Mobiilipelin kehitys Unity-pelikehitysalustalla

Jani Linnimaa

2021 Laurea

Laurea-ammattikorkeakoulu

Mobiilipelin kehitys Unity-pelikehitysalustalla

Jani Linnimaa

Tietojenkäsittelyn koulutus

Opinnäytetyö

Toukokuu, 2021

Laurea-ammattikorkeakoulu **Tiivistelmä**

Tietojenkäsittelyn koulutus

Tradenomi (AMK)

Jani Linnimaa,

Mobiilipelin kehitys Unity-pelikehitysalustalla

Vuosi 2021 Sivumäärä 36

Opinnäytetyön tavoitteena oli kehittää mobiilipeli Android-puhelimille Unity-pelikehitysalustalla. Pelikehitys valikoitui työn aiheeksi halusta syventää aikaisempaa pelikehitysosaamista.

Kehitysmenetelmänä käytettiin prototypointia, jota varten opinnäytetyön tietoperustassa perehdyttiin kahteen pelikehityksessä usein käytettyyn prototyyppiin. Nämä prototyypit olivat paperiprototyyppi ja digitaalinen prototyyppi. Tässä opinnäytetyön raportissa kehitysprosessin kuvaus rajattiin pelin suunnitteluun ja pelattavan prototyypin kehittämiseen. Kehitystyö alkoi suunnitteluvaiheella, jossa suunniteltiin peli-idea, ansaintalogiikka, pelin päämekaniikka ja pelin ydinsilmukka.

Suunnitteluvaiheen jälkeen aloitettiin pelattavan prototyypin kehittäminen. Prototyyppi sisälsi yhden pelattavan kentän, pelaajan ja pelikameran kontrollit, pistelaskun sekä pelin HUD:in eli peliruudun graafisen käyttöliittymän elementit. Pelattavaa prototyyppiä testattiin Unity-editorissa ja Android-laitteilla. Lisäksi pieni joukko koehenkilöitä testasi prototyypin verkkoselaimessa pelattavaa versiota. Prototyypin valmistumisen jälkeen pelikehitys jatkui siirtymällä tuotantovaiheeseen.

Opinnäytetyön tuloksina syntyi pelattava prototyyppi, josta jatkokehitettiin pelin julkaisuversio. Valmis peli julkaistiin nimikkeellä ”Blaardz” Google Play Kaupassa.

Asiasanat: Mobiilipeli, Pelikehitys, Pelisuunnittelu, Prototyyppi, Unity

Laurea University of Applied Sciences **Abstract**

Degree Programme in Business Information Technology

Bachelor of Business Administration

Jani Linnimaa

Development of a Mobile Game with Unity Game Development Platform

Year 2021 Pages 36

The objective of this Bachelor's thesis was to develop a mobile game for Android phones on the Unity game development platform. Game development was chosen as the subject of the thesis out of a desire to deepen existing game development skills.

Prototyping was utilized as the development method and the knowledge base of the thesis was formed by examining two types of prototypes commonly used in game development. These protypes were paper prototypes and digital prototypes. In this thesis report the description of the development process is limited to the designing of the game and the development of a playable prototype. The development began with a design phase, during which the game idea, monetization, main mechanics and the core loop of the game were outlined.

The development of the playable prototype started after the design phase. The final prototype contained one playable level, player and camera controls, score keeping and the game's HUD, or Heads-up display, which contains the graphical user interface elements of the game screen. The playable prototype was tested in Unity Editor and on Android phones. In addition, a small number of play testers tested a version of the prototype that was played in a web browser. After the prototype was finished, the game development continued in the production phase.

As a result of this thesis, a playable prototype was created and it was further developed into the release version of the game. The finished game was published with the title of "Blaardz" in the Google Play Store.

Keywords: Game Design, Game Development, Mobile Game, Prototype, Unity

Sisällys

1 Johdanto

Opinnäytetyön aiheena on yksinkertaisen kolmiulotteisen mobiilipelin kehittäminen Unity-pelikehitysalustalla Android-laitteille. Aihe valittiin kiinnostuksesta pelialaa kohtaan sekä halusta syventää omaa osaamista pelikehityksestä ja Unitystä. Opinnäytetyö tarjosi tilaisuuden kokeilla peliprojektin saattamista konseptista julkaisuversioon ja antoi näkymän siihen, mitä kokonaisen pelin luominen vaatii.

Opinnäytetyö on toiminnallinen kehitystyö, jossa kehittämismenetelmänä käytettiin prototypointia. Kehitystyön tavoitteena oli luoda julkaisukelpoinen mobiilipeli. Pelikehitysprosessi alkoi suunnitteluvaiheella, jonka tuloksena kehitettiin pelattava prototyyppi. Prototyypistä jatkokehitettiin julkaisuversio, joka julkaistiin Google Play Kaupassa.

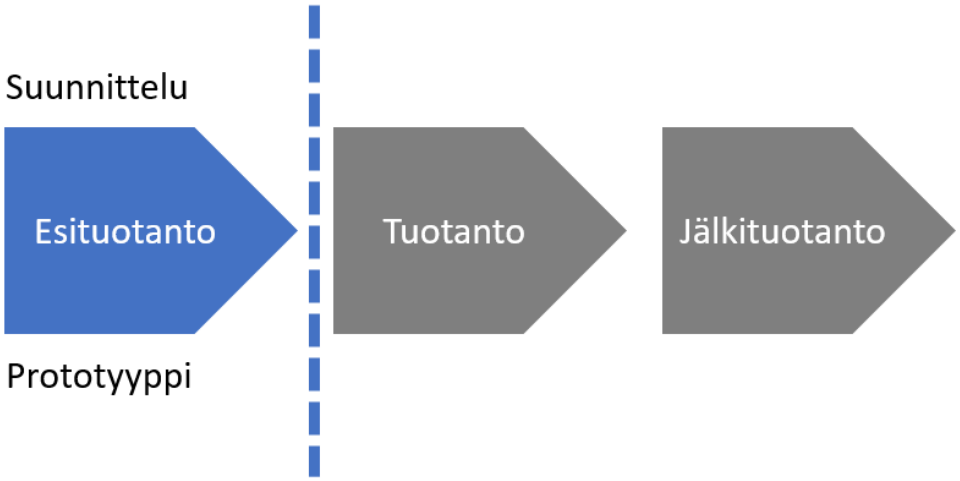
2 Työn lähtökohdat

Opinnäytetyön tavoitteena oli mobiilipelin kehitys Android-älypuhelimille. Työllä ei ollut toimeksiantajaa, vaan projekti sai alkunsa henkilökohtaisesta kiinnostuksesta. Työn aihe on tästä huolimatta ajankohtainen suomalaisen työelämän kannalta. Neogamesin julkaisemassa The Game Industry of Finland -selvityksessä todetaan, että yksi Suomen pelialan suurimpia haasteita on etenkin senioritasoisten kehittäjien puute (The Game Industry of Finland 2019, 20). Samalla pelikehitykseen on nykyään tarjolla laadukkaita kehitystyökaluja, jotka ovat käytännössä ilmaisia harrastelijoille ja pienille pelikehitystiimeille.

Ilmaisten kehitystyökalujen lisäksi kynnystä pelikehittämisen itsenäiseen opiskeluun on madaltanut lukuisat internettutoriaalit, joissa opastetaan kehitystyökalujen käyttöä. Tämän opinnäytetyön laatijan toiveena on, että kehitysraportti ja kehitystyön tuloksena syntynyt peli kannustaisi pelikehityksestä kiinnostuneita yksilöitä tutustumaan kehitystyökaluihin ja kokeilemaan pelien tekemistä.

2.1 Raportin rajaus ja teoria

Tässä raportissa pelikehitysprosessin kuvaus rajattiin esituotantovaiheeseen, johon sisältyi pelin suunnittelu ja pelattavan prototyyppi kehitys. Pelikehitys on sovelluskehitystä, joka voidaan jakaa vaiheisiin eri tavoin. Projektissa sovellettiin mallia, jossa pelikehitys jaettiin kolmeen selkeästi erotettavaan vaiheeseen. Nämä vaiheet ovat esituotanto, tuotanto ja jälkituotanto (Kuvio 1).



Kuvio 1: Pelikehitysprosessin rajaus

Raportin teoriaosuudessa tarkastellaan paperiprototyyppiä ja digitaalista prototyyppiä, jotka ovat pelikehityksessä usein käytettyjä prototypoinnin tapoja.

Raportin toiminnallinen osio koostuu pelin suunnittelusta ja prototypoinnista. Suunnitteluvaiheessa kehitettiin pelin konsepti, ideoitiin pelimekaniikat ja ansaintalogiikka sekä pelin ydinsilmukka. Prototyyppivaihe aloitetaan esittelemällä Unity-pelikehitysalusta, jonka jälkeen kuvataan pelattavan prototyypin kehittäminen suunnitteluvaiheessa syntyneen tiedon perusteella. Kehitysprosessin kuvaus on jaettu osiin aihepiireittäin, mutta käytännössä niiden kehitys tapahtui rinnakkain.

2.2 Keskeiset käsitteet

Assetti:

Assetit ovat pelissä käytettäviä elementtejä, jotka syntyvät kehityksen aikana (Unity Technologies 2021a). Esimerkiksi skriptit, kuvatiedostot, 3D-mallit ja pelin musiikki ovat assetteja.

Blender:

Blender on ilmainen avoimeen lähdekoodiin perustuva 3D-mallinnusohjelmisto. 3D-mallinnuksen lisäksi Blenderillä voidaan tehdä muun muassa 3D-kuvanveistoa, 3D- ja 2D-animaatioita ja videoeditointia. (Blender Foundation 2021.)

HUD:

HUD tulee englannin kielen sanoista heads-up display ja sillä tarkoitetaan pelin aikana ruudulla näkyviä käyttöliittymäelementtejä, kuten pelaajan pisteitä, ammusten määrää tai jäljellä olevia elämiä. (Pluralsight 2014.)

Mikromaksu:

Mikromaksut ovat mobiilipeleissä yleinen ansaintamenetelmä, jossa pelaaja ostaa oikealla rahalla pelin edistymistä nopeuttavia apuja, kuten parempia aseita, lisäelämiä tai pelin sisäisiä resursseja. Mikromaksuilla voidaan myydä myös puhtaasti kosmeettista sisältöä, kuten uusia hahmomalleja ja vaatteita pelihahmoille. Mikromaksuja hyödyntävien pelien lataaminen ja pelaaminen on usein ilmaista. (GameAnalytics 2018.)

Poikkileikkaus:

Poikkileikkaus on prototyyppi, joka sisältää kaikki pelin tiettyyn osioon vaadittavat elementit ja toiminnallisuuden. Esimerkiksi prototypoitava pelikenttä voi olla pelin poikkileikkaus, jos se sisältää pelimekaniikkojen ja HUD:in toiminnallisuuden, kuva- ja animaatioassetit sekä musiikin ja äänet. (Adams & Dormans 2012, 16.)

Skripti:

Unity-pelikehityksessä skriptit ovat lyhyitä C#-kielellä kirjoitettuja koodikokonaisuuksia, joilla lisätään toiminnallisuutta peliin (Unity Technologies 2021b). Skripteillä toteutetaan esimerkiksi pelihahmon liikuttaminen, pisteiden lasku tai uusien vihollisten tuominen pelikentälle.

3 Prototypointi

Pelikehityksessä prototypointi seuraa suunnitteluvaihetta, jossa syntyneen materiaalin pohjalta pelistä kehitetään erilaisia prototyyppejä. Prototyypeillä testataan pelin toiminnallisuutta, käyttäjäkokemusta, pelimekaniikkoja ja ulkoasua. Tämän vaiheen tarkoituksena on selvittää peli-idean toimivuus ja jatkokehityskelpoisuus. (Stefyn 2019.) Pelikehityksessä yleisesti käytettyjä prototyyppejä ovat paperiprototyyppi ja pelattava digitaalinen prototyyppi.

3.1 Paperiprototyyppi

Pelikehityksessä prototypointi alkaa usein paperiprototyypeillä, jotka ovat varsinaista peliä muistuttavia ei-digitaalisia lautapelejä. Paperiprototyypit soveltuvat erityisesti sellaisten pelikonseptien testaamiseen, joiden pelimekaniikat eivät nojaa tarkkaan ajoitukseen tai vaadi raskasta laskentatehoa. Varsinaisten mekaniikkojen lisäksi paperiprototyypeillä voidaan testata pelin muiden osa-alueiden kuten käyttöliittymän, HUD:in ja näppäinsijoittelun toimivuutta. (Adams & Dormans 2012, 17; Despain 2013, 104.)

Paperiprototyypin tekemisen tarkoituksena on saavuttaa ymmärrys peli-ideasta mahdollisimman yksityiskohtaisesti, nopeasti, halvasti ja tehokkaasti (Despain 2013, 104). Adams ja Dormans painottavat (2012, 17-18), että näennäisestä yksinkertaisuudesta huolimatta paperiprototyyppien tekemiseen ei tule suhtautua liian kevyesti: toimivan lautapelin suunnitteleminen vaatii taitoa. Lisäksi ennen prototypoinnin aloittamista on tärkeää selvittää, mitä pelin ominaisuutta halutaan testata, jotta pystytään luomaan tarkoitusta parhaiten vastaava prototyyppi.

Paperiprototyyppien etuja muihin prototyyppeihin nähden on halpuuden lisäksi niiden valmistuksen ja muokattavuuden nopeus sekä matala teknillinen kynnys. Nopea muokattavuus on arvokasta, sillä se mahdollistaa fokuksena olevan mekaniikan iteroinnin jopa pelaamisen aikana. Matala teknillinen kynnys puolestaan sallii niidenkin kehitystiimin jäsenten osallistumisen prototypointiprosessiin, joiden taitoja ei pystytä hyödyntämään digitaalisten prototyyppien tekemisessä. Koska paperiprototyyppi eroaa tuntumaltaan ja ulkoasultaan lopullisesta digitaalisesta pelistä, pystyvät testaajat keskittymään paremmin käyttäjäkokemuksen ja toiminnallisuuden arviointiin jumittumatta siihen miltä prototyyppi näyttää. (Gibson 2015, 128.)

Edellä mainittujen hyvien puolien vastapainona paperiprototypoinnissa on tiettyjä heikkouksia, jotka on hyvä tiedostaa. Paperiprototyyppien suurin heikkous on, etteivät kaikki pelityypit sovellu lautapelimäiseen testaukseen. Esimerkiksi pelejä, joiden pelimekaniikat perustuvat jatkuvaan fysiikan simulointiin, on parempi prototypoida digitaalisesti. Vaikka pelin päämekaniikkoja ei olisi mielekästä testata paperiprototyypeillä, niin niitä voidaan hyödyntää pelin muiden aspektien kuten pelin sisäisen talouden tai pelihahmon kykyjen testauksessa. Toinen heikkous on, että pelitestaajien saattaa olla hankala ymmärtää miten paperiprototyyppi liittyy varsinaiseen peliin. Lisäksi paperiprototyyppi voi vaikuttaa epäammattimaiselta, jos sen suunnitteluun ja toteutukseen ei ole panostettu tarpeeksi. (Adams & Dormans 2012, 17; Despain 2013, 104.)

Gibson (2015, 127) suosittelee seuraavia tarvikkeita paperiprototyyppien tekemiseen:

- Erilaisia noppia ja pelikortteja
- Isoja paperiarkkeja
- Valkotaulua
- Legoja ja piippurasseja
- Muistivihkoja
- Post-it -lappuja

3.2 Digitaalinen prototyyppi

Digitaalisen prototyypin tarkoituksena on saada tarkempi kuva siitä, miltä peli ja sen mekaniikat tuntuvat. Paperiprototyypin perusteella on hankala arvioida pelin lopullista pelattavuutta,

etenkin jos se poikkeaa paljon pelin lopullisesta muodosta. Testaajilta saadaan kerättyä palautetta tehokkaammin paperiprototyyppeihin verrattuna, kun he pääsevät kokeilemaan peliä käytännössä sen sijaan, että kehittäjä selittäisi heille pelin sääntöjä ja mekaniikoita. (Gibson 2015, 420.)

Digitaalinen prototyyppi alkaa yksinkertaisena, mutta pelattavana versiona pelikentästä tai pelialueesta. Prototyypin avulla arvioidaan, onko peliä hauska pelata ja kannattaako sen jatkokehittämiselle uhrata resursseja. Pelattavat prototyypit saattavat sisältää kenttäsuunnitteluelementtejä, hahmoja, aseita, toiminnallista koodia sekä muita elementtejä, jotka ovat tärkeitä peli-idean kannalta. (Despain 2013, 112.) Adams ja Dormans (2012, 17) pitävät ensiarvoisen tärkeänä sitä, että pelin muuttujia on helppo muokata prototyypin sisällä. Kehitystyö nopeutuu, kun esimerkiksi pelimaailman painovoiman tai hahmojen kestopisteiden muuttaminen ei vaadi lisäohjelmointia, vaan pelisuunnittelijat pystyvät säätämään niitä suoraan testaamisen aikana.

Digitaalisten prototyyppien suurin heikkous on, että niiden kehittämisessä kuluu aikaa muita prototyyppejä enemmän. Ne ovat hyödyllisiä tästä huolimatta, sillä testauksesta kertyvän tiedon lisäksi ne auttavat kehitystiimiä keskittymään oikeisiin asioihin. Pelattavan prototyypin avulla projektin ohjelmoijat tietävät mitä mekaniikkoja ja pelielementtejä peliin täytyy vielä kehittää. Samoin kenttäsuunnittelijat pysyvät kartalla siitä mitä heiltä odotetaan ja pelisuunnittelijoilla on ympäristö, jossa he pystyvät kokeilemaan ideoitaan. (Adams & Dormans 2012, 17.)

Rahan ja ajan säästämiseksi pelattavan prototyypin kehittämisessä kannattaa usein hyödyntää ilmaisia pelikehitysalustoja. Vaikka pelin varsinainen kehitys tapahtuisi tuotantovaiheessa kehitystiimin omilla työkaluilla, nopeuttavat Unreal Enginen ja Unityn kaltaiset alustat prototypointia. (Adams & Dormans 2012, 16-17.)

Prosessia voidaan nopeuttaa myös käyttämällä tilapäisiä assetteja. Nämä assetit voivat olla karkeita sijaiselementtejä pelin aseista, hahmomalleista, ääniefekteistä, tai grafiikasta. Näitä assetteja käytetään prototypoinnin aikana ja kehitysprosessin edetessä ne korvataan lopullisilla korkealaatuisilla versioilla. Tilapäisiä assetteja on mahdollista tehdä itse tai ostaa, mutta niitä on myös saatavilla ilmaiseksi internetin eri sivustoilla. Lisäksi pelikehitysalustojen mukana tulee usein yksinkertaisia tilapäisiä assetteja. (Stefyn 2019.)

Kun prototyyppivaihe päättyy, kehittäjien tulee päättää mitä pelattavalle prototyypille tehdään kehitysprosessin edetessä. Pelikehittäjien keskuudessa on asiasta eriäviä näkemyksiä: toiset käyttävät digitaalista prototyyppiä pelin ensimmäisenä iteraationa ja toiset taas aloittavat tuotantovaiheen puhtaalta pöydältä. (Despain 2013, 112.)

4 Pelin suunnittelu

Pelikehitysprojekti aloitettiin suunnitteluvaiheella, jossa ensiksi kehiteltiin peli-idea, pohdittiin pelin kohdeyleisöä ja julkaisualustaa. Suunnittelu jatkui pelin ansaintalogiikan, pelimekaniikan ja pelin ydinsilmukan hahmottelulla. Seuraavaksi suunniteltiin pelin rakenne ja pelaajan liikkuminen pelin sisällä sekä mietittiin alustavasti, mitä assetteja peliin tulisi luoda.

4.1 Peli-idea

Ensimmäinen pidemmälle hahmoteltu peli-idea oli älypuhelimella pelattava 3D-biljardipeli. Pelaajat olisivat pelanneet toisten ihmisten kanssa langattoman verkon välityksellä tai yksinpelinä tekoälyvastustajaa vastaan. Tämä konsepti hylättiin nopeasti, sillä se osoittautui liian kunnianhimoiseksi aloittelijan toteutettavaksi.

Seuraava idea oli riisutumpi versio edellisestä. Biljardi säilyi mukana temaattisesti, mutta konseptista karsittiin pois moninpeli, tekoälyvastustaja ja pallojen lyöminen biljardikepillä. Pelaaja kilpailisi kelloa vastaan yrittäen pussittaa pistepalloja ennen ajan loppumista. Pistepallojen pussittaminen tapahtuisi laukaisemalla pelipallo kimpoilemaan pelialueelle. Jokainen pussitettu pallo antaisi pelaajalle pisteitä, joilla voidaan avata uusia kenttiä pelattavaksi. Tämä konsepti valittiin jatkokehitettäväksi ja se tiivistettiin lauseeseen: ”Biljardista ammentava 3D-peli, jossa pelaaja yrittää pussittaa mahdollisimman monta palloa ennen ajan loppumista”.

4.2 Kohdeyleisö ja julkaisualusta

Seuraavaksi pohdittiin pelin kohdeyleisöä, julkaisualustaa ja ansaintalogiikkaa. Koska tämän projektin tarkoituksena oli ensisijaisesti kartuttaa omaa pelikehitysosaamista, kohderyhmän kartoituksessa ei tehty tarkempaa tutkimusta. Pelin ideaalipelaajaksi ajateltiin pelaajia, jotka pelaavat mobiilipelejä lyhyissä pyrähdyksissä ja nauttivat taitoelementeillä höystetyistä ajanvietepeleistä. Pelistä haluttiin myös tehdä niin helppo omaksua, että nuoremmatkin pelaajat kykenisivät pelaamaan sitä.

Koska projektissa kehitettiin mobiilipeli, täytyi valita mille alustalle se julkaistaisiin. Vaihtoehtoina olivat Android- ja iOS-älypuhelimet, joista lopulta päädyttiin Android-laitteisiin ja julkaisupaikaksi valikoitui Google Play Kauppa. Tärkeimmät syyt Androidin ja Google Play Kaupan valintaan oli kehittäjän suurempi kokemus Android-laitteista ja se, että testausta varten oli saatavilla useampia älypuhelimia.

4.3 Pelin ansaintalogiikka

Pelin ansaintalogiikaksi valittiin mainosrahoitteinen malli. Tässä mallissa pelin lataaminen ja pelaaminen on ilmaista, mutta pelaajille näytetään mainoksia pelin tietyissä kohdissa. Muita

mobiilipeleissä suosittuja ansaintamalleja ovat pelin sisäiset mikromaksut ja pelin myyminen kertaostoksena. Ansaintamalleista mainosrahoitteinen koettiin parhaimmaksi siksi, että pelaajilla ajateltiin olevan matalampi kynnys tutustua uuteen ilmaispeliin kuin kokeilla maksullista peliä. Mikromaksuja sisältävät pelit ovat yleensä myös ilmaisia, mutta mikromaksuja pidettiin sopimattomina tähän projektiin, koska peliin ei suunniteltu pelikenttien lisäksi muuta avattavaa sisältöä.

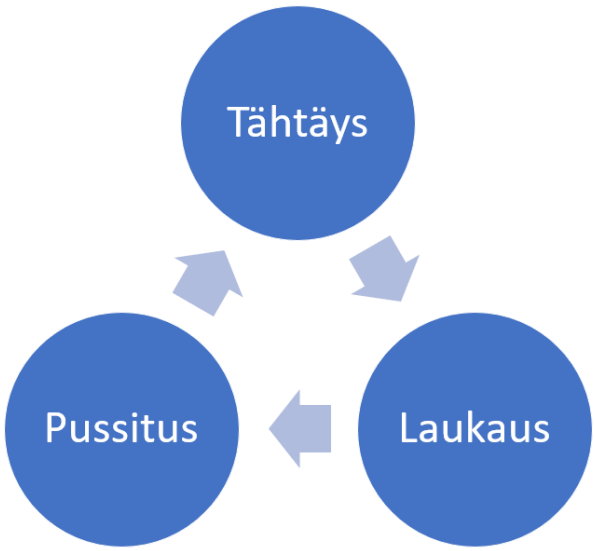
Pelaajille päätettiin näyttää mainoksia joka kolmannen pelikierroksen jälkeen. Heille annettaisiin myös mahdollisuus ansaita kenttien avaamiseen tarkoitettuja pisteitä katsomalla mainoksia vapaaehtoisesti. Tuotantovaiheessa tämä idea jalostui siten, mainosten vapaaehtoisesta katselusta palkittiin suoraan pelikentän avaamisella.

4.4 Pelimekaniikka ja pelin ydinsilmukka

Hyvän pelimekaniikka kehittäminen on yksi pelisuunnittelun tärkeimmistä osioista ja mobiilipeleissä pelaamisen tulisi olla mahdollisimman helppoa ja intuitiivista. Koska pelin konseptina oli pisteiden kerääminen palloja pussittamalla, tarkentui pelin päämekaniikaksi pelipallon laukaiseminen.

Laukaisumekaniikka pyrittiin ideoimaan mahdollisimman yksinkertaiseksi ja lopulta päädyttiin ratkaisuun, jossa tähtäys ja laukaisu tapahtuisivat yhdellä sormella. Pelaaja tähtäisi liikuttamalla sormea ruudulla ja pelipallo laukaistaisiin, kun sormi nostetaan ruudulta. Tässä vaiheessa päätettiin myös, että pallojen pussittamisesta palkittaisiin pisteiden lisäksi lisäämällä pelikelloon aikaa.

Kun laukaisumekaniikka oli selvitetty, voitiin sen ja pelin tavoitteen perusteella hahmotella pelin ydinsilmukka. Ydinsilmukalla kuvataan niitä pelissä toistuvia toimintoja, joista pelaajan pelikokemus muodostuu. Tässä tapauksessa pelin ydinsilmukka sisälsi kolme vaihetta, jotka olivat tähtäys, laukaisu ja pallojen pussitus. Ydinsilmukka on kuvattu Kuviossa 2. Peli sisältäisi myös palkintosilmukan eli uusien kenttien avaamisen pelikierrosten aikana ansaituilla pisteillä.

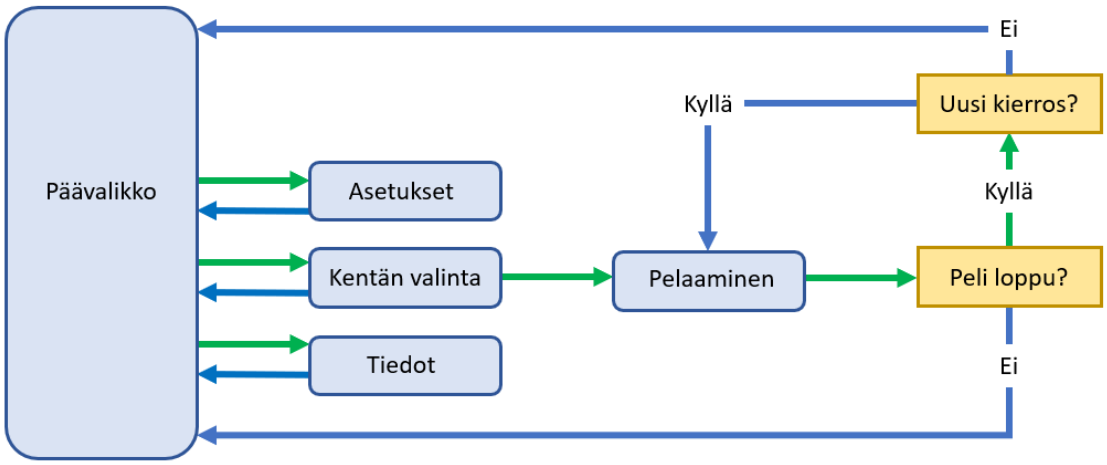


Kuvio 2: Pelin ydinsilmukka

4.5 Pelin rakenne

Pelin rakenteen hahmoteltiin muodostuvan kahdesta osiosta, jotka ovat päävalikko ja pelitila. Pelin käynnistyessä pelaaja siirtyy pelin päävalikkoon. Päävalikosta pelaaja voi aloittaa pelaamisen valitsemalla pelikentän, avaamaan uuden kentän maksamalla pisteitä, siirtyä asetusvalikkoon tai siirtyä tarkastelemaan kehittäjän tietoja. Asetusvalikossa voidaan hiljentää pelin musiikki ja ääniefektit sekä tarvittaessa alentaa pelin ruudunpäivitysnopeutta, jotta ruudunpäivitys pysyisi tasaisena vanhemmillakin mobiililaitteilla.

Pelitilassa peli jatkuu, kunnes aika loppuu, kaikki pistepallot on pussitettu tai pelaajan elämät loppuvat. Pelin päättyessä pelaaja voi aloittaa uuden kierroksen tai siirtyä takaisin päävalikkoon. Taukovalikkoa painamalla pelin saa laitettua tauolle ja siitä pääsee myös takaisin päävalikkoon. Kuviossa 3 on kuvattu pelaajan liikkuminen pelin sisällä.

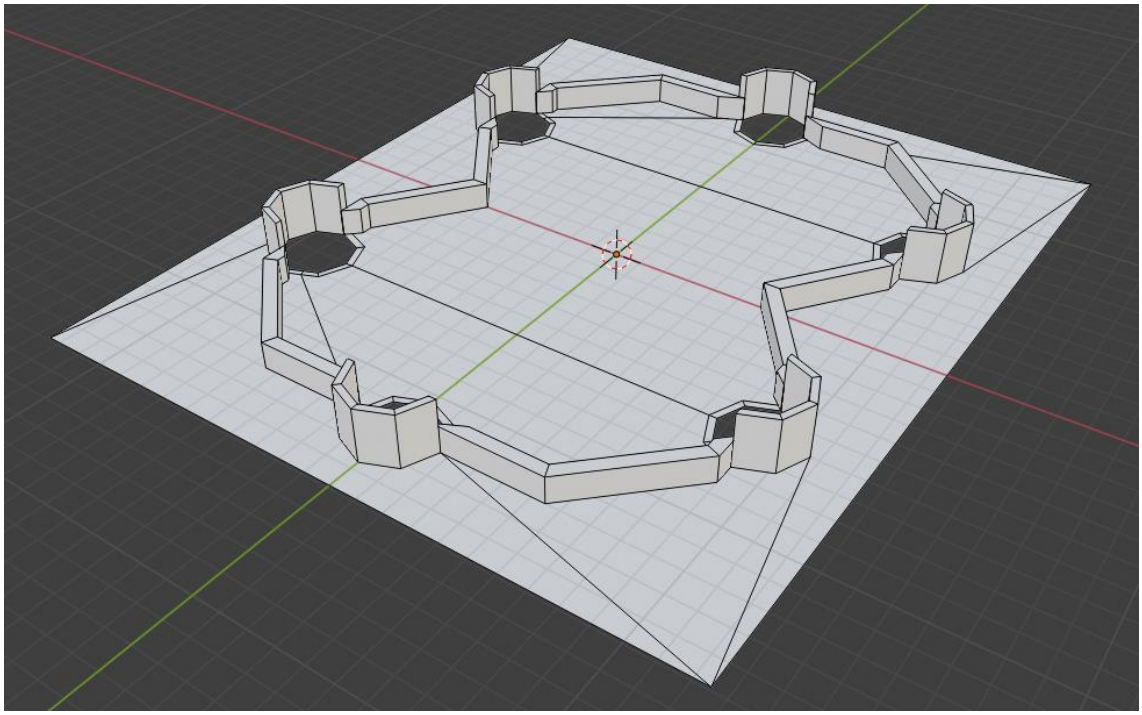


Kuvio 3: Pelaajan liikkuminen pelissä

Pelialueiksi ideoitiin neljä erimuotoista kenttää, jotka sisältävät vaihtelevan määrän pusseja ja mahdollisia esteitä. Koska pelin alkuperäinen innoitus oli biljardi, oli luonnollista, että ensimmäinen kenttä mukailisi biljardipöytää. Vain tämä kenttä olisi aluksi pelattavana uudelle pelaajalle. Seuraavaksi kentäksi suunniteltiin pyöreää, tornien ympäröimää linnanpihaa. Pyöreän kentän ajateltiin tuovan lisää haastetta pallojen pussittamiselle. Kolmanneksi pelialueeksi pohdittiin raunioitunutta rakennusta, jossa olisi vain neljä pussia palloille. Viimeiseksi avattavaksi kentäksi suunniteltiin kasvimaata, jossa olisi kuusi pussia ja esteinä vihanneksia.

4.6 Assetit

Suunnitteluvaiheen lopuksi pohdittiin alustavasti mitä mahdollisia assetteja peli tarvitsisi. Pelin kolmiulotteisuus tarkoitti, että pelikentistä täytyi luoda 3D-mallit. Kuviossa 4 on kuvattu varhainen versio Kasvimaa-kentästä Blender-mallinnusohjelmistossa.



Kuvio 4: Kasvimaa-kentän varhainen 3D-malli Blenderissä

Neljän pelialueen lisäksi peliin päätettiin luoda pienempiä 3D-malleja rekvisiitaksi kenttiä elävöittämään. Alustavaksi rekvisiitaksi suunniteltiin seuraavia 3D-malleja:

- Biljardikeppi
- Juomalasi
- Lautanen
- Torni
- Tiiliskivi
- Kasvimaan aita
- Tomaatti
- Porkkana

Pelissä tarvittaisiin 3D-mallien lisäksi erilaisia kaksiulotteisia graafisia elementtejä käyttöliittymää varten. Näitä olivat ainakin eri valikoiden napit, kenttävalikon ikonit ja HUD:in elementit, kuten peli-, pistepallojen ja taukovalikon kuvakkeet sekä pelikello. Peli tarvitsi myös taustamusiikkia ja ääniefektejä, sillä ne ovat tärkeä osa pelikokemuksen luomisessa. Musiikiksi suunniteltiin lyhyttä ja toistuvaa kappaletta. Pelissä tarvittavia ääniefektejä olisivat pallojen törmäyksistä, pisteiden ansaitsemisesta ja pelipallon pussituksesta aiheutuvat äänet.

5 Pelattava prototyyppi

Prototyyppivaiheessa pelistä kehitettiin suunnittelun perusteella toimiva digitaalinen prototyyppi Unity-pelikehitysalustalla. Unity valikoitui kehitysalustaksi aiemman kokemuksen perusteella. Tämän projektin puitteissa ei tehty paperiprototyyppiä, sillä ne eivät sovellu hyvin fysiikan simulointiin perustuvien kolmiulotteisten pelien prototypointiin. Pelin HUD:in hahmottelemisessa paperiprototyypistä olisi ehkä ollut hyötyä, mutta käyttöliittymän elementit saatiin toteutettua Unityn UI-työkaluilla nopeasti suoraan editorissa.

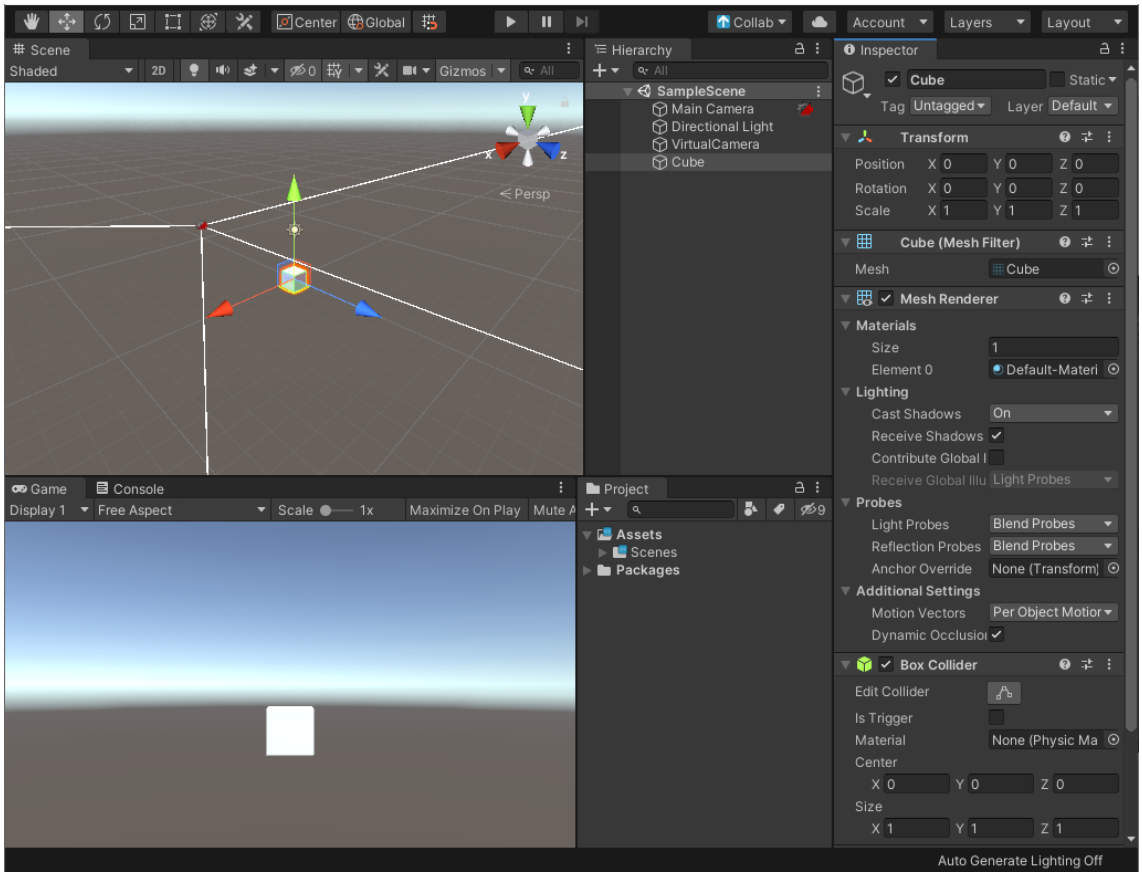
Laajuudeltaan prototyyppi rajattiin yhteen pelattavaan kenttään. Prototyyppi sisälsi pelialueen, peli- ja pistepallot, pelaaja- ja kamerakontrollit, pistelaskun ja päivittyvän HUD:in. Prototyypistä ei tullut aivan täydellistä pelin poikkileikkausta, sillä siitä jätettiin pois pelin pääja taukovalikot, musiikki ja ääniefektit. Seuraavaksi luodaan suppea katsaus Unityyn, jonka jälkeen esitellään prototyypin kehitysprosessin tärkeimmät alueet.

5.1 Unity

Unity on pelikehitysalusta, jolla voidaan kehittää pelejä muun muassa PC- Mac- ja Linux-tietokoneille, Android- ja iOS-laitteille sekä Xbox- ja PlayStation-konsoleille. Unity on ilmainen opiskelijoille ja yksittäisille henkilöille, joiden edeltävän vuoden liikevaihto tai rahoitus jäävät 100 000 dollarin alapuolelle (Unity Technologies 2021c). Ammattilaisille ja isommille tiimeille on tarjolla maksullisia ratkaisuja, joiden vuosimaksu vaihtelee 399 dollarin ja 2000 dollarin välillä.

Pelikehitys tapahtuu Unity-editorissa, jonka näkymä on jaettu ikkunoihin. Ikkunoista tärkeimmät ovat Scene-, Hierarchy-, Inspector-, Game- ja Project-ikkunat. Unity-pelit rakentuvat kohtauksista (Scene) ja niiden sisältämistä peliobjekteista (GameObject). Esimerkiksi pelin aloitusvalikko ja yksittäiset pelikentät voivat olla kohtauksia, ja pelaaja, aseet sekä pelissä kerättävät esineet peliobjekteja.

Scene-ikkunassa liikutaan muokattavan kohtauksen sisällä ja siinä voidaan lisätä ja poistaa peliobjekteja sekä siirtää ja muokata niitä. Hierarchy-ikkuna näyttää kohtauksen peliobjektit listana, jossa voidaan myös lisätä ja poistaa peliobjekteja. Inspector-ikkunassa tarkastellaan peliobjektien ja assettien tietoja sekä muokataan niiden asetuksia. Peliä testataan Game-ikkunassa, jonka saa tarvittaessa koko editorin kokoiseksi. Kehitysprosessin aikana syntyvät assetit, kuten skriptit, animaatiot ja 3D-mallit, sijaitsevat Project-ikkunan Assets-kansiossa. Kuviossa 5 on kuvattu Unity-editori ja tärkeimmät ikkunat.

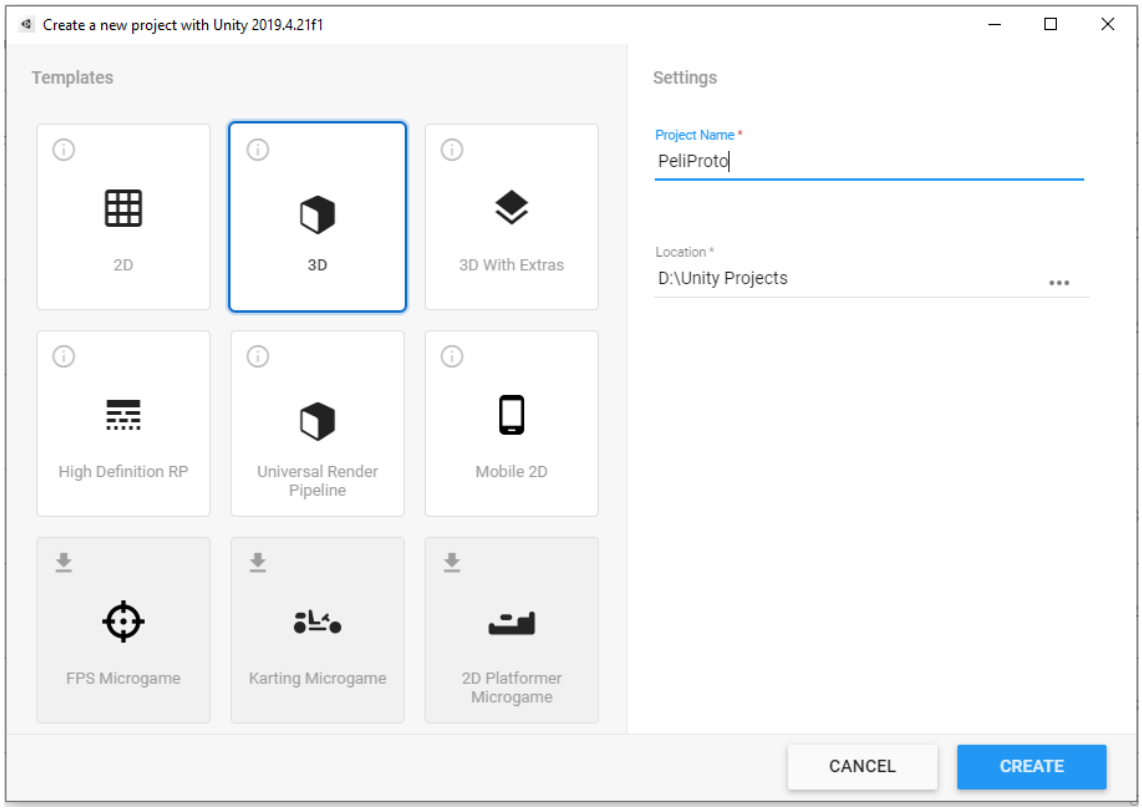


Kuvio 5: Unity-editori

Peliobjekteihin lisätään toiminnallisuutta skripteillä. Aikaisemmin skriptejä voitiin tehdä joko C#-kielellä tai Unityn omalla UnityScriptillä. UnityScriptin tuki on lopetettu, joten nykyään skriptaamisessa käytetään C#-kieltä. C#-skriptien kirjoittamiseen voidaan käyttää Unityn mukana tulevaa Microsoft Visual Studio -ohjelmankehitysympäristöä.

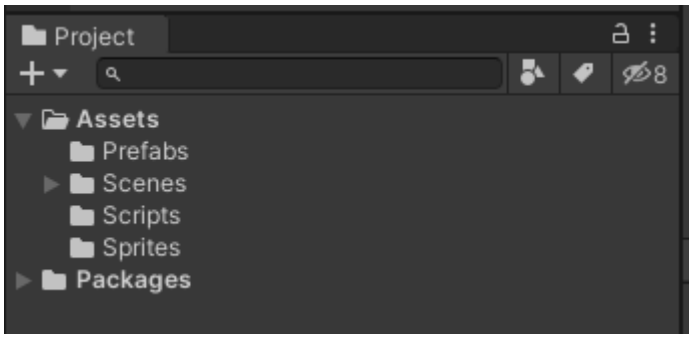
5.2 Projektin luonti ja esivalmistelut

Prototyypin kehitys aloitettiin luomalla Unity Hubissa uusi peliprojekti. Koska projektin tarkoituksena oli kehittää kolmiulotteinen peli, valittiin projektin sapluunaksi ”3D”. Uuden projektin luominen on havainnollistettu Kuviossa 6. Ennen siirtymistä varsinaiseen kehitystyöhön tehtiin Unity-editorissa vielä kaksi projektinhallintaan liittyvää toimenpidettä. Ensimmäinen toimenpide oli projektin alustavan kansiorakenteen hahmotteleminen ja toinen Unityn oman versionhallintapalvelun käyttöönotto.



Kuvio 6: Uuden projektin luominen

Erityyppisille assetteille luotiin Assets-kansioon omat alikansiot, jotta projekti pysyisi hyvässä järjestyksessä. Tämä on hyvä käytäntö pienissäkin projekteissa, sillä hyvin organisoitu rakenne helpottaa kulloinkin tarvittavan assetin löytämistä. Projektin aloitusvaiheen rakenne on kuvattu Kuviossa 7.



Kuvio 7: Projektin alustava kansiorakenne

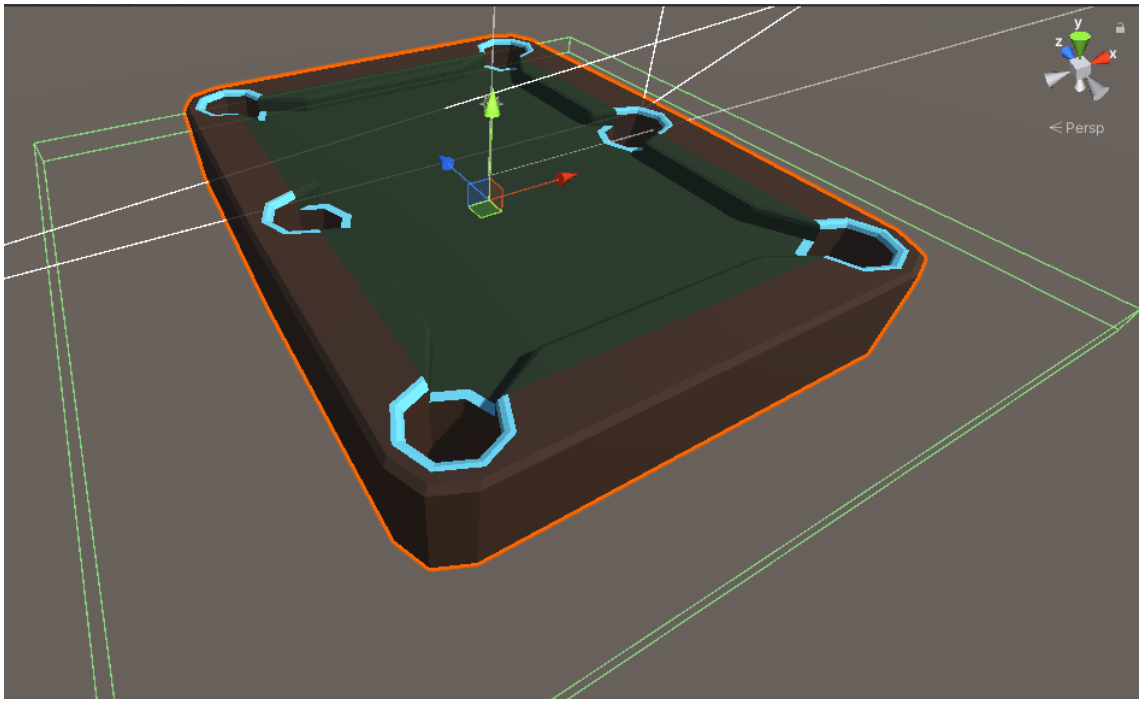
Prototyypin versiohallinnassa käytettiin Unity Collaborate -palvelua. Collaboraten avulla projektiin tehtyjä muutoksia voidaan katsella ja hallinnoida suoraan Unity-editorissa. Jotta versionhallinnan saa käyttöön, täytyy Collaborate aktivoida editorin Services-ikkunan kautta. (Unity Technologies 2021d). Services-ikkunasta voidaan aktivoida myös muita hyödyllisiä palveluita, kuten mainosintegraatio ja moninpeli.

5.3 Pelialue

Esivalmistelujen jälkeen aloitettiin prototyypin varsinainen kehitys. Ensimmäinen askel oli pelikentän luominen. Biljardipöytää mukaileva pelikenttä mallinnettiin Blenderissä, josta se tuotiin Unityyn.

Unityssä kentästä tehtiin peliobjekti, johon lisättiin Mesh Collider-komponentti. Mesh Collider on Collider-komponentti, joka mukautuu 3D-mallin muotoon ja soveltuu monimutkaisten peliobjektien Collideriksi (Unity Technologies 2021e). Pelikenttä vaatii Colliderin siksi, että ilman sitä muut peliobjektit putoaisivat kentän läpi. Collideriin lisättiin Physic Material -assetti, jolla voitiin säädellä kentän pinnan kitka- ja kimpoiluominaisuuksia (Unity Technologies 2021f).

Seuraavaksi pelikenttään lisättiin ScoreArea-lapsiobjekti, johon kiinnitettiin oma Box Collider-komponentti ja ScoreAreaTrigger-skripti. ScoreArean Colliderissa aktivoitiin ”Is Trigger”-ominaisuus mikä saa aikaan sen, ettei Collider-komponentti estä siihen osuvien peliobjektien läpikulkua (Unity Technologies 2021g). Sen sijaan, että muut peliobjektit törmäisivät Collideriin, lähettää se OnTriggerEnter-, OnTriggerExit- ja OnTriggerStay-viestit peliobjektien osuessa tai poistuessa Colliderin alueelta. ScoreArea-objektin tagiksi asetettiin ”ScoreArea”. Pelikenttä ja ScoreArean Collideri on kuvattu Kuviossa 8.



Kuvio 8: Pelikenttä ja ScoreArea-objektin Collider-komponentti

ScoreAreaTrigger-skriptissä toteutettiin pelin pistelaskuun ja HUD-elementtien päivittämiseen liittyvä toiminnallisuus. Skripti sisältää kokonaislukumuuttujan totalBalls, joka pitää kirjaa

siitä, kuinka monta pistepalloa kentällä on jäljellä. Kun Rigidbody-komponentin sisältävä peliobjekti osuu ScoreArean Collideriin, tarkistaa skriptin OnTriggerEnter-metodi törmäävän peliobjektin tagin. Kuviossa 9 on havainnollistettu OnTriggerEnter-metodin sisältö.

```
26      private void OnTriggerEnter(Collider other)
27      {
28          if (other.CompareTag("ScoreBall"))
29          {
30              totalBalls--;
31              UpdateScoreUI();
32              UpdateRemainingBallsUI();
33              AddTime();
34              other.gameObject.SetActive(false);
35              if (totalBalls <= 0)
36              {
37                  Time.timeScale = 0;
38              }
39          }
40          else if (other.CompareTag("Player"))
41          {
42              UpdatePlayerLifeUI();
43          }
44      }
```

Kuvio 9: ScoreAreaTrigger-skriptin OnTriggerEnter-metodi

Jos peliobjektin tagi on ”ScoreBall”, vähennetään totalBalls-muuttujaa yhdellä, jonka jälkeen kutsutaan kolmea metodia. Ensimmäinen metodi lisää 100 pistettä pistesaldoon ja päivittää pisteet HUDi:n. Seuraava metodi päivittää jäljellä olevien pallojen määrän HUD:ssa ja viimeinen lisää 5 sekuntia pelikelloon. Lopuksi pistepallo asetetaan ei-aktiiviseksi. Peli pysähtyy, kun totalBalls-muuttuja saa arvon nolla.

Pelipallon törmätessä ScoreArea-objektin Collider-komponenttiin, vähentää UpdatePlayerLifeUI-metodi yhden pelipalloa kuvaavan elementin HUD:ssa. Tuotantovaiheessa ScoreAreaTrigger-skriptin toiminnallisuus pilkottiin erillisiin skripteihin. ScoreAreaTriggerin metodit on kuvattu Kuviossa 10.

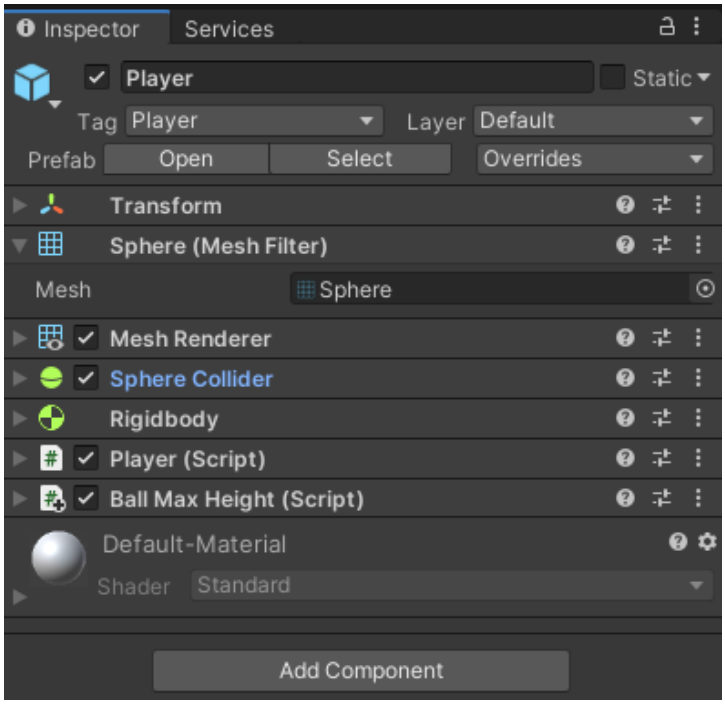
```
46      private void UpdatePlayerLifeUI()
47      {
48          livesArray[lifeCounter].enabled = false;
49          lifeCounter--;
50      }
        1 reference
51      private void UpdateScoreUI()
52      {
53          score += scoreAmount;
54          scoreText.text = "Score: " + score;
55      }
        1 reference
56      private void UpdateRemainingBallsUI()
57      {
58          ballsLeft--;
59          remainingBallsText.text = "x " + ballsLeft;
60      }
        1 reference
61      private void AddTime()
62      {
63          timer.timeLeft += 5;
64      }
```

Kuvio 10: ScoreAreaTrigger-skriptin metodit

5.4 Pelaaja

Pelissä pelaajaa edustaa valkoinen pelipallo. Unity sisältää useita valmiita peliobjekteja, joita voidaan hyödyntää prototypoinnin nopeuttamiseksi. Pelipallon luonnissa käytettiin valmista pallonmuotoista 3D-objektia. Näin toteutetussa peliobjektissa on valmiiksi kiinnitettyinä muutamia komponentteja, joista tärkeimmät ovat fyysisiin törmäyksiin vaadittava Sphere Collider-komponentti ja pallon pelissä näkyväksi tekevä Mesh Renderer-komponentti.

Koska pelin idea perustuu pallojen yhteentörmäyksiin ja niiden kimpoiluun ympäri pelikenttää, täytyi pelipallo saattaa Unityn fysiikanmallinnuksen alaisuuteen. Tätä varten pelipalloon tuli kiinnittää Collider-komponentin lisäksi Rigidbody-komponentti, jonka kautta Unity fysiikanmallinnus liikuttaa palloa (Unity Technologies 2021h). Nyt pelipallo liikkui fysiikanmallinuksen mukaisesti, mutta pallon kimpoilu ei vielä toiminut toivotulla tavalla. Ongelma ratkaistiin lisäämällä myös pelaajan Collideriin Physics Material -asetti. Seuraavaksi pelipalloon lisättiin Player- ja BallMaxHeight-skriptit sekä asetettiin pallon tagiksi ”Player”. Pelipallon komponentit on havainnollistettu kuviossa 11.



Kuvio 11: Pelipallon komponentit

Player-skriptissä määriteltiin pelipallojen eli elämien lukumäärä kierroksen alussa. Prototyypin kehityksen aikana elämien määrä rajattiin kolmeen. Elämien lisäksi Player-skripti sisältää toiminnallisuuden sille, mitä tapahtuu pelipallon pudotessa pussiin ja törmätessä ScoreArea-objektin Collider-komponenttiin. Törmäyksen yhteydessä pelaajalta vähennetään yksi elämä ja jos pelaajalla on törmäyksen jälkeen vielä elämiä jäljellä, kutsutaan Respawn-metodia (Kuvio 12). Respawn siirtää pelipallon takaisin kentälle ja asettaa pallon vauhdin nollaan. Elämien loppuessa peli pysähtyy.

```
35      private void Respawn()
36      {
37          transform.position = startPosition;
38          playerBody.velocity = Vector3.zero;
39          playerBody.angularVelocity = Vector3.zero;
40      }
41  }
```

Kuvio 12: Pelaajan Respawn-metodi

Prototyypin testauksen aikana kävi ilmi, että peli- ja pistepallot kimposivat helposti pelikentän ulkopuolelle. Jossain toisessa pelissä pallojen pomppinen voisi olla toivottava ominaisuus, mutta tässä projektissa niiden tuli pysyä pelikentällä, jotta pelaaja pystyisi pussittamaan ne. Ongelma ratkaistiin luomalla Kuviossa 13 kuvattu BallMaxHeight-skripti, joka rajoittaa pallon liikkumista pystysuunnassa. Skriptin Update-metodi vertaa pallon sijaintia määritettyyn maksimikorkeuteen, jonka ylittyessä skripti siirtää pallon takaisin halutulle korkeudelle.

```
8    private void Update()
9    {
10       if (transform.position.y > ballMaxYPosition)
11       {
12           transform.position = new Vector3(transform.position.x, ballMaxYPosition,
13           transform.position.z);
14       }
15   }
```

Kuvio 13: BallMaxHeight-skriptin Update-metodi

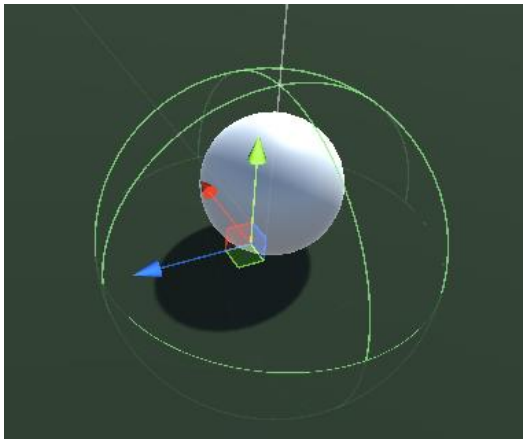
Pistepallot luotiin kopioimalla pelipallo. Pistepalloista poistettiin Player-skripti, niiden tagiksi vaihdettiin ”ScoreBall” ja niistä tehtiin punaisia lisäämällä niihin uusi Material-assetti.

5.5 Pelipallon laukaisumekaniikka

Suunnitteluvaiheessa pelin päämekaniikaksi valikoitui pelipallon laukaisu. Jotta peli olisi helposti lähestyttävä eri ikäisille ja myös vähemmän kokeneille pelaajille, haluttiin laukaisumekaniikka kehittää mahdollisimman yksinkertaiseksi. Toiminnallisuus hioutui sellaiseksi, että mobiililaitteella palloa kontrolloitiin yhdellä sormella.

Pallon laukaisu tapahtuu kolmessa vaiheessa: ensin pelaaja koskettaa pelipalloa sormella, sitten sormea liikutetaan kosketusnäytöllä ja lopuksi sormi nostetaan näytöltä. Kun sormi nostetaan ruudulta, syöksyy pelipallo vastakkaiseen suuntaan kuin sormea vedettiin.

Laukaisumekaniikan kehittämisestä tuli prototyypin monimutkaisin osio. Kehitys aloitettiin luomalla kolme tyhjää peliobjektia: LaunchArea, PointA ja PointB. LaunchArea-objektiin lisättiin Line Renderer -komponentti, Sphere Collider -komponentti ja laukausta kontrolloiva LaunchController-skripti. LaunchArea-objekti kulkee pelipallon mukana ja sen Collider-komponentti reagoi pelaajan kosketukseen. Jotta ohjaus ei olisi liian pikkutarkkaa, tehtiin Colliderista huomattavasti pelipalloa suurempi (Kuvio 14).



Kuvio 14: LaunchArean Collider verrattuna pelipalloon

PointA- ja PointB-peliobjektit ovat 3D-objekteja, jotka avustavat pallon suuntaamisessa. PointA-objektiin lisättiin PointerFollow-skripti, jolla objekti seuraa pelaajan sormen liikettä. Kuviossa 15 on kuvattu skriptin LateUpdate-metodi.

```
0 references
void LateUpdate()
{
    temporaryPosition = Camera.main.ScreenToWorldPoint(new Vector3(
    Input.mousePosition.x,Input.mousePosition.y, 10));

    transform.position = new Vector3(temporaryPosition.x, yOffset,
    temporaryPosition.z);
}
```

Kuvio 15: PointerFollow-skriptin LateUpdate-metodi

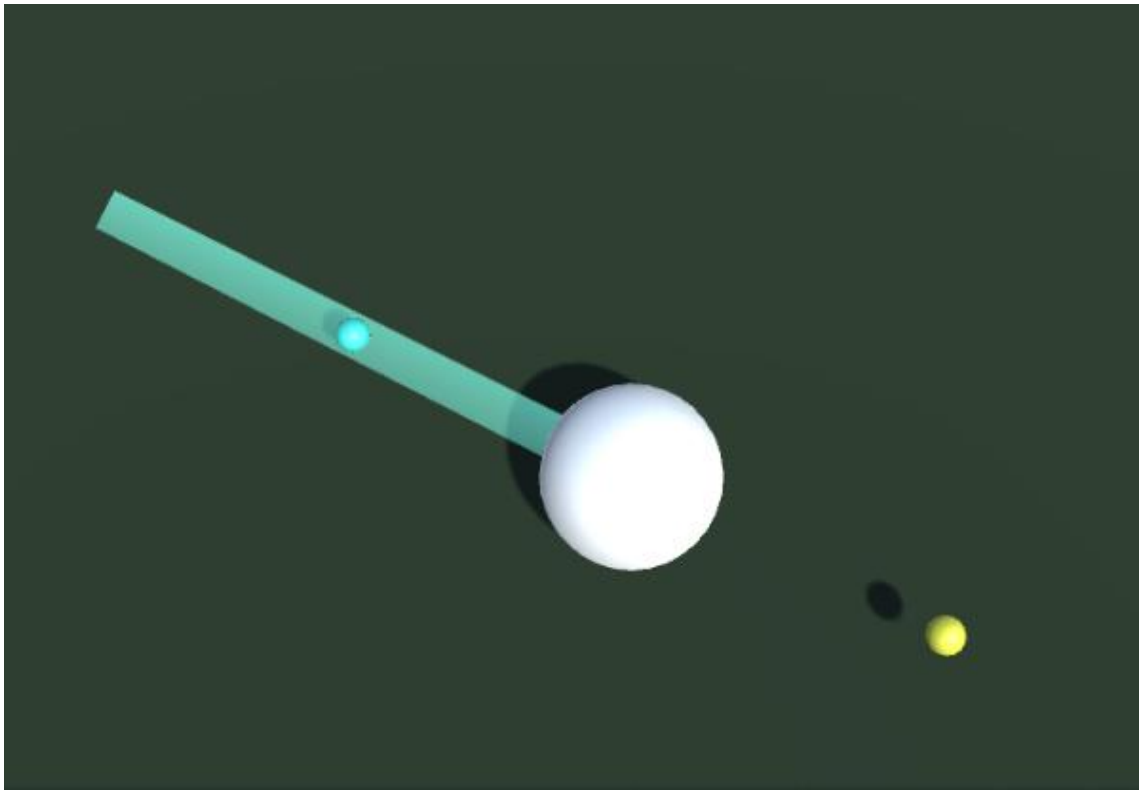
PointB-peliobjekti liikkuu PointA:n peilikuvana pelipallon vastakkaisella puolella. Pelipallon laukaisun suunnan määrittää LaunchController-skriptissä launchDirection Vector3-muuttujan arvo, joka saadaan suorittamalla Vector3.Normalize-metodi. Vector3.Normalize-metodille annetaan parametriksi PointB:n sijainti, josta vähennetään pelipallon sijainti (Kuvio 16).

```
83      launchDirection = Vector3.Normalize(pointB.transform.position -
84      transform.position);
```

Kuvio 16: Laukaisun suuntaus

Laukauksen suunta näytetään pelaajalle piirtämällä ruudulle LaunchArea-objektin Line Renderer-komponentilla viiva, joka kulkee pelipallon keskipisteestä PointB:n läpi. Viivan pituus

indikoi laukauksen voimakkuutta. Kuvio 17 näyttää Unityn Play-tilassa, kuinka laukaisun tähtäys tapahtuu keltaisen PointA:n, sinisen PointB:n ja Line Rendererin yhteispelillä.



Kuvio 17: PointA, PointB ja Line Renderer -tähtäysviiva

Tuotantovaiheeseen siirryttäessä PointA:n ja PointB:n Mesh Renderer-komponentit kytkettiin pois päältä, jotta avustavat objektit eivät näkyisi pelaajan ruudulla.

Laukaisun voima saadaan kertomalla LaunchController-skriptissä määritelty launchMultiplier-muuttuja pelipallon ja PointA-peliobjektin välisellä etäisyydellä. Mitä suurempi etäisyys, sen kovempaa pallo laukaistaan. Lisäksi LaunchController-skriptissä on määritelty muuttuja maxPullDistance, joka asettaa etäisyydelle ylärajan. Kun tämä raja ylitetään, etäisyyteen perustuvaa laukaisuvoimaa ei enää kasvateta. Etäisyyden ylittäessä maxPullDistance-muuttujan arvon, laukaisuun lisätään etäisyyskertoimen lisäksi ennalta määritellyn arvon verran extra-voimaa (Kuvio 18).

```
58          if (currentDistance <= maxPullDistance)
59              launchArea = currentDistance;
60          else
61              launchArea = maxPullDistance;
62
63          if (launchArea == maxPullDistance)
64              launchPower = Mathf.Abs(launchArea) * (launchMultiplier + 5);
65          else
66              launchPower = Mathf.Abs(launchArea) * launchMultiplier;
```

Kuvio 18: Laukauksen voiman laskeminen

Pelaajan nostaessa sormensa ruudulta tapahtuu laukaisu, joka toteutetaan prototyypissä OnMouseUp-metodilla. Android-laitteilla OnMouseUp-metodi vastaa sormen nostamista ruudulta. OnMouseUp on tarkoitettu nimensä mukaisesti hiiriohjaukseen, eikä sitä ole optimoitu mobiilikäyttöön. Prototyypin tarpeisiin kuitenkin riitti, että laukaisumekaniikka saatiin toteutettua nopeasti ja että se toimi testilaitteissa. Tuotantovaiheessa OnMouseUp-metodi korvattiin kosketukseen perustuvalla ohjauksella. Laukaisun OnMouseUp-metodi on kuvattu kuviossa 19.

```
84      private void OnMouseUp()
85      {
86          lineRenderer.enabled = false;
87
88          Vector3 launch = launchDirection * launchPower;
89          playerBody.AddForce(launch, ForceMode.Impulse);
90      }
```

Kuvio 19: Pelipallon laukaisu OnMouseUp-metodilla

5.6 Kamerakontrollit

Projektissa haluttiin, että kamera seuraisi pelipalloa takaapäin yläviistosta ja että pelaaja pystyisi kääntämään kameraa. Ratkaisuun päädyttiin kahdesta syystä. Ensinnäkin pallon mukana liikkuva kamera tuo peliin vauhdin ja toiminnan tuntua. Toiseksi sopivan etäältä seuraava ja käännettävä kamera auttaa pelaajaa hahmottamaan ympäristöä ja tähtäämään laukauksia paremmin.

Kamerakontrollien toteuttamisessa hyödynnettiin Unityn Cinemachine-pakettia, joka nopeutti kontrollien kehitystä. Cinemachine-paketti ladattiin ja asennettiin editorin Package Manager-ikkunasta. Jotta Cinemachine toimisi, täytyi kohtaukseen MainCamera-peliobjektiin liittää CinemachineBrain-komponentti. Tämä komponentti monitoroi kaikkia kohtauksessa olevia virtuaalikameroita, jotka näyttävät pelin tapahtumat pelaajalle (Unity Technologies 2021i). Koska tässä pelissä seurataan vain yhtä peliobjektia, riitti kohtaukseen yksi virtuaalikamera.

Virtuaalikamera tehtiin luomalla tyhjä peliobjekti, johon kiinnitettiin CinemachineVirtualCamera-komponentti ja CameraController-skripti.

Kuten pelipallon laukaisumekaniikan, myös kamerakontrollien tuli olla mahdollisimman yksinkertaisia. Kontrollit päätettiin toteuttaa siten, että pelaaja kääntää kameraa liikuttamalla kahta sormea vaakatasossa älypuhelimen ruudulla. Näin pelin pelaaminen vaatisi vain kahta erilaista kosketusta: yhdellä sormella laukaistaan pelipallo ja kahdella sormella käännettään kameraa. Kuviossa 20 on esitetty kameran kääntäminen mobiililaitteella CameraController-skriptin LateUpdate- ja TurnCamera-metodeilla. Kun peliä testattiin Unity-editorissa, kameran kääntäminen tapahtui painamalla oikea hiirennappi pohjaan ja liikuttamalla hiirtä.

```
27      void LateUpdate()
28      {
29          foreach (Touch touch in Input.touches)
30          {
31              if (touch.phase == TouchPhase.Began)
32              {
33                  initTouch = touch;
34              }
35              else if (Input.touchCount >= 2 && touch.phase == TouchPhase.Moved)
36              {
37                  TurnCamera();
38              }
39              else if (touch.phase == TouchPhase.Ended)
40              {
41                  initTouch = new Touch();
42              }
43          }
44          transform.position = player.transform.position + offset;
45          transform.LookAt(player.transform.position);
46          transform.Rotate(-10f, transform.rotation.y, transform.rotation.z);
47      }
        1 reference
48      private void TurnCamera()
49      {
50          offset = Quaternion.AngleAxis(Input.GetAxis("Mouse X") * turnSpeed,
51          Vector3.up) * offset;
52      }
```

Kuvio 20: Kameran kääntäminen Android-laitteella

5.7 HUD

Prototyypin HUD toteutettiin Unity UI -työkalulla, joka on peliobjekteihin perustuva käyttöliittymäjärjestelmä (Unity Technologies 2021j). Ensimmäinen askel HUDi:n kehityksessä oli lisätä kohtaukseen Canvas-peliobjekti, jonka lapsiobjekteiksi varsinaiset UI-elementit tulivat. Seuraavaksi luotiin UI-elementit aloittaen pelikelloa kuvaavasta tekstiobjektista. Pelikellon lisäksi HUD tarvitsi elementit kuvaamaan pelaajan jäljellä olevia elämiä, jäljellä olevia pistepalloja, pelaajan pistesaldoa ja taukovalikkoa.

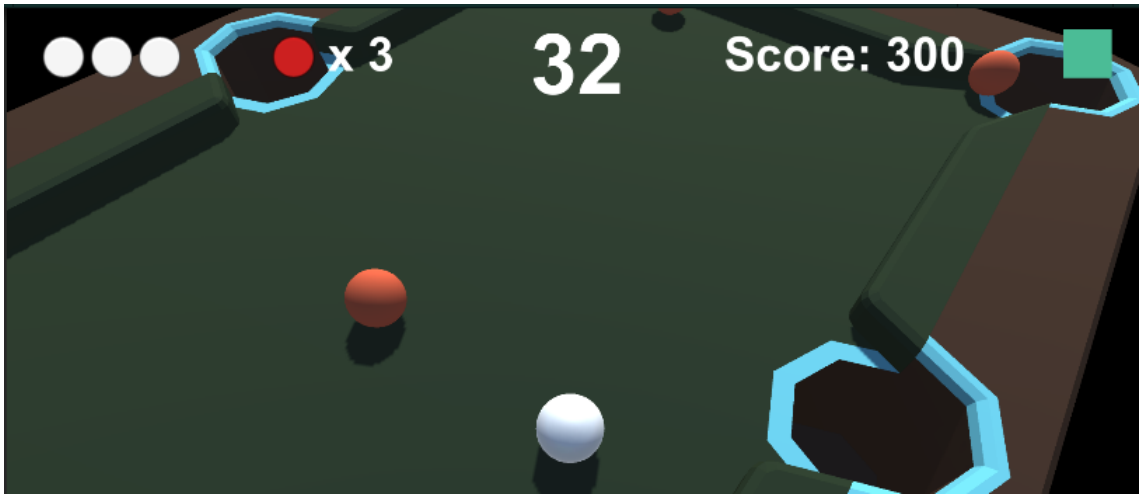
Pelaajan elämät ja pistepallo toteutettiin kuvaobjekteina, joiden Image-komponentteihin vaihdettiin lähdekuviksi Unityn mukana tulevat ympyränmuotoiset kuvat. Taukovalikossa käytettiin kuvaobjektia, jonka lähdekuvaksi vaihdettiin neliö. Pistepallojen varsinainen lukumäärä ja pelaajan pistesaldo tehtiin tekstiobjekteista.

Kuten kohdassa 5.3 esitettiin, prototyypissä HUD:in päivittäminen tapahtui pääasiassa ScoreArea-objektiin kiinnitetyssä skriptissä. Poikkeuksena tästä, kiinnitettiin pelikello-objektiin yksinkertainen aikaa vähentävä Timer-skripti. Protyypissä pelikelloon asetettiin kierroksen kestoksi kolmekymmentä sekuntia. HUD:in taukovalikkoon ei lisätty toiminnallisuutta tässä vaiheessa. Kuviossa 21 on kuvattu Timer-skriptin Update-metodi.

```
0 references
11  private void Update()
12  {
13      if (timeLeft > 0)
14      {
15          timeLeft -= Time.deltaTime;
16          timerText.text = timeLeft.ToString("0");
17      }
18      else
19          Time.timeScale = 0;
20  }
```

Kuvio 21: Timer-skriptin Update-metodi

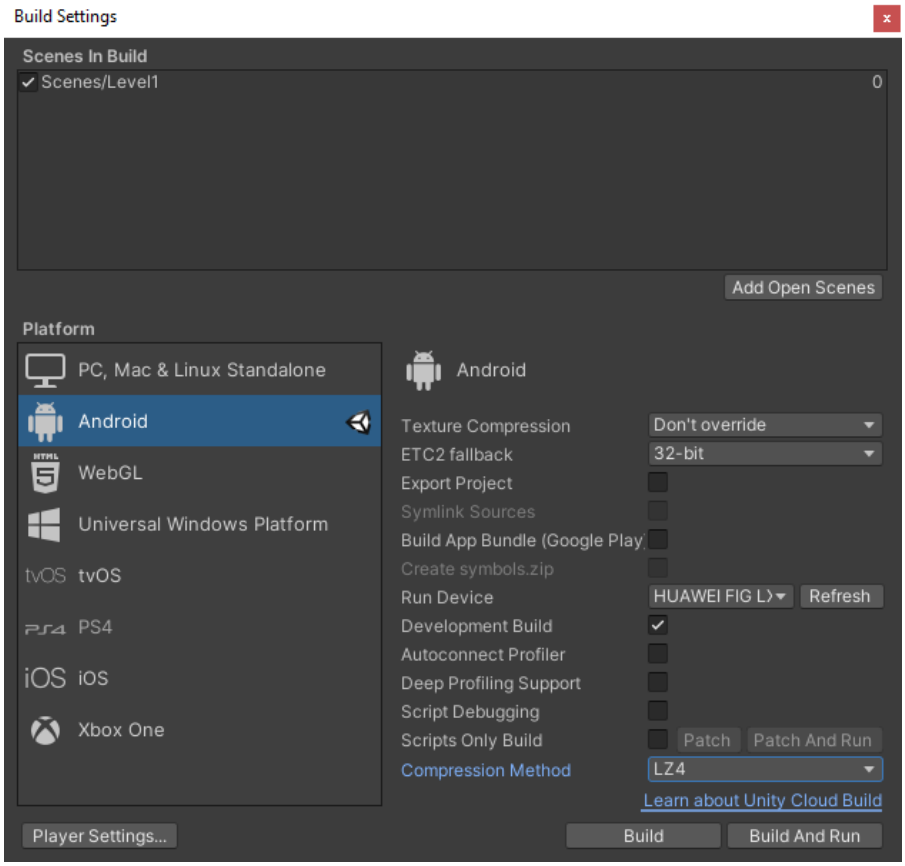
Kun HUD:in elementit oli luotu, täytyi ne sommitella peliruudulle. Elementit päätettiin sijoittaa näytön yläreunaan, jotta pelaajan käsi ei peittäisi niitä kameraa käännettäessä tai pelipalloa laukaistessa. Pelikellosta tehtiin muita elementtejä suurempi ja se sijoitettiin yläreunassa keskelle. Ratkaisuun päädyttiin siksi, että jäljellä olevaa peliaikaa pidettiin pelaajan kannalta tärkeimpänä informaationa. Kellon vasemmalle puolelle sijoitettiin pelaajan elämät ja jäljellä olevat pistepallot. Pistesaldo ja taukovalikko sijoitettiin pelikellon oikealle puolelle. Kuvio 22 havainnollistaa prototyypin HUD:ia.



Kuvio 22: Prototyypin HUD

5.8 Prototyypin testaus

Kehityksen aikana prototyyppiä testattiin aina, kun jokin uusi ominaisuus saatiin valmiiksi. Alkuvaiheessa testaus tapahtui editorin Play-tilassa ja toiminnallisuuden karttuessa aloitettiin suurempien muutosten testaaminen myös Android-puhelimilla. Jotta prototyyppiä voitiin testata Android-laitteella, täytyi pelin Build Settings -asetuksista vaihtaa alustaksi Android ja rakentaa prototyypistä puhelimeen asennettava Android-pakettitiedosto. Kuvio 23 näyttää Build Settings -asetukset Android-testilaitteelle.



Kuvio 23: Build Settings -ikkuna

Testauksen aikana selvisi muun muassa, että prototyypin fysiikka-asetuksia täytyi säätää useaan otteeseen, sillä pallot kimpoilivat välillä kummallisesti pelikentän tasaisella pinnalla. Syyksi osoittautui, että pallot osuivat kentän 3D-mallin saumoihin ja lähtivät törmäyksistä vääriin suuntiin. Ongelmaa ratkottiin muokkaamalla pelikentän ja pallojen Colliderien Physics Material -assettien ja Rigidbodyjen asetuksia. Lopulta päästiin tyydyttävään tulokseen, jossa pallojen liike muistutti tarpeeksi hyvin biljardipallojen törmäilyä, eivätkä ne enää kimpoilleet kentän saumoista.

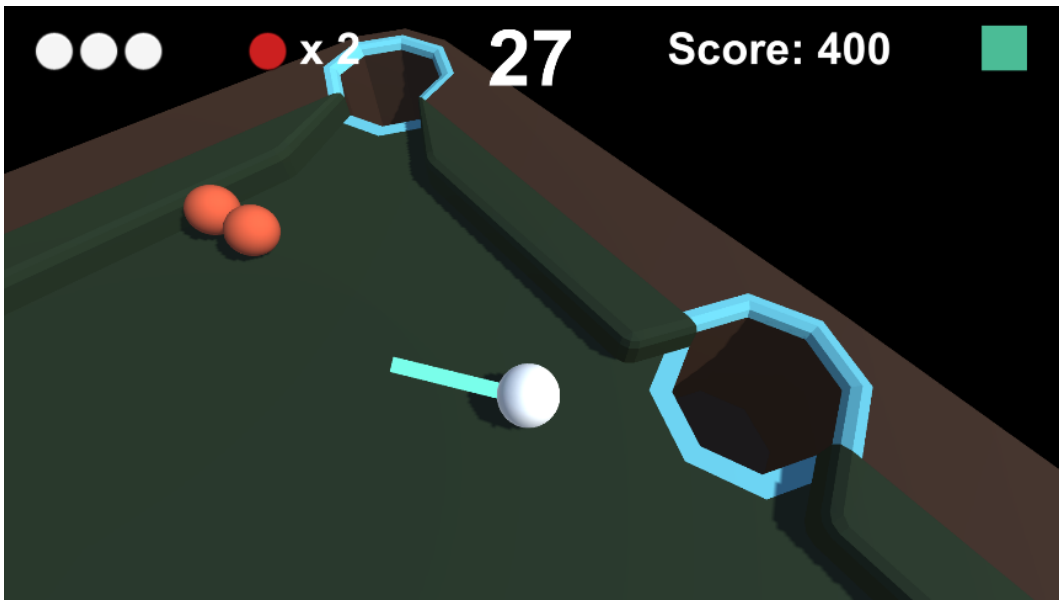
Prototyyppi toimi testilaitteilla suurimmaksi osaksi halutulla tavalla. Yhdellä sormella tapahtuva tähtäys ja laukaisu sekä kameran kääntäminen kahdella sormella sopivat hyvin mobiililaitteen kontrolleiksi. Koska testilaitteet olivat vanhempia Android-puhelimia, ei prototyypin ruudunpäivitys pysynyt aina tasaisena. Suurin syy ruudunpäivityksen nytkähtelyyn oli, ettei prototyypin valaistusta ollut optimoitu mobiililaitteille. Optimointi jätettiin tuotantovaiheeseen, sillä prototyypin kannalta oli tärkeintä, että mekaniikat toimivat ja pelaaminen oli tarpeeksi viihdyttävää.

Testaus tapahtui pääosin itsenäisesti, mutta prototypoinnin loppuvaiheessa päätettiin tiedustella pelaamista harrastavilta lähipiiriläisiltä, kiinnostaisiko heitä prototyypin testaaminen.

Lähipiiriläisistä viisi vastasi myöntävästi, ja he osallistuivat testaukseen pelaamalla prototyypin verkkoselaimessa pyörivää versiota. Testaustilanteet järjestettiin etänä siten, että testaajat pelasivat prototyyppiä omilta tietokoneiltaan ja kommunikointi tapahtui reaaliajassa Discord-viestintäpalvelun välityksellä. Lähipiiriläiset testasivat prototyyppiä vapaamuotoisesti kokeillen ja he raportoivat havaintojaan suullisesti pelaamisen yhteydessä.

Prototyypin testaajilta saatiin palautetta, että hiirellä pelattaessa kolmenkymmenen sekunnin aikaraja vaikutti liian armolliselta. Aikaraja pidettiin kuitenkin kolmessakymmenessä sekunnissa, sillä älypuhelimella testattaessa kierrokset olivat hyvin tiukkoja. Tämän takia päätettiin myös nostaa pussitetuista palloista ansaittua lisäaikaa viidestä sekunnista kymmeneen sekuntiin. Lisäksi huomattiin, että pussiin uppoava pelipallo oli mahdollista pelastaa kimmottamalla se takaisin kentälle ennen pallon törmäämistä ScoreArean Collider-komponenttiin. Ominaisuutta ei ollut suunniteltu tietoisesti, mutta se vaikutti lisäävän peliin mielenkiintoisen taitoelementin.

Kun prototyypin pelattavuus ja eri osa-alueiden toiminnallisuudet oli todettu tyydyttäviksi, siirryttiin esituotannosta tuotantovaiheeseen. Kuviossa 24 on Unity-editorissa otettu kuvakaappaus siitä, miltä valmis prototyyppi näyttää pelitilassa.



Kuvio 24: Pelikuvaa valmiista prototyypistä

6 Yhteenveto

Opinnäytetyön tavoitteena oli kehittää mobiilipeli Android-älypuhelimille ja kartuttaa siten omaa osaamista pelikehityksestä Unityllä. Raportin teoriaosuudessa luotiin katsaus

pelikehityksessä yleisesti käytettyihin prototyyppeihin. Opinnäytetyön raportti rajattiin käsittelemään kehitysprosessin suunnittelu- ja prototypointivaiheita. Suunnittelun aikana kehitettiin peli-idea, päätettiin julkaisualusta ja ansaintalogiikka sekä kirkastettiin pelin päämekaniikka ja ydinsilmukka. Lisäksi hahmoteltiin pelaajan kulkua pelin sisällä ja pohdittiin mitä assetteja peli tarvitsisi.

Vaikka suunnitteluprosessin aikana ei pelin kaikkia osa-alueita pohdittu kovin syvällisesti, antoi se silti hyvän suunnan prototyypin kehittämiselle. Myös prototypoinnin teoriaan perehtyminen auttoi keskittymään oikeisiin asioihin. Prototyyppivaiheessa päätettiin ohittaa paperiprototyyppien tekeminen ja siirtyä suoraan kehittämään pelattavaa prototyyppiä. Digitaalisen prototyypin kehitys sujui jouhevasti lukuun ottamatta laukaisumekaniikkaa, joka vaati eniten pohdintaa sekä erinäisiä fysiikan simulointiin liittyviä ongelmia.

Pelin ensimmäisestä pelattavasta prototyypistä tuli lähes valmis poikkileikkaus pelistä: siitä puuttui ainoastaan eri valikoiden toiminnallisuudet, musiikki ja ääniefektit. Tuotantovaiheeseen siirryttäessä prototyyppiä käytettiin pelin ensimmäisenä iteraationa, jota alettiin jatkokehittää lisäämällä siihen uusia kenttiä ja toiminnallisuutta.

Tuotantovaiheessa peliin tehtiin muutamia suunnitellusta poikkeavia muutoksia. Suurin muutoksista oli se, että jokaisesta pelikentästä kehitettiin kaksi versiota. Toinen versioista oli normaali kenttä, jossa pelattiin aikaa vastaan ja kerättiin pisteitä ja toinen oli ”meditaatioversio”. Kenttien meditaatioversioista poistettiin pelikello, elämät, pussit ja pisteiden kerääminen. Meditaatioversion ideana oli, että pelaaja voisi rentoutua laukomalla pelipalloa ja katselemalla pallojen kimpoilua kentällä.

Valmis peli julkaistiin Google Play Kaupassa nimikkeellä ”Blaardz”. Raportin kirjoittamishetkellä peliä oli pelannut kymmenisen pelaajaa, joista suurin osa oli pelitestaajia ja muuta lähipiiriä. Pelin hukkuminen Play Kaupan tarjontaan oli odotettavissa, sillä suunnitteluvaiheessa ei mietitty pelin markkinointia, kilpailijoiden kartoittamista tai kohderyhmän tarkempaa analysointia. Tulevissa projekteissa täytyy pelisuunnittelun näihin aspekteihin kiinnittää enemmän huomiota.

7 Oman oppimisen arviointi

Ennen opinnäytetyön aloittamista olin tehnyt Unityllä vain pienempiä harjoitusprojekteja ja käynyt läpi lyhyitä pelikehityksen eri osa-alueisiin liittyviä tutoriaaleja. Olin opetellut Unityn skriptien kirjoittamisessa käytetyn C#-ohjelmointikielen perusteet ammattikorkeakoulussa ja syventänyt osaamistani siitä itsenäisesti.

Opinnäytetyössäni kehitin ensimmäistä kertaa pelin konseptista julkaisukelpoiseksi tuotteeksi. Aikaisemmissa harjoitusprojekteissani olin hyödyntänyt valmiita assetteja, kuten musiikkia, äänitehosteita ja peligrafiikkaa. Päätin tätä projektia aloittaessani, että tekisin, mahdollisuuksien rajoissa, itse kaikki pelissä tarvittavat assetit.

Peli-ideaksi valikoitui kolmiulotteinen älypuhelimilla pelattava ajanvietepeli, joka tarkoitti, että minun täytyi luoda pelin kentistä ja niitä elävöittävästä rekvisiitasta 3D-mallit. Minulla ei ollut aikaisempaa kokemusta 3D-mallinnuksesta, joten opettelin mallinnuksen perusteet internetistä löytämieni videotutoriaalien avulla. 3D-mallien tuottaminen eteni aluksi hitaasti yrityksen ja erehdyksen kautta, mutta prosessi nopeutui kokemuksen karttuessa.

Grafiikan lisäksi peliin täytyi tehdä myös musiikki ja muutamia äänitehosteita. En ollut aikaisemmin tehnyt musiikkia, mutta ääniefektien tuottamisesta minulla oli vähän aikaisempaa kokemusta. Opettelin musiikinteorian alkeet samaan tapaan kuin olin opetellut 3D-mallinnuksen internetvideoiden avulla. Sävelsin pelimusiikin iPadin mukana tulevalla GarageBand-musiikkiohjelmalla ja äänitehosteiden luomisessa käytin Audacity-editoria, joka on ilmainen avoimeen lähdekoodiin perustuva äänieditori.

Prototyypin testaamisen aikana huomasin, ettei pelin ruudunpäivitys pysynyt tasaisena testilaitteilla. Suurimman osan ajasta peli pyöri hyvin, mutta välillä ruudunpäivitys nytkähteli. Tuotantovaiheessa tätä ongelmaa ratkottiin optimoimalla pelin skriptejä, grafiikkaa ja valaistusta siten, että peli pyörisi tasaisesti vanhemmillakin Android-puhelimilla.

Jouduin opettelemaan pelioptimoinnin perusteet tuotantovaiheen aikana, sillä en ollut harrastanut optimointia aikaisemmissa Unity-projekteissani tai ohjelmointiharjoituksissani. Optimoinnin jälkeen peli toimi testilaitteilla hyvin, mutta sen asetuksiin lisättiin varmuuden vuoksi mahdollisuus rajoittaa ruudunpäivitys kolmeenkymmeneen ruutuun sekunnissa. Ruudunpäivityksen pakottaminen kolmeenkymmeneen ruutuun sekunnissa varmistaa, että peli saadaan toimimaan nykimättä myös testilaitteita vanhemmissa älypuhelimissa.

Näinkin yksinkertaisen pelin kehittäminen suunnittelusta julkaisuvaiheeseen oli yllättävän työläs prosessi yhdelle ihmiselle, koska kaikki pelin osa-alueet grafiikasta ja musiikista ohjelmointiin täytyi toteuttaa itse. Projektin jälkeen ymmärrän paremmin, miksi pelikehitys tapahtuu yleensä tiimeissä, joissa jäsenet keskittyvät pelin eri osa-alueiden työstämiseen. Vaikka pelikehitys oli välillä työlästä ja pelin julkaisu sujui heikosti, oli projektin tekeminen silti opettavainen ja palkitseva kokemus.

Lähteet

Painetut

Adams, E., Dormans, J. 2012. Game Mechanics: Advanced Game Design. Berkeley: New Riders.

Despain, W. 2013. 100 Principles of Game Design. Berkeley: New Riders.

Gibson, J. 2015. Introduction to Game Design, Prototyping and Development. 3. painos. Crawfordsville: Addison-Wesley: Pearson.

Sähköiset

Blender Foundation. 2021. Blender. Viitattu 20.5.2021. https://www.blender.org

GameAnalytics. 2018. Microtransactions In Games: The Good, The Bad, And The Ugly. Viitattu 20.5.2021. https://gameanalytics.com/blog/microtransactions-games-good-bad-ugly/

Neogames. 2019. The Game Industry of Finland. Viitattu 22.5.2021. https://neogames.fi/wp-content/uploads/2020/07/FGIR-2018-Report.pdf

Pluralsight. 2014. Designing a HUD That Works for Your Game. Viitattu 20.5.2021. https://www.pluralsight.com/blog/film-games/designing-a-hud-that-works-for-your-game

Stefyn, Nadia. 2019. How video games are made: the game development process. CG Spectrum. Viitattu 1.5.2021. https://www.cgspectrum.com/blog/game-development-process

Unity Technologies. 2021a. Quick guide to the Unity Asset Store. Viitattu 20.5.2021. https://unity3d.com/quick-guide-to-unity-asset-store

Unity Technologies. 2021b. Creating and Using Scripts. Viitattu 20.5.2021.https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html

Unity Technologies. 2021c. Plans and pricing. Viitattu 5.5.2021. https://store.unity.com/#plans-individual

Unity Technologies. 2021d. Unity Collaborate. Viitattu 5.5.2021. https://docs.unity3d.com/Manual/UnityCollaborate.html

Unity Technologies 2021e. Mesh Collider. Viitattu 29.4.2021. https://docs.unity3d.com/Manual/class-MeshCollider.html

Unity Technologies. 2021f. Physic Material. Viitattu 29.4.2021. https://docs.unity3d.com/Manual/class-PhysicMaterial.html

Unity Technologies. 2021g. Box Collider. Viitattu 29.4.2021. https://docs.unity3d.com/Manual/class-BoxCollider.html

Unity Technologies. 2021h. Rigidbody. Viitattu 29.4.2021. https://docs.unity3d.com/Manual/class-Rigidbody.html

Unity Technologies 2021i. Using Cinemachine. Viitattu 6.5.2021. https://docs.unity3d.com/Packages/com.unity.cinemachine@2.5/manual/CinemachineUsing.html

Unity Technologies 2021j. Unity UI: Unity User Interface. Viitattu 5.5.2021. https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/index.html

Kuviot