Tallinna Ülikool
Informaatika Instituut

Unity 2D keskkonnas rakenduste loomine Javascripti abil

Seminaritöö

Autor: Magnus Kvell
Juhendaja: Jaagup Kippar

Autor: ..... ,, ... ,, 2014
Juhendaja: ..... ,, ... ,, 2014
Instituudi direktor: ..... ,, ... ,, 2014

Tallinn 2014

Sisukord

Sissejuhatus

Käesoleva seminaritöö eesmärgiks on luua eestikeelne õppematerjal, mis tutvustab, kuidas panna kokku kahemõõtmelisi rakendusi Unity mängumootori abil. Seminaritöö käigus on kasutusel Unity 4.5 versiooni ja Javascripti programmeerimiskeel. Täpsemalt 4.5.4f on kasutusel sellepärast, et tegu on kõige uuema versiooniga ja see annab ligipääsu kõige uuematele mängumootori vahenditele. Samuti puudub vanematel versioonidel kui 4.3 sisse ehitatud 2D tugi, mis on kasutusel siinses õppematerjalis. Töös käsitletakse Unity mängumootorit, sest see on hetkel üks populaarsemaid ja sobib pea igale platvormile(arvuti, konsool, nutitelefon).

Õppematerjal on peamiselt suunatud inimestele, kes soovivad luua rakendusi või mänge Unity 2D keskkonnas. Soovitatav on varajasem kokkupuude Javascripti programmeerimiskeelega. Kasuks tuleb kogemus antud või mõne teise sarnase mängumootori kasutamisel(Source engine, Unreal 3 engine).

Seminaritöö käigus käsitletakse lühidalt objektide tegemist, nendele füüsikaliste omaduste andmist, tekstuuride lisamist, objektide liikuma panemist ja keskkonna tutvustust. Õppematerjal on sissejuhatav ning sellepärast peab õppur arvestama, et õppematerjalis käsitletakse Unity mängumootorist ainult väikest osa. Tööst on välja jäetud teemad nagu näiteks üle võrgu mitmikmängu loomine ning muusika lisamine rakendusele.

Seminaritöö lõpuks valmib Unity *2D* vahendeid kasutades õhuhoki mäng, mille eesmärgiks on anda esmane kogemus Unity mängumootori keskkonnas ringi liikumisest ning selle lihtsamate vahendite kasutamisest. Mäng on mõeldud kahe inimese jaoks ja kasutab kohaliku mitmikmängu võimalust.

Teema valiku põhjuseks on autori isiklik huvi selle keskkonna ja mängude loomise vastu. Peale selle leiab veel autor, et arvutimängud arendavad inimese loogilist mõtlemist ning probleemide lahendamise oskust. Sellest tulenevalt on mängude koostamine loominguline ning keerukas protsess, mis vajaks rohkem lahti seletamist. Samuti on viimase kümnendi jooksul žanri ja mängude tegemise populaarsus eriti antud keskkonnas kasvanud, tänu sellele on mängude kaudu võimalik näiteks inimesi harida.

Antud õppematerjal käsitletakse uudseid 2D vahendeid Unity mängumootoris, mille kohta puudub eestikeelne informatsioon. Seminaritöö koostamisel võttis autor eeskuju teistest sarnastest õppematerjalidest.

Mõisted

Mängumootor(inglise k. game engine)- „Mängumootor on süsteem, mis on kavandatud videomängude loomiseks ja arendamiseks. Eksisteerib väga palju mängumootoreid, mis on kavandatud töötama konsoolidel ja personaalarvutitel. Mängumootor sisaldab tavaliselt renderlus(visualiseerimis) mootorit 2D või 3D graafikaks, füüsikamootorit või kokkupõrke avastajat, heli, skriptimist, animatsioone, tehisintellekti, võrgundust, mälu haldamist, lokaliseerimise toetust ja stseeni graafikut (Stseeni graaf on andmestruktuur, mida kasutavad vektoripõhised graafikatöötlusrakendused ja tänapäeva arvutimängud)“ (Jeff Ward 2014).

Funktsioon(inglise k. function)- Programmeerimises nimetatakse funktsiooniks programmi osa, mis teostab mingit konkreetset tegumit. Selles mõttes kujutab funktsioon endast teatud tüüpi protseduuri või rutiini(Vallaste, 2014).

Skript(inglise k. script) Põhimõtteliselt sama, mis makro või pakkfail (batch file). Skript kujutab endast käsujada, mida täidetakse ilma kasutajapoolse vahelesegamiseta. Skriptikeel on lihtne programmeerimiskeel, millega saab skripte kirjutada(vallaste, 2014).

String, sõne- Arvutiterminoloogias tervikuna käsitletav elemendi- või märgijada. Erinevalt sõnast või arvust ei pruugi stringil olla mingit konkreetset tähendust – string võib näiteks olla ka suvaline sõnaosa(vallaste, 2014).

Liides(inglise k. interface)- „Sõltumatute funktsionaalüksuste vaheline ühispiir, kus kehtivad koostöötingimused, mis puudutavad funktsioone, füüsilisi ühendusi, signaalivahetust jms“ (vallaste, 2014).

SDK(inglise k. Software development kit)- „arendustarkvara Programmipakett, mis võimaldab programmeerijal luua rakendusi konkreetsele platvormile. Harilikult sisaldab arendustarkvara üht või enamat rakendusliidest, programmeerimisvahendeid ja dokumentatsiooni“ (vallaste, 2014).

graafiline kasutajaliides(inglise k. Graphical user interface)- Arvuti graafikakuvamise võimalusi kasutav tarkvaraliides, mis teeb programmide kasutamise lihtsamaks(vallaste, 2014).

Objekt(inglise k. object)- „Üldises mõttes nimetatakse objektiks mistahes asja, mida saab individuaalselt välja valida ja manipuleerida. Arvutustehnikas võib see olla näiteks kuvarile

ilmuv pilt või mingi tarkvarakomponent. Objektorienteeritud programmeerimises kutsutakse objektiks andmete ja neile rakendatavate meetodite terviklikku komplekti“ (vallaste, 2014).

Vektor(inglise k. Vector)- „Arvutigraafikas joon, mis on määratud oma otspunktidega xy või xyz koordinaadistikus. Kui arvuti joonistab ringjoont, siis koosneb see tegelikult paljudest lühikestest vektoritest“ (vallaste, 2014).

Stereopilt, kolmemõõtmeline graafika(inglise k. 3 Dimensional graphics)- Kolmemõõtmeliste objektide loomine, kuvamine ja töötlemine arvutis. Kolmemõõtmelised CAD ja graafikaprogrammid võimaldavad luua objekte XYZ-teljestikus (pikkus, laius, kõrgus). Neid saab pöörata ja vaadelda suvalise nurga all, samuti proportsionaalselt suurendada või vähendada (skaleerida). Objekte saab ka soovitava nurga all valgustada ja tekitada neile varje(vallaste, 2014).

Kahemõõtmeline graafika (inglise k. 2D Graphics)- Arvuti genereeritud digitaalsete piltide kogum, koosneb enamasti kahemõõtmelistest geomeetrilistest mudelitest, tekstist ja digitaalsetest piltidest. Matemaatikas tasapinnaline ehk võimaldab luua objekte XY-teljestikul(Pile, J., & Jr., 2014).

Stseeni vaade(inglise k. Scene View)- „mängumootori vaade, mida kasutatakse, mängija, keskkonna, kaamera ja teiste mängu objektide valimiseks ning positsioneerimiseks“ (Unity Technologies, 2014).

Mängu vaade(inglise k. Game view)- „Kaamera järgi visualiseeritud mängu vaade, mis kuvab ülevaate lõpp produktist“ (Unity Technologies, 2014).

Kaamera(inglise k. Camera)- „Kaamerad on seadmed, mis salvestavad ja kuvavad mängumaailma mängijale. Kaameraid saab kasutada visualiseerimiseks ükskõik, mis järjekorras ja suvalises asukohas. Neid võib olla lõpmatu arv“(Unity Technologies, 2014).

Inspector- „Mängud Unity mängumootoris koosnevad objektidest ehk muusikast, valgustusest, koodist ning teistest graafilistest elementidest. Inspector on sektsioon Unity mängumootoris, mis kuvab detailse informatsiooni mängu objektide kohta“(Unity Technologies, 2014).

Hierarchy- „Unity mängumootoris sektsioon, kus asuvad kõik stseenis olevad mängu objektid“(Unity Technologies, 2014).

Assets- „Sektstioon Unity mängumootoris, mis kuvab visuaalselt mänguobjektid“(Unity Technologies, 2014).

1 Unity mängumootor ja 2D

Unity mängumootor arendati 2005. aastal *Mac* operatsioonisüsteemile. Nüüdseks on levinud mootor enam kui viieteistkümnele erinevale platvormile ja samuti on see „Nintendo Wii U“ mängukonsooli *SDK* arendusvahendiks. Unity peamiseks plussiks on lihtsus rakenduste loomisel mitmele erinevale platvormile korraga, näiteks saab samaaegselt arendada mänge nutitelefoni ja arvuti peale. Mängumootori laialdase leviku tõttu on kasutajate poolt tehtud keskkonna jaoks mitmeid erinevaid liideseid, tekstuure ja abivahendeid, mida saab osta või tasuta alla laadi lehelt https://www.assetstore.unity3d.com/en/ (Unity - What is Unity, 2014).

Tasulise versiooni eeliseks Unity mängumootori puhul on rohkemad funktsionaalsused, näiteks saab heli paremini kohandada ja valgusefekte mugavamalt sättida. 2D Mängude tegemisel väga suurt erinevust tegelikkuses ei ole (Unity - Licenses Comparison, 2014).

Selles õppematerjalis liideseid ei ole vaja kasutada, aga enne versioon 4.3 Unity enda poolsed 2D tööriistad puudusid ja ainus võimalus oli kas ülaltoodud veebikeskkonnast liides selle jaoks alla laadida või improviseerida 3D vahendeid kasutades. Nüüdseks on mootorisse sisseehitatud 2D keskkond, mis iga versiooniga täieneb (Unity - What is Unity, 2014).

1.1 Unity ja Javascript

Koodihalduseks on mängumootorisse sisseehitatud programm *„Mono Developer“*, milles saab kirjutada Javascripti, C# ja Boo programmeerimiskeeltes. Üleüldiselt kõige lihtsamaks õpetamiskeeleks peetakse Unity puhul Javascripti, mis on ka kasutusel siinses seminaritöös. Põhjuseks on arvatavasti, et Javascript on nendest kolmest keelest kõige kasutajasõbralikum ja selle keele kohta on kõige rohkem tasuta avalikke õppematerjale. Tehnilise poole pealt ei ole vahet väga, mis keeles rakendusi kirjutada, kuid tasuks meeles pidada, et Unity peal on Javascripti kõvasti kohandatud mängumootori jaoks ja see erineb veebis kirjutatavast, sellest on ka tulnud Unity kasutajate poolne hüüdnimi Javascriptile „*UnityScript*“ (Unity - What is Unity, 2014).

1.2 Teisi Unity 2D õpetusi ja abivahendeid

Inglisekeelseid õppematerjale Unity mängumootori kohta võib leida päris palju, siinkohal toon välja oma kogemusel kergemad 2D mängude loomise materjalid ja parimad abivahendid Unityga rakenduste loomiseks algajatele. Tuleks arvestada, et kõik järgmised õppematerjalid on inglise keeles. Nendes materjalides käsitletud funktsioone eeskujuks võttes on koostatud ka seminaritöö.

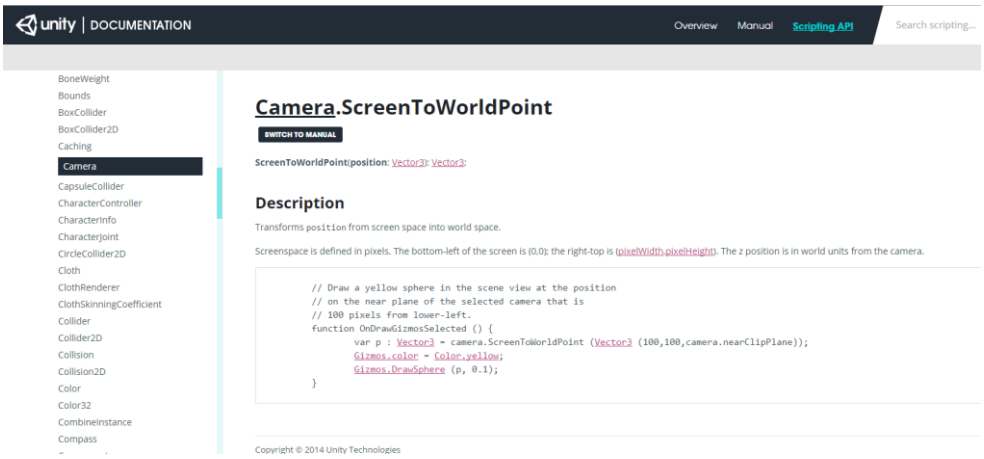
http://walkerboystudio.com/html/unity_course_lab_4.html - Videoõpetusest leiab „Super Mario“ mängu klooni. Selles juhendis näidatakse alguses kuidas näeb välja mängude õige dokumentatsioon ja hiljem saab teha õpetuse järgi ise valmis Mario mängu. Soovitan antud veebisaidilt vaadata ka teisi juhendeid, kui huvitab 3D mängude tegemine.

Probleemiks on selle õpetuse puhul, et õpetus on ligi 23 tundi pikk. Õpetus on samas väga põhjalik ja sealt võib leida ideid kuidas teisi mänge teha. Soovitan seda õppematerjali lugeda pärast seda kui autori poolt tehtud õppematerjal on juba läbitud. Miinuseks nende videote puhul on see, et õpetus on kolm aastat vana, mis tähendab, et õppematerjal on natukene aegunud.

- Õpetuses puudub ülevaade 2D keskkonna kohta ning seal kasutatakse kõige jaoks kolmemõõtmelist skaalat. Autoripoolse õpetuse puhul saab mängijate külge panna 2D füüsika komponente, mis teeb materjali erinevaks. Kihtide asemel, mida kasutame hiljem siinses materjalis kasutatakse sealses õpetuses z-telje koordinaate.
- Õpetuse plussiks on seal käsitletav animeerimise ja muusika pool, mis autori tehtud õppematerjalis puudub. Üks olulisi Mario õppematerjali teemasid on kaamera paika panemine nii, et see jälitaks mängijat. Lahti seletatakse ka tasemete koostamine, mida samuti autori tehtud mängu puhul vaja ei läinud, aga on Unitys on see väga oluline osa (Walker Boys, 2014).

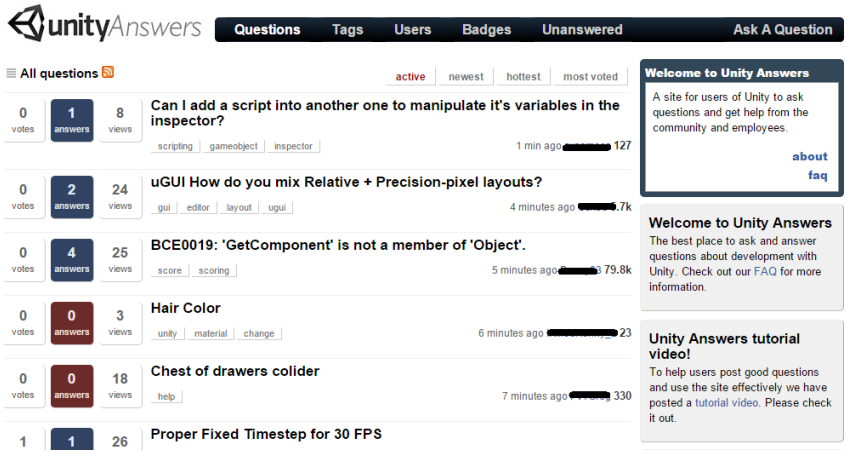
Joonis 1. Super Mario kloon(Walker Boys, 2014).

http://docs.unity3d.com/Manual/index.html - Tegu ametliku Unity dokumentatsiooniga, kust leiab kõik mängumootori siseselt kasutatavad funktsioonid. Dokumentatsioonis on veel lahti seletatud kõikide tööriistade funktsionaalsused. (Unity Technologies, 2014).



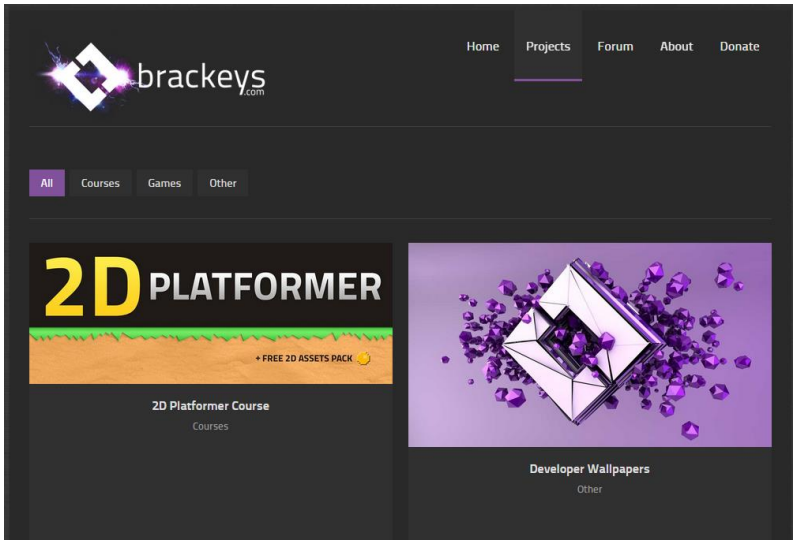
Joonis 2. Unity dokumentatsioon(Unity Technologies, 2014).

answers.unity3d.com/ - Unity ametlik veebileht, kus saab küsimusi küsida ja lahendusi otsida tekkinud probleemidele. Tagasiside saamine kommuunist on kiire ja vastused asjakohased. (AnswerHub, 2014).



Joonis 3. Unity Answers leht(AnswerHub, 2014).

https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA, http://brackeys.com/ *Brackeys* nimeline *youtube'i* kanal ja sama autori veebileht, kust leiab õpetusi, nii uue 2D keskkonna kui ka 3D mängude koostamise kohta. Samuti on seal graafika rakenduse õpetus, mis on suunatud algajatele. Kanalilt või lehelt leiab veel *assetstore*i jaoks tehtud tasuta tekstuure ja muid abivahendeid (Asbjørn Thirslund, 2014).



Joonis 4. Brackeys projektid(Asbjørn Thirslund, 2014).

1.3 Õppematerjalide uurimise järeldus

Autori poolt uuritud eelnevates huvitavamates õppematerjalides on Unity funktsionaalsusi käsitletud kui tervikuna mängus. Peale selle on mängumootoris objektide, skriptide ja füüsika

kokku panemine keskmisest keerukama raskusastmega. Sellest tulenevalt ja lähtudes vaadatud materjalidest on mõttekam autori arvates teha valmis rakendus õppematerjalis, mis annaks ülevaate kuidas reaalselt mängumootoris midagi ehitada. Lihtsate funktsionaalsuste ükshaaval seletamine ei anna teadmisi kuidas koostada Unitys mänge inimesele, kes pole mängumootoriga varem kokku puutunud.

Peale selle peaks õppematerjal andma eelduse Unity põhifunktsionaalsustest mängu kokku panemisel, mis aitaksid edasi minna keerukamate rakenduste juurde nagu näiteks „Super Mario“ mäng (Walker Boys). Sellepärast toimuski keerukama materjali uurimine, et näha, mis funktsionaalsusi kasutatakse nende mängude juures.

1.4 Unity olulisemad funktsioonid

Erinevaid õppematerjale sirvides on autor võtnud suuna luua mingi terviklik rakendus, et anda õpetatavatele pilt kuidas rakendust kokku panna ja mis etappidest koosneb tervikliku mängu koostamine Unity mängumootoris. Peale kahemõõtmelise keskkonna on autor lugenud õppematerjale Unity 3D poole kohta, aga neid töö kirjutamisel ei ole otseselt kasutatud, sellepärast ei ole nende materjalid töös välja toodud. Eelnevast tulenevalt on õppematerjali valitud järgmised funktsioonid:

- Tausta loomine- Iga mängu aluseks on keskkond, kus tegevus toimub, sellepärast üks olulisemaid funktsioone on taust ja selle paika panemine kaamera suhtes.
- Mängijate loomine ning nende kontrollimine- Mängija loomine on oluline osa, sest igas mängus peaks olema vähemalt üks mängija, ilma mängijata oleks tegu animatsiooniga.
- Objektide liikumise piiramine mängus- Kaamera vaateväli on mängudes piiratud ja oleks otstarbekas, et mängijad ja erinevad liikuvad objektid ei liiguks kaamera vaateväljast välja.
- Füüsikalised omadused- Igale objektile mängus kuuluvad füüsikalised omadused, mille paika panemine sõltub objekti olemusest.
- Graafilise kasutaja liidese(GUI) kasutamine punktide loendamiseks- enamus mängudes on kasutusel mingi punktide süsteem. Graafiline liides on veel kasutusel näiteks mängu menüüde tegemisel.

1.5 Unity eelised teiste mängumootorite ees

Mis funktsionaalsused on Unity mängumootoris ja mis funktsionaalsused peaksid üldse mängumootorites olema, ei ole autori arvates antud õppematerjalis asjakohane välja tuua, sellepärast et formaat on selle jaoks liiga lühike. Siin toon selle asemel välja mõned olulisemad Unity mängumootori puudused ja eelised võrreldes teiste mootoritega.

Eelised:

- Finants - mängude tegemine on ülimalt kallis ettevõtmine. Paljud arendajad kasutavad enda loodud mängumootoreid. Tänu Unityle saavad väiksemad arendajad raha kokku hoida ja ei pea enda mootorit arendama(Thorn, A., 2014).
- Unity toetab standardseid mängutööstuses kasutatavaid formaate nagu näiteks FBX, 3DS, MA, MB, WAV,PNG, PSD jne(Thorn, A., 2014).
- Säästab aega mängude tegemisel mitmele platvormile, konkurendid on enamuses enamjaolt pühendunud ainult ühele platvormile.
- Platvormid, mida Unity mängumootor toetab - BlackBerry 10, Windows Phone 8, Windows, OS X, Linux, (Android, iOS, Unity Web Player (k. a. Facebook), Adobe Flash, PlayStation 3, PlayStation 4, PlayStation Vita, Xbox 360, Xbox One, Wii U, and Wii(Unity Technologies, 2014).

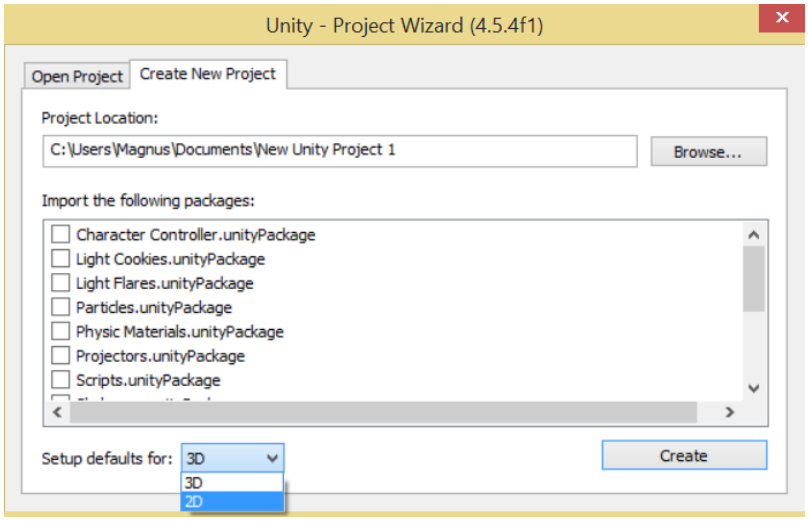
Puudused:

- Arvatakse, et Unity 2D mängude tegemise vahendid ei ole väga arenenud. Mõneti vaadatakse Unityst sellepärast mööda ja kasutatakse näiteks „Construct2“ mängumootorit, mis on täielikult pühendunud 2D mängudele. Tegelikult annab Unity keskkonnaga väga edukalt 2D mänge teha, ainuke puudus on selles et, peab kasutama vahepeal 3D funktsioone (Thorn, A., 2014).
- Kahemõõtmeliste mängude tegemiseks on lihtsamaid vahendeid nagu näiteks „*Construct2*“ või „*RPG maker*“. Tegelikult sõltub vahendite lihtsus keskkonnaga harjumisest ning Unity mängumootoriga harjudes ei ole mängude tegemine raskem kui teistes programmides (Thorn, A., 2014).

2 Projekti alustamine Unitys

Unity mängumootori saab alla laadida lehelt http://unity3d.com/unity/download. Mootori kasutamine on tasuta, kui on vajadus võimsamatele mängumootori vahenditele ligi pääseda, siis on võimalik osta Unity kodulehelt tasuline versioon. Mängumootori paigaldamine on lihtne, pärast alla laadimist tuleb mängumootor installeerida ja käima panna, midagi muud juurde ei ole vaja alla laadida.

Pärast mängumootor alla laadimist tuleb alustuseks luua uus projekt. Esimesel korral Unity keskkonda käima pannes avaneb aken, kus saab uut projekti alustada või vana valida. Akna mitte avamise puhul(vt joonis 5) tuleks teha kaks vasakut klõpsu Unity ikooni peal ja kohe pärast seda ALT nuppu klaviatuuril all hoida, kui programm käima läheb(Asbjørn Thirslund, 2014). Algselt peaks avanenud olema sektsioon *Open Project*, uut projekti tehes tuleb vajutada lehele *Create New Project.* Aknas on võimalik valida, kuhu projekt salvestada ja seal saab anda nime uuele projektile. All nurgas, kus on kirjas *Setup defaults for*, tuleb valida 2D ja vajutada nupule *Create*, et avada 2D keskkond. Siinses õpetuses ühtegi lisa importida ei tule(vt Joonis 5). Muidu on võimalik importida pakette näiteks mängule valguse või füüsika efektide lisamiseks.



Joonis 5. 2D Keskkonna avamine.

2.1 Keskkond ja sellega tutvumine

Avanenud keskkonnas üleval võib näha lahtreid *Scene view* ja *Game view(*vt joonis 6*). Scene view* on aken, kus toimub põhiline mängu ülesehitamine ja *Game view* on aken, kus on näha, milline mäng kaamera suhtes hetkel välja näeb.

Stseeni käivitamiseks mängu vaates on üleval paremal nupp *play*, mida vajutades toimub üleminek *Scene viewst, Game viewsse.* Kõrval on *paus*i nupp, mida vajutades jääb mängu stseen mängu vaates hetkeliselt seisma (vt joonis 6). Mängumootori keskkonda on võimalik enda jaoks mugavamaks teha, lohistades mõlemad aknad korraga nähtavale. Välimust saab muuta kas aknaid keskkonnast hiire abil välja vedades või tööriista real üleval valida *Window->Layouts* ning sealt sobiv kujundus leida.

Joonis 6. Tähtsamad nupud.

Ülesandeid

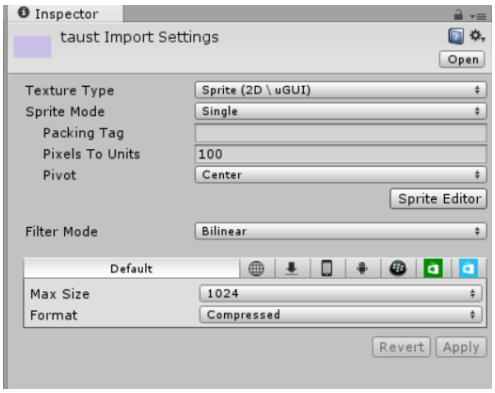
1. Uuri Unity mängumootorit iseseisvalt, proovi näiteks liigutada keskkonda enda jaoks mugavaks.

2.2 Tausta paika panemine ning kihid

Esimese etapina 2D mängu loomisel tuleb mõelda mängu keskkonna peale ja panna paika taust. Lae alla aadressilt http://www.tlu.ee/~manz/hoki/hokiassets.rar autori poolne tekstuuride või ehk siis *Spritede* kogum. *Sprited* on lühidalt teise nimega 2D tekstuurid (2DWillNeverDie, 2014).

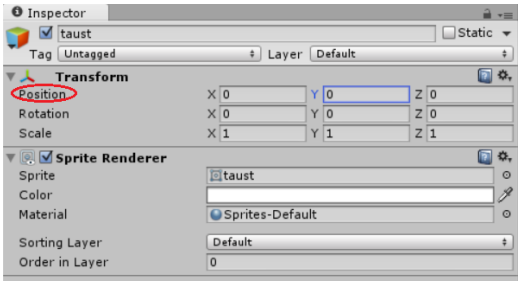
Impordi alla laaditud kaustast tekstuur nimega „taust“. Üks moodus selle jaoks on kaustast „hokiassets“ paremat klõpsu peal hoides tekstuur vedada Unity keskkonnas kasti *assets*. Koodiga probleemide tekkimisel leiab valmis skriptid aadressilt: http://www.tlu.ee/~manz/hoki/skriptid.rar.

Unity keskkonnas pildi „taust“ peale hiirega vajutades on paremal pool nurgas näha sektsiooni *Inspector*. Seda vaadates võib tähele panna, et tausta pildi puhul on tegu tekstuuriga *sprite*. Keskkond lubab antud lahtris muuta, kui suurena pilt ilmub ja kuhu peaks ta ekraani ja stseeni vaate suhtes jääma(*Pivot*). (vt joonis 7).



Joonis 7. Tekstuuri seaded.

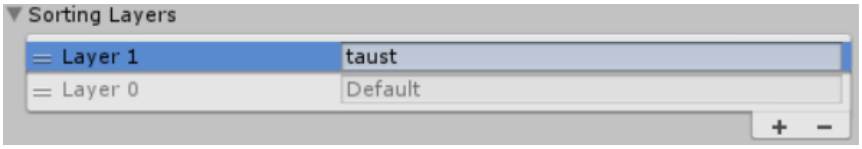
Stseeni vaatesse tekstuuri lisamiseks kiireim võimalus on tõsta *Assets* kastist pildid hiire abil kaamera vaate peale(Asbjørn Thirslund, 2014). Kui see on tehtud, oleks sobilik pildi koordinaadid kõik nulliks seada, siis on teada, et pilt on täpselt keskpunktis. Selleks tuleb x, y, ja z väärtuste kohale 0 kirjutada, z peaks igal juhul juba 0 olema.(vt joonis 8).



Joonis 8. Tausta positsioneerimine.

Kaheksanda joonise pealt on näha lahtrit *Sorting Layer*. See Unity funktsioon annab võimaluse grupeerida objekte kihtide kaupa (näiteks: tausta kiht, mängijate kiht jne). Taustapildile tuleb lisada enda kiht, et see jääks teiste objektidega võrreldes kihiliselt allapoole. Selleks, et seda teha, tuleb valida *Sorting Layeri* (*Default*)- *Add sorting layer*.

Paremas nurgas on näha pluss ja miinus märki, kihi lisamiseks tuleb plussi peale vajutada. Anname kihile (*Layer 1*) nimeks „taust“ ja lohistame selle pealmiseks kihiks (vt joonis 9).



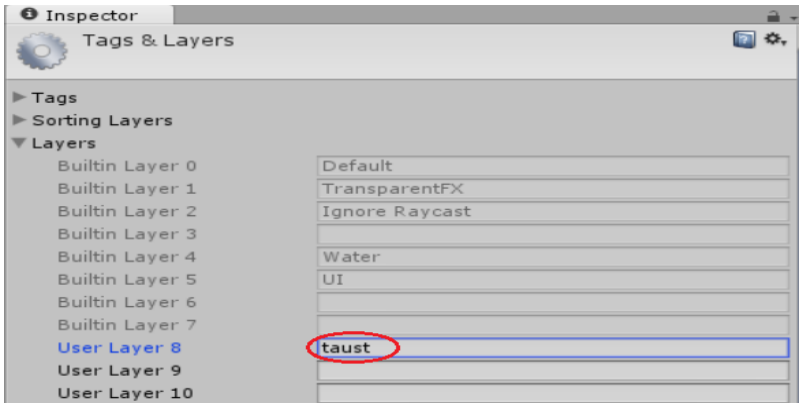
Joonis 9. Kihtide hierarhia.

Pärast seda saab liikuda tagasi samasse positsiooni, nagu Joonis 8 peal, selleks võib valida vasakult kastist *Hierarchy*, objekti „taust“. Klõpsates kihi nupu *Sorting Layer*(vt joonis 8) peal, saab näha, et sinna on ilmunud kiht taust, mille saab seal valida.

Ära tuleks muuta *Order in Layer* lahtris number 0, miinus kahekümneks. Siinkohal *Order in Layer* näitab kihtide järjekorda. Numbri näiteks -20 kirjutades on kiht mängus kindlalt viimane.

2.3 Tausta liikumatuks tegemine

Suure tausta puhul, et seda mitte paigast liigutada stseeni vaates, tuleks see ära lukustada teatud positsioonile. Tausta lukustamiseks saab *Inspectori* alt nupu *Layer* ja sealt *Add Layer*. Sinna saab lisada uue kasutaja tehtud kihi(*User Layer 8*) nimega „taust“ (vt joonis 10).

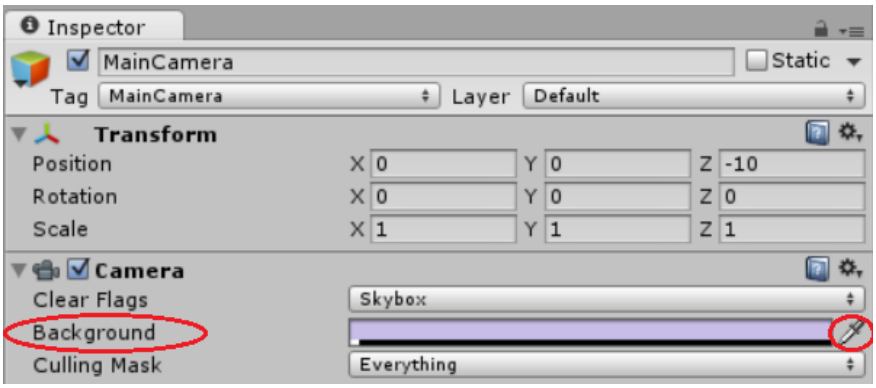


Joonis 10. Uue kihi lisamine

Märgime ära *Hierarchy* all tausta uuesti. Peale seda saab Layer nupu alt valida tehtud kihi „taust“. *Inspectori* sektsioonist valides rippmenüü nimega *Layers* on nähtaval seal tehtud kiht „taust“ ning väike lahtine tabaluku märk selle kõrval paremal. Kihi paigale lukustamiseks tuleb vajutada tabaluku peale. Mängu taust on peale neid toiminguid liikumatu.

2.4 Unity keskkonna taustavärvi muutmine

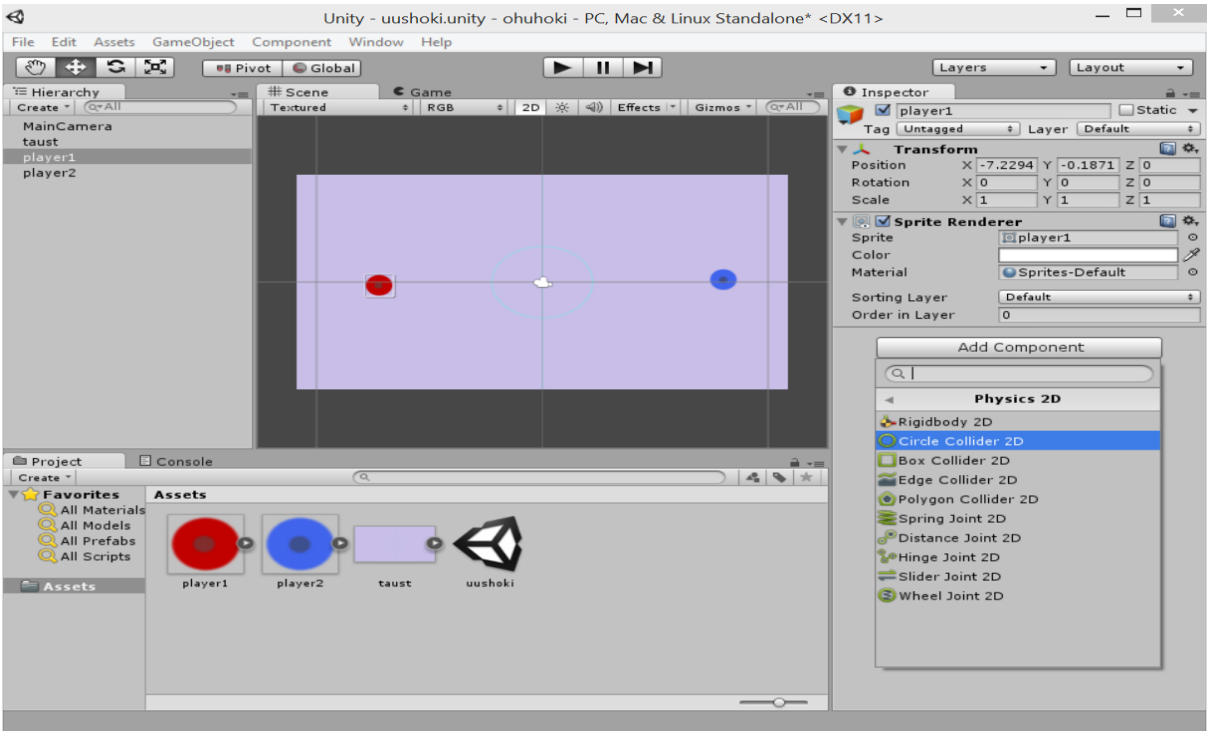
Selleks, et mäng töötaks kõigi resolutsioonide peal, ei tohiks olla mänguväljaku suurust ekraani peal muutes näha Unity keskkonna sinine taust. *Game view* tausta värvi muutmiseks vajutame *Hierarchy* alt *Main Camera* peale. Hiljem koodi lihtsamaks kirjutamiseks võib nimetada *Main Camera* ümber, võttes vahelt tühiku ja uueks nimeks on vastavalt *MainCamera.* Pärast seda tuleb uuesti *Inspectori* alla liikuda, kust saab valida *Background* lahtrist värvivalija ja klõpsata tausta pildi peale *Game views*. Peale seda peaks taust paigas olema(vt joonis 11).



Joonis 11. Keskkonna tausta värvimine.

2.5 Mängija kontrollimine

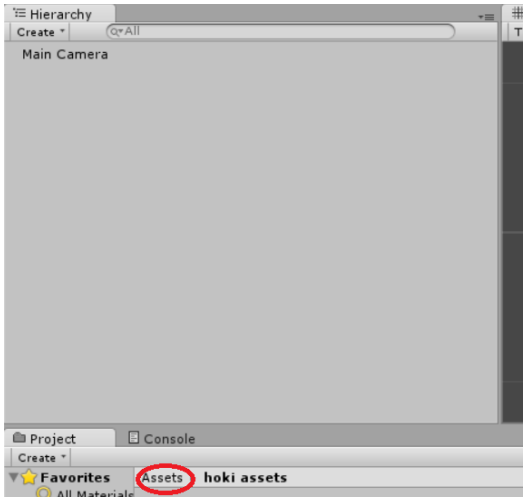
Objektidele klahvide määramiseks, tuleks importida autori tehtud tekstuuride kogumist mängijate *sprited.* Tekstuuride nimed kaustas on „*player1*“ ja „*player2*“. Samamoodi nagu tausta puhul tuleb need hiirega vedada *Assets* kasti. Kui see samm on tehtud, saab mõlemad mängijad stseeni vaates väljaku peale lohistada. Väljakule vedades võib asetada mängijad sinna positsiooni, kus nad peaksid asetsema, kui mäng käima panna. Esmalt peab mängijatele kinnitama põrke tuvastuse, seda saab teha vasakult *Hierarchy* alt valides objekti, millele see kinnitada(player1) ja peale seda *Inspectory* sektsioonist valides *Add Component ->Physics 2D-> Circle Collider 2D*(vt joonis 12).Tekstuuride põrketuvastus raadiuse paneb Unity ise automaatselt paika. Kui tekstuurile on lisatud effektid, saab muuta tekstuuri põrke tuvastusraadiust mängumootori sees.



Joonis 12. Põrke tuvastuse loomine.

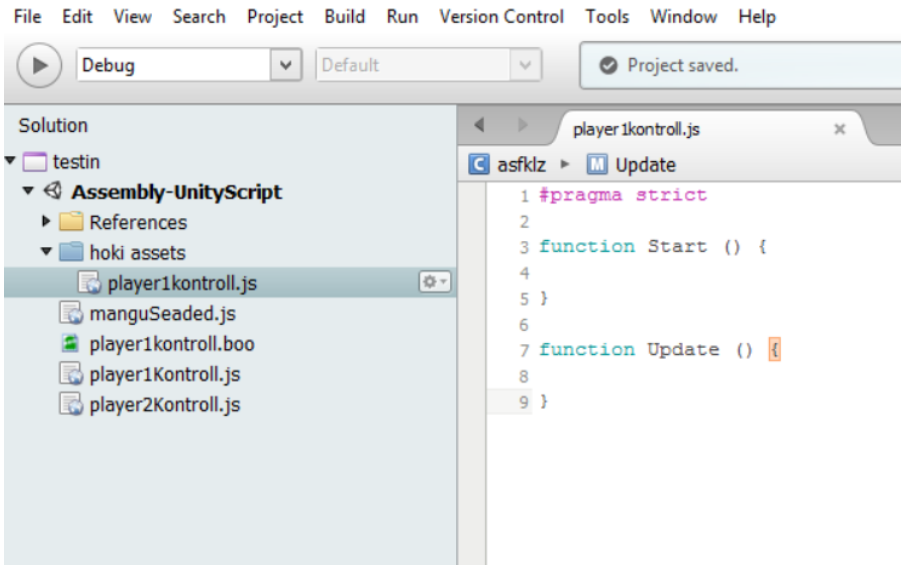
2.6 Klahvide seadmine mängijatele

Selleks, et mängijad liikuma panna klaviatuuri abil, peab tegema uue skripti. Selleks *Hierarchy* all võiks olla valitud objekt, millele koodi kirjutama hakata, hetkel siis *player1*. Skripti tegemiseks võib minna *Inspector* sektsiooni ja sealt vajutada *Add Component ->New Script*. Programmeerimiskeeleks saab valida Javascripti, nimeks võib panna skriptile "player1Kontroll". Lõpetuseks vajutame *Create and Add*. *Inspector* aknas kaks klõpsu "player1Kontroll" peal tehes, peaks avanema "Mono Developer"(vt joonis 14), skripti saab avada ka alt *Assets* kastist.



Joonis 13. Assets sektsioon.

Hetkel skripti juures on vaja ainult funktsiooni *Update*. Koodi kirjutamist peaks alustama muutujate nimetamisega, kuhu objektid liiguvad ja mis kiirusega mängijad liiguvad. Näiteks (var on muutujatele ees javascriptis) liiguYles, alla, paremale ja vasakule. Muutujate järgi tuleks kasutada funktsiooni *KeyCode, Unity* tuvastab selle abil, et objektile määratakse mingi klahv, klaviatuuri pealt. Seejärel saab panna paika mängijate liikumiskiirused. Selle saavutamiseks peab tegema uue muutuja kiirus, mille tüübiks on *float* ja mis oleks võrdne kümnega (vt koodinäide 1).



Joonis 14. Mono developer.

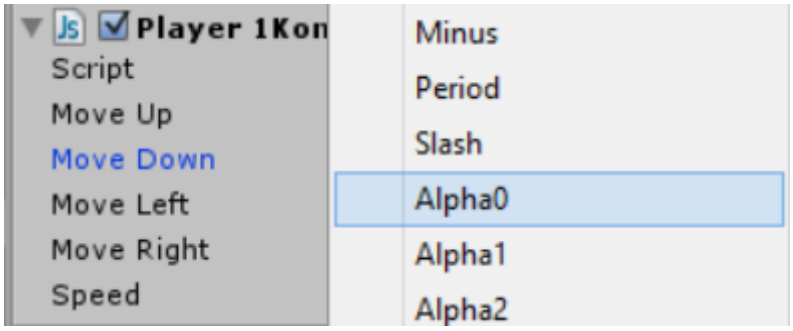
Mängija Unity füüsika mootoriga ühendamiseks saab kasutada funktsiooni rigidbody2D, hiljem on võimalik kinnitada see Unity keskkonnas sees mängija külge. Objekti liikuma saamiseks saab kasutada *rigidbody2D* funktsiooni *velocity* seda võrduma pannes varem määratud kiirusega. Funktsioonis *velocity* on tähtis ära määrata, mis telge mööda mängija liigub, ehk siis kahemõõtmelisel pinnal saab kasutada x ja y telgi. Vastassuunas liikumiseks oleks mõttekas korrutada x ja y teljed läbi miinus ühega. Niimoodi saab kinnitada igale liikumissuunale klahvi. Mängija seisma panemiseks saab kasutada *if* plokki ja *velocity* lõpus võrduma panna nulliga.(vt koodinäide 1).

```
var liiguYles : KeyCode;
	var kiirus : float = 10;
	//liikumissuuna paika panemine
	function Update ()
	{
		if(Input.GetKey(liiguYles))
		{
			rigidbody2D.velocity.y = kiirus;
		}
		else
		{
			//mängija kiirus kui nupule ei vajutata
			rigidbody2D.velocity.y = 0;
			rigidbody2D.velocity.x = 0;
		}

	}
```

Koodinäide 1. Mängija liikumine.

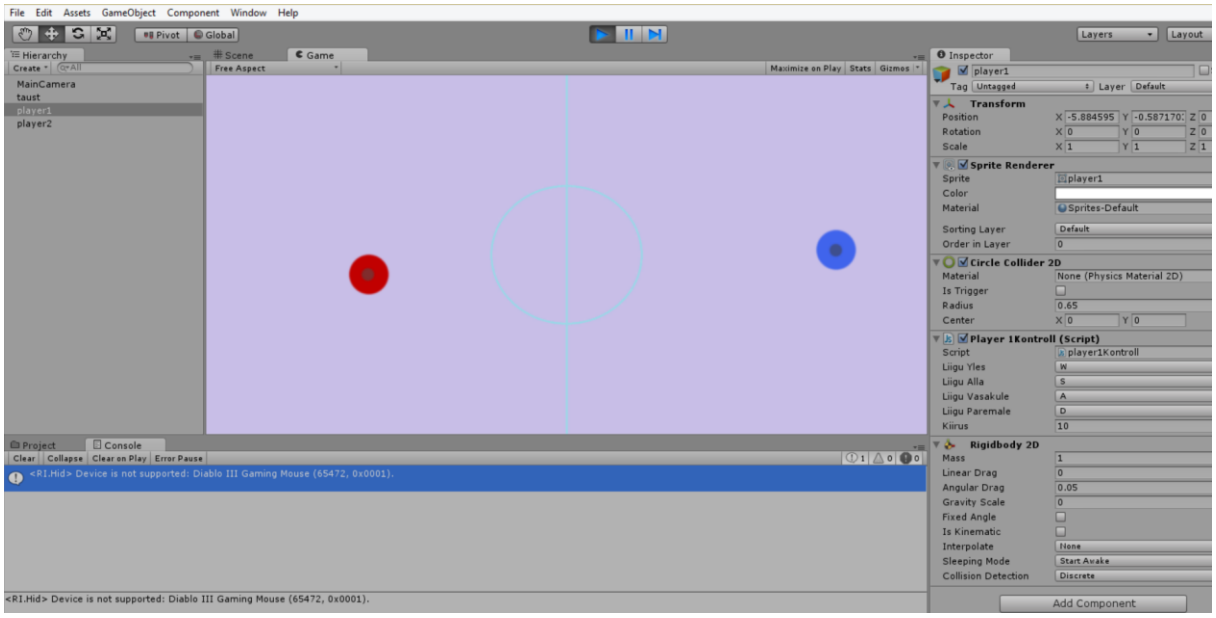
Uuesti Hierarchy kastis *player1*e valides on näha *Inspectory* lahtri all, et skripti on tekkinud muutujad, mis said koodi kirjutamise käigus ära määratud. Käsklus *keyCode* lubab Unity keskkonnas sees objektidele klahve määrata. Nüüd saab rippmenüüst valida esimese mängija kontrollerid(vt joonis 15). Tüüpilisemad klahvid esimese mängija kontrollimiseks on W-üles liikumiseks, S-alla liikumiseks, D-paremale liikumiseks ja A-vasakule liikumiseks.



Joonis 15.Klahvide valimine.

Kui nupud paigas, siis on vaja kinnitada mängijale füüsikalised omadused. Selleks lisame uue komponendi *Rigidbody 2D,Add Component-> Physics 2D-> Rigidbody 2D*(vt joonis 13*)*. Kui see on tehtud peab gravitatsiooni muutma nulliks, et mängija ei hakkaks iseenesest allapoole vajuma. Seda saab teha *Inspectory* alt muutes *Rigidbody 2D* komponendi gravitatsiooni skaala(*Gravity Scale*) väärtuse nulliks.

Mängu käima pannes peaks mängu vaates saama liigutada esimest mängijat. Mängija kahega saab samamoodi teha, ainult klahvid klaviatuuril näiteks noole klahvideks ümber muuta. Järgmisena tuleks teine mängija *spriti'de* kaustast tõsta väljakule kui see veel tegemata. Joonis 16 kujutab lõplikke mängijale külge pandud komponente(vt joonis 16).



Joonis 16. Esimese mängija komponendid.

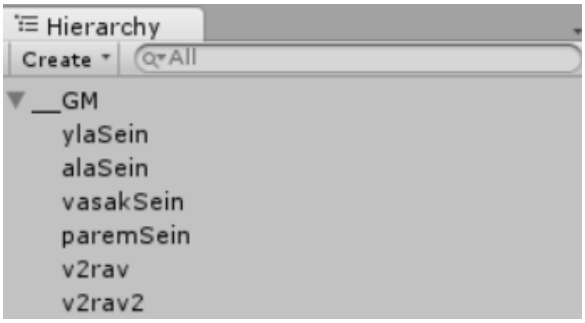
Ülesandeid

1. Pane esimese mängija liikumine igas suunas paika.
2. Tõsta teine mängija platsile ja pane paika.
3. Anna teisele mängijale liikumiseks esimesest mängijast erinevad nupud klaviatuuril.

3 Objektide piiramine kaamera suhtes

Õhuhoki mängu üks keerulisemaid ülesandeid on väljaku piiramine nii et, litter ja mängijad ei saaks liikuda kaamera vaateväljast välja. Selleks kasutame 2DBoxColliderit, et iga seina äärde luua kastikujuline tühi objekt. Vajalik on, et mäng töötaks mängu resolutsiooni muutes, selleks saab kasutada kaamera vaadet ja käsklust *ScreenToWorldPoint*(Asbjørn Thirslund, 2014).

Alustada tuleks uue tühja objekti loomisega. Selleks tuleb valida ülevalt mängumootorist GameObject->New Empty või vajutada ctrl+shift+N. Objekt, mille loome on selleks, et paigutada ära objektid võimalikult kompaktselt. Objekti nimeks võib anda näiteks „*__GM“*. Uue objekti tegemine on vajalik selleks, et saaks kuhugi külge koodi riputada ja hiljem kuskile tühjad objektid panna.



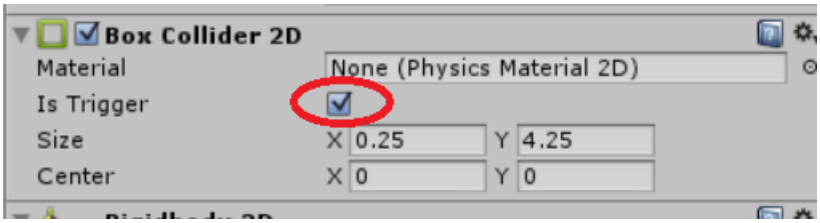
Joonis 17. Objektipuu GM__ .

Väljaku piirete tegemise ja nii mängijate kui väljaku paika seadmisega hakkab mängu alguses tegelema üks skript. Nüüd saab teha uue skripti loodud tühja mängu objekti alla, selle tegemiseks peab valima *Hierarchy* alt __GM(vt joonis 17) ning *Inspectory* alt *Add Component ->New script*. Skripti nimeks võib anda näiteks „manguSeaded“. Pärast „Mono Developeri“ avanemist saab ära kustuda funktsiooni *Start*.

Koodi kirjutamist võib alustada uuesti uute muutujate nimetamisega, kuhu saame panna *BoxColliderid* ehk näiteks muutujad *var* ylaSein, alasein ning vasak ja paremSein. Veel paneme selles skriptis paika väravate algus asukohad ja ka, mängija asukoha kui mäng algab. See tähendab seda, et väravad tuleks samuti vedada *Assets* kasti(tekstuuri nimeks näiteks v2rav) ja sealt stseeni vaatesse.

3.1 Väravate paigutamine

Kui värav on *hierarchy* all, siis tuleks see valida ning sellele kinnitada *rigidbody 2D* ja *boxColliderid 2D*, et hiljem saaks punkte lugeda. Selleks tuleb samamoodi nagu *circleCollideri2D* puhul vajutada *Hierarchy* all v2rava peale ning sealt lisada komponendid(*Add Component->Physics 2D*). Väravad ei tohiks füüsikalist mõju avaldada ei litrile ega mängijatele, sellepärast tuleks valida linnukene, *Is Trigger BoxCollider2D* alt värava puhul(vt joonis 18).



Joonis 18. Värava põrketuvastaja.

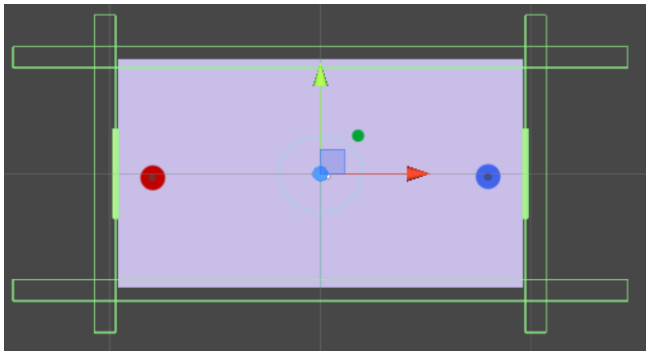
Väravate jaoks on vaja kahte ühesugust objekti, sellepärast on kõige lihtsam kasutada *duplicate* võimalust. Duplikaadi tegemiseks on võimalik valida parema klõpsuga *hierarchy* alt v2rava peale vajutades *duplicate*. Teine võimalus on vajutada CTRL+D klaviatuurilt, mis puhul peab olema objekt valitud, millest duplikaati luuakse. Pärast väravate paigutamist väljaku peale võib edasi minna koodi kirjutamisega, võttes uuesti lahti *Monost „*mänguSeaded“ skripti.

3.2 Algkauguste ja piirete kodeerimine

Järgmisena saab lisada mängu seadete alla ülejäänud objektid, ehk siis loome muutujad mängijate ja värava jaoks. Eesmärgiks on antud juhul, et kui mäng algab on väravad ja mängijad paika pandud määratud positsioonidele. Selleks kasutame Unity funktsiooni *Transform* muutujate taga, see lubab objektide positsiooniga mängida koodis. Ehk siis muutujad nii mängijate ja värava puhul peaks välja nägema midagi sellist. *var Player1 : Transform;*.

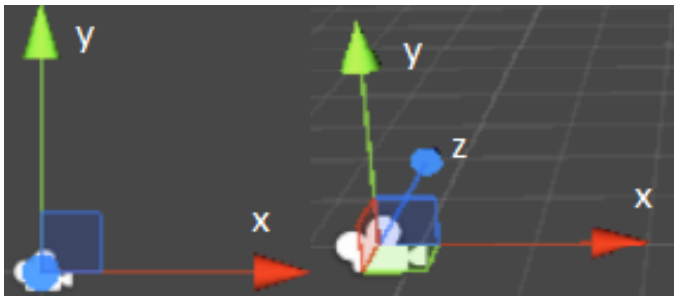
Samamoodi nagu ka värava ja mängijate puhul saab luua muutujad seintele, mille puhul on saab kasutada käsklust *2DBoxCollider*. Välja peaksid seinte muutujad nägema sellised: `var ylaSein: BoxCollider2D;`.

Ülal pool mainituna peame kasutama funktsiooni *ScreenToWorldPoint*, et teha *BoxCollider*, mis oleks täpselt sama suur kui kaamera vaateväljas näiteks ülemine sein(vt joonis 19). See lahendus on kõige parem ka sellepärast, et kui resolutsiooni muudame, jääb mänguväljaku suurus samaks kaamera suhtes. See funktsioon muudab mänguväljaku suuruse sama suureks kui on kaamera vaateväli ekraani peal. *ScreenToWorldPoint* käsklus muudab iseseisvalt põrketuvastuskastid selles skriptis sama suureks kui on kaamera vaateväli mängu suhtes (Asbjørn Thirslund, 2014).



Joonis 19. Objektid stseeni vaate käima panemisel(rohelised tühjad kastid on loodavad tühjad objektid).

ScreenToWorldPoint kasutab ainult *Vector3* funktsiooni, see tähendab seda, et kolmas väärtus ehk z telg peab seda funktsiooni kasutades null olema. Pikslite paika panemisel saab lisaks kasutada näiteks funktsiooni *Screen.width Vector2* ehk kahemõõtmelist skaalat. *Vector2* ja *Vector3* on funktsioonid koordinaatide ära määramiseks stseeni vaate asukoha suhtes pikslite kaupa, näiteks kui kõik kaamerale ette antud arvud funktsioonides on nullid, siis asub objekt stseeni vaates täpselt keskel. *Vector2* tähendab seda, et kaks joont on telje peale ja *Vector3*, et telg koosneb kolmest vektorist (Unity - Scripting API, 2014)(vt joonis 20).



Joonis 20. Vector2(vasakul) ja vector3(paremal) näiliselt kaamera suhtes.

Tänu nendele funktsioonidele ning paarile arvutusele saab panna paika mängijate, väravate asukohad ja seinad, kust mängija ja litter läbi ei saa liikuda. See skript jääbki üleüldiste mängu seadmete jaoks ja nende paika panemiseks (vt koodinäide 2).

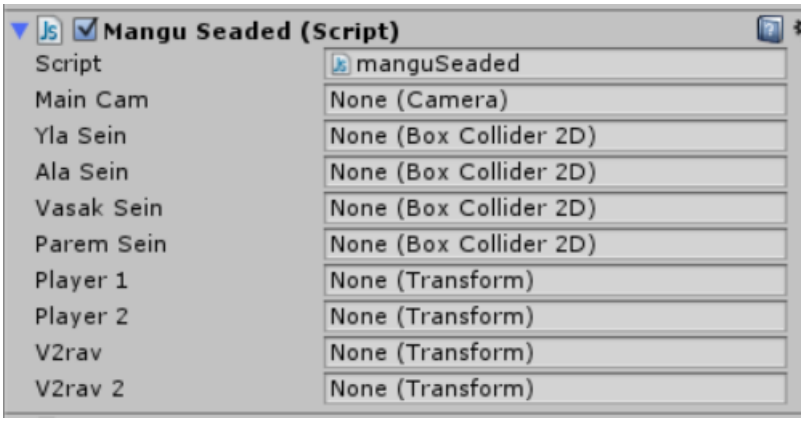
```
var mainCam : Camera;
var ylaSein : BoxCollider2D;
var Player1 : Transform;
var v2rav : Transform;

function Start () {

ylaSein.size = new Vector2 (mainCam.ScreenToWorldPoint (new Vector3 (Screen.width * 2f, 0f, 0f)).x, 1f);
       ylaSein.center = new Vector2 (0f, mainCam.ScreenToWorldPoint (new Vector3 ( 0f, Screen.height, 0f)).y + 0.5f);
Player1.position.x = mainCam.ScreenToWorldPoint (new Vector3 (75f, 0f, 0f)).x;
v2rav.position.x = mainCam.ScreenToWorldPoint (new Vector3 (0f, 0f, 0f)).x;
```

koodinäide 2. seinade, mängijate ja väravate paika panemine mängu alustades.

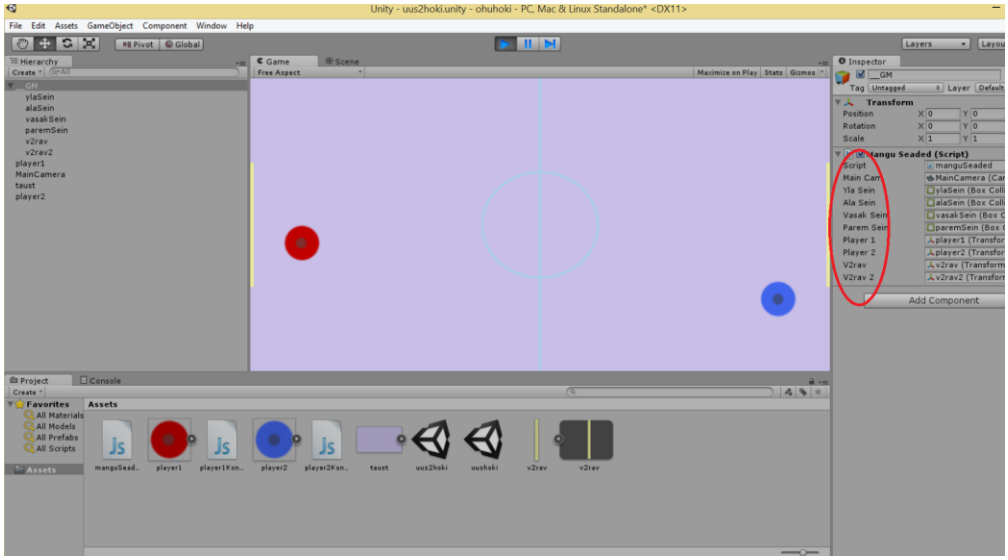
Kui koodiga on korras, siis saab teha uued tühjad põrketuvastajad(*2DBoxCollider*) seinte jaoks. Uusi objekte saab teha *Hierarchy* all, vajutades CTRL+SHIFT+n, siis tekib tühi objekt(Asbjørn Thirslund, 2014). Nimetame tühja objekti ylaSein, alaSein, vasakSein ja paremSein(vt joonis 22). Järgmisena tuleb lisada *2DBoxCollider* samamoodi nagu värava puhul. Peale seda saab teha seintest duplikaadid ning need *Hierarchy* all ümber nimetada.Lõpuks kui see on tehtud, siis tuleks objektid lihtsalt vasaku hiire nupu abil *Hierarchy* all objekti __GM alla vedada.



Joonis 21. Tühjad objektid.

Vajutades __GM peale, on näha, et mängu seadete skripti alla on tekkinud suurel hulgal uusi tühje lahtreid(vt joonis 21). Sinna lahtritesse saab vedada kõik uued objektid parema hiire klahvi abil, et *Unity* saaks aru, mis objektid on koodis kirjas. Kaasa arvatuna tuleks ka

kaamera(*MainCamera*) sinna vedada. Nüüd mängu käivitades peaksid väravad olema täpselt väljaku ääres paigas ning mängijad määratud positsioonil. Mängijaid väljaku äärde liigutades on näha, et nad kaamera vaateväljast välja liikuda enam ei saa. (vt joonis 22).



Joonis 22. Mängu seadete objektid.

Ülesandeid

1. Pane paika kõik seinad väljaku ääres, nii et mängijad ja litter ei saaks väljaku pealt ära liikuda.
2. Pane paika mõlemad mängijad, nii et nad mängu alguses alustaksid mängu võrdselt positsioonilt.

3.3 Keskjoone paika seadmine

Selleks, et antud mängule keskjoon teha tuleb arvestada, et litter peab saama üle selle liikuda, mängijad seda ei tohi saada. Sellest tulenevalt peab mõlemale mängijale piirangud seadma, et nad ei saaks üle keskjoone liikuda. Selleks tuleb avada uuesti player1(player1Kontroll) skript ja hiljem lisada sama funktsioon player2 skriptile. Antud juhul on kõige lihtsam kasutada sellist funktsiooni nagu *Mathf.Clamp*. See käsklus laseb paika panna piirkonna, kus mingi objekti antud koordinaatidest üle liikumise korral viskab see funktsioon objekti tagasi koordinaatide algusesse, mille *transform* käsu all ära märgime. Funktsioon tunneb ära kas mängija on või ei ole ületanud antud koordinaadid kiiruse järgi. Siin mängus hetkel asub kaamera 0,0

koordinaatidel ja objektil(mängijal) on ka oma suurus, siis peaksid funktsiooni koordinaadid olema kuskil 20 ja 0.6 kandis (vt Koodinäide 3) (Unity Technologies, 2014).

```
if(Input.GetKey(liiguYles))
        {
                rigidbody2D.velocity.y = kiirus;
                transform.position.x = Mathf.Clamp(transform.position.x, -20f, -0.6f);
        }
```

Koodinäide 3. mängijate piiramine.

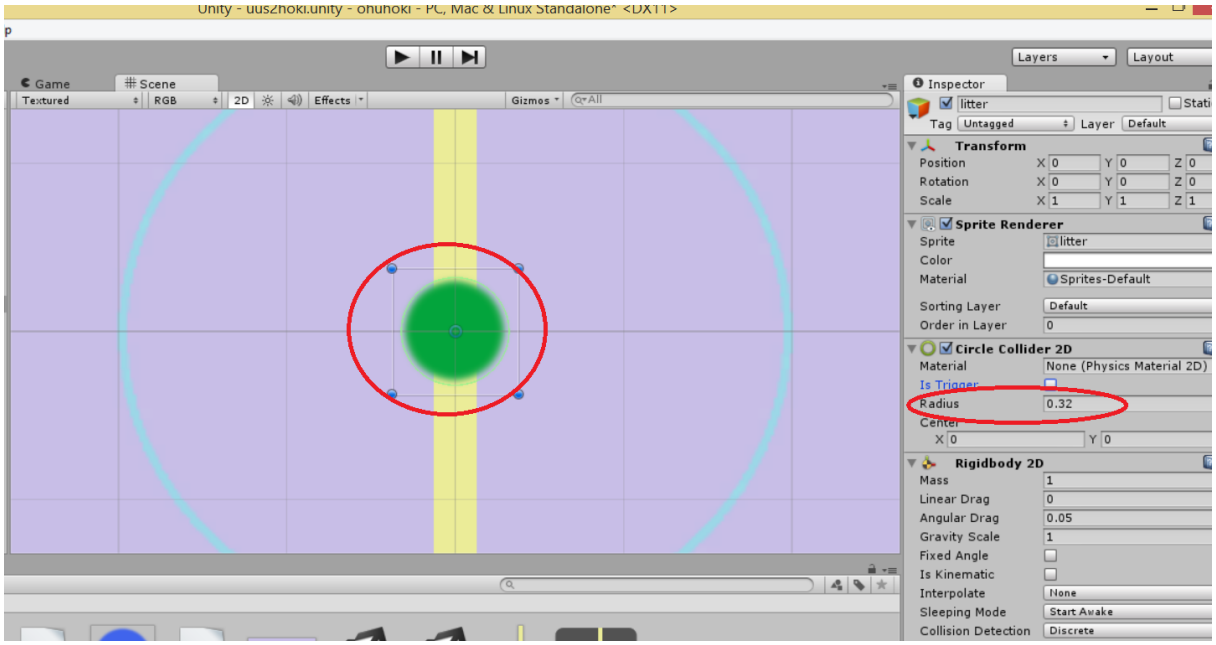
Ülesandeid

1. Vii arvutused lõpuni esimese mängijaga.
2. Tee ümber arvutused teise mängija puhul nii, et ta ei saaks samuti üle väljaku liikuda.

4 Litter ning graafiline liides

Litrit tehes tuleb allalaaditud hoki tekstuuride kaustast vedada Unity *Assets* kasti tekstuur „litter“. Peale seda tuleks teha samamoodi nagu teiste autori poolt tehtud tekstuuridega, kas tõmmata stseeni vaate alt väljakule või vedada hiirega *Hierarchy* sektsiooni. Litrile peaks lisama esiteks *2DCircleCollider* samamoodi nagu mängijate puhul ja samuti *2DRigidBody*.Seda saab teha *Hierarchy* all litri peale vasaku klõpsuga vajutades ning *Inspector* kastist *Add Component->Physics2D->CircleCollider 2D* ja *Add Component->Physics2D->RigidBody 2D.*

Litri puhul on tegemist tekstuuriga, millel on ääres väiksed lisaefektid, sellepärast tuleks raadius *Inspectori* all muuta 0,32-eks. Tekstuuride juures tasub alati jälgida rohelist ringi, mis ümbritseb objekte, selle järgi saab tekstuuride põrketuvastuspiirkonna täpsemalt paika panna(vt Joonis 23) (Asbjørn Thirslund, 2014) .

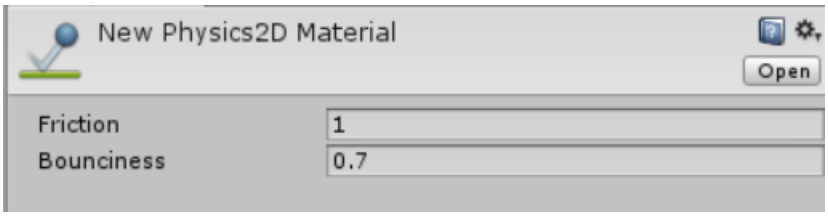


Joonis 23. Tekstuuri raadius.

Veel tuleks muuta gravitatsioon nulliks sellepärast et, õhuhoki puhul on tegemist pealtvaates väljakuga, mille puhul gravitatsioon ei toimi. Ühe peale gravitatsiooni jättes, hakkab litter allapoole vajuma, sest rigidbody2D lisades, arvestab Unity, et maapind asub y telje suhtes kahanevatel väärtustel. Siin mängus on litri all olev väljak tegelikult z-teljel ja gravitatsioon sellepärast litrile ei kehti, sest objekt ei saa läbi väljaku vajuda.

Mängu käigus on näha, et litter asub väljaku peal, aga ei liigu väga seda mängijaga lüües. See on sellepärast, et litril hetkel puuduvad mõned omadused, näiteks põrkevõime ja hõõrdejõud.

Litri põrkama panemiseks tuleb liikuda alla *assets* kasti ja sealt vajutada parema klõpsuga kasti peale, et luua uus materjal *Create->Physics2DMaterial. Hierarchy* all litri peale minnes on näha *CircleCollider 2D* all lahtrit *Material,* kuhu saab vedada tühja materjali. Materjal laseb objektile kinnitada tema põrkamise kiiruse muutumise ehk kui *Bounciness* on objekti juures 1 ja ta põrkab vastu mingit teist objekti, siis tema kiirus ei muutu(vt joonis 24). Selles mängus on vaja, et kiirus kahaneks kui litter vastu mängijat või seina liiguks, sest tegemist on litriga.

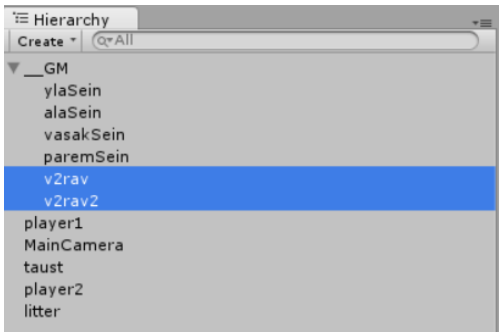


Joonis 24. Materjal.

Friction on materjali juures hõõrdejõud. *Friction määrab* ära, mis suunas litter põrkab kui ta mingit objekti puudutab. Mida suurem *Friction* seda vähem litter muudab oma liikumissuunda kui ta vastu objekti põrkab. Hetkel muudaks põrkamise ligikaudu 0.7-me peale ja hõõrdejõu ühe peale(Unity - Scripting API, 2014).

4.1 Punktide lugemine

Siin õpetuses on kasutusel Unity sisse ehitatud graafilised vahendid skoori laua tegemiseks. Selleks on olemas käsk GUI(Graphical User Interface) mida saab kasutada koodis ning hiljem keskkonnas. Skoori lugemiseks võib teha loenduri skripti *if* lausetega ja selle näiteks lisada __GM objekti alla. Väravate külge peame tegema põrke tuvastus koodi, et loendur saaks aru, kui litter puudutab objekti „v2rav“. Selleks, et seda teha tuleb CTRL klahvi abil valida mõlemad väravad, peale seda saab valida ühe värava *hierarchy* kastis ja siis CTRL klahvi klaviatuuril all hoides saab valida teise värava.(vt joonis 25).



Joonis 25.Väravate valimine.

4.2 Loenduri tegemine

Kui see on tehtud, siis tuleb teha uus skript mõlemale väravale(*Add Component->Add Script->Create and Add*). Skripti nimeks võib panna „varavaSkript“ nagu autori näidetes. Peale seda tuleks teha teine skript, mille võib lisada varem tehtud mänguobjekti alla __GM(vt joonis 25), selle saab teha samamoodi nagu varem(vt peatükk 5) ja nimeks võib panna „skooriLugemine“ ning seejärel avame selle. Esiteks võib ära kustuda mõlemad Unity poolt ette antud funktsioonid, sest siin on vaja teha loenduri jaoks ise uus staatiline(*static*) funktsioon, mida saab kasutada hiljem väravate külge pandud skriptis. *Static* Javascripti puhul tähendab seda, et antud funktsiooni saame kasutada ka mingis teises skriptis. Kui varem on Javascriptiga pistmist olnud, siis loenduri tegemine väga raske ei ole. Unityga saab kasutada stringe(vt joonis 26), et nimetada objekte koodis. Esiteks tuleb teha uued muutujad ja need peab samuti tegema staatiliseks, et neid saaks kasutada. Hoki puhul alustame skoori loendurit nullist, sellepärast

peab muutujad nulliga võrdseks tegema ehk siis antud juhul `static var playerSkoor1 : int = 0;` ja samamoodi tegema teise muutuja, teise mängija loenduri jaoks.

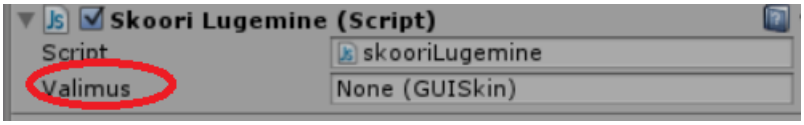
Joonis 26. String Unity keskkonnas, objekti nime saab kasutada stringina.

Loenduri saab teha *stringi* abil, kui tegu on esimese väravaga, siis liidetakse üks muutujatele. Esimese mängija puhul peaks kood siis välja nägema selline(vt koodinäide 4) ja teise mängija puhul võib teha sama(Asbjørn Thirslund, 2014).

```
static function Skoor(v2ravNimi : String) {
//loendur, mis lisab skooridele väärtusi
// esimesele väravale lisatakse 1 punkt
                if (v2ravNimi == "v2rav2")
        {
                playerSkoor1 += 1;
        }
```

Koodinäide 4. väravate lugemine.

Punktide mänguväljakul nägemiseks on võimalus kasutada graafilist kasutajaliidest *GUI* ja kinnitada see skoori külge (*playerScore1*)(vt joonis 27). Selleks tuleb uuesti teha funktsiooni *OnGUI*. Graafilise liidesega saab joonistada väljaku ülesse nurka(vt joonis 28) kaks ristkülikut ning nende sisse võib paigutada üleval tehtud loenduri (Asbjørn Thirslund, 2014).



Joonis 27. OnGUI Unitys.

```
GUI.Label (new Rect (Screen.width/2-150, 25, 100, 100), "mängija1 skoor: " + playerSkoor1);
```

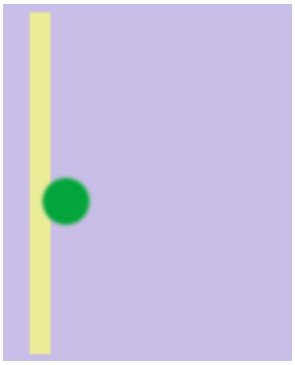
Koodinäide 5. GUI koordinaadid.

Ristküliku joonistamiseks on vaja nelja külge ja funktsioon *Screen.width*'i jagame kahega ning lahutame 150, et ristkülik oleks ligikaudu ekraani keskel. Rea lõpus lisame üleval tehtud muutuja, mis hakkab liidesele punkte loendama.(vt koodinäide 5).

Joonis 28. Loendur mängus.

4.3 Loendurile teisest skriptist objekti liitmine

Hetkel mängu käivitades ja litter väravasse lüües on näha, et punktide lugemise laud on tekkinud, aga punkte ei loendata. See on sellepärast, et loendur ei tea veel, mida ta peaks loendama. Peatüki alguses sai tehtud skript väravatele, nüüd peab lisama funktsiooni, et väravad saaks aru kui litter neid puudutab(vt joonis 29).



Joonis 29. Värava lugemine.

Selleks saab kasutada funktsiooni *OnTriggerEnter2D.* See käsklus tuvastab informatsiooni kui üks objekt liigub teise peale, selleks objektiks tuleb määrata litter ja lisada varem tehtud loendur. Mängumootor veel ei saa aru, et eelmises skriptis loodud muutuja v2ravNimi puhul on tegu objektiga, selle probleemi lahendamiseks saab kasutada käsklust *transform.name(*vt koodinäide 6*).* Peale seda saab *if* ploki ja hitInfo.name funktsioonide abil ära määrata, et loendur töötaks ainult siis, kui litter puutub väravat.

```
#pragma strict
// funktsioon saab kätte kui litter põrkab kokku väravaga
function OnTriggerEnter2D (hitInfo : Collider2D) {
        //saab teisest funktsioonist objekti
        if (hitInfo.name == "litter")
        {
                //lubab kasutada litter objekti stringina
                var v2ravNimi = transform.name;
                skooriLugemine.Skoor (v2ravNimi);
                //võtab litriSkriptist andmed positsiooni ja kiiruse //nulliks seadmiseks
```

```
                hitInfo.gameObject.SendMessage ("ResetLitter");
        }
}
```

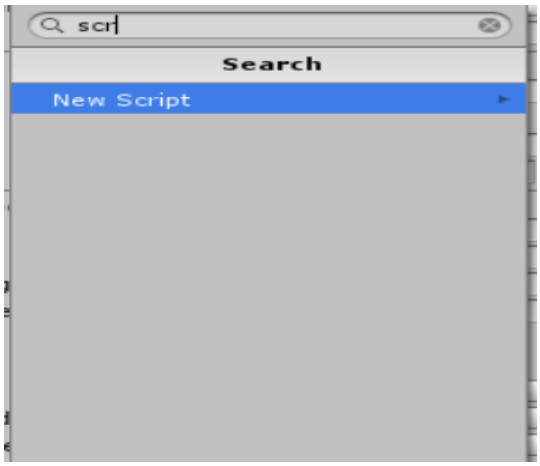
Koodinäide 6. Värava skripti lõplik kood.

Ülesandeid

1. Loo iseseisvalt uus *GUI skin* ja kinnita see skoori lugejale.
2. Muuda GUI *skin*i abil punktide lugemis laua kirja suurust ja värvi.
3. Mine Unity *assetstorei* aadressil https://www.assetstore.unity3d.com/ ning otsi sealt üles erinevate fontide kogum, muuda skoori laua font endale meelepäraseks.

4.4 Litri keskele uuesti toomine pärast värava löömist

Pärast seda, kui üks mängijatest lööb õhuhoki mängus värava, tuleks uuesti keskelt alustada mängu. Teine võimalus oleks võib-olla pärast värava löömist anda litter mängijale, kes värava lasi. Siin mängus kasutame esimest võimalust ja pärast igat väravat toome litri uuesti väljaku keskele, et litter päris staatiline ei oleks, paneme selle liikuma. Esimese asjana peaks tegema uue skripti litrile. See toimub samamoodi nagu eelmiste skriptide puhul, valides *Hierarchy* alt litri ja *Inspectory* alt *Add Component->Add Script->Create and Add(vt joonis 30).*



Joonis 30. Komponenti leidmiseks saab ka otsingut kasutada.

Määrame litrile kiiruse, millega ta mängu alguses või pärast värava löömist liikuma hakkab. Tuleks teha uus muutuja näiteks var „litriKiirus“ ja see panna võrduma hetkel ligikaudu kümnega. Funktsiooni *Update* võib kustutada. Pärast seda saab teha uue funktsiooni nimega

„AlustaPall“, et litter mängijate poolele liikuma panna. Kasutame funktsiooni *Random.Range*, et litter suvalisse kohta mängu alguses või pärast väravat liiguks. Funktsioon *Random.Range* valib suvalise arvu, seda arvu saab kasutada selleks, et suvaliselt valida kuhu poole litter hakkab liikuma. Näiteks võib teha uue muutuja ja see panna antud funktsiooniga võrduma peale seda ütleme, et funktsioon valiks numbri ühe ja nulli vahelt. Käsklus *Random.Range* kasutab *float* numbreid, mis tähendab, et kui kasutada nulli ja ühte loeb ta ka komakohtadega arve. Unitys saame öelda, et kui sama funktsioon võtab suvalise arvu alla 0,5 siis, hakaku litter liikuma ühes suunas ja kui üle 0,5, siis teisele poole.

Litri liikuma panemiseks, saab kasutada mootori enda käsku *AddForce*, mille saab *rigidbody2D*-le lisada. Kasutades *if* lausete blokki saab litri suvalisele poole nii öelda välja lasta mängu alguses ja pärast väravat(vt koodinäide 7) (Asbjørn Thirslund, 2014).

```
function AlustaLitter () {
//valitakse suvaline number ja litter liigub suvalisele //positsioonile
        var randomNumber = Random.Range(0,1);
        if (randomNumber <=0.5) {
                rigidbody2D.AddForce(new Vector2(litriKiirus, 10));
        }
        else {
                rigidbody2D.AddForce(new Vector2(litriKiirus, -10));
        }
}
```

Koodinäide 7. litri välja laskmine.

Liikudes funktsiooni Start alla ja kirjutades sinna samuti eelmise funktsiooni, peaks litter mängu alguses liikuma hakkama. Selle jaoks saab lisada funktsiooni *start* alla uuesti tehtud funktsiooni `AlustaLitter ();`.

4.5 Litri taasalustamine keskelt

Viimaseks saame teha litri keskele viimise jaoks uue funktsiooni, nimetame selle „*ResetLitter*“. Kõige lihtsam on teha kood, mis muudab litri asukoha uuesti 0 koordinaatide peale ning samuti muudab kiirenduse nulliks pärast värava löömist. Selle jaoks tuleb võtta lahti väravate külge kinnitatud skript(„varavaSkript“).

Väravale tehtud koodi uuesti avades on näha, et on kasutatud käsku *hitInfo*. Selle käsu abil saab objektilt kätte informatsiooni kui midagi sellega kokku põrkab, litri puhul on võimalik

kokkupõrke puhul muuta litri koordinaadid ja positsiooni uuesti nulliks. Kiirenduse uuesti nulliks muutmisel saame kasutada käsku *rigidbody2D.velocity.y* ja selle nulliga võrduma panna. Funktsioon *velocity* muudab objekti kiirendust kui see skriptiga objekti külge panna. Litri kiirenduse peab y teljel ära nullima, et litter sama kiirusega nagu enne värava löömist antud teljel ei liiguks (Unity - Scripting API, 2014).

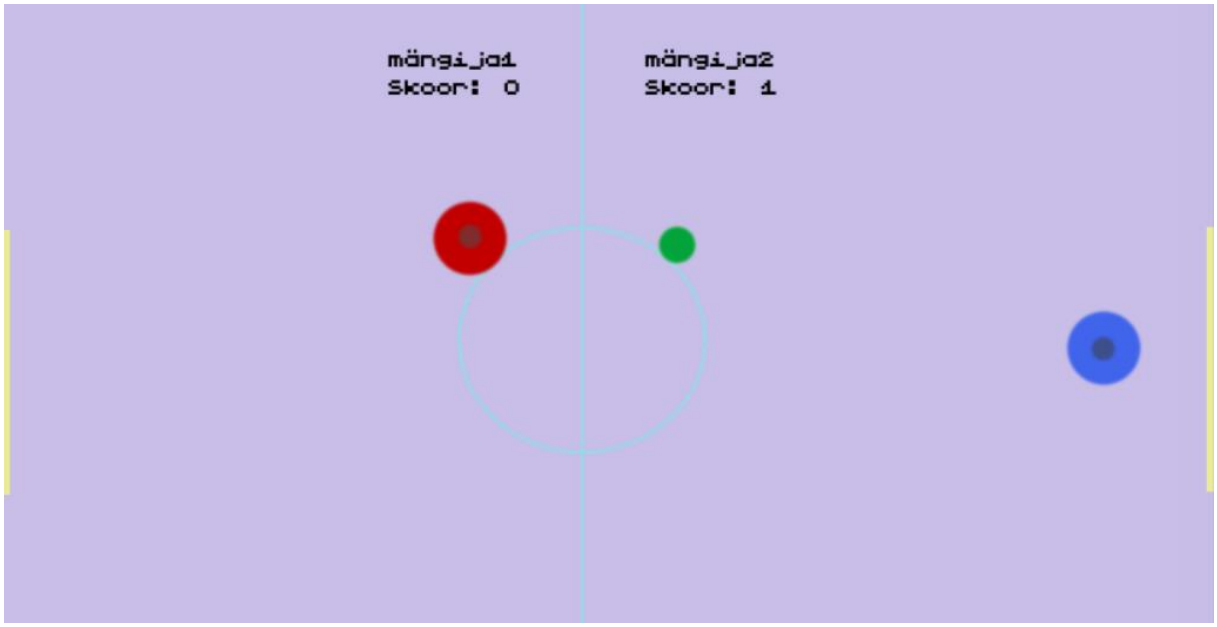
Litri positsiooni mängu algkoordinaatidele saamiseks võib kasutada käsklust *transform.position.x* ja see võrduma panna nulliga. Nulliga just sellepärast, et kaamera on mängu vaates alguses täpselt keskel, see tähendab seda, et väljaku keskpunkt on täpselt null koordinaatide peal. Käsk *transform* muudab objekti asukoha ära määratud koordinaatide peale.

Nagu üleval mainitud, informatsiooni kätte saamiseks peab lisama värava koodi(„varavaSkript“) ka väikese rea, et litter saaks aru kui ta väravat puudub. Selleks kasutame käske *hitInfo.gameObject.SendMessage*(„ResetLitter“). *SendMessage* on Unity käsk, mis laseb kutsuda välja funktsiooni pärast litriga kokkupõrget Stringina(*hitInfo*) (vt lisa 5).

```
rigidbody2D.velocity.x=0;
//muudab positsiooni nulliks
transform.position.x=0;
```

koodinäide 8. Litri keskele liigutamine.

Peale seda on mäng mängitav(vt joonis 31).



Joonis 31. valminud mäng

Ülesandeid

a)

1. Proovi mäng panna kuhugile internetti ülesse ja katseta erinevaid platvorme mängu installeerimiseks.
2. Lõpeta litri taasalustamise skript, nii et litter ei liiguks kuhugi pärast taasalustamist.

b)

Tee ise Ping-Pongi mäng kasutades samu Unity vahendeid:

1. Ping-Pongi puhul oleks erinevus selles, et mängijad peaksid saama liikuda ainult ülesse ja alla, sellepärast nelja liikumissuuna asemel tuleks kasutada kahte.
2. Väljaku piiride tegemine käib samamoodi nagu hoki puhul. Punktide lugemine peaks toimuma näiteks väljaku piirina vasakul ja paremal pool kasutatava *BoxCollideri* abil.
3. Litri liikumine ei tohiks aeglustuda Pongi mängus ja peaks hoopis kiirenema, näiteks kui pall puudutab mängijat. Mängu alguses ei tohiks pall keskele jääda vaid tuleks kuskile suunda välja visata.

Need oleks põhi erinevused kui Ping-Pong teha. Tekstuure võib samu kasutada, mis on juba olemas, värava tekstuur võib olla mängija eest. Tausta võib teha ise valmis kas või *painti* või muud sarnast programmi kasutades võib ka sama tausta kasutada.

4.6 Võimalikke edasiarendusi

Mängu võimalikke täiendusi oleks mitu. Esiteks saaks juurde teha mängule näiteks läbi interneti mitmikmängu võimaluse. Veel oleks võimalik lisada mängule taustamuusika ja litrile efektid kui litter näiteks väravasse läheb või mängija puudutab litrit, siis nende kokkupuute korral kutsuda välja heli.

Mängu tekstuure võib ka parandada. Praegu on tekstuurid mängus suhteliselt madala kvaliteediga ja kuna tegemist on ikkagi Unity keskkonda sissejuhatava materjaliga, ei keskendunud autor väga tekstuuride tegemisele. Selleks saaks kasutada erinevaid pilditöötlus keskkondi nagu näiteks „Gimp“ või „Photoshop“.

Materjali eesmärk oli tutvustada põhifunktsionaalsusi lihtsamas formaadis,sellega anda alus raskemate mängude tegemiseks ja tekitada huvi Unity mootoriga edasi tegelemise vastu. Sellest tulenevalt võib edasi proovida teha iseseisvalt mõnda väiksemat mängu antud funktsioone kasutades või juurde õppida ja minna edasi keerukamate lahenduste juurde.

Kokkuvõte

Seminaritöös tutvustatakse lühidalt Unity *2D* keskkonna vahendeid ja nende kasutamist rakenduse valmis tegemise näol. Seminaritöö on sissejuhatus Unity kahemõõtmeliste tööriistade kasutamisse ja annab keskkonnast ainult väikese ülevaate. Lisaks sisaldab seminaritöö baasteadmisi Javascripti programmeerimiskeele kasutamiseks mängumootoris. Seminaritööd luues omandasin teadmisi mängumootori kasutamisest, õppematerjali loomisest ja Javascripti keele kasutamisest. Õppematerjali loomisega autor varem ei ole kokku puutunud. Kõige raskem koht sellise õppematerjali puhul oli välja mõelda rakendus, mida arendada ja lahti seletada. Samuti on küllaltki tülikas teha tekstuure, kui varem sellise meediumiga tegelenud ei ole.

Seminaritöö põhieesmärk sai täidetud ehk siis lahti seletatud, kuidas mingit väiksemat rakendust Unity mängumootoris 2D vahendite abil, ning kasutades Javascripti programmeerimiskeelt kokku panna. Vähese aja või mõningatel juhtudel piisava kogenematuse tõttu ei olnud võimalik kõiki algselt plaanitud funktsioone rakendusele lisada. Eesmärgiks oli saada õppematerjali lõpuks valmis rakendus, mille abil saab arendada sarnaseid mänge või minna edasi suuremate projektide peale.

Teema valikul lähtusin sellest, mis võiks seminaritöö formaati sobida. Suuremate mängude puhul võib õppematerjali kestvus ulatuda liiga pikaks, sellest tingituna tuli valida väiksema mahuga mäng. Samuti sai lähtutud sellest, et mängude tegemine on üha kasvav trend IT maailmas ning Unity mängumootor on üks populaarsemaid vahendeid mängude tegemiseks.

Õppematerjali sai küll testitud kuid võib-olla mitte piisavalt palju, sellest tulenevalt oleks õppematerjali tegemise ülesannet saanud natuke paremini lahendada.

Kasutatud materjalid

Walker Boys. *Mario 2D Side Scroller*.(2011). Kasutamise kuupäev:14.09.2014.,allikas: Super Mario Clone http://walkerboystudio.com/html/unity_course_lab_4.html

Asbjørn Thirslund.(2014). *How to make a 2D Game*. Kasutamise kuupäev: 24.09.2014., Allikas Introduction to 2D: https://www.youtube.com/playlist?list=PLPV2KyIb3jR4_IYZY2V0G3IUYcx1zZkJe

Asbjørn Thirslund.(2014). *2D Unity 4.3 Course*. Kasutamise kuupäev: 24.09.2014., Allikas Going 2D: http://brackeys.com/preview/2d-course/

Unity Technologies.(2014). *Camera.ScreenToWorldPoint*. Kasutamise kuupäev: 10.10.2014., Allikas Unity documentation: http://docs.unity3d.com/ScriptReference/Camera.ScreenToWorldPoint.html

Unity Technologies.(2014). *Mathf.Clamp*. Kasutamise kuupäev: 20.10.2014., Allikas Unity documentation: http://docs.unity3d.com/ScriptReference/Mathf.Clamp.html

Unity Technologies.(2014). *KeyCode*. Kasutamise kuupäev: 12.10.2014., Allikas Unity documentation: http://docs.unity3d.com/ScriptReference/KeyCode.html

AnswerHub.(2014). *Unity Answers*. Kasutamise kuupäev: 20.10.2014., Allikas Unity Answers: http://answers.unity3d.com/

2DWillNeverDie.(2014). *A basic sprite tutorial*. Kasutamise kuupäev: 18.09.2014., Allikas: http://2dwillneverdie.com/tutorial/a-basic-sprite-tutorial/

Jeff Ward.(2014). *What is a Game Engine?*. Kasutamise kuupäev: 30.11.2014., Allikas: http://www.gamecareerguide.com/features/529/what_is_a_game_.php?page=2

Pile, J., & Jr. (2013). *2D Graphics Programming for Games* (Vol. 0, p. 240)., CRC Press. Allikas: http://books.google.com/books?id=Kuh45Irvt0QC&pgis=1

Unity - Licenses Comparison. (n.d.). Kasutamise kuupäev: 30.11.2014., Allikas: http://unity3d.com/unity/licenses

Vallaste, H. (2013). E-teatmik. Kasutamise kuupäev: 30.11.2014. a., Allikas: http://www.vallaste.ee/.

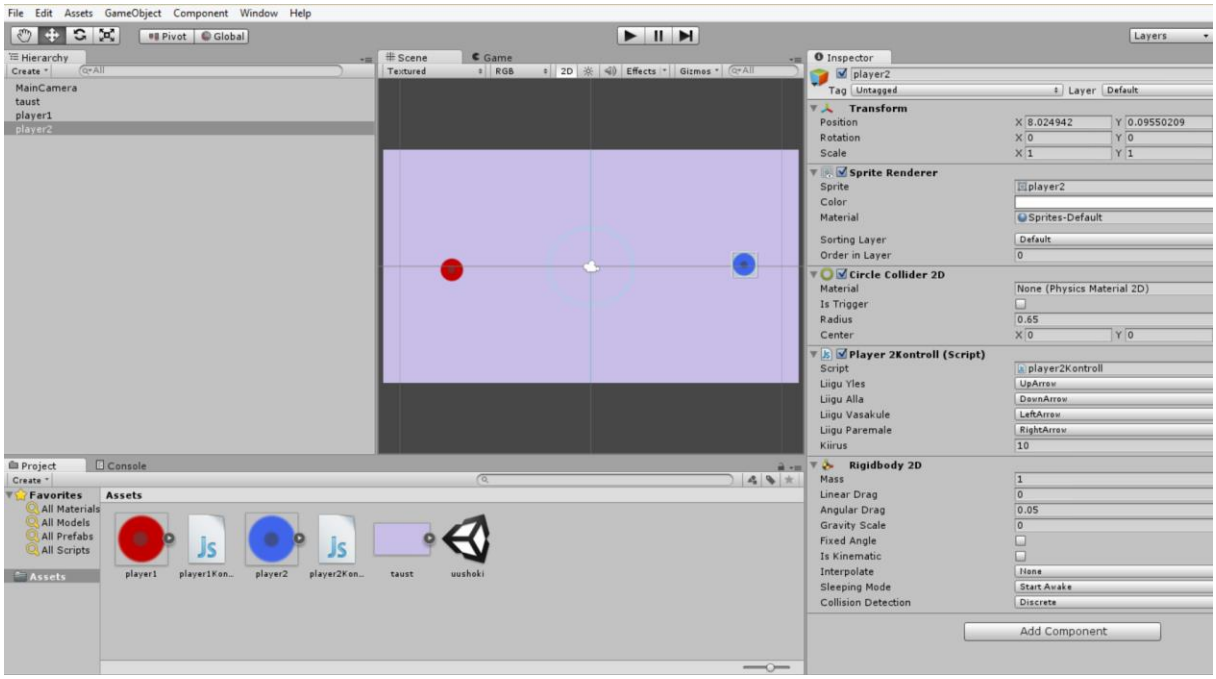
Unity - What is Unity. (n.d.). Kasutamise kuupäev: 31.11.2014., Allikas: http://unity3d.com/pages/what-is-unity

Unity - Scripting API: Vector2. (n.d.). Kasutamise kuupäev: 03.12.2014., Allikas: http://docs.unity3d.com/ScriptReference/Vector2.html

Unity - Scripting API: Physics2DReference. (n.d.). Kasutamise kuupäev: 03.12.2014., Allikas: http://docs.unity3d.com/Manual/Physics2DReference.html

Thorn, A. (2013). *Unity 4 Fundamentals: Get Started Making Games with Unity: Get Started at Making Games with Unity* (p. 312). CRC Press. Allikas: http://books.google.com/books?id=i9JJAgAAQBAJ&pgis=1

Lisad



Lisa 1. mängija 2 objektid.

```
#pragma strict

// nuppude ja suuna määramine

	var liiguYles : KeyCode;

	var liiguAlla : KeyCode;

	var liiguVasakule : KeyCode;

	var liiguParemale : KeyCode;

	//kiiruse määramine objektile

	var kiirus : float = 10;

	//liikumissuuna paika panemine

	function Update ()

	{

		if(Input.GetKey(liiguYles))

		{

			rigidbody2D.velocity.y = kiirus;
```

```
            }
            else if(Input.GetKey(liiguAlla))
            {
                    rigidbody2D.velocity.y = kiirus *-1;
            }
            else if(Input.GetKey(liiguVasakule))
            {
                    rigidbody2D.velocity.x = kiirus *-1;
            }
            else if(Input.GetKey(liiguParemale))
            {
                    rigidbody2D.velocity.x = kiirus;
            }
            else
            {
                    //mängija kiirus kui nupule ei vajutata
                    rigidbody2D.velocity.y = 0;
                    rigidbody2D.velocity.x = 0;
            }

    }
```

Lisa 2. Mängijate liikuma panemine.

```
#pragma strict
```

```
//Nimetab kaamera

var mainCam : Camera;


//Nimetab seinad

var ylaSein : BoxCollider2D;

var alaSein : BoxCollider2D;

var vasakSein : BoxCollider2D;

var paremSein : BoxCollider2D;



// nimetab mängijad ja väravad

var Player1 : Transform;

var Player2 : Transform;

var v2rav : Transform;

var v2rav2 : Transform;

function Start () {


        //Seinade liigutamine väljaku äärtele


        ylaSein.size = new Vector2 (mainCam.ScreenToWorldPoint (new Vector3 (Screen.width * 2f,
0f, 0f)).x, 1f);

        ylaSein.center = new Vector2 (0f, mainCam.ScreenToWorldPoint (new Vector3 ( 0f,
Screen.height, 0f)).y + 0.5f);


        alaSein.size = new Vector2 (mainCam.ScreenToWorldPoint (new Vector3 (Screen.width * 2,
0f, 0f)).x, 1f);

        alaSein.center = new Vector2 (0f, mainCam.ScreenToWorldPoint (new Vector3( 0f, 0f, 0f)).y
- 0.5f);


        vasakSein.size = new Vector2(1f, mainCam.ScreenToWorldPoint(new Vector3(0f,
Screen.height*2f, 0f)).y);;
```

```
        vasakSein.center = new Vector2(mainCam.ScreenToWorldPoint(new Vector3(0f, 0f, 0f)).x -
0.5f, 0f);


        paremSein.size    =    new    Vector2(1f,    mainCam.ScreenToWorldPoint(new    Vector3(0f,
Screen.height*2f, 0f)).y);

        paremSein.center = new Vector2(mainCam.ScreenToWorldPoint(new Vector3(Screen.width, 0f,
0f)).x + 0.5f, 0f);




        //värava, mängija liigutamine positsioonile stseeni kaivitamisel:

        Player1.position.x = mainCam.ScreenToWorldPoint (new Vector3 (75f, 0f, 0f)).x;

        Player2.position.x  =  mainCam.ScreenToWorldPoint  (new  Vector3  (Screen.width  -75f,  0f,
0f)).x;



        v2rav.position.x = mainCam.ScreenToWorldPoint (new Vector3 (0f, 0f, 0f)).x;

        v2rav2.position.x  =  mainCam.ScreenToWorldPoint  (new  Vector3  (Screen.width  -0f,  0f,
0f)).x;


}
```

Lisa 3. mängu seaded.

```
#pragma strict

// määratakse liikumise suunad

        var liiguYles : KeyCode;

        var liiguAlla : KeyCode;

        var liiguVasakule : KeyCode;

        var liiguParemale : KeyCode;

// mängija kiirus

        var kiirus : float = 10;

// mängijatele liikumissuuna määramine nuppudega ja keskjoone piiramine
```

```
function Update ()
{
        if(Input.GetKey(liiguYles))
        {
                rigidbody2D.velocity.y = kiirus;
                transform.position.x = Mathf.Clamp(transform.position.x, -20f, -0.6f);
        }
        else if(Input.GetKey(liiguAlla))
        {
                rigidbody2D.velocity.y = kiirus*-1;
                transform.position.x = Mathf.Clamp(transform.position.x, -20f, -0.6f);

        }
        else if(Input.GetKey(liiguVasakule))
        {
                rigidbody2D.velocity.x = kiirus* -1;
                transform.position.x = Mathf.Clamp(transform.position.x, -20f, -0.6f);
        }
        else if(Input.GetKey(liiguParemale))
        {
                rigidbody2D.velocity.x = kiirus;
                transform.position.x = Mathf.Clamp(transform.position.x, -20f, -0.6f);
        }
        else
        //mängija kiirus kui nupule ei vajutata
        {
                rigidbody2D.velocity.y = 0;
                rigidbody2D.velocity.x = 0;
        }
```

```
        }
```

Lisa 4. player1Kontrollimise täiendamine.

```
#pragma strict

//mängu alguse skoor, millest loendur hakkab lisama

static var playerSkoor1 : int = 0;

static var playerSkoor2 : int = 0;




//funktsioon, mis saab Stringina objekti nime ja loendab //väärtusi juurde

static function Skoor(v2ravNimi : String) {

//loendur, mis lisab skooridele väärtusi

// esimesele väravale lisatakse 1 punkt

                if (v2ravNimi == "v2rav2")

        {

                playerSkoor1 += 1;

        }

//teisele väravale lisatakse punkt

        if (v2ravNimi == "v2rav")

        {

                playerSkoor2 += 1;

        }



}

// Unity Graafiline kasutajaliides, mis kuvab skoori

function OnGUI () {

        GUI.Label (new Rect (Screen.width/2-150, 25, 100, 100), "mängija1 skoor: " +
playerSkoor1);
```

```
        GUI.Label (new Rect (Screen.width/2+150, 25, 100, 100), "mängija2 skoor: " +
playerSkoor2);

}
```

Lisa 5. Lõplik mänguSeaded skripti kood.

```
#pragma strict

// litri liikumise kiirus mängu alguses

var litriKiirus = 10;

// funktsiooni AlustaLitter käivitamine mängu alguses

function Start () {

        AlustaLitter();


}

//liigutab litri kordinaatidele 0 x ja y teljel, samuti muudab //litri kiiruse korraks olematuks

function ResetLitter ()       {

        //muudab kiiruse nulliks

        rigidbody2D.velocity.y=0;

        rigidbody2D.velocity.x=0;

        //muudab positsiooni nulliks

        transform.position.x=0;

        transform.position.y=0;

        AlustaLitter();

}

// liigutab litrit mängu alguses

function AlustaLitter () {

//valitakse suvaline number ja litter liigub suvalisele //positsioonile

        var randomNumber = Random.Range(0,1);

        if (randomNumber <=0.5) {
```

```
                rigidbody2D.AddForce(new Vector2(litriKiirus, 10));
        }
        else {
                rigidbody2D.AddForce(new Vector2(litriKiirus, -10));
        }
}
```

Lisa 6. Lõplik litri skript.