's* väga lihtsaks tehtud. Kasutaja jaoks on kõik alguseks vajalik töö ära tehtud. Tuleb vaid valida *Standard Assets* alt *Character Controllers* komponent ning lisada see kaardile. Tegelaskujul on olemas juba vajalikud skriptid:

- *Mouse Look*, mis aitab sul mängus hiirega paremini jälgida olemasolevat maailma.
- *Character Motor,* mis aitab tegelaskujul liikuda nii nagu vaja. Füüsikalised omadused nagu gravitatsioon ja kiirendus on selles skriptis hästi ära toodud.
- *FPSInput Controller*, mis aitab klahvivajutustele reageerida.

Üldjuhul oleks vaja ka *Collider* mängutegelasele juurde lisada ning kindlasti ka *Rigidbody,* et mängijale kehtiks füüsikalised seadused..

6.3 Mängu temaatika

Esiteks peab importima relva mudelid maastikule, mida mängija koguda saab. Importimine käib kergelt. Tuleb vaid arvutis lohistada mudel *Unity Assets'ite* alla ning mudel ongi kasutamisvalmis. Tõstame relvad äsja tehtud maastikule ning kindlasti tuleb lisada mudeli mänguobjektile *Collider* ning panna *Inspector'is Tag* sektsiooni alla näiteks „relv", et me pärast aru saaksime, mis objektiga mängija kokku põrkas. Lisab lisaks relvadele ka mudeli, mis kehastab inimest, kellele tuleb lõpuks ette kanda, et kõik relvad on kogutud. Mängu pilt näeks välja siis selline: pildil on kaks relva ning tagaplaanil on inimene.

Joonis 18. Valminud mängu vaade.

Mängu funktsiooniliseks koodiks on vaja ainult ühte skripti, sest siin mängus pole palju funktsionaalsust. Esiteks loeb skript iga kord kogutud relvade arvu, mis mängija on üles korjanud ning kui ta on kõik kuus relva üles korjanud, siis saab ta alles relvad viia inimesele ning peale kokkupõrget inimesega algab mäng uuesti otsast peale. Iga kord kui mängija on teatud relva endale võtnud ehk siis relvale otsa jooksnud, kaob see relv maastikult *Destroy* funtksiooni abil.

Koodis on olemas ka funktsioon nimega *OnGUI* , mis väljastab väikese kasti mängija jaoks, mis näitab palju tal relvi kogutud on.*OnGUI* on samasugune sisseehitatud funktsioon nagu seda on näiteks *Update.* Relvade arv on skriptis juba määratud muutujaga *GunsNumber.*

Sarnane mängu näide on saadaval leheküljel http://www.filedropper.com/unityexample.

```
using UnityEngine;
  using System.Collections;

public class kasti_plahvatus : MonoBehaviour {
      static private int blockCount=0;
      static private int GunsNumber=6;
      static private int sum;
      public bool showGUI;
      void OnControllerColliderHit(ControllerColliderHit collision){

            if (collision.gameObject.tag == "relv") {
                  blockCount++;
                  Debug.Log ("Relvi on kogutud:" + blockCount);
                  Destroy (collision.gameObject);

            }
            else if (blockCount >= GunsNumber) {
```

```
                if (collision.gameObject.tag == "inimene") {
                    blockCount = 0;
                    Application.LoadLevel
 (Application.loadedLevelName);
                }
            }
        }    // Use this for initialization
        void Start () {
        }


        // Update is called once per frame
        void Update () {
        }


        void OnGUI() {
            GUI.Button(new Rect(10, 10, 250, 100), "Sul on hetkel" +
blockCount + „relva");
```

Koodinäide 10. Relvade kogumine ning nende hävitamine ja kogutud relvade lugemine. Juhul kui relvad on kõik koos ja inimesele viidud, siis algab mäng uuesti. Skript tuleb lisada mängutegelase komponentide alla.

Kasutajate tagasiside

Kas mõni mõiste jäi segaseks?	Kas koodinäiteid oli piisavalt?	Kas ekraaanipilte oli piisavalt?	Kas õppematerjali ülesehitus oli loogiline?	Kas õppematerjal on läbitav 4*45 minutiga?	Kui huvitavaks hindad õppematerjali skaalal 0 kuni 10?
Ei	Ei	Jah	Jah	Ei	5
Ei	Jah	Jah	Vägagi	Ei	6
EI	Jah	Jah	Jah	Jah	8
Mesh	Jah	Jah	Vägagi	Jah	9

Autor lasi oma töö läbi töötada ka neljal inimesel, kellest kaks inimest on üsnagi suure IT baasiga ning ning kaks pole IT-ga kokku puutunud. Üldjoontes saadi seminaritööst aru ning suudeti teha ka mõned koodinäited kaasa. Arusaadaval põhjusel olid kahel inimesel raskused mõnest skriptist aru saamisega. Õppematerjali ülesehituse kohta suuremaid etteheiteid polnud, kuid kohati pidid lugejad *Unity's* ise asju järgi proovida, et teemast korralikult aru saada või asjad üles leida. Teema oli huvitav inimestele, kes on IT taustaga ja mitte nii väga huvitav tava arvuti kasutajale. Vahe oli sees ka arvamusel, kas õppematerjal on läbitav 4*45 minutiga. IT taustaga inimeste jaoks oli see igati läbitav selle ajaga, tavakasutajate arust mitte. Ekraanipiltide rohkusega oldi rahul, kuid üks õppematerjali läbija arvas, et võiks olla rohkem koodinäiteid. Suurim ajakulu oli objektide liigutamise selgeks saamine ja ära harjumine.

Kokkuvõte

Seminaritöö eesmärk oli luua väikese programmeerimiskogemusega huvilistele eestikeelne õppematerjal Unity 3d õppimiseks.

Selle jaoks uuris autor Unity keskkonda ning läbis programmi tähtsamad teemad ning lisaks valmistas ta ühe väiksema mängu.

Absoluutselt kõigist Unity eripäradest ning funktsioonidest kirjutamine oleks olnud liiga mahukas, nii et töö autor piiras nende hulka rohkesti, kirjutades ainult elementaarsematest teemadest. Kindlasti ei pea seda õppematerjali läbides olema väga kogenud IT spetsialist, kuid mingi arusaamine kodeerimisest peab siiski olemas olema. Kogenumatel inimestel kulub lihtsalt materjali läbimiseks vähem aega. Õppematerjal on sobilik inimestele, kelle kogemus mängude tegemise vastu on veel väike ning kes alles tutvuvad mängumootoritega. Õppematerjali läbi töötanud õpilane peaks olema võimeline looma mõne lihtsama töötava mängu.

Tööd on võimalik mitmeti edasi arendada. Autor kindlasti jätkab töö arendamist mahukama mängu näite abil, kus tulevad mängu juba keerukamad komponendid.

Kasutatud kirjandus

Creighton, R (2012, veebruar 21). *Introduction to Game Development Using Unity 3D.* Retrieved jaanuar 2, 2014, from GameDev:

http://www.gamedev.net/page/resources/_/technical/game-programming/introduction-to-game-development-using-unity-3d-r2875

Unity Tehnologies (2013, mai 29). *Learning the Interface.* Retrieved jaanuar 5, 2014, from Unity:

http://docs.unity3d.com/Documentation/Manual/LearningtheInterface.html

Ward, J (2008, aprill 29). *What is a Game Engine?.* Retrieved jaanuar 5, 2014, from Gamecareerguide:

http://www.gamecareerguide.com/features/529/what_is_a_game_.php

Brodkin, J (2013, juuni 3). *How Unity3D Became a Game-Development Beast.* Retrieved jaanuar 10, 2014, from Slashdot:

http://slashdot.org/topic/cloud/how-unity3d-become-a-game-development-beast/

Soelkner, R (2013, november 8). *Bringing AAA quality to Unity.* Retrieved jaanuar 10, 2014, from Bitsalive:

http://www.bitsalive.com/bringing-aaa-quality-to-unity/

Baumgart, B (1975, mai 1). *Winged-Edge Polyhedron Representation for Computer Vision.* Retrieved jaanuar 12, 2014, from Baumgart:

http://www.baumgart.org/winged-edge/winged-edge.html

Unity Tehnologies (2012, oktoober 19). *Game View.* Retrieved jaanuar 15, 2014, from Unity:

http://docs.unity3d.com/Documentation/Manual/GameView40.html

Unity Tehnologies (2012, oktoober 15). *Project Browser.* Retrieved jaanuar 17, 2014, from Unity:

http://docs.unity3d.com/Documentation/Manual/ProjectView40.html

Unity Tehnologies (2013, september 25). *Inspector.* Retrieved jaanuar 17, 2014, from Unity:

https://docs.unity3d.com/Documentation/Manual/Inspector.html

Unity Tehnologies (2013, juuli 5). *Camera.* Retrieved jaanuar 18, 2014, from Unity:

http://docs.unity3d.com/Documentation/Components/class-Camera.html

Unity Tehnologies (2013, juuli 12). *Light.* Retrieved jaanuar 18, 2014, from Unity:

https://docs.unity3d.com/Documentation/Components/class-Light.html

Unity Tehnologies (2013, aprill 12). *Rigibody.* Retrieved jaanuar 22, 2014, from Unity:

http://docs.unity3d.com/Documentation/Components/class-Rigidbody.html

Unity Tehnologies (2014, märts 13). *Joints.* Retrieved jaanuar 22, 2014, from Unity:

http://unity3d.com/learn/tutorials/modules/beginner/physics/joints

Unity Tehnologies (2014, märts 13). *GetButton and GetKey.* Retrieved jaanuar 27, 2014, from Unity:

http://unity3d.com/learn/tutorials/modules/beginner/scripting/get-button-and-get-key

Unity Tehnologies (2014, märts 13). *Instantiate.* Retrieved jaanuar 27, 2014, from Unity: https://unity3d.com/learn/tutorials/modules/beginner/scripting/instantiate