Jussi Kangasvieri

Sosiaalisten ulottuvuuksien ja moninpeliominaisuuksien ohjelmointi Android-mobiilipelissä

Tradenomi
Tietojenkäsittely
Syksy 2024

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Kangasvieri Jussi

Työn nimi: Sosiaalisten ulottuvuuksien ja moninpeliominaisuuksien ohjelmointi Android-mobiilipelissä

Tutkintonimike: Tradenomi (AMK), tietojenkäsittely

Asiasanat: Peliala, peliohjelmointi, Android, moninpeli, sosiaaliset ominaisuudet peleissä

Opinnäytetyössä perehdyttiin videopeleissä tyypillisiin sosiaalisiin ulottuvuuksiin ja moninpeliominaisuuksiin. Työ keskittyi etenkin peliohjelmoijan perspektiiviin ja työssä pyrittiin tutkimaan ensisijaisesti Android-alustan peleistä löytyviä ominaisuuksia, mutta koska ominaisuudet ovat luonnoltaan pitkälti yleispäteviä, teoriatieto soveltuu myös muille alustoille. Teoriatiedosta pohjan muodostamisen lisäksi työssä oli tavoitteena myös soveltaa tätä tietoa ohjelmointiosiossa, jossa tarkkailtiin teoriaosuudessa tutkittuihin elementteihin liittyvää ohjelmointityötä.

Työn avauksessa tutkittiin joitain tilastoja, jotka liittyvät sosiaalisiin ulottuvuuksiin ja moninpeliominaisuuksiin videopeleissä. Niitä tarkkailemalla pystyttiin arvioimaan kyseisten videopelien elementtien merkitystä peliteollisuudessa. Teoriaosuus aloitettiin moninpelikokemuksen oleellisten osien esittelyllä, jossa tavoitteena oli saada tarkkaillusta tiedosta muodostettua laaja perusymmärrys. Sitä seurasi osuus, jossa käytiin läpi erinäisiä sosiaalisia ominaisuuksia. Teoria-aiheita seurasi soveltava osuus, jossa pyrittiin demonstroimaan joidenkin tutkittujen ominaisuuksien ohjelmointia. Soveltavan osuuden tarkoitus oli antaa peliohjelmoijalle näkemys siitä, millaista työ niiden parissa voi olla.

Työssä havaittiin, miten merkittäviä tarkastelun alaiset videopelien elementit ovat peliteollisuudessa. Tilastoja tutkimalla kävi ilmi, että nykypäivänä moni ihminen sosialisoituu juuri videopelien kautta. Täten eri sosiaalisten ulottuvuuksien lisäys peliin voi olla erityisen hyödyllistä pelaajien pysyvyyden ja vuorovaikutuksen rohkaisemiseen. Moninpeliominaisuuksia sisältävien videopelien onnistuminen markkinoilla taas kertoo niiden tarjoaman vuorovaikutuksen suosiosta. Eri moninpelikokemuksen elementtejä tarkkaillessa havaittiin myös se, miten monimuotoisia eri moninpeliominaisuudet ovat videopeleissä.

Peliohjelmoijan perspektiivistä havainnot kertoivat siitä, miten hyödyllistä kyseisten ominaisuuksien ymmärtäminen on nykypäivänä ja että se todennäköisesti tulee olemaan vain merkityksellisempää tulevaisuudessa. Soveltavassa osuudessa päästiin keskittymään juuri ohjelmointiin, sillä olemassa olevia palveluita käyttäen voitiin ohittaa mahdollinen haastava pohjustava työ. Tämä edisti etenkin kyseisten ominaisuuksien ohjelmoinnista tietämättömän kykyä päästä harjaantumaan niiden kanssa työskentelemisessä.

KAMK • University of Applied Sciences

Abstract

Author(s): Kangasvieri Jussi

Title of the Publication: The Programming of Social and Multiplayer Features in an Android Mobile Game

Degree Title: Bachelor of Business Administration, business information technology

Keywords: Game industry, game programming, Android, multiplayer, social features in games

The goal of this thesis was to delve into various social and multiplayer features that are typical in video games. The work focused primarily on the perspective of a game programmer and was aimed particularly at features found in games on the Android platform, but due to the largely universal nature of the features in question, the knowledge is applicable to other platforms as well. Beyond forming a baseline understanding of the theory on the topics, the work also aimed to apply this in a practical section, in which the programming work involved was demonstrated.

The introduction inspected some statistics that are related to social and multiplayer features in video games, by which the significance of the features in the game industry could be evaluated. The theory section consisted of two parts. The first chapter went over the various aspects of a multiplayer game experience, aiming to form a broad baseline understanding of what goes into it. That was followed by a chapter exploring various social features. After these topics were covered, a practical section followed, which aimed to give a game programmer an idea of the programming work that goes into implementing the features discussed.

The work highlights the significance of social and multiplayer features in the video game industry. By studying statistics, it was discovered that many people socialize, especially through video games. Therefore, implementing various social features into a video game can encourage player retention and interaction. The market success of video games, including multiplayer elements, indicates the popularity of the form of gameplay they offer. While studying the various elements of a multiplayer game experience, it was also noticed how multiform they can be in video games.

From the perspective of a game programmer, the findings indicate the usefulness of understanding these features in the modern day, and the likely growth of their significance in the future. In the practical section, the use of pre-existing services made it possible to focus specifically on the programming work, as it allowed bypassing any potential challenging preliminary work. This promoted the accessibility of programming work involving these features especially to someone who is not initiated in it.

Sisällys

Symboliluettelo

Callback – Ohjelmoinnissa callback on sellainen funktio, joka kutsutaan reaktiona tietyn ehdon tai tilan toteutumiselle.

Käyttöliittymä (User Interface, UI) – Videopeleissä käyttöliittymä sisältää ne fyysiset ja graafiset elementit, joiden kautta pelaaja voi vaikuttaa peliin.

Matchmaking – Videopeleissä matchmaking on pelaajien saattaminen yhteen moninpelisessiota varten.

MMORPG (Massively Multiplayer Online Roleplaying Game) – Videopeli, johon voi osallistua samanaikaisesti todella suuri määrä pelaajia.

NPC (Non-Playable Character) – Ei-pelaajahahmo. Hahmot pelimaailmassa, jotka eivät ole pelaajien hallitsemia.

PvE (Player versus Environment) – Pelaaja vastaan ympäristö. Peli, jossa oleellista on keskittyminen pelaajien väliseen kilpailuun.

PvP (Player versus Player) – Pelaaja vastaan maailma. Peli, jossa oleellista on keskittyminen pelimaailman tarjoamien haasteiden kohtaamiseen.

PvPvE (Player versus Player versus Environment) – Pelaaja vastaan pelaaja vastaan ympäristö. Peli, jossa oleellista on sekä pelaajien kilpailu keskenään että pelimaailman tarjoamat haasteet.

Rajapinta (Interface) – C#-ohjelmointikielessä rajapinta on ominaisuus, joka määrittää kokoelman funktioita ja ominaisuuksia, jotka rajapinnan perivän luokan on implementoitava. Tällä saadaan aikaiseksi johdonmukainen käytös ja keskustelu eri luokkien välillä.

Skene (Scene) – Unity-pelimoottorin kontekstissa skene on se osa peliä, joka sisältää muita pelin osia ja tapahtumia, kuten maailman ja hahmot.

1 Johdanto

Moninpelit ovat jatkuvassa kasvussa. Pelimoottori Unityn julkaisijoiden tekemän tutkimuksen mukaan vuonna 2023 suuri osa pelinkehittäjiä keskittyi juuri moninpelien kehitykseen, vaikka se onkin yleensä haastavampaa kuin yksinpelien kehitys. Tutkimus sisälsi kyselyn, jossa 300 vastaajasta 68 % kehitti jonkinlaista moninpeliä. Siitä 62 prosenttiyksikköä oli verkkomoninpelejä, kun taas 6 prosenttiyksikköä oli offline-moninpelejä. Tämä kertoo samalla myös siitä, miten yleisesti ottaen verkkomoninpelit ovat jättäneet muut tyypit jälkeensä, mutta eivät kuitenkaan täysin, vaan myös offline-moninpeleillä on edelleen oma markkinarakonsa peliteollisuudessa. [1.]

Kun puhutaan alustariippumattomista peleistä, oli kyselyssä sellaisten moninpelien osuus vielä suurempi. Verkkomoninpelien osuus oli 69 prosenttiyksikköä, ja offline-moninpelien osuus oli kooltaan 7, eli yhteensä 76 % kyselyyn vastanneista kehitti jonkinlaista moninpeliä. Tämä trendi moninpelien kehityksen määrässä ei liene mikään yllätys, sillä moninpelien tuottavuus on myös jatkuvassa kasvussa.

VG Insightsin tekemän tutkimuksen mukaan vuonna 2023 yhteistyömoninpelejä julkaistiin Steamissa, PC-alustan suurimmassa digitaalisessa pelikauppapaikassa, noin 800, mikä vastasi noin 6 % kaikista pelijulkaisuista Steamissa sinä vuonna. Vaikka niiden osuus pelijulkaisuista oli 6 %, niiden osuus pelimyynneistä oli 36 %, mikä kertoo senkaltaisten pelien suuresta suosiosta pelaajien keskuudessa. Tutkimusdatan mukaan yhteistyömoninpelien mediaania myydään noin 38 tuhatta kopiota, siinä missä muunkaltaisen pelin mediaania noin 5 tuhatta. Kun pelit jaetaan myytävyydeltään neljään osaan myyntien perusteella, niin alin 25 % yhteistyömoninpeleistä myy keskimäärin kaksi kertaa niin paljon, kuin muunkaltaiset pelit, ja ylin 25 % myy keskimäärin 298 tuhatta kopiota siinä missä muun tyyppinen peli myy noin 21 tuhatta. Erot ovat suuret joka myyntiluokassa. [2.]

Kun keskitytään vain mobiilialustalle, nähdään myös sillä moninpelitrendin olevan jatkuvassa nousussa. Aikaisemmin käsitellyn Unityn tutkimuksen mukaan jonkinlaisen moninpeliominaisuuden omaavilla puhtaasti mobiilialustan peleillä oli noin 40 % enemmän aktiivisia käyttäjiä kuin sellaisilla, missä ei mitään moninpeliominaisuutta ollut [1]. Teknologian kehittyminen tarjoaa kyvyn aina vain paremman näköisten ja suurempien pelien toteutukseen mobiilipeliteollisuudessa [3]. Nykypäivänä mobiilipelit ovatkin suuri osa sekä peli- että yleisesti viihdeteollisuutta. Vaikka sen

tuottavuus onkin laskenut pienissä määrin muihin peliteollisuuden puoliin verrattuna koronapandemian jäljiltä, se on edelleen ylivoimaisesti tuottavin peliteollisuus. Peliteollisuudessa mobiilipelien osuus tuottavuudesta on 49 %, Yhdysvaltain dollareissa noin 89,9 miljardia. [4.]

Android on alustana johtavassa asemassa mobiilipelien kehitykseen monista syistä. Ensimmäinen lienee se, että se on laajalti käytössä oleva alusta. Vuoden 2024 toisella neljänneksellä Androidin markkinaosuus mobiilikäyttöjärjestelmistä oli 71,65 %, kun taas iOSin osuus oli 27,62 %, ja kaikkien muiden käyttöjärjestelmien osuus oli siis yhteensä alle yksi prosentti [5]. Google Play -palvelut tulevat valmiiksi asennettuna lähes kaikkiin Android-laitteisiin, jotka ovat Google Play -sertifioituja, eli ovat tarkastettu ja todettu olevan turvallisia ja pystyvän käyttämään kyseisiä palveluita [6; 7]. Tällaisia laitteita oli lähes 16 000 eri mallia vuonna 2018 [8].

Kyselyn mukaan Z-sukupolven jäsenistä ja milleniaaleista noin 40 % sosialisoituu enemmän videopeleissä kuin tosimaailmassa [9]. Myös alfasukupolvelle merkittävä osuus sosiaalisesta interaktiosta ovat mobiilipelit. Noin 69 % Z-sukupolvesta ja 73 % alfasukupolvesta pelaa mobiilipelejä [10]. Kummatkin sukupolvet myös arvostavat erilaisia sosiaalisia elementtejä pelissä, ja etenkin Z-sukupolven pelaajat suosivat niitä. Kaiken kaikkiaan keskustellusta datasta voidaan päätellä, että Android-alustalle pelien kehittäminen lienee tie suurille markkinoille. Niin sosiaaliset ulottuvuudet kuin myös moninpeliominaisuudetkin ovat merkittävä ja kasvava osa peliteollisuutta, joten niiden osaaminen on hyödyksi kehittäjälle.

Tämä opinnäytetyö alkaa tietoperustalla erinäisistä moninpeliominaisuuksista. Osiossa keskustellaan aiheista, kuten offline- ja verkkomoninpelien eroista, pelin kulusta ajallisesti ja pelaajien määrästä ja vuorovaikutuksesta. Moninpeliominaisuuksien keskustelusta opinnäytetyö jatkuu vastaavaan osuuteen myös sosiaalisista ominaisuuksista, jossa tutkitaan olemassa olevien sosiaalisten ominaisuuksien kirjoa. Näillä teoriaosioilla saadaan aikaiseksi yleistasoinen ymmärrys siitä, minkälaisia sosiaalisia ulottuvuuksia ja moninpeliominaisuuksia peleissä voi olla. Teoria-aiheita seuraa vielä soveltava osuus, missä tutkitaan sitä, miltä kyseisten elementtien ohjelmointi voisi videopelissä näyttää. Se on erityisesti ohjelmointia osaaville suunnattu. Siinä tutkitaan kirjoitettua koodia, missä käydään läpi, mikä ominaisuus on kyseessä, miten koodi toimii ja mahdollisia huomioita tai esimerkkejä jatkotyöhön. Työ tapahtuu Unity-pelimoottorissa, ja siinä käytetään hyödyksi Photon Unity Networking 2 -moninpelimoottoria sekä avoimen lähdekoodin Unity-lisäosaa, Google Play Games plugin for Unitya, joka tarjoaa toiminnallisuutta Google Play -ominaisuuksien kanssa. Olemassa olevat palvelut tarjoavat erinomaiset puitteet lähteä tutustumaan kyseisen työn tekoon. Soveltavan osuuden jälkeen seuraa vielä yhteenveto, jossa käydään läpi keskusteltuja aiheita ja tärkeimpiä poimintoja niistä.

2 Moninpeliominaisuudet

Aloitetaan moninpeliominaisuuksista keskustelu nostamalla esiin se, tapahtuuko peli verkon ylitse tai ilman sitä. Paikallinen offline-moninpeli on sellainen peli, jota useampi pelaaja voi pelata offline-tilassa yhtä aikaa. Se toimii samalla laitteella, joko samassa näkymässä tai niin, että näyttö on jaettu pelaajien kesken, ja kullakin on oma näkymänsä. Tyypillisiä pelejä, joissa on jaetulla näytöllä offline-pelattavuutta ovat esimerkiksi Mario Bros. -sarjan pelit. Siinä missä tietokonealustalla moninpelejä oli ollut myös lähi- ja myös täysin verkkomoninpeleinä jo vuosikymmeniä, konsolimoninpelit toimivat pelkästään tällä tavalla 2000-luvun alkuun asti, jolloin niihin alkoi ilmestymään kyky saada verkkoyhteys. [11; 12.]

Lähiverkkomoninpelit ovat sellaisia moninpelejä, joiden pelaamista varten pelaajat tyypillisesti kokoontuvat myös samaan paikkaan ja yhdistyvät lähiverkkoon, jonka kautta peliä pelataan [13]. Jotkin verkkomoninpelit toimivat pelkästään lähiverkon kautta, mutta tällaisetkin ovat suhteellisen harvinaisia nykyaikana, kun moninpelit toimivat myös laajemman verkon kautta.

Verkkomoninpelit yleisesti ovat sellaisia mitä voidaan pelata verkkoyhteyden kautta. Ensimmäisiä esimerkkejä senkaltaisista peleistä on esimerkiksi vuonna 1973 ilmestynyt peli Empire, joka toimi PLATO-järjestelmällä, tai vuoden 1980 MUD (multiuser dungeon), johon yhdistettiin ARPANETin kautta [14, 15]. Vaikka ne eivät täysin vastaakaan verkkomoninpelejä niiden nykyisessä muodossa, olivat ne silti merkittäviä moninpeliteoksia sen aikaisella teknologialla. 1980-luvulla ilmestynyt Internet ohitti lopulta aikaisen kilpailunsa, ja luonnollisesti myös verkkomoninpelit alkoivat toimimaan internetin kautta, kuten muukin julkinen liikenne. Ensimmäinen peli, jota voitiin pelata internetin kautta verkkomoninpelinä oli ensimmäisen persoonan PvP-räiskintäpeli Quake, julkaistu vuonna 1996. Siinä käytössä oli moninpelin isännöitsemiseen omistetut palvelimet, mutta pelaajilla oli myös mahdollisuus käyttää omia laitteitaan mukautettavina palvelimina. [16.]

1990-luvun loppupäässä ilmestyi myös yksi peliteollisuuden merkittävimmistä peligenreistä, eli MMORPG-pelit, joissa saattaa olla tuhansia, ellei miljoonakin, pelaajaa yhtä aikaa kirjautuneena peliin. 2000-luvulla taas moninpelien kirjoon liittyi tunnettuja räiskintäpelien sarjoja, kuten Battlefield ja Call of Duty, sekä alun perin yhteisön jäsenen Warcraft 3 -peliin luoman modifikaation “Dota - Defense of the Ancients” synnyttämä suuren suosin saavuttanut MOBA-genre, jossa muutaman pelaajan tiimit hallitsevat kukin yhtä suuresta kirjosta eri rooleihin erikoistuneita hahmoja ja pyrkivät päihittämään toisen tiimin kartalla, jossa on kolme linjaa, pelaajatiimien tukikohtien välillä, joita pitkin taistella. Alun perin yksinpeliin Half-Life modifikaationa luotu Counter-Strike

taas kehittyi sarjana yhdeksi kilpapelaamisen uranuurtajista 2010-luvulla, kun sarjan pelille Counter-Strike: Global Offensive alettiin järjestämään maailmanlaajuisia kilpapelaamistapahtumia, ja kilpapelaaminen onkin vuonna 2024 merkittävä osa peliteollisuutta. [17, 18]. Moninpelien, etenkin verkkomoninpelien historiaa, monimuotoisuutta ja jatkuvaa kasvua on vaikea kuvailla lyhyesti.

Alustariippumattomuus on moninpeleissä noususssa oleva trendi. Käytännössä se tarkoittaa sitä, että eri alustoilla, esimerkiksi PC- ja konsolialustalla pelaavat pelaajat, voivat kohdata toisensa pelissä. Tämä on noussut pelialla aina useammissa peleissä tavoitteeksi. Alustariippumattomuudelta odotetaan myös jatkossa kehitystä sen suhteen, että yhdellä alustalla tehty edistyminen pelissä kantautuu saumattomasti myös toisille alustoille, mutta tämä ei ole vielä laajalti käytössä tai helppoa. [19; 20.]

Joissakin moninpeleissä merkittävä ominaisuus on sen pelimaailman pysyvyys. Pysyvä pelimaailma tarkoittaa sitä, että maailma on jatkuvassa olemassaolossa ja muutoksessa, joten tämä tyypillisesti tarkoittaa pelipalvelimen olevan aina päällä. Etenkin MMORPG-peligenrelle on tyypillistä pysyvä keskeinen pelimaailma, joka on olemassa jatkuvasti, ja esimerkiksi NPC-hahmojen sijainnit tai resurssien kasvu päivittyvät, vaikkei yksikään pelaaja olisi pelissä. Tämä on yksi elementti MMORPG-pelien erikoisuutta. Muidenkin genrejen peleissä, esimerkiksi joissain selviytymispeleissä, voi olla vastaavanlaista toiminnallisuutta etenkin julkisilla palvelimilla, jotka ovat jatkuvasti päällä. [21; 22.]

Joissain peleissä voidaan maailmassa havaita eräänlaista ”pysyvyyttä”, sillä esimerkiksi talletettu maailma rooli-, hiekkalaatikko- tai suurstrategiapelissä jatkuu siitä tilanteesta, mihin se jäi. Se ei kuitenkaan muutu riippumatta siitä, onko siinä pelaajia, joten se ei vastaa todellista videopelimaailman pysyvyyden merkitystä. Joissain peleissä ei ole tällaistakaan pysyvyyttä, ja esimerkiksi seurapelien ja otteluihin perustuvien räiskintäpelien maailma alkaa tyypillisesti samasta tilanteesta kuin aikaisempienkin otteluiden tai erien alussa, ja pelaajien muutokset siihen poistuvat, jotta se on valmis uuteen käyttöön. Samoin myös peleissä, joiden keskeinen maailma on pysyvä, voi olla pienempiä sessioita, kuten tyrmässä seikkailu, jossa ei ole pysyvyyttä, vaan se alkaa jokaisella kerralla sen normaalista aloitustilanteesta.

2.1 Moninpelin ajoitus

Ensimmäinen kysymys moninpelin ajoituksessa on se, onko se synkroninen vai asynkroninen. Otetaan esimerkiksi peli, jossa on PvP-pelimuoto. Synkronisena pelimuoto vaatii pelaajien olevan samaan aikaan pelissä, jotta kyseinen interaktio voi tapahtua. Siinä siis pelin tapahtumat ja toisen pelaajan päätösten vaikutukset ilmenevät samanaikaisesti kummallekin pelaajalle. Synkronisuus on yleisempää moninpelattavissa videopeleissä kuin asynkronisuus etenkin PC- ja konsolialustalla. [23; 24.]

Asynkronisena taas kyseinen pelimuoto olisi sellainen, missä pelaajien ei tarvitse olla yhtä aikaa osallistumassa peliin, vaan he voivat omalla ajallaan tehdä omat päätöksensä kyseisessä pelimuodossa, ja päätösten vaikutus vastustajaan voi käydä ilmi vasta kyseisen pelaajan avatessa pelin. Etenkin mobiilipeleissä ainakin osittainen asynkroninen pelin toiminta on yleistä, koska mobiililaitteet yleensäkin sallivat pelaajan nopeasti avata pelin ja käydä pelaamassa hetken vaikkapa bussipysäkillä odottaessaan, ja pelisessiot ovat tyypillisesti lyhyempiä [25]. Esimerkki asynkronisesta pelaaja vastaan PvP-pelimuodosta löytyy pelistä Raid: Shadow Legends, jonka areenamuodon tarjoamasta listasta vastustajia on näkymä kuvassa 1. Pelissä on eräänlainen matchmaking, jossa otteluita voittamalla tai häviämällä saadaan pisteitä, ja peli tarjoaa vastustajiksi pelaajia, joilla on mahdollisimman vastaava pistemäärä.

Kuva 1. Raid: Shadow Legendsin PvP-pelimuodon lista mahdollisista vastustajista [26.]

Raid: Shadow Legendsissä pelaaja kasaa omista hahmoistaan puolustustiimin, jota vastaan muut pelaajat voivat hyökätä silloinkin, kun kyseinen pelaaja ei ole pelissä. Tekoäly hallitsee puolustustiimiä. Hyökkäävälle pelaajalle taistelun vaikutukset tulevat välittömästi, ja puolustustiimin omistavalle pelaajalle ne ilmenevät hänen avatessaan pelin.

Seuraavaksi nostetaan esiin kysymys siitä, onko peli vuoropohjainen vai reaaliaikainen. Vuoropohjainen peli on sellainen, missä kukin pelaaja saa oman vuoronsa, joka kestää niin kauan, kunnes pelaaja on valmis päättämään sen, kun taas reaaliaikaisessa pelissä pelaajat saavat tehdä asioita samanaikaisesti, ja yleensä jonkinlaisen aikarajan puitteissa tai muutoin suuremmassa ajallisessa paineessa [27]. Eri peligenreissä on yleensä yleisempää nähdä jompikumpi näistä. Esimerkiksi toimintapelit ja simulaatiopelit ovat yleisemmin reaaliaikaisia, kun taas etenkin erilaiset perinteiset strategia- tai taktiikkapelit ovat usein vuoropohjaisia. Silti sekä reaaliaikaisia että vuoropohjaisia pelejä löytyy useista eri genreistä, myös sellaisista, missä toinen tyyppi on dominoiva.

Reaaliaikaisia pelejä voidaan kuvailla olevan eräällä tavalla samankaltaista kuin tosielämässäkin toimiminen. Maailma liikkuu jatkuvasti eteenpäin, ja siinä tapahtuu koko ajan uusia asioita, jotka vaikuttavat tilanteeseen. Tietyt asiat vievät tietyn verran aikaa reaaliaikaisessa pelissä, aivan kuten esimerkiksi yhdestä paikasta toiseen käveleminen tosielämässäkin. [28.]

Vuoropohjaiset pelit sen sijaan painostavat syvempää, strategista pelaamista. Niille on tyypillistä suurempi määrä mikrohallinnointia ja huomattavasti monimutkaisemmat mekaniikat, joissa pelaajan kyky ennakoida ja suunnitella merkitsee. Pelinsisäisesti pelaaja voi tehdä tarkan pelisuunnitelman, koska peli etenee vuorojen pohjalta eikä sekunti sekunnilta. Aikaisemmin mainttu Raid: Shadow Legends on vuoropohjainen peli. Siinä vastustajien hahmotiimit toimivat kukin omalla vuorollaan, ja vuoroja ei ole aikarajattu. Hahmoilla on eri kykyjä, ja niiden oikeassa järjestyksessä ja -tavalla käyttö on tärkeää, etenkin haastavissa tehtävissä tai PvP-pelimuodossa. Tällöin pelin asynkronisuus ja rajaamattomien vuorojen pituuksien tarjoama aika suunnittelulle on eduksi.

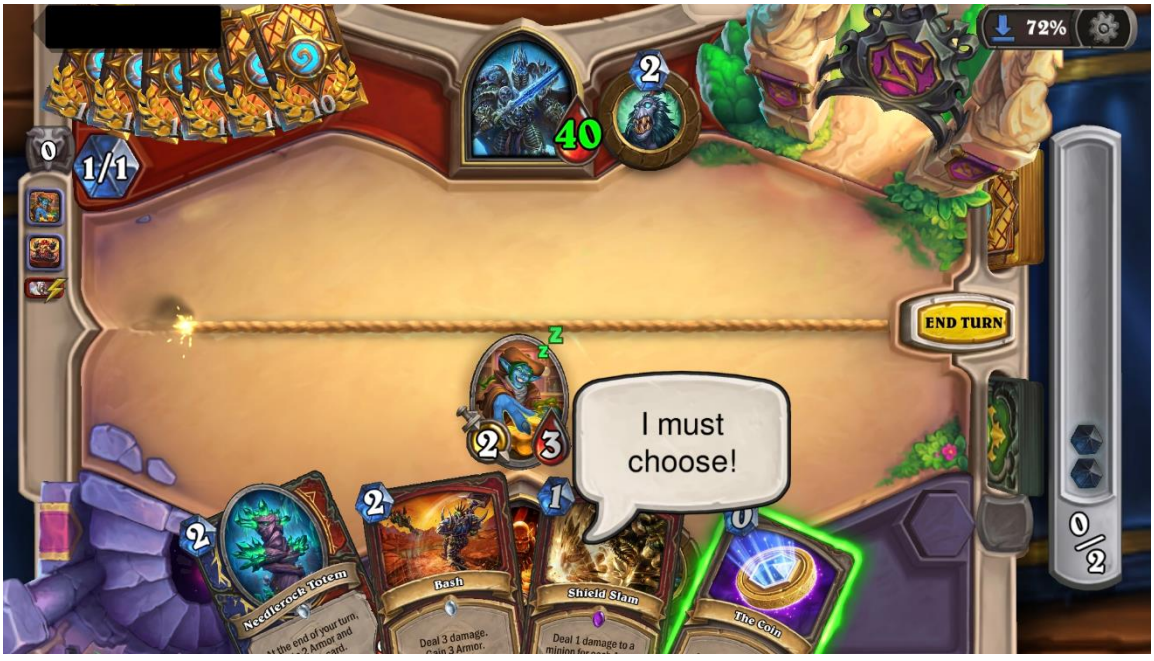
Nostetaan reaaliaikaisista peleistä esimerkiksi reaaliaikaiset strategiapelit, jolla saadaan verrattua niiden suunnitelmallista pelityyliä vuoropohjaisiin peleihin, joissa se on myös ominaista. Reaaliaikaisissa strategiapeleissä pelaajat tyypillisesti rakentavat, hallitsevat armeijoita ja muutoin edistyvät yhtä aikaa [29]. Reaaliaikaisille peleille on ominaista tietty nopeatempoisuus verrattuna vuoropohjaiseen peliin, myös strategiagenressä. Sellaisten pelien suunnittelu suosii yleensä yksinkertaisempia mekaniikoita, joita on pelaajien nopeampi käyttää ja reagoida jatkuvasti muuttuvaan pelin tilanteeseen. Esimerkki eräänlaisesta reaaliaikaisesta strategiapelistä mobiilialustalla on Clash Royale, jonka PvP-ottelusta on näkymä kuvassa 2.

Kuva 2. PvP-ottelu Clash Royalessa [30.]

Clash Royale sisältää elementtejä myös keräilykorttipeli- ja tornipuolustuspeligenreistä, eikä siinä ei ole täysin samanlaista resurssienkeräämistä, tukikohtien rakennuselementtiä ja armeijoiden vapaata liikuttelua kuin tyypillisemmissä reaaliaikaisissa strategiapeleissä. Yksiköt liikkuvat itsekseen ja pelin ainoa resurssi, eliksiiri, jolla yksikkö-, rakennus- ja loitsukortteja ”ostetaan” pelaajan korttipakasta ottelussa, generoidaan. Kuitenkin pelin voitto riippuu pitkälti pelaajan strategisoinnista eliksiirin kulutuksessa ja yksiköiden, rakennuksien ja loitsujen käytöstä ja sijoituksesta oikeaan aikaan ja oikeassa paikassa. Clash Royale -ottelut ovat synkronisia, kolmen minuutin mittaisia ja nopeatempoisia, missä nopea päätöksenteko on tärkeää. Niiden lyhytkestoisuus sopii mobiilipeleille tyypillisiin lyhyisiin sessioihin.

Kuitenkaan termit reaaliaikainen ja vuoropohjainen eivät ole monoliitteja, joihin sopiakseen pelien tulee olla juuri tietynlaisia. Monestikin pelin toiminnallisuus vastaa tietyn verran sekä reaaliaikaista että vuoropohjaista peliä, vaikka onkin ensisijaisesti jompikumpi. Joissakin vuoropohjaisissa peleissä on esimerkiksi vaihtoehtona ihmispelaajien pelata yhteisellä vuorolla, jossa kaikki saavat samanaikaisesti tehdä asiat, jotka heidän vuoroillensa kuuluvat [31]. Kun kaikki ovat valmiita lopettamaan vuoron, jatkuu pelin kierros tyypilliseen tapaan ja kukin tekoälypelaaja pelaa oman vuoronsa, kunnes palataan taas ihmispelaajien yhteiseen vuoroon. Tällaisessa vuoropohjaisessa pelin toiminnassa etuna on se, että ihmispelaajien ei tarvitse odottaa yhtä pitkään, jotta he saavat taas toimia, koska heidän ei tarvitse odottaa erikseen kaikkien muiden vuoroja.

Keräilykorttipeli Hearthstone taas on esimerkki vuoropohjaisesta pelistä, jossa on tietty reaaliaikainen elementti. Se onkin itse asiassa niin sanottu reaaliaikainen vuoropohjainen peli. Hearthstonessa reaaliaikaisuus näkyy siinä, että vaikka peli pohjautuu vuoroihin, joilla pelaajat vuorotellen pelaavat korttejaan, ei kuitenkaan pelaajalla ole loputtomasti aikaa pelata yhtä vuoroa. Vuorossa on aikaraja, jonka jälkeen se päättyy väkisin, jos pelaaja ei itse ole niin tehnyt. Kuvassa 3 näkyy käyttöliittymässä visuaalinen elementti, köysi keskellä pelikenttää, joka palaa, kunnes se on kokonaan kulunut. Sillä pelaajalle näytetään, että hänellä on aika loppumassa ja kuinka kauan sitä on jäljellä. Vuorojen pituuden rajauksella saadaan myös samalla rajattua otteluiden pituutta, ja mobiilipeleille lyhyemmät sessiot ovat tyypillisiä, joten otteluiden aikarajaus voi olla eduksi.



Kuva 3. Vuoro, jossa aikaraja on loppumassa Hearthstonessa [32].

Otetaan esille vielä kaksi tapaa muokata pelin kulkua. Toimintopisteet ovat yksi mahdollinen pelien ajoituksen elementti. Esimerkki toimintopisteistä on XCOM-sarjan peleissä, jotka ovat vuoropohjaisia ja missä pelaaja hallitsee useaa hahmoa kerrallaan. Kullakin hahmolla on rajallinen määrä toimintopisteitä, millä esimerkiksi ampua, ladata ase tai ottaa jokin esine kantoon, joka rajaa tehtävien toimintojen määrää ja siten lisää enemmän suunnitelmallisuutta pelin pelaamiseen. [33]. Toimintopisteet voivat siis olla pelin kulun pohja tai niillä voidaan tuoda peliin uusi taktinen ulottuvuus, ja lisäksi ne voivat toimia pelin tasapainotusta helpottavana järjestelmänä, koska niillä voidaan rajata sitä, mitä vaikkapa yhden vuoropohjaisen pelin vuoron aikana voidaan tehdä, tai kuinka nopeasti pelaaja voi käyttää toimintoja reaaliaikaisessa pelissä, selkeästi ja loogisesti.

Aikaisemminkin maintussa Clash Royalessa sen eliksiiriresurssi on eräänlainen toimintopistejärjestelmä, johon pelin kulku pohjautuu, koska kaikki pelaajan toiminnot pelissä maksavat sitä. Reaaliaikaisesti ajan kuluessa generoidaan lisää eliksiiriä, ja pelaaja kuluttaa sitä pelatessaan. Kuvassa 4 näkyy eliksiirimittari ja sillä hetkellä pelattavissa olevat kortit, joilla kullakin on oma eliksiirihintansa.

Kuva 4. Eliksiirimittari ja käytettävissä olevat kortit sekä niiden hinnat Clash Royalessa [34].

Samoin myös aikaisemmin mainitussa vuoropohjaisessa Hearthstonessa korttien pelaaminen maksaa ”manaa”. Manaa on ensimmäisellä vuorolla käytettävissä yksi, ja käytettävän manan raja kasvaa lopulta kymmeneen vuorojen kuluessa, ellei pelaaja joitain kortteja pelaamalla ylitä vuoron oletusarvoista manarajaa. Mana täytetään aina vuoron alkaessa sen hetkiseen rajaan, ellei jokin kortti muokkaa manan käytöstä. Tästä havaitaan se, että toimintopisteitä voidaan siis käyttää sekä reaaliaikaisessa että vuoropohjaisessa pelissä.

Toinen vaihtoehtoinen toimintaperiaate moninpeliin, tai lisäys siihen, on vaiheeseen perustuva ajoitus. Sillä voidaan muuttaa pelin intensiivisyyttä pelin vaiheeseen perustuen ja muokata pelin tasapainotusta myös tiettyyn aikaan tai pelin tilanteeseen, joka määrittää vaiheen, liittyen. Esimerkki pelistä, jossa vaiheeseen perustuvaa ajoitus on keskeistä, ovat toiseen maailmansotaan

sijoittuvat Steel Division -sarjan reaaliaikaiset strategiapelit. Niissä yksi ottelu on taistelu, jossa kukin pelaaja hallitsee omaa ”divisioonaansa”, mikä määrittää taisteluun valittavissa olevat yksiköt, kuten lentokoneet, jalkaväkiryhmät ja panssarivaunut. [35.]

Taistelussa pelaaja tuottaa passiivisesti resurssia, jolla hän voi ostaa yksikköjä kentälle käyttämänsä divisioonan reserveistä. Käytössä olevan divisioonan lisäksi myös pelin vaihe, joita on kolme, A, B ja C, määrittää, minkä tyyppisiä yksiköitä pelaaja voi kutsua kentälle, ja kuinka monta niitä on tarjolla käyttöönotettavaksi. Vaihe etenee yhtä aikaa kummallakin pelaajalla, alkaen A-vaiheesta, ja päätyy lopulta C-vaiheeseen, jolloin kaikki yksiköt ovat saatavilla. Samoin yksiköiden ostamiseen käytetty resurssi generoituu nopeammin, kun pelin vaihe etenee. Tällä saadaan aikaiseksi haluttua tahtia eskaloituva taistelu, joka kasvaa vaiheiden myötä intensiivisemmäksi, kun voimakkaammat yksiköt tulevat käytettäviksi. Vaiheeseen perustuva järjestelmä voi myös auttaa pelin tasapainottamisessa, kun voimakkaammat yksiköt tulevat vasta esimerkiksi tietyn ajan kuluttua käytettäväksi.

2.2 Pelaajien määrä ja vuorovaikutus toisiinsa ja pelimaailmaan

Pelaajien määrä sessiossa on myös oleellinen ominaisuus moninpelissä. Eri peligenreissä on tyypillistä hyvinkin toisistaan eriävät määrät pelaajia, kuin myös pelaajien jakautuminen pelisession sisäisesti. Yleisesti ottaen pelaajat voivat vain olla olemassa pelisessiossa yksilöinä, tai kuulua ryhmiin tai joukkueisiin. Riippuen pelistä, niin pelaajamäärällä kuin mahdollisella ryhmäjakautumisella on jonkin verran toleranssia myös muutoksille pelin aikana. [36.]

Esimerkiksi MMORPG-genren pelit yleensä kykenevät sisältämään satojakin pelaajia yhdessä sessiossa. Senkaltaiset pelit eivät tyypillisesti välitä siitä, kuinka monta pelaajaa itse pelimaailmassa on, sillä kuten aikaisemmin mainittiin, niiden pysyvät maailmat ovat olemassa silloinkin, kun kukaan ei ole pelissä. Pelaajat voivat siis tulla peliin ja poistua siitä vapaasti, niin kauan kuin palvelin on päällä. Pelaajat ovat yleensä oletusarvoisesti yksin, mutta voivat tyypillisesti muodostaa myös pieniä ryhmiä, missä seikkailla yhdessä.

Joidenkin muiden genrejen pelit voivat myös sisältää suhteellisen suuren määrän pelaajia samassa sessiossa. Esimerkiksi aikaisemmin mainitun selviytymisgenren jotkin pelit sekä Battle Royale-genren pelit, voivat sisältää useita kymmeniä, tai satakin, pelaajaa kerralla. Battle Royale-peleissä yleensä pyritään täyttämään sessio ennen kuin itse ottelu alkaa, kun taas suuremmissa

selviytymispelisessioissa pelaajamäärä voi toimia samankaltaisesti kuin MMORPG-peleissä, niin että pelaajat voivat tulla ja mennä kuten haluavat.

Joissain peleissä taas moninpeli on paljon pienemmissä sessioissa. Teoreettisessa kilpailullisessa räiskintäpelissä, ottelussa voi esimerkiksi olla vain kymmenen pelaajaa, jotka on jaettu kahteen viiden pelaajan tiimiin. Jossain moninpelattavissa roolipeleissä taas voisi olla tilaa esimerkiksi vain neljälle pelaajalle yhtä aikaa. Myös on mainittavaa, että samassa pelissä voi olla eri määrä pelaajia eri pelimuodoissa. Jokin kilpailullinen räiskintäpeli voi sisältää myös Battle Royale-pelimuodon, tai MMORPG-peli pienemmän tyrmäseikkailun, joissa pelaajamäärät eroavat tavallisesta pelimuodosta. Kaiken kaikkiaan on vaikea määritellä, kuinka monta pelaajaa pelisessioon sallitaan, kun puhutaan puhtaasti konseptista tai genrestä, koska se riippuu pitkälti pelin tarkoituksista.

Myös pelaajien vaikuttaminen toisiinsa ja pelimaailmaan on tärkeä moninpelin komponentti. Pelaaja vastaan pelaaja (PvP) -peli on sellainen, missä ensisijaista on pelaajien välinen kilpailullinen vuorovaikutus toisiinsa [37]. Niissä on erityisen tärkeää saavuttaa tasapaino, jossa pelaajien välinen kilpailu on reilua. Pelaaja vastaan ympäristö (PvE) -pelit taas keskittyvät pelikokemukseen, jossa keskeistä on pelaajien vaikutus maailmaan [38]. Esimerkiksi moninpelattavat roolipelit ovat tyypillisesti PvE-pelejä. Koska niissä pelaajien vastustajana on pelimaailma ja NPC-hahmot, voivat niissä pelaajahahmot usein olla paljon voimakkaampia kuin PvP-peleissä.

Monissa peleissä voi olla sekä PvP- ja PvE-elementtejä, mutta sellainen peli, missä kumpikin on keskeistä, on nimeltään PvPvE, eli pelaaja vastaan pelaaja vastaan maailma. Esimerkiksi Hunt: Showdown on tällainen. Siinä pelaajat menevät lyhytkestoiseen seikkailuun joko yksin tai pienissä tiimeissä. Seikkailussa on muiden pelaajatiimien lisäksi NPC-vihollisia, jotka voivat olla pelaajalle yhtä vaarallisia kuin toiset pelaajatkin, mutta lisäksi peli rohkaisee pelaajien välistä kilpailua tehtävän suorittamiseksi. Tällaisessa pelissä voi olla erityisen haastavaa saada aikaiseksi tasapainoinen pelikokemus, koska pelaajien odotetaan taistelevan sekä toisiaan että vaarallisia tekoälyvihollisia vastaan. [39.]

3 Sosiaaliset ominaisuudet

Kaverilista on sosiaalinen ominaisuus, joka toimii erinomaisena pohjana monille eri sosiaalisille ominaisuuksille videopeleissä. Perusmuoto kaverilistaominaisuudesta sisältää kyvyn lähettää ja hyväksyä kaverikutsuja ja hallinnoida listaa kavereista muun muassa poistamalla kavereita. Yleisiä lisäominaisuuksia ovat muun muassa kyky blokata pelaajia, joiden kanssa ei halua olla tekemisissä, ja kyky ”seurata” muita pelaajia, jolloin heidän aktiviteettinsa voivat saada prioriteetin mahdollisissa ilmoituksissa tai yhteistyöhön pohjautuvissa pelin elementeissä. [40.]

Joissain videopeliekosysteemeissä, kuten Microsoftin Xbox Live -palvelussa, pelaajilla voi olla profiili, mikä voi sisältää yksityistä tietoa, kuten pelaajan oikean nimen. Tällaisissa tapauksissa voi olla järkevää olla olemassa järjestelmä, jossa pelaaja voi määrittää kyseisen tiedon näkyvyyden. Yksityisen tiedon tarkkailu voitaisiin sallia vain kavereille, tai kaverilistassakin voi olla prioriteettijärjestelmä, jossa vain erityisen läheiset kaverit saavat nähdä yksityistä tietoa toisesta pelaajasta. Prioriteettijärjestelmä voi olla muutenkin hyödyllinen, lisäys, jolla pelaajat voivat määrittää, ketkä pelaajat tulevat ensimmäisenä tarjolle, kun kaverilistaa tai jotain sitä hyödyntävää mekaniikkaa, kuten peliin kutsumista, käytetään. [41.]

Videopelien kontekstissa kilta, klaani tai vastaava on organisoitu ryhmä pelaajia [42]. Ne ovat eriytyisen yleisiä etenkin massiivisissa moninpeleissä, kuten Guild Wars 2 tai The Elder Scrolls Online. Riippuen pelistä ja killan käyttötarkoituksesta, pelaajien kuulumista kiltoihin voidaan rajata eri tavoin. Jos erityisen tärkeää pelissä on PvP tai muu kilpailullisuus, kiltoihin kuuluminen voi olla rajattu esimerkiksi niin, että koko tili kuuluu kiltaan, tai jos pelaajalla voi olla monta hahmoa, niin kukin niistä voi kuulua erikseen johonkin kiltaan. Joissain peleissä voidaan sallia myös yhden hahmon kuulua useaan kiltaan yhtä aikaa, jos peli esimerkiksi keskittyy enemmän PvE-sisältöön tai muuten yhteistyöhön, kuten The Elder Scrolls Onlinessa.

Vaikka killat ovat erityisen yleisiä massiivisissa moninpeleissä, voi sellaisia järjestelmiä olla monen muunkin tyyppisissä peleissä, kuten selviytymispeleissä tai hahmojen keräilyä sisältävässä vuoropohjaisessa roolipelissä pelissä Raid: Shadow Legendsissä, jonka yhdestä satunnaisesti valitusta pelaajien klaanin julkisivusta on otettu kuva 5. Se on esimerkki pelistä, jossa yksi pelaajan tili voi kuulua yhteen kiltaan kerrallaan.

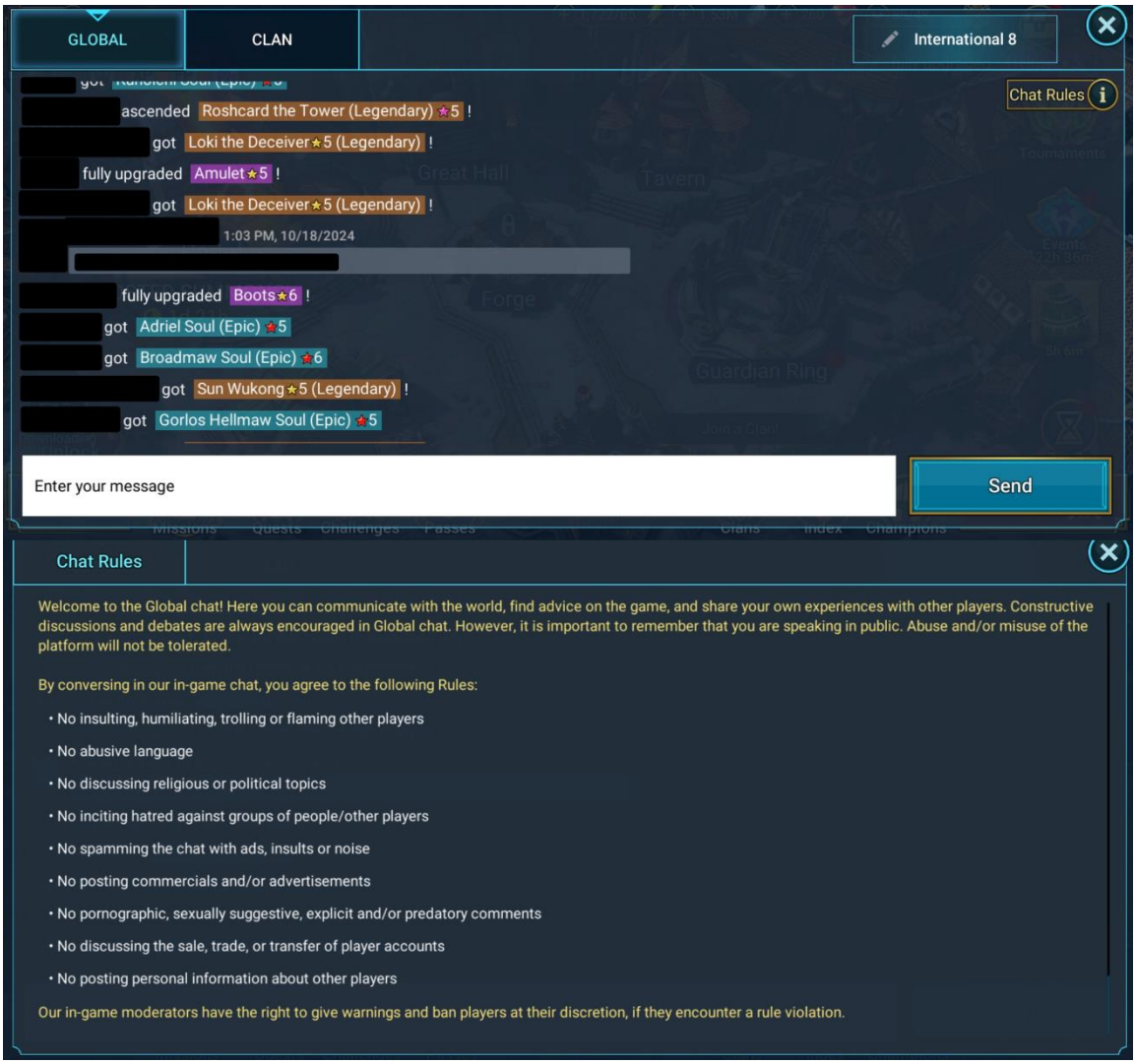
Kuva 5. Klaanin julkinen profiili Raid: Shadow Legendsissa [43].

Killoilla on monia käyttötarkoituksia. Niille voidaan luoda esimerkiksi tyyppijärjestelmä, jossa on esimerkiksi haastavan sisällön suorituksiin keskittyvä kilta, kaupankäyntikilta, uusien pelaajien kilta ja niin edelleen. Pelin tarpeisiin sopiessa esimerkiksi juuri uusien pelaajien kiltaan kuuluminen voisi tarjota nopeampaa hahmon tasojen saavuttamista. Kuitenkaan tällainen ei aina ole tarpeen, vaan pelaajat muodostavat omiin tarkoituksiinsa sopivia ryhmiä itsenäisestikin. Kuvaan 5 valikoitu klaani kertoo kuvauksessaan, että se on vain tiettyyn sisältöön osallistuville pelaajille luotu. Kuvassa nähdään myös vaatimuksia, joita kiltaan liittymiselle on asetettu, esimerkiksi minimivoimakkuustaso pelaajalla.

Esimerkki konkreettisesta hyödystä kiltaan kuulumisessa on The Elder Scrolls Onlinessa, jossa on ominaisuus, jossa kilta voi vuokrata jostain pelin keskusmaailmasta tai kaupungista oman NPC-kauppiaansa, joka toimii sitten killan ja sen ulkopuolisten pelaajien käyttöliittymänä pelaajien väliseen kaupankäyntiin. Killan pelaajat voivat laittaa tavaraa myyntiin, jolloin ulkopuoliset pelaajat voivat käydä ostamassa killan pelaajien myyntiin laittamia esineitä kyseiseltä NPC-kauppiaalta. Muita hyötyjä killoista voivat olla esimerkiksi jokin keskinäinen tukikohta, jossa pelin oleelliset palvelut ovat tarjolla helposti, killoille eksklusiivisia palveluita tai etuja, tai jotkin haastavat tehtävät voivat olla rajattuja niin, että vain kilta voi avata sellaisen pelattavaksi. [44.]

Pelinsisäiset chat-ominaisuudet sallivat pelaajien keskustelun pelin aikana. Se on erittäin yleinen sosiaalinen ominaisuus videopeleissä. Chat-ominaisuus voidaan jakaa kolmeen tyyppiin, joista ensimmäinen ja yleisin, tekstichat, on ollut olemassa 1970-luvun loppupään ensimmäisien moninpelejen ajoista lähtien [45]. Se on edelleen monissa peleissä ensisijainen tai ainoa chat-vaihto-

ehto. Verraten muihin se on suhteellisen helppo implementoida toimivassa ja turvallisessa muodossa. Teksichatissa voidaan usein myös lähettää yksityisviestejä pelaajien kesken tai klaanin sisäisesti, ja joissain peleissä, kuten kuvan 6 Raid: Shadow Legendsin julkisessa tekstichatissa, tulee myös ilmoituksia asioista, kuten siitä, kun toiset pelaajat saavat harvinaisia esineitä. Joidenkin pelaajien perspektiivistä teksti voi olla vähemmän miellyttävä vaihtoehto, ja joissain peleissä tekstin kirjoittamisen ja lukemisen vaatima huomio voi olla haitaksi pelikokemukselle. Aina se ei kuitenkaan ole myöskään negatiivinen, vaan pelaajia on erilaisia. Jotkut saattavat suosia hiljaisempaa ja vähemmän henkilökohtaista kommunikaatiota. Jos pelin muut elementit tukevat sitä, voi olla eduksi tarjota useampi kuin vain yksi chat-ominaisuus, jotta katetaan eri pelaajien tarpeet.



Kuva 6. Raid: Shadow Legendsin julkinen tekstichat ja sen säännöt [46].

Äänichat on nykyään monissa peleissä sisäänrakennettu ominaisuus. Se tarjoaa voimakkaamman sosiaalisen kokemuksen, koska ihmiset puhuvat omine äänineen, eivätkä vain tekstin kautta. Lisäksi äänichat sallii nopeammat reaktiot ja tiedon jaon pelin sisäisistä tapahtumista, joka voi auttaa pelaajia strategisoimaan pelin aikana. Esimerkiksi räiskintäpeleissä se voi tarjota tärkeän kyvyn kommunikoida tiimin sisäisesti. Kehittäjän kannalta sen moderoiminen ja muutoin turvallisesti implementointi voi kuitenkin olla haastavampaa.

Videochatin voidaan kuvitella olevan samankaltainen kuin äänichat, mutta lisätyin ulottuvuuksin sekä hyödyissä että haitoissa, mitä tulee esiin, kun äänen lisäksi myös videokuvaa voidaan jakaa. Se on todella harvinainen ominaisuus peleissä, ja yleensä sitä varten pelaajat käyttävätkin jotain kolmannen osapuolen sovellusta. Kuitenkin sekin voi olla hyödyksi peleissä, joissa sosiaalinen elementti on erityisen tärkeää. Esimerkiksi pokerin videopeliadaptaatiossa se voi olla hyödyllinen, sillä pokeri on korttipeli, jossa toisen ihmisen ”lukeminen” voi olla yhtä tärkeää kuin omat kortit ja niiden käyttö oikein.

Moderointi taas on oleellinen osa miellyttävää ja turvallista pelikokemusta. Mahdolliset loukkaavat sanat tai kiusaaminen pitäisi pystyä estämään tai ainakin tarjota vaihtoehto epämukavien asioiden pois filtteröintiin. Ensimmäinen askel on tehdä selkeät säännöt, jotka kieltävät mahdollisesti haitallisen käytöksen chatissa, kuten kuvasta 6 löytyvät Raid: Shadow Legendsin tekstichatin säännöt. Pelin ylläpidon puolella on kannattavaa olla henkilöstöä tai ainakin botti, joka antaa chat-kiellon henkilölle, joka käyttää haitallista kieltä. Myös pelaajalle itselleen on viisasta tarjota vaihtoehto muiden pelaajien hiljentämiseen, jottei heidän ole pakko keskustella epämukavien yksilöiden kanssa. Tekstichatissa tämä on jokseenkin helpompaa, kun taas ääneen pohjautuvaa keskustelua on vaikeampi moderoinnin perspektiivistä tarkkailla. Lisäksi on olemassa myös kysymys yksityisyydensuojasta, joka saattaa vaatia myös tarkempaa lain tuntemusta, jos puhetta tai videokuvaa on tarpeen moderoida.

Pistetaulukot ja muut pistejärjestelmät ovat suhteellisen yksinkertainen sosiaalinen ominaisuus, jolla voidaan ajaa pelaajien pysyvyyttä ja mielenkiintoa peliä kohtaan. Ne ovat yksi vanhimmista sosiaalisista ominaisuuksista peleissä, ja ovat yksi yleisimmistä sosiaalisista ominaisuuksista peleissä nykyaikaankin [47]. Yleensä puhuen, ne toimivat pelaajan tai killan edistystä ja onnistumista esittävänä elementtinä pelissä, joten ne voivat olla hyvinkin erilaisia pelin mukaan. Kuvassa 7 on näkymä Clash of Clansin pistetaulukkokäyttöliittymässä, jossa on valittuna taulukko, joka esittää eniten pisteitä saaneita pelaajia. Taulukossa on kaikkein näkyvimpänä pelaajien pistemäärät, mutta koska pelille on ominaista sen PvP-ominaisuus, näytetään siinä myös pelaajien voitetut hyökkäykset ja puolustukset. Pistetaulukot voivat keskittyä myös PvE-sisältöön.

Kuva 7. Pelinsisäinen pistetaulukkojärjestelmä Clash of Clansissa [48].

Vuonna 2005 videopelien sisäisten saavutuksien todellisena uranuurtajana Microsoft julkaisi Xbox 360:n Gamerscore-järjestelmän, joka toi saavutukset yhteen koko alustalla. Siinä missä ennen ne olivat kuin omia saarekkeitaan, Xbox-saavutuksista ansaittiin koko tilille pisteitä ja eri saavutukset tulivat näkyviin muillekin pelaajille, myös pelin ulkopuolella. Kyseinen järjestelmä sai suurta suosiota, ja pelaajat nauttivat mahdollisuudesta kilpailla keskenään saavutuksien ansaitsemisessa. Microsoft taas nautti järjestelmän tuottamasta huomiosta ja aktiviteetista, mikä oli omiaan kasvattamaan pelaajien pysyvyyttä ja jatkuvaa osallistumista alustalla. 2010-luvun aikana muun muassa Valven Steam ja Sonyn PlayStation 3 myös julkaisivat omat samankaltaiset saavutusjärjestelmänsä. Siitä saavutukset ovat vain yleistyneet peliteollisuudessa. [49.]

Saavutuksia on erilaisia. Jotkin saavutukset perustuvat pelaajan edistykseen pelissä, esimerkiksi tarinan vaiheisiin. Toiset perustuvat toistoon, kuten tiettyjen NPC-hahmojen päihittämiseen, kun taas toiset voivat olla jonkin salaisuuden löytämiseen tai jonkin haasteen suorittamiseen liittyviä, kuten pelin suoritus kovimmalla vaikeustasolla. Saavutuksien ympärillä on myös olemassa saavutustenmetsästyskulttuuri. Jotkut pelaajat saattavat pelata pelejä nimenomaan kaikkien pelistä löytyvien saavutuksien hankkimisekseen, kun taas toiset voivat vain haluta näyttää tehneensä kaiken suosikkipelissään. Saavutuksien metsästyksestä voi olla jotain hyötyä pelaajalle, sillä niiden hankkimisesta voidaan tarjota esimerkiksi pelinsisäisiä esineitä, tai kuten aikaisemmin mainittiin,

voisi olla olemassa yleinen pistejärjestelmä sosiaalisessa profiilissa. Myös killoilla voi olla saavutuksia, kuten Raid: Shadow Legendsin klaanin julkisen profiilin sisältävässä kuvassa 5, jossa näkyi myös joitain klaanin saavutuksia näytillä. [50.]

Pelinsisäiset tapahtumat ovat monimuotoinen ominaisuus peleissä. Niillä voidaan ajaa pelaajien pysyvyyttä ja palautuvuutta tarjoamalla mielenokiintoisia pelikokemuksia ja harvinaisia tai eksklusiivisia palkintoja. Ajallisesti niitä on kahdenlaisia. Toistuvat tapahtumat ovat sellaisia, joiden suunnitellaan tulevan takaisin niiden ensimmäisen tapahtumakerran jälkeen, kun taas kertaluontoiset tapahtumat ovat yksittäistapauksia, joiden ei suunnitella tapahtuvan uudelleen, ainakaan kovin pian [51]. Jotkin tapahtumat sijoittuvat puhtaasti pelimaailmaan, esimerkiksi sen tarinaa ajavana elementtinä tai jotain muuta peliin liittyvää asiaa juhlistaen, joten ne voivat olla kumpaa vain tyyppiä riippuen niiden käyttötarkoituksesta. Kausiluonteiset tapahtumat taas perustuvat tosielämään, esimerkiksi vuodenaikaa tai juhlapäivää, kuten joulua juhlistava tapahtuma, josta niiden voidaan myös päätellä olevan toistuvia. Yhteistyötapahtumat sen sijaan ovat esimerkiksi toisen pelin tai jonkin brändin kanssa tehtyä markkinointiyhteistyötä, ja ne ovat yleisemmin kertaluontoisia. Kuvassa 8 näkyy pelin Clash of Clans senhetkiset tapahtumat, joista toinen on yleinen kiltojen välinen kilpailutapahtuma ja toinen on haastetehtävän sisältämä tapahtuma omine palkintoineen.

Kuva 8. Clash of Clansin tapahtumataulukko [52].

Sosiaalisen median integraatio peliin sallii pelaajien yhdistävän pelinsisäisen tilinsä sosiaalisen median tiliinsä, esimerkiksi Facebookiin. Sitä voidaan käyttää verkostoitumiseen ja sosiaaliseen kanssakäymiseen sekä rakentaa yhteisöllisyyttä. Pelaaja voi esimerkiksi löytää sosiaalisen median kaverinsa pelissä helpommin tai jakaa omaa pelissä edistymistään sosiaalisen median sivulle, joka samalla nostaa myös pelin näkyvyyttä. Sosiaaliseen mediaan yhdistäminen ei yleensä ole pakollista, mutta pelaajaa voidaan rohkaista tekemään se esimerkiksi tarjoamalla jokin pelinsisäinen palkkio siitä. [53.]

Avunpyyntö ja -anto-ominaisuus ja muu lahjojen lähettäminen rohkaisevat positiiviseen pelaajien väliseen kanssakäymiseen. Apua voidaan tarjota esimerkiksi pelinsisäisessä valuutassa, resursseissa tai vaikka sotilain, jos pelissä on armeijanrakennuselementti, kuten esimerkiksi Clash of Clansissa, missä pelaaja voi pyytää kiltansa jäseniltä sotilaita. Lahjanantojärjestelmä toimii samalla tavalla niin, että yksi pelaaja antaa toiselle jotain, mutta siinä ei ole erikseen mitään pyyntömekaniikkaa. Tällaiset toiminnot rohkaisevat pelaajia positiiviseen vuorovaikutukseen toistensa kanssa ja koordinaatioon peleissä.

4 Ominaisuuksien ohjelmointi

Tässä osiossa toteutetaan joitain edellä mainituista sosiaalisista- ja moninpeliominaisuuksista Unity-pelimoottorissa. Unity on suosittu pelimoottori, joka julkaistiin vuonna 2005 tavoitteenaan olla lähestyttävä ratkaisu pelien kehitykseen pyrkiville henkilöille, ja on ollut siitä lähtien jatkuvassa kehityksessä [54]. Ohjelmointi tapahtuu C#-ohjelmointikielellä. Moninpeliominaisuuksien ohjelmoinnissa käytetään hyödyksi moninpelimoottoria Photon Unity Networking 2 (PUN 2), ja sosiaalisten ominaisuuksien toteutukseen otetaan käyttöön Google Play Games plugin for Unity. Näitä olemassa olevia palveluita käyttäen saadaan puitteet keskittyä juuri sosiaalisten ulottuvuuksien ja moninpeliominaisuuksien ohjelmoimisen tarkasteluun, ilman haastavampaa pohjustustyötä. Siksi ne tarjoavat erinomaisen mahdollisuuden lähestyä kyseisten ominaisuuksien ohjelmointia.

PUN 2 on suosittu moninpeliratkaisu Unity-pelimoottorille. Se käyttää ns. Client-Server-arkkitehtuuria, jossa pelaajat yhdistävät pilvipalvelimelle, joka on keskus pelisessioiden hallinnoimiseen, pelin synkronointiin ja muuhun moninpelin ylläpitoon. Pelaajien laitteet lähettävät palvelimelle päivityksiä pelin tilasta, jonka kautta paikalliset muutokset tulevat näkyville myös muille pelaajille pelisessiossa eli ”huoneessa”. PUN 2 perustuu näihin huoneisiin, jotka sisältävät yhden pelisession, jonka pelaajat kanssakäyvät toistensa kanssa. Huoneessa on yksi ”isäntä”, jolla on oikeudet hallinnoida huonetta enemmän kuin muilla, ja loput ovat sen jäseniä. PUN 2 ei tue asynkronisia pelejä, vaan soveltuu monenlaisiin synkronisiin peleihin. Se soveltuu monenlaisiin peligenreihin, joissa on jokin rajallinen pelisession koko. Esimerkki peleistä, joihin PUN 2:a ei kannata käyttää, ovat MMORPG-genren pelit, sillä vaikka se tukee kohtalaisen kokoisia pelisessioita, se ei ole suunniteltu mahdollisesti satojen tai tuhansien pelaajien kokoisien sessioiden toteutukseen. [55].

Google Play Games plugin for Unity taas on avoimen lähdekoodin projekti, joka tarjoaa Unity-pelimoottoriin sopivan lisäosan, jolla voidaan tuoda Google Play Games -palvelun toiminnallisuus peliin. Se tukee Google Play Games -sisäänkirjautumista, kaveri-, saavutus-, pistetaulukko- ja tapahtumaominaisuuksia, pilvitallennusta ja läheltä löytyviin laitteisiin yhdistämistä. Sitä voidaan käyttää suoraan Unitysta löytyvän Social-rajapinnan kautta, mutta sisältää myös oman toiminnallisuuden, jos sitä halutaan käyttää muuhun tarkoitukseen. [56].

Moninpeliominaisuuksien ohjelmointiosuudessa käydään ensin lyhyesti läpi, mitä täytyy tehdä lisäosan asentamiseksi ja käyttöönottamiseksi. Sen jälkeen yhdistetään pelaajat palvelimelle, jonka kautta heidät voi sitten jakaa pienempiin pelisessioihin PUN 2:n matchmaking-järjestelmää

käyttäen. Matchmakingiin liittyen esitellään lisäksi, miten huonetta voidaan konfiguroida rajaamalla pelaajamäärää tai tekemällä siitä yksityinen. Kun pelaajat voidaan yhdistää toisiinsa, tutkitaan pelitapahtumien synkronointia ottamalla tarkkailuun, miten pelaajien sijainti ja tilanne voidaan saada näkyviin muillekin pelaajille, ja laajennetaan siinä tutkittavaa sisältöä tekemällä esimerkki aikaisemmin keskustellusta reaaliaikaisesta vuoropohjaisesta pelin tahdista, jossa pelaajilla on omat ajallisesti rajatut vuoronsa. Lopuksi pohditaan vielä, miten yhteydessä ilmeneviin ongelmiin, kuten sen katkeaminen kokonaan, voidaan reagoida.

Moninpeliosuutta seuraa sosiaalisten ominaisuuksien ohjelmointiosuus. Google Play Games plugin for Unityn käyttö vaatii myös Google Developer Consolen puolella työskentelyä sovelluksen ja sosiaalisten ominaisuuksien konfiguroinniksi. Tässä työssä sitä ei kuitenkaan käsitellä, vaan oletetaan, että pohjustava työ ennen itse ohjelmointia on saatu suoritettua. Ensimmäinen osuus on myös asennus ja käyttöönotto, jota seuraa pelaajan sisäänkirjautuminen Google Play Games -palveluun. Sitten käydään läpi Google Play Gamesin tarjoamien sosiaalisten ominaisuuksien ohjelmointia lisäosaa käyttäen ja myös Unityn Social-rajapinnan kautta.

4.1 PUN 2:n asennus ja käyttöönotto

Moni Photon Enginen tarjoamista lisäosista Unity-pelimoottoriin löytyy suoraan Unity Asset Storesta, josta voi ostaa maksullisia tai lisätä ilmaisia lisäosia ja muuta lisäsisältöä Unityssa käytettäväksi. Tässä työssä käytössä on PUN 2:n ilmaisversio, jonka voi lisätä käyttäjätililleen maksutta. Se on kehitystarkoitukseen tehty PUN 2 -versio, joka tarjoaa tarvittavat puitteet moninpeliominaisuuksien ohjelmoinnin esittelyyn tässä työssä.

Jotta lisäosan voi ottaa käyttöön, tulee myös tehdä Photon Engine-käyttäjätili ja konfiguroida palvelun verkkosivun Dashboard-osiosta ”app”, sovellus, jonka kautta yhtä Photon-palveluita käyttävää sovellusta hallinnoidaan. Jokaisella sovelluksella on oma App ID, jota tarvitaan Unity-lisäosan käytössä. Kun PUN 2 lisäosa on hankittu Unity Asset Storesta, voidaan se lisätä projektiin Package Manager -ikkunasta Unity-projektin ollessa avoinna ja sitten ottaa käyttöön avaamalla lisäosan Unityyn lisäämä PUN Wizard, johon App ID liitetään.

4.2 Pelaajien yhdistäminen Photon-palvelimelle

Pelaajan yhdistäminen Photon-palvelimelle PUN 2:ssa tapahtuu funktiolla ConnectUsingSettings, joka on osa PhotonNetwork-luokkaa. Kuvan 9 esimerkkikoodissa se on sisällytetty sille tehtyyn omaan funktioonsa, joka kutsutaan Start-funktiosta, joka kutsutaan automaattisesti, kun skene, jossa kyseinen koodi on olemassa, aukeaa. Esimerkkikoodissa on mainittavaa myös string-tyypin muuttuja gameVersion, joka voi olla tärkeä muistaa asettaa, koska sillä Photon-palvelin osaa erotella eri peliversiot toisistaan. Tällä voidaan estää esimerkiksi suuria muutoksia tehdessä kahden eri version pelaajien liittyminen toisiinsa.

```
using UnityEngine;
using System.Collections.Generic;
using Photon.Pun;
using Photon.Realtime;

Unity Script (1 asset reference) | 0 references
public class PhotonConnection : MonoBehaviour, IConnectionCallbacks
{
    // When scene opens, Start sets game version and calls a function attempts to make a connection to Photon the master server.
    // Made it a separate function, so that it could be easily reused elsewhere in a theoretical game.
    Unity Message | 0 references
    void Start()
    {
        PhotonNetwork.GameVersion = "1";
        ConnectToPhoton();
    }
    2 references
    public void ConnectToPhoton() { PhotonNetwork.ConnectUsingSettings(); }

    // If a connection is succesfully made AND the local client is ready to communicate with Master server (so it can matchmake etc).
    2 references
    public void OnConnectedToMaster() { Debug.Log("Connection to Photon has been established."); }
    // If a connection is lost or fails to be made in the first place. Right now it just calls the function that attemps to connect, again.
    6 references
    public void OnDisconnected(DisconnectCause cause) { ConnectToPhoton(); }

    // These ones are not in use, but are part of the IConnectionCallbacks.
    2 references
    public void OnConnected() { Debug.Log("OnConnected"); }
    2 references
    public void OnRegionListReceived(RegionHandler regionHandler) { Debug.Log("OnRegionListReceived"); }
    2 references
    public void OnCustomAuthenticationResponse(Dictionary<string, object> data) { Debug.Log("OnCustomAuthenticationResponse"); }
    3 references
    public void OnCustomAuthenticationFailed(string debugMessage) { Debug.Log("OnCustomAuthenticationFailed"); }
}
```

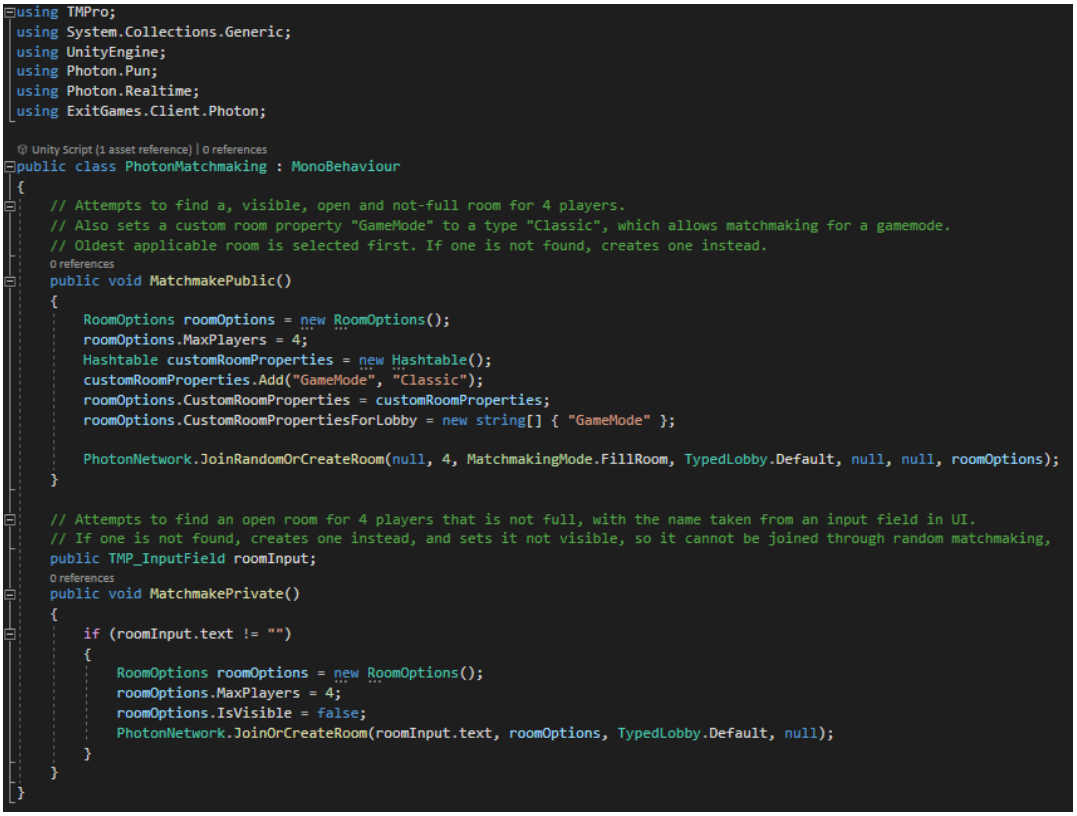
Kuva 9. Pelaajan yhdistäminen palvelimelle ja eri callback-funktioita yhteydelle. [57].

Callback-funktio OnConnectedToMaster kutsutaan automaattisesti, jos onnistutaan muodostamaan yhteys keskeiselle palvelimelle ja ollaan valmiita muuhun PUN 2 -toiminnalisuuteen. Tässä tapauksessa se vain kirjoittaa Unityn debuggauskonsoliin viestin, joka ilmoittaa onnistumisesta. OnDisconnected taas kattaa laajan kirjon eri syitä, miksi pelaajalla katkeaa yhteys Photon-palvelimelle. Myös se, että pelaajalla kestää liian kauan saada vastaus palvelimelta ja yhdistettyä siihen, on yksi syy, miksi kyseinen funktio voi tulla automaattisesti kutsutuksi. Esimerkkikoodissa sekin kutsuu funktion, jossa aloitetaan palvelimelle yhdistämisprosessi, eli toisin sanoen, jos yhteyden muodostus epäonnistuu, tässä tilanteessa sitä yritetään automaattisesti uudelleen.

Kokonaisen pelin toiminnallisuuteen koodia voidaan laajentaa pelin tarpeiden mukaisesti. Jatkokehityksessä OnDisconnected voitaisiin vaihtoehtoisesti laittaa tuomaan esimerkiksi esiin napin, josta yhteyden muodostuksen funktio kutsutaan, jos halutaan, että se tehdään vain manuaalisesti. OnConnectedToMaster voisi esimerkiksi siirtyä johonkin skeneen tai tuoda esiin pelin käyttöliittymän osion, joka sisältää seuraavan osuuden koodin eli pelaajien yhdistämisen toisiinsa. IConnectionCallbacks sisältää muutaman muunkin callback-funktion, jotka ovat kuvan alalaidassa, kuten OnConnected, joka tapahtuu, kun yhteys on muodostettavissa, mutta pelaaja ei ole vielä yhteydessä keskeiselle palvelimelle, eli ei ole valmis esimerkiksi matchmakingiin.

4.3 Pelaajien yhdistäminen toisiinsa

Kuvassa 10 on esimerkki koodista, jossa toteutetaan pelaajien yhdistäminen toisiinsa sekä satunnaista että yksityistä matchmakingia käyttäen. Julkisessa käyttöön on otettu funktio JoinRandomOrCreateRoom, joka pyrkii etsimään avoimen huoneen, jossa on tilaa pelaajalle. Sellaisen löytyessä liitytään siihen, kun taas jos sellaista ei löydy, luodaan automaattisesti uusi avoin huone. Matchmaking-prosessia voidaan määrittää halutunlaiseksi. Tässä tapauksessa tehdään kaksi asiaa. Funktioon syötetään pitkälti arvot, jotka vastaavat sen oletusarvoista toiminnallisuutta. Asetetaan siihen kuitenkin myös tietty rajallinen pelaajamäärä, neljä, ja luodaan mukautettu arvo, tässä tapauksessa ”GameMode”, jota käyttäen pelissä voitaisiin suorittaa matchmaking vain tiettyyn pelimuotoon, esimerkissä tämä olisi ”Classic”.



```
using TMPro;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;
using ExitGames.Client.Photon;

Unity Script (1 asset reference) | 0 references
public class PhotonMatchmaking : MonoBehaviour
{
    // Attempts to find a, visible, open and not-full room for 4 players.
    // Also sets a custom room property "GameMode" to a type "Classic", which allows matchmaking for a gamemode.
    // Oldest applicable room is selected first. If one is not found, creates one instead.
    0 references
    public void MatchmakePublic()
    {
        RoomOptions roomOptions = new RoomOptions();
        roomOptions.MaxPlayers = 4;
        Hashtable customRoomProperties = new Hashtable();
        customRoomProperties.Add("GameMode", "Classic");
        roomOptions.CustomRoomProperties = customRoomProperties;
        roomOptions.CustomRoomPropertiesForLobby = new string[] { "GameMode" };

        PhotonNetwork.JoinRandomOrCreateRoom(null, 4, MatchmakingMode.FillRoom, TypedLobby.Default, null, null, roomOptions);
    }

    // Attempts to find an open room for 4 players that is not full, with the name taken from an input field in UI.
    // If one is not found, creates one instead, and sets it not visible, so it cannot be joined through random matchmaking,
    public TMP_InputField roomInput;
    0 references
    public void MatchmakePrivate()
    {
        if (roomInput.text != "")
        {
            RoomOptions roomOptions = new RoomOptions();
            roomOptions.MaxPlayers = 4;
            roomOptions.IsVisible = false;
            PhotonNetwork.JoinOrCreateRoom(roomInput.text, roomOptions, TypedLobby.Default, null);
        }
    }
}
```

Kuva 10. Esimerkit sekä julkisesta että yksityisestä matchmaking-prosessista.

Matchmaking-prosessi voidaan myös rajata vain tietynnimisen huoneen hakemiseen tai luomiseen, jolla voidaan saada aikaiseksi yksityisiä huoneita, mihin pelaajat eivät voi liittyä satunnaisesti. Sitä varten kuvan esimerkkikoodissa on luotu myös pelin käyttöliittymään määritettävä syöttökenttä, josta saadaan huoneen nimi sitä hakiessa tai luodessa. Koska funktio vaatii sen, koodissa on myös tarkastus sille, että syöttökentässä on jokin arvo, ennen kuin se aloittaa matchmaking-prosessin. Tässä tapauksessa käytetään myös eri funktiota, JoinOrCreateRoom, jossa suurin ero julkisessa matchmakingissa käytettyyn on se, että se vaatii tietyt argumentit syötettäväksi, jotka konfiguroivat sen toteuttaman matchmaking-prosessin, ja kyseinen prosessi ei sisällä satunnaista huoneen etsimistä. Määritellyin asetuksin funktio etsii tietynnimisen huoneen ja liittyy siihen, kun taas jos sellaista ei löydy, luo se uuden huoneen. Jotta huoneeseen voi liittyä vain sen nimen tietämällä, pitää sen näkyvyys myös asettaa piilotetuksi, jolloin satunnainen matchmaking ei huomioi kyseistä huonetta.

Kuvassa 11 on lisäys kuvassa 10 luotuun luokkaan, jossa asetetaan automaattinen skenejen synkronointi päälle ja on näytillä muutama matchmaking-prosessin callback-funktio. Automaattinen skenejen sykronointi tarkoittaa sitä, että kun huoneen isäntä siirtyy johonkin eri pelinäkymään, siirtyvät muutkin pelaajat automaattisesti. Kuten aikaisemmin mainittiin, huoneen isännällä on

enemmän oikeuksia huoneen pelilogiikan hallinnointiin, ja asioiden kuten skenejen vaihdosten asettaminen huoneen isännän vastuulle, voi auttaa ehkäisemään ongelmia esimerkiksi pelisession synkronoinnissa, joita saattaa ilmentyä, jos kaikilla pelaajilla on tasaväkinen vaikutus pelin logiikkaan.

```
// Sets automatic scene synchronisation on, so if master loads a new scene, others automatically follow.
Unity Message | 0 references
private void Awake() { PhotonNetwork.AutomaticallySyncScene = true; }
// A callback that happens when a room is joined (created or not).
// Also demonstrates how one can check if local client is the master of the room.
0 references
public void OnJoinedRoom()
{
    if(PhotonNetwork.IsMasterClient) { Debug.Log("OnJoinedRoom,IsMasterClient"); }
    else { Debug.Log("OnJoinedRoom,!IsMasterClient"); }
}
// A callback that happens when a room is created.
0 references
public void OnCreatedRoom() { Debug.Log("OnCreatedRoom"); }
// A callback that happens when a room is left for any reason.
0 references
public void OnLeftRoom() { Debug.Log("OnLeftRoom"); }
```

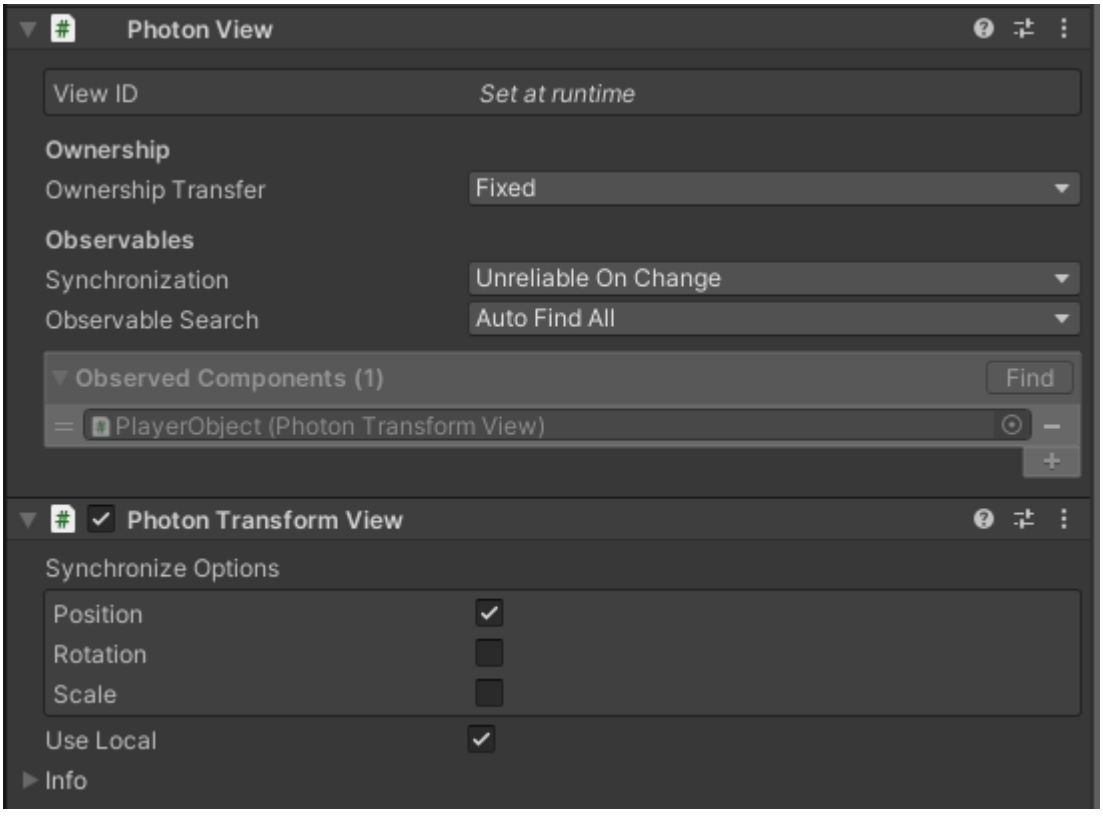
Kuva 11. Automaattisen skenejen synkronoinnin asettaminen käyttöön ja erinäisiä matchmaking-prosessin callback-funktioita.

IMatchmakingCallbacks-rajapinta tuo käytettäväksi matchmaking-prosessiin liittyviä callback-funktioita. Kuvaan 11 näistä on laitettu esimerkeiksi OnJoinedRoom, OnCreatedRoom ja OnLeftRoom, mutta se sisältää myös muun muassa matchmaking-prosessin epäonnistumiseen liittyviä funktioita. Kukin esimerkissä käytetyistä funktioista sisältää vain debuggauskonsoliin lähetettävän viestin, mutta pelissä niitä voidaan käyttää suorittamaan logiikkaa, joka halutaan toteutuvan juuri silloin, kun callback-funktion kutsuma tapahtuma on toteutunut varmasti.

OnJoinedRoom tapahtuu silloin, kun pelaaja liittyy huoneeseen, riippumatta siitä, onko se pelaajan luoma vai aikaisemmin olemassa ollut. Jos halutaan esimerkiksi tuoda pelissä esiin käyttöliittymä, joka on luotu tilanteeseen, missä pelaaja on huoneessa, voidaan se toteuttaa tällä funktiolla. Jos pelissä pitää pelaajien esimerkiksi asettaa itsensä valmiiksi ottelun alkuun, voidaan siihen liittyvä nappi tuoda esiin tämän funktion toteutuksessa, koska pelaaja on silloin huoneessa. Lisäksi joissain harvinaisissa tapauksissa pelaajasta voi teoriassa tulla huoneen isäntä samalla hetkellä, kun hän liittyy siihen. Se voidaan tarkastaa esimerkkikuvan mukaisesti, jos vaikka huoneen pelaajamäärän muuttuessa on tarve erilliselle logiikalle, jonka isäntä toteuttaa. OnCreatedRoom taas tapahtuu vain silloin, kun pelaaja luo huoneen, ja tällöin hän on myös oletusarvoisesti aina isäntä, ellei sitä erikseen vaihdeta. OnLefRoom toteutuu silloin, kun pelaaja poistuu huoneesta. Jos käytössä on esimerkiksi aikaisemmin mainittu käyttöliittymä, voidaan se piilottaa, kun tämä funktio tapahtuu.

4.4 Moninpelin synkronointi

Yksinkertainen peliobjektin, esimerkiksi pelaajahahmon, sijainnin synkronointi vaatii kaksi PUN 2:n lisäämää komponenttia. Ensimmäinen on Photon View, joka tekee kyseisestä objektista osallisen pelin synkronointiin. Se sisältää muutaman tärkeän eri ominaisuuden. Ensimmäinen on kyseisen Photon View'n ID, joka asetetaan automaattisesti, ja mitä voidaan tarvita joissain pelin tapahtumissa, mutta aina ei siihen tarvitse ohjelmoijan koskea millään tavalla. Photon View'lle voidaan määritellä, "omistaako" aina vain sen luoja kyseisen objektin vai voiko joku muu ottaa sen haltuun joko väkisin tai pyynnöstä. Objektin omistaja on se, jolta Photon View jakaa muillekin pelaajille tiedon objektin tilasta. Kuvassa 12, omistajan tila on jätetty muuttumattomaksi eli se säilyy sillä pelaajalla, jonka päässä objekti luodaan pelin aikana. Tämä toimii esimerkiksi pelaajahahmoissa, koska yleensä niitä ei hallitse kukaan muu kuin pelaaja itse.



Kuva 12. Objektin liikkeen yksinkertaiseen synkronointiin vaadittavat komponentit.

Photon View toimii tarkkailemalla joitain muita komponentteja, joiden tietoja se lähettää eteenpäin muille pelaajille. Synkronointimetodia voidaan muuttaa riippuen siitä, halutaanko siitä tehokkaampi tai luotettavampi, tai se voidaan ottaa kokonaan pois päältä tilapäisesti. Tarkkailtavat

objektit voidaan hakea automaattisesti, tai ne voidaan asettaa itse. Kuvassa 12 Photon View on löytänyt automaattisesti toisen objektiin lisätyn komponentin, nimeltään Photon Transform View. Se seuraa objektin tilaa pelimaailmassa, sijaintia, rotaatiota ja kokoa. Näillä kahdella komponentilla saadaan synkronoitua objektin tilan tiedot yksinkertaisesti.

Jos olisi tarve monimutkaisempien asioiden synkronoinnille, voidaan luoda skripti sitä varten ja käyttää yhtä eri keinoista, mitä PUN 2 sen toteutukseen tarjoaa. Kuvassa 13 on käytössä IPunObservable-rajapinta, jolla saadaan ohjelmoitua PhotonView'n tarkastelema komponentti. Se sisältää funktion OnPhotonSerializeView, joka kirjoittaa tai lukee dataa PhotonView'n kautta. Esimerkkikoodissa sitä käytetään teoreettisen objektin sijaintitiedon synkronointiin. Jos tietoa kirjoitetaan, on paikallinen pelaaja sen PhotonView'n omistaja, joka tarkkailee tätä koodia, ja siten voidaan lähettää muuttujan tieto muille pelaajille. Jos hän ei ole objektin omistaja, voidaan lukea tietoa, ja muuttaa paikallisesti kyseisen muuttujan arvo siksi, mikä luettu arvo on.

```
using UnityEngine;
using Photon.Pun;
Unity Script | 0 references
public class PhotonObservableExample : MonoBehaviourPun, IPunObservable
{
    // The variable to be synchronised, in this case a theoretical position in the game world.
    public Vector3 Position;

    // If this script is assigned as an observed component for a PhotonView, this function is called at a given interval.
    // If the local player is the owner of that PhotonView, it sends the information, and if not, it receives it.
    3 references
    public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if(stream.IsWriting) { stream.SendNext(Position); }
        else { Position = (Vector3)stream.ReceiveNext(); }
    }
}
```

Kuva 13. Muuttujan arvon synkronointi IPunObservable-rajapintaa hyödyntäen.

PhotonView'n sisältämiä objekteja voi olla skenessä jo sen alkaessa, ja niiden haltuunotto voidaan suorittaa pelin aikana. Usein peleissä on kuitenkin myös tarpeen luoda objekti itse pelin aikana, kuten aikaisemminkin mainittiin. PUN 2 sisältää oman Instantiate-funktionsa, joka vastaa pitkälti Unityn omaa, mutta on olemassa juuri PUN 2-synkronoitavien objektien luomiseen. Se asettaa objektin PhotonView'n ID:n sitä luodessa, ja myös lisää objektiin kyseisen komponentin, jos sitä ei siinä valmiiksi ole, jotta se on heti valmis synkronointiin.

```
using UnityEngine;
using Photon.Pun;

Unity Script (1 asset reference) | 0 references
public class PhotonInstantiateExample : MonoBehaviour
{
    // The variable that will store the instantiated object.
    GameObject myCharacter;
    // This function uses Photon's custom Instantiate that attempts to find a "PlayerObject" from resources.
    0 references
    public void MakePlayerCharacter()
    {
        myCharacter = PhotonNetwork.Instantiate("PlayerObject", new Vector3(0, 0, 0), Quaternion.identity);
    }
}
```

Kuva 14. Synkronoitavan objektin luominen pelimaailmaan.

Tärkeintä kyseistä funktiota käyttäessä on muistaa, että se käyttää Unityn toiminnallisuutta, jossa ladataan Resources-kansiosta objekti, eikä siis käytä koodiin kirjoitettua objektiin viittausta. Käytännössä se tarkoittaa sitä, että objekti, joka halutaan luoda pelimaailmaan sitä käyttäen, sijoitetaan Resources-kansioon Unityn editori-ikkunassa, ja sen nimi syötetään funktioon. Esimerkkikuvassa 14 objektin nimi on ”PlayerObject”. Muuten funktio tarvitsee sijaintitiedon ja kvaternionin, joka merkitsee objektin rotaatiota, kuten Unityn omakin Instantiate-funktio.

4.5 Moninpelin kulun esimerkki

PUN 2 tarjoaa suoranaisesti toiminnallisuuden synkroniseen, reaaliaikaiseen pelin kulkuun, josta kirjoitettiin aiemmin teoriaosiossa. Objektien tila voi olla jatkuvasti muuttuva, ja kukin pelaaja, ellei heitä erikseen estetä, voi olla koko ajan vaikutuksessa pelin tilaan. Tässä osiossa annetaan malliesimerkki siitä, miten voitaisiin ohjelmoida erilainen pelin kulku. Otetaan esimerkiksi reaaliaikainen vuoropohjainen peli. Kukin pelaaja saa omalla vuorollaan liikuttaa hahmojaan ja olla aloittavassa roolissa eri toiminnoissa, mitä pelissä on mahdollista käyttää. Oman vuoronsa ulkopuolella taas ei voi tehdä mitään. Sanotaan, että kullakin on minuutti aikaa tehdä toimintonsa vuorollaan, ja jos sitä ei itse päätä, niin vuoro vaihdetaan automaattisesti seuraavaan. Tutkitaan tässä osiossa, miten voitaisiin toteuttaa pelaajien kommunikointi vuorojen vaihtojen suhteen. Asetetaan kaikki logiikka tapahtumaan huoneen isännän kautta, koska kun isäntä toimii pelilogiikan keskuksena, voidaan ehkäistä mahdollisia synkronointiongelmia.

Lähdetään liikkeelle siitä, että pelin alussa huoneen isäntä ottaa ylös jokaisen pelaajan Actor Numberin, kuten kuvassa 15 Start-funktion sisällä. Jokaisella huoneessa olevalla pelaajalla on

omansa, ja niistä ei voi olla kopioita, joten se on hyvä keino pitää selvillä, kuka kukin on. Seuraavaksi kuvassa isäntä käyttää Remote Procedure Callia, eli RPC:tä, nimeltä UpdateTurnStatus, jossa kukin pelaaja selvittää, onko hänen vuoronsa vai ei. Tutkitaan sitä hieman myöhemmin.

```
List<int> turnOrder = new List<int>(); // The list of Actor Numbers of each of the players in session.
float turnTimer = 60; // Countdown for the turns, in seconds (so 1 minute).
bool isTurn = false; // Is it the local client's turn currently?
bool countDown = false; // Should the turn timer be counting down?

// Start doesn't need to do anything unless the local client is the master.
// If local client is master, first we get the actor numbers of all the clients in room and add them to the List.
// Then we remotely call an RPC on all players, where each checks if the sent integer matches their Actor Number.
// If it does, it's their turn and isTurn is set to true, otherwise isTurn is set to false.
// countDown is also set as true to begin the 60 second timer on the turn.
Unity Message | 0 references
void Start()
{
    if (PhotonNetwork.IsMasterClient)
    {
        turnOrder = PhotonNetwork.PlayerList.Select(player => player.ActorNumber).ToList();
        photonView.RPC("UpdateTurnStatus", RpcTarget.All, turnOrder[0]);
        countDown = true;
    }
}
// Only if master and countDown is true, if turn timer is still not 0, reduce the amount of time in seconds that
// has elapsed since the last frame before this one, if it is 0, call the function that starts a new turn.
Unity Message | 0 references
private void Update()
{
    if (PhotonNetwork.IsMasterClient && countDown)
    {
        if (turnTimer >= 0) { turnTimer -= Time.deltaTime; }
        else { StartNewTurn(); }
    }
}
// The first function is to be attached to a button that allows ending the turn before timer reaches 0.
// It remotely calls an RPC function on the master client, that tells the master to start changing the turn.
0 references
public void EndTurn() { if(isTurn) { photonView.RPC("EndTurnInform", RpcTarget.MasterClient); } }
0 references
[PunRPC] public void EndTurnInform() { StartNewTurn(); }
```

Kuva 15. Teoreettisen reaaliaikaisen vuoropohjaisen pelin aloitus ja vuorojen päättäminen.

Käytännössä RPC on funktio, joka voidaan kutsua etänä toisen pelaajan laitteelta. RPC:ksi halutut funktiot pitää sellaisiksi erikseen merkitä, ja PUN 2:ssa tämä tehdään laittamalla funktion eteen [PunRPC], kuten kuvan 15 RPC:ksi merkitty funktio EndTurnInform. RPC:n kutsumiseen tarvitaan Photon View -komponentti, jonka kautta se tapahtuu, ja sitä kutsuessa siihen syötetään funktion nimi string-muodossa ja se, kenellä funktio etänä kutsutaan. Jos RPC:ksi merkityssä funktiossa on joitain muuttujia, mitä se vaatii, syötetään ne järjestyksessä niiden kahden jälkeen. Kuvassa UpdateTurnStatus kutsutaan etänä kaikille huoneen pelaajille, sillä jokainen tarkistaa itse, onko heidän vuoronsa, isännän antaman tiedon perusteella. EndTurnInform kutsutaan etänä aina vain isännällä, koska sen tarkoitus on kertoa isännälle, että on aika aloittaa uusi vuoro ennenaikaisesti.

Start-funktio päättyy vuoron ajastimen aloittamiseen, sillä ensimmäinen vuoro on alkanut. Ajastinta seuraa vain isäntä, ja se yksinkertaisesti laskee noin 60 sekuntia, ja sitten kutsuu funktion StartNewTurn, jolla aloitetaan uusi vuoro, jos pelaaja ei ole päättänyt vuoroansa, ennen kuin aika

päättyy. Kuvassa on vielä funktio EndTurn, jolla pelaaja päättää vuoronsa, ja voitaisiin liittää esimerkiksi nappiin. Se kutsuu etänä funktion EndTurnInform isännällä, kuten aikaisemmin mainittiin. Kyseisestä funktiostakin kutsutaan StartNewTurn, joten tarkkaillaan sitä seuraavaksi.

Kuvassa 16 tapahtuu vuoron vaihdoksen logiikka. StartNewTurn on funktio, jossa pysäytetään ajastin vuoron vaihdoksen ajaksi, ettei vuoronvaihdosta aloiteta vahingossa uudelleen ennenaikaisesti, ja funktion lopussa se resetoidaan ja aloitetaan uudelleen uuden vuoron alettua. Itse vuoron vaihdos tapahtuu yksinkertaisesti. Ensin muuttuja, joka jäljittää, kenen vuoro on, muutetaan joko aloittamaan vuorolistan alusta tai jatkamaan seuraavaan, riippuen siitä, onko kukin pelaaja saanut vuoronsa tällä ”kierroksellsa”. Sitä käytetään aivan alussa täytetyltä Actor Numberit sisältävältä listalta arvon hakemiseen, joka syötetään UpdateTurnStatus-etäkutsun mukana. Kuten aikaisemmin mainittiin, UpdateTurnStatus sisältää vain yksinkertaisen vertailun, jossa kukin vertaa lähetettyä arvoa omaan Actor Numberiinsa, ja niiden ollessa samat, on kyseisen pelaajan vuoro, kun taas jos ne eivät ole, ei ole pelaajan vuoro.

```
int currentTurn = 0; // Which turn is it currently?
// When the turn changing process begins, first the timer is stopped.
// Then, if increasing currentTurn by 1 would reach a value equal to players in room, it is set to 0,
// as a new round needs to be started from the first player. Otherwise currentTurn is increased by one.
// Then we remotely call an RPC on all players, where each checks if the sent integer matches their Actor Number.
// Whoever it matches, starts their turn. Finish by resetting and restarting the turn countdown for this new turn.
2 references
public void StartNewTurn()
{
    countDown = false;
    if (currentTurn + 1 == PhotonNetwork.CurrentRoom.PlayerCount) { currentTurn = 0; }
    else { currentTurn++; }
    photonView.RPC("UpdateTurnStatus", RpcTarget.All, turnOrder[currentTurn]);
    turnTimer = 60;
    countDown = true;
}
// A simple logic where each client checks if the sent newTurn value matches their actor number.
// If it does, it's their turn, otherwise it's not.
0 references
[PunRPC] public void UpdateTurnStatus(int newTurn)
{
    if (PhotonNetwork.LocalPlayer.ActorNumber == newTurn) { isTurn = true; }
    else { isTurn = false; }
}
```

Kuva 16. Voron vaihdoksen logiikka.

Tämä logiikka on kaiken kaikkiaan yksinkertainen, ja pelikäyttöön sitä todennäköisesti tulisi laajentaa ainakin kahdella tavalla. Ensinnäkin, kun pelaajien Actor Numberit haetaan suoraan pelaajalistalta, tulevat järjestykseen vanhimmasta uusimpaan. Jos vuorojärjestyksen halutaan olevan jokin muu, täytyy kirjoittaa sen uudelleenjärjestely. Toiseksi esimerkkikoodi ei sisällä reaktioita siihen, jos jokin pelaaja poistuu kesken pelin, tai jostain muusta pelaajasta tulee isäntä. Tällöin pelilogiikka todennäköisesti sekoaa. Kuvassa 17 näkyviä huoneen pelaajakokoonpanoon liittyviä

callback-funktioita hyödyntäen voidaan teoreettisesti toteuttaa logiikka, jolla peliä voidaan jatkaa sujuvasti tällaisen tilanteen tullen.

```
// The first callback is called on a player in a room, when someone else leaves the room.
// The second callback is called on a player in a room, when someone else becomes the master.
3 references
public override void OnPlayerLeftRoom(Player otherPlayer) { }
3 references
public override void OnMasterClientSwitched(Player newMasterClient) { }
```

Kuva 17. Callback-funktiot, joita voitaisiin hyödyntää, jos pelaajamäärä tai huoneen isäntä muuttuu pelin aikana.

4.6 Yhteydessä ilmenevien ongelmien seuraamukset

Pelissä on mahdollista ilmentyä erilaisia ongelmia yhteydessä palvelimelle. Tällaisiin kannattaa valmistautua. IConnectionCallbacks-rajapinta sisältää callback-funktion OnDisconnected, jonka toteutuessa se antaa myös syyn, miksi pelaajan yhteys palvelimelle katkaistiin. Syitä on useita, ja kuvassa 18 on näytillä viisi näistä syistä. Tutkitaan niitä lähemmin, jotta saadaan ajatus, millaisia syitä yhteyden katkeamiselle voi olla ja miten niihin voitaisiin reagoida pelissä.

```
0 references
public void OnDisconnected(DisconnectCause cause)
{
    // Server is not receiving communication from client within expected time frame.
    if (cause == DisconnectCause.ServerTimeout) { }

    // Client is not receiving communication from server within expected time frame.
    if (cause == DisconnectCause.ClientTimeout) { }

    // Too many operation calls from client within a given time frame.
    if (cause == DisconnectCause.DisconnectByOperationLimit) { }

    // Server logic disconnects the client due to custom server-side conditions.
    if (cause == DisconnectCause.DisconnectByServerLogic) { }

    // Server disconnects the client for an unknown reason.
    if (cause == DisconnectCause.DisconnectByServerReasonUnknown) { }
}
```

Kuva 18. PUN 2:n tunnistamia syitä yhteyden katkeamiselle.

Ensimmäiset kaksi syytä ovat samankaltaisia. Ensimmäisessä palvelin ei saanut vastausta pelaajan kanssa aikarajan sisällä, ja toinen sitä, että pelaaja ei saanut vastausta palvelimelta. Jos palvelin ei saa pelaajalta vastausta, voi syy olla pelaajan päässä, esimerkiksi huonossa verkkoyhteydessä,

ja toisessa palvelimella voi olla jotain ongelmia. Tällöin ei välttämättä ole muuta tehtävissä, kuin ilmoittaa pelaajalle ongelman laadusta ja mahdollisesti yrittää ottaa yhteys uudelleen automaattisesti. Kolmas esimerkkikuvan syy johtuu siitä, että pelaaja yrittää liikaa eri toimintoja, kuten palvelimelle liittymisyrityksiä tai vastaavia lyhyellä aikavälillä. Palvelin voi ylikuormittua, jos sille tulee liikaa kommunikaatioyrityksiä. Jonkinlainen rajaus sille, kuinka nopeasti pelaaja voi yrittää eri moninpelioperaatioita, voi estää tämän ongelman kokonaan.

Neljäs syy johtuu siitä, että pelaajan yhteys palvelimelle on katkaistu mukautetun logiikan takia, esimerkiksi, jos peliin on ohjelmoitu jotain sääntöjä, joiden rikkomisen takia pelaajan yhteys voidaan katkaista automaattisesti. Tällöin parasta lienee vain ilmoittaa pelaajalle syy yhteyden katkaisemiselle, jotta hän ei riko kyseisiä sääntöjä uudelleen. Viimeinen esimerkkikuvassa oleva syy taas on tuntematon syy. Joskus saattaa tapahtua yhteyden katkeaminen, jota ei voida selittää millään yleisistä syistä, mitä IConnectionCallbacks tuntee. Tässä tapauksessakin voi olla olemassa jokin geneerinen viesti, joka kommunikoi pelaajalle yhteyden katkenneen. Siitä voidaan jatkaa esimerkiksi yritykseen ottaa yhteys uudelleen.

4.7 Google Play Games plugin for Unityn asennus ja käyttöönotto

Jotta Google Play Games plugin for Unitya voidaan käyttää, pitää olla pääsy Google Play Console -sivulle, jossa konfiguroidaan sovellus ja sen mahdolliset sosiaaliset ominaisuudet. Se vaatii Google Play -kehittäjätilin, joka pitää luoda erikseen ja jonka avaaminen maksaa 25 Yhdysvaltain dollaria. Tässä työssä keskitytään ohjelmointiin, joten jatkossa oletetaan, että Google Play Consolessa on onnistuneesti saatu luotua kaikki tarpeelliset puitteet sille.

Google Play Games plugin for Unity on saatavilla sen GitHub-versionhallintasivulta. Se voidaan ladata pakattuna tiedostona, josta löytyy muun muassa unitypackage-tyyppinen tiedosto. Se voidaan tuoda avoinna olevaan Unity-projektiin. Käyttöönottoa varten pitää olla tehty valmisteleva työ Developer Consolessa, josta pitää kopioida Android Resources-osiosta teksti, jolle on oma kenttänsä lisäosan käyttöönoton käyttöliittymässä. Kyseinen teksti sisältää tiedot muun muassa sovellukseen luoduista sosiaalisista ominaisuuksista. Jos Google Play Consolessa tehdään muutoksia myöhemmin, pitää myös muuttunut Android Resources -teksti käydä päivittämässä Unityyn, jotta uusia tai muuttuneita ominaisuuksia voi käyttää myöhemmin.

4.8 Play Games -sisäänkirjautuminen

Käytännön osion avauksessa mainittiin, että jotkin Play Games plugin for Unityn ominaisuudet toimivat myös Unityn oman Social-rajapinnan kautta. Lisäosan funktio Activate ottaa tämän toiminnallisuuden käyttöön, ja kuvan 19 esimerkkikoodissa se kutsutaankin heti Start-funktiossa, jotta tämä tapahtuu automaattisesti skenen auetessa. Lisäksi siinä kutsutaan funktio Authenthicate, joka on funktio, joka toteuttaa automaattisen Google Play Games-sisäänkirjautumisyrityksen. Se vaatii callback-funktion, joka kutsutaan automaattisesti sisäänkirjautumisyrityksen päätyttyä ja sisältää tietoa yrityksen onnistumisesta tai epäonnistumisesta.

```
using UnityEngine;
using GooglePlayGames;
using GooglePlayGames.BasicApi;

Unity Script | 0 references
public class GoogleSignIn : MonoBehaviour
{
    // Start activates the plugin functionality that allows using Unity's Social interface features for Play Games services.
    // Then it attempts to sign into the local Play Games account for this game.
    Unity Message | 0 references
    void Start()
    {
        PlayGamesPlatform.Activate();
        PlayGamesPlatform.Instance.Authenticate(AuthenticationResult);
    }
    // Manual authentication can be called if automatic Authenticate fails, to prompt the player to sign in.
    0 references
    internal void ManualAuthentication() { PlayGamesPlatform.Instance.ManuallyAuthenticate(AuthenticationResult); }
    // This function informs us whether Sign In succeeded. Right now it just Debug Logs.
    2 references
    internal void AuthenticationResult(SignInStatus status)
    {
        if (status == SignInStatus.Success) { Debug.Log("Google Play Games sign in succesful, features can be used."); }
        else { Debug.Log("Google Play Games sign in failed."); }
    }
}
```

Kuva 19. Google Play Games -sisäänkirjautuminen.

Esimerkkikoodissa callback-funktio on toteutettu nimellä AuthenthicationResult. Siinä ei tässä tapauksessa tehdä muuta kuin tarkistetaan, että onko SignInStatus-muuttujan arvo Success vai jokin muu. Muut vaihtoehdot sen arvolle ovat epäonnistuminen ja keskeytys. Jos sen arvo on Success, on Google Play Games -sisäänkirjautuminen onnistunut. Jos se on jokin muu, se ei ole onnistunut, ja pelissä saattaa olla tarpeen implementoida vielä tarkastukset sille, onko prosessi epäonnistunut virheen vai tarkoituksellisen keskeytyksen takia. Esimerkkikoodissa on myös käytössä funktio ManuallyAuthenthicate, joka eroaa aikaisemmin käytetystä Authenthicate-funktiosta lähinnä sillä, että se erikseen pyytää käyttäjää kirjautumaan sisään. Sekin vaatii callback-funktion, ja tässä tapauksessa siihen on yhdistetty sama kuin aikaisemmin.

4.9 Kaverit

Vaikka kaverilistan näkyvyyden tarkastaminen ei ole täysin pakollista toteuttaa ennen kuin siitä voidaan alkaa hakemaan tietoa, on useimmissa tapauksissa viisainta tehdä niin, jotta voidaan jatkossa välttää mahdolliset ongelmat, jotka johtuvat jo näkyvyyden puutteesta. Esimerkkikuvassa 20 se on toteutettu esimerkissä funktion CheckIfFriendsListVisible sisällä, ja se on laitettu vaatimaan boolean-muuttujan syötettäväksi sitä kutsuessa. Kyseisen muuttujan arvo määrittelee sen, kuuluuko kaverilista ladata uudelleen palvelimelta, jos se on tosi, vai käytetäänkö paikallisesti talletettua dataa, joka on nopeampi ratkaisu, jos epätosi. Kaverilistan näkyvyyden tarkastus tapahtuu funktiolla GetFriendsListVisibility, joka toteuttaa itse kaverilistan näkyvyyden tarkastuksen ja johon aikaisempi boolean-muuttuja myös syötetään. Se vaatii myös FriendsListVisibilityStatus-tyyppisen callback-implementaation.

```
using UnityEngine;
using GooglePlayGames;
using GooglePlayGames.BasicApi;
Unity Script | 0 references
public class GoogleFriends : MonoBehaviour
{
    // A function that checks the friends list visibility. shouldReload determines if it's loaded from local cache or fetched from cloud.
    // Implemented visibility statuses are: Visible, which means it can be used and loaded.
    // ResolutionRequired, which means the player needs to be prompted for a permission to access it.
    // Unknown, which means... failure reason unknown. If local cache was loaded, attempts to fetch from cloud instead, if that'd work.
    1 reference
    public void CheckIfFriendslistVisible(bool shouldReload)
    {
        PlayGamesPlatform.Instance.GetFriendsListVisibility(shouldReload, (FriendsListVisibilityStatus status) =>
        {
            if(status == FriendsListVisibilityStatus.Visible) { Debug.Log("Friends list accessible, can be loaded."); }
            else if(status == FriendsListVisibilityStatus.ResolutionRequired) { AskForFriendAccess(); }
            else if(status == FriendsListVisibilityStatus.Unknown) { if(shouldReload == false) { CheckIfFriendslistVisible(true); } }
            // If need be, implement reactions to other FriendsListVisibilityStatuses as well.
        });
    }
```

Kuva 20. Kaverilistan näkyvyyden tarkastus.

Kyseinen callback kertoo, löytyikö kaverilista ja onko se näkyvä vai vaatiiko sen tarkkailuluvan, tai mahdollisia muita funktion lopputuloksia. Ensimmäinen implementoitu lopputulos on yksinkertainen Visible, joka tarkoittaa, että kaverilistan etsiminen onnistui ja on näkyvillä eli käytettävissä. Siitä voidaan siis alkaa hakemaan tietoa. Siihen on kaksi eri toteutuskeinoa, jotka kummatkin esitellään hieman myöhemmin. Toinen lopputulos taas on ResolutionRequired, joka tarkoittaa, että kaverilista löytyi, mutta sen käsittelyyn ei ole vielä lupaa. Esimerkissä on implementoitu luvan kysyminen funktiossa AskForFriendAccess, joka kutsutaan tässä tapauksessa, ja esitellään myös myöhemmin.

Kolmas implementoitu tulos on Unknown, joka on yleinen tunnistamaton syy epäonnistumiselle, eli ei osata sanoa esimerkiksi, oliko puute yhteydessä tai jossain muussa. Tässä tapauksessa esi-

merkki siihen reagoinnista on se, että jos yritettiin käyttää paikallista dataa, voi puute olla esimerkiksi paikallisen datan viallisuudessa. Jos käytettiin paikallista dataa, kokeillaan siis vielä kutsua CheckIfFriendsListVisible-funktio uudelleen, mutta syötetään haluttu boolean-arvo totena, eli toisin sanoen yritetään hakea data palvelimelta paikallisen sijaan. Esimerkissä on siis implementoitu reaktio kolmeen eri lopputulokseen, jolla pääsee aluilleen kaverilistan näkyvyyden tarkastuksessa. Pelistä ja tilanteesta riippuen saatetaan myös haluta erilliset reaktiot esimerkiksi siihen, oliko kaverilistan haussa jokin yhteyteen liittyvä ongelma. Joihinkin yleisistä lopputuloksista on omat FriendsListVisibilityStatus-arvonsa, ja funktiota voidaan siis laajentaa niillä tarpeen tullen.

Yksi keino päästä käsiksi kaverilistaan on Unityn Social-rajapintaa hyödyntäen, joka on kuvan 21 esimerkkikoodissa toteutettu funktiossa LoadFriendsSocial. Se on yksinkertainen ratkaisu, jossa käytetään kyseisen rajapinnan funktiota LoadFriends, joka ei vaadi muuttujiakaan syötöksi, vaan pelkästään boolean-tyyppisen callback-implementaation, joka kertoo, onnistuiko kaverilistan haku vai ei. Jos se onnistui, tulee se käsiteltäväksi muiden Social-rajapinnan ominaisuuksien kautta. Jos yritys epäonnistui, voidaan sen syytä tarkastella eräällä funktiolla, mikä esimerkissä kutsutaan epäonnistumisen takia. Tutkitaan sen toimintaa myöhemmin. Huomioitavaa on, että lisäosan funktio Activate täytyy olla kutsuttu, jotta Social-rajapinta on yhdistetty Play Games -palveluun ja voi käsitellä sen ominaisuuksia.

```
// Loading friends through Unity's social interface. It can only succeed or fail. If it fails, run the function that checks the reason.
0 references
public void LoadFriendsSocial()
{
    Social.localUser.LoadFriends((bool success) =>
    {
        if(success == true) { Debug.Log("Success, friends can be accessed with Social.localUser.friends."); }
        else { CheckFailureReason(); }
    });
}
// Loading friends through the plugin. It loads a specified number and reloads only if told to.
// There may be more friends left to be loaded, in which case LoadMoreFriends can be called to continue where it was left off.
// If it fails, run the function that checks the reason.
0 references
public void LoadFriendsGPG(int size, bool reload)
{
    PlayGamesPlatform.Instance.LoadFriends(size, reload, (status) =>
    {
        if (status == LoadFriendsStatus.Completed) { Debug.Log("Success, and all friends are loaded."); }
        if (status == LoadFriendsStatus.LoadMore) { Debug.Log("Success, but more friends can be loaded."); }
        else { CheckFailureReason(); }
    });
}
// Loading more friends through the plugin. It can have the same results as the prior function.
// It just cannot restart the process, only continue where previous one left off.
0 references
public void LoadMoreFriendsGPG(int size)
{
    PlayGamesPlatform.Instance.LoadMoreFriends(size, (status) =>
    {
        if (status == LoadFriendsStatus.Completed) { Debug.Log("Success, and all friends are loaded."); }
        if (status == LoadFriendsStatus.LoadMore) { Debug.Log("Success, but more friends can be loaded."); }
        else { CheckFailureReason(); }
    });
}
```

Kuva 21. Kaverilistan lataaminen sekä Unityn Social-rajapinnan, että Google Play Games plugin for Unityn oman toiminnallisuuden kautta.

Toinen keino hakea kaverilista on Google Play Games plugin for Unityn oma ratkaisu, joka on hieman monimutkaisempi, mutta sen takia myös hallittavampi. Siinä on kaksi funktiota, joista toinen on LoadFriends ja toinen LoadMoreFriends, jotka ovat implementoitu omissa funktioissaan LoadFriendsGPG ja LoadMoreFriendsGPG.

Yhteistä kummassakin funktiossa on se, että ne haluavat tasalukutyyppisen muuttujan, joka kertoo, kuinka monta kaveria listalta ladataan sillä funktion kutsulla, ja kummassakin on myös vaatimus LoadFriendsStatus-callback-implementaatiolle. Tarkkaillaan ensin funktiota LoadFriends, joka on kaverien lataamisprosessin aloittava funktio ja vaatii myös edellä mainittujen asioiden lisäksi boolean-tyyppisen muuttujan, joka määrittää, ladataanko lista uudelleen. Sen ollessa tosi aikaisemmasta kaverilistan lataamisesta saatu ja tallennettu tieto poistetaan ja aloitetaan siis taas tyhjästä. Yleensä näin halutaan tehdä, koska funktion tarkoitus on aloittaa kaverilistan tutkiminen alusta. LoadMoreFriends sen sijaan jatkaa jo olemassa olevaa kaverilistan lataamisprosessia automaattisesti siitä, mihin se edellisestä LoadFriends- tai LoadMoreFriends-funktion kutsusta jäi. Se ei siis vaadi muuta kuin edellä mainitun int-muuttujan ja callback-implementoinnin.

Callback-implementaatiossa on kolmeen eri tulokseen reaktiot kummassakin funktioista. Ensimmäinen lopputulos on Completed, eli koko kaverilista on saatu ladattua eikä siinä ole jäljellä kavereita ladattavaksi. Silloin voidaan jatkaa kavereiden käsittelyyn ilman huolia siitä, jäikö jotain kavereita puuttumaan. LoadMore taas tarkoittaa sitä, että kaverilistan jatkolataaminen on onnistunut, ja listalla on vielä lisää kavereita ladattavana. Tässä tapauksessa siis saatetaan haluta kutsua uusi LoadMoreFriends-funktio, tai käsitellä sillä hetkellä ladattuja kavereita jotenkin. Kolmas on yleinen epäonnistumisia varten luotu reaktio, joka kutsuu aikaisemminkin mainitun CheckFailureReason-funktion. Tutkitaan sen implementaatiota seuraavaksi.

Kuvassa 22 CheckFailureReason-funktio käyttää funktiota GetLastLoadFriendsStatus, joka sisältää talletetun tiedon edellisen kaverilistan hakemisyrityksen lopputuloksesta. Se on tallennettu jo aikaisemminkin tutkitussa LoadFriendsStatus-tyyppisessä muuttujassa. Syitä on monia, kuten esimerkiksi yhteysongelma, ja funktiota voikin laajentaa tarvittaessa reagoimaan myös muihin epäonnistumisen syihin.

```
3 references
public void CheckFailureReason()
{
    // Fetch the failure reason from the latest attempt to load friends.
    LoadFriendsStatus status = PlayGamesPlatform.Instance.GetLastLoadFriendsStatus();

    // If a player's permission is required to access friends list, call the appropriate function for it.
    if(status == LoadFriendsStatus.ResolutionRequired) { AskForFriendAccess(); }

    // If need be, implement reactions to other LoadFriendsStatuses as well, other failure reasons are:
    if(status == LoadFriendsStatus.NotAuthorized) { }
    if (status == LoadFriendsStatus.InternalError) { }
    if (status == LoadFriendsStatus.NetworkError) { }
    if(status == LoadFriendsStatus.Unknown) { }
}
// Brings up the built-in UI element that requests a permission to access friends list.
2 references
public void AskForFriendAccess()
{
    PlayGamesPlatform.Instance.AskForLoadFriendsResolution((UIStatus status) =>
    {
        if (status == UIStatus.Valid) { Debug.Log("Now able to access friends list and load friends."); }
        else { Debug.Log("No permission to access friends has been granted."); }
    });
}
```

Kuva 22. Kaverien haun epäonnistumisen syyn tarkastelu ja kaverilistaan pääsyn luvan kysyminen.

Syitä on viisi. Ensimmäinen on NotAuthorized, joka tarkoittaa sitä, että kaverilistaa ei ole oikeus käsitellä. Tämä voi johtua esimerkiksi siitä, että pelaaja pääsi kirjautumaan ulos Google Play Games -palvelusta jossain välissä, joten sen ratkaisemiseksi voidaan kokeilla esimerkiksi pyytää uutta sisäänkirjautumista. InternalError on virhe itse palvelussa, ja sille pelaaja ei välttämättä voi tehdä mitään muuta kuin yrittää ladata kaverilista uudelleen. NetworkError johtuu useimmiten heikosta kommunikoinnista pelin ja palvelun välillä, minkä syynä voi olla esimerkiksi huono verkkoyhteys, mistä voidaan ilmoittaa pelaajalle. Unknown on tuntematon virhe, joka tapahtuu, jos mikään muista syistä ei käynyt toteen. Tällöinkään ei välttämättä ole tehtävissä muuta kuin ilmoittaa virheen tapahtuneen ja ehdottaa uudelleen yrittämistä. Esimerkissä ainoa implementoitu ratkaisu on ResolutionRequired, joka tarkoittaa sitä, että kaverilista löytyi, mutta sitä ei ole lupa käsitellä. Tässä tapauksessa kutsutaan esimerkkifunktio AskForFriendAccess, joka sisältää tämän luvan kysymisen implementoinnin.

Kyseinen funktio käyttää lisäosan funktiota AskForLoadFriendsResolution. Se tuo esiin Google Play Games -pelinsisäisen käyttöliittymän, joka pyytää lupaa käsitellä kaverilistaa. Funktio vaatii myös callback-implementaation UIStatus, josta tässä tapauksessa tarkistetaan vain, onko UIStatus Valid vai jokin muu. Sen ollessa Valid, on lupa kaverilistan tarkkailuun annettu, ja voidaan siis kutsua jompikumpi aikaisemmin tutkituista kaverilistan lataamisvaihtoehdoista. Jos se taas on jotain muuta, tarkoittaa se joka tapauksessa sitä, että lupaa käsitellä kaverilistaa ei ole, jolloin prosessi päättyy siihen. Pelissä saattaa olla tarpeen reagoida myös muutoin tilanteeseen.

4.10 Saavutukset

Google Play Games sisältää muitakin pelinsisäisiä käyttöliittymiä kuin sisäänkirjautumisen. Esimerkiksi saavutuksia voi tarkkailla pelinsisäisesti. Funktio ShowAchievementsUI tuo esiin kyseisen käyttöliittymän. Sitä voidaan käyttää helppoon toiminnallisuuteen sallimaan pelaajan tarkastella saavutuksiaan Google Play Gamesin tarjoaman käyttöliittymän kautta. Kuvassa 23 se on sijoitettu funktioon ShowAchievementsUI, joka voitaisiin yhdistää vaikka nappiin pelissä.

```
using UnityEngine;
using GooglePlayGames;

Unity Script | 0 references
public class GoogleAchievement : MonoBehaviour
{
    // Brings up the built-in Play Games achievements UI.
    0 references
    public void ShowAchievementsUI() { PlayGamesPlatform.Instance.ShowAchievementsUI(); }
    // Attempts to reveal an achievement of the given ID.
    0 references
    public void RevealAchievement(string ID)
    {
        PlayGamesPlatform.Instance.RevealAchievement(ID, (bool success) =>
        {
            if (success == true) { Debug.Log("Achievement " + ID + " was succesfully revealed."); }
            else { Debug.Log("Attempt to reveal achievement " + ID + " has failed."); }
        });
    }
    // Attempts to increment a multi-stage achievement of the given ID by given value.
    0 references
    public void IncrementAchievement(string ID, int value)
    {
        PlayGamesPlatform.Instance.IncrementAchievement(ID, value, (bool success) =>
        {
            if(success == true) { Debug.Log("Achievement " + ID + " was succesfully incremented by " + value); }
            else { Debug.Log("Attempt to increment achievement " + ID + " by " + value + " has failed."); }
        });
    }
    // Unlocks an achievement of the given ID. Also works for incrementable achievements, in which case unlocks them in full.
    0 references
    public void UnlockAchievement(string ID)
    {
        PlayGamesPlatform.Instance.UnlockAchievement(ID, (bool success) =>
        {
            if (success == true) { Debug.Log("Achievement " + ID + " was succesfully unlocked"); }
            else { Debug.Log("Attempt to unlock achievement " + ID + " has failed."); }
        });

    }
}
```

Kuva 23. Play Games -saavutuskäyttöliittymän esiin tuova koodi sekä saavutukset paljastavat ja antavat koodit.

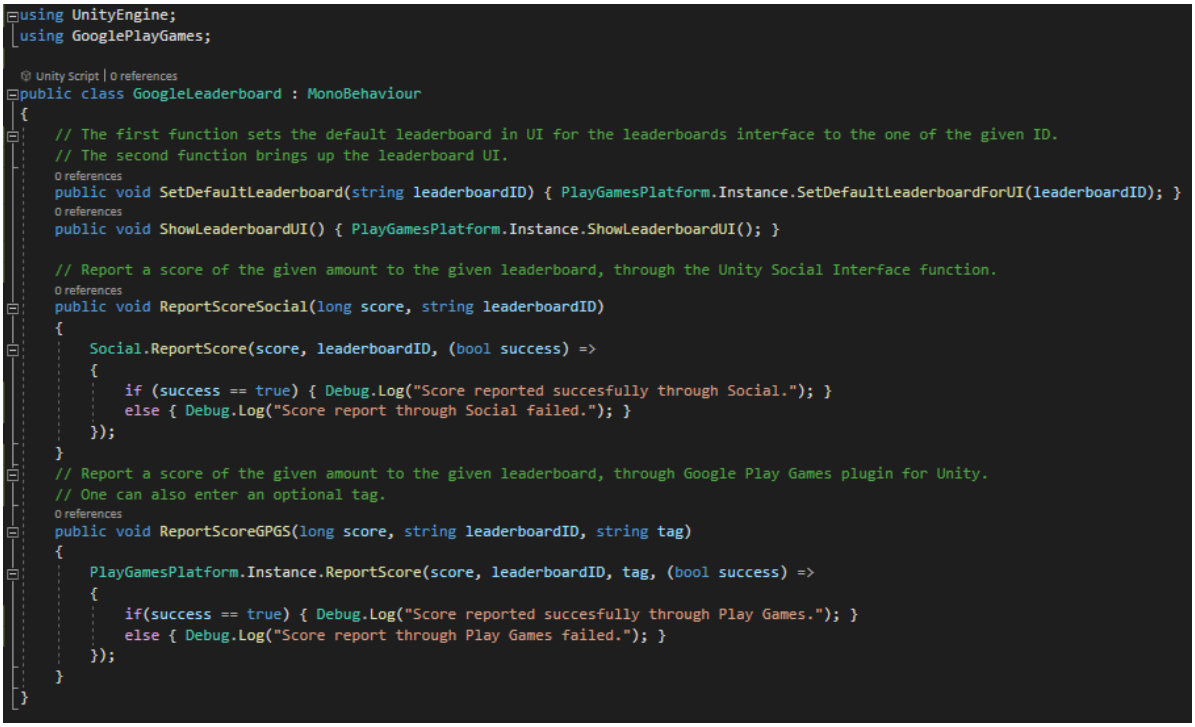
Tutkitaan seuraavaksi kolmea funktiota, joilla käsitellään saavutuksia. Ensimmäinen esitellyistä funktioista on RevealAchievement. Myöhemmin esiteltävät kaksi muuta funktiota eivät automaattisesti paljasta saavutusta, joka on määritelty piilotelluksi sitä konfiguroidessa Developer Consolessa. Sitä varten pitää käyttää tätä funktiota. Se vaatii kutsuessa syöttönä saavutuksen ID:n, ja siihen pitää myös toteuttaa callback-implementaatio, joka kertoo, onnistuiko operaatio vai ei. Tässä tapauksessa callback-implementaatio on vain debuggauskonsoliviesti, joka kertoo, onnistuiko funktio vai ei.

Seuraavana ovat IncrementAchievement ja UnlockAchievement. Kuten mainittu, nämä eivät tuo näkyviin sellaista saavutusta, mikä on piilotetuksi määritelty alun perin. IncrementAchievement pelkästään nostattaa sellaisen saavutuksen edistystä, missä on useampi askel, kun taas UnlockAchievement avaa sellaisen saavutuksen, jossa ei ole. UnlockAchievement-funktiota voidaan käyttää myös moniaskelisen saavutuksen avaamiseen välittömästi, jos se on tarpeen. Kumpikin vaatii saavutuksen ID:n, ja IncrementAchievement vaatii myös luvun, joka lisää sen määrän askeleiden suorituksia kyseiseen saavutukseen.

Kummassakin on myös niiden callback-funktioimplementaatio, joka on esimerkissä taas vain debuggauskonsoliviestikäytössä. Pelissä voitaisiin ohjelmoida paikallisesti tallennettu tieto siitä, mitkä saavutukset ovat jääneet paljastamatta tai antamatta niille oikeassa tilanteessa ja johonkin pelin vaiheeseen logiikka, mikä käy listan läpi ja yrittää myöhemmin antaa tai paljastaa kyseiset saavutukset uudelleen. Tilanne, jossa tällainen voisi olla tarpeen, on se, että pelaajalla katkeaa yhteys Google Play Games -palveluun kesken pelin tilanteessa, milloin saavutus pitäisi antaa tai jos pelaaja ei ole alun perinkään kirjautunut palveluun, ja halutaan tarjota mahdollisuus avata saavutukset takautuvasti, kun hän niin tekee. Jotkin saavutukset voivat mahdollisesti olla saatavissa vain tietyssä tilanteessa pelissä, jolloin niiden antamisen epäonnistuminen voi olla oleellinen ongelma ratkaistavaksi.

4.11 Pistetaulukot

Pistetaulukkojen tarkkailuun on myös olemassa lisäosassa funktiot. ShowLeaderBoardUI tuo esiin pelinsisäisen Google Play Games -pistetaulukkokäyttöliittymän, jonka kautta pelaaja voi tarkkailla pelin pistetaulukoita, samaan tapaan kuin aikaisemmissakin osioissa. Funktiota SetDefaultLeaderboardForUI, joka vaatii jonkin pistetaulukon ID:n, käyttäen voidaan määrittää, mikä on oletusarvoinen pistetaulukko, joka tulee ensinnä näkymiin pistetaulukkokäyttöliittymän avatessa. Jos pelissä on monta eri pistetaulukkoa ja halutaan pelaajan näkevän tietyn pistetaulukon ensin, voidaan kutsua ensin funktio, joka asettaa sen oletusarvoiseksi, ja sitten funktion, joka avaa käyttöliittymän.



```
using UnityEngine;
using GooglePlayGames;

Unity Script | 0 references
public class GoogleLeaderboard : MonoBehaviour
{
    // The first function sets the default leaderboard in UI for the leaderboards interface to the one of the given ID.
    // The second function brings up the leaderboard UI.
    0 references
    public void SetDefaultLeaderboard(string leaderboardID) { PlayGamesPlatform.Instance.SetDefaultLeaderboardForUI(leaderboardID); }
    0 references
    public void ShowLeaderboardUI() { PlayGamesPlatform.Instance.ShowLeaderboardUI(); }

    // Report a score of the given amount to the given leaderboard, through the Unity Social Interface function.
    0 references
    public void ReportScoreSocial(long score, string leaderboardID)
    {
        Social.ReportScore(score, leaderboardID, (bool success) =>
        {
            if (success == true) { Debug.Log("Score reported succesfully through Social."); }
            else { Debug.Log("Score report through Social failed."); }
        });
    }
    // Report a score of the given amount to the given leaderboard, through Google Play Games plugin for Unity.
    // One can also enter an optional tag.
    0 references
    public void ReportScoreGPGS(long score, string leaderboardID, string tag)
    {
        PlayGamesPlatform.Instance.ReportScore(score, leaderboardID, tag, (bool success) =>
        {
            if(success == true) { Debug.Log("Score reported succesfully through Play Games."); }
            else { Debug.Log("Score report through Play Games failed."); }
        });
    }
}
```

Kuva 24. Pistetaulukkokäyttöliittymän oletusarvoisen pistetaulukon asettava ja käyttöliittymän esiintuova sekä pistetaulukkoon pisteet lähettävä koodi.

Google Play Games -pistetaulukkoon voi lähettää pelisession aikana saadut pisteet kahdella eri tavalla. Voidaan käyttää joko Google Play Games plugin for Unityn omaa implementaatiota tai Unityn Social-rajapinnan ratkaisua. Ne toimivat muuten samalla periaatteella, mutta jos halutaan samalla raportoida jokin metadataan liittyvä tagi, tulee käyttää lisäosan vaihtoehtoa. Esimerkkejä siitä, mitä metadataa voidaan lähettää, on pisteisiin liittyvää kontekstuaalista tietoa pelistä, kuten se, missä pelimoodissa pistemäärä on saavutettu. Tagi ei kuitenkaan ole pakollinen osa Play Games -ratkaisun implementaatiota, joten sitä voidaan käyttää myös silloin, kun tageja ei ole käytössä, ja jättää se vain pois funktion kutsusta.

Kumpaankin pakollista on syöttää saavutettu pistemäärä, joka on oletusarvoisesti tyyppiä ”long”, mutta funktiot hyväksyvät myös tyypin ”int”. Kummatkin ovat tasalukuja, mutta long tukee pidempiä arvoja kuin int. Lisäksi tarvitaan pistetaulukon ID. Ohjelmoinnin puolella ei tarvitse huolehtia pisteiden vertailusta, vaan se tapahtuu automaattisesti, joten pelaajan pistemäärää ei voida vahingossa korvata matalammalla näitä funktioita käyttäen. Kumpikin vaatii myös boolean callback-implementaation, joka kertoo, onnistuiko pisteiden raportointi pistetaulukkoon. Esimerkkikoodissa se taas lähettää vain debuggauskonsoliin viestin, mutta saavutusosiossa mainittiinkin jo, mihin kyseistä ominaisuutta voi käyttää pelissä. Jos esimerkiksi pelaajan yhteys Play Games -palveluun katkeaa kesken pelisession tai jos hän ei ollut siihen vielä kirjautunut, voitaisiin

esimerkiksi pisteiden raportoimisen epäonnistuessa tallentaa se johonkin paikallisesti tilapäisesti. Jos pelaajalle muodostuu yhteys Play Games -palveluun myöhemmin, voidaan silloin lukea tallennetut epäonnistuneet tapahtumat ja yrittää niitä uudelleen.

5 Yhteenveto

Avauskappaleessa tutkittiin joitain tilastoja ja kyselyitä, jotka liittyivät sosiaalisiin ulottuvuuksiin ja moninpeliominaisuuksiin. Niin videopeliteollisuudessa yleisesti, kuin myös tarkemmin Android-alustalla, oli havaittavissa kyseisten ominaisuuksien merkitys. Moninpelit tyypillisesti myyvät paremmin ja saavuttavat suuremman suosion, ja nykypäivänä ihmiset usein sosialisoituvat laitteiden ja pelien välityksellä.

Moninpeliominaisuuksiin liittyen työn teoriaosuudessa tutustuttiin elementteihin, joista moninpelikokemus koostuu. Se, miten pelaajat saatetaan yhteen esimerkiksi puhtaassa offline-pelissä, eroaa suuresti verkkopeleistä, ja pelaajien määrä ja heidän vuorovaikutuksensa toisiinsa ja maailmaan ovat oleellisia asioita peliä suunnitellessa ja ohjelmoitaessa. Myös pelin ajoitukseen liittyvät asiat ovat tärkeitä tunnistaa, sillä reaaliaikainen peli yleisesti puhuen toimii hyvin eri tavalla kuin vuoropohjainen, ja pelin kulkua voidaan muokata vielä enemmän esimerkiksi toimintopistejärjestelmällä. Tällaiset asiat ymmärtämällä saadaan aikaiseksi perustavanlaatuinen ajatus siitä, miten monimuotoinen moninpelikokemus voi olla ja mitä sellaisten ominaisuuksien työstämiseen kuuluu.

Moninpeliteoriaosuuden jälkeen käytiin läpi yleisiä sosiaalisia ominaisuuksia peleissä. Kaveri- ja kiltaominaisuudet sallivat pelaajien muodostaa ihmissuhteita toistensa kanssa pelinsisäisesti, ja eri chat-ominaisuuksin sallitaan kommunikointi. Pistetaulukot, saavutukset ja tapahtumat voivat nostattaa pelaajien pysyvyyttä ja intoa pelin pelaamiseen rohkaisemalla seikkailuun, tutkimiseen ja muuhun pelin sisältöön osallistumiseen sekä myös muiden pelaajien kanssa kilpailuun. Yhteistyötä taas voidaan rohkaista tarjoamalla mahdollisuus esimerkiksi avun antamiseen ja muuhun lahjojen lähettämiseen. Sosiaalisilla ominaisuuksilla on usein positiivinen vaikutus pelin suosioon, ja eri sosiaalisten ominaisuuksien ennalta tunteminen voi auttaa pelin suunnittelussa ja ohjelmointityöhön valmistautumisessa.

Käytännön osuudessa tutustuttiin sekä moninpeli- että sosiaalisten ominaisuuksien ohjelmointiin Photon Unity Networking 2 (PUN 2):a ja Play Games plugin for Unitya käyttäen. Moninpeliominaisuuksien osuudessa saatiin pelaajat PUN 2:lla yhdistettyä toisiinsa ja tutkittiin, miten pelisessiota voi konfiguroida esimerkiksi rajaamalla sen pelaajamäärä. Lisäksi opittiin synkronoinnista ja sovellettiin sitä vielä vuorojärjestelmän ohjelmointiin. Lopuksi tutkailtiin vielä, miten voidaan reagoida ongelmiin pelaajan yhteydessä palvelimelle. Sosiaalisien ominaisuuksien osiossa taas opit-

tiin yhdistämään Google Play Games -palveluun ja ohjelmoitiin toiminnallisuutta palvelun eri ominaisuuksien kanssa. Työssä pyrittiin myös ehdottamaan, miten työtä voisi laajentaa jatkossa, mikä toimii suuntaa antavana mallina peliohjelmoijalle.

Kaiken kaikkiaan opinnäytetyössä tutustuttiin sosiaalisiin ulottuvuuksiin ja moninpeliominaisuuksiin, joita videopeleissä voi olla sekä sitä, millaista työ niiden parissa voi olla. Teoriaosuudessa rakennetiin pohja yleisestä perustiedosta, jonka omaaminen on hyödyksi etenkin videopelien suunnittelemisesta tai ohjelmoinnista kiinnostuneelle. Työn käytännön osuudessa tätä perustavanlaatuista ymmärrystä avarrettiin etenkin peliohjelmoijille. Näiden ominaisuuksien ymmärtäminen lienee ohjelmoijalle hyödyksi, sillä avauskappaleessa tarkkaillut tilastot kertovat moninpelien ja sosiaalisten ominaisuuksien nykyisestä ja todennäköisestä kasvavasta merkityksestä peliteollisuudessa ja mobiilialustalla.

Lähteet

1 2024 Unity Gaming Report. Unity. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://unity.com/resources/gaming-report?isGated=false

2 Rise of the Co-op Games. Video Game Insights. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://vginsights.com/insights/article/rise-of-the-co-op-games

3 sanket (käyttäjänimi). The Rise of Multiplayer Android Games: Key Trends Shaping the Future. SDLC Corp. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://sdlccorp.com/post/the-rise-of-multiplayer-android-games/

4 Wijman, T. Last looks: The global games market in 2023. Newzoo. [Internet]. [Viitattu 15.09.2024]. Saatavilla: https://newzoo.com/resources/blog/last-looks-the-global-games-market-in-2023#

5 Sherif, A. Market Share of mobile operating systems worldwide 2009-2024, by quarter. Statista. [Internet] [Viitattu 30.09.2024]. Saatavilla: http://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009

6 Play Protect Certified Android devices: safe and secure. Android. [Internet]. [Viitattu 30.09.2024]. Saatavilla: https://www.android.com/certified/

7 Find the Google Play Store app. Google Play Help. [Internet]. [Viitattu 30.09.2024]. Saatavilla: https://support.google.com/googleplay/answer/190860?hl=en

8 Rahman, M. There are nearly 16,000 Google Play Certified Android devices. XDA Developers. [Internet]. [Viitattu 30.09.2024]. Saatavilla: https://www.xda-developers.com/number-of-google-play-certified-android-devices/

9 Motta, M. Social gaming: When games become social networks for Gen Z and Millenials. Adjust. [Internet]. [Viitattu 01.10.2024]. Saatavilla: https://www.adjust.com/blog/social-gaming-gen-z/

10 Knezovic, A. Gen Alpha and Gen Z Gamers: How They Engage With Games. Udonis. [Internet]. [Viitattu 01.10.2024]. Saatavilla: https://www.blog.udonis.co/mobile-marketing/mobile-games/gen-alpha-gen-z-gamers

11 What is local co-op? All in! Games. [Internet]. [Viitattu 18.09.2024]. Saatavilla: https://www.allingames.com/what-is-local-co-op/

12 Jamal_Aladdin (käyttäjänimi). The Evolution of Multiplayer Gaming: A Journey Through Time. Medium. [Internet]. [Viitattu 01.10.2024]. Saatavilla: https://medium.com/@Jamal_Aladdin/the-evolution-of-multiplayer-gaming-a-journey-through-time-e34ef59294c2

13 McIntosh, J. What is a LAN Party and is it Still a Thing? ggParty. [Internet]. [Viitattu 18.09.2024]. Saatavilla: https://blog.ggcircuit.com/what-is-a-lan-party

14 Ray, M. online gaming. Britannica. [Internet]. [Viitattu 01.10.2024]. Saatavilla: https://www.britannica.com/technology/online-gaming

15 A short history of the internet. Science and Media Museum. [Internet]. [Viitattu 02.10.2024]. Saatavilla: https://www.scienceandmediamuseum.org.uk/objects-and-stories/short-history-internet#

16 Balasubramanian, K. The Rise of Online Multiplayer. Gameopedia. [Internet]. [Viitattu 18.09.2024]. Saatavilla: https://www.gameopedia.com/online-multiplayer-games/

17 Henningson, J. Ridsdale, J. From Esports fever to cheating… Counter-Strike: Global Offensive had it all. Red Bull. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.redbull.com/int-en/history-of-counterstrike

18 Schirch, B. Gaming Trends 2024's Latest Industry Developments. Bruno Scirch. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.brunoschirch.com/blog/gaming-trends-2024

19 Ldn-Post (käyttäjänimi). Online Multiplayer Games Revolutionize the Game Development Field: Multiplayer Experience Trends are Shaping the Future of Gaming Technology in 2024. London Post. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://london-post.co.uk/online-multiplayer-games-revolutionize-the-game-development-field-multiplayer-experience-trends-are-shaping-the-future-of-gaming-technology-in-2024/

20 Hollingsworth, D. Prepare To Level Up: 5 Trends To Dominate Gaming In 2024. Kontrol.gg. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://kontrol.gg/prepare-to-level-up-5-trends-to-dominate-gaming-in-2024/

21 Andy (käyttäjänimi). The Future of Gaming: Integrating Online and Offline Experiences. Orah. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://orah.co/the-future-of-gaming-integrating-online-and-offline-experiences/

22 Lark Editorial Team. Persistent World. Lark. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.larksuite.com/en_us/topics/gaming-glossary/persistent-world

23 Competitive Multiplayer Mobile Games: Synchronous vs. Asynchronous. Skillz. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.skillz.com/news/competitive-multiplayer-mobile-games-synchronous-vs-asynchronous/

24 Plizga, B. (Dis)Connected: The Joys of Asynchronous Play in Multiplayer Games. Medium. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://bradplizga.medium.com/dis-connected-the-joys-of-asynchronous-play-in-multiplayer-games-abe535c9cb1f

25 Sirlin, D. Asynchronous Games and Codex. Game Developer. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.gamedeveloper.com/design/asynchronous-games-and-codex

26 Raid: Shadow Legends. (2018) [videopeli]. Plarium. Kuvasta poistettu yksityiset yksilön tai ryhmän tiedot kuten nimet ja tägit.

27 Designing Turn-Based vs Real-Time Game Mechanics. Mahtgician Games. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://mahtgiciangames.com/blogs/the-creative-workshop-game-design-blueprints/designing-turn-based-vs-real-time-game-mechanics

28 Shafer, J. Turn-Based VS Real-Time. Game Developer. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.gamedeveloper.com/design/turn-based-vs-real-time

29 Hoekstra, K. What Is an RTS game? A Guide To Real-Time Strategy. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.historyhit.com/gaming/what-is-an-rts-game-a-guide-to-real-time-strategy/

30 Clash Royale. (2016). [videopeli]. Supercell. Kuvasta poistettu yksityiset yksilön tai ryhmän tiedot kuten nimet ja tägit.

31 Kvachev, V. Simultaneous Turns. rasie1's blog. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://kvachev.com/blog/posts/simultaneous-turns/

32 Hearthstone. (2014). [videopeli]. Blizzard Entertainment. Kuvasta poistettu yksityiset yksilön tai ryhmän tiedot kuten nimet ja tägit.

33 Bycer, J. A Deep Dive Into XCOM and XCOM 2. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.gamedeveloper.com/design/a-deep-dive-into-xcom-and-xcom-2

34 Clash Royale. (2016). [videopeli]. Supercell. Kuvasta poistettu yksityiset yksilön tai ryhmän tiedot kuten nimet ja tägit.

35 Casteel, B. Let's Talk About Combat Phases. Wayward Strategy. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://waywardstrategy.com/2017/06/12/lets-talk-about-combat-phases/

36 Lord, S. Video Game Genres: A Full List Of Gaming Genres (Updated). GamePro. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.gamepro.com.au/tutorials/video-game-genres/

37 What Is PvP In Gaming? (PvP). Dreams Quest. [Internet]. [Viitattu 19.10.2024]. Saataviilla: https://dreams.quest/post/what-is-pvp-in-gaming-pvp

38 What Is PvE In Gaming? (PvE). Dreams Quest. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://dreams.quest/post/what-is-pve-in-gaming-pve-dreams-quest

39 Fillery, J. 9 Best Games With PvPvE, Ranked. GameRant. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://gamerant.com/best-pvpve-games/

40 Unity Documentation. Welcome to Friends. Unity. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://docs.unity.com/ugs/en-us/manual/friends/manual/welcome

41 Microsoft Learn. People System (Friends List) overview. Microsoft. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://learn.microsoft.com/en-us/gaming/gdk/_content/gc/live/features/social/people-system/live-people-system-overview

42 Zalace, J. 8 Best Guild Systems in MMORPGs, Ranked. The Gamer. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.thegamer.com/best-guild-systems-mmorpgs-list/

43 Raid: Shadow Legends. (2018) [videopeli]. Plarium. Kuvasta poistettu yksityiset yksilön tai ryhmän tiedot kuten nimet ja tägit.

44 Rice, E. 5 Best Guild Systems in MMOs, Ranked. GameRant. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://gamerant.com/best-mmo-guild-systems/

45 In-game chat: Eight key features and how to deliver them. Ably. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://ably.com/blog/in-game-chat-features

46 Raid: Shadow Legends. (2018) [videopeli]. Plarium. Kuvasta poistettu yksityiset yksilön tai ryhmän tiedot kuten nimet ja tägit.

47 Griffin, J. Leaderboards – the original and best social feature. Game Developer. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.gamedeveloper.com/design/leaderboards---the-original-and-best-social-feature-

48 Clash of Clans. (2013). [videopeli]. Supercell. Kuvista poistettu yksityiset yksilön tai ryhmän tiedot kuten nimet ja tägit.

49 Hon, A. How Achievements took over the video game industry. Fast Company. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.fastcompany.com/90786023/how-achievements-took-over-the-video-game-industry

50 Saenz, C. The Art and Communities of Achievement Hunting. Side Quest. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://sidequest.zone/2024/06/10/the-art-and-communities-of-achievement-hunting/#google_vignette

51 Moran, M. How To Make In-Game Events Work For Your Game. Medium. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://medium.com/ironsource-levelup/how-to-make-in-game-events-work-for-your-game-3a78d234caeb

52 Clash of Clans. (2013). [videopeli]. Supercell. Kuvat 7, 8. Kuvista poistettu yksityiset yksilön tai ryhmän tiedot kuten nimet ja tägit.

53 Knezovic, A. Top 13 Social Features in Mobile Games with Examples. Udonis. [Internet]. [Viitattu 19.10.2024]. Saatavilla: https://www.blog.udonis.co/mobile-marketing/mobile-games/social-features-mobile-games

54 Unity 2021.3.17f1. [pelimoottori]. Unity Technologies. Saatavilla: https://unity.com

55 Photon Unity Networking 2. [lisäosa Unityyn]. Exit Games. Saatavilla: https://www.photonengine.com/pun

56 Google Play Games plugin for Unity. [lisäosa Unityyn]. Google. Saatavilla: https://github.com/playgameservices/play-games-plugin-for-unity

57 Kuvat 9-24. Itse tuotettu koodi.