VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Teemu Paunonen

Pelinkehitys Unity 3D:llä

Datan serialisointi ja tallentaminen

Liiketalous
2018

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittely

TIIVISTELMÄ

Tekijä	Teemu Paunonen
Opinnäytetyön nimi	Pelinkehitys Unity 3D:llä
Vuosi	2018
Kieli	suomi
Sivumäärä	40
Ohjaaja	Raija Tuomaala

Tässä opinnäytetyössä selvitetään, millä tekniikalla saadaan luotua joukkueita ja pelaajia, sekä mitenkä saadaan nämä tiedot tallennettua ja serialisoitua. Tämän jälkeen saadaan kyseiset tiedot vielä deserialisoitua ja ladattua uudelleen.

Työssä käyn läpi mitä pelinkehitys vaatii yksittäiseltä kehittäjältä tai mitä se vaatii kehitystiimiltä. Katsastetaan muita pelimoottoreita ja vertaillaan kahta suosituinta pelimoottoria eli Unitya ja Unreal Engineä.

Työssä käy ilmi, että ilman kunnollista suunnittelua, sekä yksin työstäessä peliä voi pelinkehityksestä tulla pitkä prosessi, jossa on paljon tehtävää. Lopujen lopuksi esitellään hyvä tekniikka tietojen luomiseksi, sekä hyvät tavat tallentamisen ja serialisoinnin tekemiseen. Tämän jälkeen saadan tiedot vielä deserialisoitua ja ladattua uudelleen peliin.

Avainsanat Pelikehitys, serialisointi, Unity3D, C#, tallennus, lataus

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietojenkäsittely

ABSTRACT

Author	Teemu Paunonen
Title	Game Development with Unity3D
Year	2018
Language	Finnish
Pages	40
Name of Supervisor	Raija Tuomaala

In this thesis the aim was to find out the best method for creating data that will be saved and serialized and then loaded and deserialized the created data.

The research studied what is needed to be a lone wolf game developer or what is needed to work in a development team. In the thesis, game engines were studied, and two most popular game engines Unity and Unreal Engine were compared.

I found out that if there is no a plan and the developer is working alone, then the journey might be long, and there will be many things to do. At least a good method of creating data was discovered. I also learned how to easily save and serialize data, and after that load and deserialize the data back to the game.

Keywords Game programming, serializing, Unity3D, C#, save, load

SISÄLLYS

KUVALUETTELO

1 JOHDANTO

Suurena jääkiekon ja manageripelien ystävänä saatiin idea kehitellä omanlainen versio manageripelistä, jossa käyttäjä pääsee manageroimaan harrastetasoisen liigan joukkuetta. Pelisarjat, jotka ovat innostaneet pelin tekemiseen ovat Out Of The Parkin luoma pelisarja, Franchise Hockey Manager, jossa käyttäjä pääsee manageroimaan jääkiekkojoukkueita eri maiden sarjoissa, Sports Interactiven luoma, Eatside Hockey Manager-pelisarja ja EA Sportsin luoma NHL-pelisarja, sekä FIFA-pelisarja. Näitä pelejä pelatessa voi tulla mieleen, mitä peleistä puuttuu ja mitä niihen voisi lisätä. Pelatessa näitä pelejä saatiin idea, että entä jos saisikin itse luotua oman pelin, kun pelit tuntuvat välillä niin puutteellisilta joiltakin osin. Tästä saatiinkin idea, että opinnäytetyön voisi tehdä jostakin pelin osasta. Työssä selvitetään millä tekniikalla saa manageripelin alkutiedot luotua, sekä mikä on paras vaihtoehto tallentaa tiedot ja mitenkä saadaan tiedostosta sellainen, mikä ei ole ihmisluettava. Kun kaikki tämä on tehty, pitäisi tiedosto vielä saada avattua ja tiedot siirrettyä peliin.

Hyvä pohja pelin luomiseen saatiin, kun ammattikorkeakoulussa on opittu C#:ia ja työharjoittelussa oltiin Platonic Partnershipillä, jossa saatiin työskennellä Unityn parissa. Apuna työssä on käytetty Stackoverflowin foorumin hakutoimintoa, mutta siellä ei kuitenkaan olla esitetty kysymyksiä työhön liittyen (Stack Exchange Inc, 2018). Suuri apu työhön on ollut myös Unityn tutoriaali videoista ja livesessioista, joita on voinut katsoa jälkikäteenkin (Unity Technologies 2018a & Unity Technologies 2018b).

Unityn käyttöliittymää ei lähdetä avaamaan tässä sen tarkemmin vaan oletetaan, että työn arvioija ja lukijat tietävät Unitysta perusasiat. Unity on muutenkin melko helppokäyttöinen. Peli on tehty canvas- eli paneelielementeillä ja koodin kautta ohjataan, mitkä paneelit ovat näkyvillä ja mitkä piilotettuna. Ideana pelissä on ensimmäiseksi joko itse luoda peliin joukkueet ja pelaajat tai sitten jos pelaaja haluaa mennä suoraan peliin. Pelaaja voi luoda joukkueet ja pelaajat automaattisesti. Kun joukkueet ja pelaajat on luotu, pelin tiedot voidaan tämän jälkeen serialisoida ja tallentaa, jonka jälkeen tiedot voidaan deserialisoida ja ladata uudestaan. Tässä

työssä ei olla panostettu audiovisuaaliseen eikä graafiseen sisältöön, koska tarkoitus on näyttää mitenkä Unitylla ja C#:illa saa kirjoittamalla ja satunnaistekniikalla luotua tekstimuotoista tietoa ja mitenkä niiden serialisoiminen, tallentaminen ja lataaminen toimivat. Työstä ei näytetä suoraa koodia, ettei joku kopioi työtä suoraan itsellensä. Koodinpätkät, joita näytetään ovat esimerkkejä, miten kyseiset tilanteet on työssä tehty. Työssä ei siis ole suoraa koodia pelistä vaan erilaisia havainnollistavia esimerkkejä, siitä miten ongelmat on ratkaistu. Esimerkeissä tulee kuitenkin hyvin ilmi, mitenkä kyseiset ongelmat on ratkaistu. Työtä on tarkoitus jatkaa, sekä kolmen vuoden sisään peli olisi tarkoitus saada julkaistua Steamiin eli videopelien jakelualusta PC-peleille.

2 MITÄ PELINKEHITYS VAATII?

Pelit ovat koko ajan kasvava ala ympäri maailmaa ja oikeastaan kuka tahansa voi ruveta suunnittelemaan ja tekemään pelejä ilman minkään näköistä kokemusta ohjelmoinnista tai pelien kehittämisestä. Mitä vähemmän kokemusta, niin sitä enemmän asioita joutuu opettelemaan ja sitä kauemmin projektissa todennäköisesti menee. Monella aloittelijalla loppuu motivaatio siihen, kun koko ajan joutuu opettelemaan uusia asioita.

Jos haluaa ruveta pelinkehittäjäksi, tulee oikeasti olla kiinnostunut peleistä. Ei vaan pelaamisesta, vaan ymmärtää myös, kuinka pelit on rakennettu ja miettiä mitenkä se on rakennettu eri osista, jotka on koottu yhteen (Gruber 2000, 3). Pelinkehityksessä on erilaisia työtehtäviä, joista vastaa tuottaja. Tuottaja voi vastata kaikesta itse tai koota kehitystiimin, jossa jokaiselle tiimissä on suunnattu omat tehtävät:

- Suunnittelija, joka kokoaa kaikki pelin elementit ja asetukset toimivaksi kokonaisuudeksi.
- Ohjelmoija, vastaa pelin kehityksestä ja ohjelmoinnista.
- Graafikko, joka suunnittelee ja luo pelille visuaalisen ilmeen.
- Ääniteknikko, joka vastaa pelin äänimaailmasta.
- Testaaja, joka vastaa pelin toimivuudesta ja etsii peleistä virhetilanteista, joista hän ilmoittaa eteenpäin, jotta nämä virhetilanteet saadaan korjattua.
- Julkaisija, joka vastaa pelin markkinoinnista. Yleensä julkaisijana toimii erillinen yritys, joka keskittyy pelien julkaisuun.

Haluttaessa tuottaja voi vastata kaikesta peliin liittyvästä itse. Tällöin hän ottaa vastuullensa kaikki nämä roolit. Rahallista hyötyä on se, että ei tarvitse maksaa muille palkkoja, mutta pelin tekemisessä kuitenkin kuluu enemmän aikaa. Jos olet yksin tuottaja, niin yksin tuottajalla on oikeus hylätä peli tai jatkaa sitä, milloin itse haluaa, eikä kukaan odota yksin tuottajalta mitään (Gruber 2000, 3). No mitä tietoja ja taitoja pelinkehitys sitten vaatii yleisesti? Olisi hyvä, että osaisi jonkun ohjelmointikielen, jolla pääsee alkuun. Nykyään on myös paljon erilaisia peli-

moottoreita, joissa käytetään eri ohjelmointikieliä ja näin ollen kannattaa valita pelimoottori, joka käyttää osaamaasi ohjelmointikieltä. Toki pelin voi luoda ilman pelimoottoria, mutta silloin pelin tekeminen vaatii enemmän, koska ei ole valmiiksi luotua pohjaa ja ohjelmistoa, mistä saa osan pelin elementeistä valmiina. Pelinkehitys vaatii myös hyvää ongelmanratkaisukykyä, koska peliä kehittäessä tulee paljon virhetilanteita, jotka pitää osata ratkaista (Gruber 2000, 3). Vaikka olisi kuinka hyvä idea päässä, niin aloittelevan pelinkehittäjän on sitä todennäköisesti vaikea toteuttaa. Kannattaa siis aloittaa tekemällä jokin yksinkertainen peli aluksi, vaikkapa ristinolla tai viidensuora (Gruber 2000, 4). Samat säännöt pätevät vieläkin, eli jos meinaa ruveta pelinkehittäjäksi, niin kannattaa ensimmäisenä tehdä yksinkertainen peli. Kaiken lisäksi, jos tehdään valmiilla pelimoottorilla, niin näiden julkaisijan sivuilta löytyy tutoriaaleja yksinkertaisten pelien tekemiseen. Näissä oppii käyttämään erilaisia tekniikoita. Unitylläkin on monta erilaista tutoriaalia, mistä oppii erilaisia pelimekaniikoita (Unity Technologies, 2018a). Kun rupeaa tekemään peliä kannattaa myös kysyä itseltä seuraavat kysymykset:

- Miksi teet peliä ja mitä haluat peliltä?
- Haluatko tienata rahaa pelillä?
- Yritätkö vain kehittää itseäsi tietyillä osa-alueilla?
- Yritätkö saada osaamisesi esille, esittelemällä luomustasi muille?
- Haluatko mahdollisesti työskennellä pelialalla?
- Haluatko vain harrastaa pelinkehitystä ilman paineita onnistumisesta, koska se on mielestäsi mukavaa ajanvietettä?

Jos haluaa tienata pelillä ja työskennellä pelialalla, kannattaa miettiä hakemista johonkin valmiiseen pelifirmaan, joka on jo valmiiksi päässyt sisään pelimarkkinoille ja täten pystyy maksamaan palkkaa, koska peliala on riskialtis siltä osin, että sillä ei välttämättä tienaa muuta kuin pieniä hiluja. Jos taas luot yksinään pelisi, joudut mahdollisesti maksamaan joillekin palkkaa töistä, joita ei itse ehdi tehdä ja jos on vähän työntekijöitä ja on vaativa peli tuotannossa, niin pelin tekemiseen voi vierähtää yhtäkkiä kymmenenkin vuotta. Yksin työskennellä on myös jatkuvasti kehitettävä itseään eri asioissa ja julkaisuvaiheen lähestyessä pitää itse löytää oikea markkinarako, että peli saa myyntiä (Gruber 2000, 8). Isoilla pelifirmoilla

onkin vähintään 200 työntekijää, mikä alkaa olemaan minimäärä onnistuneen pelin luomisessa ja markkinoille tuotaessa (Gruber 2000, 5). Nykyään kaikista isoimmilla pelifirmoilla on tuhansia työntekijöitä, mutta näissä tehdään montaa peliä samanaikaisesti. Myydyimmät ja suosituimmat pelit tulevatkin pääosin isoilta peliyrityksiltä, muutamia poikkeuksia lukuun ottamatta, mutta isotkin yritykset on ollut joskus pienempiä. Kun pelin kehittäjä tai kehittäjät osuvat kultasuoneen eli saadaan julkaistua peli, jolla tienaa paljon rahaa, niin kyseinen yritys voi olla myöhemmin isompi yritys, kun on rahaa palkata enemmän tekijöitä, mikä tarkoittaa, että pelejä voidaan saada nopeammin valmiiksi työntekijä määrän kasvaessa. Yrityksen täytyy kuitenkin tehdä voittoa, jotta tuotantoa kannattaa pitää, eli pitää saada selvitettyä minkälaisia pelejä ihmiset haluavat ja mitä niiltä odotetaan, jotta saadaan luotua uusi menestyvä peli ja saadaan pidettyä yritys voitolla.

3 MANAGERIPELIN RAKENTAMINEN UNITYSSÄ

Unitystä on monta erilaista versiota ja projekti ei välttämättä toimi uudemmalla tai vanhemmalla versiolla kuin sillä, millä projekti on tehty. Tämä työ on tehty Unity 2017.3.1f1-versiolla. Aloittaessa projektia valitaan onko peli 2D vai 3D, sekä keksitään pelille nimi. Peli on tehty 2D:nä, mutta sen voi tehdä myös 3D:nä. Aluksi kannattaa luoda kansiot scripteille, materiaaleille, tallennuksille ja prefabeille eli esikoostetuille peliobjekteille, joista voi luoda useita instansseja. Tarvittaessa voi tehdä myöhemmin lisää kansioita, mutta tähän pelin vaiheeseen ei tarvita muita kansioita. Loogisesti scriptit sijoitetaan scriptit-kansioon, tallennukset tallennukset-kansioon, materiaalit materiaalit-kansioon ja prefabit prefabit-kansioon. Tässä vaiheessa kansioihin ei vielä luoda mitään.

3.1 Miksi Unity?

Tämän tyylisen urheilumanageripelin olisi saanut tehtyä ilman Unityä, esimerkiksi C# ja Windows Presentation Foundationin avulla, missä luodaan ikkunoita ja sijoitetaan tietoja ikkunoissa oleviin kenttiin ja valikoihin. Siirtyminen ikkunasta toiseen tapahtuu nappia painamalla. Samaa tekniikkaa käytetään Unityssä urheilumanageripeliä tehdessä.

Työhön valittiin kuitenkin Unity, koska Unityn käyttöliittymä kiinnostaa ja on halua oppia enemmän Unityn toimintoja. Unityssä on myös valmiit pohjat pelin rakentamiseksi eri alustoille, kuten Windowsille, Linuxille, Androidille tai iOSille, eikä sen suhteen tarvitse miettiä toisen käyttöjärjestelmänversion rakentamista itse. Graafisen puolen toteutus on mielestäni Unityn kautta helpompi tehdä kyseiseen peliin kuin ilman Unitya.

Erilaisia pelimoottoreita löytyy yli 100, erilaisilla ominaisuuksilla ja vahvuuksilla. Näistä tunnetuimpia Unityn lisäksi ovat AppGameKit, CryEngine Frostbite Engine, GameMaker, Godot ja Unreal Engine, mutta Unitystä oli saatu jo kokemusta työharjoittelun kautta ja Unity vaikutti mielenkiintoiselta. Osa pelimoottoreista on ilmaisia ja osa kaupallisia. Useissa ilmaisissa ohjelmissa on kuitenkin erikseen esimerkiksi 3D-malleja, grafiikoita ja tietoja, jotka saa käyttöön maksua vastaan.

Unityssakin on erikseen kauppa, mistä voi ostaa muiden luomaa grafiikkaa, äänimaailmaa tai valmista koodia. Unitysta on myös maksullisia versioita, jolloin saa Unitylta reaaliaikaisesti apua pelin tekemiseen ja halvemmalla Unityn kaupan tavaroita. Unitylle löytyy myös tutoriaalikirjoja videoiden lisäksi. Kirjoja löytyy niin aloittelijalle kuin kokeneemmallekin. Vasta-alkajan, joka ei ole aikaisemmin koskaan koodannut, on mahdollisuus päästä saman tien perille C#:in perusteista aloittelijalle suunnatun kirjan avulla ja perusteiden harjoittelun jälkeen voi vaan todeta, että tämähän onkin helppoa (Norton, 2013, 18).

3.2 Unity vastaan Unreal Engine

Unity ja Unreal Engine ovat kaksi suosituinta pelimoottoria, joita käytetään pelien kehityksessä. Unityn suosio johtuu siitä, että se sopii hyvin pelin kehittäjille, jotka toimivat yksin tai pienessä ryhmässä. Unityn avulla voi myös luoda melkeinpä minkä tyylisen pelin tahansa. Unityn suurin ongelma peliä tehtäessä voi tulla siinä, että halutessaan käyttöön pelin suorituskyvyn raportoinnit, muokattavat aloitusruudut, joukkueiden lisenssit saa käyttöön vasta, kun maksaa kuukausittain Unityn Professional Editionista (GameDesigning, 2018). Unreal Engine on noussut suosituksi pelimoottoriksi, kun se mahdollistaa pelien tekemisen erilaisiksi ja ainutlaatuisiksi pelikokemuksiksi, mutta se ei ole kovin helppokäyttöinen aloittelevalle pelinkehittäjälle (GameDesigning, 2018).

3.3 Pelin käyttöliittymä

Unityn UI eli User Interface eli suomeksi käyttöliittymä tarkoittaa Unityn päänäkymää. Tällä ei kuitenkaan tekstissä tarkoiteta itse Unityn yleistä käyttöliittymää vaan pelin sisälle luotavaa käyttöliittymää ja sen asetuksia. Tekstissä tulen käyttämään UI-lyhennettä.

3.3.1 Paneelit

Peli koostuu paneeleista, joille on laitettu kaikki peliin tarvittavat asiat, kuten painikkeet, pudotusvalikot, liukusäätimet, syöttö- ja tekstikentät. Ensimmäisenä luodaan pääpaneeli, jonka alle kaikki pelin muut paneelit tulevat (Kuva 1). Unityssa

paneelit löytyvät UI- valikosta canvas-nimellä. Paneelin voi halutessaan nimetä haluamallaan nimellä.

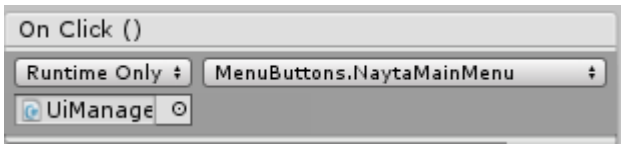
Seuraavaksi luodaan uusi paneeli ja nimetään se päävalikoksi. Tämä paneeli on pelin aloituspaneeli. Kun peli käynnistetään, on tämä paneeli ensimmäisenä näkyvissä. Samalla tavalla luodaan kaikki muut paneelit, joita tarvitaan pääpaneelin alle ja nimetään ne, jotta selaaminen olisi helpompaa. Paneeleista kannattaa laittaa kaikki muut näkymättömäksi paitsi se, mitä parhaillaan muokataan.

Kuva 1. Tarvittavat paneelit pääpaneelin alla.

3.3.2 Painikkeet

Painikkeiden avulla pelissä siirrytään paneelista toiseen ja luodaan satunnaisia nimitietoja tekstikenttiin. Ensimmäisenä luodaan mainPanelin ensimmäiseen paneeliin eli MainMenun alle kolme painiketta ja nimetään ne uusi peli-, lataus- ja exit-painikkeeksi. Laitetaan jokaisen painikkeen tekstikenttään teksti eli uusi peli -painikkeen painikkeeseen tulee teksti New Game, lataus-painikkeeseen tulee Load ja exit-painikkeeseen tulee teksti Exit. Tässä vaiheessa ei vielä tehdä muuta painikkeille.

Painikkeille ei Unityn sisällä tehdä muuta kuin määritellä, mitä tapahtuu, kun painiketta painaa. Esimerkiksi paneeliin vaihtumiseen tarkoitetuille painikkeille valitaan UiManager-prefab, mikä pitää sisällään MenuButtons-scriptin. Tämän jälkeen vetovalikko-näkymästä valitaan metodi, joka suoritetaan MenuButtons-scriptistä, kun nappia painetaan ja näin ollen oikea paneeli avautuu ja muut sulkeutuvat (Kuva 2).



Kuva 2. Painikkeiden asetukset paneelin vaihtumisen.

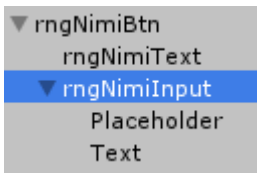
Melkein kaikki painikkeen painallukset on määritelty scriptien kautta, eikä Unityn kautta tarvitse muuta kuin siirtää painikkeet niille scriptin sisällä määriteltyyn kohtiin. Esimerkiksi määritetään scriptin sisällä olevalle exit-buttonille painike Unityn sisällä eli vedetään haluttu painike painikkeen määrittämiskenttään (Kuva 3).

Kuva 3. Määritetään painike scriptissä käytettävälle oliomuuttujalle.

3.3.3 Syöttökentät ja tekstikentät

Tekstikenttä löytyy UI-valikosta nimellä Text. Tekstikenttiä on käytetty vain ohjeistukseen, mitä pelaajan pitää tehdä, jotta pääsee pelissä eteenpäin.

Syöttökentät löytyvät UI-valikosta nimellä Input Field. Syöttökenttiin käyttäjä voi joko itse kirjoittaa halutun tekstin tai painaa syöttökentälle määrättyä painiketta, joka antaa scriptistä satunnaisen arvon syöttökenttään. Tämän jälkeen kenttää voi vielä muokata tai arpoa uuden arvon. Pelissä syöttökentät on luotu näppäinten alle, koska näppäimet voivat määritellä kentälle satunnaisen arvon (Kuva 4).



Kuva 4. Syöttökentän näkymä painikkeen aliobjektina.

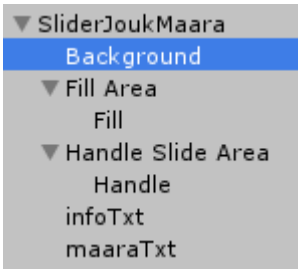
Syöttö- ja tekstikenttiä ei Unityn kautta sen kummemmin muokata muuten kuin nimetään ne loogisesti, jotta tietää tarvittaessa, mistä ne löytyvät. Syöttö- ja teks-

tikentät määritellään samalla tavalla kuin painikkeetkin eli scriptin kautta luodaan oliomuuttuja ja vedetään Unityssa haluttu objekti ja haluttuun oliomuuttujaan (Kuva 3).

3.3.4 Liukusäätimet

Liukusäädinten avulla pelissä säädetään erilaisia asetuksia kuten joukkueiden määriä ja montako peliä kauden aikana pelataan toisia vastaan. Kun joukkuemäärä on päätetty ja siirrytään joukkueiden luontiin. Silloin ilmestyy joukkueenluomis-paneeliin liukusäädin, jonka arvoa ei voi muuttaa vaan sen arvo kasvaa aina kun käyttäjä luo joukkueen, kunnes valittu joukkue määrä on saavutettu.

Luodaan liukusäädin asetukset-paneeliin. Liukusäätimen löytää UI-valikosta nimellä slider. Liukusäätimen oletusasetuksia ei ruveta sen kummemmin muokkailemaan, mutta luodaan liukusäätimelle kuitenkin kaksi tekstikenttää, joista toinen kertoo mitä liukusäätimestä tapahtuu ja toinen tekstikenttä kertoo liukusäätimen arvon (Kuva 5).



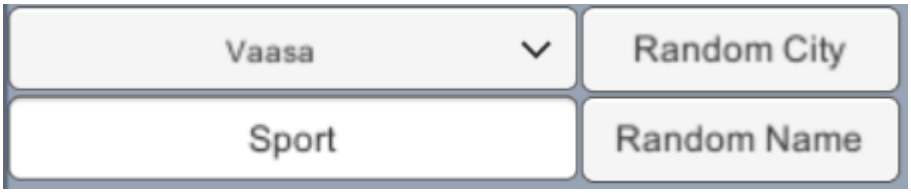
Kuva 5. Liukusäätimen näkymä kahdella lisätyllä tekstikentällä.

Tässä vaiheessa peliä liukusäätimiä ei ole vielä paljon, mutta kun ruvetaan tekemään tekoälyä ja tarvitaan enemmän säädeltäviä asetuksia, niin liukusäätimiä tulee olemaan enemmän.

3.3.5 Alasvetovalikot

Alavetovalikko löytyy UI-valikosta nimellä Dropdown. Alasvetovalikoita käytetään pelissä, jos halutaan, että käyttäjä ei voi itse kirjoittaa kenttään, vaan joutuu

valitsemaan arvon tietyistä valikoista. Painiketta painamalla voi myös alasvetovalikon arvon valita satunnaisesti (Kuva 6).



Kuva 6. Näkymä alasvetovalikosta, syöttökentästä ja painikkeista.

Alasvetovalikon arvoja ovat esimerkiksi maat, koska pelissä voi valita vain tiettyjä maita. Toinen on kaupunki, jonka arvot määräytyvät valitun maan perusteella. Pelaajan luomisessa alasvetovalikkoja käytetään myös syntymäajan valinnassa, pelipaikan valinnassa, kätisyyden valinnassa, sekä joukkueen valinnassa.

4 C#:ISSA TOTEUTETTAVAT OSAT

Unityssa toteutetussa pelissä scriptien kautta määritellään ja tehdään melkein kaikki mahdollinen asia, mitä pelissä tapahtuu. Näitä tapahtumia ovat paneelien selaaminen, tietojen syöttäminen, alasvetovalikoiden luonti, tiedon serialisoiminen, tietojen tallentaminen ja tietojen lataaminen.

4.1 Paneelien selaaminen

Ensimmäisenä tehdään MenuButtons-niminen scripti Scripts-kansioon, jonka jälkeen Prefabs-kansioon luodaan prefab ja nimetään se UiManageriksi. Tämän jälkeen vedetään MenuButtons-scripti UiManagerin asetuksiin. Tämän jälkeen siirrytään scriptin muokkaamiseen. UiManageri on objekti, joka hallitsee mitkä paneelit ovat näkyvissä ja mitkä ovat piilotettuna.

Aluksi lisätään scriptin alkuun `using UnityEngine.UI`, mikä mahdollistaa UI-elementtien muokkaamisen scriptin kautta. Seuraavaksi tehdään oliomuuttujat, jotka luokitellaan julkisiksi eli `public` koodin sisällä. Tällöin saadaan Unityn kautta vedettyä haluttu objekti oliomuuttujaan ja näin ollen kyseisen objektin muokkaaminen onnistuu pelin aikana. Paneelit ovat pelin objekteja, joten kirjoitetaan kaikki paneelit `public GameObject paneelinNimi`. Tämän jälkeen voidaan käydä siirtämässä Unityn kautta paneelit, niille määrätyille oliomuuttujille (Kuva 7) ja siirtää UiManager prefab Unityn hierarkiaan. Nyt voidaan määritellä painikkeet, joita painamalla paneelit vaihtuvat (Kuva 2).

Kuva 7. Yhdistetään oliomuuttujat oikeisiin objekteihin.

Kun kaikki oliomuuttujapaneelit on luotu, niin laitetaan Start-metodiin kaikki paneelien alkunäkymät eli onko paneeli näkyvissä vai piilossa. Nämä saadaan, kun kirjoitetaan `Start`-metodin sisälle `olioMuuttujaPaneeli.SetActive(true` tai `false)`. `true` -arvo tarkoittaa, että paneeli on näkyvissä ja `false` -arvo tarkoittaa, että paneeli on näkymätön (Kuva 8).

```
public GameObject oliomuuttujaPaneeli1;
public GameObject oliomuuttujaPaneeli2;

void Start()
{
    oliomuuttujaPaneeli1.SetActive(true);
    oliomuuttujaPaneeli2.SetActive(false);
}
```

Kuva 8. Oliomuuttujat ja paneelien alkuasetusten määrittäminen.

Seuraavaksi luodaan jokaiselle paneelille julkinen paneelinvaihtometodi, minkä avulla määritetään, mikä paneeli on näkyvissä (Kuva 9). Niin monta metodia tarvitaan, kuin paneeleita on ja jokaiseen metodiin pitää merkitä kaikki muut paneelit näkymättömäksi ja haluttu paneeli näkyväksi.

```
public void paneelinVaihtoMetodi()
{
    oliomuuttujaPaneeli1.SetActive(false);
    oliomuuttujaPaneeli2.SetActive(true);
}
```

Kuva 9. Paneelinvaihto-metodi.

4.2 Serialisoitavat luokat

Serialisoitavat luokat ovat pelin tärkein osa, koska joukkueiden ja pelaajien tallentaminen tapahtuu näiden luokkien avulla, jolloin saadaan tiedostosta ei ihmisluettava eikä tiedoston muokkaaminenkaan onnistu helposti ilman, että tiedosto korruptoituu. Luodaan scripti tiedoille, jotka halutaan serialisoida ja tallentaa. Scriptistä voi saman tien poistaa oletus `MonoBehaviour`-luokan ja kaikki muut paitsi `using System.Collections.Generic`. Tämän jälkeen lisätään `using System`, jonka jälkeen luodaan serialisoitavat luokat ja muuttujat luokille (Kuva 10).

```
using System.Collections.Generic;
using System;

[Serializable] //Määritellään serialisointi mahdollisuus
public class luokka1
{
    public int id;
    public string nimi;
    public List<luokka2> pelaajat = new List<luokka2>(); //Lista joukkueen pelaajista,
}                                                        //jossa muuttujat ovat toisen luokan muuttujien mukaan
[Serializable]
public class luokka2
{
    public int id;
    public string nimi;
}
```

Kuva 10. Serialisoitavien luokkien luominen.

Näistä luokista luodaan myöhemmin listat, mihinkä saadaan kaikki joukkueet, esimerkiksi jos joukkueiden luokan nimi on luokka1 tehdään joukkueidenluonti scriptiin julkinen lista, missä käytetään luokka1:sen muuttujia (Kuva 11).

```
public List<luokka1> joukkueet = new List<luokka1>();
```

Kuva 11. Listan luominen serialisoitavaa luokkaa käyttäen.

4.3 Joukkuemäärän määrittäminen

Ennen joukkueiden luomista pelissä kysytään, kuinka monta joukkuetta pelaaja haluaa peliin. Tähän voi liukusäädintä vetämällä valita 2-16, minkä tahansa numeron, tarkoittaen kuinka monta joukkuetta pelaaja haluaa. Liukusäätimen minimi- ja maksimiarvo valinnat lisätään Unityn kautta, niille valmiiksi määritetyille kentille, jotka ovat loogisesti Min Value ja Max Value. Pelissä on minimiarvo kaksi ja maksimiarvo on kuusitoista, kun valitaan joukkuemäärää (Kuva 12).

Kuva 12. Joukkuemäärän määrittäminen.

Tämän näköisen liukusäätimen teksteineen (Kuva 12) saadaan, kun lisätään liukusäätimelle kaksi tekstikenttää, joista toiseen laitetaan tieto mitä liukusäätimellä tehdään ja toiselle puolelle laitetaan liukusäätimen arvo. Number of teams -tekstin voi kirjoittaa suoraan Unityssa tekstikentän tekstiosioon. Liukusäätimen arvo saadaan, kun kirjoitetaan scriptiin muuttujat tekstikentälle ja liukusäätimelle. Tämän

jälkeen laitetaan `void Update`-metodin sisälle määritykset, jossa liukusäätimen arvo on sama kuin tekstikentän arvo.

4.4 Tietojen luominen

Listat ovat tärkeässä asemassa pelissä, kun niiden avulla luodaan joukkueet ja pelaajat. Ensiksi tehdään uusi scripti ja nimetään se haluamalla nimellä esimerkiksi tietojenLuonti. Tämän jälkeen lisätään scriptin alkuun `using System.Text`, jotta voidaan muokata tekstikenttiä scriptin kautta, `using System.IO`. Nyt voidaan tallentaa ja ladata tietoa, `using UnityEngine.UI`, jotta voidaan muokata pelinsisäistä käyttöliittymää, `using System.Runtime.Serialization.Formatters.Binary`. Tämä mahdollistaa tietojen serialisoinnin ja deserialisoinnin, sekä tallentamisen ja lataamisen. Seuraavaksi luodaan listamuuttujat joukkueet ja pelaajat, jotka käyttävät niille tarkoitettuja serialisoitavia luokkia (Kuva 11).

Luodaan muuttuja nimeltä maat, joka on string-tyyppinen listamuuttuja (Kuva 13).

```
List<string> maat = new List<string>() { "Lisää", "Kaikki", "Maat", "Mitä", "Haluat", "Käyttää", "Tähän" };
```

Kuva 13. Listan arvojen määrittäminen.

Samalla tavalla luodaan string-tyyppiset listat joukkueiden nimivaihtoehdoista, pelaajien pelipaikoista, pelaajien kätisyydestä, sekä jokaisen maan etunimistä, sukunimistä ja kaupungeista.

Näihin kahteen päälistaan eli ”joukkueet ja pelaajat” luodaan pelaajat Unityn kautta, joko painamalla painiketta. Tämä luo kaikki joukkueet ja/tai pelaajat kerrallaan tai sitten pelaaja voi itse luoda kaikki tai halutessaan pelaaja voi luoda osan itse ja loput painiketta painamalla. Tällöin loput saavat satunnaiset arvot eri listoista. Aluksi tulee luoda oliomuuttujat painikkeille, pudotusvalikoille, syöttö- ja tekstikentille. Tämä onnistuu samalla tavalla kuin aikaisemmin luotiin paneelinvaihto scriptiin oliomuuttujat (Kuva 8), eli tehdään kaikista julkisia, niin voi Unityn kautta määrittää oikean objektin oikealle oliomuuttujalle (Kuva 7). GameObjectin tilalle vaihdetaan Button, jos painikkeesta kyse, Dropdown jos alasve-

tovalikosta kyse, InputField jos syöttökentästä kyse, Text jos tekstikentästä kyse ja Slider jos liukusäätimestä kyse. Luodaan siis muuttujat jokaiselle Unityssa luodulle painikkeelle, alasvetovalikolle, liukusäätimelle, syöttö- ja tekstikentälle.

Kun kaikki tarvittavat muuttujat on lisätty, alustetaan kaikki painikkeet Start-metodissa eli kun peli käynnistyy, scripti ilmoittaa minkä metodin painike tekee, kun sitä painaa (Kuva 14).

```
public Button rngNappi;
void Start()
{
    rngNappi.onClick.AddListener(metodiJokaTapahtuuNappiaPainamalla);
}
public void metodiJokaTapahtuuNappiaPainamalla()
{
    //...Koodia
    //...Koodia
}
```

Kuva 14. Painikkeiden painamisen määrittäminen.

4.4.1 Oikean collection-tyypin valinta

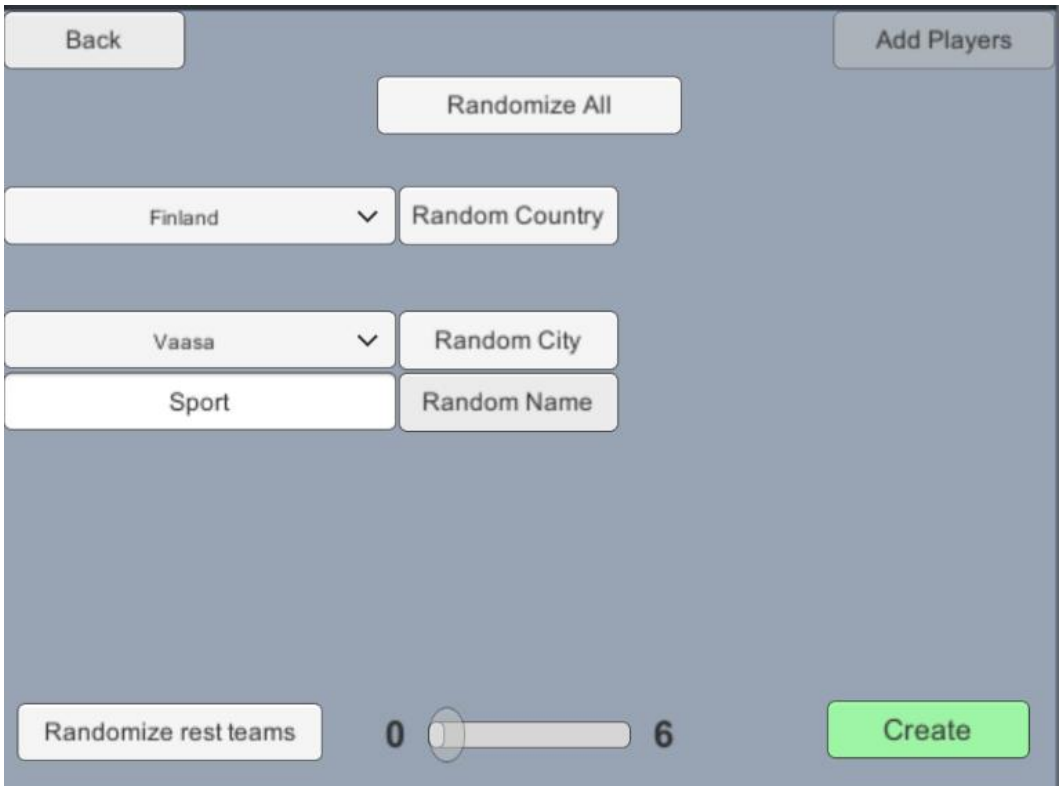
Unityssa on kahdeksan erilaista collection-tyyppiä, joista pitää valita mitä haluaa käyttää:

1. Javascript Arrays, joita voi käyttää vain Javascriptissä (Unity Technologies, 2018c), joten tältä osin tämän vaihtoehdon voin unohtaa.
2. Built-in Array, jota suunnittelin aluksi käyttävän, sen nopeuden takia, mutta built-in array on vaikeakäyttöinen, jos et tiedä arrayn kokoa ja sen koon täytyy voida muuttua pelin aikana (Unity Technologies, 2018c). Tällä perusteella tämä vaihtoehto voidaan sulkea pois.
3. Hashtable, jolla voi tallettaa avainpareja (Unity Technologies, 2018d). Ei ole tarvitsemaamme tarkoitukseen sopiva collection-tyyppi.
4. Generic List, jonka kokoa voi muokata tarvittaessa, tietoja voi siis lisätä ja muokata myöhemminkin. Geneeristen listojen avulla myös tietojen tallentaminen ja tiedoston serialisoiminen kävi yllättävänkin helposti.

5. Generic Dictionary on hashtablen tapanen listä, mutta monipuolisempi. Tähän vaihtoehtoon talletetaan myös avainsanoja (Unity Technologies, 2018e).
6. 2D Array, joka ei ole tähän käyttötarkoitukseen sopiva.
7. Hashset, joka on joukko objekteja, joten tämäkään ei ole tähän käyttötarkoitukseen sopiva.

4.4.2 Joukkueiden luominen

Jos joukkue määrä on valittu, niin pelaaja voi valita kahdesta eri painikkeesta luoko hän kaikki joukkueet ja pelaajat automaattisesti vai haluaako hän luoda itse kaikki. Pelaaja voi myös luoda vain osan itse ja loput automaattisesti. Jos pelaaja valitsee, että kaikki luodaan automaattisesti, pääsee pelaaja suoraan pelin aloitusruutuun. Jos pelaaja haluaa luoda itse joukkueita ja/tai pelaajia, avautuu joukkueiden luontipaneeli (Kuva 15), josta ei pääse eteenpäin ennen kuin kaikki joukkueet on tehty, mitkä määriteltiin aikaisemmin (Kuva 12). Joukkueet voi luoda joko painamalla Randomize rest teams -painiketta, jolloin kaikkiin kenttiin määräytyy satunnaiset arvot, niin monta kertaa, että kaikki tarvittavat joukkueet on luotu. Toinen vaihtoehto on tehdä, joko kaikki joukkueet itse tai vaikka vain osan, jonka jälkeen voidaan painaa Randomize rest teams -painiketta, jolloin peli luo satunnaisesti loput joukkueet. Back -painikkeesta pääsee takaisin aloitusvalikkoon, mikä on sama joka, aukeaa pelaajan avatessa pelin. Randomize All -painike antaa kaikille kentille uuden satunnaisen arvon ja muista random-painikkeista voi tehdä uuden satunnaisen arvon, sitä osoittavalle kentälle. Jos alasvetovalikosta pelaaja valitsee itse uuden arvon maakenttään, niin kaupunkikenttään tulee uusi satunnainen arvo kyseisen maan kaupunkilistasta.



Kuva 15. Joukkueen luomispaneeli.

Scriptin sisällä on määritetty, että jos pelajaa painaa Create-painiketta, joukkue lisätään joukkuelistaan. Eli ensimmäiseksi scriptin alussa on määritetty, jos painiketta painaa, mikä metodi tapahtuu (Kuva 14). Kun painaa Create-painiketta, metodi ottaa kentissä olevat arvot ja lisää ne joukkuelistaan uutena joukkueena (Kuva 16).

```
joukkueet.Add(new joukkue { //Määritetään uusi joukkue joukkuelistaan
    id = joukkueet.Count, //Saadaan id, joka kasvaa aina yhdellä
    maa = maaDropdownTxt.text, //Otetaan joukkueelle maa, maakentästä
    kaupunki = kaupunkiDropdownTxt.text, //Otetaan joukkueelle kaupunki, kaupunkikentästä
    nimi = nimiInput.ToString() }); //Otetaan joukkueelle nimi, nimikentästä
```

Kuva 16. Joukkueiden lisääminen scriptin sisällä.

Painamalla Random Country -painiketta scripti ajaa tälle painikkeelle määritetyn metodin, jossa ensimmäisenä tyhjennetään vaihtoehdot ja tämän jälkeen lisätään muuttujan maat, string-tyyppisestä listasta uudet vaihtoehdot alasvetovalikkoon, josta lopulta valitaan kenttään satunnainen arvo (Kuva 17). Eri maille voi päättää

myös prosentuaalisen määrän, millä todennäköisyydellä kyseinen maa tulee valituksi. Tällöin käytetään `if (Random.value <= 0.2)` -lausetta ja hakasulkeisiin tulee mikä maa saa tämän kyseisen prosentuaalisen arvon. Tämä esimerkkilauseke antaa 20% mahdollisuuden kyseiselle maalle.

```
public void maaSatunnainen()
{
    maaDrop.ClearOptions(); //Poistetaan vaihtoehdot
    maaDrop.AddOptions(maat); //Lisätään vaihtoehdot listasta maat
    maaDropJoukTextJouk.text = maat[Random.Range(0, maat.Count)]; //Otetaan satunnainen arvo listasta maat

    kaupunkiSatunnainen(); //Tekee metodin, joka arpoo satunnaisen kaupungin, kaupunkikenttään
}
```

Kuva 17. Satunnaisen maa arvon saaminen.

Kaupungille satunnaisen arvon saaminen tapahtuu muuten samalla lailla, kuin satunnaisen maan saaminen, mutta nämä alasvetovalikon vaihtoehdot riippuu siitä, mikä maa on valittuna maalle tarkoitetussa kentässä. Esimerkiksi jos valittuna on Finland, niin haetaan tiedot listasta, johon on lisätty Suomen kaupunkeja.

Lopuksi on vielä nimikenttä, joka on syöttökenttä eikä alasvetovalikko. Se tarkoittaa, että pelaaja ei näe valmiiksi määritettyjä vaihtoehtoja ja hänellä on mahdollisuus itse kirjoittaa tälle kentälle arvo. Tähän kenttään satunnainen arvo saadaan, kun haetaan satunnainen arvo string-tyyppisestä listasta, joka sisältää joukkueiden nimiä. Samalla tavalla siis, kuin listasta maat haettiin satunnainen arvo (Kuva 17).

Paneelin alhaalla (Kuva 15) pelaaja näkee liukusäädinvalikon joka kasvaa, aina kun joukkue luodaan ja kun liukusäädin näyttää, että se on täynnä, oikealla ylhäällä oleva Add Players -painike muuttuu painettavaksi ja kaikki muut paitsi Back-painike harmaiksi eli näitä harmaita painikkeita ei voi painaa. Tämän jälkeen pelaaja voi siirtyä pelaajien luomiseen.

Jos pelaaja haluaakin painaa painiketta mikä luo kaikki tai loput joukkueet. Silloin liukusäädin menee täyteen ja joukkueita ei voi enää luoda. Joten haluamat joukkueet pitää luoda ennen painikkeen painamista. Joukkueiden luominen aloitetaan laskemalla, montako joukkuetta tarvitaan ja montako vielä puuttuu. Tämän jälkeen määritetään joukkueet-listalle uusi joukkue, jossa kaikille kentille on määrät-

ty satunnaiset arvot ja tämä tehdään, niin monta kertaa, että kaikki joukkueet on luotu (Kuva 18).

```
for (int i = joukkueet.Count; i < teamsSlid.maxValue; i++) //Lasketaan montako joukkuetta on ja montako joukkuetta tehdään
{
    if (teamsSlid.value < teamsSlid.maxValue) //Jos joukkueita on vähemmän mitä on määritetty
    {
        string rngMaa = maat[Random.Range(0, maat.Count)]; //Luo satunnainen maa joukkueelle

        string rngKau = ""; //Määritellään tyhjä string-tyyppi myöhempää käyttöä varten
        string rngNimi = joukNimet[Random.Range(0, joukNimet.Count)]; //Luo satunnainen nimen joukkueelle joukNimet-listassa, missä on joukkueiden nimiä

        if (rngMaa.Contains("Finland"))
        {
            rngKau = kaupunkiFIN[Random.Range(0, kaupunkiFIN.Count)]; //Luo satunnaisen kaupungin joukkueelle maan perusteella
            joukkueet.Add(new joukkue { id = i, maa = rngMaa, kaupunki = rngKau, nimi = rngNimi, kokoNimi = rngKau + " " + rngNimi }); //Lisää joukkueen
        }
        else if (rngMaa.Contains("Sweden"))
        {
            rngKau = kaupunkiSWE[Random.Range(0, kaupunkiSWE.Count)];
            joukkueet.Add(new joukkue { id = i, maa = rngMaa, kaupunki = rngKau, nimi = rngNimi, kokoNimi = rngKau + " " + rngNimi });
        }
    }
    else
    {
        return;
    }
}
```

Kuva 18. Kaikkien joukkueiden luominen painiketta painamalla.

4.4.3 Pelaajien luominen

Pelaajien luominen toimii melko samalla tavalla kuin joukkueiden luonti eli pelaaja voi luoda kaikille joukkueille tarvittavan määrän pelaajia tai sitten luoda itse osan pelaajista. Joukkueiden minimi pelaajamäärä on 20 ja kun kaikilla joukkueilla on 20 pelaajaa, niin pelaaja voi mennä seuraavaan kohtaa eli valitsemaan joukkuetta jolla haluaa pelata (Kuva 19).

```
public void minimiPelaajaMaara()
{
    for (int i = 0; i < joukkueet.Count; i++) //Lasketaan kaikki joukkueet
    {
        if (joukkueet[i].pelaajat.Count > 20) //Lasketaan kaikkien joukkueiden pelaajat
        {
            tiiminValinta.interactable = true; //Jos kaikkien joukkueiden pelaajamäärä on suurempi kuin 20,
        }                                      //niin nappi joukkueen valintaan tulee näkyviin
        else
        {
            tiiminValinta.interactable = false; //Joukkueen valinta nappia ei voi painaa, jos kaikilla joukkueilla
        }                                       //ei ole vähintään 20 pelaajaa
    }
}
```

Kuva 19. Joukkueiden vähimmäis-pelaajamäärän laskeminen.

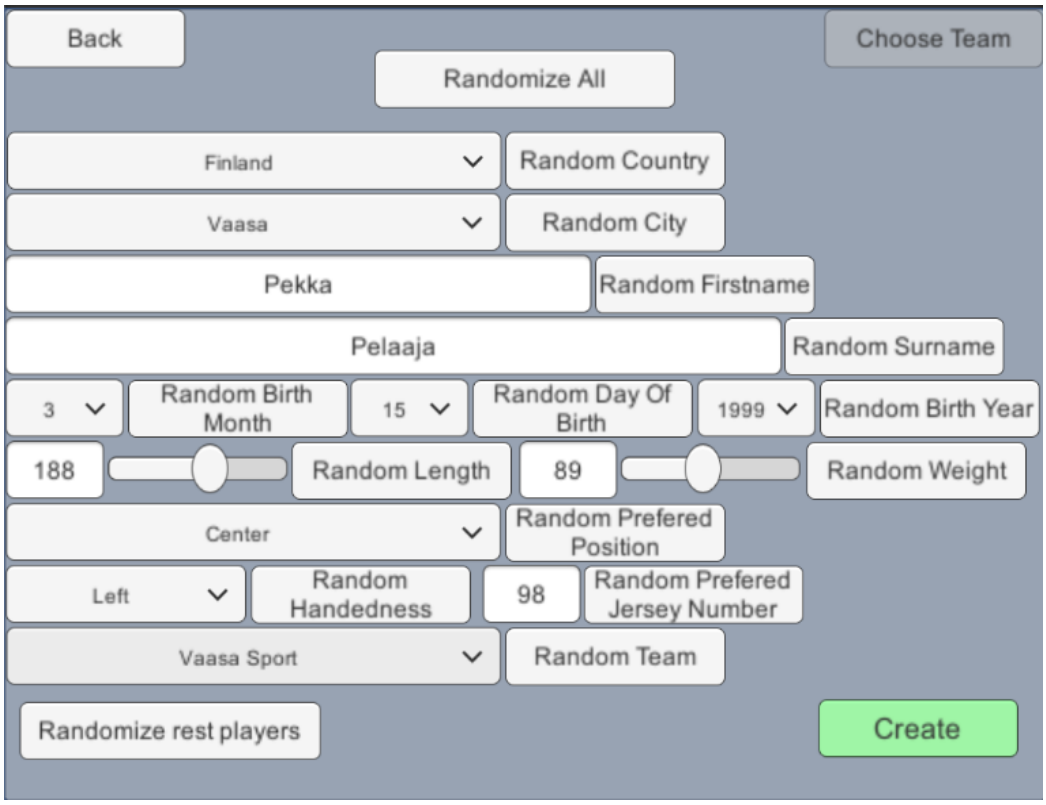
Joukkueiden maksimipelaajamäärä on 30 ja kun joukkueelle tulee 30 pelaajaa, joukkue katoaa joukkueen valintalistasta, eikä kyseiselle joukkueelle voi enää lisätä pelaajia. Jos kaikilla joukkueilla on 30 pelaajaa, niin pelaajia ei voi enää luoda ja pelaajan on siirryttävä joukkueen valintaan (Kuva 20).

```
List<string> joukDrop = new List<string>(); //Luodaan uusi lista alasvetovalikolle

foreach (joukkue j in joukkueet) //Lasketaan kaikki joukkueet
{
    if (j.pelaajat.Count >= 30) //Lasketaan jos joukkueella on 30 tai enemmän pelaajaa
    {
      joukkueDrop.ClearOptions(); //Tyhjennetään joukkueiden alasvetovalikko
    }
    else
    {
       joukDrop.Add(j.kokoNimi); //Lisätään joukkueet, joilla ei ole vielä 30 pelaajaa listaan
       joukkueDrop.ClearOptions(); //Tyhjennetään joukkueiden alasvetovalikko
       joukkueDrop.AddOptions(joukDrop); //Lisätän listan joukkueet alasvetovalikkoon
    }
    if (joukDrop.Count == 0) //Jos listassa ei ole enään yhtäkään joukkuetta
    {
       lisaa.interactable = false; //Pelaajia ei voi enään luoda
    }
    else
    {
       lisaa.interactable = true;
    }
}
```

Kuva 20. Joukkueiden maksimi pelaajamäärän laskeminen.

Pelaajien luomispaneeli näyttää nopeasti katsottuna melko sekavalta, kun ei ole visuaalista panosta ollenkaan tehty (Kuva 21). Paneelista kuitenkin löytyy kaikki, mitä tarvitaan pelaajien luomiseen ja kun muutaman kerran on luonut pelaajan ei se vaikuta enää sekavalta. Paneelista löytyy kaikki samat tavat luoda arvoja, kuin joukkueen luomispaneelistakin. Alasvetovalikkoja, syöttökenttiä, liukusäätimiä, sekä kaikille omat painikkeet, joilla saa kyseisiin kenttiin satunnaisen arvon. Randomize All -painikkeesta kaikki kentät saavat satunnaisen arvon ja Randomize rest players -painikkeesta pelaaja voi luoda kaikille joukkueille tarvittavan määrän pelaajia.



Kuva 21. Pelaajan luomispaneeli.

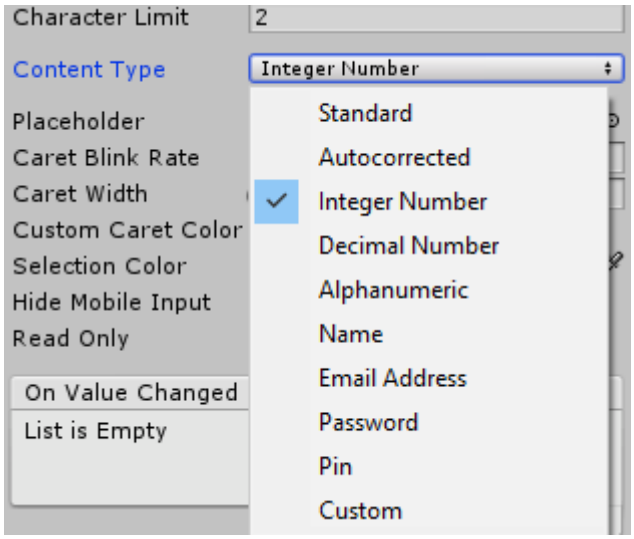
Maan ja kaupungin valinta tapahtuu samalla tavalla, kuin joukkueen luonnissakin (Kuva 17) eli kun vaihtaa maata, kaupungin vaihtoehdot muuttuvat ja kenttä saa uuden arvon. Etu- ja sukunimi kentät toimivat samalla tavalla, kuin joukkueen luomisen nimenluominen eli on valmiiksi määritellyt listat jokaisen maan etu- ja sukunimille ja näistä saadaan satunnainen arvo painiketta painamalla tai pelaaja voi itse keksiä pelaajille etu- ja sukunimet. Pelinumeron pelaaja voi itse määrittää tai painiketta painamalla saada satunnaisen numeron yhden ja 99 väliltä (Kuva 22).

```
peliNroInput.text = Random.Range(1, 99).ToString();
```

Kuva 22. Satunnaisen arvon saaminen pelinumero-kenttään.

Tähän kenttään pystyy ainoastaan syöttämään numeroita ja maksimissaan kaksi numeroa. Tämän asetuksen on saanut määrittää suoraan Unityn kautta syöttökentän asetuksista (Kuva 23) eli sijoitetaan Character Limit -kenttään 2 eli pelaaja voi

antaa maksimissaan kaksi numeroa ja Content Type -alasvetovalikosta valitaan Integer Number eli pelaaja voi antaa vain numeroita kyseiseen kenttään.



Kuva 23. Syöttökentän asetuksien näkymää.

Pituuden ja painon määrittäminen käy samalla tavalla kuin pelinumeron määrittäminen, paitsi että kenttään ei voi itse syöttää arvoa vaan liukusäädintä liikuttelemalla valitaan arvot. Pituudeksi voi valita arvon 164 ja 205 väliltä, painoksi voi valita arvon 64 ja 120 väliltä. Nämä arvot saa määriteltyä Unityn liukusäätimen asetuksien kautta (Kuva 24).

Kuva 24. Liukusäätimen minimi- ja maksimiarvo.

Syntymävuodeksi voi valita alasvetovalikosta, minkä tahansa vuoden 1968 ja 2002 väliltä. Syntymäkuukaudeksi voi valita alasvetovalikosta, minkä tahansa numeron yhden ja kahdentoista väliltä, eli käytössä on kaikki kuukaudet. Unityssa alasvetovalikon inspectorin kautta on määritelty kentälle vaihtoehdot yhdestä kahteentoista. Syntymäpäivän vaihtoehtoja ei voida määrittää Unityn kautta, koska kuukauden päivämäärä riippuu siitä mikä kuukausi on valittuna. Määritetään scriptin kautta päivän alasvetovalikkoon arvot eli jos on helmikuu eli toinen valittu, syntymäpäivän alasvetovalikkoon tulee vaihtoehdot 1-28 jos taas on huhtikuu,

kesäkuu, syyskuu tai marraskuu valittuna, niin syntymäpäivän alasvetovalikkoon tulee vaihtoehdot 1-30, jos jokin muu kuukausi on valittuna, vaihtoehdot ovat 1-31 (Kuva 25).

```
if (syntymaKkText.text == "2")
{
    List<string> pv1 = new List<string>() { "1", "2", "27", "28" };//Kaikki numerot tällä tavalla
    syntymaPvDrop.ClearOptions();
    syntymaPvDrop.AddOptions(pv1);
    syntymaPvText.text = Random.Range(1, 28).ToString();
}
else if (syntymaKkText.text == "4" || syntymaKkText.text == "6" || syntymaKkText.text == "9" || syntymaKkText.text == "11")
{
    List<string> pv2 = new List<string>() { "1", "30" };//Kaikki numerot tällä tavalla
    syntymaPvDrop.ClearOptions();
    syntymaPvDrop.AddOptions(pv2);
    syntymaPvText.text = Random.Range(1, 30).ToString();
}
else
{
    List<string> pv3 = new List<string>() { "1", "31" };//Kaikki numerot tällä tavalla
    syntymaPvDrop.ClearOptions();
    syntymaPvDrop.AddOptions(pv3);
    syntymaPvText.text = Random.Range(1, 31).ToString();
}
```

Kuva 25. Päivien määrittäminen valitun kuukauden perusteella.

Nyt oikeiden päivien määrittäminen toimii, jos painiketta painetaan, mutta se ei vaihdu vielä reaaliaikaisesti, jos kuukautta vaihtaa valitsemalla vaihtoehdon alasvetovalikosta. Tehdään siis metodi, joka tehdään koko ajan eli laitetaan se void Updaten sisälle. Luodaan scriptin alkuun tyhjä string-tyyppinen muuttuja ja metodin sisälle string arvo, jonka arvo on sama kuin syntymäkuukausi-kentän teksti ja aina kun tyhjän stringin arvo muuttuu, niin luodaan uudet päivät syntymäpäivän alasvetovalikkoon samalla tavalla kuin aikaisemminkin (Kuva 25). Tämän jälkeen annetaan tyhjälle stringille arvoksi tämä arvo-muuttujan arvo eli tämän hetkinen kentän arvo ja aina kun tulee kuukausi, jossa on erimäärä päiviä kuin nykyisessä, niin syntymäpäivän alasvetovalikon vaihtoehdot muuttuvat. Pelipaikan ja kätisyyden määrittäminen tapahtuu valmiiksi luotujen listojen avulla, samalla tavalla kuin maan luominen (Kuva 17). Kätisyys-lista sisältää vasen ja oikea arvot, joista arvotaan pelaajan kätisyys. Pelipaikka-lista sisältää kaikki mahdolliset pelipaikat eli maalivahti, vasen ja oikea puolustaja, sekä hyökkääjä ja keskushyökkääjä. Pelipaikoille ei ole määritelty mitään rajaa, koska pelin on tarkoitus olla ”kaljaliiga” eli harrastetasoisen liigan simulaattori, jossa esimerkiksi jollakin joukkueella voi olla kaikki pelaajat maalivahteja. Tämä kenttä tarkoittaakin, että jos pelaaja pelaa kyseisellä paikalla saa hän ominaisuuksiin, jonkin verran lisäystä. Pelaajien ominaisuudet määräytyvät satunnaisesti yhden ja 99 väliltä. Näitä arvoja edes pelaaja

ei voi muokata, eikä pelaaja myöskään näe näitä arvoja mistään. Joukkueen valinnan alasvetovalikon vaihtoehdot saadaan laskemalla kaikki joukkueet ja lisäämällä ne erilliseen listaan joukkueen kaupungin ja nimen kanssa (Kuva 20), jos joukkueella on yli 30 pelaajaa, niin poistuu se vaihtoehto listasta.

4.4.4 Kaikkien pelaajien luominen painiketta painamalla

Jos pelaaja painaa painiketta, josta luodaan loput tarvittavat pelaajat, niin lasketaan montako joukkuetta ja pelaajaa on yhteensä ja paljonko pelaajia on eri joukkueissa. Maan ja kaupungin luominen käy samalla tavalla kuin kaikkien joukkueiden luomisessa (Kuva 18) etu- ja sukunimen luominen samalla tavalla kuin joukkueen nimen luomien, paitsi nimien listavaihtoehdot määräytyvät maan mukaan. Syntymävuosi ja -kuukausi määritellään `Random.Range` toiminnon avulla ja syntymäpäivä `Random.Range`:n avulla niin, että arvot vaihtuvat riippuen siitä mikä kuukausi on valittu (Kuva 26).

```
syntymaVuosi = Random.Range(1968, 2002); //Vuoden määrittäminen
syntymaKk = Random.Range(1, 12); //Kuukauden määrittäminen
if (syntymaKk == 1 || syntymaKk == 3 || syntymaKk == 5 || syntymaKk == 7 || syntymaKk == 8 || syntymaKk == 10 || syntymaKk == 12)
{
    syntymaPv = Random.Range(1, 31); //Päivän määrittämien, vaihtoehto 1
}
else if (syntymaKk == 4 || syntymaKk == 6 || syntymaKk == 9 || syntymaKk == 11)
{
    syntymaPv = Random.Range(1, 30); //Päivän määrittämien, vaihtoehto 2
}
else
{
    syntymaPv = Random.Range(1, 28); //Päivän määrittämien, vaihtoehto 3
}
```

Kuva 26. Syntymäajan määrittämien.

Pituus luodaan `Random.Range`:n avulla eli saadaan satunnainen arvo 164 ja 205 väliltä ja paino luodaan pituuden perusteella. Jos pelaaja esimerkiksi on alle 175 senttimetriä pitkä, niin paino saa satunnaisen arvon 64 ja 95 väliltä, jos taas pelaaja on yli 190 senttimetriä pitkä, niin pelaaja satunnaisen painon 90 ja 120 väliltä, muussa tapauksessa painon arvo saadaan 75 ja 2015 väliltä. Joukkue pelaajalle saadaan luomalla lista, jonne lisätään kaikkien tehtyjen joukkueiden koko nimet ja tästä listasta `Random.Range`:n avulla saadaan satunnainen arvo. Pelipaikka ja kätisyys saadaan hakemalla satunnainen arvio niille tehdyistä listoista. Pelinumero saa satunnaisen arvon yhden ja 99 väliltä. Kun kaikkiin kenttiin on saatu arvot, niin lisätään pelaaja (Kuva 27).

```
pelaajat.Add(new pelaaja //Uuden pelaajan lisäys pelaajat-listaan
{
    id = p,
    kansalaisuus = rngMaa,
    kaupunki = rngKaupunki,
    etunimi = rngEtunimi,
    sukunimi = rngSukunimi,
    syntymaKk = syntymaKk,
    syntymaPv = syntymaPv,
    syntymaVuos = syntymaVuosi,
    pituus = pituus,
    paino = paino,
    peliNro = Random.Range(1, 99),
    pelipaikka = pelipaikat[Random.Range(0, pelipaikat.Count)],
    katisyys = katisyys[Random.Range(0, katisyys.Count)],
    joukkue = rngJouk
});
```

Kuva 27. Pelaajan lisääminen pelaajat listaan.

Jos pelaajien lukumäärä joukkueessa on 27, kyseiselle joukkueelle ei enää luoda pelaajia. Tällöin käyttäjä voi lopuksi lisätä vielä muutamia pelaajia joukkueille, maksimi pelaajamäärän ollessa 30. Pelin voi kuitenkin aloittaa, vaikka ei joukkueella ole 30 pelaajaa, kunhan on enemmän kuin 20 pelaajaa. Metodin lopussa juoksutetaan vielä toinen metodi, joka luo joukkueiden pelaajalistoihin kaikki pelaajat, jotka on määritelty kyseiselle joukkueelle (Kuva 28).

```
foreach (joukkue j in joukkueet) //Haetaan jokainen joukkue
{
    j.pelaajat = new List<pelaaja>(); //Määritetään jokaiselle joukkueelle omat listat pelaajille
    for (int i = 0; i < pelaajat.Count; i++) //Lasketaan kaikki pelaajat
    {
        if (j.kokoNimi == pelaajat[i].joukkue) //Tarkistetaan jos pelaajan joukkue vastaa joukkueen kokonimeä
        {
            j.pelaajat.Add(new pelaaja { //Lisätään jokaiselle joukkueelle pelaajat erilliseen joukkueen pelaajat listaan
                id = pelaajat[i].id,
                kansalaisuus = pelaajat[i].kansalaisuus,
                kaupunki = pelaajat[i].kaupunki,
                etunimi = pelaajat[i].etunimi,
                sukunimi = pelaajat[i].sukunimi,
                syntymaKk = pelaajat[i].syntymaKk,
                syntymaPv = pelaajat[i].syntymaPv,
                syntymaVuos = pelaajat[i].syntymaVuos,
                pituus = pelaajat[i].pituus,
                paino = pelaajat[i].paino,
                peliNro = pelaajat[i].peliNro,
                pelipaikka = pelaajat[i].pelipaikka,
                katisyys = pelaajat[i].katisyys,
                joukkue = pelaajat[i].joukkue,
            });
        }
    }
}
```

Kuva 28. Pelaajan määrittäminen oikeisiin joukkueisiin kuuluviin listoihin.

4.5 Tallennustavan valinta

Tarkoituksena oli löytää mahdollisimman sopiva tapa, jolla manageripelin tietoja voisi tallentaa, niin että ne voisi myös serialisoida eli niitä ei voida muokata. Vaihtoehtoja tallentamiseen on monta erilaista, mutta itse kokeilin seuraavia ennen kuin löysin halutun tavan:

1. SQL eli loin relaatiotietokannan, jonne tallensin tietoja. Tästä kävi kuitenkin ilmi, että jos peliä pelaa monta pelaajaa samaan aikaan, niin kaikilla olisi sama tietokanta siellä ja muokkautuisi kaikkien pelit samaan aikaan. Tämä on siis huono vaihtoehto pelille, jota pelataan offlinessa.
2. XML ja XmlSerializer kuulosti hyvältä vaihtoehdolta, mutta en saanut sitä missäkään vaiheessa toimivaan halutulla tavalla. Tiedot sain tallennettua, mutta niitä pystyi muokata, kun avasin tiedoston Notepad++:lla.
3. JsonUtility kaatui siihen, kun se ei tukenut Unityn geneerisiä listoja. Sain tallenettua ainoastaan yhden joukkueen tiedot ja muuta Unityn JsonUtilityn avulla ei pystynyt tallentamaan vaan olisi pitänyt vaihtaa Built-In arrayhin, mutta se olisi ollut liian aikaa vievä operaatio, joten hylkäsin tämänkin vaihtoehdon. Listat olisi kuitenkin saanut toimimaan, jos olisi luonut scriptin ja koodannut jsonin uudestaan.

4. BinaryFormatter. ja FileStream. Lopulta löysin Unityn luoman tutoriaali live videon 2014-vuodelta, jossa demonstroitiin hyvin ja näytettiin koodiesimerkki, mitenkä saa tallennuksen, lataamisen, serialisoinnin ja deserialisoinnin tehtyä BinaryFormatterin ja FileStreamin avulla (Unity Technologies 2018f).
5. Luodaan tiedoista Excel-tiedosto eli .xlsx. Tämä on mielenkiintoisen oloinen tapa myös, joka olisi voinut sopia tähän peliin. Tätä en kuitenkaan ehtinyt koittaa, kun löysin tämän tavan vasta, kun olin saanut jo BinaryFormatterin ja FileStreamin avulla tallennettua ja serialisoitua tiedostot onnistuneiksi.

4.6 Serialisointi ja tallentaminen

Serialisointia tehdessä on hyvä varmistaa, että serialisoitavat tiedot on määritelty julkisiksi (public) ja se sisältää oikeanmuotoisia tietotyyppejä, joita voidaaan serialisoida. Serialisoitavia tietoja ei saa määrittää static-, const- tai readonly-tietotyypeiksi (Unity Technologies, 2018g). Aiemmin kerrottiin luokista, jotka serialisoidaan (Kuva 10). Luokat, jotka määriteltiin serialisoitaviksi ovat joukkue, pelaajat ja pelaajien ominaisuudet. Tietojen serialisoinnissa tehdään tiedostosta sellainen, että se ei ole ihmisluettava ja täten tietoja ei voi muokata tai katsoa tallennuksen kautta (Kuva 29).

Kuva 29. Notepad++:an näkymää serialisoidusta tiedostosta.

Tietojen tallentamiseksi ja serialisoitavaksi on luotava metodi ja käytettävä C#:pin `BinaryFormatter` serialisointi tapaa ja `FileStream` tapaa. Eli luodaan `BinaryFormatter` muuttuja, sekä `FileStream` muuttuja, jolle määritellään saman tien tallennettavan tiedon polku, sekä tiedoston nimi. Tallennettavan tiedoston tyyppi voi olla mikä tahansa vaikka .abcdefghij tai minkä nimiseksi tiedostotyypin ikinä haluaakaan nimetä. Tämän jälkeen serialisoidaan tallennettaviksi määritetyt joukkueet. Lopuksi vielä suljetaan avoinna oleva tiedosto (Kuva 30).

```
public void Save()
{
        BinaryFormatter bfj = new BinaryFormatter(); //Luodaan BinaryFormatter muuttuja
        FileStream filej = File.Create(Application.persistentDataPath + "/joukkueet.dat");
                                        //Määritellään tallennettavan tiedoston polku ja nimi

        bfj.Serialize(filej, joukkueet); //Serialisoidaan tallennettavat joukkueet
        filej.Close(); //Suljetaan tiedosto
}
```

Kuva 30. Joukkueiden tallentaminen ja serialisointi.

4.7 Deserialisointi ja lataaminen

Deserialisointi ja lataaminen käyvät myös helposti `BinaryFormatter`:in ja `FileStream`:in avulla. Luodaan metodi ja `if`-lauseella etsitään, löytyykö kyseistä tiedostoa, eli tiedoston nimen on oltava sama mikä on tallennettu aiemmin. Tämän jälkeen luodaan `BinaryFormatter`-muuttuja ja `FileStream` muuttuja, jolle määritel-

lään polku, josta ladattava tiedosto löytyy, sekä tiedoston nimi mikä ladataan. Kun nämä kaikki on tehty, voidaan tiedosto deserialisoida ja siirtää ladattavan tiedoston tiedot joukkueet listaan ja lopuksi suljetaan tiedosto. Sama operaatio tehdään pelaajille, jonka jälkeen ajetaan metodi jossa sijoitetaan kaikki pelaajat oikeisiin joukkueisiin (Kuva 31).

```
public void Load()
{
    if (File.Exists(Application.persistentDataPath + "/joukkueet.dat")) //Etsitään löytyykö kyseinen tallennus
    {
        BinaryFormatter bfj = new BinaryFormatter(); //Luodaan BinaryFormatter muuttuja
        FileStream filej = File.Open(Application.persistentDataPath + "/joukkueet.dat", FileMode.Open);
        //Määritetään polku missä tallennettu tiedosto sijaitsee

        joukkueet = (List<joukkue>)bfj.Deserialize(filej);
            //Deserialisoidaan ladattava tiedosto ja listäään tiedot joukkueet listaan

        filej.Close(); //Suljetaan joukkueiden tallennustiedosto
    }

    if (File.Exists(Application.persistentDataPath + "/pelaajat.dat"))
    {
        BinaryFormatter bfp = new BinaryFormatter();
        FileStream filep = File.Open(Application.persistentDataPath + "/pelaajat.dat", FileMode.Open);
        pelaajat = (List<pelaaja>)bfp.Deserialize(filep);
        filep.Close(); //Suljetaan pelaajien tallennustiedosto
    }
    pelToJouk(); //Metodi jolla sijoitetaan pelaajat oikeisiin joukkueisiin
}
```

Kuva 31. Joukkueiden deserialisointi ja lataaminen.

5 JATKOKEHITYS

Peli on tarkoitus jatkaa loppuun, aina kun aikaa riittää tehdä. Tavoite olisi saada kokonainen manageripeli valmiiksi kahden vuoden sisällä. Dokumentoidessa työtäni tuli siitä esille muutamia virheitä ja uusia ideoita, mitä voisin tehdä toisin tai lisätä peliin. Tallentamisessa on ainakin parannettavaa, että pelaaja voisi itse määritellä tallentaessa tiedostolle nimen ja näin ollen voisi olla enemmän pelejä, joita pelaaja voisi pelata samanaikaisesti, kun nyt saa vain yhden tallennuksen, jota joutuu käyttämään. Tällöin ladatessa tiedostoa voisi myös päättää, minkä tiedoston pelaaja haluaa ladata. Peliä on tarkoitus jatkaa tekemällä aluksi InGameMainMenu-paneeli loppuun, mikä näkyy tynkänä pelin tämän version loppuvaiheessa. Tähän olisi tarkoitus lisätä nappeja, joista näkee joukkueiden rosterit, mahdollisuuden vaihtaa tai ostaa pelaajia, joukkueiden ja pelaajien tilastoja, mahdollisuuden selata vapaita pelaajia, tehdä jatkosopimuksia, muokata kenttiä ja taktiikoita, sekä kalenteri josta näkee otteluohjelman. Kun kausi on ohi, niin osa pelaajista lopettaa, osa siirtyy vapaaksi pelaajaksi ja muilla joukkueilla on mahdollisuus tehdä sopimus näiden kanssa. Vapaisiin pelaajiin ilmestyy samalla satunnaisesti generoituja pelaajia kauden loputtua. Pelaajien ominaisuudet ovat salattuja, joten pelaajan on tehtävä sopimuksia pelipaikan ja nimen perusteella.

6 YHTEENVETO

Henkilökohtaisesti olen hieman pettynyt, kun alun perin oli tarkoituksena saada kokonainen peli valmiiksi, mutta siinä on niin paljon tehtävää, että ei millään onnistu yhden miehen kokoonpanolla, graafisesta puolesta ja äänimaailmasta puhumattakaan. Kaiken lisäksi alkujoukkueiden ja pelaajien luonti vei niin paljon aikaa. Toisaalta jos olisin tekoälyn saanut tehtyä niin hyväksi, kun olen sen suunnitellut, en olisi sitä halunnut esitellä tekijänoikeuksiin vedoten. Loppujen lopuksi olen kuitenkin tyytyväinen, kun opin luomaan tietoa, jonka voi tallentaa ja serialisoida ja tämän jälkeen deserialsoida ja ladata uudelleen. Sain myös selville hyvän keinon luoda näitä tietoja, eli serialisoitavilla listoilla. Löysin myös listoihin liittyen, hyvän tavan tallentaa listojen tiedot ja tehdä niistä ei ihmisluettavaa serialisoimalla nämä tiedostot.

6.1 Ei suunnittelua

Peliä tehtäessä ei ollut missään vaiheessa tehty minkäänlaista kunnollista suunnittelua. Mentiin sen mukaan, mitenkä ideoita syntyi peliä luodessa. Suurin syy alun perin oli se, että peliä aloittaessa ei tiedetty minkälainen lopputulos halutaan. Tästä johtuen välillä, kun tehtiin jokin asian toisella tavalla ja sitten saatiinkin idea, että voitaisiinkin tehdä jokin uusi asia, mutta se edellyttää, että aiemmin tehty pitää tehdä kokonaan toisella tavalla, jotta tämä uusi idea toimisi. Tämän tyyppisiä tapauksia oli työssä liian monta, minkä takia aikaa kului periaatteessa turhaan saman asian tekemiseen toisella tavalla. Yksi esimerkki on, kun alun perin koitettin saada tallentamisen ja serialisoinnin tehtyä jsonin avulla, mutta Unityn JsonUtility ei tue lista-arraylla tallentamista, joten olisi jouduttu muuttamaan listaluokat, joksikin toiseksi ja tämä taas olisi tarkoittanut sitä, että olisi jouduttu melkein kaikki koodikin tekemään uudestaan. Pelissä oli aluksi myös switch-lauseita, mutta näistä jouduttiin myöhemmin siirtymään else if -lauseisiin, jotta saatiin koodista oikean haluttu arvo. Näin ollen voidaan suositella, että jos rupeaa tekemään jotain peliä tai projektia, kannattaa suunnitella ensiksi, ennen kuin alkaa työtä tekemään, koska muuten voi joutua tekemään paljon turhaa työtä, jos suunnittelu tapahtuu lennosta tulevien ajatusten pohjalta. Jatkokehityksen kannalta on kuitenkin tehty

jonkinlainen suunnitelma, että tämä ei tapahdu vain yhtäkkiä tulevien ajatusten pohjalta. Voihan suunnitelmat muuttuakin, jos jokin toinen idea tuntuu myöhemmin paremmalta ja tällöin poiketaan suunnitelmasta, mutta ainakin on suunnitelma, niin voi miettiä mitenkä tekee asiat, että seuraavatkin asiat toimivat kunnolla.

LÄHTEET

Gruber, D. 2000. So you want to be a Computer Game Developer?. Viitattu 18.11.2018. Fastgraph. http://www.fastgraph.com/makegames/intro.html

Norton, T. 2013. Learning C# by Developing Games with Unity 3D Beginner's Guide. Viitattu 18.11.2018. Packt http://netsites.mx/Unity3dTutorials/Learning%20C%23%20by%20Developing%20Games%20with%20Unity%203D%20Beginner's%20Guide%20%5BeBook%5D.pdf

GameDesigning. 2018. The Top 10 Video Game Engines. Viitattu 18.11.2018. https://www.gamedesigning.org/career/video-game-engines/

Stack Exchange Inc. 2018. Search bar. Viitattu 09.11.2018. https://stackoverflow.com

Unity Technologies. 2018a. Tutorials. Viitattu 09.11.2018. https://unity3d.com/learn/tutorials

Unity Technologies. 2018b. Past Events. Viitattu 09.11.2018. https://connect.unity.com/events

Unity Technologies. 2018c. Array. Viitattu 17.11.2018. https://docs.unity3d.com/ScriptReference/Array.html

Unity Technologies. 2018d. Hashtable. Viitattu 17.11.2018. https://docs.unity3d.com/ScriptReference/Hashtable.html

Unity Technologies. 2018e. Preparing The Use Item Dictionary. Viitattu 17.11.2018. https://unity3d.com/learn/tutorials/topics/scripting/preparing-use-item-dictionary

Unity Technologies. 2018f. Persistence – Saving and Loading Data. Viitattu 17.11.2018. https://unity3d.com/learn/tutorials/topics/scripting/persistence-saving-and-loading-data

Unity Technologies. 2018g. Script Serialization. Viitattu 25.11.2018. https://docs.unity3d.com/Manual/script-Serialization.html