KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Niko Sarkkinen

UNITYN KÄYTTÖLIITTYMÄLISÄOSIEN KÄYTTÖOHJEET JA VERTAILU

Opinnäytetyö
Marraskuu 2014

OPINNÄYTETYÖ
Marraskuu 2014
Tietojenkäsittelyn koulutusohjelma

Karjalankatu 3
80200 JOENSUU
013 260 600

Tekijä
Niko Sarkkinen

Nimeke
Unityn käyttöliittymälisäosien käyttöohjeet ja vertailu

Toimeksiantaja
Karelia-ammattikorkeakoulu

Tiivistelmä

Opinnäytetyö käsittelee Unityn käyttöliittymälisäosien ominaisuuksia. Unityn Asset Storessa on myynnissä suuri määrä erilaisia lisäosia, ja työn tarkoitus on helpottaa lisäosan valintaprosessia. Työn tavoite on tehdä Karelia-ammattikorkeakoululle käyttöohjeet käyttöliittymälisäosille ja vertailla lisäosien ominaisuuksia saadun kokemuksen perusteella.

Työssä käsitellään pelien käyttöliittymien suunnittelemista ja toteuttamista yleisellä tasolla. Työssä tarkastellaan pelien käyttöliittymien kehitystä, pelien ja hyötyohjelmien käyttöliittymien eroja, immersion eli peliin uppoutumisen merkitystä ja erilaisia tapoja toteuttaa käyttöliittymä. Lisäosista vertaillaan käyttöönoton ja opettelun helppoutta, valmiita käyttöliittymäelementtejä, työskentelyn sujuvuutta ja suoritustehokkuutta.

Käytännön osuudessa toteutettiin toimeksiantajalle käyttöohjeet kolmelle eri käyttöliittymälisäosalle. Toimeksiantaja tulee hyödyntämään ohjeita opetuskäytössä. Toteuttamisesta saatua käyttökokemusta hyödynnetään lisäosien ominaisuuksien vertailussa. Tarkemmassa vertailussa on lisäosista NGUI, Daikon Forge GUI Library ja iGUI. Vertailun tuloksena NGUI todetaan monipuolisimmaksi ja tehokkaimmaksi lisäosaksi. Daikon Forgella ja iGUI:lla on kuitenkin omat vahvuutensa erityisesti käytettävyydessä.

Kieli
suomi

Sivuja 73
Liitteet 2
Liitesivumäärä 135

Asiasanat
Unity, käyttöliittymä, käyttöliittymälisäosa

THESIS
November 2014
Degree Programme in Information Technology

Karjalankatu 3
80200 JOENSUU
FINLAND
013 260 600

Author
Niko Sarkkinen

Title
Unity User Interface Addons tutorials and comparisons

Commissioned by
Karelia University of Applied Sciences

Abstract

This thesis discusses the features of Unity's User Interface Addons. Unity Asset Store offers for sale a wide variety of addons and this thesis aims to help in the selection process for finding the best addon . The goal of this thesis is to provide Karelia University of Applied Sciences tutorials for creating user interfaces with the addons and to use the experience received from doing the tutorials to do comparisons bethween different addons.

The theoretical part discusses the design decissions and implementation of a video game user interface on a general level. Subject discusses the evolution of game user interfaces in video games, differences bethween user interfeces of games and utility softwares, the importance of immersion, and the variety of ways to implement the user interface. Addons will be compared by their ease of use, learning curve, built-in elements, fluidity of work flow and performance efficiency.

In the practical part of this thesis I will implement user interface with three selected UI-addons. Commissioner will use the tutorials in educational use. The experience gained from implementing the user interfaces will be utilized when comparing the features of each addon. The addons compared are NGUI, Daikon Forge GUI Library and iGUI. Results of comparison indicates NGUI is the most versatile and powerful GUI addon. Daikon Forge and iGUI have their strengths though especially in usability.

Language
Finnish

Pages 73
Appendices 2
Pages of Appendices 135

Keywords
Unity, User Interface, Addon

Sisältö

1 Johdanto

Tämän opinnäytetyön tavoitteena on selvittää, kuinka hyödyllisiä Unityn käyttöliittymälisäosat ovat peliprojektissa. Toimeksiantaja Karelia-ammattikorkeakoulu toivoi tehtäväksi vertailua ja käyttöohjeita muutamalle eri lisäosalle.

Työ keskittyy käyttöliittymälisäosiin uuden käyttäjän näkökulmasta. Käyttöohjeet tehdään käyttäjälle, joka osaa käyttää Unityä, mutta ei ole koskaan käyttänyt lisäosia. Tällainen käyttäjä on esimerkiksi todennäköisesti Karelia-ammattokorkeakoulun opiskelija, joka on juuri opetellut käyttämään Unityä. Myös lisäosien vertailussa on otettu huomioon, mitkä asiat uutta käyttäjää niissä voisi kiinnostaa. Yleensä tästä aiheesta löytyvissä vertailuissa on keskitytty joko yhden käyttäjän mielipiteeseen yhdestä hänen käyttämästään lisäosasta tai keskitytty useamman lisäosan teknisien ominaisuuksien listaamiseen. Tämä työ pyrkii olemaan hyödyllinen uudelle käyttäjälle, koska siinä vertaillaan useamman lisäosan käyttökokemusta.

Unityn Asset Storessa on ostettavissa suuri määrä erilaisia käyttöliittymälisäosia. Yleensä ne lupaavat helpot ja monipuoliset visuaaliset työkalut käyttöliittymän tekemiseen. Opinnäytetyön käytännön osuudessa tehdään käyttöliittymä muutamalla valikoidulla lisäosalla ja arvioidaan, kuinka paljon lisäosat helpottavat käyttöliittymän tekemistä. Käyttöliittymä toteutetaan NGUI:lla, Daikon Forge GUI Libraryllä ja iGUI:lla. Näiden lisäksi tarkastellaan myös EzGUI:n ja GUIMakerin ominaisuuksia.

Opinnäytetyössä tutkitaan käyttöliittymälisäosien ominaisuuksia. Tutkimuskysymyksiä ovat, miten eri käyttöliittymälisäosat eroavat toisistaan ja kuinka paljon niistä on hyötyä peliprojektissa. Eri lisäosia vertaillaan keskenään sekä yleisesti Unityn omaan GUI-luokkaan. Käyttöohjeiden tekemisestä saadaan lisäosista käyttökokemus uuden käyttäjän näkökulmasta. Lisäosien ominaisuuksia vertaillaan tämän käyttökokemuksen perusteella. Lisäosien

hyötyä peliprojektissa arvioidaan sillä, kuinka paljon lisäosat helpottavat käyttöliittymien tekemistä ja kuinka paljon niillä voi säästää ohjelmointikustannuksissa.

Aihe oli mielenkiintoinen, koska olin työharjoittelussa päässyt tekemään käyttöliittymän NGUI:lla, joka on yksi tähän opinnäytetyöhön valituista lisäosista. Silloin lisäosa oli valittu jo ennen kuin itse aloitin työskentelyn, joten muihin vastaaviin lisäosiin tutustuminen kiinnosti.

Tässä tutkielmaosuudessa käydään läpi pelien käyttöliittymän suunnittelua yleisesti. Siihen liittyviä asioita ovat käyttöliittymien kehitys, immersion merkitys sekä erilaiset tavat tehdä käyttöliittymiä. Luvussa 2 kerrotaan käyttöliittymien merkityksestä peleissä sekä niiden erityispiirteistä ja kehityksestä. Luvussa 3 käydään läpi esimerkkien avulla erilaisia tapoja tehdä pelien käyttöliittymä. Luvussa 4 kerrotaan Unityn GUI-luokan ominaisuuksista. Luvussa 5 käydään läpi eri käyttöliittymälisäosia sekä määritellään tarkemmin vertailtavat ominaisuudet. Luvussa 6 kerrotaan tutkimuskysymysten kannalta eri lisäosien käyttökokemuksesta. Luvussa 7 vertaillaan käyttökokemuksia eri lisäosien välillä. Luku 8 on opinnäytetyön pohdintaosuus.

2 Pelien käyttöliittymien erityispiirteitä

Tässä luvussa kerrotaan käyttöliittymän merkityksestä ja siitä, miksi se korostuu peleissä. Graafisen käyttöliittymän käsitettä tarkennetaan ja selvitetään sen eroa käyttöliittymään yleisesti. Luvussa käsitellään yleisellä tasolla sitä, mitä pelin käyttöliittymältä odotetaan ja mitä erityispirteitä sillä on hyötyohjelmien käyttöliittymiin verrattuna. Lopussa avataan immersion käsitettä.

2.1 Käyttöliittymän määritelmiä

Käyttöliittymä mahdollistaa ihmisen ja laitteen välisen kommunikaation. Käytettävä laite voi olla täysin tekninen tai elektroninen laite. Molemmissa tapauksissa käyttäjä tarvitsee rajapinnan laitteen käyttämiseen, ja sitä kutsutaan käyttöliittymäksi. (Tech Terms 2009.)

Tietotekniikassa tyypillinen käyttöliittymä on sekoitus ohjauslaitteita ja ruudulla näkyviä graafisia elementtejä. Erimerkiksi tyypillisen tietokoneohjelman käyttämiseen tarvitaan hiirtä ja näppäimistöä, ja laitteen ruudulla näkyy sen piirtämä graafinen käyttöliittymä. (Tech Terms 2009.)

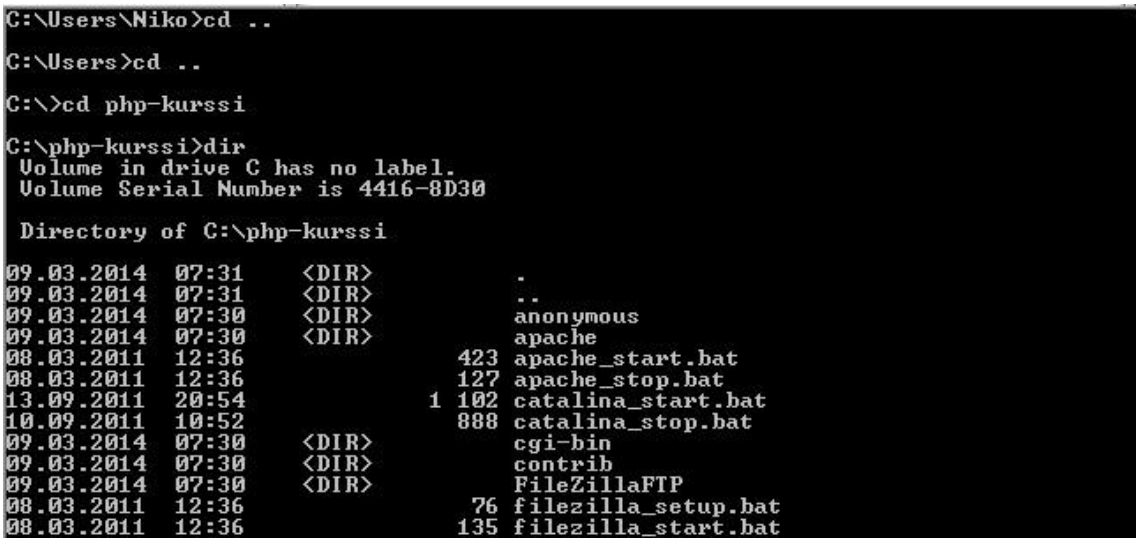
Käyttöliittymän välityksellä käyttäjä ohjaa ohjelmaa tai laitetta. Ohjelmisto voi sisältää esimerkiksi ikkunoita, kuvakkeita, valkoita ja painikkeita, jotka mahdollistavat käyttäjän vuorovaikuttaa ohjelman kanssa. Se tunnetaan myös graafisena käyttöliittymänä. Laitteen käyttöliittymä voi olla kauko-ohjain tai videopeliohjain. Sillä voidaan tarkoittaa myös videokameran, digitaalisen kameran tai iPodin ohjauslaitteita. Useimmat modernit käyttöliittymät on suunniteltu hyödyntämään sekä laitteistoa että ohjelmistoja. (PC.net 2009.)

Käyttöliittymä on ohjelmiston tai käyttöjärjestelmän visuaalinen osa. Se määrää kuinka käyttäjä käyttää tietokonetta tai ohjelmistoa, ja kuinka tieto esitetään näytöllä. Käyttöliittymän päätyypit ovat komentokieli: käyttäjän täytyy tuntea laite- ja ohjelmistokohtaiset komennot tai koodit, valikot: Käyttäjä valitsee komennon näytöllä olevasta listasta, graafinen käyttöliittymä: käyttäjä antaa komentoja valitsemalla ja klikkaamalla ruudulla näytettäviä kuvakkeita. (Business Dictionary 2014.)

2.2 Graafinen käyttöliittymä

Graafinen käyttöliittymä eli GUI (Graphical User Interface) on yksi tapa toteuttaa käyttöliittymä. Termi otettiin käyttöön erottamaan graafiset käyttöliittymät

tekstipohjaisista käyttöliittymistä. Ennen graafisia käyttöliittymiä käyttäjä kommunikoi tietokoneen kanssa tekstipohjaisilla komentotyökaluilla (Kuva 1). Sujuvan käytön edellytys oli syötettävien komentojen opetteleminen ulkoa, ja tietokoneen antaman palautteen tulkitseminen vaati vahvaa osaamista. (Rouse 2006.)

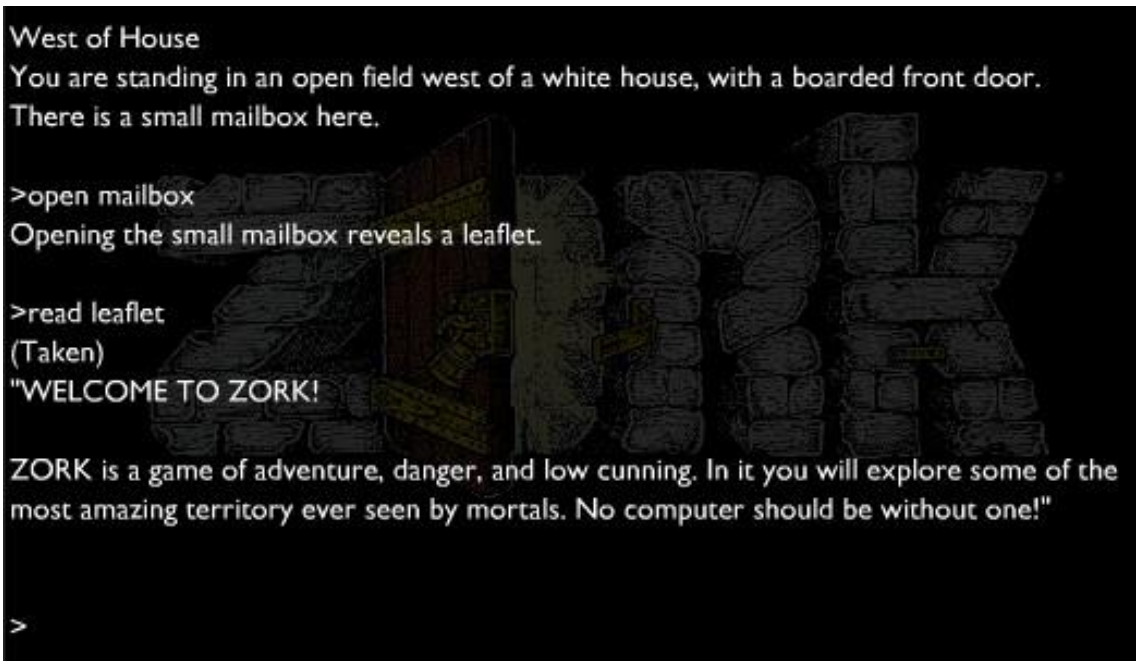


Kuva 1. Windows-käyttöjärjestelmän komentoikkuna (Kuva: Niko Sarkkinen).

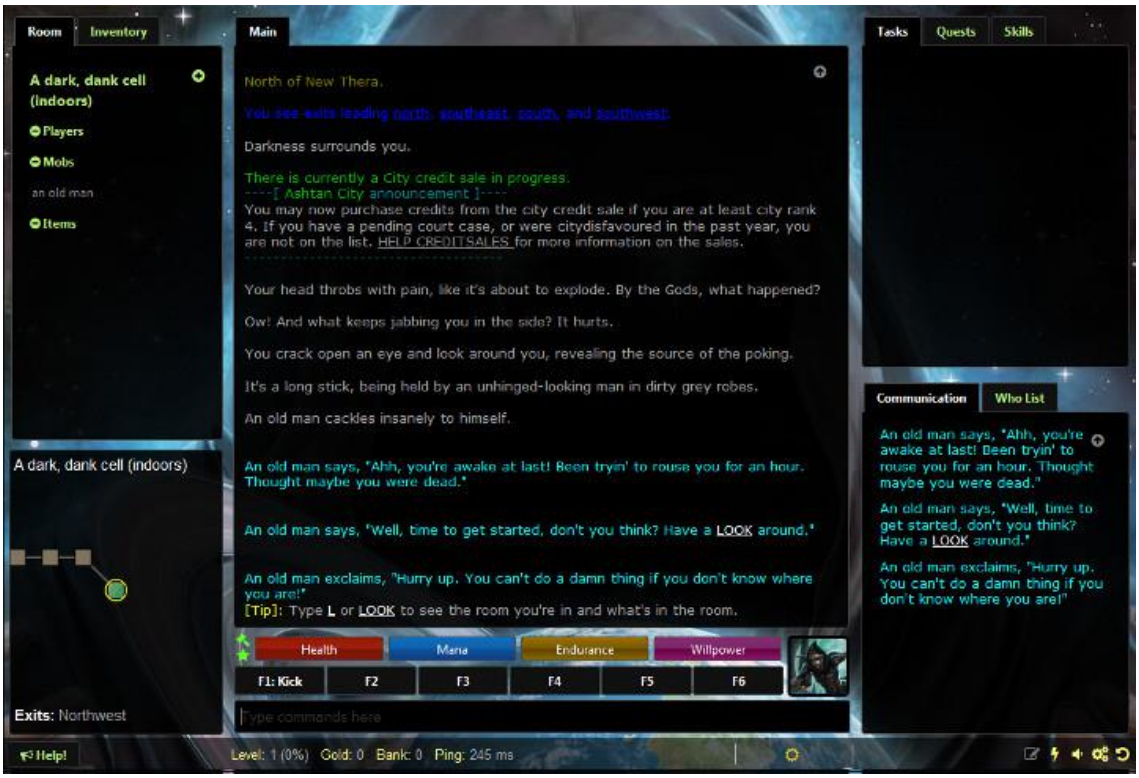
Graafinen käyttöliittymä koostuu useimmiten ikkunoista, valikoista, painikkeista, vierityspalkeista, kuvakkeista ja hiiren kursorista. Nykyään multimedian kasvava käyttö mahdollistaa liikkuvan kuvan ja äänen käytön graafisissa käyttöliittymissä. (Rouse 2006.)

Graafinen käyttöliittymä on ohjelmistoissa, jotka käyttävät graafisia elementtejä (tekstikenttiä, kuvakkeita, valikoita, vierityspalkkeja) tekstikomentojen sijasta. GUI elementtejä käytetään yleensä osoitinlaitteella kuten hiirellä, kynällä tai stylus-kynällä. Kaikilla graafista käyttöliittymää käyttävillä ohjelmistoilla on tapana käyttää yhteneviä graafisia elementtejä. Silloin käyttäjä osaa ne elementit opeteltuaan käyttää kaikkia ohjelmistoja, jotka käyttävät samantyylistä käyttöliittymää. Graafisen käyttöliittymän kehitti aluksi Xerox, ja sitä hyödynsi ja kehitti ensimmäisenä Apple. Kaikki modernit käyttöjärjestelmät ja ohjelmistot käyttävät nykyisin graafista käyttölittymää.

Pelien käyttöliittymä on lähes aina graafinen käyttöliittymä. Poikkeustapauksena voitaneen pitää tekstiseikkailupelejä ja Multi-User Dungeon- eli MUD-pelejä. Ne ovat vahvasti tekstipohjaisia pelejä, joissa peliä ohjataan komentoikkunan välityksellä. Tekstiseikkailut (Kuva 2) olivat yksi ensimmäisistä peligenreistä, ja ensimmäiset niistä julkaistiin 1970-luvulla. MUD-pelit (Kuva 3) ovat verkossa pelattavia tekstiseikkailuja, ja ne olivat suosionsa huipulla internetin alkuaikoina 1990-luvulla. Nykyään näitä pelejä pelaa enää hyvin vannoutunut pelaajakunta. (Business Dictionary 2014.)



Kuva 2. Zork on yksi tunnetuimmista tekstiseikkailupeleistä (Kuva: Niko Sarkkinen).



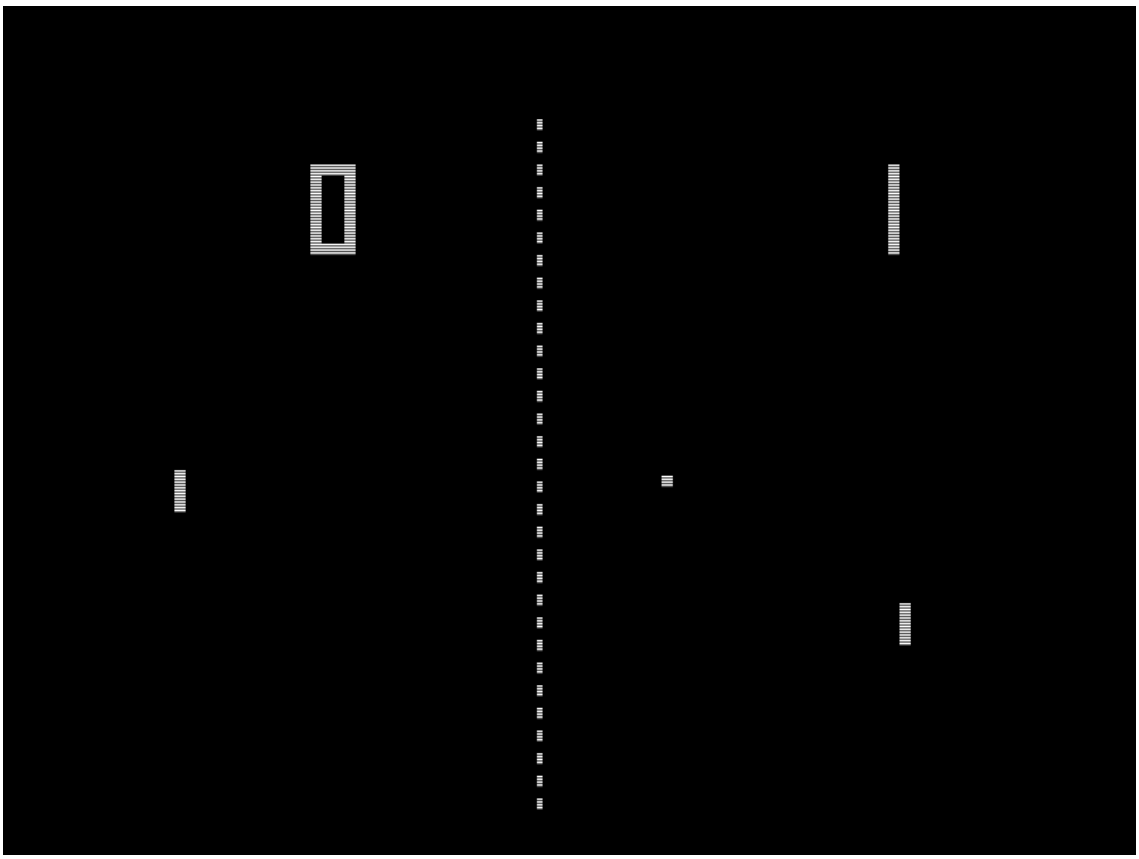
Kuva 3. Achea on suosituimpia moderneja MUD-pelejä (Kuva: Niko Sarkkinen).

2.3 Käyttöliittymä peleissä

Käyttöliittymä on työkalu, jolla pelissä viestitään pelaajalle tietoa pelistä. Sen avulla pelaaja voi myös vaikuttaa pelin tapahtumiin. Käyttöliittymä antaa pelaajalle tietoa mahdollisista valinnoista ja toimenpiteistä ja mahdollistaa niiden tekemisen. Samalla pelaajan tekemät toimenpiteet näkyvät hänelle vastaanottimen ruudulla. Pelissä mahdollisesti ohjattavan pelihahmon ei yleisesti oleteta olevan tietoinen käyttöliittymän avulla välitettävästä tiedosta. Tästä on poikkeuksena jotkin pelimaailmaan upotetut käyttöliittymäelementit, joista kerrotaan tarkemmin luvussa 3. Käyttöliittymän toimintaan ja ulkonäköön vaikuttaa laite, jolla pelaaja ohjaa peliä.

Käyttöliittymän avulla pelaajalle viestitään paljon tietoa pelissä tapahtuvista asioista, ja sen merkitys peleissä on ollut aina tärkeä. Aivan ensimmäisissä peleissä käyttöliittymät olivat pelkistettyjä, ja ne välittivät pelaajalle vain kaikkein välttämättömimmän tiedon pelin tapahtumista. Esimerkki tästä on vuonna 1972

julkaistu Pong-tennispeli, joka on ensimmäinen suuren suosion saavuttanut videopeli (Kuva 4). Pelien kehittyessä ovat käyttöliittymätkin kehittyneet, ja uusimmat teknologiat mahdollistavat pelaajan koko kehon hyödyntämisen pelien ohjaamisessa. Uusimmissa pelikonsoleissa on mahdollista tunnistaa pelaajan liikkeet ilman, että pelaaja on fyysisesti kosketuksissa minkään ohjaimen kanssa. (Poh 2014.)



Kuva 4. Pong-peli näyttää pelaajien pistemäärän.

Yksinkertaisimmissakin peleissä täytyy pelaajalle näyttää jotakin perustietoa siitä, miten peli sujuu. Yleisiä näytettäviä tietoja ovat esimerkiksi jäljellä olevien elämien määrä tai pistetilanne. Kuvassa 5 on tyypillinen näkymä 1990-luvun pelien tavasta esittää pelaajalle tietoa pelitilanteesta. Siinä kaikki GUI-elementit on sijoitettu ruudun alalaitaan. Vasemmassa reunassa näkyvät pelaajan nykyiset pisteet ja niiden alapuolella kentän ennätyspisteet. Keskellä on palkit elämäpisteille ja erikoisvoimalle. Palkkien ja jäljellä olevien elämien määrää osoittavavan elementin välissä oleva tyhjä alue on vihje pelaajalle siitä, että

palkkien enimmäiskokoa voi nostaa. Pelissä löytyykin esineitä, joilla pelihahmo tulee vahvemmaksi, ja se näkyy palkkien kasvamisena. Bucky O'Hare on julkaistu NES-konsolille vuonna 1992.

Kuva 5. Ruudunkaappaus Bucky O'Hare pelistä (Kuva: Niko Sarkkinen).

Näytettävän tiedon määrä sekä visuaalinen tyyli vaativat suunnittelua sekä artistilta että tekniseltä toteuttajalta. Näytettävän tiedon tyyppi ja määrä täytyy olla hyvin perusteltua ja tarkoituksenmukaista. Liiallinen tiedon määrä helposti hämmentää pelaajan ja olennainen tieto sekoittuu toisarvoisen tiedon kanssa. Visuaalisen suunnittelun tärkeimpiä tehtäviä on tehdä olennaisesta tiedosta erottuvaa ja selkeätä.

2.4 Immersio

Peleissä on yleensä tarkoitus saada pelaaja uppoutumaan ja eläytymään pelimaailmaan. Tätä kutsutaan immersioksi. Kun pelaaja tuntee olevansa läsnä pelissä ja unohtaa ympäröivän maailman, on hän immersoitunut peliin. Immersoitumisen tunnusmerkkejä ovat esimerkiksi ajantajun heikkeneminen, ympäröivien ihmisten ja tapahtumien havainnointikyvyn heikkeneminen, pulssin

kiihtyminen pelottavissa tai jännittävissä kohdissa, ja empatian tunteminen pelihahmoja kohtaan (Stuart 2010).

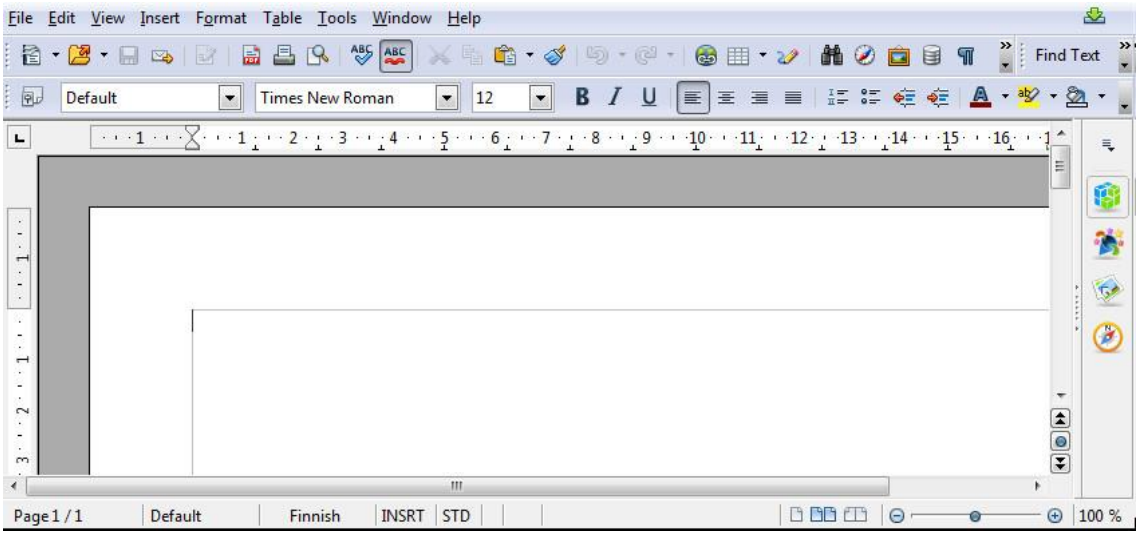
Yksi immersion suurimmista rikkojista voi olla huono käyttöliittymä. Toisaalta pelien käyttöliittymää voidaan hyödyntää immersion luomisessa. Vaikka pelaajat usein pitävät minimalistista käyttöliittymää esteettisesti viehättävänä ja eleganttina, pitävät he tärkeämpänä, että olennainen ja tarkoituksenmukainen informaatio on esillä ja käytettävissä. Tarkoituksenmukainen informaatio on kuitenkin riippuvainen pelaajasta, ja liiallinen tiedon näyttäminen voi ärsyttää pelaajaa. (Jørgensen & Llanos 2011, 10.)

Immersion merkitys pelien suunnittelussa on kasvanut. Pelaaja halutaan saada uskomaan, että pelimaailma on aidontuntuinen. Immersion luomisessa hyödynnetään entistä enemmän kaikkia aisteja (Madigan 2010). Käyttöliittymässä voidaan hyödyntä erityisesti tuntoaistia, ja monet innovatiiviset ohjainratkaisut, kuten Steam Controller, tähtäävät juuri tähän (Valve 2014).

Perinteinen tapa, jossa käyttöliittymäelementit ovat kuin liimattuna kameraan, on muuttunut, kun pelien kehittäjät ovat keksineet uusia tapoja tuoda pelaajalle sama tieto. Käyttöliittymäelementtejä voidaan upottaa pelimaailmaan, esimerkiksi pelihahmon kelloon tai kypärän visiiriin. Näistä tavoista kerrotaan tarkemmin luvussa 3.

2.5 Käyttöliittymän erot peleissä ja hyötyohjelmissa

Hyötyhjelmien käyttöliittymät eroavat pelien käyttöliittymistä hyvin olennaisesti. Hyötyohjelmissa on tärkeätä näyttää tieto mahdollisimman selkeästi ja jäsennetysti. Peleissä hyvä käyttöliittymä ei saa estää pelaajan eläytymistä pelimaailmaan. Immersion saavuttamiseksi peleissä onkin sallittua, ja joskus jopa pakollista, jättää jotain tietoa näyttämättä. Tämä ei yleensä ole tehokas lähestymistapa hyötyohjelmien käyttöliittymää suunnitellessa.



Kuva 6. OpenOffice-ohjelmistossa kaikki työkalut on sijoitettu yläreunaan niiden löytämisen helpottamiseksi (Kuva: Niko Sarkkinen).

Peleissä lähimpänä hyötyohjelmien käyttöliittymää ovat strategia- ja taktiikkapelit. Niissä on yleensä paljon eri asioita, joita pelaajan täytyy hallita. Esimerkiksi Civilization 5:ssä täytyy pelaajan valikkorakenteiden kautta pystyä mm. komentamaan sotilasyksiköitä, hallitsemaan tuotantoa ja taloutta, hoitamaan diplomaattisia suhteita ja kaupankäyntiä muiden valtioiden kanssa sekä tekemään päätöksiä valtionuskontoon liittyvistä asioista (Kuva 7). Toinen vastaava esimerkki on Football Manager 14, jossa pelaajan tehtävä on valmentaa jalkapalloseura menestykseen hoitamalla mm. sen talouden, huoltohenkilöstön palkkaamisen, joukkueen rakentamisen ja taktiikoiden tekemisenkin. Siinä pelaaja saa etusivulla nopean yleiskuvan seuran tilanteesta, mutta tarkempi tieto sekä mahdollisuus vaikuttaa asioihin avautuu valikoiden kautta (Kuva 8).

Kuva 7. Civilization 5 -pelin valikkoja (Kuva: Niko Sarkkinen).

Kuva 8. Foorball Manager 14 -pelissä seuran etusivulla näkee tietoa sen hetkisestä tilanteesta (Kuva: Niko Sarkkinen).

3 Erilaisia tapoja tehdä käyttöliittymä

Peleissä voidaan käyttöliittymä toteuttaa monella eri tavalla. Erik Fagerholt ja Magnus Lorentzon ovat luokitelleet tutkielmassaan *Beyond the HUD -- User Interfaces for Increased Player Immersion in FPS Games* käyttöliittymät neljään eri kategoriaan. (Fagerholt & Lorentzon 2009, 46–52.)

3.1 Non-diegetic eli upottamattomat elementit

Non-diegetic on perinteinen tapa tehdä UI (User Interface eli käyttöliittymä), jossa elementtejä ei ole upotettu pelimaailmaan. Elementit ovat 2D-objekteja, jotka ovat kohtisuorassa kameraan nähden. (Fagerholt & Lorentzon 2009, 51.)

Kuvassa 9 on erimerkkejä useasta upottamattomasta elementistä. Vasemmassa alanurkassa on hyvin perinteinen elämäpalkki, ja sen vieressä Hitman-peleistä tuttu hiippailumittari. Vasemmassa ylänurkassa on ryhmitelty eri toimintopainikkeita, joilla voi esimerkiksi avata oven tai poimia tavaroita. Oikeassa ylänurkassa on kompassi suunnan löytämisen helpottamiseksi.

Kuva 9. Hitman: Blood Money näyttää pelaajalle tietoa upottamattomilla elementeillä (Kuva: Niko Sarkkinen).

Upottamattomien elementtien etuna on se, että ne eivät ole sidottuna pelimaailman tapahtumiin tai maastoon. Niiden ulkonäkö voidaan tehdä juuri halutunlaiseksi, vaikkakin ne yleensä tehdään pelin yleisen ulkonäkösuunnitelman mukaisiksi. Niiden käyttäminen on suositeltavaa kun pelimaailmaan upotettujen elementtien käyttäminen rikkoo pelin saumattomuuden. (Stonehouse 2014.)

Upottamattomilla elementeillä esitetyn tiedon pelaaja näkee koko ajan. Sen takia erittäin suuri osa pelaajalle annettavasta tiedosta esitetään juuri tässä upottamattomassa muodossa. Perinteiset UI-elementit ovat suuressa suosiossa varsinkin MMO-, FPS- ja strategiapeleissä, joissa tiedon on oltava helposti saatavilla.

Upottamattomilla elementeillä toteutettu käyttöliittymä on tehokas, jos pelaajalle halutaan näyttää paljon tietoa selkeässä muodossa. Tällöin kuitenkin helposti kärsii immersiivisyys. Upotettujen elementtien lisäksi voidaankin käyttää

elementtejä, joiden voidaan ajatella kuuluvan pelimaailmaan.

3.2 Diegetic eli upotetut elementit

Pelimaailmaan voidaan upottaa UI-elementtejä. Tällöin niitä ei ole ankkuroitu kameraan, vaan ne ovat kiinni jossain pelimaailman esineessä. Pelihahmo ikään kuin katsoo näitä elementtejä omilla silmillään. Esimerkiksi pelihahmolla voi olla rannekello, josta näkee jäljellä olevan elämän. (Fagerholt & Lorentzon 2009, 52.)

Kuva 10. Metro 2033 -pelissä rannekellosta näkee tietoa säteilyn määrästä (Kuva: Niko Sarkkinen).

Perinteiset upottamattomat elementit voidaan laskea upotetuiksi elementeiksi, jos pelin asetelma sen mahdollistaa. Futuristisessa pelissä voidaan selitykseksi antaa, että pelihahmolla on silmäimplantit. Nämä implantit voivat antaa pelaajalle tietoa jäljellä olevasta elämästä, ammusten lukumäärästä tai vaikkapa pelaajan sijainnista (Kuva 11). (Fagerholt & Lorentzon 2009, 52.)

Kuva 11. Deus Ex: Human Revolution -pelissä pelihahmolla on silmäimplantit, jotka antavat tietoa pelitilanteesta ja ympäristöstä (Kuva: Niko Sarkkinen).

Upotetut elementit ohjaavat pelaajaa pelimaailmaan sijoitetuilla vihjeillä ja informaatiolla ilman, että pelaajan huomio irtautuu pelimaailmasta. Pelihahmo ja muut hahmot pelissä ovat tietoisia näistä asioista, ja se tekee kokemuksesta immersiivisemmän ja elokuvamaisen. (Russell 2011.)

3.3 Meta-elementit

Meta-elementit ovat sekoitus kahta edellistä tapaa tehdä käyttöliittymä. Siinä elementit on usein ankkuroitu kameraan, mutta niiden alkuperän voidaan ajatella olevan pelimaailmasta. Esimerkki tästä on ruudun muuttuminen punaiseksi tai jopa veriseksi, kun pelaaja vahingoittuu (Kuva 10). Veri on peräisin pelimaailmasta tai pelaajasta, mutta efekti on toteutettu perinteisellä kameraefektillä. (Fagerholt & Lorentzon 2009, 52.)

Kuva 12. Call of Duty 2 -pelissä ei ole UI-elementtiä elämän määrälle, vaan punainen näkökenttä antaa tietoa terveystilasta (Kuva: Niko Sarkkinen).

Toinen esimerkki on jokin esine, jonka käyttäminen tuo esille perinteisen 2D-käyttöliittymän, vaikka pelihahmo näennäisesti käyttääkin jotain pelimaailman laitetta (Fagerholt & Lorentzon 2009, 52).

Kuva 13. Fallout 3 -pelissä pelihahmolla on ranteessa pienoistietokone (Kuva: Niko Sarkkinen).

Meta-elementtien tavoite on antaa pelille enemmän realistisuuden tunnetta.

Pelaajalle pyritään antamaan tunne, että hän on vuorovaikutuksessa pelimaailman kanssa. Meta-elementit myös vaikuttavat pelaamiseen peittämällä osan pelinäkymästä. (Russell 2011.)

3.4 Spatiaaliset elementit

Spatiaaliset UI-elementit ovat pelimaailmaan sijoitettuja graafisia opasteita. Esimerkiksi maahan merkitty reitti, jota pelaajan tulee seurata, on spatiaalinen. Nämä opasteet lasketaan UI-elementeiksi, koska pelihahmon ei ajatella olevan tietoinen näistä opasteista. Esimerkiksi liikennemerkki ei ole UI-elementti, vaikka se antaakin pelaajalle tietoa, koska pelihahmon voidaan olettaa olevan tietoinen siitä. (Fagerholt & Lorentzon 2009, 46–52.)

Kuva 14. Dungeons & Dragons Neverwinter opastaa pelaajaa oikeaan suuntaan maahan ilmestyvällä säihkyvällä viivalla (Kuva: Niko Sarkkinen).

Spatiaalisia elementtejä käytetään antamaan pelaajalle tietoa, mistä pelihahmo ei ole tietoinen. Tiedon on oltava tärkeää ja tarkkaan harkittua, koska se rikkoo pelimaailman yhteinäisyyden. Oikein käytettynä ne voivat kuitenkin auttaa immersion luomisessa, koska pelaajan ei tarvitse selata valikkorakenteita tärkeän tiedon löytämiseksi. (Stonehouse 2014.)

4 Unityn GUI-luokka

Unityn mukana tulee sisäänrakennettu GUI-luokka. Se mahdollistaa yksinkertaisten käyttöliittymäelementtien tekemisen, mutta sillä on omat rajoitteensa. Tässä luvussa käydään läpi Unityn GUI-luokan yleisiä ominaisuuksia sekä sen vahvuuksia ja heikkouksia.

4.1 Unity

Unity on erittäin suosittu pelinkehitysohjelmisto 45 %:n markkinaosuudella (Unity Technologies 2014). Se on monialustainen kehitysympäristö, ja sillä voidaankin kehittää pelejä kaikille suosituimmille alustoille kuten PC:lle, konsoleille ja mobiililaitteille.

Unityn suosiolle voi löytää monta syytä, mutta suurimpia syitä ovat varmaankin sen ilmaisuus ja helppokäyttöisyys. Sillä voi tehdä joustavasti monenlaisia 2D- ja 3D-pelejä. Siihen on myös saatavilla paljon lisäosia ja laajennoksia, mikä mahdollistaa Unityn räätälöinnin juuri omalle peliprojektille sopivaksi. (Polsinelli 2013.)

Unityä käytetään ja opetetaan myös tämän opinnäytetyön toimeksiantajan Karelia-ammattikorkeakoulun opetussuunnitelmassa. Siitä on osoituksena suuri määrä siellä tehtyjä Unityyn liittyviä opinnäytetöitä. Vuonna 2014 on tehty Unityä hyödyntämällä esimerkiksi sellaisia opinnäytetöitä kuin Roolipelin kehittäminen Unity 3D:llä (Eronen 2014), Videopelit ja käytettävyys (Nevalainen 2014) ja Microtransactions in an Android Game (Kokkonen 2014). Näissä kaikissa on tehty peli Unityllä ja siitä on tutkittu aiheen mukaista osa-aluetta. Theseus-julkaisuarkiston mukaan näiden lisäksi Unityyn liittyviä opinnäytetöitä on tehty useita kymmeniä.

4.2 Yleistä

Unityn oma GUI-luokka mahdollistaa käyttöliittymän tekemisen skriptaamalla. Siinä ei kuitenkaan ole visuaalista työkalua, joka helpottaisi käyttöliittymän tekemistä. (Van Oosten 2013.) Se voi muodostua ongelmaksi, jos käyttäjällä ei ole skriptaamisosaamista.

Unityn GUI-luokka piirtää käyttöliittymän elementit OnGUI-metodissa. Se kutsutaan jokaisella framella aivan kuten Update-metodikin. Kaikki GUI-renderöinti täytyy tapahtua joko OnGUI-metodin sisällä tai funktioissa, joita OnGUI-metodi kutsuu. (Van Oosten 2013.) Unityn sisäänrakennettu GUI-luokka onkin todella helppo ottaa käyttöön. Sen yksinkertainen perusidea mahdollistaa käyttöliittymän tekemisen aloittamisen nopeasti.

4.3 Esimerkki Unityn GUI-luokasta

Tutkielman käytännön osuuden pohjana on peli, jossa alkuvalikossa pystyy tekemään pelipalvelimen tai liittymään valmiiseen peliin. Alkuvalikko on siinä toteutettu Unityn GUI-luokalla. Kuvassa 11 näkyy pelin alkuvalikko. Se on hyvin tyypillinen esimerkki Unityn GUI-luokalla tehdystä graafisesta käyttöliittymästä. Siinä erottuvat selkeästi eri elementit, kuten Window, Label, TextField, Slider ja Button. Pelipalvelimen luonti on myös toiminnallisuudeltaan hyvin yksinkertainen. Siinä kerätään käyttäjän syöttämät tiedot, jotka lähetetään nappia painamalla koodille käsiteltäväksi. Näinkin yksinkertainen ikkuna vaatii kuitenkin jo aika paljon koodaamista.

Kuvan 15 mukaisessa yksinkertaisessakin graafisessa käyttöliittymässä on jo merkittävä määrä käyttöliittymäelementtejä. Jokaiselle niistä täytyy erikseen koodissa määrittää sijaintiin ja ulkonäköön vaikuttavat arvot. GUI-luokassa sijainnin määrittelyä voi helpottaa jonkin verran hyödyntämällä VerticalLayout- ja HorizontalLayout-ominaisuutta (Listaus 1).

Kuva 15. Esimerkki. Unityn GUI-luokalla toteutettu alkuvalikko (Kuva: Niko Sarkkinen).

```
void Start()
{
    connectRect = new Rect(Screen.width/2-220, Screen.height/3, 160, 150);
    hostRect    = new Rect(Screen.width/2-60+10, Screen.height/3, 220, 150);
}

void OnGUI()
{
    GUILayout.BeginVertical();
    GUILayout.BeginHorizontal();
    GUILayout.FlexibleSpace();
    if ( GUILayout.Button("Poll servers"))
    {
        UpdateServerList();
    }
    connectRect = GUILayout.Window(0, connectRect, HandleListServers, "Join an online game");
    GUILayout.EndHorizontal();

    GUILayout.BeginHorizontal();
    GUILayout.FlexibleSpace();
    hostRect = GUILayout.Window(1, hostRect, HandleHostServer, "Host a game");
    GUILayout.EndHorizontal();
    GUILayout.EndVertical();
}
```

Listaus 1. Elementtien sijoitteluun käytettävää koodia.

Pelipalvelimien listaamiselle ja luomiselle tarkoitettujen ikkunoiden sijainti määritellään listauksen 1 koodissa. Ikkunoiden tarkempi määrittely tehdään erillisissä metodeissa (Listaus 2 ja 3).

Listauksen 2 koodissa hyödynnetään HorizontalSlider-elementtiä pelaajien halutun enimmäismäärän lukemiselle. Label-elementtiä käytetään monessa kohtaa kertomaan pelaajalle, mikä eri elementtien tehtävä on. Button-elementtiä painamalla kutsutaan palvelimen käynnistävää metodia.

```
void HandleHostServer(int windowID)
{
    // slider with min and max values
    GUILayout.BeginHorizontal();
    GUILayout.Label("Max players: ");
    numPlayers = GUILayout.HorizontalSlider(numPlayers, 2.0f, 16.0f,GUILayout.Width(50.0f));

    // floor selected value
    numPlayers = Mathf.Floor(numPlayers);
    GUILayout.Label(""+numPlayers);
    GUILayout.EndHorizontal();

    GUILayout.BeginHorizontal();
    GUILayout.Label("Your name:");

    string tmp = GUILayout.TextField(playerName, GUILayout.Width(105.0f));

    if ( tmp.Length <= 15 ) playerName = tmp;

    // truncate name to specific length
    if ( playerName.Length > 15 )
    {
        playerName = playerName.Substring(0,15);
    }

    GUILayout.EndHorizontal();
    if ( GUILayout.Button("Start game") )
    {
        gameIsOn = true;
        LaunchServer();
        Application.LoadLevel("01_BattleScene");
    }
}
```

Listaus 2. Peliserverin luonti-ikkunan koodia.

Palvelimien listauksessa hyödynnetään SelectionGrid-elementtiä. Listauksessa 3 näkyy, kuinka palvelinlista annetaan parametrina SelectionGridille. Kun pelaaja valitsee jonkin palvelimista, vaihtuu valittu index-arvo, ja peli yhdistyy kyseiselle palvelimelle.

```
void HandleListServers(int windowID)
{
   // Update array so things can be displayed.
    string[] serverArray = new string[serverList.Count];
    int i = 0;
    foreach( ServerEntry s in serverList )
    {
        serverArray[i++] = s.ToString();
    }
    // display array
    int selection = GUILayout.SelectionGrid(-1,serverArray,1);

    if ( selection != -1 )
    {
        string address = serverList.ToArray()[selection].address;
        msgQueue.Enqueue("Selected " + selection + ", addr: " + address);
        NetworkConnectionError err = Network.Connect( address, GAME_PORT);
        switch( err )
        {
        case NetworkConnectionError.NoError:
            msgQueue.Enqueue("Waiting for reply...");
            break;
        default:
            msgQueue.Enqueue("Could not connect: " + err );
            break;
        }
    }
}
```

Listaus 3. Peliservereiden listaamiseen ja serverille liittymiseen tarvittavaa koodia.

4.4 Vahvuuksia

Unityn GUI-luokka ei vaadi mitään lisäosia tai ennakkovalmisteluja projektin asetuksiin. Se sisältää paljon ominaisuuksia hyvän ja monipuolisen käyttöliittymän tekemiseen. Näihin ominaisuuksiin kuuluu esimerkiksi funktiot yleisimpien käyttöliittymäelementtien luomiseen. Aiemmassa esimerkissä oli käytetty neljää eri elementtiä, mutta niitä on todella monta muutakin. Elementtien lisäksi GUI-luokassa on paljon apufunktioita, jotka tekevät käyttöliittymän piirtämisestä helpompaa. Niillä funktioilla voi esimerkiksi piirtää tekstuureita tai järjestellä elementtejä ryhmiin. Täysi listaus GUI-luokan muuttujista ja funktioista löytyy Unityn Scripting API:sta.

Unityn dokumentaatio on todella selkeä ja helppo omaksua, eikä GUI-luokka ole tässä asiassa poikkeus. Scripting API:n lisäksi Unityn yhteisö on aktiivinen ja

tutoriaaleja ja esimerkkejä erilaisten toimintojen toteuttamiseksi on helppo löytää.

4.5 Heikkouksia

Unityn GUI-luokka on hyvä pohja luoda yksinkertaisia GUI-elementtejä. Se on kuitenkin vain pohja. Jos käyttöliittymään haluaa toteuttaa monimutkaisempia ominaisuuksia, on kaikki toiminnallisuus tehtävä itse. Moderneissa käyttöliittymissä on monia ominaisuuksia, joiden toteuttaminen tyhjästä on vaikeaa ja paljon aikaa vievää. Esimerkiksi rullaavat valikot tai elementistä toiseen siirrettävät kuvakkeet voivat kuulostaa yksinkertaiselta asialta, mutta ne ovat monimutkaisia toteuttaa. Rullaavissa valikoissa on näkyvissä vain osa valikon sisällöstä ja käyttäjä valitsee näkyvissä olevan alueen käyttöliittymän tarjoamilla keinoilla. Elementistä toiseen siirrettäviä kuvakkeita voidaan raahata ikkunasta toiseen esimerkiksi hiirellä. Peleissä niitä voidaan hyödyntää esimerkiksi esineiden keräämiseen ja säilyttämiseen liittyvissä toimenpiteissä. Näitä elementtejä tehdessä voi eteen tulla monta yllättävää ongelmaa esimerkiksi elementtien animointiin tai käyttäjän syötteiden tunnistamiseen liittyen.

Käyttöliittymän luominen Unityn GUI-luokalla ei onnistu sellaiselta graafikolta, joka ei osaa juurikaan skriptata. GUI-luokassa ei ole visuaalista editoria, joka auttaisi edes yksinkertaisimpien käyttöliittymäelementtien toteuttamisessa.

Unityn GUI-luokkaa on optimoitu paljon viimeisen parin vuoden aikana. Silti siinä on vielä suorituskykyongelmia, jotka on otettava huomioon käyttöliittymää tehdessä. Jokainen elementti tarvitsee oman piirtokutsun, jolloin vähänkin monimutkaisemman käyttöliittymän piirtokutsut alkavat nopeasti kasvamaan huomattaviksi.

Unityn käyttöliittymälisäosilla on yhteisenä piirteenä juuri näihin heikkouksiin puuttuminen. Ne lupaavat helppoja työkaluja monimutkaistenkin ominaisuuksien

toteuttamiseen sekä olevansa piirtokutsujen suhteen optimoidumpia kuin GUI-luokka. Tässä opinnäytetyössä arvioidaan näiden väittämien paikkansapitävyyttä.

4.6 Tulevaisuus

Unityn kehittäjät ovat tiedostaneet UI-työkalujen heikon tilan, ja he ovat kehittäneet uutta UI-järjestelmää (Goldstone 2014). GUI-luokan puutteista johtuen uutta GUI-järjestelmää on odotettu useita vuosia, ja pitkä odotus on omalta osaltaan edesauttanut monen käyttöliittymälisäosan julkaisua. Sen mahdollisia vaikutuksia käyttöliittymälisäosien kehitykseen tai myyntimääriin on vaikea vielä arvioida, mutta sen luulisi näkyvän myyntimäärien tippumisena.

Näillä näkymin uusi GUI-järjestelmä olisi tulossa Unityn versioon 4.6 (Johansen 2014). Siitä löytyy opinnäytetyön tekohetkellä vasta muutamia videoita ja blogikirjoituksia, jotka esittelevät uusia ominaisuuksia. Ominaisuudet vaikuttavat kuitenkin korjaavan useimpia vanhan GUI-järjestelmän puutteita. Siinä lisätään uudet graafiset työkalut käyttöliittymän tekemiseen, jolloin kaikkea ei tarvitse toteuttaa ohjelmoimalla OnGUI-metodissa. Myös suoritustehokkuuteen on tulossa parannusta.

5 Käyttöliittymälisäosat

Käyttöliittymälisäosa on kolmannen osapuolen kehittämä työkalu, jonka tarkoitus on mahdollistaa käyttöliittymän toteuttaminen tehokkaammin kuin Unityn GUI-luokka. Tässä luvussa listataan muutamia yleisempiä käyttöliittymälisäosia sekä tarkastellaan, miten ne pyrkivät ratkaisemaan aiemmin esitellyt Unityn GUI-luokan puutteet. Kolmella lisäosalla toteutetaan käyttöliittymä Slaughter-avaruuspelille, ja tässä luvussa esitellään opinnäytetyön toimeksiantajan sille tekemä käyttöliittymämäärittely. Lopuksi

kerrotaan eri lisäosien ominaisuuksista käyttöliittymän toteutuksesta saadun käyttökokemuksen perusteella.

5.1 Laajempi listaus lisäosista

Tässä luvussa listataan Unity Asset Storesta löytyviä käyttöliittymälisäosia. Asset Store on kauppapaikka Unityn maksullisille lisäosille. Lisäosat kuvataan varsin yleisellä tasolla, ja lisäksi tarkastellaan, millaisia arvioita käyttäjät ovat niista antaneet. Kaikkia tässä kuvattuja lisäosia ei ole valittu opinnäytetyön käytännön toteutukseen, vaan siihen on valittu muutama erilainen lisäosa Asset Storen kevään 2014 suosituimpien laajennosten listalta.

5.1.1 NGUI

NGUI (Next-Gen UI) on Tasharen Entertainmentin kehittämä lisäosa, jonka hinta Asset Storessa on 95 $. Se on yksi suosituimmista Asset Storen maksullisista lisäosista. Se on tehokas UI-työkalu, jolla voi tehdä käyttöliittymän PC:lle, konsoleille tai mobiililaitteille. Sen rakenne on tehty helposti ymmärrettäväksi ja muokattavaksi, joten sen käyttäminen hyvin erilaisissa peleissä on suhteellisen helppoa.

NGUI:n opettelu on tässä esitellyistä lisäosista haastavinta. Sen toimintaperiaate on tarjota käyttäjälle mahdollisimman hyvä kontrolli käyttöliittymää tehdessä. Sen kääntöpuolena on suuri määrä asetuksia ja säätöjä, joista käyttäjän pitää olla tietoinen. Käyttöliittymän toimintalogiikka määritellään laittamalla komponentteja käyttöliittymäelementteihin. Komponentteja luovasti hyödyntämällä voi NGUI:lla tehdä todella monipuolisia käyttöliittymäratkaisuja.

NGUI:n suosio näkyy useina hyvinä arvosteluina ja suosituksina. Peliyhtiö Proletariatin blogissa Joe Mukai vertailee muutaman eri lisäosan ominaisuuksia, ja päätyy käyttämään omassa peliprojektissaan NGUI:ta (Mukai 2014).

Vertailussa hän kiittelee erityisesti muutoksien tekemisen helppoutta, suoritustehokkuutta ja tukea useille eri alustoille. Andrei Chaiko vertailee blogikirjoituksessaan (2012) NGUI:n ja EzGUI:n eroja, ja vertailun tulos on selkeästi NGUI:lle myönteinen. Hän kehuu erityisesti NGUI:n käytettävyyttä ja luotettavuutta.

5.1.2 Daikon Forge GUI Library

Daikon Forge on Daikon Forgen kehittämä lisäosa, jonka hinta Asset Storessa oli 100 $. Se on huomattavasti uudempi käyttöliittymälisäosa kuin NGUI. Se on noussut nopeasti yhdeksi Asset Storen suosituimmista lisäosista. Daikon Forge on helppo ottaa käyttöön graafisen käyttöliittymän avulla, eikä se vaadi ohjelmointiosaamista. Sen toimintaperiaate on tarjota käyttäjälle valmiita elementtikokonaisuuksia, joita olisi mahdollisimman helppo luoda ja hallita.

Daikon Forgesta löytyy lähinnä myönteisiä arviointeja. Sitä vertaillaan monesti NGUI:hin, ja moni NGUI:n käyttäjä on siirtynyt käyttämään sitä. Tonio Leowald kehuu blogikirjoituksessaan (2014) sen toiminnallisuutta ja suorituskykyä, mutta moittii sillä työskentelyä hieman kankeaksi. Unityn virallisilla foorumeilla Daikon Forgea yleisesti kehutaan, mutta se saa moitteita sen uutuudesta johtuvista pikkubugeista (Unity Forum 2014).

Daikon Forge on tämän opinnäytetyön tekemisen aikana poistunut Unityn Asset Storesta. Sen kehittäjä lopetti lisäosan tukemisen ja sen kehittäminen jatkuu nykyään avoimen lähdekoodin periaatteella. Peruskäyttäjälle se tarkoittaa lähinnä sitä, että lisäosan voi ottaa käyttöön ilmaiseksi.

5.1.3 EzGUI

EzGUI on Above and Beyond Softwaren kehittämä lisäosa, jonka hinta Asset

Storessa on 200 $. Se on ollut perinteisesti NGUI:n ohella toinen suuri käyttöliittymälisäosa. Myös sen lähtökohta on ollut tarjota mahdollisuus luoda pelin käyttöliittymä graafisesti. EzGUI tarjoaa käyttäjälle enemmän säädettäviä ominaisuuksia kuin NGUI. Se vaatii kuitenkin enemmän paneutumista EzGUI:n ominaisuuksiin ja sen käyttöönotto voikin olla hankalampaa. EzGUI:ssa on myös vähemmän valmiita käyttöliittymäelementtejä kuin NGUI:ssa, joten joidenkin ominaisuuksien tekemiseen saatetaan tarvita ohjelmointiosaamista.

EzGUI on viime aikoina menettänyt suosiotaan jopa siinä määrin, että siitä on vaikea löytää tuoreita arviointeja. Philip Chu toteaa blogikirjoituksessaan (2012), että EzGUI:n käyttöönotto on vaikeaa, ja päätyy suosittelemaan NGUI:ta. Unity Asset Storen käyttäjäarvioissa EzGUI:ta moititaan heikosta dokumentaatiosta ja graafisten työkalujen puutteesta, mikä tekee sen käyttöönotosta vaikeata (Unity Asset Store 2014). Sitä kehutaan kuitenkin tehokkaaksi lisäosaksi, kunhan alkuvaikeuksista on päässyt ohi, ja myös sen suoritustehokkuutta kiitellään (Unity Asset Store 2014).

5.1.4 IGUI

IGUI on Avam Studiosin kehittämä lisäosa, jonka hinta Asset Storessa on 65 $. Se on erittäin helposti lähestyttävä hyvien visuaalisten työkalujen ansiosta. Sen perusidea on luoda asetustiedosto, johon säädetään kaikkien elementtien ulkonäkö. Kun elementtejä lisää käyttöliittymään, saavat ne automaattisesti asetetun ulkomuodon, eikä sitä tarvitse aina asettaa erikseen. Sen lisäksi, jos asetustiedostoa muokataan, muuttuvat jo olemassa olevat elementit uusien asetusten mukaisiksi. Se muistuttaa siis paljon web-ohjelmoinnista tuttua CSS-ajattelua.

IGUI:sta on vaikea löytää arviointeja tai vertailuja muihin lisäosiin. Unity-foorumeiden IGUI-keskustelussa (2011) sitä kehutaan helppokäyttöiseksi ja intuitiiviseksi. Sen huonoksi puoleksi moititaan sen suorituskykyä, koska se käyttää Unityn omaa raskasta GUI-luokkaa käyttöliittymän piirtämiseen. Unity

Asset Storen käyttäjäarvioissa kehutaan iGUI:n helppoa käyttöönottoa ja kehittämän tarjoamaa hyvää tukea asiakkaille (Unity Asset Store 2014).

5.1.5 GUIMaker

GUIMaker on viiden dollarin hinnallaan huomattavasti edullisempi kuin edellä mainitut käyttöliittymälisäosat. Sen kehittäjä on hpjohn, joka on pitkäaikainen Unityn käyttäjä, ja hyvin tunnettu ja aktiivinen henkilö Unityn foorumeilla. Sen toimintaperiaate eroaa aiemmin mainituista lisäosista huomattavasti. Siinä luodaan graafisilla työkaluilla käyttöliittymä Unityn omista elementeistä ikään kuin piirtämällä. Lopputuloksesta luodaan kooditiedosto, ja se piirtää luodun käyttöliittymän käyttäen vain Unityn GUI-luokan elementtejä.

GUIMaker saa yleisesti kehuja käyttöliittymän tekemisen helpottamisesta, mutta sitä ei pidetä kovin mullistavana lisäosana. Vaikka sillä onkin nopea luoda käyttöliittymä, pidetään sillä työskentelyä hieman kankeana epäintuitiivisen työjärjestyksen takia. (Unity Asset Store 2014.)

5.1.6 Ominaisuuksien yhteenveto ja vertailu

Käyttöliittymälisäosien yleistiedot on koottu taulukkoon 1. Tiedot on haettu Unity Asset Storesta keväällä 2014, ja päivitetty Daikon Forgen osalta syksyllä 2014. Tekniset tiedot on haettu lisäosan kehittäjän sivuilta. Käyttäjäarvostelut-kohtaan on koottu, millaisia arvioita eri käyttäjät ovat lisäosista antaneet. Opettelu-kohtaan vaikuttaa lisäosan käyttöönotto ja työkalujen helppous. Työskentely-kohdassa arvioidaan lisäosan käyttämisen sujuvuutta ja muutoksenteon helppoutta. Dokumentaatiolla tarkoitetaan koodin dokumentaatioita ja tutoriaalien saatavuutta ja selkeyttä. Tehokkuudella tarkoitetaan edistyneempien käyttäjien tarpeiden huomiointia, kuten monimutkaisempien ominaisuuksien toteuttamista ja suoritustehokkuutta.

Taulukko 1. Käyttöliittymälisäosien ominaisuudet.

Nimi	NGUI	Daikon Forge	iGUI	EzGUI	GUIMaker
		Yleistä			
Kehittäjä	Tasharen Entertainment	Daikon Forge	Avam Studios	Above and Beyond Software	hpjohn
Hinta	95$	Opensource	65$	200$	5$
		Teknistä tietoa			
Tekstuuriatlas	Kyllä	Kyllä	Ei	Kyllä	Ei
Piirtokutsujen optimointi	Kyllä	Kyllä	Ei	Kyllä	Ei
		Käyttäjäarvostelut			
Opettelu	Keskivaikea	Helppo	Helppo	Vaikea	Helppo
Työskentely	Helppo	Helppo	Helppo	Vaikea	Keskivaikea
Dokumentaatio	Keskitasoa	Hyvä	Hyvä	Keskitasoa	Heikko
Tehokkuus	Hyvä	Hyvä	Heikko	Hyvä	Heikko

Opinnäytetyön käytännön osuudessa toteutettiin käyttöliittymä kolmella lisäosalla, ja niille kirjoitettiin käyttöohje. Käyttöliittymä toteutettiin NGUI:lla, Daikon Forgella ja iGUI:lla. Valinta perustui lisäosien ominaisuuksien ja käyttäjäarvosteluiden vertailusta saatuun arvioon. NGUI valikoitui ylivoimaisen suosionsa takia. IGUI ja GUIMaker ovat toteutukseltaan samankaltaisia, koska ne käyttävät Unityn GUI-luokkaa käyttöliittymän piirtämiseen, ja valinta kääntyi iGUI:hin, koska se vaikuttaa kokonaisvaltaisemmalta. Daikon Forge valikoitui kolmanneksi, koska se edustaa uudempaa sukupolvea kuin EzGUI ja NGUI. Se on myös saanut parempia arvosteluja kuin EzGUI. EzGUI näyttäisi myös menettäneen suosiotaan viime vuosina, ja se vaikuttaa jonkin verran sen jatkosta pois jättämiseen.

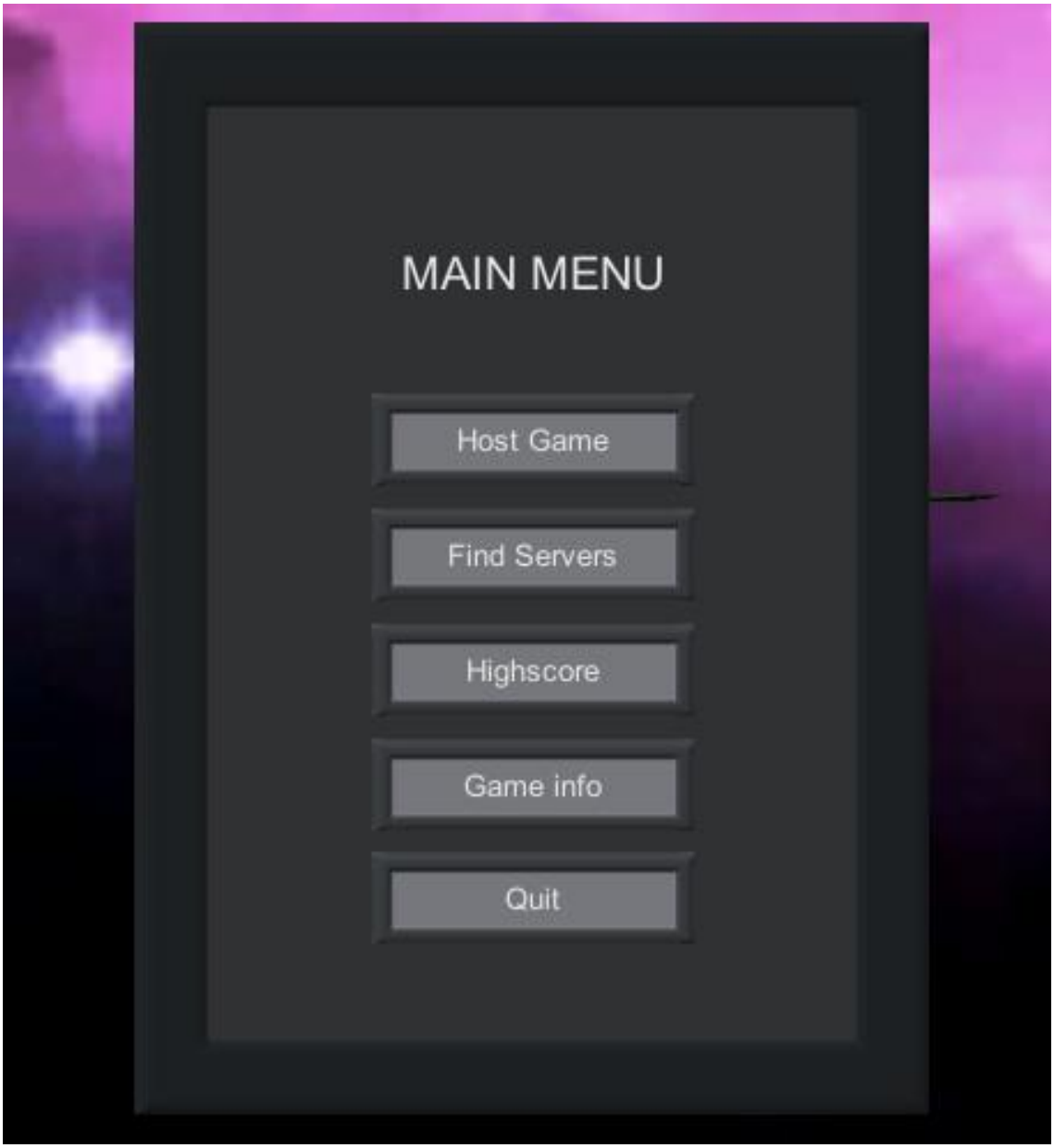
5.2 Käytännön osuus

Opinnäytetyön käytännön osuudessa tehtiin graafinen käyttöliittymä Slaughter-avaruuspeliin. Peli on tehty Unityllä, ja käyttöliittymät toteutettiin kolmella Unityn käyttöliittymälisäosalla. Käytettävät lisäosat valittiin pääsosin suosion ja käyttäjäpalautteen perusteella. Käytettävät lisäosat olivat NGUI, iGUI ja Daikon Forge GUI Library.

5.2.1 Slaughter-pelin käyttöliittymämäärittely

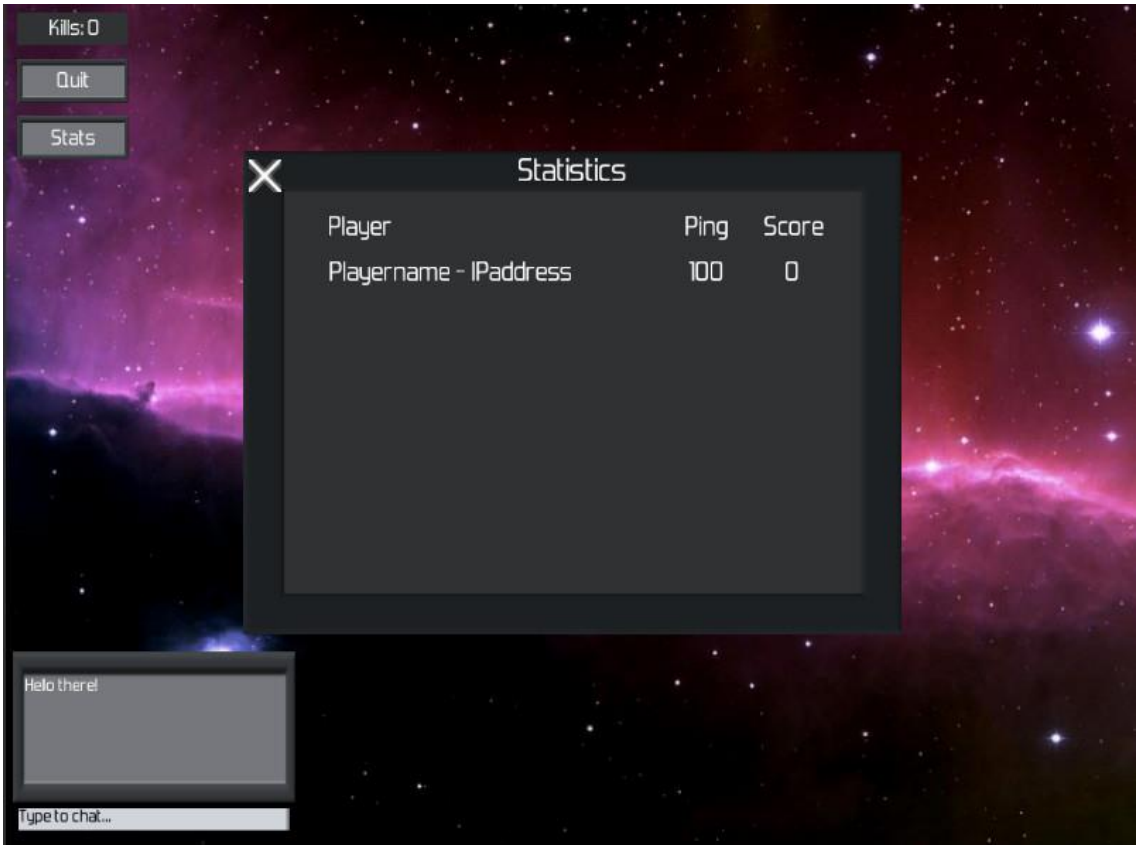
Työn alussa toimeksiantajalta tuli käyttöliittymämäärittely, jossa listataan käyttöliittymältä vaadittavat ominaisuudet (Liite 1). Siinä vaaditaan toteutettavaksi erikseen alkuvalikko ja pelinäkymä. Määrittely sisältää paljon peleille tyypillisiä ominaisuuksia, joten se on hyvä lähtökohta vertailla eri käyttöliittymälisäosien ominaisuuksia.

Alkuvalikko (Kuva 16) on toiminnaltaan hyvin perinteinen. Siinä on päävalikko, josta pääsee alavalikoihin painikkeita painamalla ja niistä takaisin päävalikkoon sulkemispainikkeesta. Alavalikoissa toteutetaan niissä vaadittava toiminnallisuus, kuten pelipalvelimen perustaminen tai ennätyspisteiden tarkastelu.



Kuva 16. Slaughter-pelin alkuvalikko (Kuva: Niko Sarkkinen).

Pelinäkymässä (Kuva 17) pelaajan omat pisteet ovat koko ajan nähtävissä. Sen lisäksi painiketta painamalla avautuu ikkuna, josta näkyy kaikkien pelaajien pisteet sekä ping-vasteaika. Pelissä on myös chat-ominaisuus, ja sen toteuttaminen vaatii monen eri käyttöliittymäelementin yhteistoimintaa. Sen totetuttaminen onkin hyvä testi käyttöliittymälisäosille.



Kuva 17. Slaughter-pelin pelinäkymä (Kuva: Niko Sarkkinen).

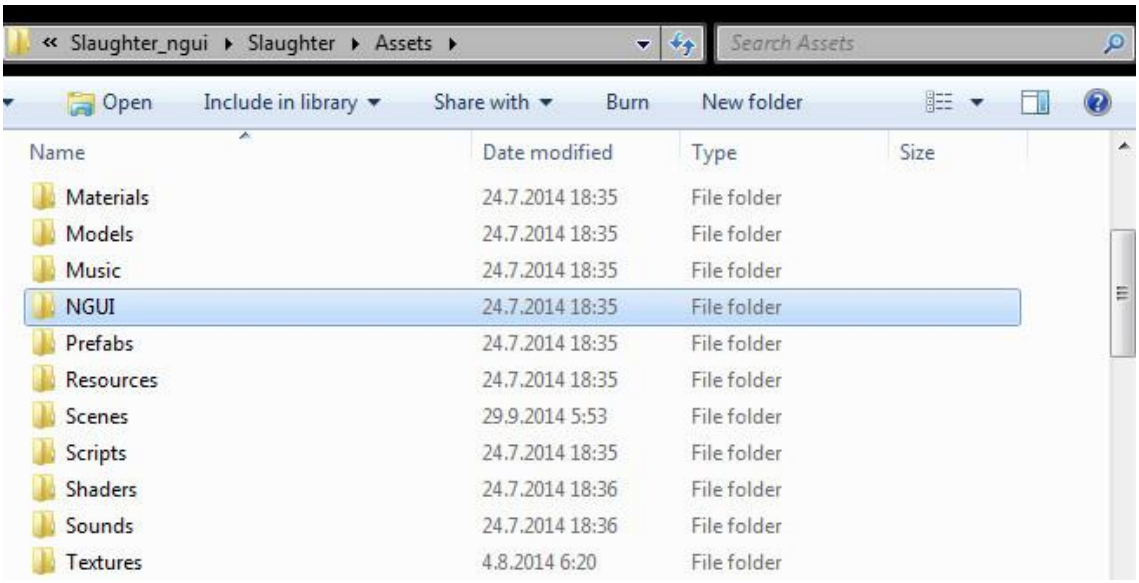
5.3 Vertailtavat ominaisuudet

Käyttöliittymälisäosia vertailtiin toisiinsa Slaughter-pelin käyttöliittymän tekemisestä saatujen käyttökokemusten perusteella. Käyttökokemus on subjektiivinen, ja perustuu omiin havaintoihin ja huomioihin. Lisäosia vertaillaan niiden helppokäyttöisyyden, monipuolisuuden ja suoritustehokkuuden kannalta.

5.3.1 Käyttöönotto ja opettelu

Käyttöliittymälisäosat lupaavat helppoja ja intuitiivisia työkaluja käyttöliittymän tekemiseen. Ensimmäinen arviointikriteeri onkin lisäosan käyttöönoton helppous ja tarvittavan opettelun määrä.

Lisäosien käyttöönotto Unityssä on yleisesti ottaen todella helppoa. Lisäosa tarvitsee vain lisätä Unity-projektin tiedostoihin (Kuva 18), ja Unity suorittaa tarvittavat koodin kääntämiset automaattisesti. Käyttöönotolla tarkoitetaankin tässä yhteydessä mahdollisten asetustiedostojen tai tekstuuriatlaksien luomista.



Kuva 18. Käyttöliittymälisäosa lisätään Unity-projektin Assets-kansioon (Kuva: Niko Sarkkinen).

Lisäosan työkalujen ja toimintalogiikan oppimiseen tarvitaan aina aikaa. Työkalujen selkeys ja intuitiivisuus on tietenkin aina riippuvainen käyttäjästä, mutta niitä vertaillaan siltä osin kun se on mahdollista. Oppimista helpottaa hyvä manuaali tai API-referenssi. Useimpien lisäosien mukana tulee myös esimerkkitiedostoja, joista saa hyvin katsottua mallia, kuinka erilaisia ominaisuuksia voi toteuttaa.

5.3.2 Kontrolli

Kontrollilla tarkoitetaan sitä, kuinka paljon käyttöliittymän suunnittelussa on otettava huomioon lisäosan aiheuttamat tekniset rajoitukset. Pelin genre ja visuaalinen tyyli vaikuttavat siihen, mitä käyttöliittymäelementtejä helppokäyttöisen ja selkeän käyttöliittymän toteuttamiseen tarvitaan. Tässä

kohdassa arvioidaan siis lisäosien antamaa mahdollisuutta muokata elementtien rakennetta.

Monissa peliprojekteissa käyttöliittymän suunnittelija ja toteuttaja on eri henkilö. Tyypillisesti suunnittelijalla on graafinen tausta, joten hänellä ei välttämättä ole käsitystä teknisen toteutuksen vaikeudesta tai helppoudesta. Käyttöliittymän toteuttajan vastuulla on siis huolehtia, että suunnitelma voidaan toteuttaa, ja mielellään ilman kohtuutonta työmäärän kasvua. Käyttöliittymälisäosan hyvä muokattavuus mahdollistaa juuri halutunlaisen käyttöliittymän toteutuksen ilman, että suunnitelmista täytyy karsia.

Pienemmissä projekteissa käyttöliittymän suunnittelija ja toteuttaja on hyvin todennäköisesti sama henkilö. Siinä tapauksessa tekniset rajoitukset tulee otettua huomioon jo käyttöliittymän suunnittelussa. Käyttöliittymälisäosan mahdollisuudet ja rajoitukset ovat siinäkin tapauksessa tärkeässä osassa, koska hyvät suunitelmat jäävät helposti toteuttamatta, jos lisäosan rajoitukset tekevät siitä vaikeata.

5.3.3 Peruselementit

Joissakin peliprojekteissa käyttöliittymä suunnitellaan niin, että sen voi toteuttaa täysin peruselementeillä. Peruselementeiksi lasketaan tässä arvioinnissa esimerkiksi sprite-, label-, button- ja panel-elementit. Ne ovat elementtejä, joilla on yleensä yksi perustehtävä, kuten tekstin tai tekstuurin näyttäminen. Peruselementtien helppo ja intuitiivinen toteuttaminen voi tehdä käyttöliittymän tekemisestä todella nopeaa, ja voi olla monelle juuri se syy, miksi lisäosan hankkii.

5.3.4 Monipuolisuus

Peruselementtien lisäksi käyttöliittymät yleensä tarvitsevat elementtejä jonkin erityistehtävän suorittamiseen. Sellaisia voi olla esimerkiksi alasvetovalikot ja vieritys- tai valintanäkymät. Monipuolisuutta on arvioitu siitä näkökulmasta, kuinka monipuolisesti eri lisäosat tarjoavat näitä elementtejä.

Monipuolisia ominaisuuksia voidaan toteuttaa kahdella tavalla. Ominaisuutta varten voi lisäosasta löytyä valmis elementti, joka on luotu juuri sitä tarkoitusta varten. Toinen tapa on kasata peruselementtejä niin, että ne yhdessä muodostavat uudenlaista toiminnallisuutta. Jos lisäosa on sellainen, että siinä voi luoda uusia elementtejä yhdistelemällä muiden elementtien toimintoja, on vertailussa arvioitu myös mahdollisten ohjeiden ja esimerkkien tasoa.

Käyttöliittymälisäosaa valitessa ei välttämättä vielä tiedä tarkalleen, millaista käyttöliittymää sillä lopulta tulee tekemään. Lisäosaa voidaan mahdollisesti myös käyttää useammassa peliprojektissa. Tämän vuoksi on hyvä, jos lisäosa sisältää peruselementtien lisäksi mahdollisimman suuren valikoiman monipuolisten elementtien tekemiseen.

5.3.5 Suoritustehokkuus

Käyttöliittymälisäosilla on Unityn GUI-luokkaaan verrattuna yleensä kaksi selkeästi suoritustehokkuutta parantavaa ominaisuutta. Ne ovat tekstuuriatlas ja piirtokutsujen merkittävä väheneminen.

Tekstuuriatlaksessa kaikki käytettävät tekstuurit pakataan yhteen isoon tekstuuriin. Atlaksen etu verrattuna isoon määrään pienempiä tekstuuritiedostoja on sen prosessointitehokkuudessa ja tiedonsiirrossa verkon yli. Grafiikkaohjaimen ei tarvitse prosessoida jokaista piirrettävää tekstuuria erikseen, millä voi olla suuri merkitys erityisesti mobiilialustoille kehittäessä. Jos

peliä kehitetään verkkoon, esimerkiksi web-selaimella pelattavaksi, syntyy jokaisesta yksittäisestä tiedostosta lisää verkkoliikennettä. Se pidentää pelin lataamisaikaa ja lisää pelin sisältävän palvelimen rasitusta, mikä voi nostaa kustannuksia.

Unityn GUI-luokalla tehdyt graafiset elementit vaativat jokainen aina oman piirtokutsun. Käyttöliittymälisäosat lupaavat vähentää piirtokutsuja yhdistämällä elementtejä. Elementtien tehokas yhdistäminen vaatii käyttöliittymäelementtien rakentamisen tietyllä tavalla.

Suoritustehokkuutta on vertailtu sitä näkökulmasta, kuinka helppoa eri lisäosilla on rakentaa tehokas käyttöliittymä. Siihen liittyy olennaisesti ohjeistuksen saatavuus ja selkeys. Sen lisäksi on tarkasteltu, kuinka paljon piirtokutsuja voidaan kullakin lisäosalla vähentää.

6 Käyttöliittymälisäosien käyttökokemus

Tässä luvussa käsitellään eri käyttöliittymälisäosia niistä saadun käyttökokemuksen pohjalta. Niistä arvioidaan edellisessä luvussa esiteltyjä ominaisuuksia.

6.1 NGUI

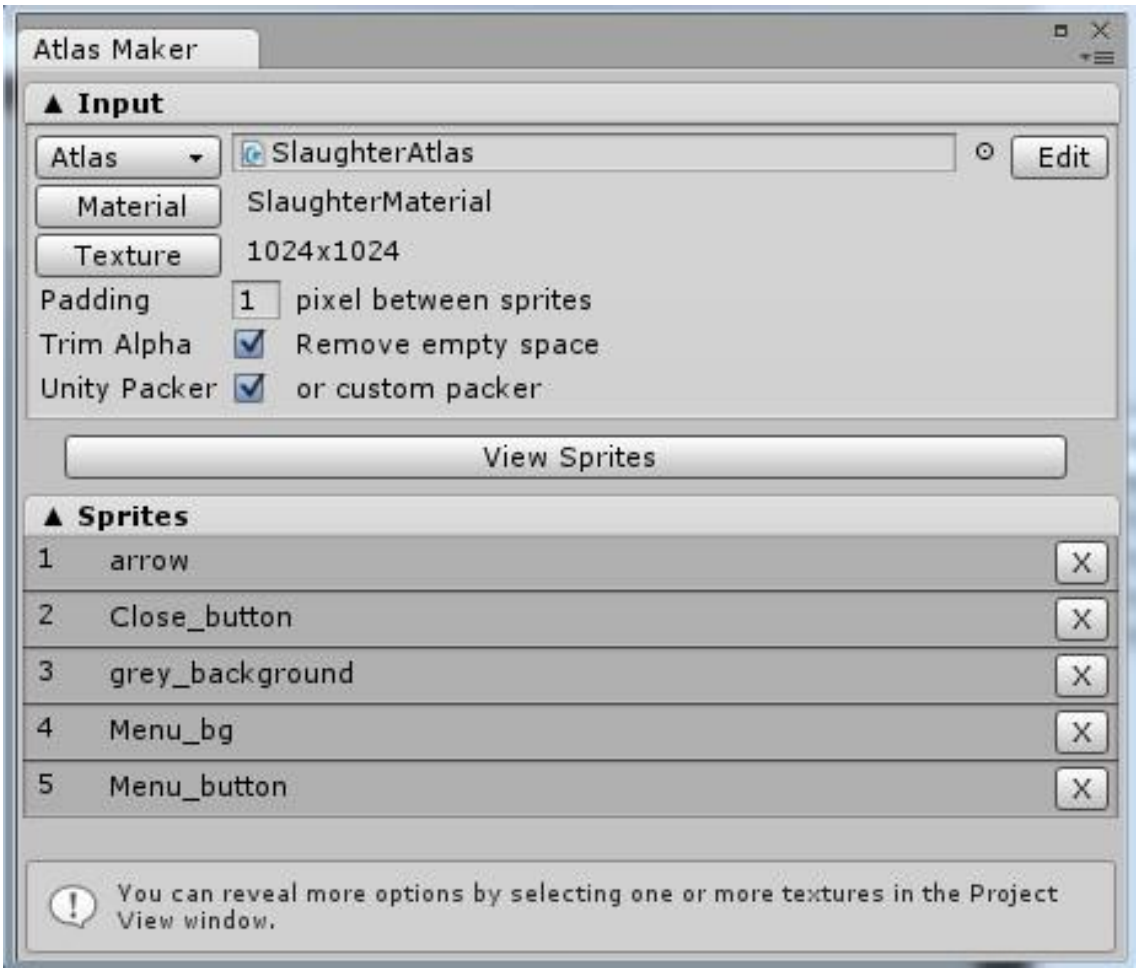
NGUI on valituista käyttöliittymälisäosista suosituin. Sen toimintaperiaate on tarjota käyttäjälle käyttöliittymän toiminnalle tarvittavat skriptit. Käyttöliittymäelementit kootaan monesta eri komponentista, ja käyttäjä voi päättää miten ne kootaan. Käyttäjä voi halutessaan myös jättää jotain komponentteja pois.

6.1.1 Käyttöönotto ja opettelu

NGUI tuntuu valituista lisäosista vaikeimmalta opetella. Sen tyyli kasata elementit monesta komponentista on aluksi haastava, koska erilaisia mahdollisuuksia on paljon. Mahdollisuus unohtaa jonkin tärkeän komponentti on olemassa.

Käyttöliittymän tekemisen aloittaminen on NGUI:lla helppoa. Käyttöliittymän asetukset määritellään luonnin yhteydessä, ja asetusten muuttaminen on mahdollista milloin vain. Asetukset ovat monipuoliset ja niistä löytyy tarpeeksi säätöjä, tekee peliä sitten pc:lle, konsolille tai mobiilialustalle.

Tekstuuriatlaksen tekemiseen tarkoitettu työkalu on todella helppo käyttää, ja atlasta on erittäin helppo muokata myöhemmin ilman, että mitään jo tehtyä työtä menee hukkaan. Atlaksen luonti ja hallinta on NGUI:n ehdoton vahvuus muihin lisäosiin verrattuna, sillä Daikon Forgen atlastyökalu on huomattavasti vaikeampi käyttää ja iGUI ei käytä atlasta.



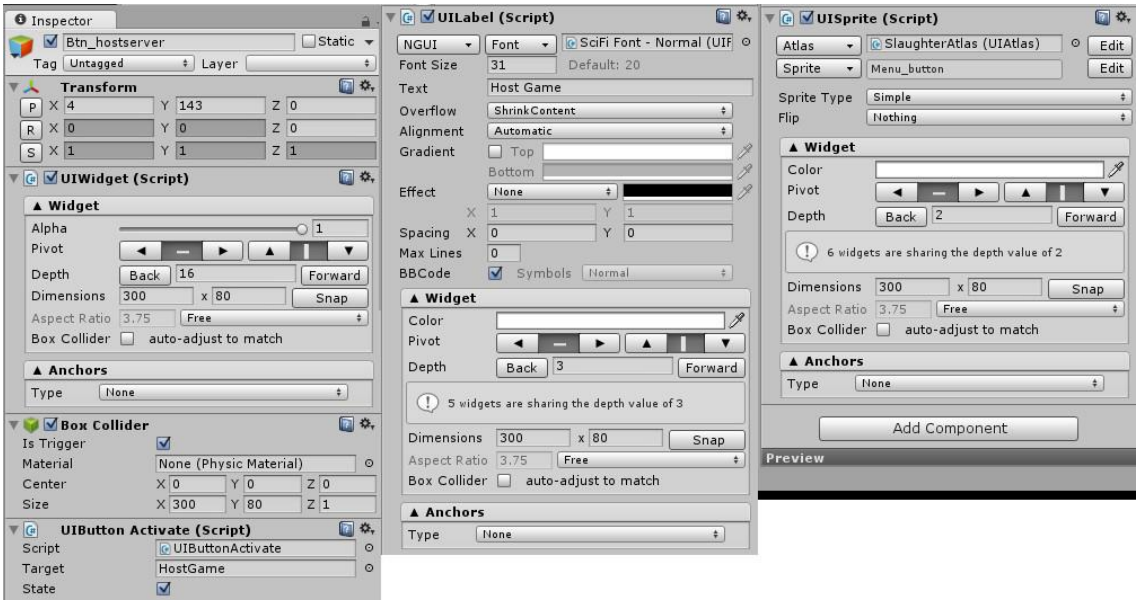
Kuva 19. NGUI:n atlastyökalu (Kuva: Niko Sarkkinen).

NGUI:n monimutkaisia ja monesta komponentista kasattuja elementtejä voi olla helpompi oppia käyttämään tutkimalla sen mukana tulevia esimerkkitiedostoja. Esimerkeissä on esitettynä monia yleisimpiä käyttöliittymien ominaisuuksia. Niistä voi poimia omaan käyttöliittymän toteutukseen joko valmiita suuria kokonaisuuksia tai pienempiä yksittäisiä elementtiratkaisuja.

NGUI:n koodin dokumentaatio on tehty suositulla Doxygen-dokumentaatiotyökalulla, joten se noudattaa yleisesti ohjelmoijille tuttuja käytänteitä. Sitä on siis helppo lukea, jos tekijällä on ohjelmointitaustaa. Dokumentaatiosta löytyy referenssit lähdekoodille sekä esimerkkitiedostoissa käytetetyille malliskripteille.

6.1.2 Kontrolli

NGUI:n tyyli luoda käyttöliittymäelementtejä useasta komponentista (Kuva 20) antaa sille valituista lisäosista parhaan kontrollin. Käyttäjä voi muokata yksittäisiä elementtejä tai suurempia kokonaisuuksia juuri halutunlaiseksi.



Kuva 20. Painikkeen toteuttamiseen tarvittavia komponentteja (Kuva: Niko Sarkkinen).

NGUI sopii siis valituista lisäosista parhaiten sellaiseen peliprojektiin, jossa käyttöliittymän suunnittelija ja toteuttaja ovat eri henkilö ja jossä suunitelmista ei haluta karsia teknisten rajoitteiden takia. Se vaatii ehkä toteuttajalta enemmän opettelua kuin toiset lisäosat, mutta se voi monesti olla sen arvoista.

NGUI on muokattavuuden ja kontrollin ansiosta hyvä lisäosa, jos sitä on tarkoitus käyttää useammassa peliprojektissa. Lisäosan hankintahetkellä ei voi tietää, millaisia käyttöliittymiä sillä mahdollisissa tulevaisuuden peliprojekteissa tullaan tekemään. Hieman vaikeamman käytön opettelun voi siis perustella paremmilla jatkokäyttömahdollisuuksilla.

6.1.3 Peruselementit

Peruselementit täytyy NGUI:ssa kasata monesta komponentista. Se tarkoittaa sitä, että niiden tekeminen on valituista lisäosista työläintä.

Peruselementit koostuvat toimintalogiikan sisältävästä skriptistä ja muista mahdollisesti tarvittavista komponenteista. Muita komponentteja voi olla esimerkiksi Collider-komponentti, joka tunnistaa jos elementtiä klikataan hiirellä tai kosketusruudulla, tai ulkonaköön vaikuttavat komponentit kuten Sprite-komponentti.

Yksinkertaisin peruselementti on varmaankin tekstiä näyttävä Label-elementti. Se vaatii yksinkertaisimmillaan Label-komponentin ja NGUI:lle erityisen Widget-komponentin, jonka tehtävä on huolehtia elementin asettelusta muuhun käyttöliittymään nähden. Widget-komponentti määrittää esimerkiksi, mihin reunaan elementin muut komponentit sijoittuvat ja elementin syvyysarvon, joka määrää missä järjestyksessä päällekkäiset elementit piirretään. Jos tekstille halutaan taustakuva, vaatii se lisättäväksi Sprite-komponentin, johon asetetaan jokin tekstuuriatlaksen tekstuureista.

Button-elementti voidaan luoda lisäämällä komponentteja Label-elementtiin. Painikkeen toimintaan vaaditaan vähintään Collider-elementti tunnistamaan painallukset ja jokin skripti, joka sisältää painikkeen toiminnallisuuden. Button-komponentti on painikkeen toimintojen kannalta ehkä hieman yllättäen vapaaehtoinen. Siinä määritellään lähinnä, kuinka painikkeen ulkonäkö muuttuu sitä painaessa tai kun hiiri viedään sen päälle.

Kuten näistä kahdesta esimerkistä voi päätellä, peruselementtien tekeminen ei ole NGUI:lla aivan yksinkertaista. Se on jopa aika kankeaa verrattuna iGUI:n tai Daikon Forgen tapaan. Niissä peruselementit ovat yksi valmis komponentti, ja NGUI:n tyylistä osista kasaamista ei vaadita.

6.1.4 Monipuolisuus

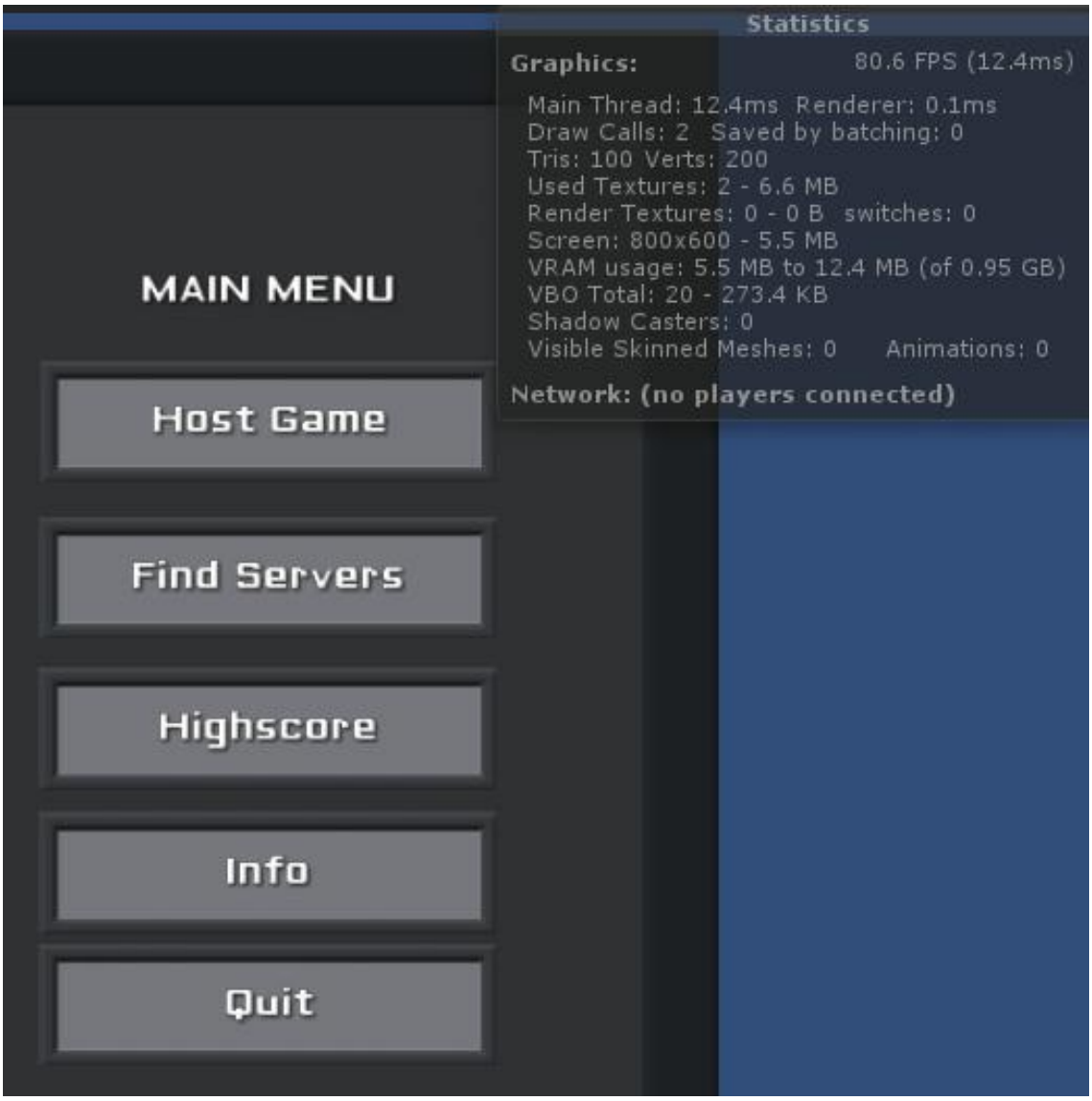
NGUI:ssa on suuri määrä monipuolisia käyttöliittymäelementtejä. Siitä löytyy elementit esimerkiksi erilaisten valinta- ja listanäkymien toteuttamiseen sekä mahdollisuus tehdä ikkunoita, joiden sijaintia ja kokoa voi muokata pelin ollessa käynnissä.

NGUI:n tyyli kasata elementit komponenteista näyttää vahvuutensa, kun halutaan tehdä monipuolisia ominaisuuksia. Samoilla peruskomponenteilla voi niitä luovasti yhdistelemällä saada aikaiseksi mielenkiintoisia ominaisuuksia.

NGUI:n mukana tulevissa esimerkkitiedostoissa on toteutettu erilaisia monipuolisia käyttöliittymäkokonaisuuksia. Niitä tutkimalla löytää hyvin ideoita ja mahdollisuuksia, joita omassa käyttöliittymässä voisi hyödyntää. Niissä on valmiiden komponenttien lisäksi hyödynnetty jonkin verran räätälöityjä skriptejä tapauskohtaisten toimintojen suorittamiseen. Se on yksi osoitus siitä, että käyttöliittymälisäosat eivät ole työkaluja, joilla voisi aina luoda täysin halutunlaisen käyttöliittymän pelkillä graafisilla työkaluilla.

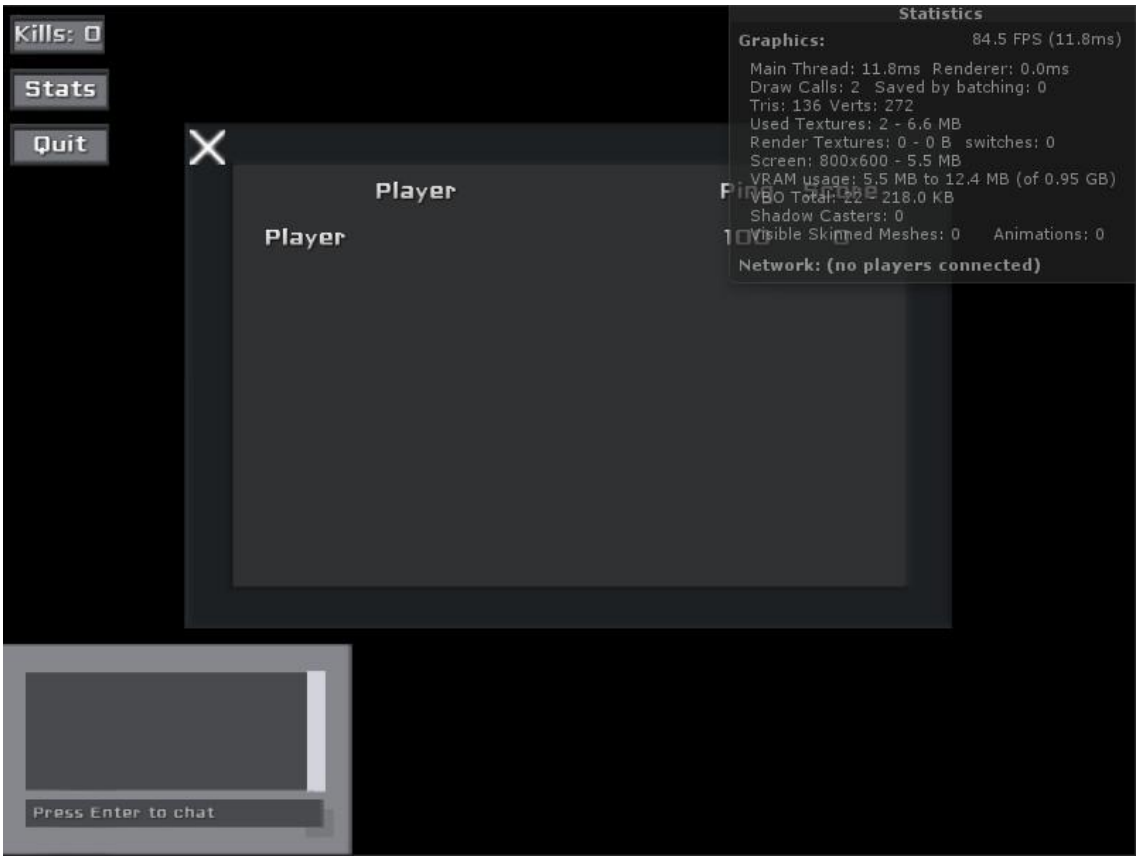
6.1.5 Suoritustehokkuus

NGUI:lla toteutettu alkuvalikko käyttää vain kaksi piirtokutsua jokaisella ruudunpäivityskierroksella (Kuva 21). Kaksi piirtokutsua syntyy, koska käyttöliittymässä käytetään kahta tekstuuriatlasta. Toinen atlaksista sisältää kaikki tekstuurit ja toinen fontin. Periaatteessa piirtokutsut voitaisiin pudottaa yhteen, mutta tekstuurien ja fontin yhdistämineen samaan tekstuuriin ei ole käytännöllistä.



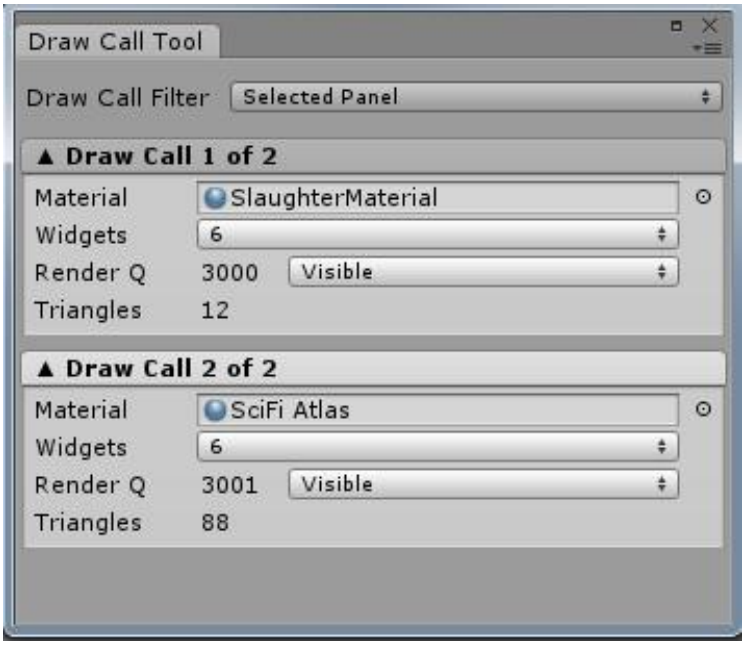
Kuva 21. Tilastoja NGUI:lla toteutetun alkuvalikon suoritustehokkuudesta (Kuva: Niko Sarkkinen).

Pelinäkymässä päästään myös kahteen piirtokutsuun (Kuva 22). Pelinäkymässä on paljon eri elementtejä ympäri ruutua, joten piirtokutsujen optimointi ei ole aivan itsestään tapahtuva toimenpide. Elementtien syvyysarvot täytyy asettaa oikein, koska NGUI käyttää useamman piirtokutsun, jos se joutuu piirtämään eri syvyyksissä olevia ja eri materiaaleja käyttäviä elementtejä päällekkäin. Sen vuoksi on tärkeä, että toteutetussa käyttöliittymässä ole tekstuurielementtejä tekstin päällä.



Kuva 22. Tilastoja NGUI:lla toteutetun pelinäkymän suoritustehokkuudesta (Kuva: Niko Sarkkinen).

NGUI:n piirtokutsujen optimointia helpottaa todella paljon siitä löytyvä Draw Call -työkalu (Kuva 23). Siinä näkyy, mitä elementtejä jokainen piirtokutsu sisältää, ja erityisesti siitä löytyy helposti ongelmatilanteet. Jos joku piirtokutsu sisältää vain yhden elementin, tai samalle materiaalille on useampi piirtokutsu, voi olla melko varma, että käyttöliittymässä on vielä varaa optimoinnille.



Kuva 23. NGUI:n Draw Call -työkalu (Kuva: Niko Sarkkinen).

6.2 IGUI

IGUI paljastui valituista käyttöliittymälisäosista helpoimmaksi käyttää. Sen käyttäminen on helppoa ja intuitiivista yksinkertaisten ja selkeiden työkalujen ansiosta (Kuva 24). Sen käyttöliittymäelementit ovat rakenteeltaan valmiita ja suljettuja kokonaisuuksia, joten niiden luonti on vaivatonta. IGUI hyödyntää Unityn GUI Skin -asetustiedostoa, johon asetetaan ennalta elementtien ulkonäkö. Elementtejä luodessa IGUI tekee niistä automaattisesti asetustiedossa määritellyn näköisen.

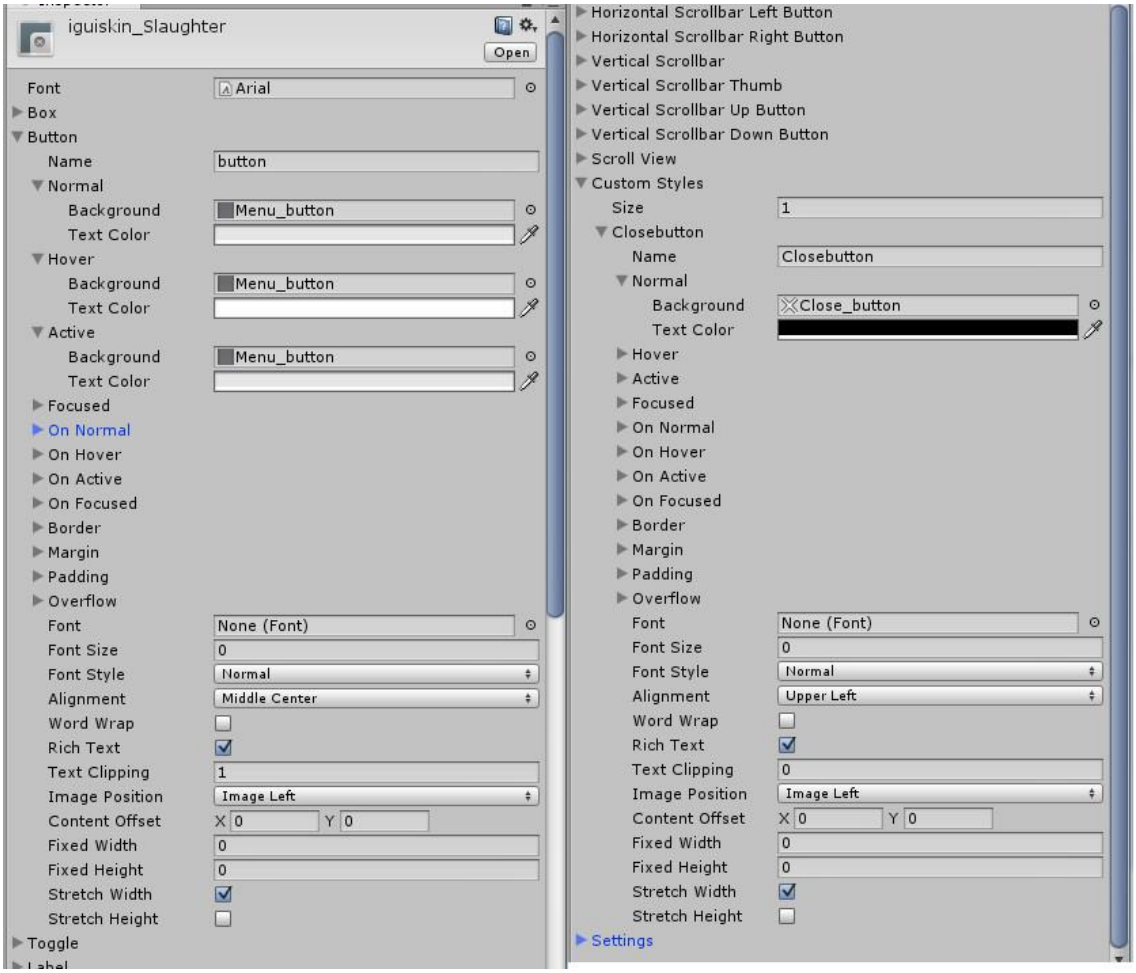
Kuva 24. IGUI:n työkalupalkki (Kuva: Niko Sarkkinen).

6.2.1 Käyttöönotto ja opettelu

IGUI:n käyttäminen on heti alusta asti selkeää ja helppoa. Se on todella intuitiivinen ja sen työkalut ovat selkeitä. Käyttöliittymäelementit ovat aina jo luontihetkellä valmiita kokonaisuuksia, ja ne vaativat vain asetusten

määrittämisen.

IGUI ei käytä tekstuuriatlasta, mikä helpottaa sen käyttöönottoa, mutta voi olla merkittävä puute suorituskyvyn kannalta. Atlaksen sijaan iGUI käyttää GUI Skin -asetustiedostoa. Siihen säädetään, miltä käyttöliittymän eri elementit, kuten otsikot ja painikkeet, näyttävät. Elementtejä luodessa niistä tulee automaattisesti asetustiedoston säätöjen mukainen. GUI Skin tiedoston toinen merkittävä etu on, jos elementtien ulkonäköä tarvitsee muokata myöhemmin. Muutokset tarvitsee tehdä vain asetustiedostoon, ja muutokset siirtyvät kaikkiin jo tehtyihin elementteihin.



Kuva 25. GUI Skin -asetustiedosto (Kuva: Niko Sarkkinen).

Uuden käyttöliittymän luonti on iGUI:lla helppoa. Uusi käyttöliittymä luodaan nappia painamalla, ja asetukset säädetään yhdestä skriptistä. Asetusten määrä

on melko suuri, eikä olennaisia asetuksia ole helppo erottaa toisarvoisten asetusten joukosta. Onneksi asetuksia voi kuitenkin muokata myöhemminkin, jos huomaa siihen tarvetta.

IGUI:n käyttöliittymäelementit on helppo luoda. Ne ovat valmiita kokonaisuuksia, eikä esimerkiksi NGUI:lle tyypillistä komponenteista kasaamista tarvita. Elementtien luonti on mutkatonta, ja elementti on käytännössä valmis heti sen luonnin yhteydessä. Edes ulkonäköasetuksia ei juuri tarvitse säätää, jos GUI Skin -asetustiedosto on säädetty oikein. Elementtien luonti on valituista lisäosista kaikkein helpointa ja intuitiivisinta.

IGUI:n koodin dokumentaatio on tehty suositulla Doxygen-dokumentaatiotyökalulla, joten se noudattaa yleisesti ohjelmoijille tuttuja käytänteitä. Sitä on siis helppo lukea, jos tekijällä on ohjelmointitaustaa. Dokumentaatiosta löytyy referenssit lähdekoodille. Käyttöliittymän toiminnan kannalta olennaista on hyödyntää erityisesti käytettävien luokkien julkisia muuttujia ja jäsenfunktioita.

6.2.2 Kontrolli

IGUI ei anna käyttäjälle niin paljon mahdollisuuksia vaikuttaa käyttöliittymäelementtien rakenteeseen kuin NGUI. Elementeissä on kuitenkin sisäänrakennettuna todella paljon asetuksia, joilla voi vaikuttaa sen toimintaan ja ulkonäköön. IGUI:n elementit tarjoavat tarpeeksi hyvän kontrollin tyypillisimpien käyttöliittymien toteuttamiseen.

IGUI:ssa aiheuttaa räätälöidyn käyttöliittymän tekemisessä ongelmia se, ettei GUI Skin -asetustiedostoon voi tallentaa kuin yhden mallipohjan kullekin elementtityypille. Siihen voi siis määrittää esimerkiksi yhdenlaisen button-elementin, ja jos käyttöliittymässä haluaa käyttää erilaisia painikkeita, täytyy niiden ulkonäkö määritellä elementtiä luodessa.

6.2.3 Peruselementit

IGUI:n peruselementit ovat valituista käyttöliittymälisäosista helpoimmat tehdä. Työkalu elementtien luomiseen on selkeä, ja luonti tapahtuu viemällä elementti haluttuun kohtaan ja sen koko määritellään hiirellä raahaamalla. Luodulle elementille voi halutessaan tehdä pientä säätämistä, mutta se ei periaatteessa tarvitse toimiakseen mitään ylimääräisiä toimenpiteitä.

Peruselementtien helppo ja intuitiivinen luonti on iGUI:n paras ominaisuus. Käyttöönoton helppous ja peruselementtien yksinkertainen rakenne tarkoittaa, että sillä on todella vaivatonta luoda yksinkertainen ja toimiva käyttöliittymä. IGUI sopiikin todella hyvin peliprojekteihin, joissa halutaan toteuttaa yksinkertainen käyttöliittymä mahdollisimman tehokkaasti.

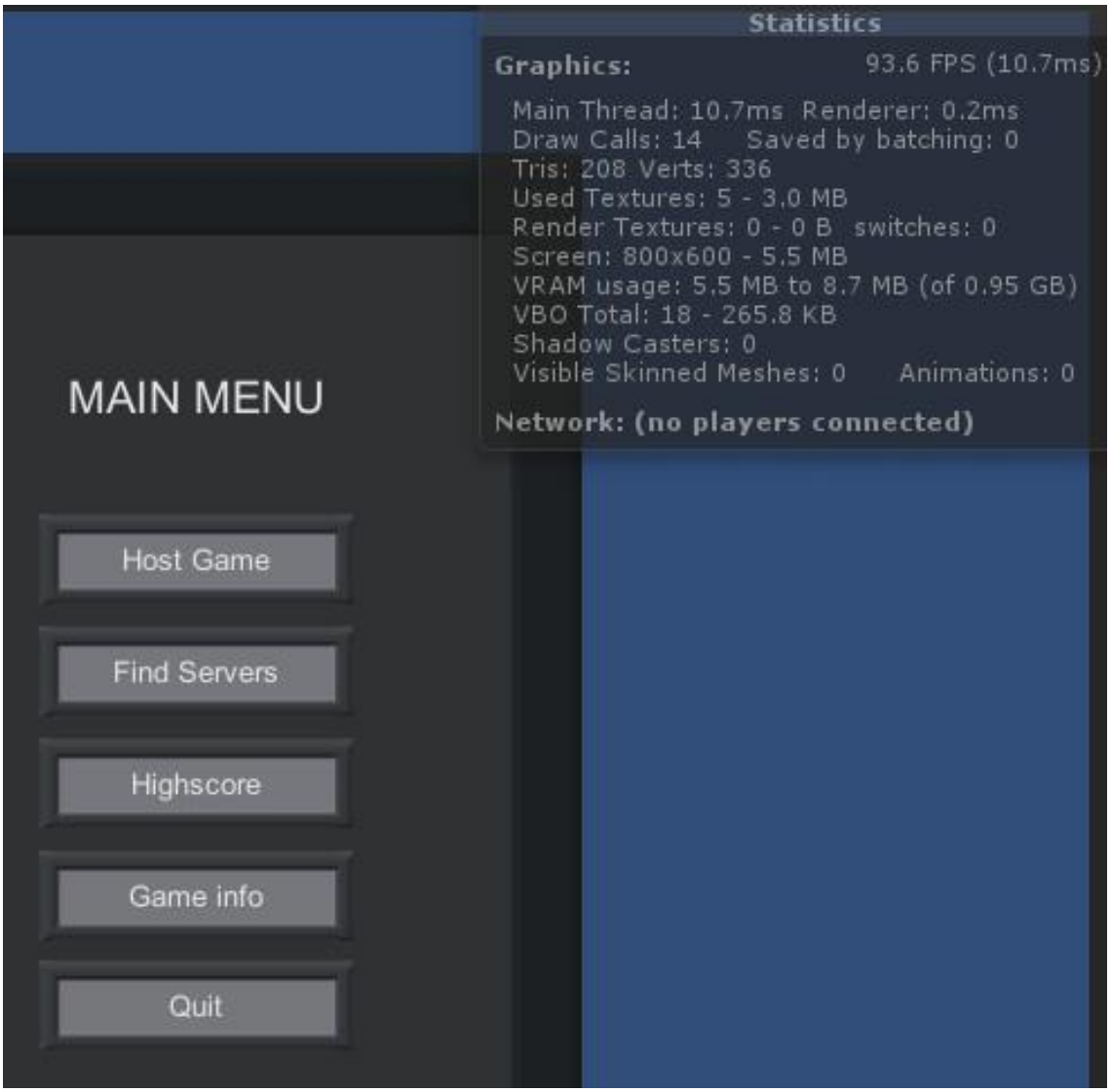
6.2.4 Monipuolisuus

IGUI:ssa on hyvä valikoima valmiita käyttöliittymäelementtejä. Siitä löytyy esimerkiksi erilaisia ikkunoita, kuten vieritettävät, siirrettävät ja useamman välilehden näkymät. Elementeistä löytyy esimerkiksi tekstin- ja numeroidensyöttö, listanäkymä, alasvetovalikko, vierityspalkkeja ja erilaisia kello- ja mittarinäkymiä. Elementtejä pystyy yhdistämään esimerkiksi niin, että eri elementtejä upotetaan vaikkapa vieritys- tai välilehtinäkymään.

IGUI:n heikkous on sen hyvin suljetussa toteutuksessa. Muutoksien tekeminen valmiisiin komponentteihin on vaikeaa, eikä onnistu ilman lähdekoodin muokkaamista. IGUI:n mukana ei myöskään tule esimerkkitiedostoja kuten NGUI:n tai Daikon Forgen mukana tulee. Se vaikeuttaa monipuolisten ominaisuuksien toteuttamista, koska ilman niitä ei ole yhtä helppo hahmottaa, miten niitä voisi lähteä tekemään.

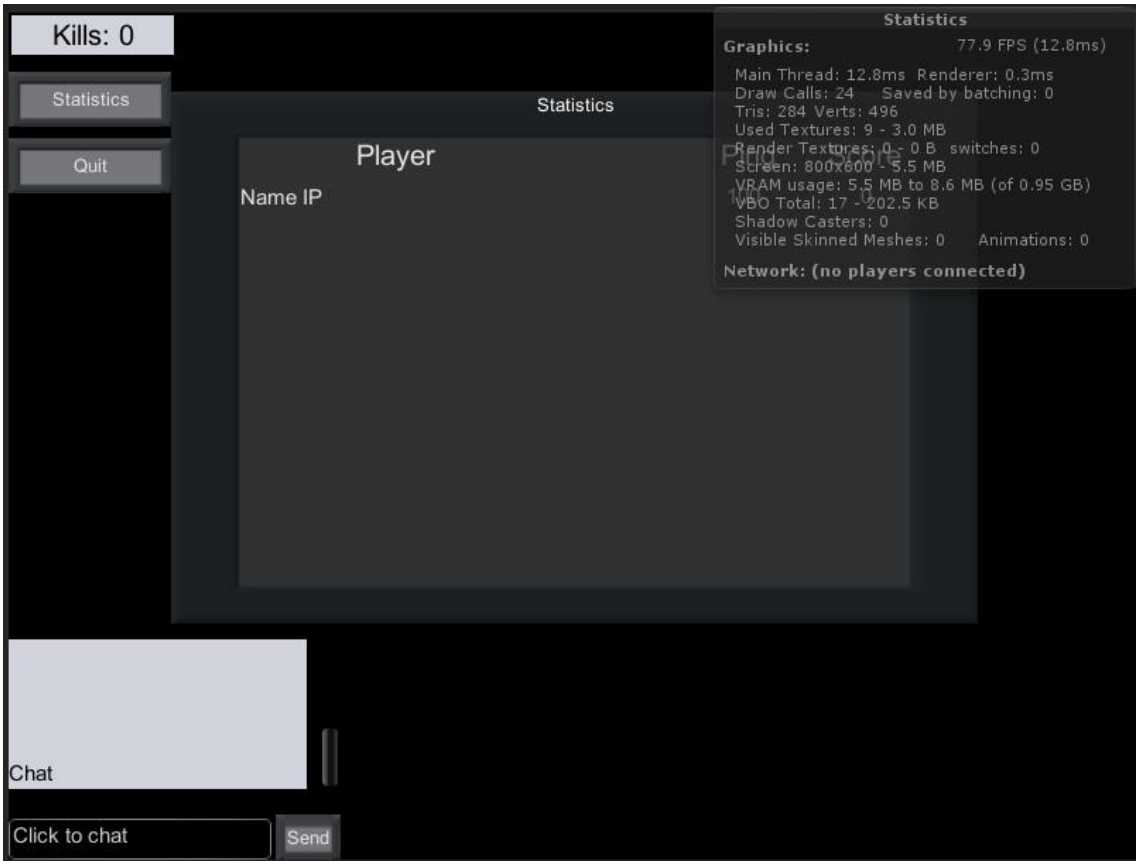
6.2.5 Suoritustehokkuus

IGUI:ssa ei ole mahdollisuutta käyttää tekstuuriatlasta, joten eri tekstuureja käyttäviä elementtejä ei ole mahdollista piirtää yhdellä piirtokutsulla. Siinä ei myöskään ole mahdollisuutta yhdistää samaa tekstuuria käyttäviä elementtejä yhteen piirtokutsuun. Se tarkoittaa käytännössä sitä, että iGUI:lla toteutettu käyttöliittymä käyttää yhtä monta piirtokutsua kuin Unityn OnGUI-luokalla toteutettu vastaava käyttöliittymäkin. Slaughter-pelin alkuvalikossa jokainen painike, taustakuva ja teksipätkä tarvitsee oman piirtokutsun (Kuva 26).



Kuva 26. Tilastoja iGUI:lla toteutetun alkuvalikon suoritustehokkuudesta (Kuva: Niko Sarkkinen).

Pelinäkymässä jokainen elementti piirretään myös erikseen (Kuva 27). Se on käyttöliittymän toteuttamista helpottava asia, koska sitä tehdessä ei päällekkäin menevien elementtien kohdalla tarvitse huolehtia syvyysarvoista tai graafisista virheistä. Piirtokutsujen optimoinnin puute on kuitenkin selkeä haittapuoli NGUI:hin ja Daigon Forgeen verrattuna, eikä tuo etua Unityn OnGUI-luokkaan verrattuna.



Kuva 27. Tilastoja iGUI:lla toteutetun pelinäkymän suoritustehokkuudesta (Kuva: Niko Sarkkinen).

6.3 Daikon Forge GUI Library

Daikon Forge on käytettävyydeltään NGUI:n ja iGUI:n välillä. Sen elementit ovat valmiita ja suljettuja kokonaisuuksia, joten ne on helppo toteuttaa kuten iGUI:ssa. Sen työkalut eivät kuitenkaan ole aivan yhtä intuitiivisia kuin iGUI:n. Muuten nämä kaksi ovat monella tapaa hyvin samantuntuisia

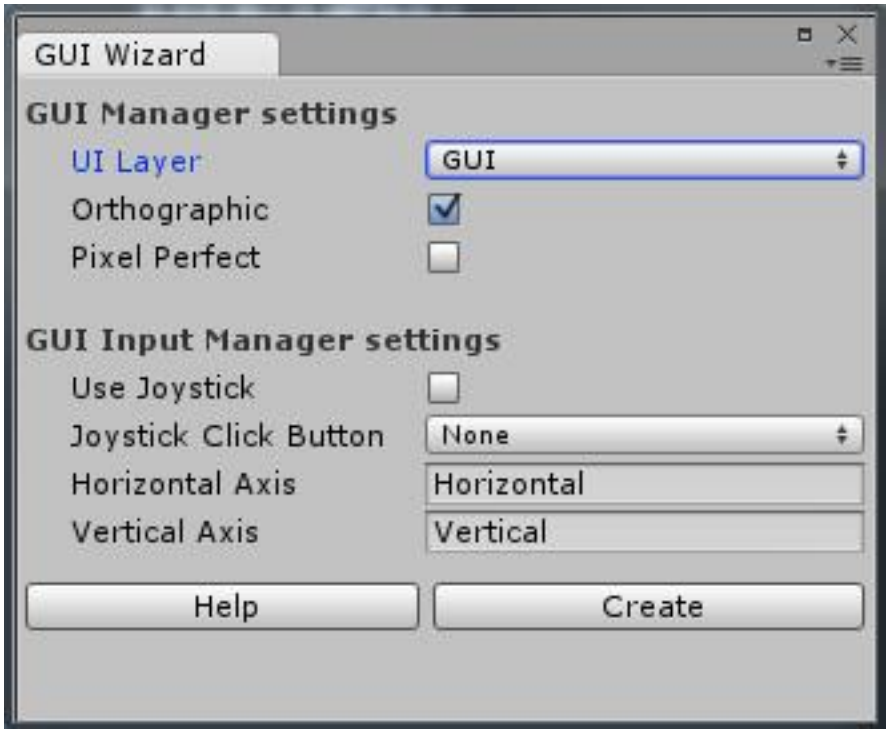
käyttöliittymälisäosia.

6.3.1 Käyttöönotto ja opettelu

Daikon Forgen käyttöönotto on yleisesti ottaen helppoa, mutta tekstuuriatlaksen luomisen vaikeudesta johtuen se ei ole ihan yhtä helppoa kuin muilla valituilla käyttöliittymälisäosilla. Daikon Forgen elementit ovat suljettuja kokonaisuuksia kuten iGUI:ssa, joten yksinkertaisen käyttöliittymän tekeminen onnistuu helposti.

Daikon Forgessa ei ole työkalua tekstuuriatlaksen luontiin, mutta se tukee tekstuuriatlasta. Tekstuuriatlas täytyy luoda jossain kolmannen osapuolen ohjelmassa ja tallentaa se Unityn tunnistamassa tiedostoformaatissa. Luotu tekstuuri ja siihen liittyvä data-tiedosto tuodaan Unityyn, jossa käytettävä atlas luodaan Unityn prefabiksi. Prosessissa on siis monta vaihetta, ja se voi tuntua vaikealta varsinkin, kun se on yksi ensimmäisiä vaiheita uuden käyttöliittymän tekemisen aloittamisessa.

Daikon Forgessa on yksinkertainen työkalu, joka avustaa uuden käyttöliittymän luonnissa (Kuva 28). Työkalussa määritellään vain käyttöliittymän tärkeimmät yleisasetukset, ja se helpottaa huomattavasti olennaisten asetusten hahmottamista. Näiden asetusten muokkaaminen myöhemmin sekä muiden hienosäätöjen määritteleminen on mahdollista käyttöliittymän GuiManager-skriptissä. Asetuksia voi muokata myös myöhemmin, jos sille huomaa tarvetta.



Kuva 28. Daikon Forgen työkalu uuden käyttöliittymän luontiin (Kuva: Niko Sarkkinen).

Daikon Forgessa elementit luodaan etsimällä ne valikkorakenteesta. Luotu elementti on toiminnallisuudeltaan heti valmis, ja sille tarvitsee määritellä vain ulkonäköasetukset. Elementtien luonti on siis huomattavasti helpompaa kuin NGUI:ssa, mutta ei aivan yhtä intuitiivista kuin iGUI:ssa.

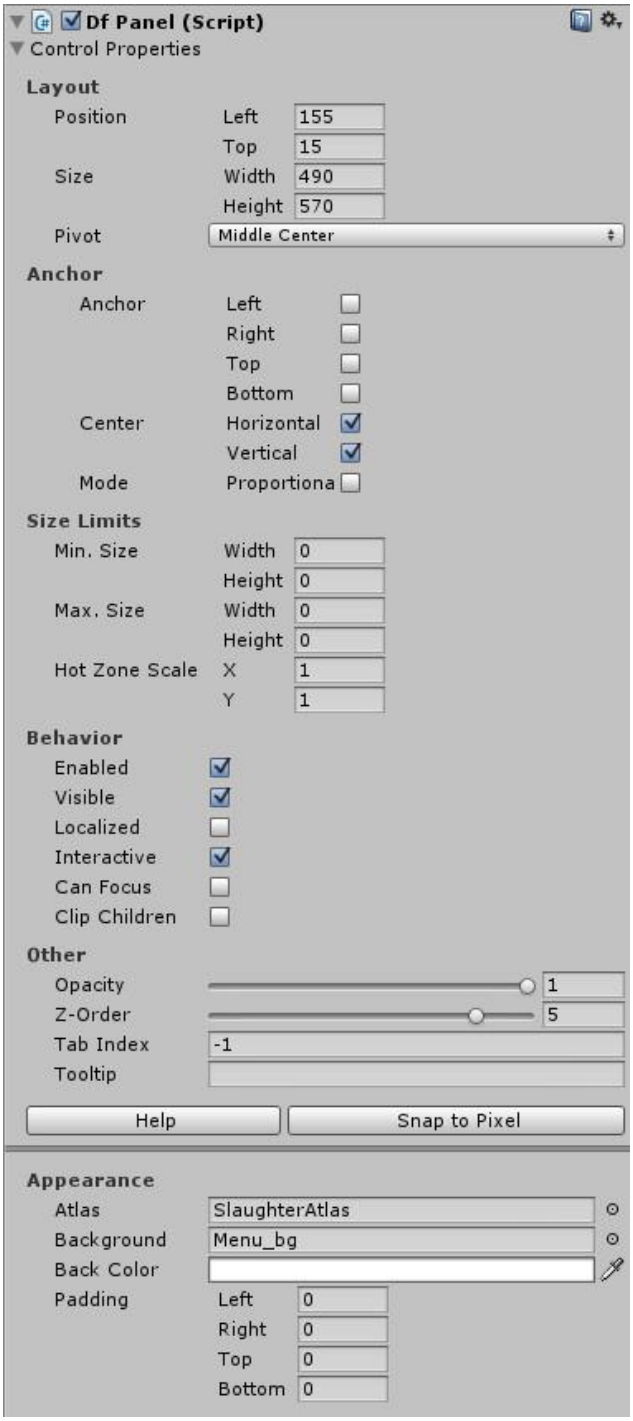
NGUI:n tapaan Daikon Forgen mukana tulee esimerkkitiedostoja, joista voi katsoa, kuinka joitain monimutkaisempia toimintoja on toteutettu. Esimerkkien tutkiminen on hyvä tapa oppia nopeasti, miten lisäosalla voi toteuttaa erilaisia ominaisuuksia. Niistä löytyy myös lisäosan lähdekoodiin kuulumattomia skriptejä, joista voi olla apua omien toiminnallisuuksien toteuttamisessa.

Daikon Forgen koodin dokumentaatio on tehty suositulla Doxygen-dokumentaatiotyökalulla, joten se noudattaa yleisesti ohjelmoijille tuttuja käytänteitä. Sitä on siis helppo lukea, jos tekijällä on ohjelmointitaustaa. Dokumentaatiosta löytyy referenssit lähdekoodille ja esimerkkitiedostoissa käytetetyille malliskripteille.

6.3.2 Kontrolli

Daikon Forgen käyttöliittymäelementtien rakenne muistuttaa paljon iGUI:n elementtien rakennetta. Ne ovat suljettuja kokonaisuuksia, eikä niitä tarvitse rakentaa itse monesta eri komponenteista kuten NGUI:ssa. Daikon Forgen elementtien asetukset on selkeäämmin jäsennelty ja olennaiset säädöt on helpompi löytää kuin iGUI:ssa.

Daikon Forgen eri elementeissä on paljon yhteisiä asetuksia. Sitä voi hyödyntää asettamalla elementtejä niin, että ne saavat ominaisuuksia toisiltaan. Esimerksi Label-elementtiin voi laittaa päälle interaktiivisuuden, jolloin voidaan hyödyntää sitä, jos pelaaja vaikkapa klikkaa elementtiä. Näin Label-elementti saa Button-elementin ominaisuuksia.



Kuva 29. Daikon Forgen elementtien asetuksia (Kuva: Niko Sarkkinen).

6.3.3 Peruselementit

Elementtien luonti on Daikon Forgessa helppoa. Niiden luonti tapahtuu valikkorakenteen kautta, joten se on yksinkertaista, vaikkakin aluksi hieman

sekava elementtien suuren määrän vuoksi. Elementit ovat iGUI:n tapaan suljettuja kokonaisuuksia. Ne sisältävät elementin toimintaan kaiken tarvittavan, ja elementti on valmis ulkonäköasetusten määrittämisen jälkeen.

Daikon Forgessa on samat helppokäyttöiset peruselementit kuin muissakin valituissa lisäosissa, joten yksinkertaisen käyttöliittymän tekeminen on yleisesti ottaen todella helppoa. Daikon Forge sopii siis hyvin peliprojekteihin, joissa halutaan toteuttaa yksinkertainen käyttöliittymä tehokkaasti. Se on helpompi käyttää kuin NGUI, ja iGUI:n tekstuuriatlaksen puute voi monelle olla ratkaiseva tekijä Daikon Forgen hyväksi.

6.3.4 Monipuolisuus

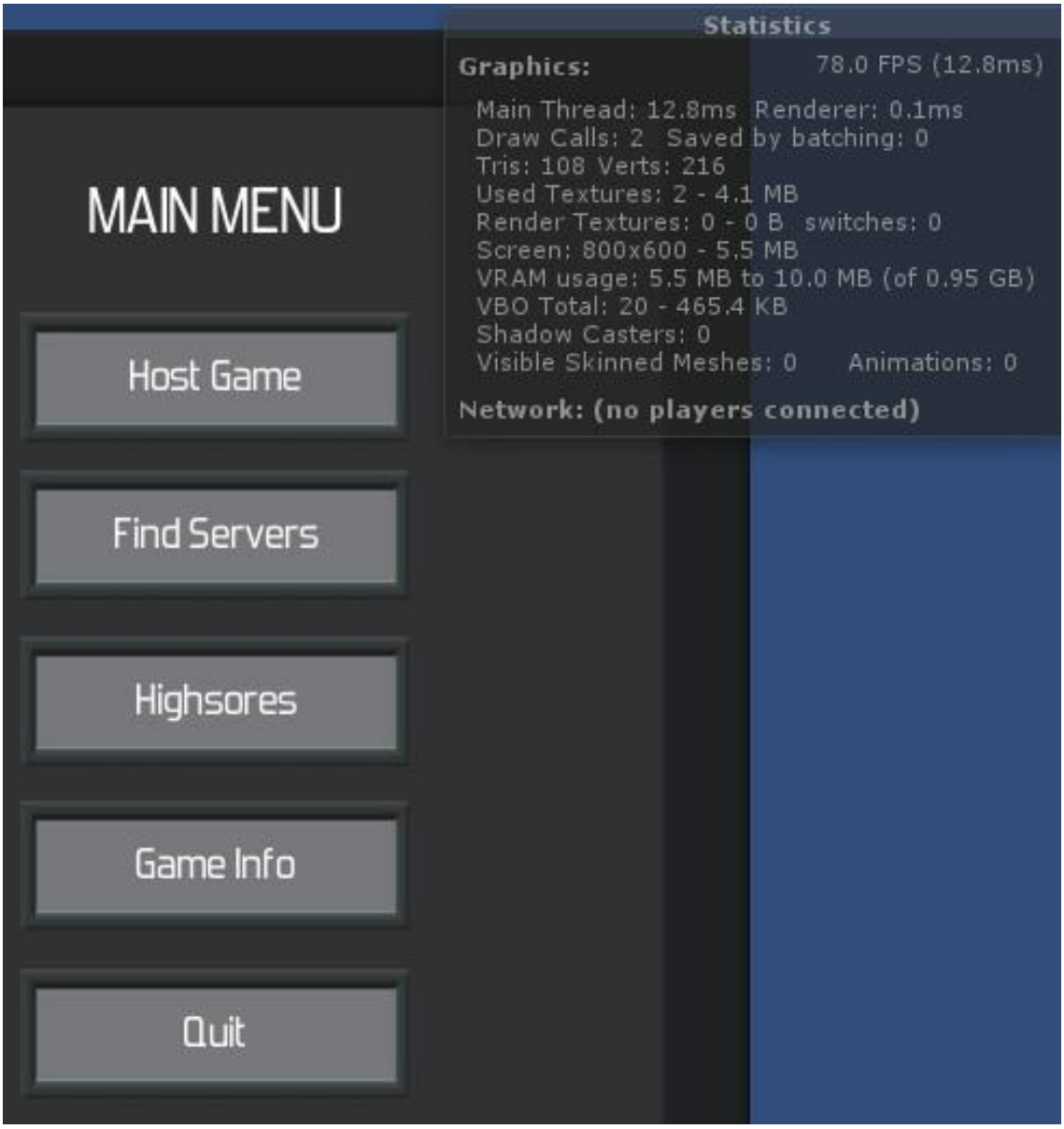
Daikon Forgesta löytyy kattava valikoima valmiita ja monipuolisia käyttöliittymäelementtejä. Paneelin lisäksi löytyy vieritettävä paneeli ja välilehtinäkymä, johon saa tehtyä useamman paneelin. Erilaisia valinta- ja listanäkymiä on hyvä valikoima. Pelin ollessa käynnissä pystyy elementtien ja paneeleiden sijaintia ja kokoa muuttamaan siihen tarkoitetuilla hallintaelementeillä. Erilaisia vierityspalkkeja ja säätimiä löytyy myös, jos sellaisille on tarvetta.

Daikon Forgen elementit ovat iGUI:n tapaan suljettuja kokonaisuuksia. Vaikka se helpottaa elementtien tekemistä ja niiden hallintaa, rajoittaa se monipuolisten ominaisuuksien toteuttamista. Yhdistelemällä elementtejä eri tavoilla ja hyödyntämällä niiden asetusten mahdollisuuksia voi kuitenkin toteuttaa monipuolisia ja mielenkiintoisia ratkaisuja.

Daikon Forgen mukana tulee esimerkkitiedostoja, joista löytyy valmiita ratkaisuja toteuttaa erilaisia elementtejä tai ominaisuuksia. Niistä löytyy esimerkkejä aina yksinkertaisten peruselementtien tehokkaasta hyödyntämisestä monimutkaisten kokonaisuuksien toteuttamiseen. Esimerkeistä voi ottaa suoraan mallia tai poimia ideoita omaan toteutukseen.

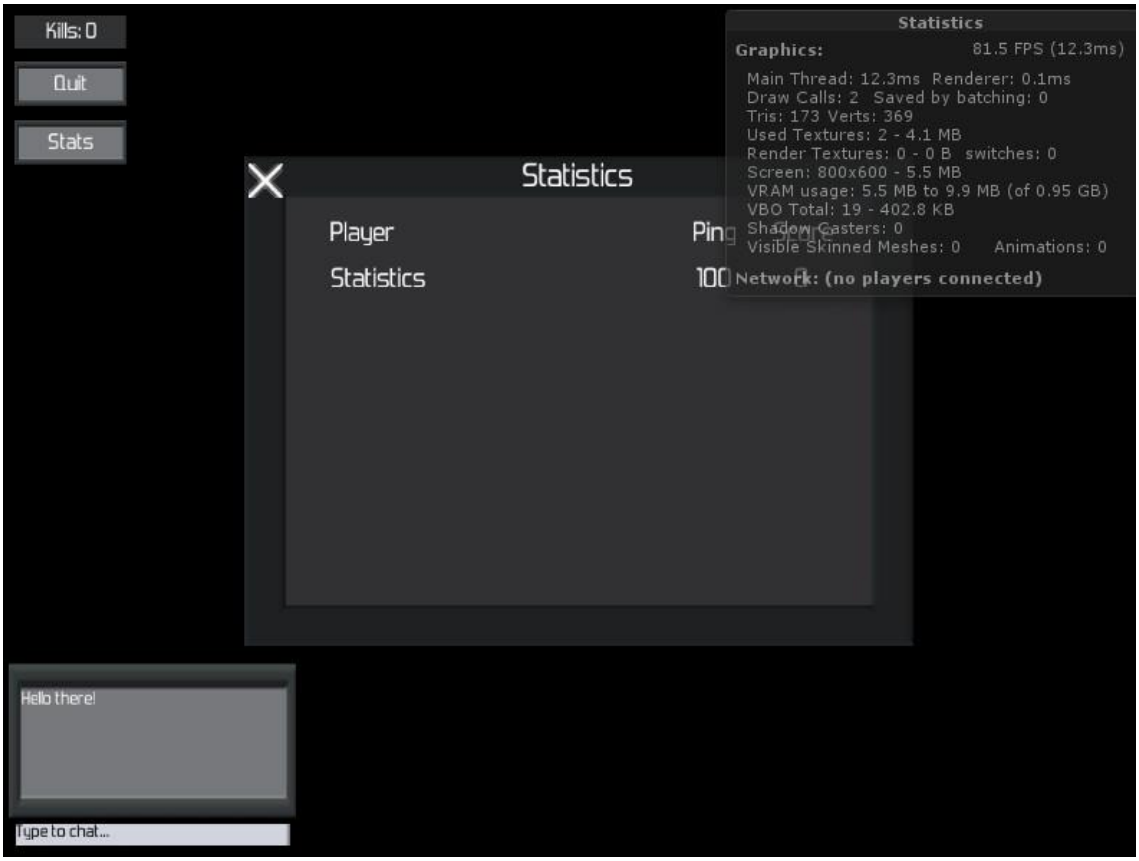
6.3.5 Suoritustehokkuus

Daikon Forgella toteutettu alkuvalikko käyttää vain kaksi piirtokutsua jokaisella ruudunpäivityskierroksella (Kuva 30). Kaksi piirtokutsua syntyy, koska käyttöliittymässä käytetään kahta tekstuuriatlasta. Toinen atlaksista sisältää kaikki tekstuurit ja toinen fontin. Periaatteessa piirtokutsut voitaisiin pudottaa yhteen, mutta tekstuurien ja fontin yhdistämineen samaan tekstuuriin ei ole käytännöllistä.



Kuva 30. Tilastoja Daikon Forgellalla toteutetun alkuvalikon suoritustehokkuudesta (Kuva: Niko Sarkkinen).

Pelinäkymässä (Kuva 31) päästään myös kahteen piirtokutsuun. Vaikka ruudulla on paljon eri elementtejä, ei kahteen piirtokutsuun pääseminen vaadi mitään erityistä säätämistä. Daikon Forge yhdistää kaikki samaa tekstuuriatlasta käyttävät elementit yhteen piirtokutsuun. Vaikka se onkin erittäin yksinkertainen tapa toteuttaa piirtokutsujen optimointi, voi se aiheuttaa ongelmia, jos eri tekstuuriatlasta käyttäviä elementtejä halutaan sijoitella limittäin. Silloin jokin osa elementistä saatetaan piirtää viimeisimpänä muiden osien päälle vaikka sen olisi tarkoitus olla fyysisesti taimmainen.



Kuva 31. Tilastoja Daikon Forgella toteutetun pelinäkymän suoritustehokkuudesta (Kuva: Niko Sarkkinen).

7 Yhteenveto

Tässä luvussa tehdään yhteenveto käyttöliittymälisäosista. Niitä vertaillaan keskenään erityisesti tämän opinnäytetyön tutkimuskysymykset mielessä.

7.1 Käyttöönotto ja käytettävyys

Käyttöliittymälisäosien käyttöönotossa oli eri lisäosilla paljon yhteistä. NGUI ja Daikon Forge vaativat tekstuuriatlaksen toimiakseen. NGUI:n atlastyökalu oli käyttökokemuksen perusteella selkeästi helppokäyttöisempi kuin Daikon Forgen. IGUI ei käytä tekstuuriatlasta, mutta se vaatii GUI Skin -asetustiedoston käyttöönoton. Sen luominen ja asetusten määrittäminen oli helppoa, ja koska sen asetuksia oli mahdollista muokata missä vaiheessa käyttöliittymän tekoa vain, ei se juuri hidastanut käyttöönottoa.

Uuden käyttöliittymän luonti on jokaisella lisäosalla helppoa. Se tapahtui kaikissa käytännössä nappia painamalla. Uuden käyttöliittymän asetukset olivat kaikissa lisäosissa myös selkeät ja niitä pystyy halutessaan muokkaamaan myöhemmin ilman, että jo tehty käyttöliittymä menee täysin hukkaan.

Käyttöliittymän peruselementtien tekemisessä oli eri lisäosilla jo huomattavia eroja. NGUI:n elementit koostuvat monesta komponentista, jolloin yksittäisiä komponentteja lisäämällä tai poistamalla saa elementeistä juuri halutunlaisen. Se vaikeuttaa kuitenkin NGUI:n opettelua, mikä tekeekin siitä käytetyistä lisäosista vaikeimman opetella ja käyttää. Daikon Forgen ja iGUI:n elementit ovat rakenteeltaan isompia suljettuja kokonaisuuksia. Se tekee niiden luomisesta ja hallinnasta paljon helpompaa, koska käyttäjän ei tarvitse miettiä mitä eri komponentteja mikäkin elementti tarvitsee. Käyttökokemuksen perusteella käyttöliittymän tekeminen on iGUI:lla hieman Daikon Forgea helpompaa intuitiivisten työkalujen ansiosta. Molemmat ovat silti selkeästi NGUI:ta helppokäyttöisempiä.

Kaikista lisäosista löytyy Doxygenillä toteutettu dokumentaatio. Lisäosan käytön opettelussa monelle ovat kuitenkin tärkeämpiä erilaiset tutoriaalit sekä esimerkkitiedostot, joista saa mallia perusasioiden tekemiseen. NGUI:n ja Daikon Forgen mukana tulee kattava valikoima esimerkkitiedostoja, joissa on toteutettu erilaisia käyttöliittymäratkaisuja. Ne ovat todella hyvä resurssi, kun lisäosan toimintaa opetellaan. Kaikilla lisäosilla on myös selkeä ”Getting Started” -tutoriaalisivusto, joka auttaa aloittelevia käyttäjiä opettelemaan lisäosan käyttöä. IGUI:n selkeä heikkous verrattuna muihin lisäosiin on esimerkkitiedostojen puute.

Käyttöliittymälisäosien käyttöä peliprojektissa ei voi perustella pelkästään käyttöönoton helppoudella tai käytettävyydellä. Unityn oman GUI-luokan käyttöönotto ei vaadi mitään erityisiä toimenpiteitä. Siinä on myös hyvä valikoima yleisimpien perustoimintojen tekemiseen, vaikkakin niiden teko vaatii aina jonkin verran ohjelmointia. Unityn dokumentaatio on yleisesti todella hyvä, ja sieltä löytyy normaalin koodilistauksen lisäksi esimerkkejä käytännön toteutuksista. GUI-luokka ei ole siinä poikkeus, joten se on myös siinä asiassa vähintään tasoissa lisäosien dokumentaatioiden kanssa.

7.2 Monipuolisuus ja kontrolli

Käyttöliittymälisäosien selkeä etu Unityn GUI-luokkaan on edistyneempien käyttöliittymäelementtien toteuttaminen. Kaikissa lisäosissa on hyvä valikoima erilaisia edistyneempiä elementtejä, kuten vieritettäviä ikkunoita, vierityspalkkeja tai alasvetovalikoita. GUI-luokassa ei ole yhtä hyvää valikoimaa näiden toteuttamiseen, ja niiden tekeminen vaatii paljon ohjelmointia.

Jokaisella lisäosalla voitiin toteuttaa määritelmän mukainen käyttöliittymä. Suurin ero eri lisäosien välillä oli listanäkymän tekemisessä. Daikon Forgella ja iGUI:lla listanäkymän teko oli helppoa, mutta NGUI:lla joutui turvautumaan esimerkkitiedoston ratkaisuun, joista poimittiin omaan ratkaisuun listanäkymälle

olennaiset skriptit.

NGUI:n etu muihin lisäosiin verrattuna on sen tapa muodostaa elementit useasta komponentista. Se tarjoaa parhaan kontrollin tehdä käyttöliittymästä juuri halutunlainen. Vaikka se tuntuu vaikealta opetella ja ehkä hieman turhauttavalta peruselementtejä tehdessä, tulevat sen parhaat puolet esille monimutkaisempia elementtejä tehtäessä. Daikon Forge ja iGUI ovat hyvin pitkälle valmiiden elementtien varassa, ja niiden rakenteen merkittävä muokkaaminen ei ole mahdollista.

7.3 Suoritustehokkuus

Valituista käyttöliittymälisäosista NGUI ja Daikon Forge käyttävät tekstuuriatlasta sekä mahdollistavat samaa materiaalia käyttävien elementtien piirtokutsujen yhdistämisen. IGUI ei käytä tekstuuriatlasta, joten sillä piirtokutsujen optimointi ei ole mahdollista.

Daikon Forgella tekstuuriatlaksen käyttöönotto oli huomattavasti hankalampaa kuin NGUI:lla. Piirtokutsujen optimointi on sillä kuitenkin paljon yksinkertaisempaa kuin NGUI:lla. Daikon Forgella tarvitsee vain laittaa piirtokutsujen yhdistäminen päälle, ja lisäosa yhdistää automaattisesti kaikki samaa tekstuuria käyttävät elementit. NGUI:lla piirtokutsujen optimointi on huomattavasti tarkempi toimenpide, jossa elementtien syvyydet täytyy ottaa huomioon. NGUI:ssa on kuitenkin hyvä työkalu siihen tarkoitukseen, ja se mahdollistaa rakenteeltaan monimutkaisen käyttöliittymän toteuttamisen ilman, että elementtien yhdistäminen aiheuttaa graafisia virheitä.

Tekstuuriatlaksen ja piirtokutsujen optimoinnin tuoma hyöty suoritustehokkuudessa riippuu aina pelistä ja erityisesti alustasta, jolle peliä kehitetään. NGUI ja Daikon Forge ovat parhaat valinnat, jos suoritustehokkuus on omassa peliprojektissa tärkeä valintakriteeri. IGUI on aiemmin todettu helppokäyttöisimmäksi lisäosaksi, mutta se ei anna suoritustehokkuudessa etua

Unityn GUI-luokkaan vertaillessa.

7.4 Vertailutaulukko

Taulukossa 2 vertaillaan testattuja käyttöliittymälisäosia numeerisesti. Arvioinnit on annettu asteikolla 1–3, missä 3 on paras ja 1 heikoin. Arvosanat perustuvat täysin subjektiiviseen käyttökokemukseen. Arvioinnit on jaoteltu aiemmin esiteltyjen vertailukohtien mukaan. Arvosanoista ei voi suoraan päätellä, mikä lisäosista on absoluuttisesti paras, vaan kannattaa arvioida, mitkä ominaisuudet ovat tärkeimmät omalle peliprojektille, ja tehdä päätelmät sen mukaan.

Nimi	NGUI	Daikon Forge	iGUI
Yleistä			
Kehittäjä	Tasharen Entertainment	Daikon Forge	Avam Studios
Hinta	95$	Opensource	65$
Teknistä tietoa			
Tekstuuriatlas	Kyllä	Kyllä	Ei
Piirtokutsujen optimointi	Kyllä	Kyllä	Ei
Vertailutulosten yhteenveto			
Käyttöönotto ja opettelu			
Intuitiivisuus	1	2	3
Dokumentaatio	2	2	2
Tutoriaalit	3	2	1
Kontrolli			
Muokattavuus	3	2	2
Peruselementit			
Valikoima	3	3	3
Työnkulku	2	3	3
Monipuolisuus			
Valikoima	3	2	2
Hallinta	2	2	3
Yht.	**20**	**18**	**19**

Taulukko 2. Käyttöliittymälisäosien vertailutaulukko.

Käyttöönotto ja opettelu erosi lisäosien välillä merkittävästi. IGUI oli intuitiivisin, mutta muille lisäosille oli paremmin tarjolla ohjeistusta. Dokumentaatiossa en huomannut eroa lisäosien välillä. Valmiiden elementtien muokkaaminen onnistui parhaiten NGUI:lla. Daikon Forgen ja iGUI:n elementtien muokkaaminen on niistä löytyvien asetusten varassa, mutta niitä on molemmissa kuitenkin tarpeeksi useimpiin tapauksiin. Peruselementtien valikoimassa ei ollut mitään eroa ja yksinkertaisen käyttöliittymän toteuttaminen onnistui kaikilla lisäosilla. Peruselementtien tekeminen ja hallitseminen oli NGUI:lla hieman muita hankalampaa. Monipuolisia elementtejä on NGUI:ssa mahdollista kasata useammasta komponentista itse, joten mahdollisuudet ovat lähinnä tekijästä kiinni. Muissa lisäosissa valikoima oli hyvä, ja elementtien tekeminen oli helppoa. IGUI:ssa monimutkaistenkin rakenteiden tekeminen ja hallinta oli helpointa. NGUI:ssa on hyvät työkalut erityisesti suoritustehokkuuden optimointiin, mutta edistyneempien elementtien ja ominaisuuksien hyödyntäminen on sillä työläintä. Daikon Forgessa monipuolisten elementtien toteuttaminen ja hallinta on pääosin vaivatonta, mutta sitä vaivaa hieman niissä olevat ohjelmointivirheet.

8 Pohdinta

Tässä luvussa käydään läpi tämän opinnäytetyön vaiheita ja pohditaan, mitä opin työn aikana. Työssä on aiemmin keskitytty käyttöliittymälisäosien etuihin, ja tässä luvussa pohditaan, onko käyttöliittymälisäosan hankkiminen peliprojektiin aina kannattavaa.

8.1 Toteutus

Opinnäytetyön käytännön osuudessa toteutin käyttöliittymän kolmella valitulla lisäosalla. Unity päätyi käytettäväksi kehitystyökaluksi toimeksiantajan toiveen

vuoksi. Toimeksiantaja antoi myös hyvin selkeän määritelmän käyttöliittymälle, joten työn aloittaminen oli selkeällä pohjalla. Ensimmäinen iso päätöksentekotilanne oli päättää, millä lisäosilla tulen käyttöliittymät toteuttamaan. Päätös syntyi pikkuhiljaa, kun luin internetistä lisäosien käyttäjien arvioita ja kun tein eri lisäosalla yksinkertaisia asioita. Aluksi hieman arvelutti, osuivatko valinnat oikein, mutta työn edetessä on päätös tuntunut koko ajan oikeammalta. Valitut lisäosat olivat mielestäni tarpeeksi erilaisia ja edustivat erilaisia lähestymistapoja käyttöliittymälisäosien toteuttamiseen. Käyttäjäarvosteluista sai kuvan, että ne olivat hyvin pidettyjä ja toimivia lisäosia, ja omat kokemukset tukivat tätä näkemystä.

Käyttöliittymien toteutuksesta ja tutoriaalien tekemisestä saatu kokemus oli hyvä pohja, kun aloin kirjoittamaan niistä vertailuja. Määritelmän mukaisen käyttöliittymän tekeminen antoi minulle mahdollisuuden tehdä sellaisia ominaisuuksia, joita uskoisin monen lisäosan käyttöä opettelevan käyttäjän miettivän. Olin siis mielestäni hyvin samankaltaisessa tilanteessa, missä moni käyttöliittymälisäosan hankkimista miettivä henkilö on.

Työn aikana olen oppinut todella paljon eri käyttöliittymälisäosista, ja niiden erilaiset lähestymistavat samaan asiaan ovat avartaneet näkemystä eri ominaisuuksien toteuttamismahdollisuuksista. En ole aiemmin tehnyt mitään niin isoa tutoriaalia, kuin tässä työssä tein. Siihen liittyvän työn määrä hieman yllätti, siitäkin huolimatta, että olin siihen varautunut. Myös vertailun tekeminen tässä mittakaavassa oli uutta, ja oli mielenkiintoista miettiä, mitkä asiat lukijaa saattaisivat kiinnostaa. Siinäkin asiassa helpotti se, että olin itse ollut lukijan asemassa opinnäytetyön alkuvaiheessa.

8.2 Käyttöliittymälisäosien hyöty

Käyttöliittymän tekemiseen on perinteisesti vaadittu suuri työpanos ohjelmoijilta. Käyttöliittymälisäosilla ohjelmointityön määrää saadaan pienennettyä ja ne resurssit voidaan suunnata muun pelimekaniikan toteuttamiseen.

Peliprojekteissa tyypillisesti ohjelmoijien työmäärä on iso ja se kasvaa varsinkin projektin loppua kohden. Ohjelmointitehtävien vähentäminen ja osittain siirtäminen graafikoiden hoidettaviksi onkin projektin ajankäytön kannalta aina suotavaa. Helppokäyttöiset käyttöliittymälisäosat tarjoavat tähän mahdollisuuden.

Käyttöliittymälisäosan ostamista voidaan perustella myös säästetyn työn määrällä. Lisäosien hinta on keskimäärin 50–100 € työasemaa kohden. Peliprojektissa palkallinen ohjelmoija ei saisi käyttää kovin montaa tuntia käyttöliittymän ohjelmointiin ennen kun tuo summa tulee vastaan. Kun otetaan huomioon se määrä työtunteja mitä maksullisten lisäosien ohjelmointiin on mennyt, voidaan pitää todennäköisenä, ettei yhtä tasokasta käyttöliittymäratkaisua saada tehtyä muutamassa tunnissa. Käyttöliittymälisäosat tarjoavat siis hinta-laatusuhteeltaan erinomaisen vaihtoehdon täysin itsekoodatulle käyttöliittymälle.

Aina lisäosan hankkiminen ei kuitenkaan ole järkevää. Käyttöliittymälisäosien käyttöönottoon ja käytön opetteluun menee huomaamatta paljon aikaa. Lisäosat eivät ole mainospuheista huolimatta aina niin helppo ottaa käyttöön kuin voisi luulla. Tästä syystä oman käyttöliittymän ohjelmointi alusta asti ei välttämättä ole huono idea. Jos käyttöliittymä on suunniteltu hyvin, voidaan siihen tarvittavat ominaisuudet ohjelmoida tehokkaasti. Tässä tapauksessa koodi on todennäköisesti myös aika tehokasta, koska turhia ominaisuuksia ei mukana ole. Jos käyttöliittymä on kuitenkin heikosti suunniteltu, kasvaa tarve tehdä käyttöliittymään muutoksia jossakin vaiheessa. Silloin tarvitaan ohjelmoijaa tekemään muutokset, mikä taas ole kustannustehokasta.

Käyttöliittymälisäosissa on yleensä hyvä pohja tehdä tietynlainen käyttöliittymä. Monesti peliin voidaan kuitenkin haluta tehdä toisenlainen käyttöliittymä. Jos suunniteltu käyttöliittymä eroaa merkittävästi siitä, millaisia käyttöliittymiä varten lisäosa on tehty, voi työmäärä kasvaa todella paljon. Siinä tilanteessa ohjelmoijan on otettava selvää, miten lisäosa on tehty, ja muokattava sitä vaatimukset täyttäväksi. Tällainen tilanne voi syödä kaikki lisäosan hankkimista puoltaneet edut. Pahimmillaan käyttöliittymälisäosan räätälöinti halutunlaiseksi

voi viedä jopa enemmän resursseja kuin oman käyttöliittymän ohjelmointi alusta asti. Tällaisten tilanteiden välttämiseksi onkin hyvä tuntea lisäosien tuomat mahdollisuudet ja verrata niitä omiin suunnitelmiin ja vaatimuksiin.

8.3 Työn tulokset ja jatkokehitysideat

Työstä tuli kaksi hyödynnettävissä olevaa kokonaisuutta. Toimeksiantaja tulee hyödyntämään käytännön osuudessa toteutettuja käyttöliittymälisäosien käyttöohjeita opetuskäytössä. Lisäosien ominaisuuksien vertailusta voi myös olla hyötyä jollekin, joka asiaa pohtii tulevaisuudessa. Kun suunnittelin käytännön osuutta ja valitsin käytettäviä lisäosia, yritin löytää vastaavaa useamman lisäosan vertailua, mutta sellaista ei löytynyt. Koitin etsiä vertailuja Unityn foorumeilta, Unity-käyttäjien blogeista, tieteellisistä julkaisuarkistoista kuten Theseuksesta ja yleisesti internetistä hakukonetta käyttämällä. Tämä tutkimus toivottavasti auttaa jotakuta, joka on tekemässä aiheeseen liittyvää tutkimustyötä. Näkisin, että tulokset kiinnostaisi myös jonkin lisäosan hankkimista pohtivaa Unity-käyttäjää.

Työssä toteutettua lisäosien vertailua voi käyttää pohjana lisäosan valintaan. Kaikilla lisäosilla on omat vahvuutensa ja oman peliprojektin tarpeet tulee ottaa valintaa tehdessä huomioon. Oma arvioni kuitenkin on, että NGUI on tehokkuutensa ja monipuolisuutensa vuoksi paras valinta. Se on oman kokemukseni perusteella vaikein opetella, mutta jos sen suhteellista vaikeutta ei pidetä ongelmana, on se muilla osa-alueilla mitattuna paras. Kun sitä on opetellut käyttämään, ovat sen käyttömahdollisuudet laajemmat kuin kahdella muulla lisäosalla. Daikon Forge tai iGUI tulevat mukaan pohdintaan, jos tärkein kriteeri lisäosaa valittaessa on helppoudesta johtuva ajan säästäminen. Tämä arvio myös osuu mielestäni hyvin yhteen työn alussa etsittyjen käyttäjäarvostelujen kanssa. NGUI:n hyvyys selittää myös, miksi se on lisäosista ylivoimaisesti suosituin. Työn tulosta voidaan toki aina kyseenalaistaa ja kritisoida. Suurin peruste kritiikille olisi varmastikin vain yhdenlaisen käyttöliittymän toteuttaminen ja nojautuminen yhden henkilön

käyttökokemuksiin.

Slaughter-pelin käyttöliittymämäärittelyssä oli listattu ominaisuuksia, joita käyttöliittymässä tuli olla. Ominaisuudet oli mahdollista tehdä käyttämällä pääosin käyttöliittymälisäosien peruselementtejä. Se auttoi lisäosien perusominaisuuksien ja työnkulun sujuvuuden arvioinnissa, mutta lisäosien syvällisemmän vertailun helpottamiseksi myös monipuolisempien ominaisuuksien toteuttaminen olisi ollut hyödyllistä. Ominaisuudet joilla olisi mielestäni saanut enemmän eroa lisäosien välille ovat ainakin erilaiset ikkunanäkymät kuten välilehdet tai raahattavat ja kokoa muuttavat ikkunat. Myös elementtien animointi olisi mielestäni tuonut lisää vertailukohtia lisäosien välille.

Yleensä eri käyttöliittymälisäosia vertaillessa arviot keskittyvät pitkälti teknisten ominaisuuksien vertailuun. Vertailtavista lisäosista listataan niissä olevia ominaisuuksia ja lukijan on muodostettava mielipide sen perusteella. Tästä on hyvä esimerkki Ogre-pelinkehitystyökalun käyttöliittymälisäosista tehty vertailu sen wiki-sivuilla (Ogre Wiki 2014). Siinä eri käyttöliittymälisäosien yleisimpiä ominaisuuksia on listattu ja jokaisesta lisäosasta on tutkittu, löytyykö siitä kyseinen ominaisuus. Tästä työstä vastaavanlainen laaja ja yksityiskohtainen vertailu puuttuu, ja jälkikäteen voisi sanoa, että sen tekeminen olisi tuonut työlle lisäarvoa. Teknisiä vertailuita voi kuitenkin tehdä myös itsenäisesti, joten uskoisin työn täyttävän lisäosien käyttökokemuksia vertailevien tutkimuksien tyhjiötä.

Työtä voisi jatkaa tekemällä käyttöliittymään lisää monimutkaisia elementtejä ja testaamalla niitä tarkemmin. Tässä työssä on keskitytty enemmän lisäosien käyttöönottoon ja opetteluun uuden käyttäjän näkökulmasta. Lisäosissa on paljon mahdollisuuksia mielenkiintoisten ominaisuuksien toteuttamiseen, ja niiden tutkiminen esimerkiksi pelin kehityksen yhteydessä voisi olla kiinnostavaa. Toinen jatkokehitysidea voisi olla käyttöliittymien suoritustehokkuuden järjestelmällinen testaaminen. Tässä työssä ei ole testattu ja mitattu lisäosien eroja suoritustehokkuudessa, vaan asia on käsitelty varsin

pintapuolisesti piirtokutsuja ja tekstuuriatlaksen tukea vertailemalla.

Lähteet

Business Dictionary. 2014. What is graphical user interface (GUI)? Definition and meaning. http://www.businessdictionary.com/definition/graphical-user-interface-GUI.html. 19.11.2014.

Business Dictionary. 2014. What is user interface? Definition and meaning. http://www.businessdictionary.com/definition/user-interface.html. 19.11.2014.

Chaiko, A. 2012. Choosing GUI framework for your Unity3D project: EZGUI vs NGUI. Heyworks-blogi. http://blog.heyworks.com/choosing-gui-framework-for-your-unity3d-project-ezgui-vs-ngui-part-i/. 8.11.2014.

Chu, P. 2012. In Search of the Perfect Unity GUI. Fugu Talk -blogi. 15.3.1012. http://www.fugutalk.com/?p=5360. 8.11.2014.

Eronen, J. 2014. Roolipelin kehittäminen Unity 3D:llä. Karelia-ammattikorkeakoulu. Tietojenkäsittelyn koulutusohjelma. Opinnäytetyö. http://urn.fi/URN:NBN:fi:amk-201404104164. 7.11.2014.

Fagerholt, E. & Lorentzon M. 2009. Beyond the HUD -- User Interfaces for Increased Player Immersion in FPS Games. http://publications.lib.chalmers.se/publication/111921. 30.8.2014.

Goldstone, W. 2014. Unity 4.6 – New UI World Space Canvas. Http://blogs.unity3d.com/2014/06/30/unity-4-6-new-ui-world-space-canvas/. 7.11.2014.

Johansen, RS. 2014. Overview of the New UI System. http://blogs.unity3d.com/2014/05/28/overview-of-the-new-ui-system/. 7.11.2014.

Jørgensen, K & Llanos, SC. 2011. Do Players Prefer Integrated User Interfaces? A Qualitative Study of Game UI Design Issues. http://www.digra.org/wp-content/uploads/digital-library/11313.34398.pdf. 19.11.2014.

Kokkonen, T. 2014. Microtransactions in an Android Game. Karelia-ammattikorkeakoulu. Tietojenkäsittelyn koulutusohjelma. Opinnäytetyö. http://urn.fi/URN:NBN:fi:amk-2014061212700. 7.11.2014.

Loewald, T. 2014. Daikon Forge. Inconsequence-blogi. 8.5.2014. http://loewald.com/blog/?p=5527. 8.11.2014.

Madigan, J. 2010. The Psychology of Immersion in Video Games. http://www.psychologyofgames.com/2010/07/the-psychology-of-immersion-in-video-games/. 15.9.2014.

Mukai, J. 2014. Choosing a Unity UI Framework: Comparing Scaleform, NGUI, and Coherent. Proletariat-blogi. 2.1.2014. http://proletariat.com/2014/01/02/choosing-a-unity-ui-framework-comparing-scaleform/. 8.11.2014.

Nevalainen, N. 2014. VIDEOPELIT JA KÄYTETTÄVYYS. Karelia-ammattikorkeakoulu. Tietojenkäsittelyn koulutusohjelma. Opinnäytetyö. http://urn.fi/URN:NBN:fi:amk-2014061112622. 7.11.2014.

Ogre Wiki. 2014. Comparison of GUIs. http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Comparison+of+GUIs. 18.11.2014.

PC.net. 2009. Definition of User Interface. http://pc.net/glossary/definition/user_interface. 19.11.2014.

Poh, M. 2014. Evolution of Video Games User Interface (UI). http://www.hongkiat.com/blog/video-games-ui-evolution/. 15.9.2014.

Polsinelli, P. 2013. Why is Unity so popular for videogame development?

http://designagame.eu/2013/12/unity-popular-videogame-development/. 7.11.2014.

Rouse, M. 2006. GUI (graphical user interface). http://searchwindevelopment.techtarget.com/definition/GUI. 3.10.2014.

Russell, D. 2011. Video game user interface design: Diegesis theory. http://devmag.org.za/2011/02/02/video-game-user-interface-design-diegesis-theory/. 21.11.2014.

Stonehouse, A. 2014. User interface design in video games. http://gamasutra.com/blogs/AnthonyStonehouse/20140227/211823/User_interface_design_in_video_games.php. 30.8. 2014.

Stuart, K. 2010. What do we mean when we call a game 'immersive'? http://www.theguardian.com/technology/gamesblog/2010/aug/10/games-science-of-immersion. 19.11.2014.

Tech Terms Computer Dictionary. 2009. User Interface definition. http://www.techterms.com/definition/user_interface. 5.10.2014.

Unity Asset Store. 2014. EzGui User Reviews. https://www.assetstore.unity3d.com/en/#!/content/32. 8.11.14.

Unity Asset Store. 2014. GUIMaker User Reviews. https://www.assetstore.unity3d.com/en/#!/content/12588. 8.11.14.

Unity Asset Store. 2014. IGUI User Reviews. https://www.assetstore.unity3d.com/en/#!/content/490. 8.11.14.

Unity Forum. 2011. iGUI Basic - The easiest GUI framework for Unity - RELEASED 50$. http://forum.unity3d.com/threads/igui-basic-the-easiest-gui-framework-for-unity-released-50.105210/. 8.11.2014.

Unity Forum. 2014. Daikon Forge or NGUI. http://forum.unity3d.com/threads/daikon-forge-or-ngui.238035/. 8.11.2014.

Unity Technologies. 2014. GUI Scripting API. http://docs.unity3d.com/ScriptReference/GUI.html. 13.9.2014.

Unity Technologies. 2014. The Leading Global Game Industry Software. http://unity3d.com/public-relations. 7.11.2014.

Valve Corporation. 2014. Steam Controller. http://store.steampowered.com/livingroom/SteamController/. 23.10.2014.

Van Oosten, J. 2013. GUI Scripting in Unity. http://3dgep.com/gui-scripting-in-unity. 30.8.2014.

Slaughter-pelin käyttöliittymämäärittely

1 Pelin käynnistäminen

Peli käynnistyy alkunäyttöön. Alkunäytöstä on mahdollista suorittaa seuraavat toiminnot:

1. Palvelimen perustaminen
2. Saatavilla olevien pelipalvelinten listaaminen
3. Katso pelin tiedot
4. Katso highscore-lista
5. Lopeta (poistu sovelluksesta)

2 Palvelimen perustaminen

Näkymässä palvelimen perustamista varten syötetään

- pelaajan nimi
- verkkoliikenneportti (1-65535)
- pelaajien sallittu enimmäismäärä (2–16)

Palvelin on voitava käynnistää näkymästä, jolloin peli siirtyy pelinäkymään ja vastaanottaa muita pelaajia paikallisen pelaajan lisäksi.

3 Saatavilla olevien pelipalvelinten listaaminen

Näkymään ilmestyy lista, jossa näkyy palvelinten nimet, ip-osoitteet, pelaajien lukumäärä, vapaat paikat. Palvelimeen on oltava mahdollista yhdistää valitsemalla listan alkio, jolloin peli yhdistyy palvelimeen ja siirtyy pelinäkymään.

4 Katso pelin tiedot

Näkymästä on voitava nähdä nopeasti

1. pelin nimi
2. projektin tekijän nimi.

Pelin tiedoista on myös nähtävä binääriversion käännöspäivämäärä ja kellonaika sekä poistua takaisin alkunäyttöön.

5 Katso highscore-lista

Näkymään latautuu lista, josta näkyy pelaajan nimi sekä pisteet.
Listaa on voitava selata siten, että kaikki pistetilastot näkyvät. Pistetilastot on voitava lajitella pelaajan nimen sekä pisteiden mukaan joko nousevaan tai laskevaan järjestykseen, ja järjestystä on voitava vaihtaa lennosta.

6 Pelinäkymä

Pelinäkymässä on oltava chat-ikkuna, jota on voitava käyttää pelin aikana. Chat-ikkunaan syötetään merkkijono, ja se on voitava lähettää joko return-näppäintä painamalla tai hiirellä.

Pelinäkymässä on oltava näkyvissä pisteet siten, etteivät ne häiritse pelaamista, mutta ovat kuitenkin selkeästi näkyvissä.

Pelinäkymässä on voitava tulostaa pelin sisäisiä viestejä näkyviin ruudulle siten, että chat-viestit tai muut käyttöliittymäelementit eivät peity.

Pelinäkymästä on voitava päättää pelisessio, jolloin peli siirtyy varmistusikkunan kautta aloitusnäkymään.

Pelinäkymästä on voitava siirtyä pelistatistiikkanäkymään, josta on voitava silmäillä nopeasti pelaajien nimet, nykyiset pisteet, ping-vasteaika sekä ip-osoite josta yhteys on muodostettu palvelimeen.

Käyttöliittymätutoriaalit

Sisältö

1 Johdanto

Tämä tutoriaali on tehty opinnäytetyön käytännön osuudeksi. Siinä tehdään käyttöliittymä Slaughter-avaruuspeliin. Käyttöliittymä toteutetaan kolmella eri Unity käyttöliittymälisäosalla. Käytettävät lisäosat valittiin pääsosin suosion ja käyttäjäpalautteen perusteella. Käytettävät lisäosat ovat NGUI, iGUI ja Daikon Forge.

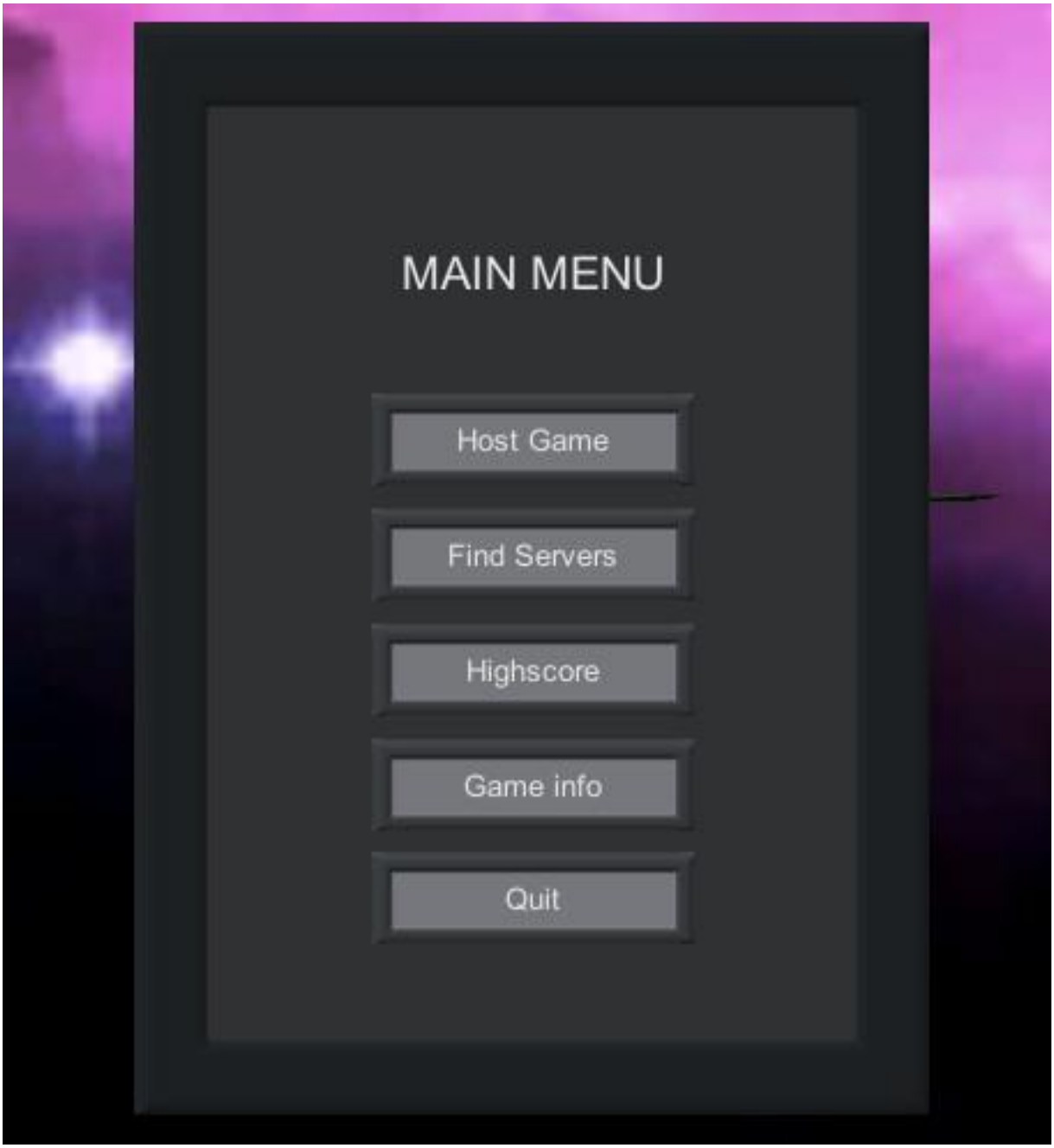
NGUI on ollut pitkään Unity Storen suosituin käyttöliittymälisäosa, ja yleisestikin yksi ostetuimmista lisäosista. Sen vahvuuksia on monipuoliset ominaisuudet ja sen sopeutuminen hyvin erilaisiin peleihin. NGUI:lla käyttöliittymän teko on näistä kolmesta työläin, koska kaikki elementit täytyy itse kasata monesta palasta. Se voi joissain tapauksissa olla tarpeettoman monimutkaista, mutta se takaa hyvän muokattavuuden ja kontrollin tehtäviin ominaisuuksiin.

IGUI on saanut hyvää palautetta helposta lähestyttävyydestä ja intuitiivisista työkaluista. Se onkin näistä kolmesta helpoin omaksua. Käyttöliittymän tekeminen on todella helppo aloittaa ja peruselementtien tekeminen onnistuu kuin itsestään. Elementit ovat yleensä valmiita kokonaisuuksia, eikä NGUI:n tyylistä osista kasaamista juuri tarvitse tehdä. Sen kääntöpuolena on tietenkin se, että kaikkia haluttuja muutoksia ei ole mahdollista tehdä, ja sen takia se ei välttämättä sovi lisäosaksi kaikkiin projekteihin.

Daikon Forge valittiin, koska se vaikutti varteenotettavimmalta kilpailijalta NGUI:lle. Se on kokemuksen perusteella monella tapaa NGUI:n ja iGUI:n välimalli. Se on helppo käyttää ja sen elementit ovat valmiita kokonaisuuksia kuten iGUI:ssa. Sen käyttämisen alottaminen on kuitenkin vaikeampaa, eikä se ole aivan yhtä intuitiivinen. Kun sen kanssa pääsee liikkeelle, on se kuitenkin looginen ja helppo käyttää. Valmiiden elementtikokonaisuuksien vuoksi se ei ole yhtä helposti muokattavissa kuin NGUI.

Tätä tutoriaalia tehdessä on Daikon Forge kokenut suurimmat mullistukset. Sen kehittäjä päätti lopettaa lisäosan tukemisen, ja päivityksiä ei ollut enää luvassa. Se on nyt kuitenkin julkaistu opensourcena, mikä tarkoittaa, että sitä voi päivittää kuka vain, ja että sen voi ladata käyttöön ilmaiseksi.

2 Slaughter-pelin käyttöliittymämäärittely



Kuva: Slaughter-pelin alkuvalikko.

2.1 Pelin käynnistäminen

Peli käynnistyy alkunäyttöön. Alkunäytöstä on mahdollista suorittaa seuraavat toiminnot

6. Palvelimen perustaminen
7. Saatavilla olevien pelipalvelinten listaaminen
8. Katso pelin tiedot
9. Katso highscore-lista
10. Lopeta (poistu sovelluksesta)

2.2 Palvelimen perustaminen

Näkymässä palvelimen perustamista varten syötetään

- pelaajan nimi
- verkkoliikenneportti (1-65535)
- pelaajien sallittu enimmäismäärä väliltä (2-16)

Palvelin on voitava käynnistää näkymästä, jolloin peli siirtyy pelinäkymään ja vastaanottaa muita pelaajia paikallisen pelaajan lisäksi.

2.3 Saatavilla olevien pelipalvelinten listaaminen

Näkymään ilmeistyy lista, jossa näkyy palvelinten nimet, ip-osoitteet, pelaajien lukumäärä, vapaat paikat. Palvelimeen on oltava mahdollista yhdistää valitsemalla listan alkio, jolloin peli yhdistyy palvelimeen ja siirtyy pelinäkymään.

2.4 Katso pelin tiedot

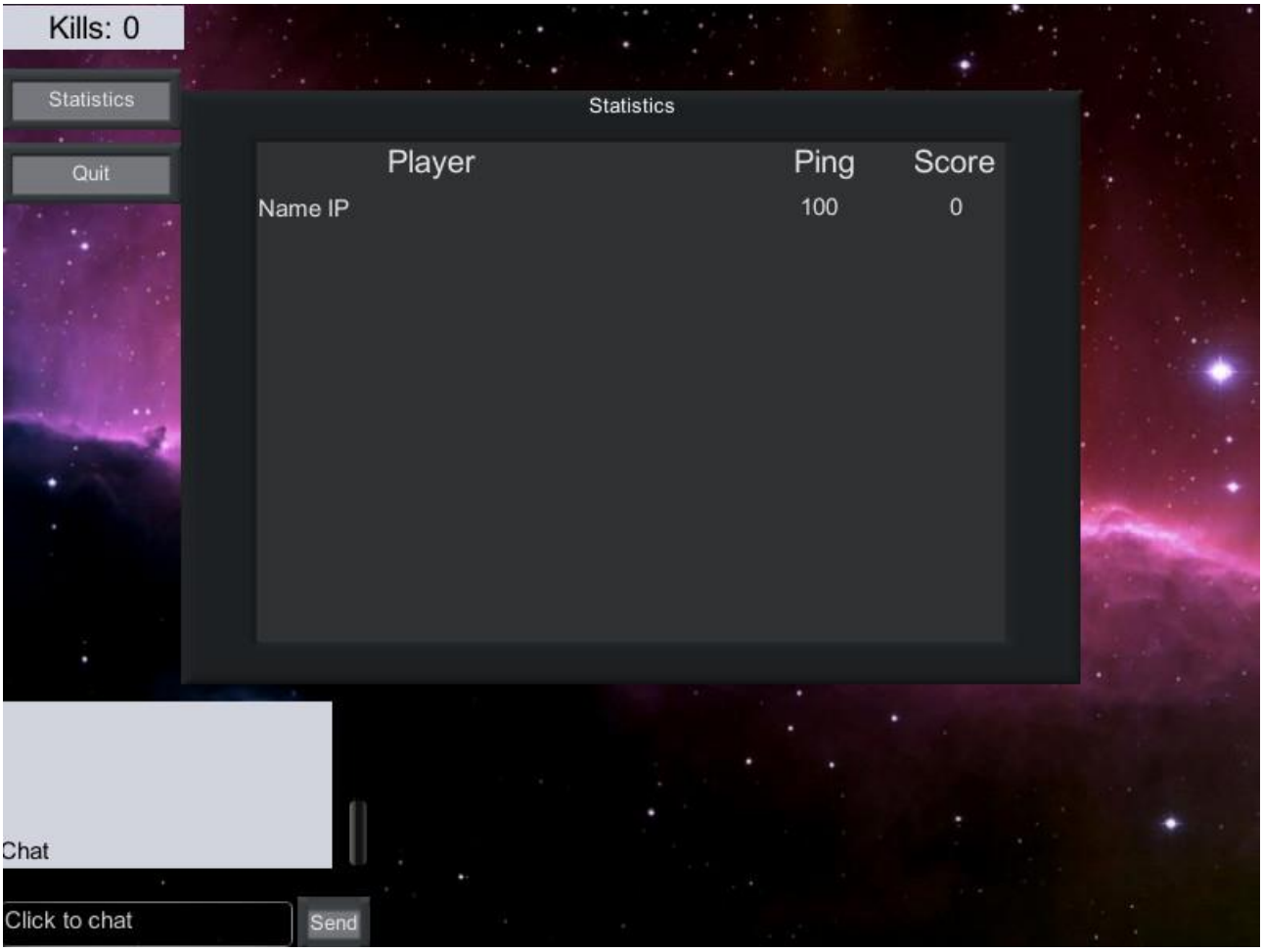
Näkymästä on voitava nähdä nopeasti

3. Pelin nimi
4. Projektin tekijän nimi
5. Binääriversion käännöspäivämäärä ja kellonaika sekä poistua takaisin alkunäyttöön.

2.5 Katso highscore-lista

Näkymään latautuu lista, josta näkyy pelaajan nimi sekä pisteet.
Listaa on voitava selata siten, että kaikki pistetilastot näkyvät. Pistetilastot on voitava lajitella pelaajan nimen sekä pisteiden mukaan joko nousevaan tai laskevaan järjestykseen, ja järjestystä on voitava vaihtaa lennosta.

2.6 Pelinäkymä



Kuva: Slaughter-pelin pelinäkymä.

Pelinäkymässä on oltava chat-ikkuna, jota on voitava käyttää pelin aikana. Chat-ikkunaan syötetään merkkijono, ja se on voitava lähettää joko return-näppäintä painamalla tai hiirellä.

Pelinäkymässä on oltava näkyvissä pisteet siten, etteivät ne häiritse pelaamista, mutta ovat kuitenkin selkeästi näkyvissä.

Pelinäkymässä on voitava tulostaa pelin sisäisiä viestejä näkyviin ruudulle siten, että chat-viestit tai muut käyttöliittymäelementit eivät peity.

Pelinäkymästä on voitava päättää pelisessio, jolloin peli siirtyy varmistusikkunan kautta aloitusnäkymään.

Pelinäkymästä on voitava siirtyä pelistatistiikkanäkymään, josta on voitava silmäillä

nopeasti pelaajien nimet, nykyiset pisteet, ping-vasteaika sekä ip-osoite josta yhteys on muodostettu palvelimeen.

3 Yleinen toteutus

Slaughter pelille tehdään käyttöliittymä kolmella eri Unity-lisäosalla. Vaikka ne kaikki ovat hyvin erilaisia, muistuttaa niiden toiminta monella tavalla toisiaan. Tässä luvussa käydään läpi yleisellä tasolla miten käyttöliittymä toteutetaan, ja myöhemmissä luvuissa on yksityiskohtaiset ohjeet miten kullakin lisäosalla toteutetaan esitellyt ominaisuudet.

3.1 Tekstuurit

Lisäosista NGUI ja Daikon Forge hyödyntävät atlas-periaatetta. Siinä kaikki käytettävät tekstuurit pakataan yhteen isoon tekstuuriin. Atlaksen etu verrattuna isoon määrään pienempiä tekstuuritiedostoja on sen prosessointitehokkuudessa ja tiedonsiirrossa verkon yli. Grafiikanohjaimen ei tarvitse prosessoida jokaista piirrettävää tekstuuria erikseen, mikä voi vaikuttaa erityisesti mobiilialustoille kehittäessä. Jos peliä kehitetään verkkoon, esimerkiksi web-selaimella pelattavaksi, syntyy jokaisesta yksittäisestä tiedostosta lisää verkkoliikennettä.

NGUI:n Atlas-työkalu on todella helppo, ja se toimii melkeimpä drag-and-drop -periaatteella. Daikonin atlas-ominaisuus vaatii kolmannen osapuolen työkalun, joten se on jonkin verran monimutkaisempi ja vaatii hieman paneutumista. IGUI:n atlaksen puute voi olla merkittävä tekijä, jos atlas on painava kriteeri käyttöliittymälisäosaa valittaessa.

3.2 Valikkorakenne

Alkuvalikko toteutetaan luomalla paneeli, jossa on painikkeet alavalikoiden avaamiseen. Alavalikot ovat myös paneeleita, ja ne sisältävät graafiset elementit ja skriptit niiden vaatimien toimintojen toteuttamiseen. Paneeleiden avaaminen ja sulkeminen toteutetaan hieman eri tavalla kaikissa lisäosissa. NGUI:ssa muokataan koko paneeli-objektin enabled-arvoa. IGUI:ssa muokataan paneeli-skriptin enabled arvoa. Daikonissa muokataan paneeli-skriptin enabled-arvon lisäksi sen visible-arvoa.

NGUI:ssa on valmis skripti, jolla paneeleiden avaaminen ja sulkeminen on helppo lisätä painikkeen toimintaan. Daikon Forgessa skriptiä ei ole valmiina, mikä on hieman yllättävää. Yhdessä sen mukana tulevista esimerkkitiedostoista on tämä toiminnallisuus, mutta se on toteutettu hyvin tarkasti juuri siihen esimerkkiin sopivaksi. Tässä ohjeessa luodaan sen pohjalta oma skripti, joka toimii yleisissäkin tapauksissa NGUIn skriptin tapaisesti. IGUI:ssa on suuressa osassa IGUICode-tiedosto, missä kaikki käyttöliittymän ominaisuus voidaan määritellä. Siihen tiedostoon generoidaan metodit painikkeiden toiminnoille. Myös iGUI:n valikkorakenne vaatii siis jonkin verran omaa koodia.

3.3 Graafiset perus-elementit

Perus-elementeiksi voidaan lukea käyttöliittymässä paljon käytetyt Panel-, Sprite-, Button- ja Label-elementit. Nämä kaikki toimivat kaikissa lisäosissa hyvin samankaltaisesti.

Panel-elementti on eräänlainen kokonaisuutta hallitseva elementti. Kaikki yhden paneelin alla olevat graafiset elementit käyttävät piirtämiseen yhden piirtokutsun. Sen takia panel-objekteja ei kannata tehdä enemmän kuin tarpeellista.

Sprite-elementti piirtää yhden tekstuurin. Se tulee tutuimmaksi NGUI:ssa, koska siinä se täytyy lisätä aina objekteihin, joissa halutaan näyttää tekstuuri. IGUI:ssa ja Daikonissa suurimmassa osassa graafisia elementtejä on spriten näyttäminen sisäänrakennettuna ominaisuutena.

Button-elementti on nimen mukaisesti tarkoitettu painikkeiden tekemiseen. Button-skriptissä on mahdollisuus muuttaa painikkeen ulkonäköä sen mukaan onko esimerkiksi hiiri sen päällä tai painetaanko sitä. IGUI:ssa ja Daikonissa on sisäänrakennettu tunnistus siihen, painetaanko nappia. NGUI:ssa täytyy painikkeeseen lisätä Collider-elementti.

Label-elementissä voidaan näyttää tekstidataa. Sillä tehdään alkuvalikossa kaikki otsikot ja tekstikentät, joilla ohjeistetaan pelaajaa. Label-objekteissa on aina mahdollisuus vaihtaa fonttia, fontin kokoa, tekstin sijaintia ja väriä sekä monia muita ulkonäköön vaikuttavia asioita. Tekstiä voi kaikilla lisäosilla vaihtaa pelin aikana lennosta, koska teksi-ominaisuus on kaikissa julkinen muuttuja.

3.4 Tekstinsyöttö

Pelaajan syöttämää tekstiä tarvitsee lukea muutamassa ominaisuudessa. Uutta serveriä tehtäessä luetaan pelaajien maksimimäärä, severin tietoliikenneportti ja pelaajan nimi. Pelin ollessa käynnissä luetaan pelaajan kirjoittama teksi ja poimitaan se chat-ominaisuutta varten.

IGUI:ssa ja Daikonissa teksikenttä on yksinkertainen toteuttaa. Muiden elementtien tavoin siinä on sisäänrakennettuna kaikki toiminnalisuus, mitä tekstinsyöttöön tarvitaan. IGUI:ssa on lisäksi erillinen elementti numeroiden lukemiselle. Se voi olla kätevä, jos haluaa rajoittaa jo syöttövaiheessa mitä pelaaja voi kenttään syöttää. NGUI:ssa tekstikenttä rakennetaan useammasta pienestä osasta, ja mukana tulevien esimerkkitiedostojen tutkiminen valmiin ja toimivan ratkaisun löytämiseksi on suositeltavaa.

3.5 Listanäkymä

Serverinvalintapaneelissa serverit listataan ja pelaajan täytyy pystyä valitsemaan mihin

niistä liitytään. IGUI:ssa ja Daikonissa on elementti listanäkymälle, johon voidaan hakea lista näytettävistä servereistä. NGUI:ssa listan tekemiseen ei ole mitään yhtä valmista ja helppoa elementtiä listan tekemiseen. Siitä löytyy Table-elementti, jolla voi helpottaa listan asettelua, mutta tässä toteutuksessa listaus tehdään omalla yksinkertaisella skriptillä.

Daikonissa on Listbox-elementti, jolla listan tekeminen on todella helppoa. Siinä on myös valmis toiminnallisuus valitun serverin visuaaliselle korostamiselle, ja valintaan pääsee koodissa käsiksi julkisen muuttujan kautta.

IGUI:ssa lista tehdään alasvetovalikkona. Myös se on erittäin valmis kokonaisuus, ja se sisältää kaiken toiminnallisuuden, mitä serverilistaus tarvitsee. Valittu serveri on skriptissä julkisena muuttujana, joten siihen pääsee koodissa helposti käsiksi.

NGUI:n serverilistaus toteutetaan template-periaatteella. Serverin tietoja varten luodaan mallipohja, ja jokaista listasta löytyvää serveriä vastaamaan instansioidaan yksi elementti. Serverin tiedot asetetaan instansioituun elementtiin ja se asetetaan edellisen elementin alapuolelle. Mallipohja on periaatteessa painike, jota painamalla peli yhdistyy serveriin painikkeen tietojen mukaisesti.

3.6 Vieritettävä näkymä

Vieritettävää näkymää tarvitaan Highscore-paneelissa ja Chat-ikunassa. Kaikissa lisäosissa on hyvät elementit, joilla saa tehtyä halutunlaisen vieritysnäkymän.

NGUI:n vieritysnäkymä on näistä kolmesta ehkäpä vaikein tehdä. NGUI:n tyyliin toimiva vieritysnäkymä vaatii, että se kasataan monesta elementistä juuri oikein. Tästä johtuen on suositeltavaa etsiä mukana tulevista esimerkeistä toimiva ratkaisu ja katsoa siitä mallia. Varsinkin Chat-ikkunan tekeminen voi olla vaikea, koska tekstidata täytyy syöttää siihen oikein, tai vieritysominaisuus ei toimi.

Daikonissa ja iGUI:ssa vieritysnäkymä toteutetaan hyvin samantyylisesti. Vieritysnäkymä-elementillä rajataan alue, missä tekstiä halutaan näyttää. Sen alle luodaan label-elementti, jonka sisältämä teksti liikkuu kun elementtiä vieritetään. Molemmissa on vapaaehtoisena elementtinä vierityspalkki. IGUIssa vierityspalkki on mukana vieritysnäkymässä, ja sen saa päälle ruksia painamalla. Daikonissa palkki täytyy tehdä erikseen ja liittää se vierityselementtiin.

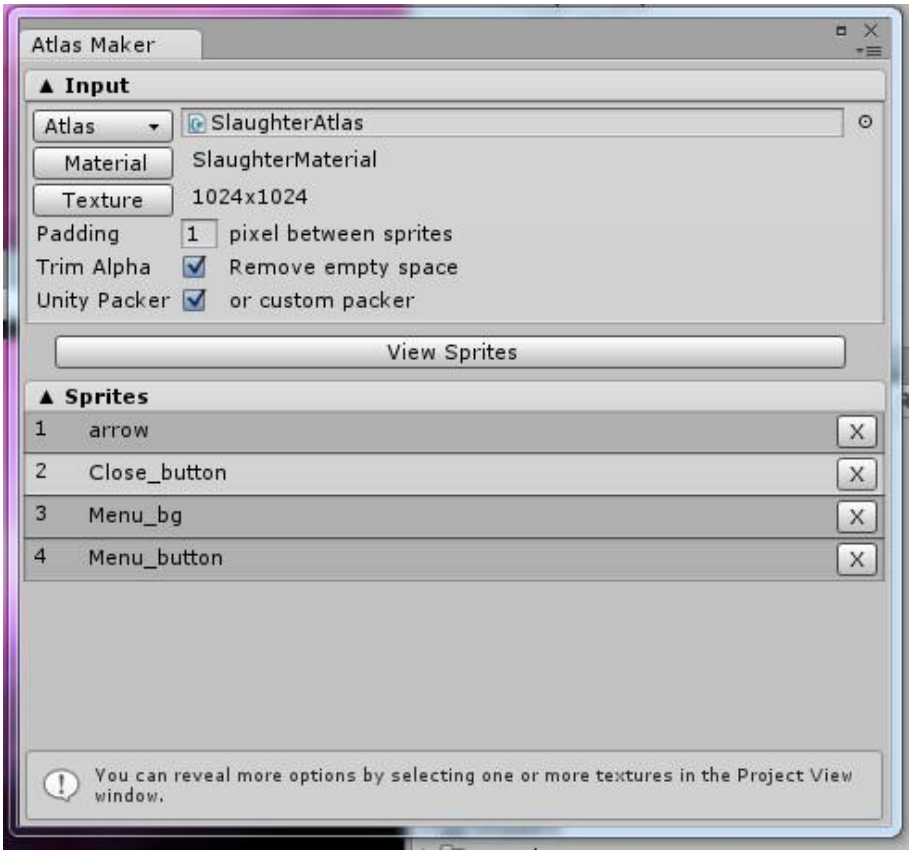
4 NGUI

4.1 Atlas ja tekstuurit

4.1.1 Atlas

Atlas on yksi iso tekstuuri joka sisältää useita eri tekstuureja. Tekstuurien kasaaminen yhdeksi isoksi tekstuuriksi on tehokkaampaa kuin usean pienemmän tekstuurin käyttäminen. NGUI:ssa on helppokäyttöinen työkalu tekstuureiden yhdistämiseen. Sillä onnistuu myös uusien tekstuureiden lisääminen sekä vanhentuneiden poistaminen ilman, että jo tehtyä työtä tarvitsee tehdä uusiksi.

Luo uusi tyhjä prefab ja lisää siihen UIAtlas-scripti. Klikkaa prefabia hiiren oikealla napilla ja valitse NGUI -> Open Atlas Maker. Avautuvassa näkymässä pysyy valittuna uusi Atlas vaikka Project-ikkunassa klikkaisi muita resursseja. Klikkaamalla haluttuja tekstuureja ne tulevat mukaan atlakseen. Kun olet valinnut kaikki haluamasti tekstuurit tarvitsee klikata Add/Update -nappia ja NGUI generoi ison atlas-tekstuurin.



Kuva: NGUI:n Atlas Maker.

4.1.2 Sprite

NGUI:n UISprite luokassa haluttu tekstuuri asetetaan valitsemalla ensin atlas, ja siitä oikea tekstuuri. Luokasta löytyy myös säädöt ja valinnat sille, miten tekstuuri halutaan näyttää. Yksi olennaisimmista säädöistä on Depth-arvo, jolla määritellään missä järjestyksessä päällekkäiset spritet näytetään.

Kuva: UISprite-komponentti

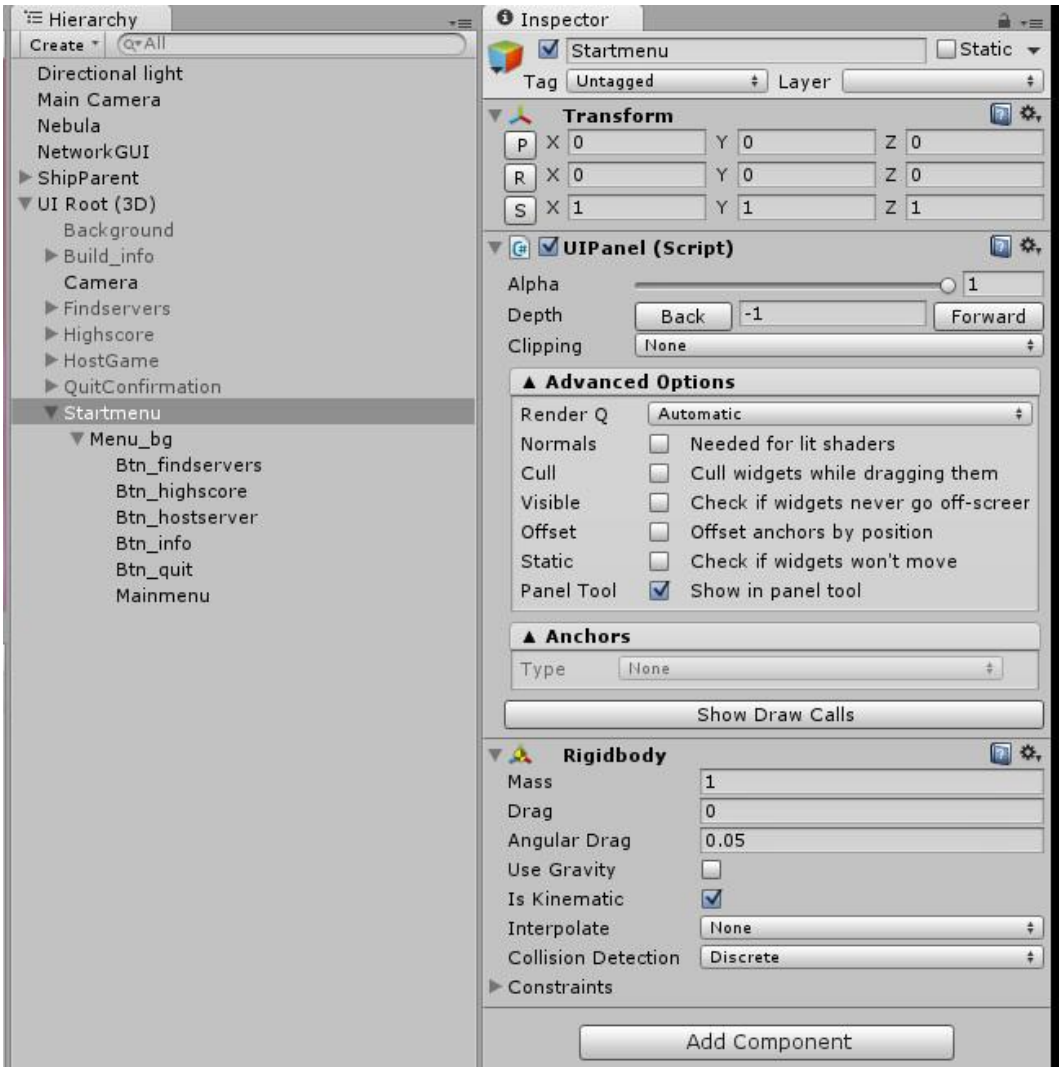
4.2 Alkuvalikko

Alkuvalikko tullaan toteuttamaan niin, että pelaajalle avautuu aluksi päävalikko, josta voi toimintapainikkeilla avata uusia valikoita. Valikoista pääsee pois sulkemisnapista.

Alkuvalikon luonti aloitetaan luomalla tyhjä UI. Valitaan ylävalikosta NGUI -> Create -> 3D UI.

4.2.1.1 Päävalikko

Päävalikko luodaan valitsemalla juuri luotu UI Root -objekti ja lisäämällä siihen UIPanel-scripti. Tämä scripti hoitaa paneeliin kuuluvuen spritejen piirtämisen automaattisesti mahdollisimman pienillä piirtokutsuilla.



Kuva: UIPanel-komponentti

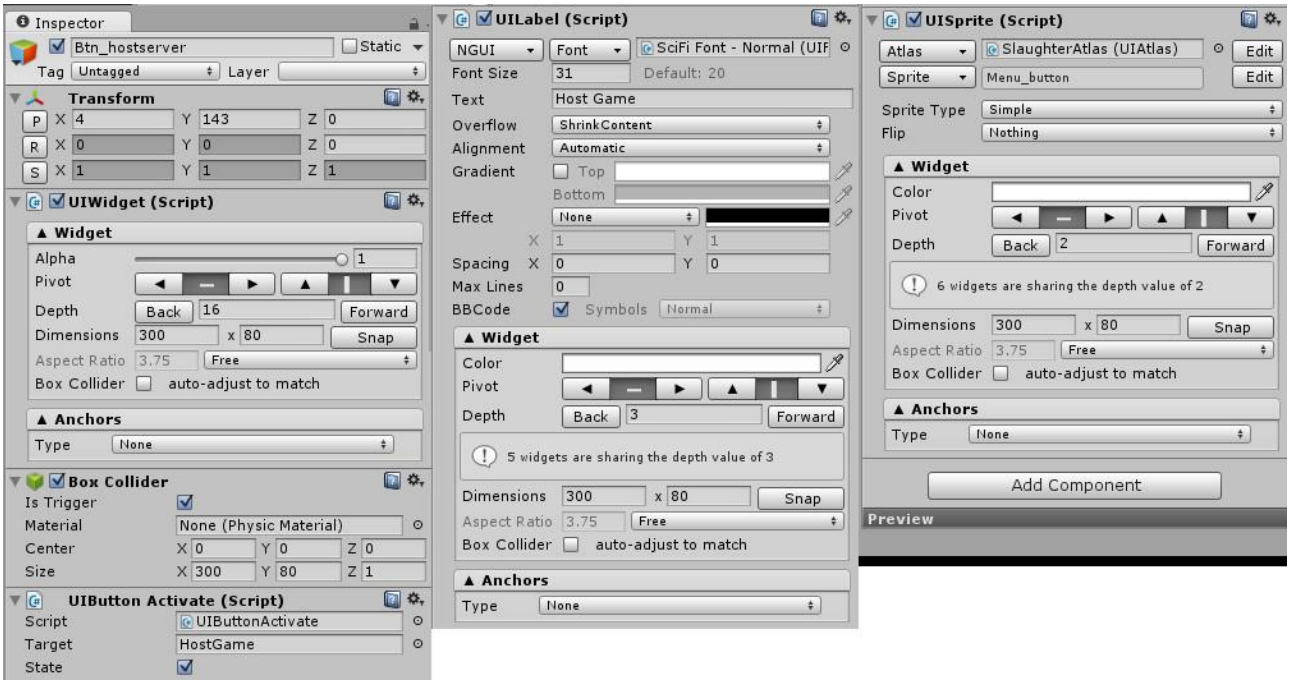
UI:n graafiset elementit lisätään luomalla uusi UISprite-objecti paneeli-objectin alapuolelle. UISprite elementtiin asetetaan atlas ja sprite -ominaisuudet halutun tekstuurin mukaan. Sen lisäksi elementtiin on säädettävä arvot kokoa ja syvyyttä varten.

4.2.1.2 Toimintopainikkeet

Toimintopainikkeet sisältävät useampia komponentteja kuin pelkän spriten. Painikkeessa näkyvä teksti lisätään UILabel scriptillä. Text-kenttään kirjoitetaan haluttu teksti. Widget-kentässä asetetaan mihin reunaan teksti tasataan.

Toimintopainike pitää myös asettaa mitä tapahtuu kun sitä painetaan. Sitä varten painikkeelle täytyy lisätä Box Collider -komponentti ja tehdä siitä Triggeri laittamalla asetus

päälle valintaruudusta. Colliderin koko määrittää miltä alueelta painikkeen painaminen rekisteröidään. Painike laitetaan tässä tapauksessa avaamaan uusi paneeli ja siihen NGUI:sta löytyy UIButtonActivate-scripti. Kun se on lisätty komponentiksi painikkeeseen, täytyy Target-kenttään raahata Hierarchy-näkymästä haluttu paneeli ja asettaa State-arvo päälle. Tämän jälkeen kannattaa vielä käydä läpi kaikkien komponenttien Depth-arvot ja varmistaa, että kaikki on asetettu näkymään oikeassa järjestyksessä.



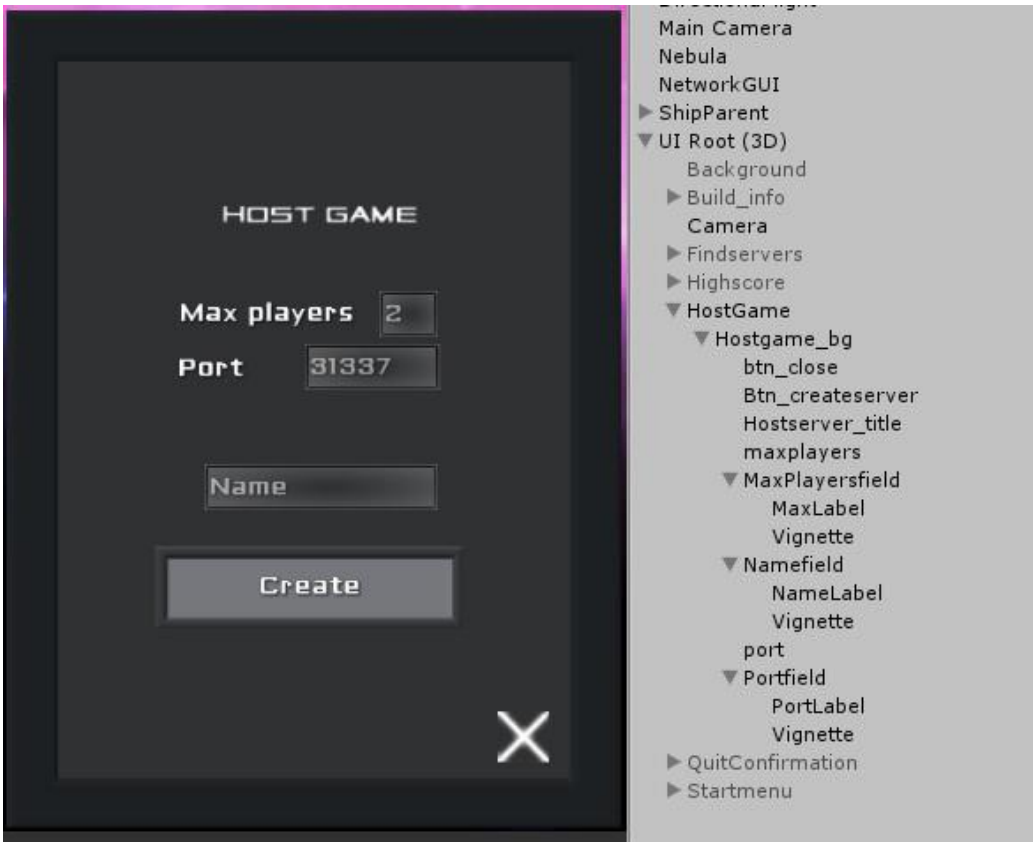
Kuva: Alkuvalikon painikkeiden asetukset.

4.3 Alavalikot

Painikkeista avautuvat valikot luodaan samalla tavalla kuin päävalikko, eli luodaan uusi UIPanel-objekti. UISprite-objekteissa täytyy varmistaa, että kaikkien Depth-arvot ovat asetettu oiken. Jokaiselle alavalikolle luodaan oma paneeli ja niihin liitetään kaikki niiden tarvitsemat toiminnallisuudet.

4.3.1 Host Game

Host Game -paneelin tärkein toiminnallinen asia on pelaajan syöttämien tietojen lukeminen ja uuden pelin aloittaminen näiden tietojen pohjalta. Tekstikenttä objektissa on kaksi pakollista komponenttia ja mahdolliset graafiset komponentit. Pakolliset komponentit ovat Box Collider triggeriksi asetettuna ja UIInput-scripti. Tekstikenttään voi asettaa vakioarvon luomalla erillisen objektin mihin lisätään UILabel-komponentti. UIInput-scriptin Label-kenttään asetetaan juuri luotu UILabel-objekti. Nyt UIInputtiin pelin aikana syötettävä teksti menee UILabel-objektiin, ja tämä näkyy pelaajalle reaaliaikaisesti.



Kuva: Host Game -paneelin rakenne.

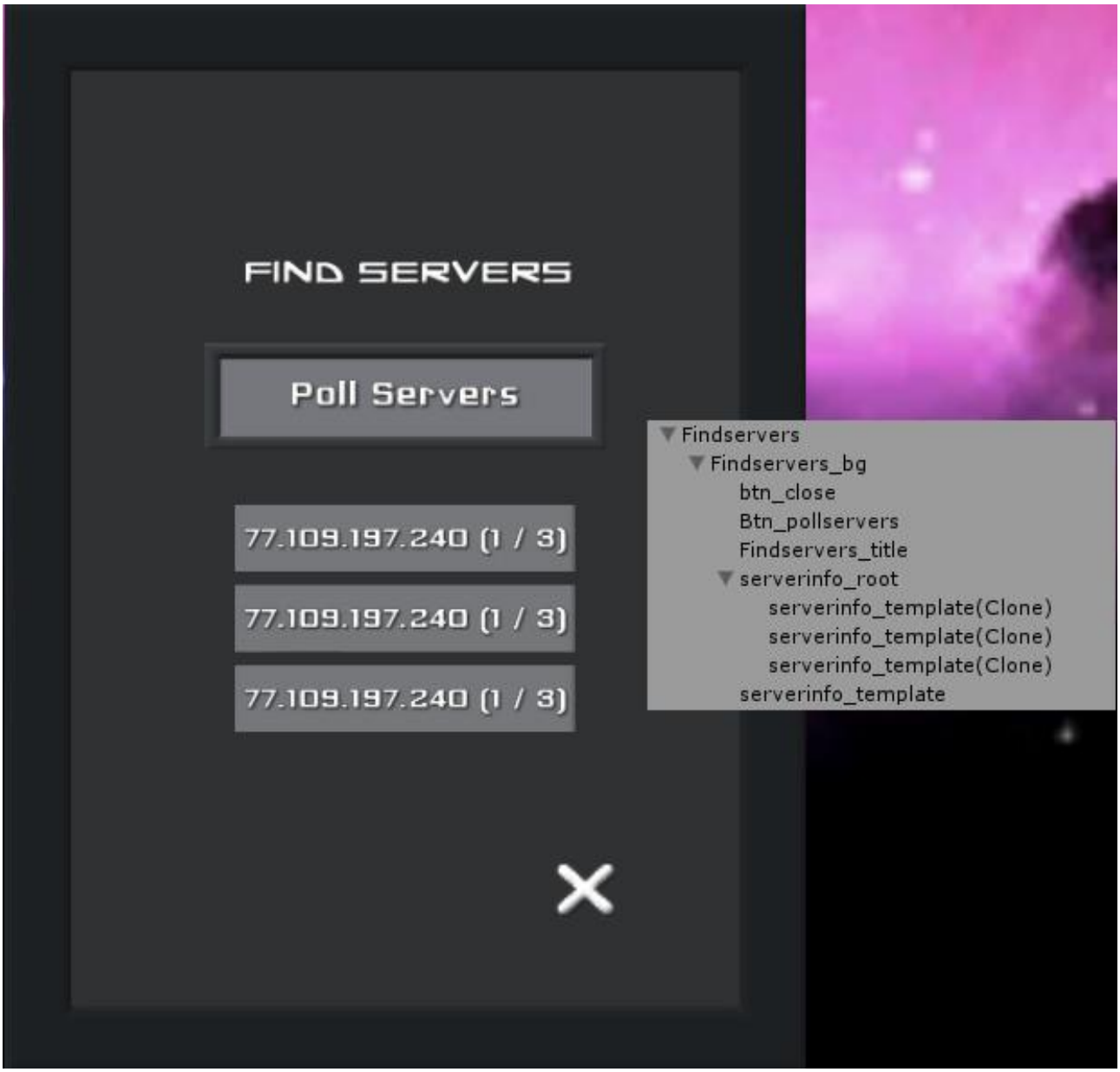
Tekstikenttiä tarvitaan kolme kappaletta pelaajan nimen, palvelimen portin ja pelaajien enimmäismäärän lukemiseksi. Paneeliin luodaan toimintopainike ja siihen lisätään erillinen scripti, jossa painikkeen toiminta määritellään. Painikkeen OnClick-metodi suoritetaan aina kun painiketta painetaan, joten toiminnallisuus kannattaa suorittaa siellä. Samalla kannattaa suorittaa tietojen haku tekstikentästä. Tämä on yksinkertaista tehdä, koska UIInputissa on julkinen value-parametri, josta arvot on helppo poimia.

```
void hostServer()
{
    // Getting the values from input fields
    int maxPlayers = Int32.Parse(transform.parent.Find("MaxPlayersfield").GetComponent<UIInput>().value);
    int port = Int32.Parse(transform.parent.Find("Portfield").GetComponent<UIInput>().value);
    string playerName = transform.parent.Find("Namefield").GetComponent<UIInput>().value;

    // Checking if values are withing accepted limits
    if(playerName.Equals("")) {playerName = "DefaultPlayer";}
    if(maxPlayers > 16) {maxPlayers = 16;}
    if(maxPlayers < 2)  {maxPlayers = 2;}

    guiScript.LaunchServer(maxPlayers, port, playerName);
}
```

Kuva: Koodissa haetaan tekstikenttien arvot ja syötetään ne palvelimen käynnistävälle metodille.

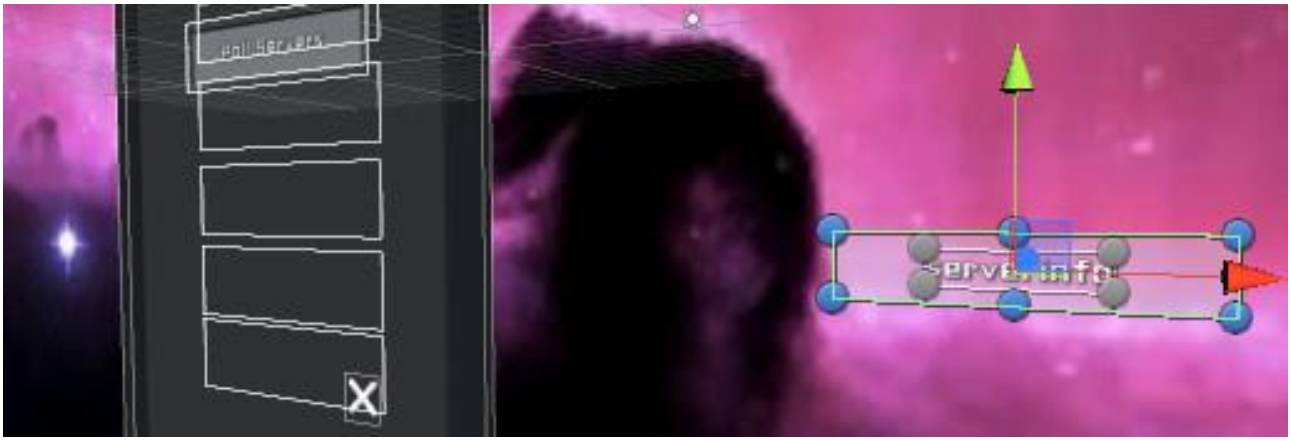
4.3.2 **Find Servers**

Find Servers -paneelissa listataan kaikki serverit, ja annetaan pelaajalle mahdollisuus liittyä valittuun peliin. Serverit listataan instansioimalla valmista pohjaa, johon tietyn serverin tiedot asetetaan. Pelaaja pystyy liittymään peliin klikkaamalla serverin tiedot näyttävää elementtiä.



Kuva: Find Servers -paneelin rakenne.

Template-objekti on periaatteeltaan aivan kuten mikä tahansa painike. Erona on se, että se sijaitsee pelinäkymän ulkopuolella, ja sitä instansioidaan haluttuun paikkoihin. Instansiointihetkellä painikkeelle asetetaan haluttu teksti-arvo ja annetaan serverin ip-osoite.



Kuva: Template-objekti.

Painikkeessa oleva koodi yhdistää pelin haluttuun serveriin.

```
public class GUIServerButton : MonoBehaviour {

    public string address;

    void OnClick()
    {
        GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
        NetworkConnectionError err = Network.Connect(address, guiScript.GetPort());
        switch (err)
        {
        case NetworkConnectionError.NoError:
            Debug.Log("Waiting for Reply");
            break;
        default:
            Debug.Log("Could not connect: " + err);
            break;
        }
    }
}
```

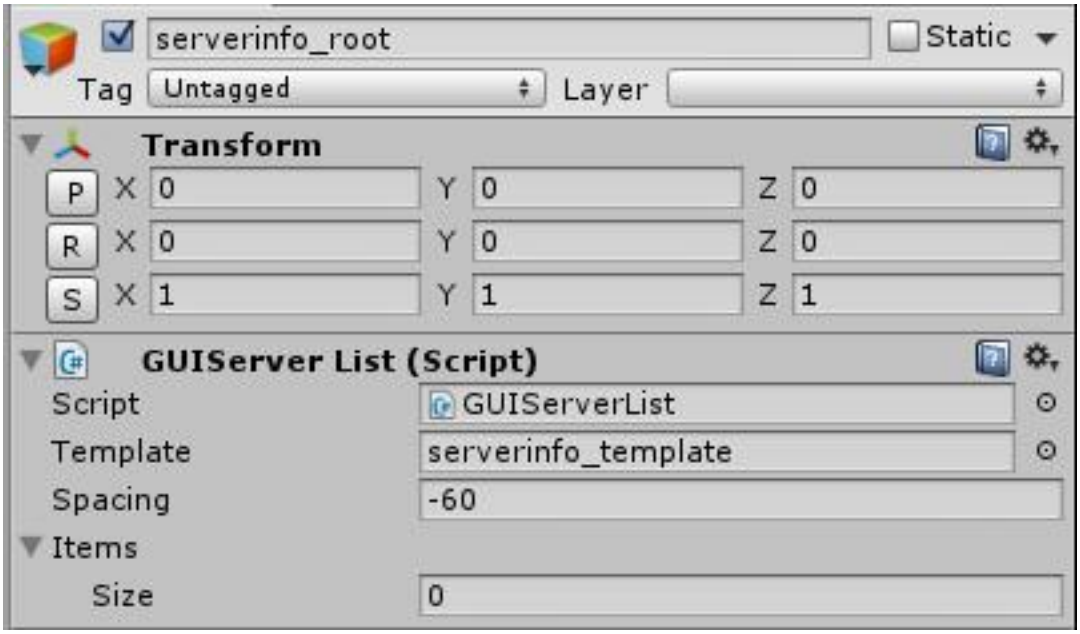
Kuva: Templatesta luotua painiketta painamalla peli yhdistää peliserverille.

Serverit listataan pelaajan painaessa Poll Servers-painiketta. Painikkeessa on skripti, joka kutsuu metodia serverilistan päivittämiseksi. Sen jälkeen se kutsuu metoria sen listan näyttämiseksi.

```
void OnClick()
{
    switch (actionButton)
    {
    case ActionButton.Host:
            hostServer();
        break;
    case ActionButton.Find:
        StartCoroutine(findServers());
        break;
    case ActionButton.Quit:
            Application.Quit();
        break;
    default:
        break;
    }
}
IEnumerator findServers()
{
    GUIServerList srvList = transform.parent.Find("serverinfo_root").GetComponent<GUIServerList>();
    guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();

    yield return StartCoroutine(guiScript.UpdateServerList());
    srvList.fillList(guiScript.serverList);
}
```

Kuva: Koodissa serverilista haetaan GUIScriptistä, ja se syötetään GUIServerList-skriptin fillList-metodiin.

Servereiden instansiointi tapahtuu serverinfo_root-objektissa. Siinä on kiinnitettynä skripti, joka hakee serverilistan ja hallitsee miten ne tuodaan näkyviin. Template-objekti asetetaan inspectorissa julkiseen GameObject -muuttujaan.



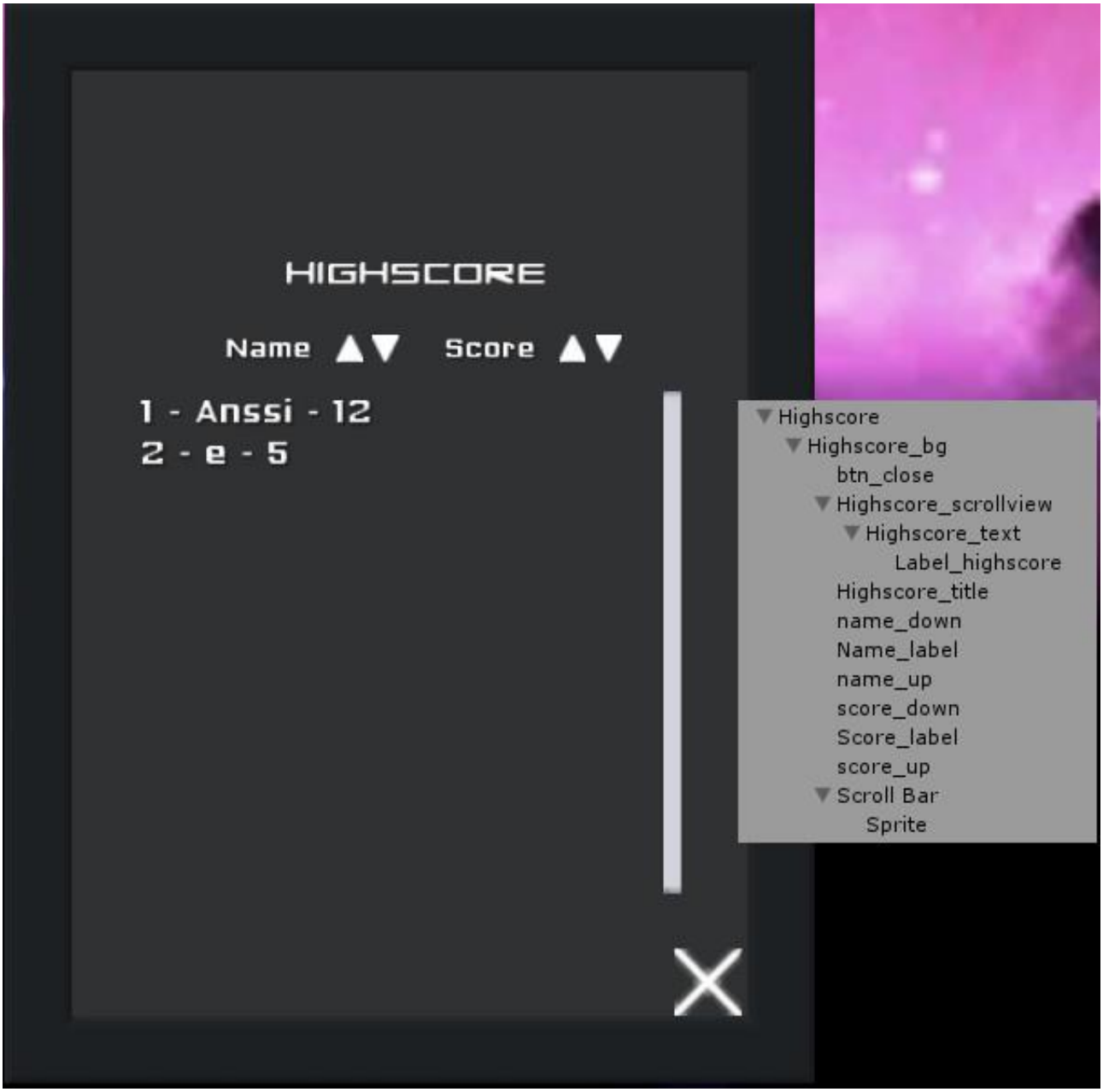
Kuva: GUIServerList-skripti instansioi templatesta severilistan.

```
public class GUIServerList : MonoBehaviour {

    public GameObject template;

    public int spacing = -60;

    public List<UILabel> items = new List<UILabel>();

    public UILabel getItem(int slot) {return items[slot];}

    public void fillList(List<ServerEntry> serverEntries)
    {
        if (template != null)
        {
            items.Clear();

            for(int i = 0; i < serverEntries.Count; i++)
            {
                // Passing server info for gameobject
                GameObject go = NGUITools.AddChild(gameObject, template);
                go.GetComponent<UILabel>().text = serverEntries[i].ToString();
                go.GetComponent<GUIServerButton>().address = serverEntries[i].address;
                // Positioning gameobject
                Transform t = go.transform;
                t.localPosition = new Vector3(0f, i * spacing, 0f);
                items.Add(go.GetComponent<UILabel>());
            }
        }
    }
}
```

Kuva: Serverilista instansioidaan, kun aiemmin haettu serverilista syötetään fillList-metodille.

4.3.3 **Highscore**

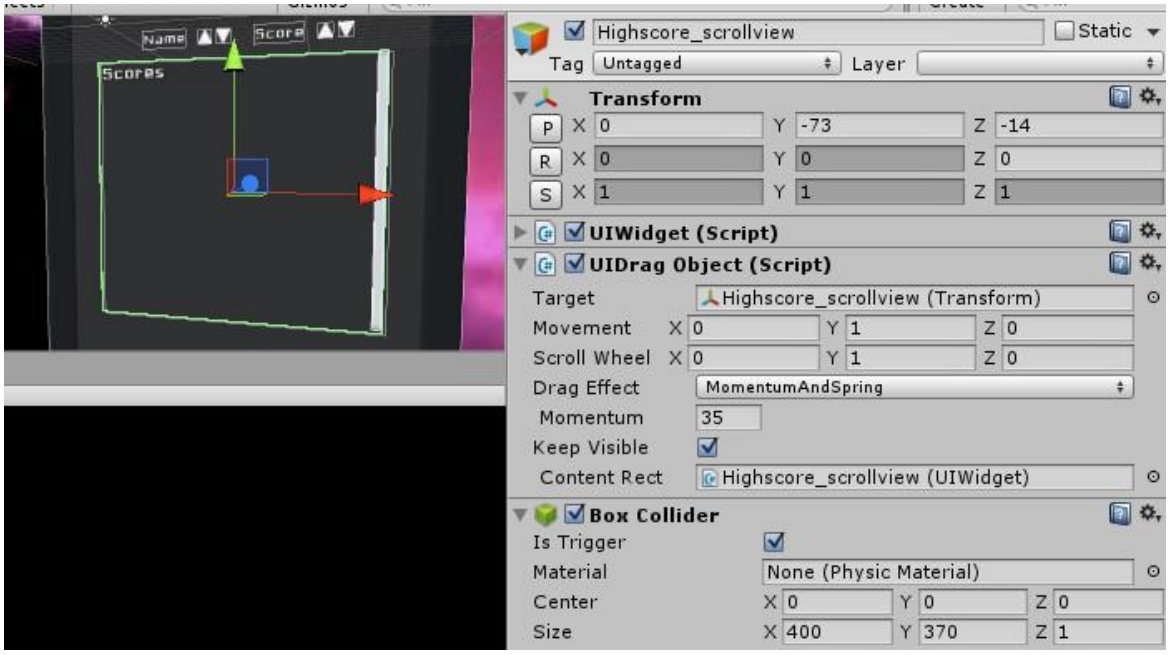
Highscore-paneelissa peli hakee serveriltä parhaat pisteet ja laittaa ne näkyville. Erityispiirteenä on vieritysominaisuus, jos pisteitä on enemmän kun varattuun tilaan mahtuu. Pisteitä pystyy myös järjestelemään pisteiden tai nimen mukaan eri järjestykseen.



Kuva: Highscore-paneelin rakenne.

4.3.3.1 **Pistealue**

Vieritettävä alue koostuu kolmen eri objektin kokonaisuudesta. Ylin objekti sisältää UIDragObject-skriptin. Siinä määritellään rullattavan alueen koko, vierityssuunta ja millainen "tuntuma" rullausefektissä on.



Kuva: UIDragObject-elementin rakenne.

Highscore_text-objektissa on vieritettävän tekstin toimintalogiikan sisältävä UITextList-skripti. Siihen asetetaan käytettävä vierityspalkki, joka luodaan myöhemmin. Sen lisäksi siinä on oma GUIHIghscore-skripti pisteiden hakemiselle ja järjestämiselle.

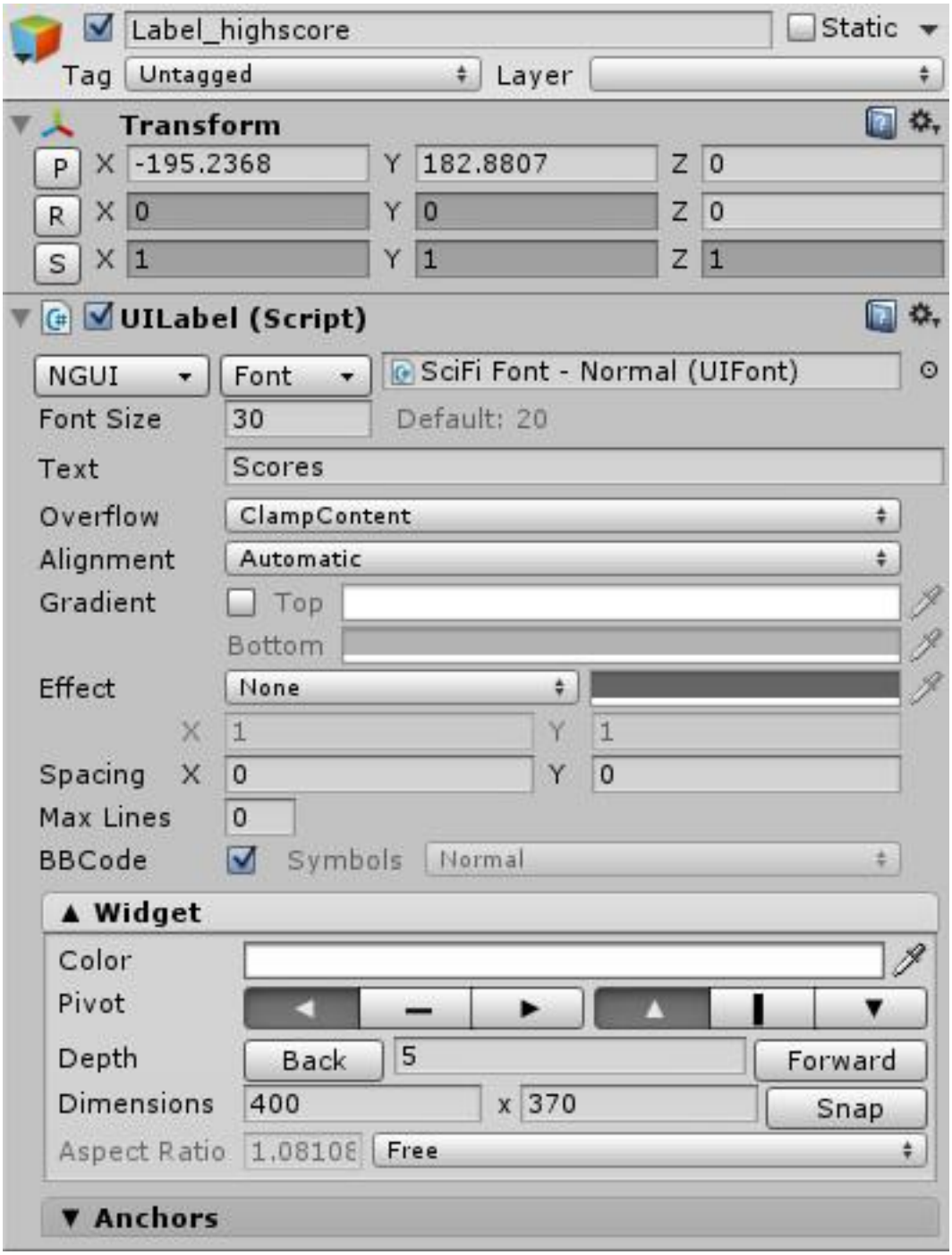
Kuva: Highscore_text-elementin rakenne.

GUIHighscore-skriptissä haetaan highscore ja järjestetään se haluttuun järjestykseen.

```
public class GUIHighscore : MonoBehaviour {

    public UILabel highscoreLabel;
    private HighScoreEntry[] highScoreList = new HighScoreEntry[2];

    public enum Order
    {
        NameUp,
        NameDown,
        ScoreUp,
        ScoreDown
    }

    void OnEnable ()
    {
        getOrderedHighscore(Order.ScoreUp);
    }

    public void getOrderedHighscore(Order order)
    {
        HighScore highScore = new HighScore();
        highScoreList = new HighScoreEntry[2];
        highScoreList[0] = new HighScoreEntry();
        if ( highScore != null )
        {
            highScoreList = highScore.GetHighScores("SlaughterGame",0, -1);
        }

        if ( highScoreList != null )
        {
            switch (order)
            {
            case Order.NameUp:
                Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                    return entry1.Name.CompareTo(entry2.Name);
                });
                break;
            case Order.NameDown:
                Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                    return entry2.Name.CompareTo(entry1.Name);
                });
                break;
            case Order.ScoreDown:
                Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                    return entry1.Score.CompareTo(entry2.Score);
                });
                break;
            case Order.ScoreUp:
                Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                    return entry2.Score.CompareTo(entry1.Score);
                });
                break;
            }

            GameObject scoreArea = GameObject.Find("Highscore_text");
            scoreArea.GetComponent<UITextList>().Clear();

            if (scoreArea != null)
            {
                int i = 0;
                foreach (HighScoreEntry hs in highScoreList)
                {
                    string text = ++i + " - " + hs.Name + " - " + hs.Score;
                    scoreArea.GetComponent<UITextList>().Add(text);
                }
            }
        }
    }
}
```

Kuva: GUIHighScore-skriptissä haetaan ja järjestetään pisteet, sekä syötetään ne UITextList-skriptille prosessoitavaksi.

Label_highscore-objekti on hyvin yksinkertainen UILabel-elementti. Sen tehtävä on näyttää UITextList-skriptin prosessoima teksti.

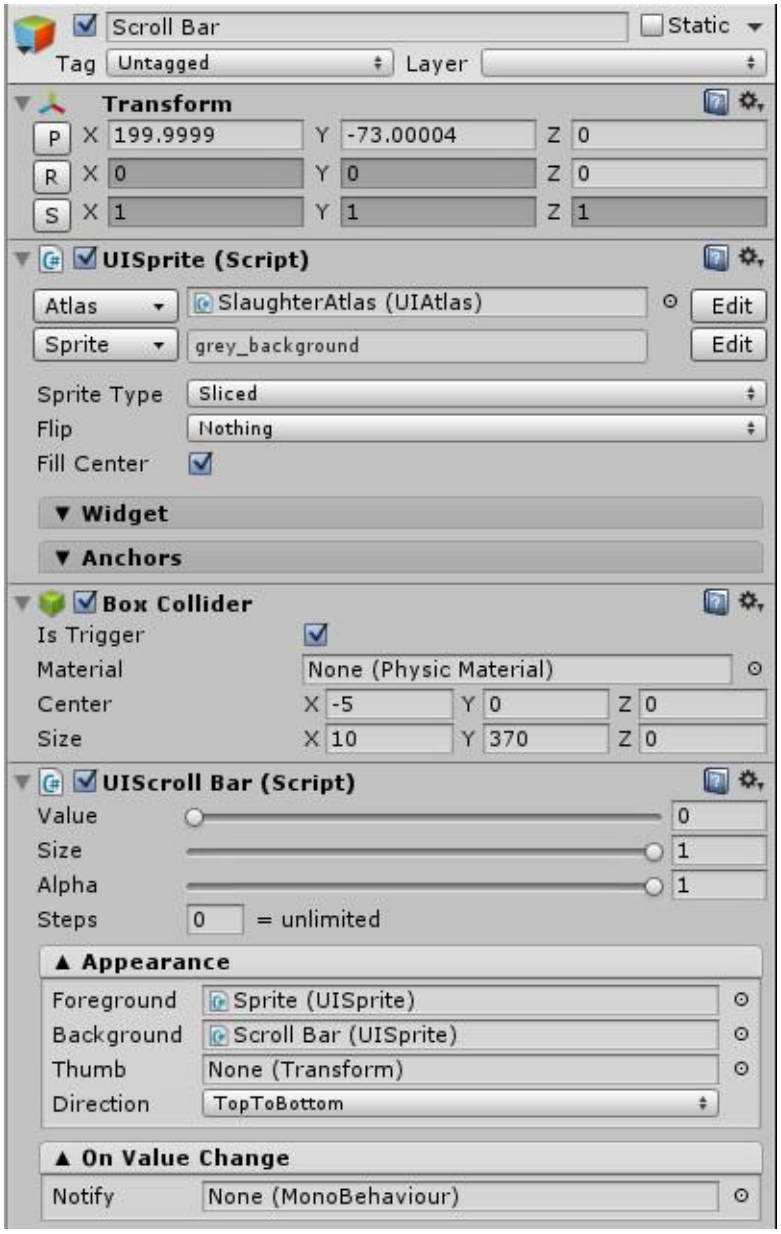


Kuva: Label_highscore-elementti sisältää UILabel-skriptin, jonka tehtävä on näyttää UITextList-elementin sisältämä teksti.

4.3.3.2 **Scroll Bar**

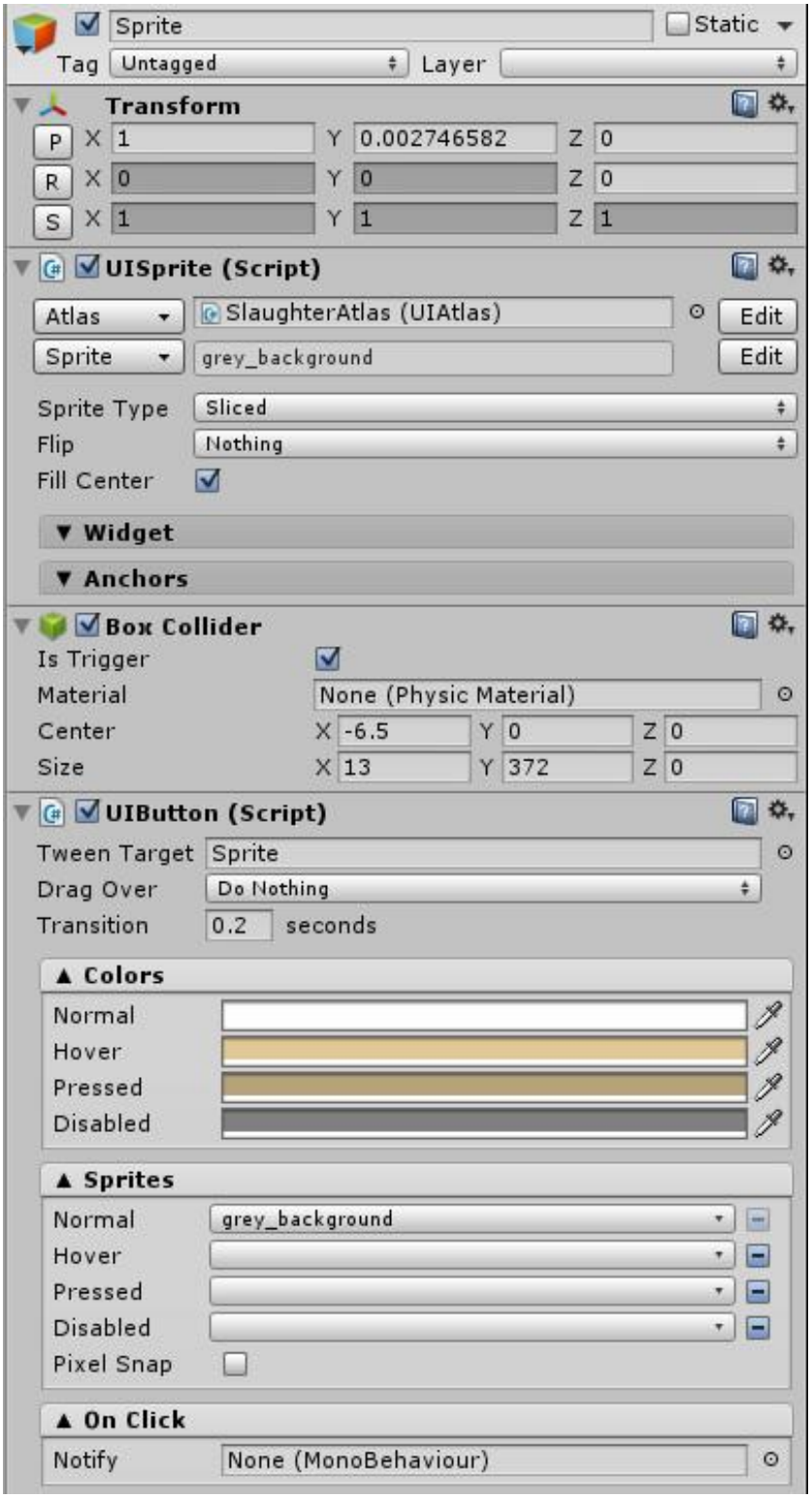
Vierityspalkki koostuu vedettävästä vieritysosasta, joka osoittaa vieritettävän tekstin sen hetkisen sijainnin, ja palkin taustaosasta, jota pitkin vieritysosaa liikutetaan.

Vierityspalkin toiminnallisuus löytyy UIScrollBar-skriptistä. Skriptiin täytyy säätää kohdilleen lähinnä palkin sijainti, suunta ja ulkonäkö. Taustaosaan tarvitsee asettaa BoxCollider triggeriksi asetettuna, jotta vierityspainike voidaan asettaa klikattuun kohtaan, jos pelaaja klikkaa sellaista kohtaa, jossa ei ole vierityspainiketta. Vierityspalkin yhdistäminen vieritettävään tekstiin tehdään asettamalla se aiemmin mainittuun UITextList-skriptiin.



Kuva: Vierityspalkin UIScrollBar-elementin rakenne.

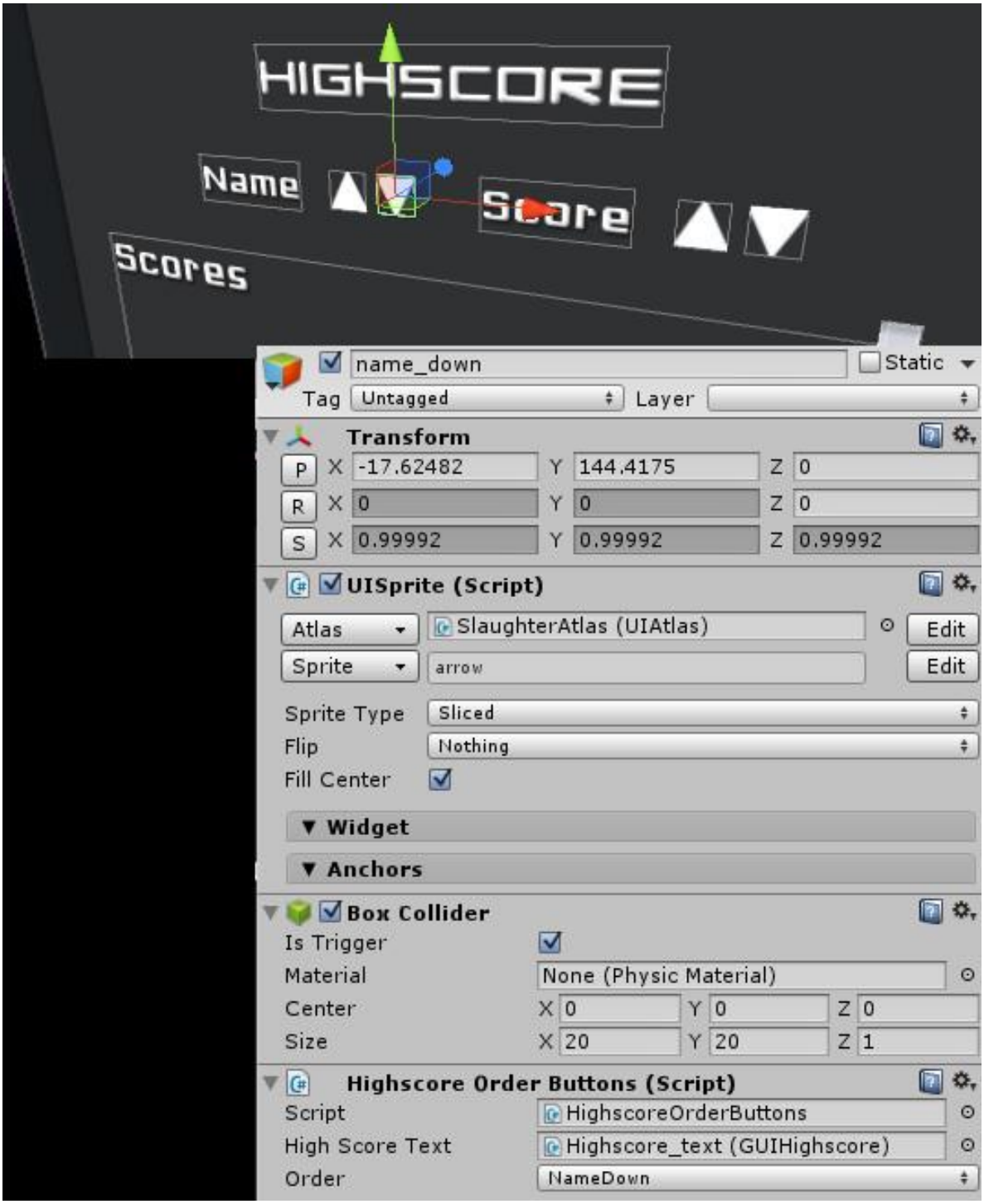
Vierityspainike on myös hyvin yksinkertainen. Ainut välttämätön elementti on BoxCollider triggeriksi asetettuna, jotta painiketta voidaan liikuttaa raahaamalla. UISprite-skriptillä painikkeelle asetetaan tekstuuri ja UIButton-skriptillä painikkeen sävyä voidaan vaihtaa sen mukaan onko hiiren kursori painikkeen päällä tai klikataanko sitä.



Kuva: Vierityspalkin painikkeen rakenne.

4.3.3.3 Järjestyspainikkeet

GUIHighscore-skriptissä oli jo valmiiksi toiminnallisuus pisteiden järjestämiseen nimen tai pisteiden mukaan. Paneeliin luodaan neljä painiketta, joista järjestystä voidaan vaihtaa pelin ollessa käynnissä.



Kuva: Järjestyspainikkeiden rakenne.

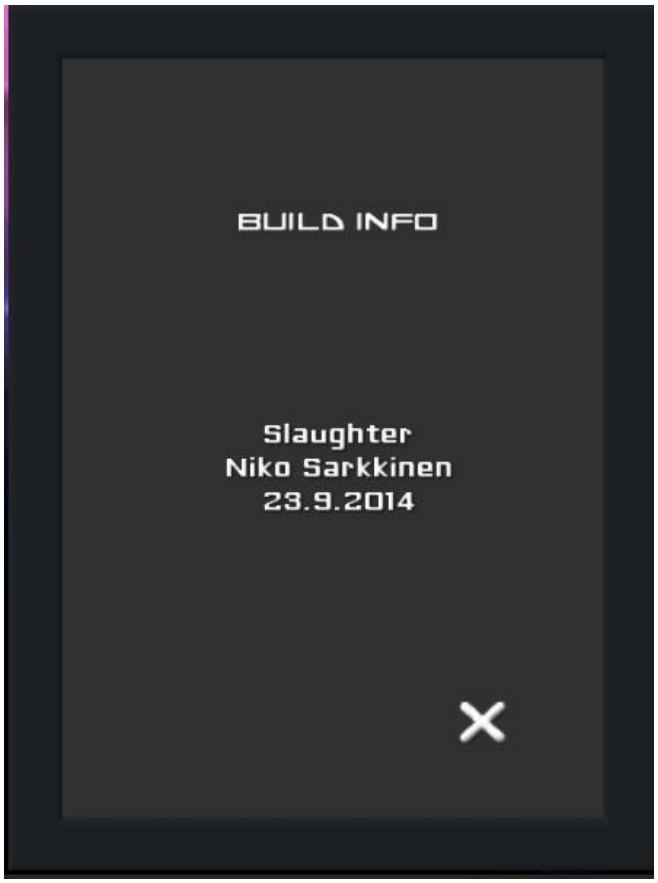
Painikkeen koodi on todella yksinkertainen. Se kutsuu GUIHighscore-skriptistä metodia, joka on tuttu aiemmasta kohdasta.

```
public class HighscoreOrderButtons : MonoBehaviour {

    public GUIHighscore highScoreText;
    public GUIHighscore.Order order;

    void OnClick()
    {
        highScoreText.getOrderedHighscore(order);
    }
}
```

Kuva: Järjestyspainikkeen koodia.

4.3.4 Build Info

Build Info -valikossa pelaaja saa tietoa buildin nimestä ja tekijästä sekä näkee ajankohdan milloin buildi on käännetty. Graafisten elementtien lisäksi tässä paneelissa olennainen asia on tietojen hakeminen ja näyttäminen UILabel-objektissa. Siihen lisätään scripti, jossa tietojen asettaminen ja näyttäminen suoritetaan Awake-metodissa. Tämä metodi suoritetaan aina kun objekti instansioidaan tai aktivoidaan, kuten UIButtonActivate-scripti tekee.



Kuva: Build Info -paneelin rakenne.

```
public class GUIBuildInfo : MonoBehaviour {

    string author;
    string productName;
    string buildTime;

    void Awake ()
    {
        author = "Niko Sarkkinen";
        productName = "Slaughter";
        buildTime = "23.9.2014";
        GetComponent<UILabel>().text = productName + "\n" + author + "\n" + buildTime;
    }
}
```

Kuva: Tiedot asetetaan yksinkertaisessa skriptissä.

4.3.5 Quit

Quit-painikkeesta avautuu ikkuna, jossa pelaajalta varmistetaan haluaako hän poistua pelistä. Ikkunan avautuminen toteutetaan UIButtonActivate-scriptillä. Avautuvassa paneelissa on kaksi painiketta. Enismmäisestä peli sulkeutuu siihen liitetyn yksinkertaisen scriptin toimesta. Toisesta paneeli sulkeutuu UIButtonActivate-scriptillä, missä Targetiksi on asetettu varmistusikkuna ja State ei ole ruksittu.

Kuva: Pelin sulkemisen varmistavan paneelin rakenne.

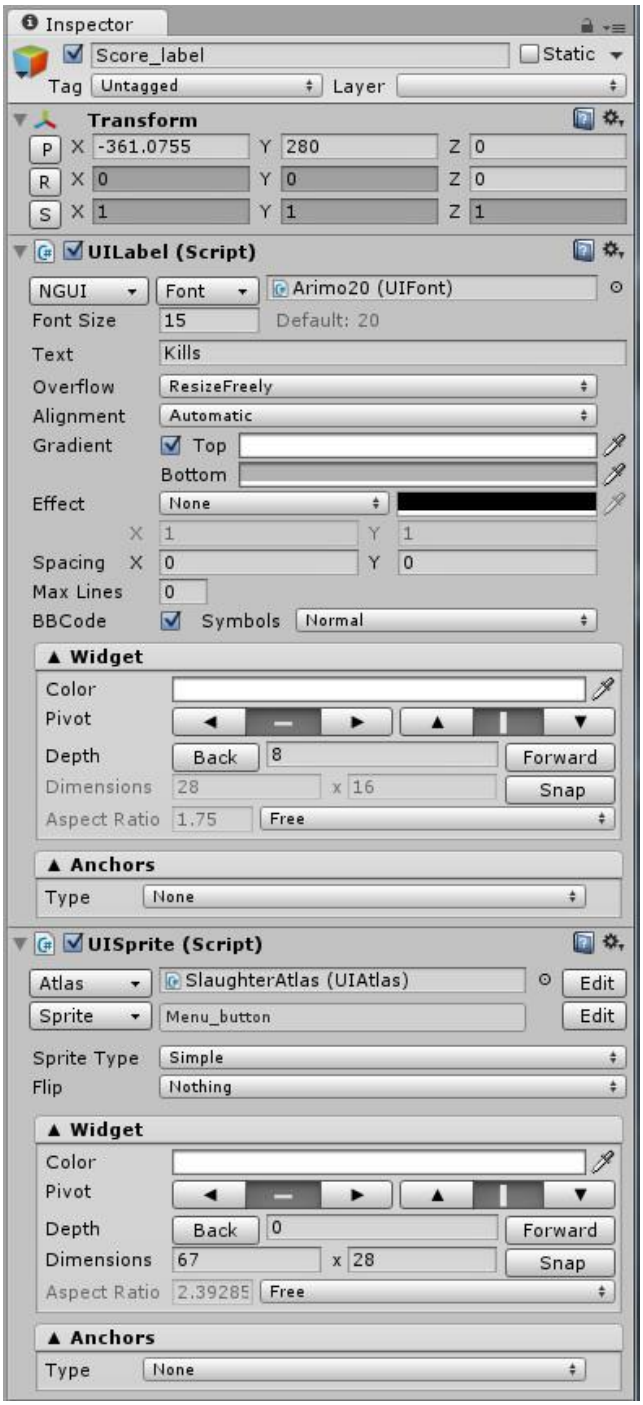
4.4 Pelinäkymä

Pelinäkymässä on näkyvissä erilaista tietoa pelin tapahtumista, chat ikkuna sekä painike pelistä poistumiseen. Tietojen näyttäminen NGUI:n UILabel-elementillä on tuttua alkuvalikon toteutuksesta. Chat-ikkunan toteutuksessa kannattaa hyödyntää NGUI:n esimerkkitiedostojen mukana tulevaa valmista mallitoteutusta.

Kuva: Pelinäkymä.

4.4.1 Kill Score

Kill Score on hyvin yksinkertainen GUI-elementti, joka on koko ajan näkyvissä peliä pelatessa. Siihen tarvitaan objekti mihin on liitetty UISprite-scripti taustakuvaa ja UILabel-scripti tekstiä varten. Komponenttien depth-arvot täytyy muistaa asettaa niin, että teksti on taustakuvan takapuolella.



Kuva: Kill Score -elementin rakenne ja asetukset.

Pelin koodissa tarvitsee löytää UILabel-skripti ja asettaa sen julkiseen text-muuttujaan haluttu pistemäärä.

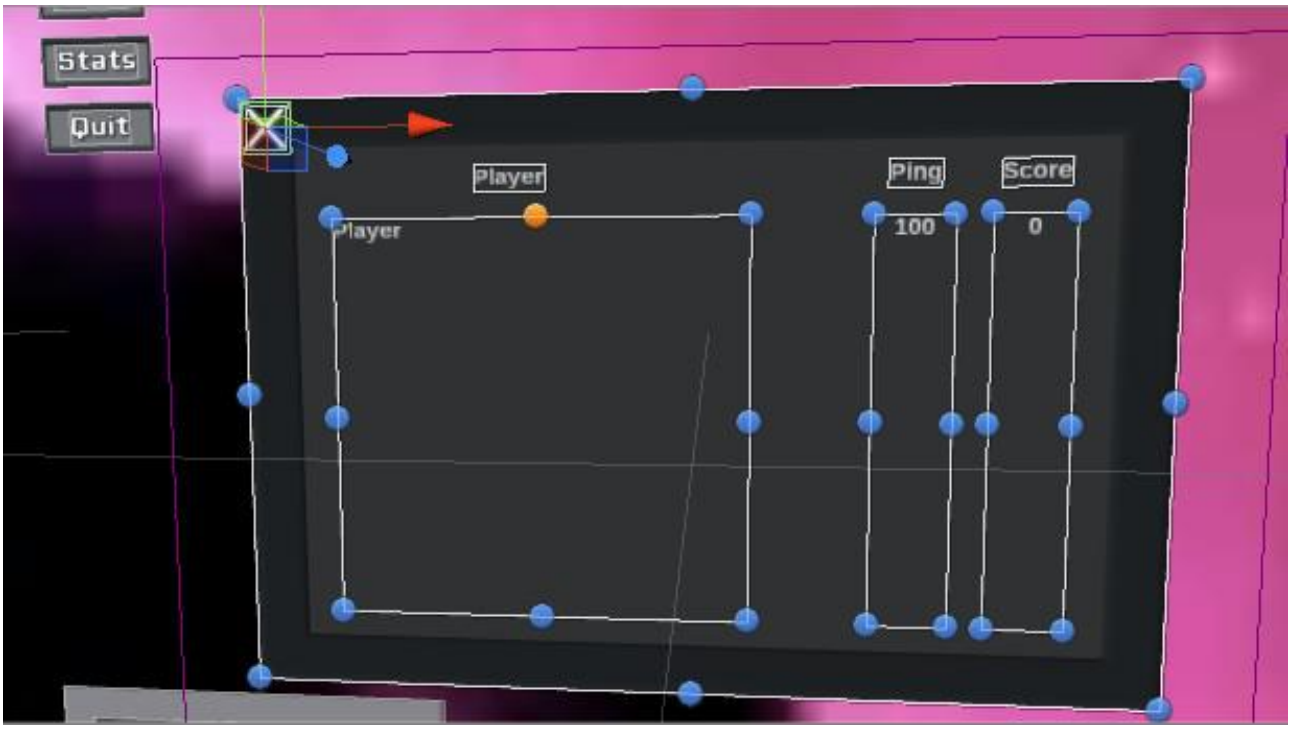
Käyttöliittymätutoriaalit

```
void OnTriggerEnter( Collider other )
{
    if ( Network.isServer )
    {
        // Server instance cheats!
        /* if ( other.gameObject.networkView.viewID ==... */

        if ( other.tag == "Ship" )
        {
            // your own dog won't bite you.
            if ( other.GetComponent<ShipController>().ownerId != ownerId )
            {
                // Blow up
                GameObject.Find("NetworkGUI").GetComponent<BattleScript>().KillPlayer(other.gameObject.networkView.viewID);
                GameObject.Find("NetworkGUI").GetComponent<GUIScript>().numberOfKills++;
                GameObject.Find("Score_label").GetComponent<UILabel>().text = "kills: " + GameObject.Find("NetworkGUI").GetComponent<GUIScript>().numberOfKills;

                //  might disappear faster
            }
        }
    }
```

Kuva: Pistemäärä päivitetään Kill Score -elementtiin koodissa.

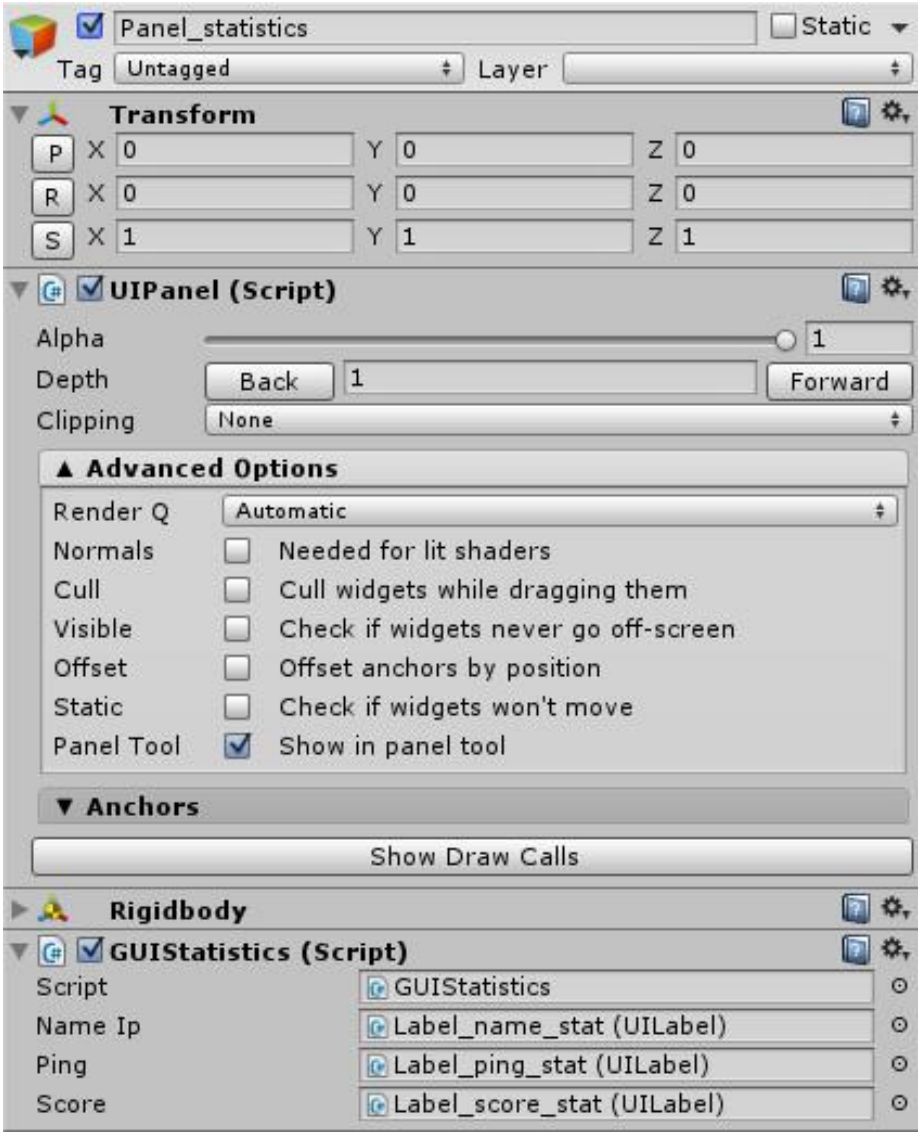
4.4.2 **Statistics-ikkuna**

Pelin tilasto-paneelistä pääsee katsomaan pelaajiin liittyvää tietoa kuten ip-osoite, ping-vasteaika ja pisteet. Paneeliin lisätään UILabel-elementeillä objektit, joihin tilastot syötetään. UILabel-elementit on jaettu kolmeen sarakkeeseen helpomman sijoittelun vuoksi.



Kuva: Statistics-paneelin rakenne.

Paneeliin lisätään skripti joka hakee ja asettaa tiedot UILabeleihin. Siihen asetetaan julkisiin UILabel-muuttujiin juuri tehdyt UILabel-objektit.



Kuva: Statistics-elementin komponentit.

GUIStatistics-skripti hakee pelaajien tiedot ja asettaa ne omiin UILabel-elementteihin OnEnable-metodissa.

```
public class GUIStatistics : MonoBehaviour {

    GUIScript guiScript;

    public UILabel nameIp;
    public UILabel ping;
    public UILabel score;

    void OnEnable()
    {

        if (guiScript == null)
        {
            guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
        }

        ConnectedPlayerStats[] connected = guiScript.GetPlayerStats();

        string nameiptext = "";
        string pingtext = "";
        string scoretext= "";

        foreach (ConnectedPlayerStats player in connected)
        {
            nameiptext = nameiptext + player.name + " " + player.address + "\n";
            pingtext = pingtext + player.ping.ToString() + "\n";
            scoretext = scoretext + player.score.ToString() + "\n";
        }

        nameIp.text = nameiptext;
        ping.text = pingtext;
        score.text = scoretext;
    }
}
```

Kuva: Tilastot haetaan GUIScript-skriptistä ja ne asetetaan UILabel-elementteihin.

Skriptissä käytettävä GetPlayerStats-funktio on määritelty GUIScript-skriptissä.

```
public struct ConnectedPlayerStats
{
    public string name;
    public string address;
    public int ping;
    public int score;
}
```

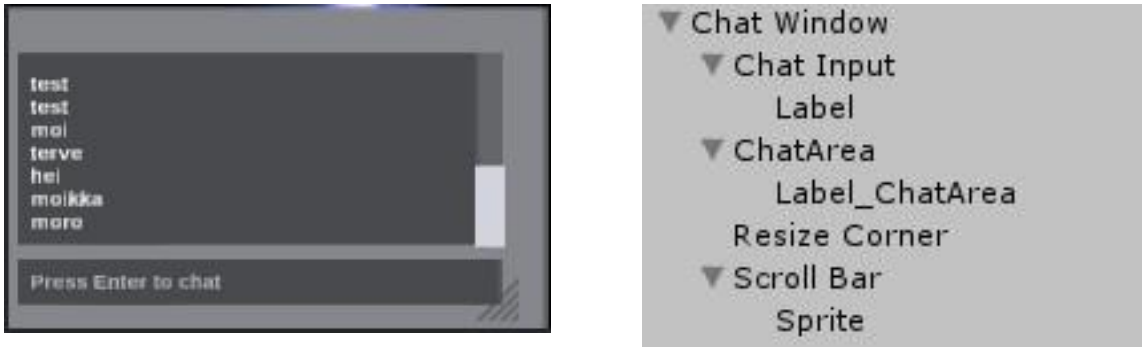
Kuva: Struct tilastojen tallentamiseen.

```
public ConnectedPlayerStats[] GetPlayerStats()
{
    ConnectedPlayerStats[] players = new ConnectedPlayerStats[Network.connections.Length+1];

    players[0].name = "Server";
    players[0].address = ipAddress;
    players[0].ping = 0;

    for (int i = 1; i <= players.Length; i++)
    {
        players[i].name = "Player " + i;
        players[i].address = Network.connections[i].ipAddress;
        players[i].ping = Network.GetAveragePing(Network.connections[i]);
    }

    // Score
    for(int i = 0; i < score.Length; i++)
    {
        for(int j = 0; j < score.Length; j++)
        {
            if (int.Parse(Network.connections[i].guid) == score[j].id)
            {
                players[i].score = score[j].id;
            }
        }
    }

    return players;
}
```

Kuva: Tilastot haetaan ja palautetaan GetPlayerStats-funktiossa.

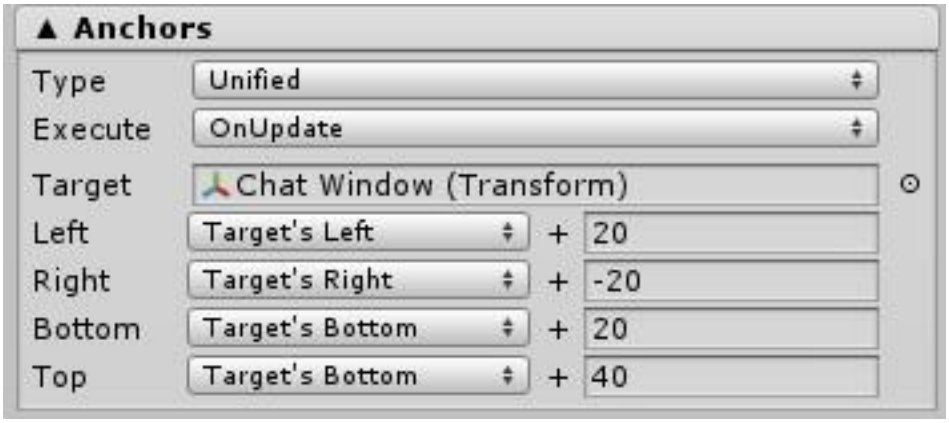
4.4.3 Chat-ikkuna

Chatin toteuttamisessa kannattaa hyödyntää NGUI:n mukana tulevaa esimerkkitiedostoa nimeltä Example 12 – Chat Window. Chatin toteutuksessa on kolme eri selkeästi eroavaa elementtiä. Nämä ovat Input-kenttä, mihin teksti syötetään, viestialue, missä kaikki oma sekä muiden pelaajien kirjoittama teksti näkyy sekä ScrollBar-elementti, joka mahdollistaa tekstin vierittämisen, jos sitä on enemmän kuin näyttökenttään mahtuu. NGUI:n valmiissa esimerkissä on mukana myös mahdollisuus muuttaa ikkunan sijaintia ja kokoa.



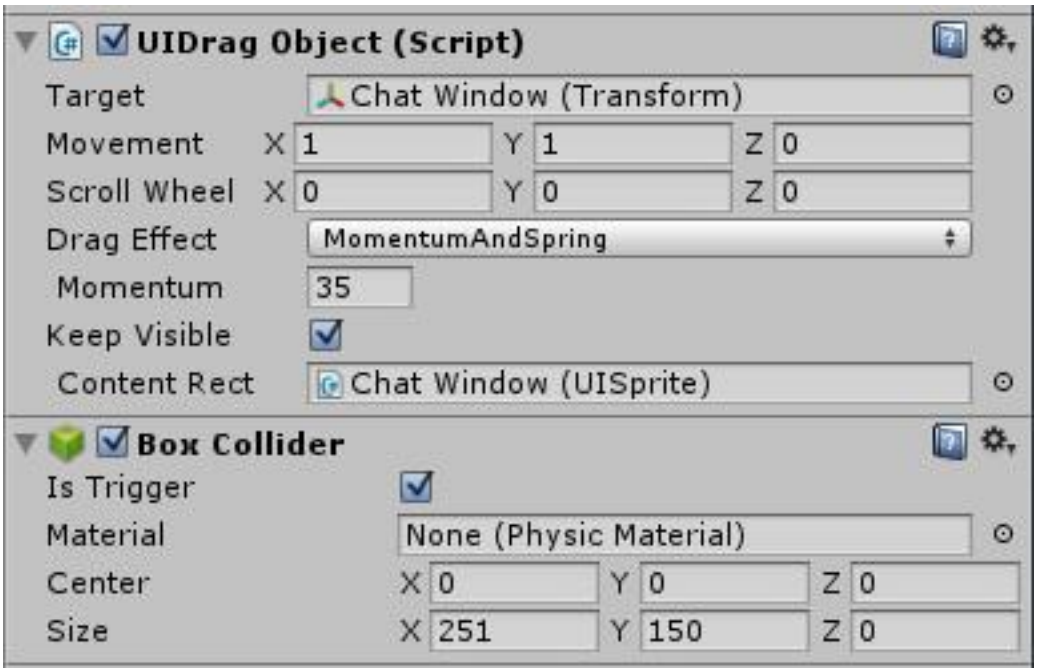
Kuva: Chat-ikkunan rakenne

Chat-ikkunan ylimpänä nodena on objekti, jolla voidaan muuttaa ikkunan kokoa ja sijaintia pelin ollessa käynnissä. Tämä objekti ei ole chatin toiminnan kannalta välttämätön, mutta se on niin hieno NGUI:n ominaisuus, että se voidaan jättää mukaan valmiista esimerkistä. Kaikissa Widgeteissä on asetettu Anchors-parametreihin mihin kohtaan ne ankkuroidaan, jotta ne pysyvät oikean kokoisina, kun ikkunan kokoa muutetaan.



Kuva: Widgettien nurkat ankkuroidaan kohde-elementtiin.

Ikkunan sijainnin ja koon muuttamisen hoitaa UIDragObject-skripti. Scriptiin tulee Target kohtaan objeketi jota halutaan siirtää, eli tässä tapauksessa tämä sama objekti, missä scripti on kiinni. Movement-kenttässä määritellään, mihin suuntaan objektia voidaan raahata. Tässä tapauksessa asetetaan x- ja y-akseleiden arvoksi 1 ja laitetaan syvyyssuunnassa siirtäminen pois päältä asettamalla z-akselin arvoksi 0. Objekti tarvitsee myös triggeriksi asetetun BoxColliderin minkä koko ja sijainti täytyy asettaa sen mukaan, mistä objektia halutaan pystyttävän raahaamaan.



Kuva: Raahattavan panelin asetuksia.

4.4.3.1 Viestialue

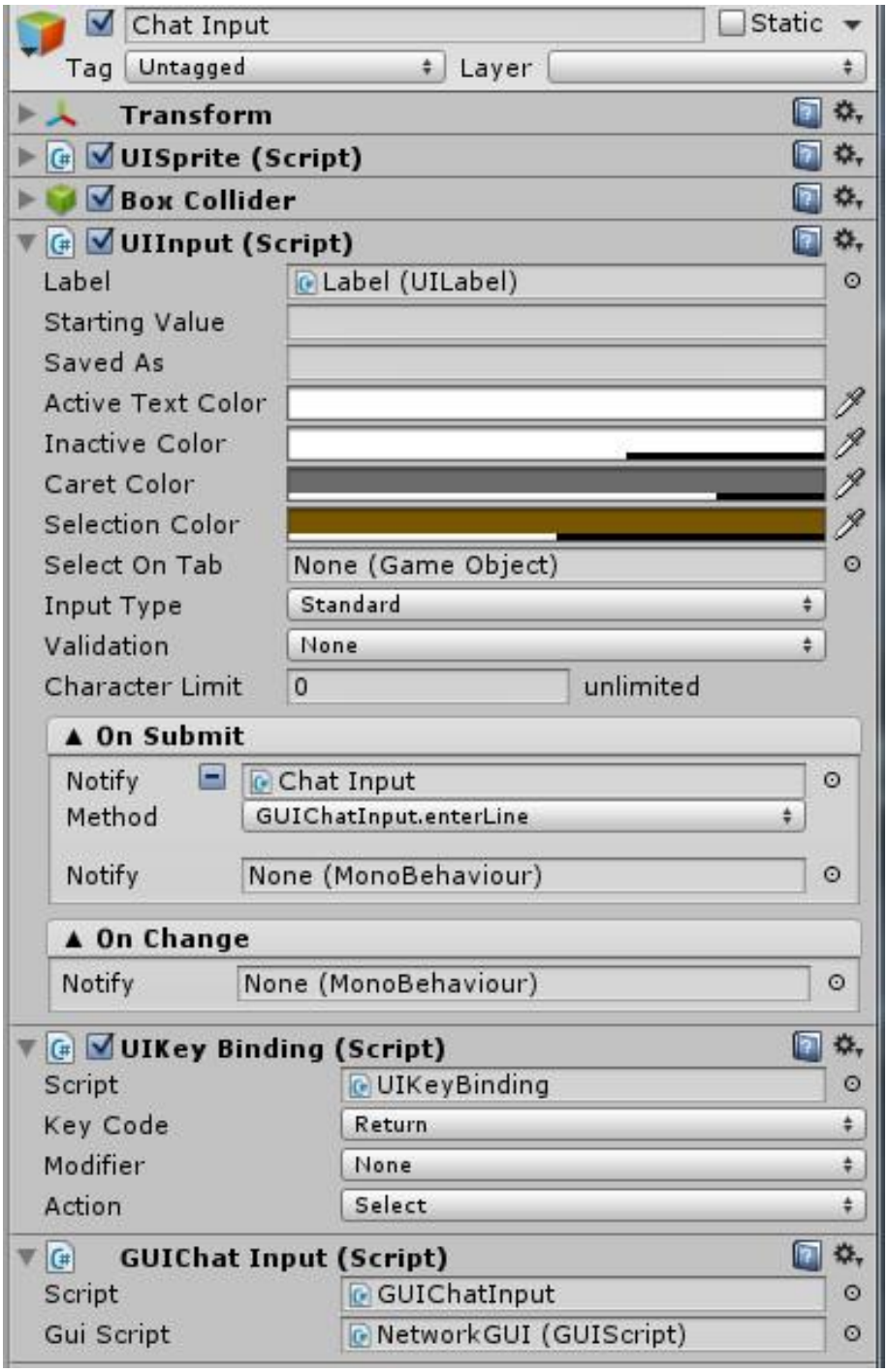
Viestialueen erityispiirre on se, että rivimäärän ylittäessä rajatun alueen, on viestihistoriaa pystyttävä selaamaan hiiren rullalla tai vierityspalkilla. Viestialue-objektiin liitetty UITextList-skripti sisältää vieritettävän tekstin toimintalogiikan. Skriptin Text Label ja Scroll Bar -kenttiin täytyy asettaa mistä löytyy tekstin sisältävä UILabel- ja UIScrollBar-skriptit. Style-valikkoon asetetaan arvoksi "Chat", ja Paragraph History -kenttään asetetaan kuinka monta riviä tekstiä halutaan pitää muistissa. Objektiin pitää liittää triggeriksi asetettu BoxCollider määrittämään miltä alueelta hiiren rullaaminen halutaan rekisteröidä.

Kuva: UITextList-komponentin asetuksia.

Viestialueessa näkyvä teksidata tallennetaan objektin ala-nodena olevaan objektiin liitettävään UILabel-skriptiin. Skriptiin täytyy asettaa Overflow-asetukseksi ClampContent. Tällä asetuksella ylimenevä teksti leikataan pois näkyvistä ylä- ja alareunasta.

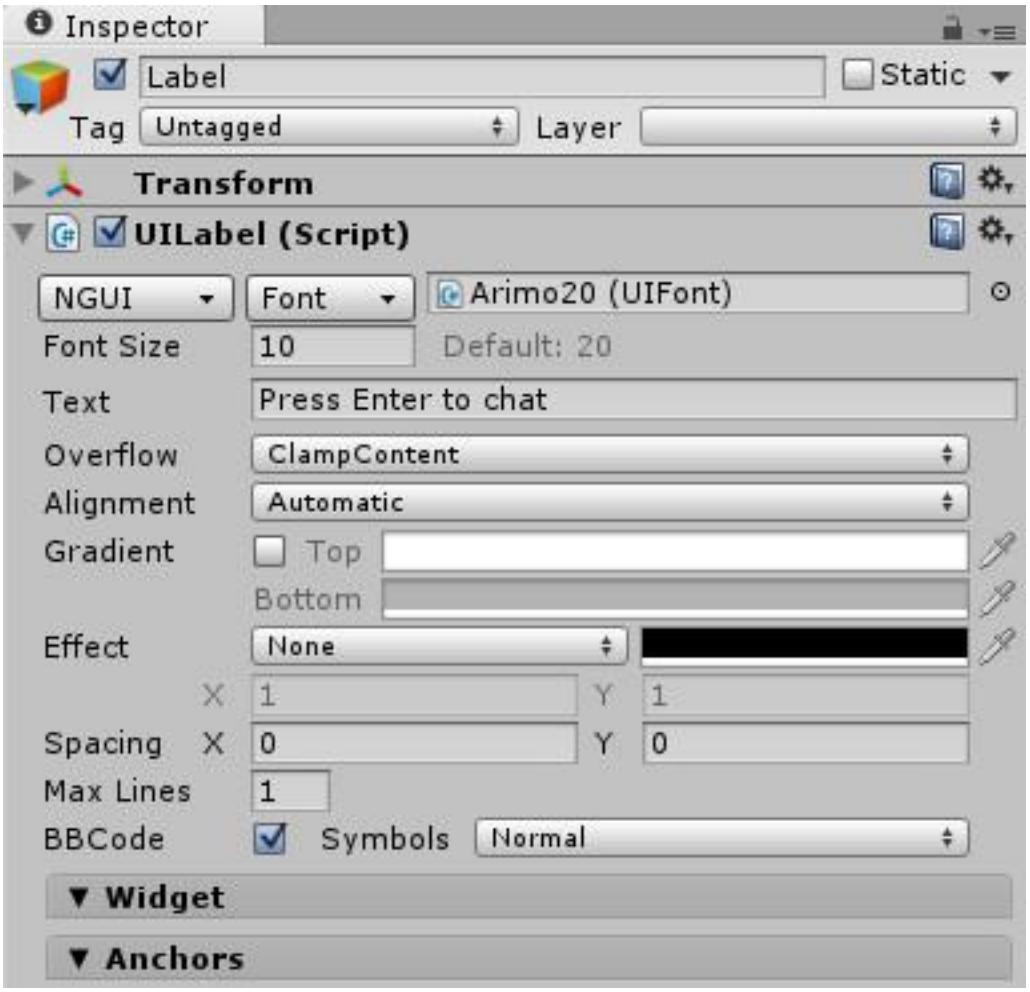
4.4.3.2 Syöttökenttä

Teksti syötetään chattiin Chat Input -kenttään. UISpriten ja triggeriksi asetetun BoxColliderin lisäksi Input-objektiin tarvitaan UIInput-skripti syötetyn tekstin lukemiseen sekä UIKeyBinding-skripti, joka asetetaan rekisteröimään Return-napin painaminen. Chatin toimimiseen verkon yli tarvitaan hieman omaa koodia, ja tässä esimerkissä se löytyy GUIChatInput-skriptistä.



Kuva: Syöttökentän rakenne ja asetukset.

UIInput-skriptin Label-kenttään asetetaan objektin ala-nodena oleva UILabel-objekti. Tämä UILabel-objekti näyttää vakiotekstin, kun syöttökenttä ei ole aktiivinen ja aktiivisena sen tekstin, mitä pelaaja on kirjoittamassa.



Kuva: UILabel-elementin rakenne.

GUIChatInput-skriptin tehtävä on ottaa UIInput-skriptin value-muuttujassa oleva arvo ja lähettää se käsiteltäväksi GUIScript-skriptin chat-toiminnosta vastaavalle koodille.

```
[RequireComponent(typeof(UIInput))]
public class GUIChatInput : MonoBehaviour {

    public GUIScript guiScript;
    UIInput mInput;
    string text;

    public void EnterLine()
    {

        mInput = GetComponent<UIInput>();
        text = mInput.value;

        if (guiScript == null)
        {
            guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
        }

        guiScript.ChatMessage(GetComponent<UIInput>().value);

        mInput.value = "";
    }
}
```

Kuva: GUIChatInput-skriptin koodia.

GUIScriptin Chat-metodi lähettää kirjoitetun viestin kaikille yhdistetyille clienteille. MessageQueue-luokan Enqueue-metodia täytyy myös muokata niin, että sen sisältämät viestit lähetetään näytettäväksi UITextList-skriptille.

```
public new void Enqueue(string msg)
{
    if ( Count > m_MaxMessagesInQueue ) Dequeue();
    base.Enqueue(msg);

    GameObject chatArea = GameObject.Find("ChatArea");
    chatArea.GetComponent<UITextList>().Add(msg);
}
```

Kuva: GUIScript-skriptin MessageQueue-luokka lähettää tekstin UITextList-elementille.

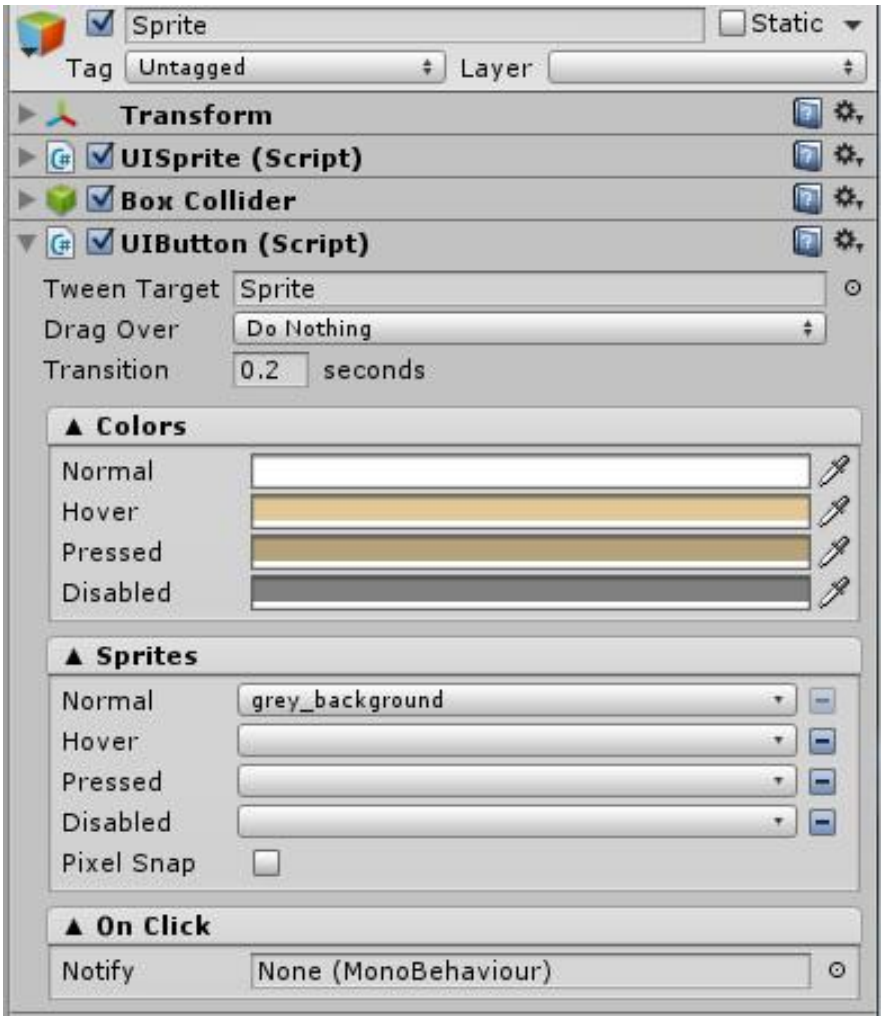
4.4.3.3 Vierityspalkki

Vierityspalkki koostuu vedettävästä vieritysosasta, joka osoittaa vieritettävän tekstin sen hetkisen sijainnin, ja palkin taustaosasta, jota pitkin vieritysosaa liikutetaan.

Vierityspalkin toiminnallisuus löytyy UIScrollBar-skriptistä. Skriptiin täytyy säätää kohdilleen lähinnä palkin sijainti, suunta ja ulkonäkö. Taustaosaan tarvitsee asettaa BoxCollider triggeriksi asetettuna, jotta vierityspainike voidaan asettaa klikattuun kohtaan, jos pelaaja klikkaa sellaista kohtaa, jossa ei ole vierityspainiketta. Vierityspalkin yhdistäminen vieritettävään tekstiin tehdään asettamalla se UITextList-skriptiin, joka sijaitsee viestialueen objektissa.

Kuva: Vierityspalkin asetukset.

Vierityspainike on myös hyvin yksinkertainen. Ainut välttämätön elementti on BoxCollider triggeriksi asetettuna, jotta painiketta voidaan liikuttaa raahaamalla. UISprite-skriptillä painikkeelle asetetaan tekstuuri ja UIButton-skriptillä painikkeen sävyä voidaan vaihtaa sen mukaan onko hiiren kursori painikkeen päällä tai klikataanko sitä.



Kuva: Vierityspainikkeen asetukset.

4.4.4 Quit

Quit-painikkeessa on UISprite- ja UILabel-komponenttien lisäksi BoxCollider ja UIButtonActivate-scripti. UIButtonActivate-scriptiin laitetaan Target-kenttään QuitConfirmation-paneeli ja State-arvo laitetaan päälle.

Quit-painikkeesta avautuu ikkuna, jossa pelaajalta varmistetaan haluaako hän poistua pelistä. Avautuvassa paneelissa on kaksi painiketta. Enismmäisestä peli sulkeutuu siihen liitetyn yksinkertaisen scriptin toimesta. Toisesta paneeli sulkeutuu UIButtonActivate-scriptillä, missä Targetiksi on asetettu varmistusikkuna ja State ei ole ruksittu.

Kuva: Pelistä poistumisen varmistusikkuna.

5 IGUI

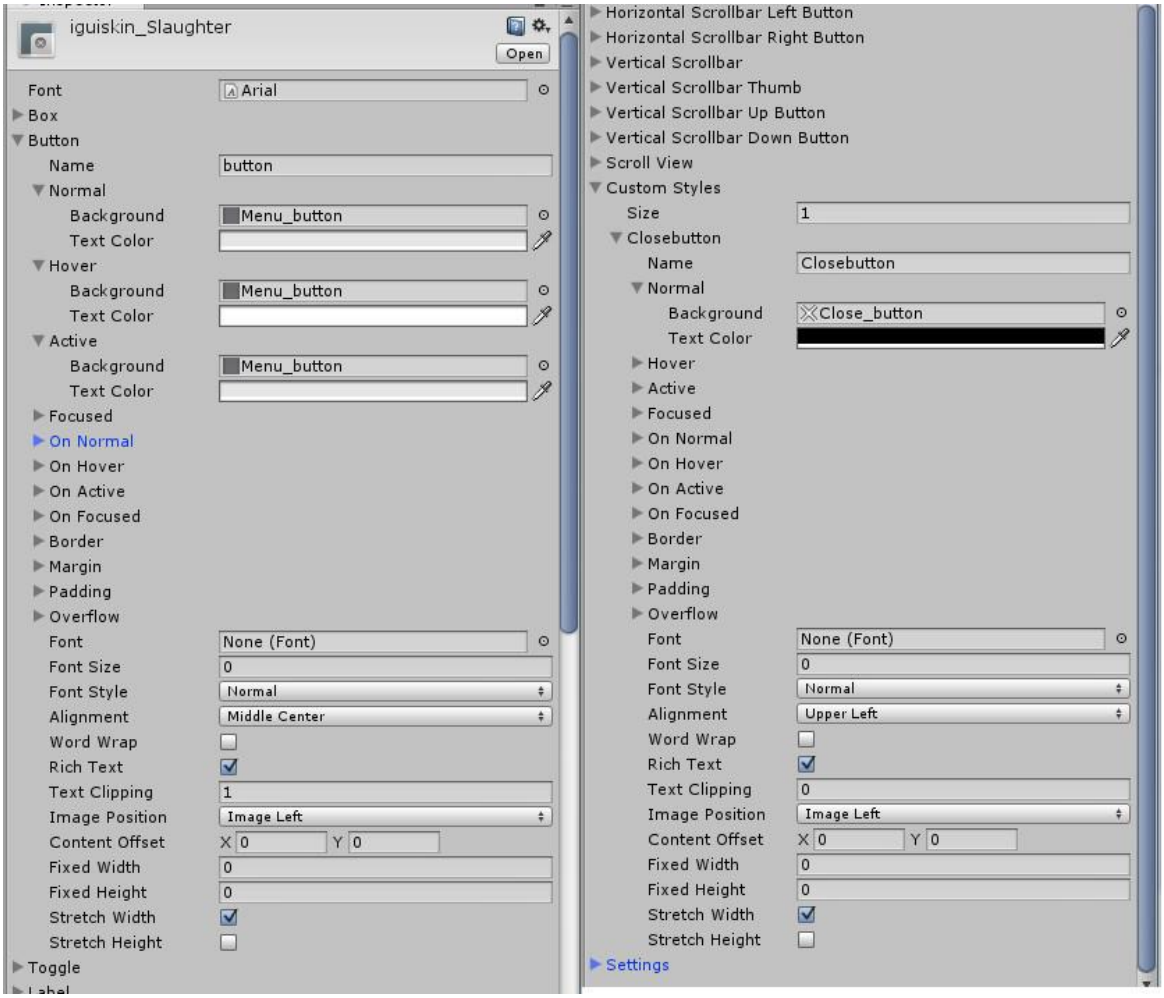
5.1 GUI Skin

GUI Skin tulee valmiiksi Unityn mukana ja iGUI hyödyntää sitä toiminnallisuudessaan. Se on asetustiedosto, johon säädetään käyttöliittymän asetukset. Siinä asetetaan tekstuurit, fontit, fontin koot ja paljon muuta painikkeille, paneeleille, tekstikentille sekä kaikille muille elementeille. Asetustiedoston säätöjä voi muokata missä tahansa vaiheessa käyttöliittymän tekoa ja uudet asetukset päivittyvät koko käyttöliittymään.

Elementtien asetuksiin voi säätää esimerkiksi eri tekstuurit sen tilan mukaan. Eri tiloja on esimerkiksi normaali tila tai hiiren pitäminen napin päällä eli hoover-tila. Elementtien tekstin kokoa ja sijaintia voi asetella haluamanlaisekseen. Myös tekstin väritystä voi muutaa elementin eri tilan mukaan.

Erilaisten nappien tekemiseen voi käyttää Custom Styles kohtaa minne voi tehdä poikkeavanlaisia elementtejä. Tosin kirjoitushetkellä Custom Stylesin käyttö on todella vaikea ja käytännössä erilaiset elementit joutuu tekemään joka kerta erikseen. Tähän on kuitenkin iGUI:n kehittäjä luvannut parannusta.

Oman GUI Skinin saa luotua klikkaamalla Assets → Create → GUI Skin.



Kuva: GUI Skin asetuksia.

5.2 IGUI Toolbox

IGUI:ssa on käyttöliittymän luonnin helpottamiseksi olemassa toolbox-työkalu. Sen saa auki klikkaamalla Window-> iGUI → Toolbox. Toolbarin etusivu on melko selkeä. Siitä löytyy napit eri elementtien piirtämiselle. Uuden käyttöliittymän tekeminen aloitetaan talon näköisestä Root-painikkeesta.

Käytössä olevan UIRootin asetuksia pääsee muokkaamaan Settings-painikkeesta. Asetuksista voidaan asettaa käytössä oleva GUI Skin. Painamalla ”Apply to existing” -painiketta päivittyy GUI Skiniin tehdyt muutokset myös kaikkiin jo tehtyihin elementteihin. Code Settings -kohdasta voi valita millä kielellä IGUI generoi kooditiedoston(tästä lisää myöhemmin). Koodin automaattinen luonti eli Batching-nappi on se, jossa on kaksi sinistä nuolta. Se on päivittää kooditiedoston nykyisen scenen mukaiseksi ja sitä tulee painettua monta kertaa käyttöliittymää tehdessä.

Kuva: iGUI Toolbox.

5.3 IGUICode-tiedosto

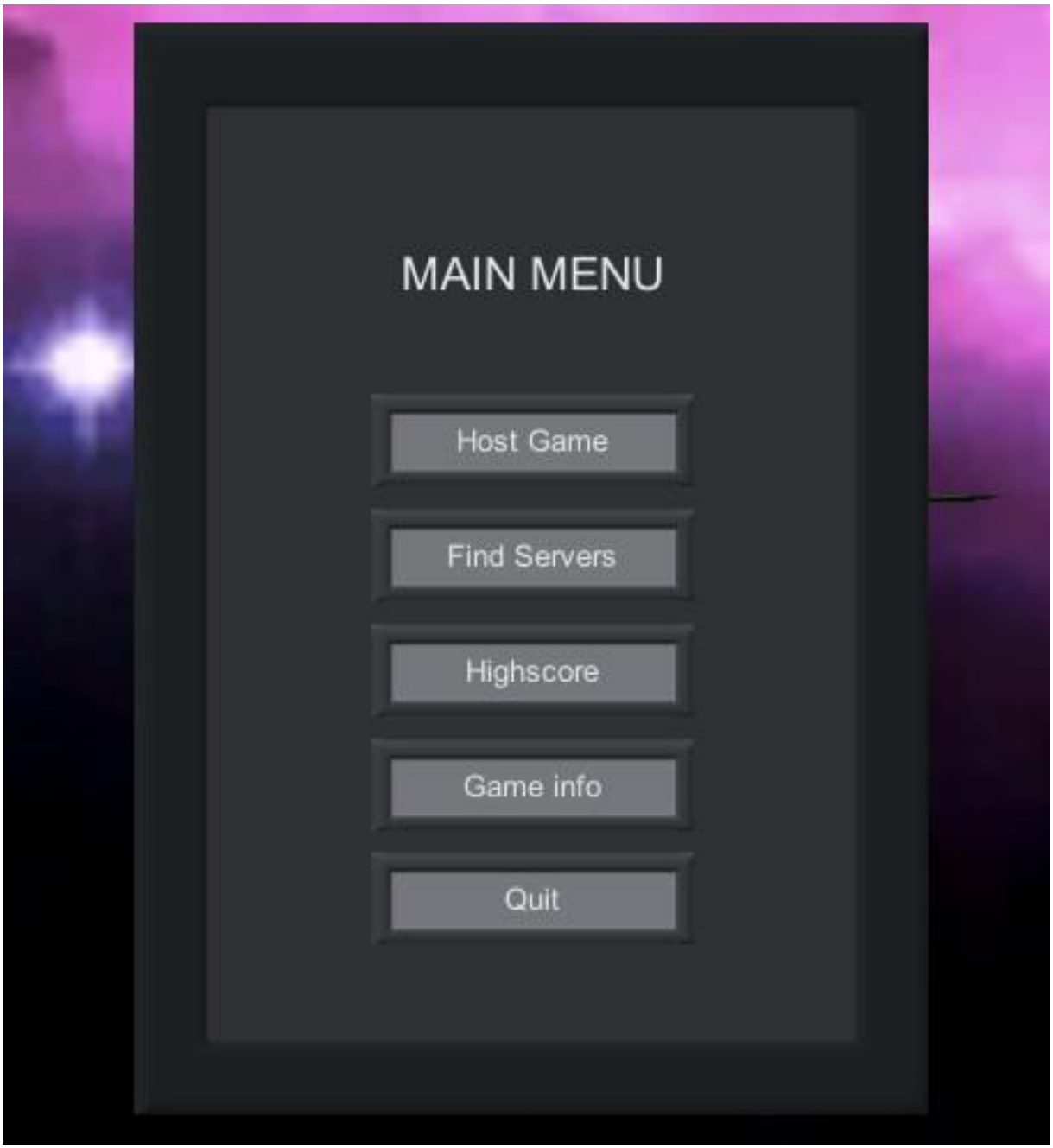
IGUI Rootin luonnin yhteydessä IGUI luo automaattisesti projektin iGui-kansioon kooditiedoston, missä käyttöliittymän toiminta voidaan määritellä. Tiedostossa on listattuna kaikki käyttöliittymän elementit. Elementtien toimintoja voi lisätä tiedostoon klikkaamalla elementtiä hiiren oikealla napilla pelinäkymässä ja valitsemalla halutun toiminnon. Tämä generoi tiedostoon funktion, joka ajetaan, kun toiminto triggeröityy.

Elementtejä nimetessä täytyy ottaa huomioon, että jokainen elementti täytyy nimetä kahdesti. IGUICode-skripti lukee elementistä skriptissä olevan nimen, eikä objektin nimeä.

Kuva: Click-metodin generointi IGUICode-tiedostoon.

5.4 Alkuvalikko

Alkuvalikko toteutetaan niin, että pelaajalle avautuu aluksi päävalikko, josta voi toimintapainikkeilla avata uusia valikoita. Valikoista pääsee pois sulkemisnapista.

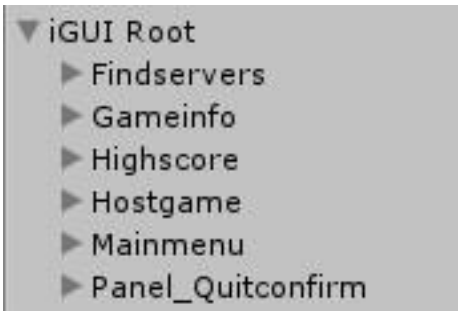


Kuva: Alkuvalikko.

5.4.1 Päävalikko

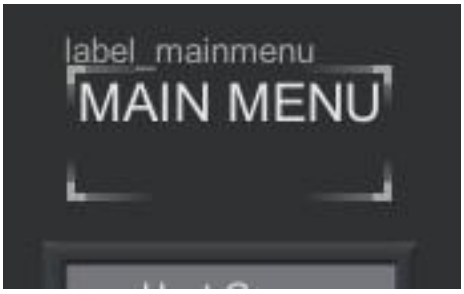
Luodaan uusi iGUI Root objekti Toolboxin Root-painikkeesta. IGUI Rootin asetukset kannattaa käydä läpi ja varmistaa, että ne ovat omalle projektille sopivat. Oma Gui Skin asetetaan tähän tiedostoon, jos sellainen on luotu. Muutoin vakioasetukset toimii useimmissa tapauksissa. Scenekohtainen IGUICode skripti on kiinni Rootissa.

Rootin alle luodaan paneeli-objektit päävalikkoa ja alavalikoita varten. Niiden luonti tapahtuu valitsemalla Toolboxista Panel-työkalu ja raahaamalla Game-näkymässä halutun kokoinen elementti. Myös IGUIPanel-skriptien vakioasetusten pitäisi olla toimivat. Kannattaa tietysti varmistaa, että ne on omaa projektia varten oikein.



Kuva: Päävalikon rakenne.

”Main Menu” -tekstiotkiskko toteutetaan Label-elementillä. Toolboxista valitaan Label-työkalu ja Game-näkymässä raahataan elementti sijainniltaan ja kooltaan oikeanlaiseksi. Tässä vaiheessa kannattaa varmistaa, että elementti näyttää hyvältä ja tallentaa mahdollisesti muuttuneet asetukset GUI Skiniin. Tällöin tulevatkin label-elementit ovat heti oikeanlaisia.



Kuva: Label-elementtiä voi säätää halutunkokoiseksi.

5.4.2 Toimintopainikkeet

Alkuvalikon paneeliin luodaan painikkeet alavalikoiden avaamiseen. Valitaan Toolboxista Button-työkalu ja raahataan Game-näkymässä painikkeesta halutun kokoinen. Painikkeen ulkonäkö kannattaa taas heti asettaa oikeanlaiseksi ja laittaa asetukset GUI Skiniin.

Kun nappeja tekee monta, kannattaa muistaa asettaa skriptissä Variable Name -kenttään joku kuvaava nimi. Tämä on se nimi, jolla elementti näkyy koodissa.

Kun kaikki Painikkeet on luotu, kannattaa Toolboxissa mennä Settings-näkymään ja klikata Batch painiketta. Se generoi iGuiCode-tiedostoon juuri luodut painikkeet.

Kuva: Batch-painike löytyy Toolboxista.

IGUICode tiedostossa pitäisi nyt näkyä kaikki tähän mennessä luodut elementit. Painikkeiden toiminnallisuus tehdään Click-metodin sisälle. Click-metodi generoidaan tuplaklikkaamalla painiketta Game-näkymässä, tai klikkaamalla sitä hiiren oikealla napilla ja valitsemalla haluttu metodi.

Tämän jälkeen avataan kooditiedosto, ja toteutetaan painikkeiden toimintalogiikka. Kuvassa kahden ylimmän painikkeen metodit.

```
public void btn_hostgame_Click(iGUIButton caller)
{
    mainmenu_panel.enabled = false;
    hostgame_panel.enabled = true;
}

public void btn_findserver_Click(iGUIButton caller)
{
    mainmenu_panel.enabled = false;
    findservers_panel.enabled = true;
}
```

Kuva: IGUI:n luomat esimerkkimetodit.

5.5 Alavalikot

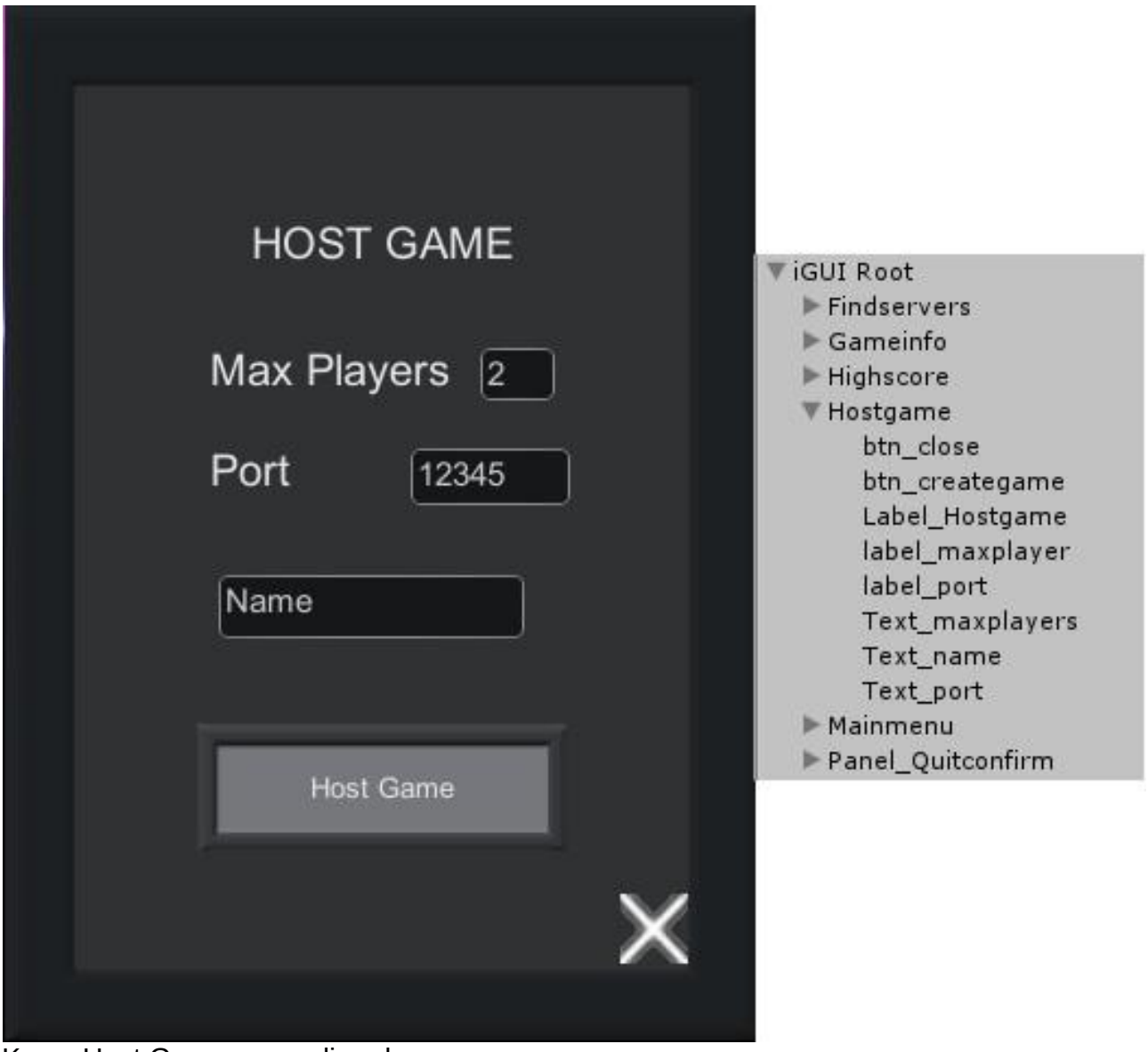
Alavalikoissa on jokaisessa oma erityinen toiminnallisuutensa. Yhteistä niille on vain sulkemispainike, jolla kyseinen paneeli sulkeutuu ja Main Menu -paneeli avautuu. Jokaisen paneelin sulkemispainikkeen Variable Name kannattaa asettaa samaksi, jolloin kooditiedostossa riittää yksi metodi kaikkien toiminnallisuuden määrittämiseen.

```
public void btn_close_Click(iGUIButton caller)
{
    // close all panels
    hostgame_panel.enabled = false;
    findservers_panel.enabled = false;
    highscore_panel.enabled = false;
    gameinfo_panel.enabled = false;

    // enable main menu
    mainmenu_panel.enabled = true;
}
```

Kuva: Alkuvalikon painikkeiden koodi.

5.5.1 Host Game

Host Game -paneelin tärkein toiminnallinen asia on pelaajan syöttämien tietojen lukeminen ja uuden pelin aloittaminen näiden tietojen pohjalta. Toolboxista löytyy erikseen syöttökentät tekstille ja numeroille. Pelaajan nimi kysytään tekstikentällä ja pelaajien määrä ja portti numerokentällä. Numerokenttään voi asettaa rajoituksia numeron ylä- ja alarajalle ja ottaa desimaaliluvut pois käytöstä.



Kuva: Host Game -paneelin rakenne.

Host Game -painikkeelle generoidaan Click-metodi ja siihen syötetään toiminnallisuus serverin käynnistämiselle.

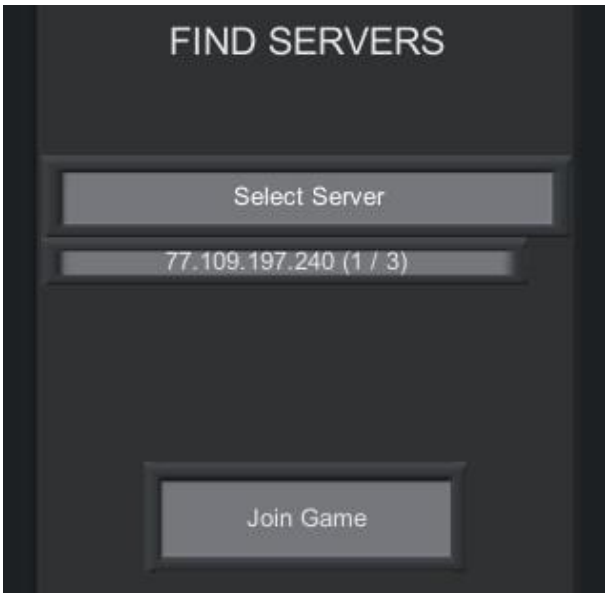
```
// Starting a server
public void btn_creategame_Click(iGUIButton caller)
{
    // Getting the values from input fields
    int maxPlayers = (int)text_maxplayers.value;
    int port = (int)text_port.value;
    string playerName = text_name.value;

    // Checking if values are within accepted limits
    if(playerName.Equals("")) {playerName = "DefaultPlayer";}
    if(maxPlayers > 16) {maxPlayers = 16;}
    if(maxPlayers < 2)  {maxPlayers = 2;}

    GameObject.Find("NetworkGUI").GetComponent<GUIScript>().LaunchServer(maxPlayers, port, playerName);
}
```

Kuva: Host Game -paneelin koodia.

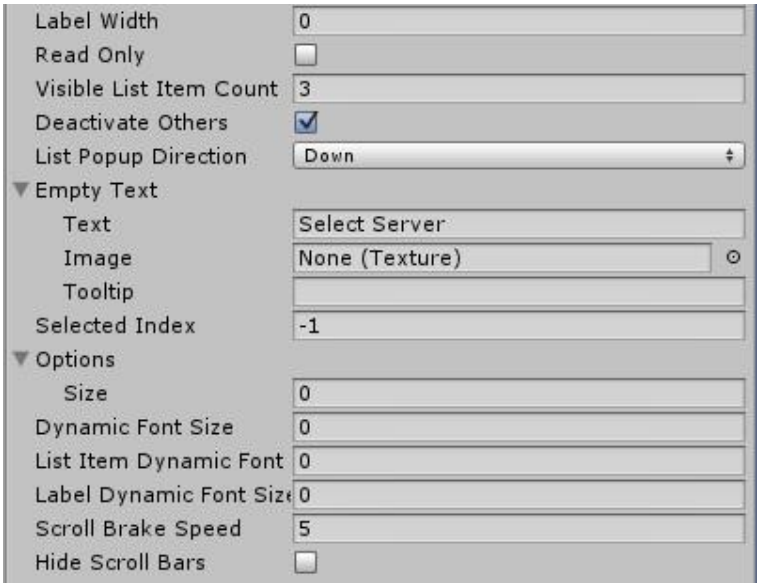
5.5.2 Find Servers

Find Servers -paneelissa listataan kaikki serverit, ja annetaan pelaajalle mahdollisuus liittyä valittuun peliin. Serverit listataan Drop Down List -elementtiin, mistä valittu serveri luetaan Join Game -nappia painettaessa.

Drop Down List -elementti luodaan valitsemalla se Toolboxista ja raahaamalla siitä sopivan kokoinen Game-näkymässä. Ensimmäiseksi kannattaa käydä laittamassa Label Width -arvo nollaksi, ettei elementin Label-osa hankaloita elementin sijoittelua. Asetuksista laitetaan Empty Text -kohtaan mikä teksti vakiovalinnassa näkyy sekä sen Index-arvo. Index-arvo kannattaa jättää -1, ja käyttää sitä myöhemmin koodissa hyödyksi. Options-kohtaan Size-arvoksi laitetaan 0, koska valinnat täytetään myöhemmin koodissa. Visible List Item Count -kenttään voi itse valita sopivan määrän. Kun servereiden määrä ylittää listassa tuon arvon, ilmestyy reunaan vierityspalkki ja ylimenevät serverit täytyy rullata esiin.



Kuva: Find Servers -paneeli.



Kuva: DropDownList-elementin asetuksia.

Drop Down Listille generoidaan iGUICode-tiedostoon Init- ja ListActivate-metodit. Init metodissa pyydetään MasterServeriä päivittämään serverilista. ListActivate-metodissa serverilista syötetään Drop Down Listille sisällöksi.

```
public void dd_serverselect_Init(iGUIDropDownList caller)
{
    GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
    // Update server list
    guiScript.UpdateServerList();
}

public void dd_serverselect_ListActivate(iGUIDropDownList caller)
{
    GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();

    // Clear options
    dd_serverselect.removeAll();

    // Fill options
    foreach( ServerEntry s in guiScript.serverList)
    {
        GUIContent cont = new GUIContent();
        cont.text = s.ToString();
        dd_serverselect.addOption(cont);
    }
}
```
Kuva: Serverilistan koodia.

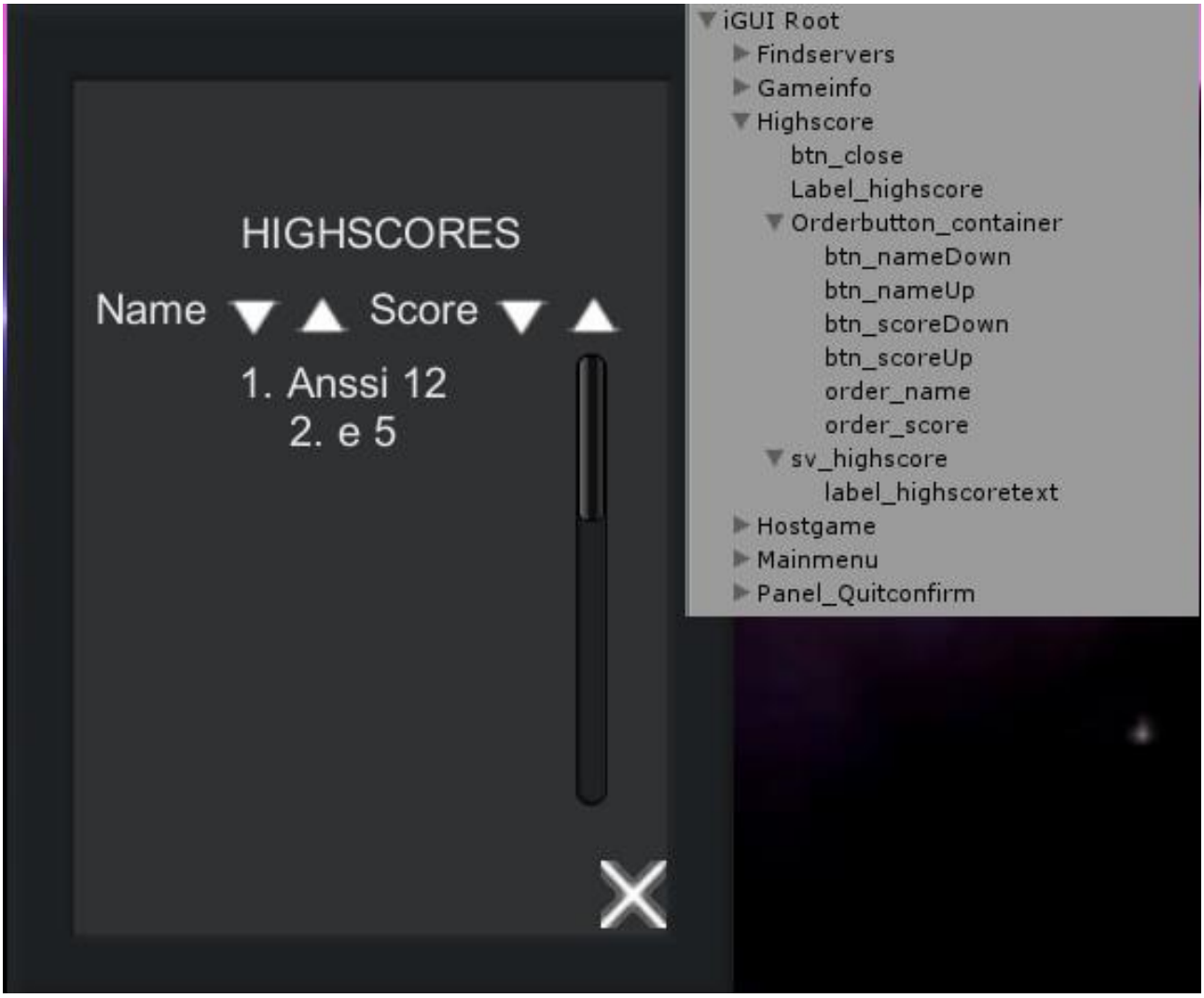
Join Game -painikkeelle generoidaan Click-metodi. Metodissa tarkistetaan, ettei valintojen Index-arvo ole -1. Serverilistasta haetaan valinnan mukaisen serverin ip-osoite, ja yhdistetään peli siinä osoitteessa sijaitsevaan serveriin.

```
public void btn_joinserver_Click(iGUIButton caller)
{
    // Joins the selected server
    if (dd_serverselect.selectedIndex != -1 )
    {
        GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
        string address = guiScript.serverList.ToArray()[dd_serverselect.selectedIndex].address;
        NetworkConnectionError err = Network.Connect( address, guiScript.GetPort());
        switch( err )
        {
        case NetworkConnectionError.NoError:
            Debug.Log("Waiting for reply...");
            break;
        default:
            Debug.Log("Could not connect: " + err );
            break;
        }
    }
}
```

Kuva: Valitulle serverille liittymisen koodia.

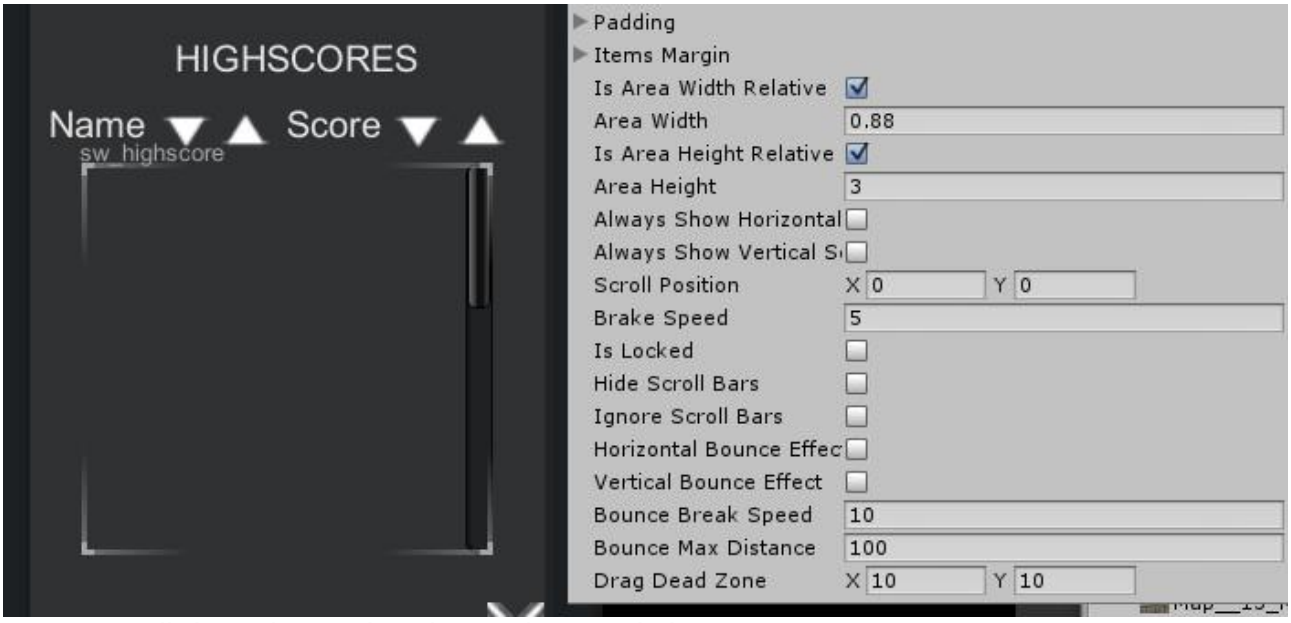
5.5.3 Highscore

Highscore-paneelissa peli hakee serveriltä parhaat pisteet ja laittaa ne näkyville. Erityispiirteenä on vieritysominaisuus, jos pisteitä on enemmän kun varattuun tilaan mahtuu. Pisteitä pystyy myös järjestelemään pisteiden tai nimen mukaan eri järjestyksiin. Vieritettävä alue koostuu kahdesta elementistä, joista toinen määrittää näkyvän alueen, ja toinen sisältää näytettävän tekstin. Järjestystä kontrolloivat painikkeet on sijoitettu omaan Container-elementtiin helpompaa siirtämistä ja hallitsemista varten.



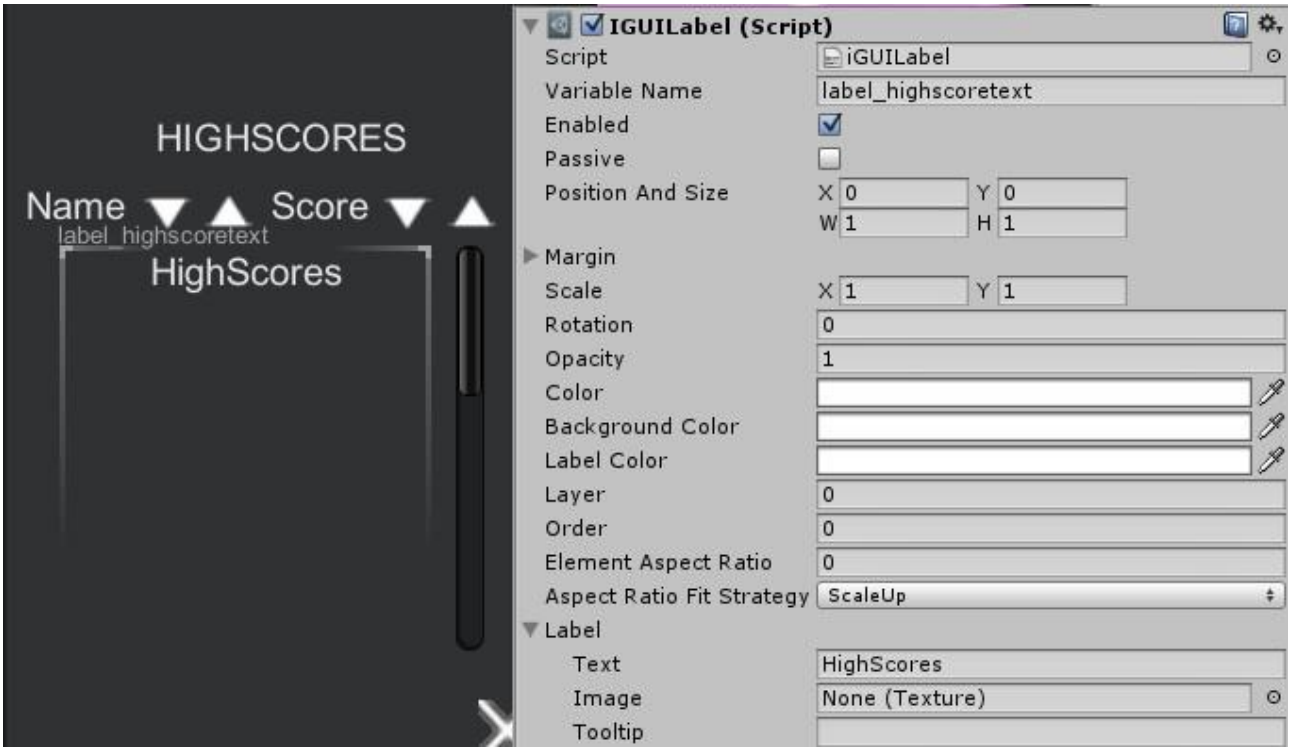
Kuva: Highscores-paneelin rakenne.

Vieritettävä alue koostuu kahdesta päällekkäisestä elementistä. Ylempi elementti on Scroll View -elementti. Se määrittää näkyvän alueen, ja sisältää vierityspalkkien toiminnallisuuden. Näkyvä alue määritellään elementin kokoa säätämällä. Alla olevan Label-elementin kokoa säädetään Area Width ja Area Height -kentissä. Tässä tapauksessa aluetta halutaan vierittää korkeussuunnassa, joten vieritettävän alueen korkeuden määritellään olevan 3 kertaa isompi kuin itse elementin koko.



Kuva: ScrollView-elementin rakenne ja asetuksia.

Scroll View -elementin childiksi asetetaan Label-elementti. Tälle elementtille ei tarvitse juuri säätöjä asettaa. Sijainti täytyy asettaa koordinaatteihin (0,0), ja koko arvoihin (1,1). Näillä arvoilla Label-elementti asettuu Scroll View -skriptissä määritettyjen Area Width ja Area Height -arvojen mukaiseksi.



Kuva: Label-elementin asetuksia.

Label-elementtiin asetetaan tekstidata sen Init-metodissa. Se generoidaan klikkaamalla sitä hiiren oikealla Game näkymässä ja valitsemalla ”Init”. Generoidaan samalla Click-metodit kaikille painikkeille, joilla voidaan muuttaa tekstidatan näyttöjärjestystä. IGUICode tiedostossa nämä metodit laitetaan hakemaan tekstidataa GUIScript-kooditiedostosta.

```
public void label_highscoretext_Init(iGUILabel caller)
{
    GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
    label_highscoretext.label.text = guiScript.GetHighscoreText(GUIScript.Order.ScoreUp);
}

public void btn_nameUp_Click(iGUIButton caller)
{
    GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
    label_highscoretext.label.text = guiScript.GetHighscoreText(GUIScript.Order.NameUp);
}
public void btn_nameDown_Click(iGUIButton caller)
{
    GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
    label_highscoretext.label.text = guiScript.GetHighscoreText(GUIScript.Order.NameDown);
}
public void btn_scoreUp_Click(iGUIButton caller)
{
    GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
    label_highscoretext.label.text = guiScript.GetHighscoreText(GUIScript.Order.ScoreUp);
}
public void btn_scoreDown_Click(iGUIButton caller)
{
    GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
    label_highscoretext.label.text = guiScript.GetHighscoreText(GUIScript.Order.ScoreDown);
}
```

Kuva: Pisteiden järjestyksen vaihtavien painikkeiden koodia.

GUIScript-tiedostossa määritellään Order-enum, jolla haetun datan järjestys valitaan. Tekstidatan hakeva ja järjestykseen laittava funktio muodostuu taulukon järjestyksen muokkaavasta switch-lohkosta, ja taulukon tekstidataksi parsettavasta foreach-loopista.

```
public enum Order
{
    NameUp,
    NameDown,
    ScoreUp,
    ScoreDown
}

public string GetHighscoreText(Order order)
{
    string hsText = "";
    if (highScoreList[0] == null)
    {
        highScoreList = HandleHighscores();
    }

    if (highScoreList != null)
    {
        switch (order)
        {
        case Order.NameUp:
            Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                return entry1.Name.CompareTo(entry2.Name);
            });
            break;
        case Order.NameDown:
            Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                return entry2.Name.CompareTo(entry1.Name);
            });
            break;
        case Order.ScoreDown:
            Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                return entry1.Score.CompareTo(entry2.Score);
            });
            break;
        case Order.ScoreUp:
            Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                return entry2.Score.CompareTo(entry1.Score);
            });
            break;
        }

        int i = 0;
        foreach (HighScoreEntry hs in highScoreList)
        {
            i++;
            hsText = hsText + i + ". " + hs.Name + " " + hs.Score + "\n";
        }

    }
    return hsText;
}
```

Kuva: Pisteiden haku ja järjestäminen.

5.5.4 Game Info

Build Info -valikossa pelaaja saa tietoa buildin nimestä ja tekijästä sekä näkee ajankohdan milloin buildi on käännetty. Graafisten elementtien lisäksi tässä paneelissa olennainen asia on tietojen asettaminen ja näyttäminen Label-objektissa. Tiedot asetetaan IGUICode-tiedostossa.

Aluksi generoidaan Init-metodi IGUICode-skriptiin. Label-objektia klikataan Game-näkymässä hiiren oikealla napilla ja valitaan Init.

Kuva: Game Info -paneeli.

Generoituun metodiin annetaan halutut tiedot ja asetetaan labelin text-arvoksi merkkijono.

```
public void label_buildinfo_Init(iGUILabel caller)
{
    string author = "Niko Sarkkinen";
    string productName = "Slaughter";
    string buildTime = "19.9.2014";

    label_buildinfo.label.text = productName + "\n" + author + "\n" + buildTime;
}
```

Kuva: Tiedot asetetaan Label-elementtiin koodissa.

5.5.5 Quit

Quit-painikkeesta avautuu ikkuna, jossa pelaajalta varmistetaan haluaako hän poistua pelistä. Paneelin avautuminen eroaa hieman muiden paneelien avautumisesta, koska Main Menu -paneelia ei oteta kokonaan pois näkyvistä vaan se asetetaan epäaktiiviseksi. Main Menu -paneeli saa jäädä näkyviin taustalle, koska se on pienempi kuin varmistusikkuna.

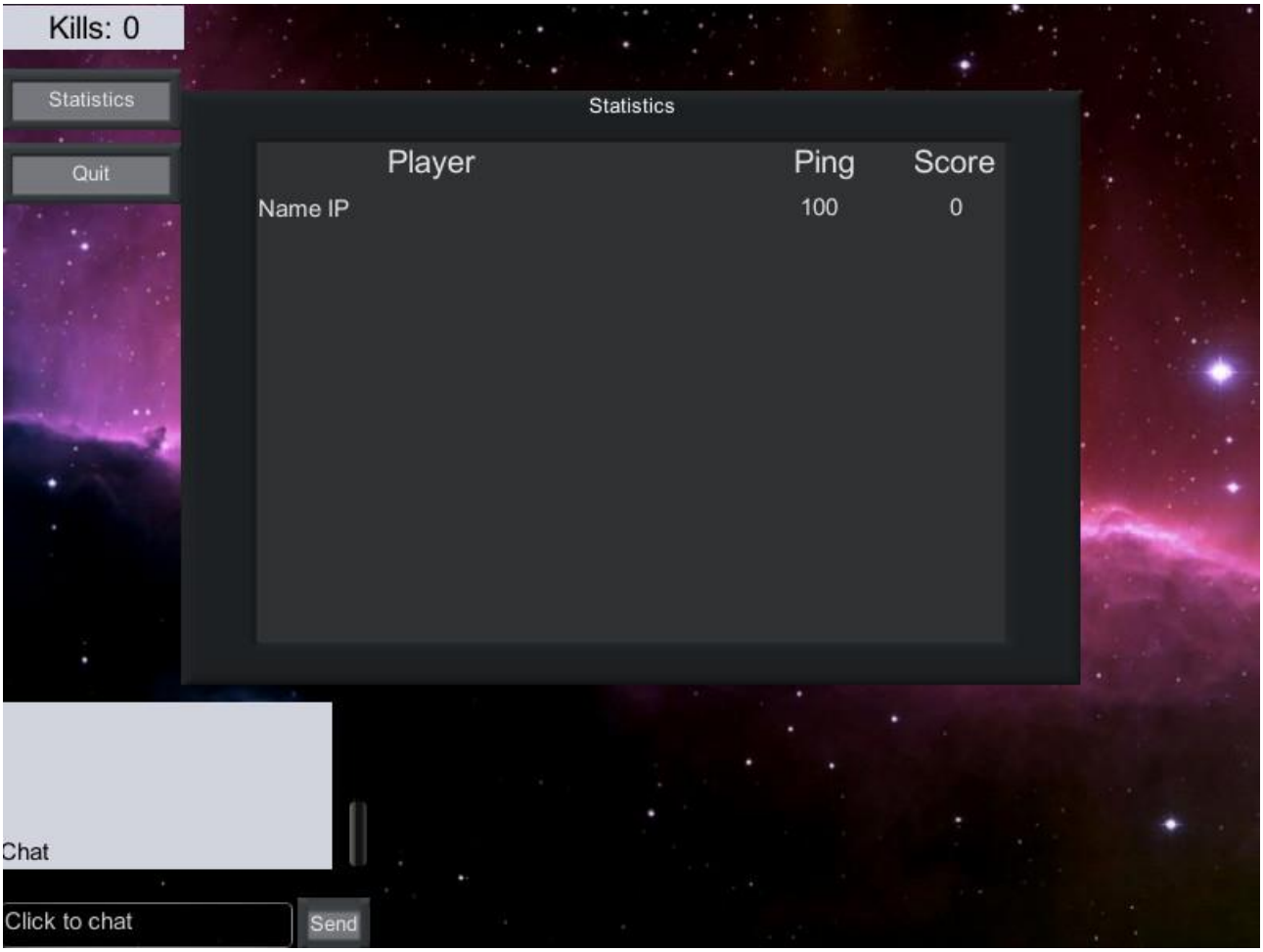
Kuva: Varmistusikkuna pelistä poistumiseen.

```
public void btn_quit_Click(iGUIButton caller)
{
    mainmenu_panel.passive = true;
    panel_quitconfirm.enabled = true;
}

public void btn_yes_Click(iGUIButton caller)
{
    Application.Quit();
}

public void btn_no_Click(iGUIButton caller)
{
    panel_quitconfirm.enabled = false;
    mainmenu_panel.passive = false;
}
```

Kuva: Varmistusikkunan painikkeiden koodi.

5.6 Pelinäkymä

Pelinäkymässä on näkyvissä erilaista tietoa pelin tapahtumista, chat ikkuna sekä painike pelistä poistumiseen.



Kuva: Pelinäkymä.

5.6.1 Kill Score

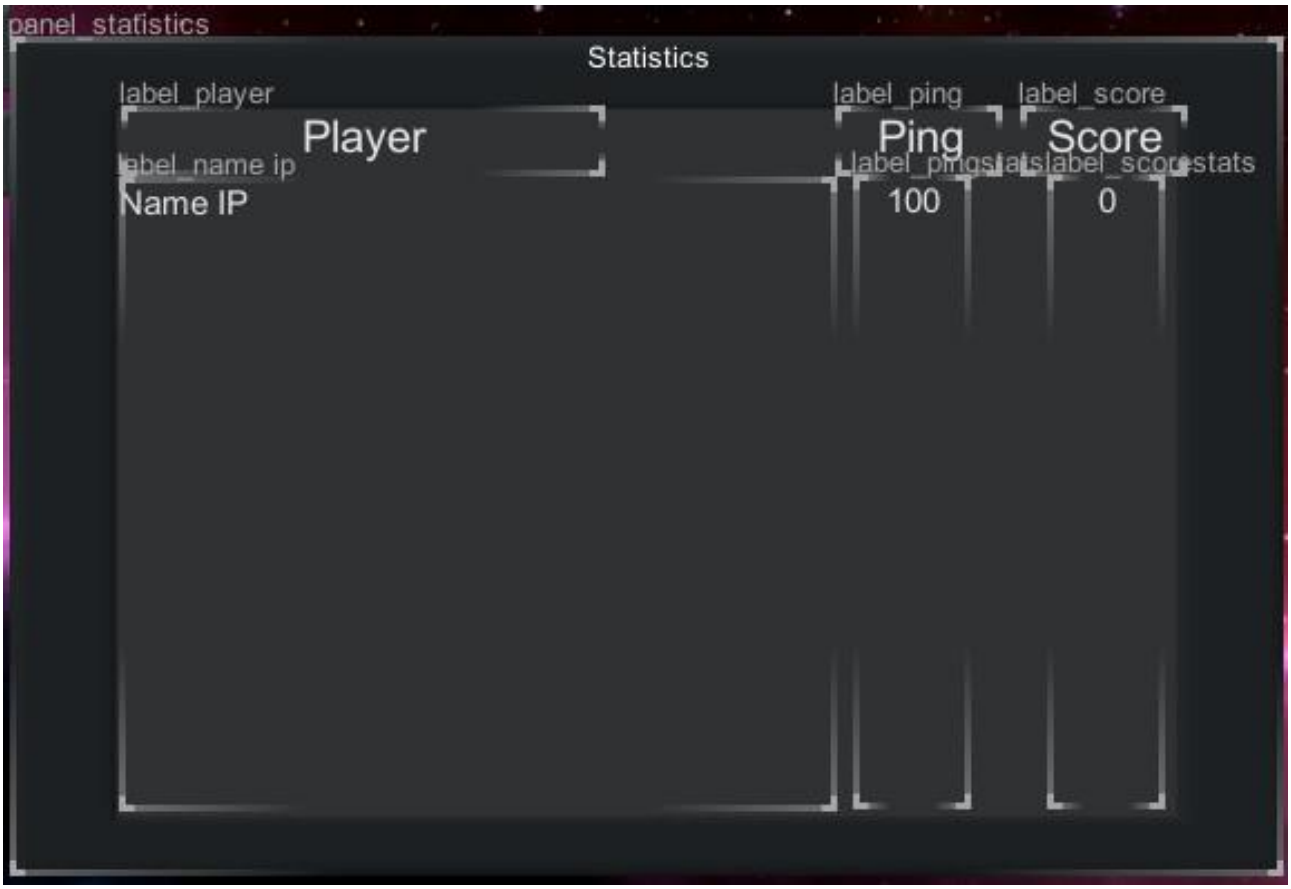
Kill Score on hyvin yksinkertainen GUI-elementti, joka on koko ajan näkyvissä peliä pelatessa. Toolboxista luodaan Label-elementti. Sille täytyy säätää ulkonäköasetukset halutunlaisiksi. Koodissa tarvitsee päästä käsiksi Label-elementin value-arvoon, ja asettaa siihen oikea pistemäärä.

```
// your own dog won't bite you.
if ( other.GetComponent<ShipController>().ownerId != ownerId )
{
    // Blow up

    GameObject.Find("NetworkGUI").GetComponent<BattleScript>().KillPlayer(other.gameObject.networkView.viewID);
    GameObject.Find("NetworkGUI").GetComponent<GUIScript>().numberOfKills++;
    GameObject.Find("iGUI Root Battle").GetComponent<iGUICode_01_BattleScene>().label_score.label.text = "Kills: " +
        GameObject.Find("NetworkGUI").GetComponent<GUIScript>().numberOfKills;
    //  might disappear faster
}
```

Kuva: Kill Score -elementtiin asetetaan arvo koodissa.

5.6.2 Statistics-ikkuna

Pelin tilasto-paneelistä pääsee katsomaan pelaajiin liittyvää tietoa kuten ip-osoite, ping-vasteaika ja pisteet. Paneeliin tehdään Label-elementit tilastojen otsikoille. Tilastot tehdään myös Label-elementteihin. Label-elementit on jaettu kolmeen sarakkeeseen helpomman sijoittelun vuoksi.



Kuva: Statistics-paneelin rakenne.

Pelistatistiikkapaneeli avautuu Statistics napista. Sille luodaan Click-metodi, missä tehdään sekä panelin avaaminen että sulkeminen. Kun paneeli avataan, parsetetaan vaaditut tiedot paneelin Label-elementteihin.

```
public void btn_stats_Click(iGUIButton caller)
{
    panel_statistics.enabled = !panel_statistics.enabled;

    if (panel_statistics.enabled)
    {
        GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
        ConnectedPlayerStats[] connected = guiScript.GetPlayerStats();

        string nameiptext = "";
        string pingtext = "";
        string scoretext= "";

        foreach (ConnectedPlayerStats player in connected)
        {
            nameiptext = nameiptext + player.name + " " + player.address + "\n";
            pingtext = pingtext + player.ping.ToString() + "\n";
            scoretext = scoretext + player.score.ToString() + "\n";
        }

        label_name_ip.label.text = nameiptext;
        label_pingstats.label.text = pingtext;
        label_scorestats.label.text = scoretext;
    }
}
```

Kuva: Statistiikat haetaan ja asetetaan oikeisiin Label-elementteihin.

GetPlayerStats-funktio palauttaa tarvittavat tiedot structissa.

```
public struct ConnectedPlayerStats
{
    public string name;
    public string address;
    public int ping;
    public int score;
}
```

Kuva: Structi tietojen tallentamiseen.

```
public ConnectedPlayerStats[] GetPlayerStats()
{
    ConnectedPlayerStats[] players = new ConnectedPlayerStats[Network.connections.Length+1];

    players[0].name = "Server";
    players[0].address = ipAddress;
    players[0].ping = 0;

    for (int i = 1; i <= players.Length; i++)
    {
        players[i].name = "Player " + i;
        players[i].address = Network.connections[i].ipAddress;
        players[i].ping = Network.GetAveragePing(Network.connections[i]);
    }

    // Score
    for(int i = 0; i < score.Length; i++)
    {
        for(int j = 0; j < score.Length; j++)
        {
            if (int.Parse(Network.connections[i].guid) == score[j].id)
            {
                players[i].score = score[j].id;
            }
        }
    }

    return players;
}
```

Kuva: Tiedot haetaan ja palautetaan GetPlayerStats-funktiossa.

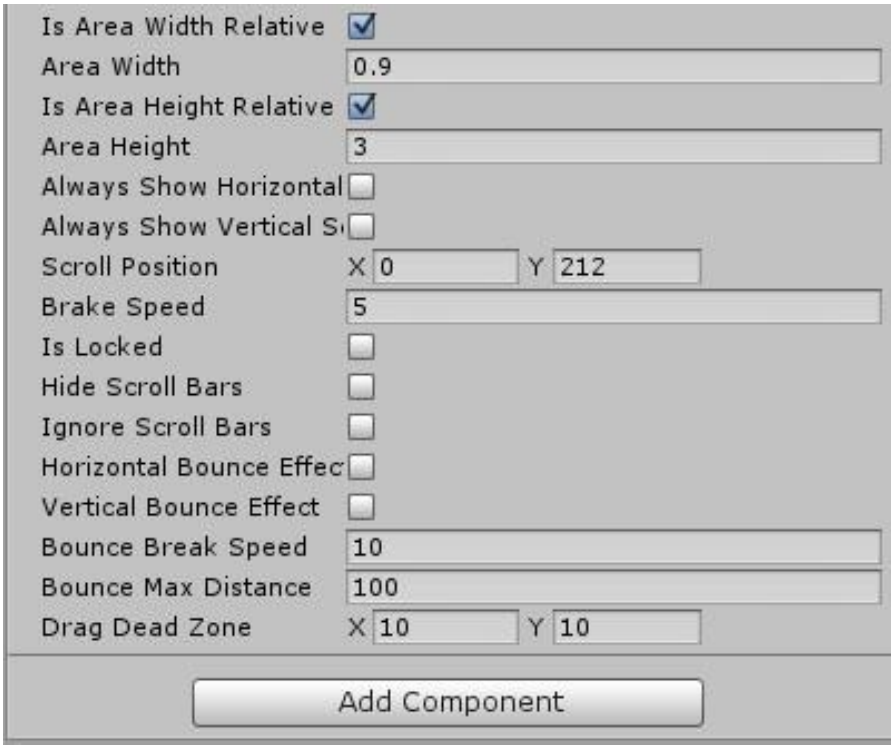
5.6.3 Chat-ikkuna

Chat-ikkuna toteutetaan asettamalla MessageQueuen sisältö Label-elementtiin. Vieritettävä elementti tehdään Scroll View -elementillä. Tekstinsyöttö-elementtiin kirjoitettu tekstijono lisätään MessageQueueen joko Send-painikkeella tai Enter-nappia painamalla.

Kuva: Chat-ikkunan rakenne.

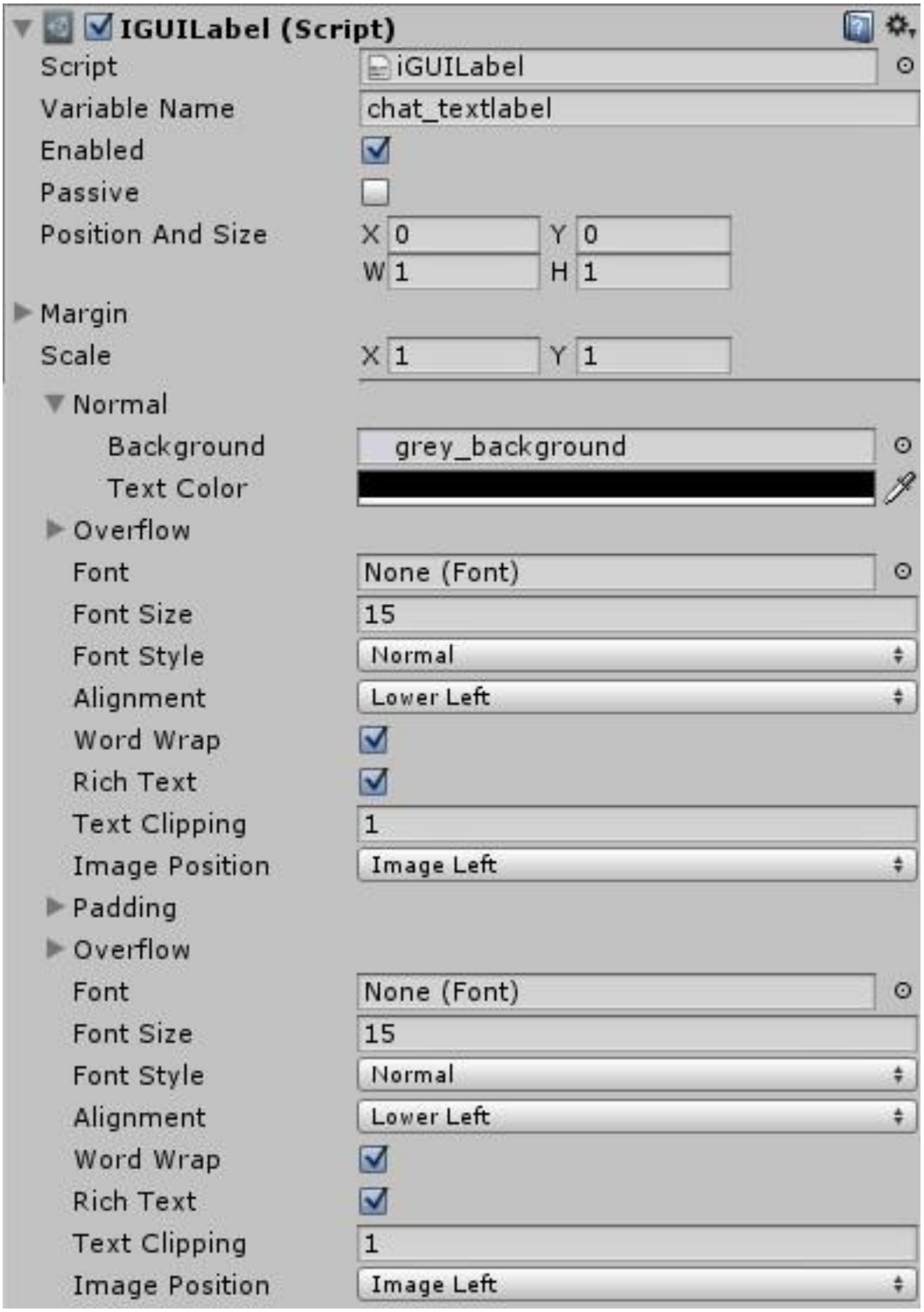
5.6.3.1 Viestialue

Chat-alueen teko aloitetaan luomalla Container-elementti kokonaisuuden helpompaa hallitsemista ja siirtämistä varten. Viestialue toteutetaan Scroll View -elementillä, jotta ylimenevää tekstiä pystyy selaamaan. Asetuksista säädetään viestialueen koko ja vierityspalkin ominaisuudet.



Kuva: Viestialueen asetuksia.

Scroll View -elementin alle luodaan Label-elementti. Kun sen sijainniksi asettaa (0,0) ja kooksi (1,1), asettuu se juuri Scroll Viewin asetuksissa määritetyn koon mukaiseksi. Tekstin Alignment-arvoksi asetetaan Lower Left, jotta saavutetaan chateille tyypillinen tekstin vierimissuunta. Elementille voi asettaa taustakuvan, jotta teksti erottuu helpommin.



Kuva: Label-elementin asetuksia.

Koodissa Label-elementin text-muuttujan arvoa muokataan MessageQueuessa. Uuden viestin käsittelyn yhteydessä parsetetaan MessageQueuen sisältö merkkijonoksi ja asetetaan se text-muuttujaan.

```
public new void Enqueue(string msg)
{
    if ( Count > m_MaxMessagesInQueue ) Dequeue();
    base.Enqueue(msg);

    GameObject guiRoot = GameObject.Find("iGUI Root Battle");
    if (guiRoot != null)
    {
        guiRoot.GetComponent<iGUICode_01_BattleScene>().chat_textlabel.label.text = this.ToString();
    }
}

public string ToString()
{
    string msg = "";
    foreach(string s in this.ToArray())
    {
        msg += "\n" + s;
    }
    return msg;
}
```

Kuva: MessageQueue asettaa sisältämänsä tekstidatan Label-elementtiin.

5.6.3.2 Syöttökenttä

Tekstinsyöttö toteutetaan kahdella elementillä. Ensimmäinen on Text Field -elementti, johon pelaaja voi kirjoittaa tekstiä. Toinen on Button-elementti, jota painamalla teksti lähetetään MessageQueuelle. Tekstin lähetys toteutetaan tapahtuvaksi myös Enter-nappia painamalla.

Kumpaankaan elementtiin ei tarvitse säätää ulkonäön lisäksi mitään erityistä. Generoidaan Text Field-elementille Focus- ja EnterKey-metodit. Focus-metodi poistaa tekstikentästä vakiotekstin. EnterKey-metodi lähettää viestin Chat-metodille ja sen jälkeen asettaa tekstikentän arvoksi vakiotekstin. Button-elementille luodaan Click-metodi ja laitetaan se tekemään sama, mitä EnterKey-metodi tekee.

```
public void chat_input_Focus(iGUITextfield caller)
{
    chat_input.value = "";
}

public void chat_input_EnterKey(iGUITextfield caller)
{
    GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
    guiScript.ChatMessage(chat_input.value);
    chat_input.value = "Click to chat";
}

public void chat_send_Click(iGUIButton caller)
{
    GUIScript guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
    guiScript.ChatMessage(chat_input.value);
    chat_input.value = "Click to chat";
}
```

Kuva: TextField-elementin koodia.

5.6.4 Quit

Quit-painike luodaan valitsemalla Button Toolboxista ja raahaamalla halutun kokoinen painike Game-näkymässä. Painikkeelle generoidaan Click-metodi, jossa vahvistusikkuna avataan.

Kuva: Pelin sulkemisen varmistusikkuna.

```
public void btn_quit_Click(iGUIButton caller)
{
    panel_quitconfirm.enabled = true;
}

public void btn_yes_Click(iGUIButton caller)
{
    Application.Quit();
}

public void btn_no_Click(iGUIButton caller)
{
    panel_quitconfirm.enabled = false;
}
```

Kuva: Varmistusikkunan painikkeiden koodi.

6 Daikon Forge

6.1 Atlas ja tekstuurit

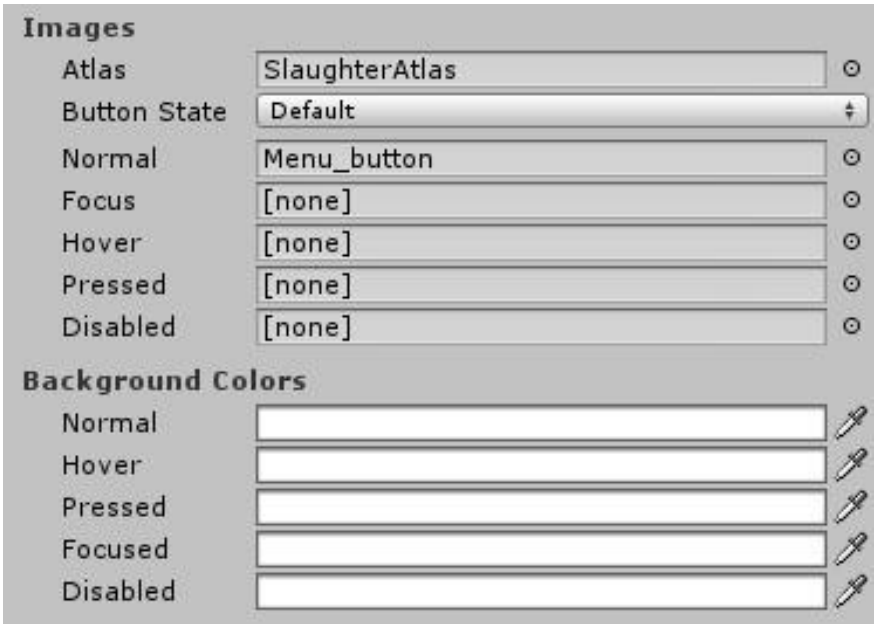
6.1.1 Atlas

Atlas on yksi iso tekstuuri joka sisältää useita eri tekstuureja. Tekstuurien kasaaminen yhdeksi isoksi tekstuuriksi on tehokkaampaa kuin usean pienemmän tekstuurin käyttäminen. Daikonissa ei tule mukana työkalua tekstuureiden pakkaamiseen, vaan se täytyy tehdä jollain kolmannen osapuolen ohjelmalla. Esimerkiksi TexturePacker on ilmainen ohjelmisto, missä on mahdollista pakata tekstuureja Unityn tunnistamassa muodossa.

Unityssä pakattu tekstuuri importataan valitsemalla Tools → Daikon Forge → Texture Atlas → Import from TexturePacker. Avautuvassa ikkunassa asetetaan Texture File -kohtaan pakattu tekstuuri, ja Data File -kohtaan TexturePackerin luoma data-tiedosto. Import-nappia painamalla avautuu tallennusvalikko, missä päätetään mihin luotava prefab tallennetaan. Valitse sijainti projektin Asset kansiosta ja klikkaa OK. Atlas on nyt valmis käytettäväksi.

6.1.2 **Tekstuuri**

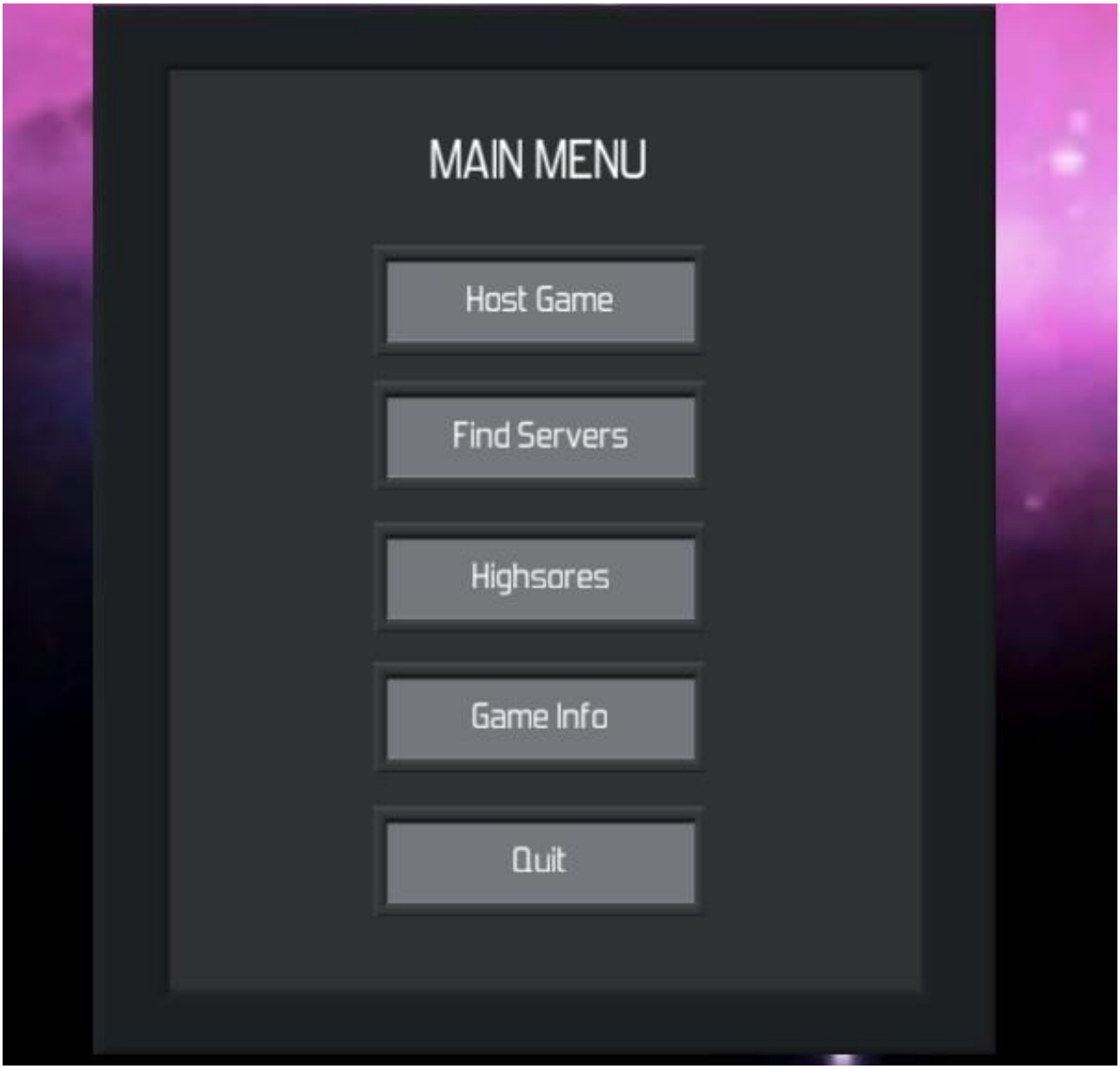
Daikon Forgessa ei ole elementeille varsinaista Sprite-komponenttia. Elementtien tekstuuri asetetaan elementin toimintalogiikan sisältävän skriptin ulkonäköasetuksista.



Kuva: Elementtien ulkonäköasetukset.

6.2 Alkuvalikko

Alkuvalikko tullaan toteuttamaan niin, että pelaajalle avautuu aluksi päävalikko, josta voi toimintapainikkeilla avata uusia valikoita. Valikoista pääsee pois sulkemisnapista.

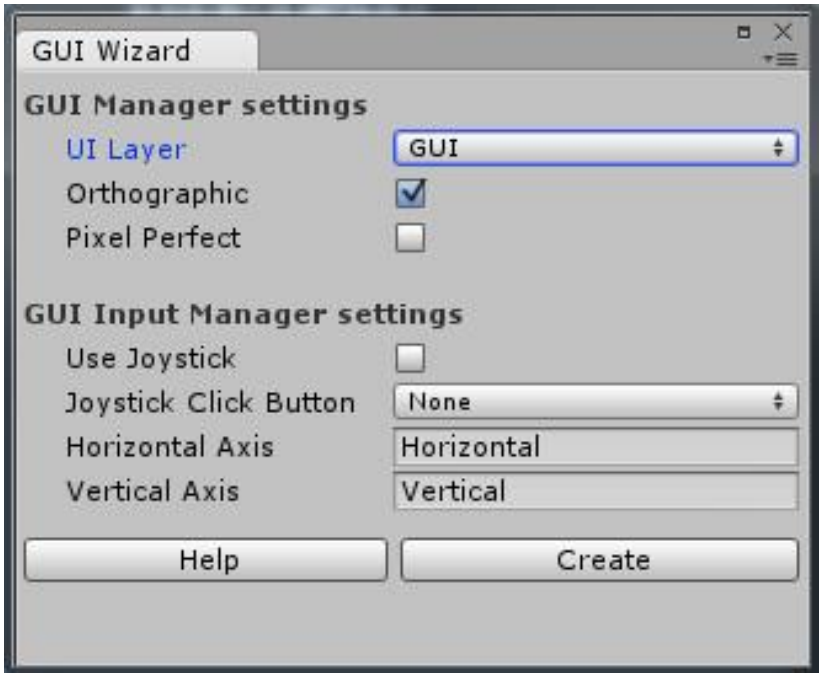


Kuva: Alkuvalikko.

Ennen kun alkuvalikon tekeminen aloitetaan, täytyy käyttöliittymää vasten luoda uusi Layer. Avataan Tags & Layers -valikko klikkaamalla Edit -> Project Settings -> Tags & Layers. Siellä lisätään Layers-listan ensimmäiseen vapaaseen kohtaan GUI-niminen User Layer.

Kuva: Layerin luonti.

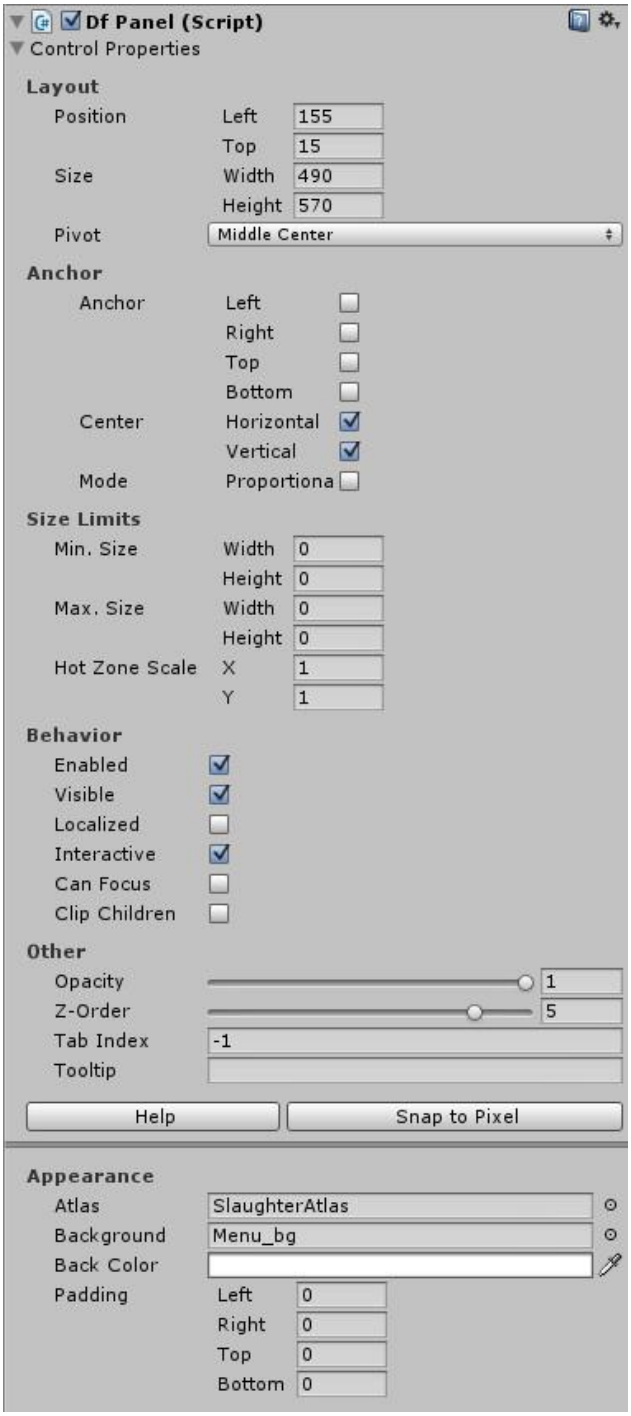
Alkuvalikon luonti aloitetaan klikkaamalla Tools -> Daikon Forge -> UI Wizard. GUI:n asetukset voi säätää tässä, mutta asetuksia voi helposti säätää myöhemminkin. Create-nappia painamalla Daikon Forge luo UI Root -objektin, mistä löytyy DfGUIManager-skripti asetusten muokkaamiseen.



Kuva: GUI Wizard.

6.2.1 Päävalikko

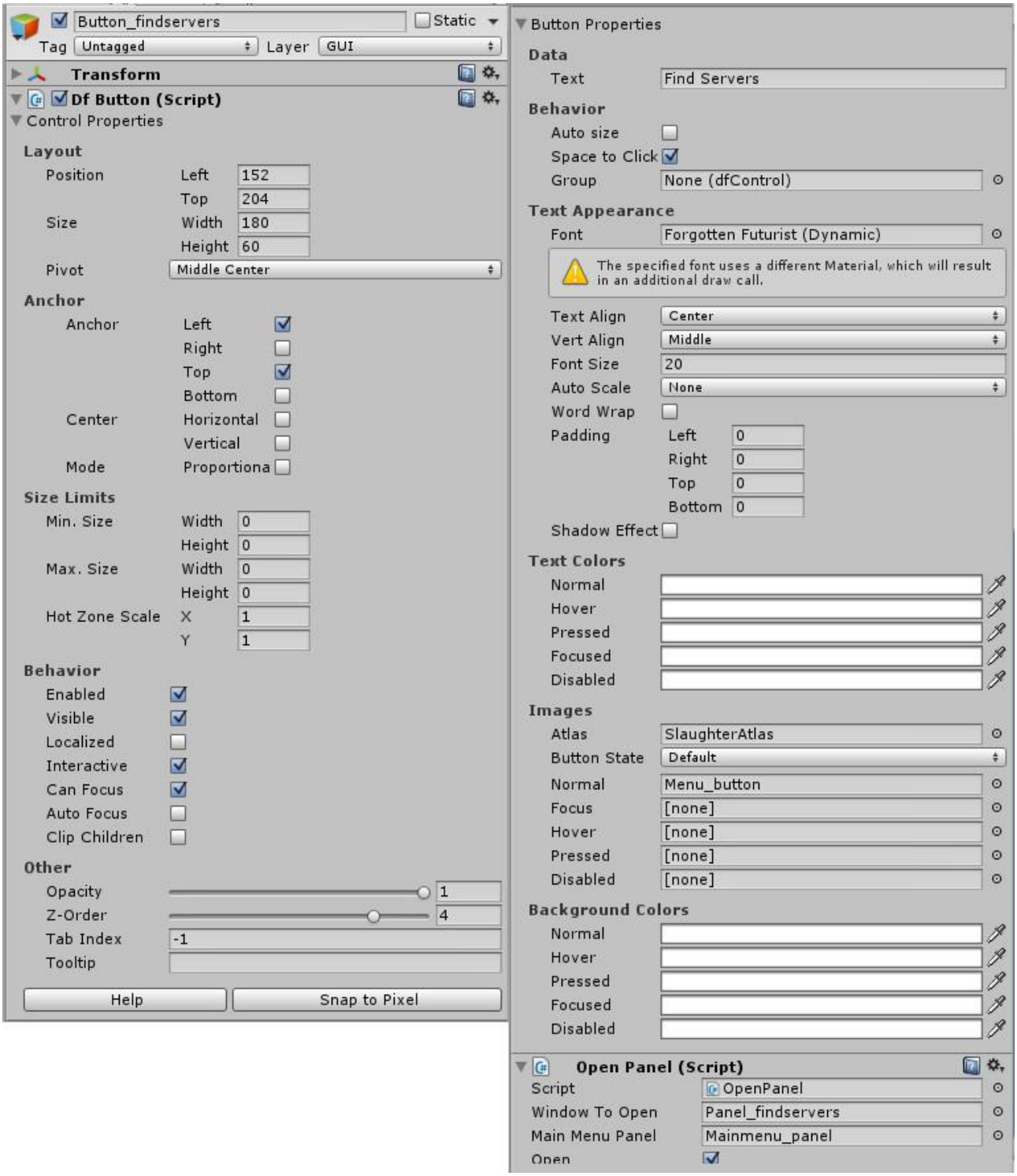
Päävalikon tekeminen aloitetaan luomalla uusi Panel-objekti. Se luodaan valitsemalla UI Root -objekti ja klikkaamalla hiiren oikealla napilla Scene-näkymässä. Avautuvassa valikossa mene Add control -> Containers -> Panel. Paneelin asetuksissa säädetään sen sijainti ja koko sekä mitä tekstuuria se käyttää atlaksesta.



Kuva: Päävalikon asetukset.

6.2.2 Toimintopainikkeet

Alkuvalikon painikkeista avautuu uusi paneeli, mikä sisältää sen tarvitsemat toiminnot. Painikkeet luodaan klikkaamalla Scene-näkymässä hiiren oikealla ja valitsemalla Add Control -> Button.



Kuva: Alkuvalikon painikkeiden asetuksia.

Painikkeeseen lisätään itse tehty skripti, jossa eri paneeleiden avaaminen ja sulkeminen toteutetaan.

```
public class OpenPanel : MonoBehaviour {

    public GameObject windowToOpen;
    public GameObject mainMenuPanel;
    public bool open;

    void OnClick()
    {
        if (windowToOpen != null && mainMenuPanel != null)
        {
            if(open)
            {
                windowToOpen.GetComponent<dfPanel>().IsEnabled = true;
                windowToOpen.GetComponent<dfPanel>().IsVisible = true;

                mainMenuPanel.GetComponent<dfPanel>().IsEnabled = false;
                mainMenuPanel.GetComponent<dfPanel>().IsVisible = false;
            }
            else
            {
                windowToOpen.GetComponent<dfPanel>().IsEnabled = false;
                windowToOpen.GetComponent<dfPanel>().IsVisible = false;

                mainMenuPanel.GetComponent<dfPanel>().IsEnabled = true;
                mainMenuPanel.GetComponent<dfPanel>().IsVisible = true;
            }
        }
    }
}
```

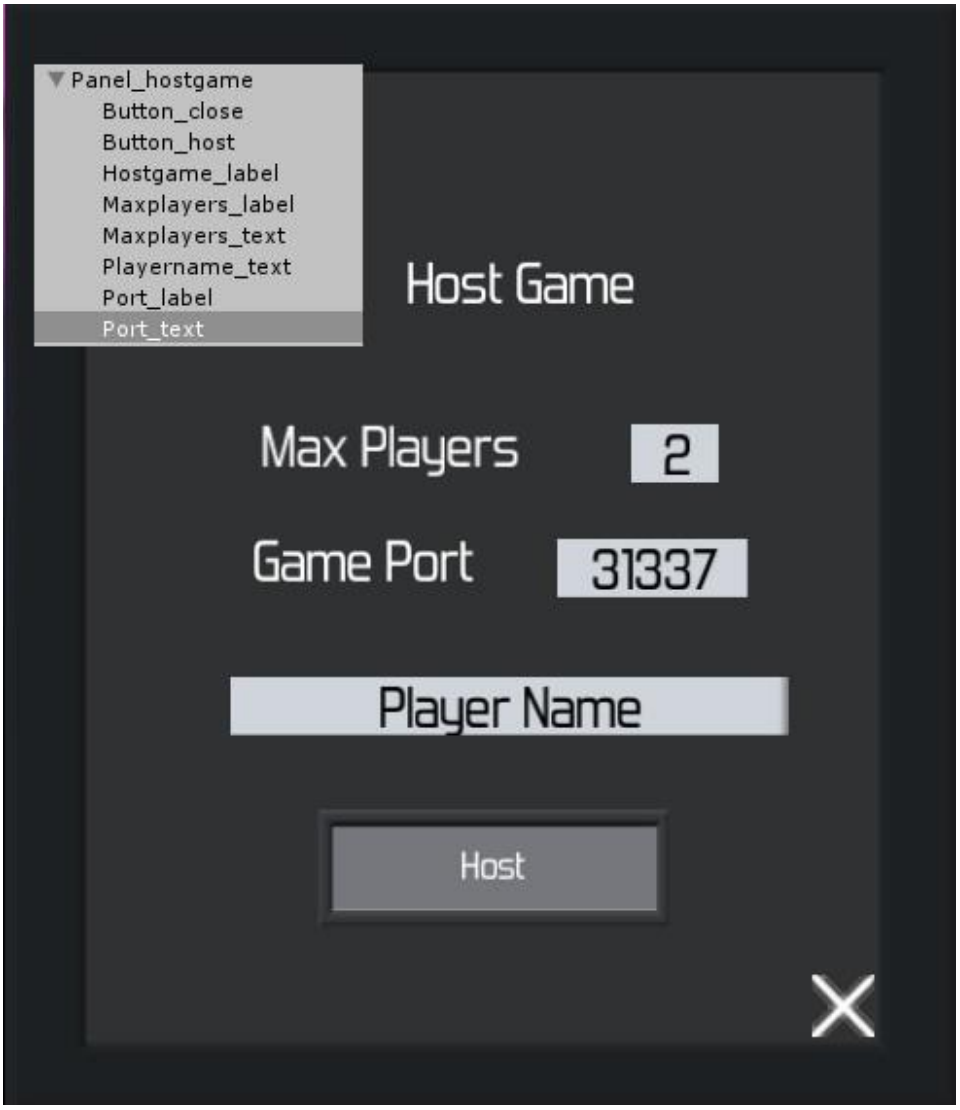
Kuva: Paneeleiden avaamisen ja sulkemiset toteuttava koodi.

6.3 Alavalikot

Jokainen alavalikko on oma DfPanel-objektinsa. Ne luodaan hiiren oikealla napilla avautuvasta valikosta kuten alkuvalikkokin.

6.3.1 Host Game

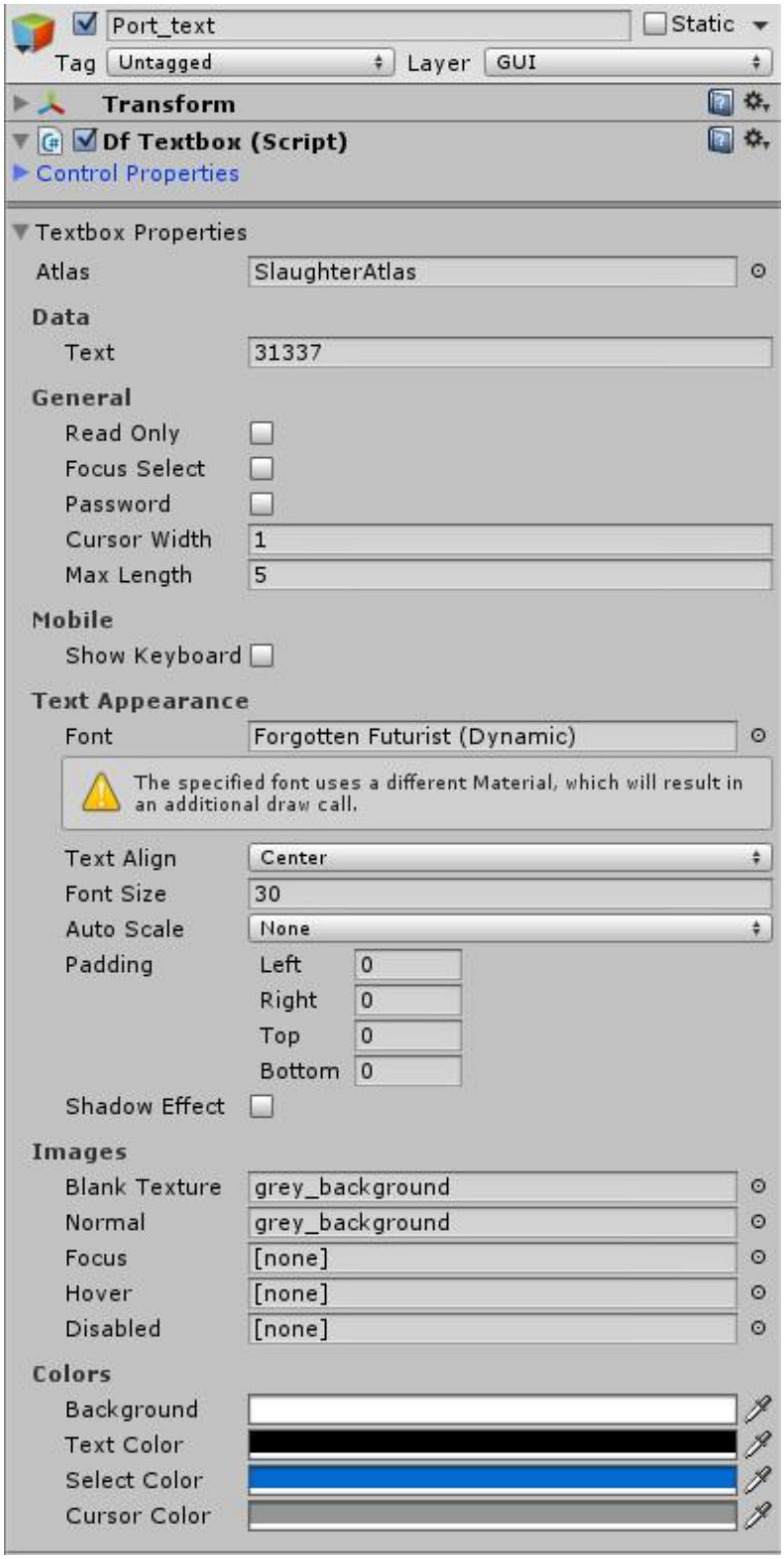
Host Game -paneelin tärkein toiminnallinen asia on pelaajan syöttämien tietojen lukeminen ja uuden pelin aloittaminen näiden tietojen pohjalta. Tekstiotsikot tehdään DfLabel-elementeillä. Niillä ei ole mitään toiminnallisuutta, kunhan laittaa ulkonäköasetukset halutunlaisiksi. Pelaajan syöttämä teksti luetaan DfTextbox-elementillä.



Kuva: Host Game -paneelin rakenne.

Käyttöliittymätutoriaalit

Textbox-skriptiin voidaan asettaa elementissä näkyvä vakioteksti ja pelaajan syöttämien merkkien sallittu enimmäismäärä. Näiden lisäksi skriptistä löytyvät elementin ulkonäköasetukset.



Kuva: Textboxin asetuksia.

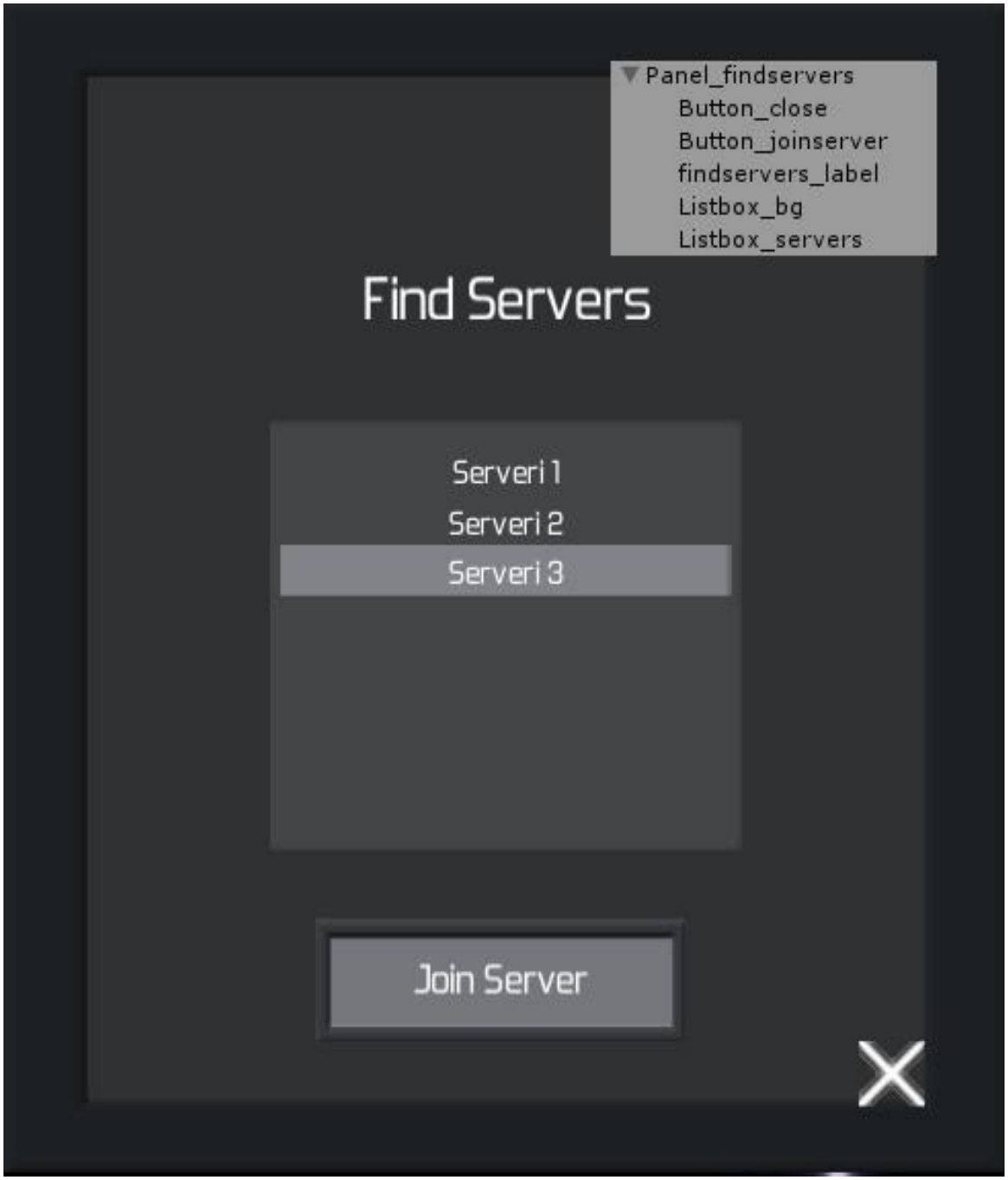
Tekstikenttiä tarvitaan kolme kappaletta pelaajan nimen, palvelimen portin ja pelaajien enimmäismäärän lukemiseksi. Paneeliin luodaan toimintopainike ja siihen lisätään erillinen skripti, jossa painikkeen toiminta määritellään. Painikkeen OnClick-metodi suoritetaan aina kun painiketta painetaan, joten toiminnallisuus kannattaa suorittaa siellä. Samalla kannattaa suorittaa tietojen haku tekstikentästä. Tämä on yksinkertaista tehdä, koska DfTextboxissa on julkinen Text-parametri, josta arvot on helppo poimia.

```
void hostServer()
{
    // Getting the values from input fields
    int maxPlayers = Int32.Parse(transform.parent.Find("Maxplayers_text").GetComponent<dfTextbox>().Text);
    int port = Int32.Parse(transform.parent.Find("Port_text").GetComponent<dfTextbox>().Text);
    string playerName = transform.parent.Find("Playername_text").GetComponent<dfTextbox>().Text;

    // Checking if values are withing accepted limits
    if(playerName.Equals("")) {playerName = "DefaultPlayer";}
    if(maxPlayers > 16) {maxPlayers = 16;}
    if(maxPlayers < 2)  {maxPlayers = 2;}

    guiScript.LaunchServer(maxPlayers, port, playerName);
}
```

Kuva: Koodia serverille liittymiseen.

6.3.2 Find Servers

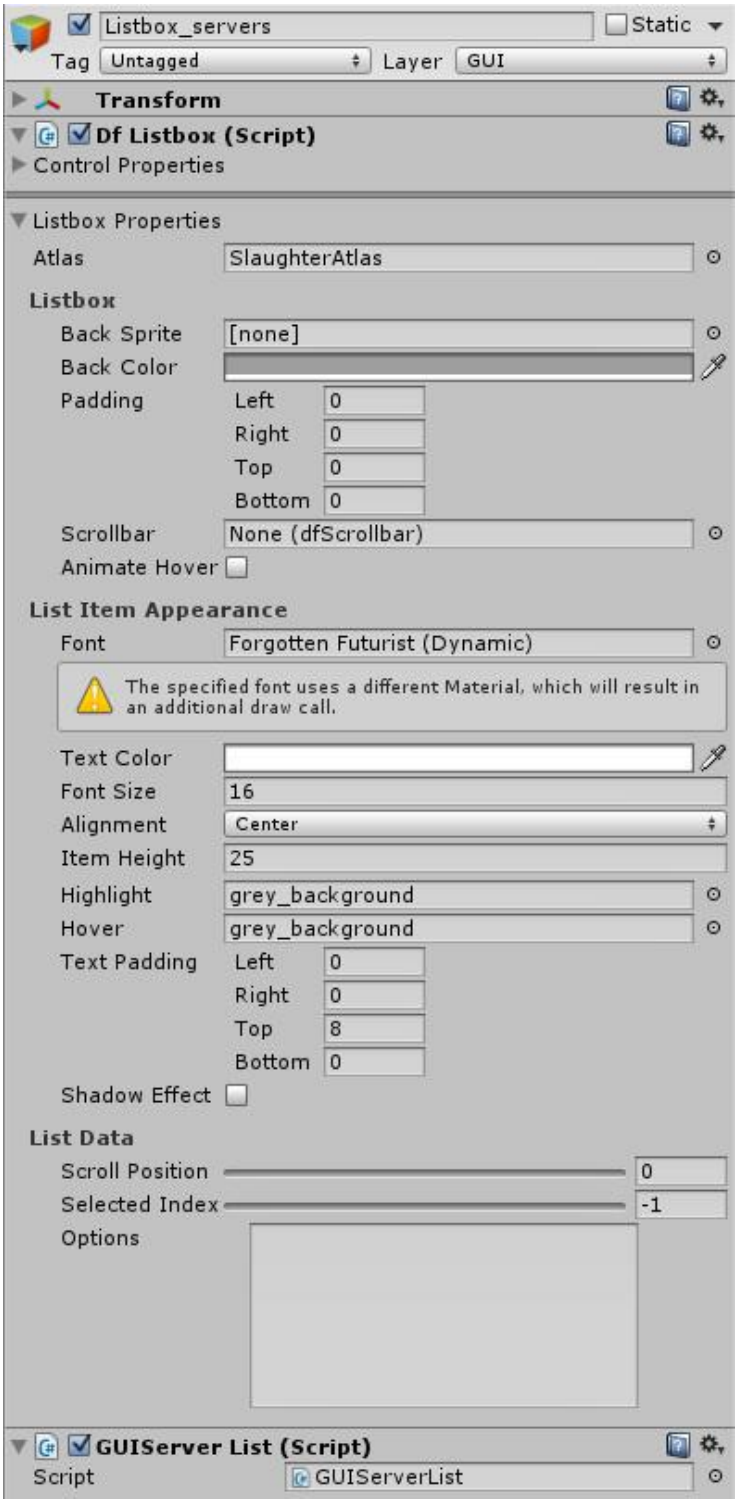
Find Servers -paneelissa listataan kaikki serverit, ja annetaan pelaajalle mahdollisuus liittyä valittuun peliin. Serverilistan näyttäminen toteutetaan DfListbox-skriptillä.



Kuva: Find Servers -paneelin rakenne.

Käyttöliittymätutoriaalit

Serverit listataan Listbox-elementtiin. Se luodaan klikkaamalla Scene-näkymässä hiiren oikealla ja valitsemalla Add Control -> Listbox. Listan ulkonäköön löytyy luonnollisesti paljon säätöjä. Highlight ja Hover -kohtiin voi laittaa tekstuurin, jotta valittu serveri on helpompi nähdä. Selected Index arvoa tullaan koodissa hyädyntämään myöhemmin. Skripti hoitaa automaattisesti elementin rullausominaisuuden, jos serverit menevät yli varatun alueen. Tässä elementissä on kiinni skripti, joka hakee serverilistan ja asettaa sen Options kohtaan.



Kuva: Listbox-elementin asetuksia.

GUIServerList-skriptissä haetaan serverilista ja lisätään jokainen serveri Listboxin valitoihin.

```
public class GUIServerList : MonoBehaviour {

    GUIScript guiscript;
    public ServerEntry[] srvList;

    IEnumerator Start()
    {
        guiscript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
        yield return StartCoroutine(guiscript.UpdateServerList());
        FillServerList();
    }

    void FillServerList()
    {
        guiscript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
        dfListbox listBox = GetComponent<dfListbox>();

        srvList = guiscript.serverList.ToArray();

        foreach (ServerEntry s in srvList)
        {
            listBox.AddItem(s.ToString());
        }
    }
}
```

Kuva: GUIServerList-skriptin koodia.

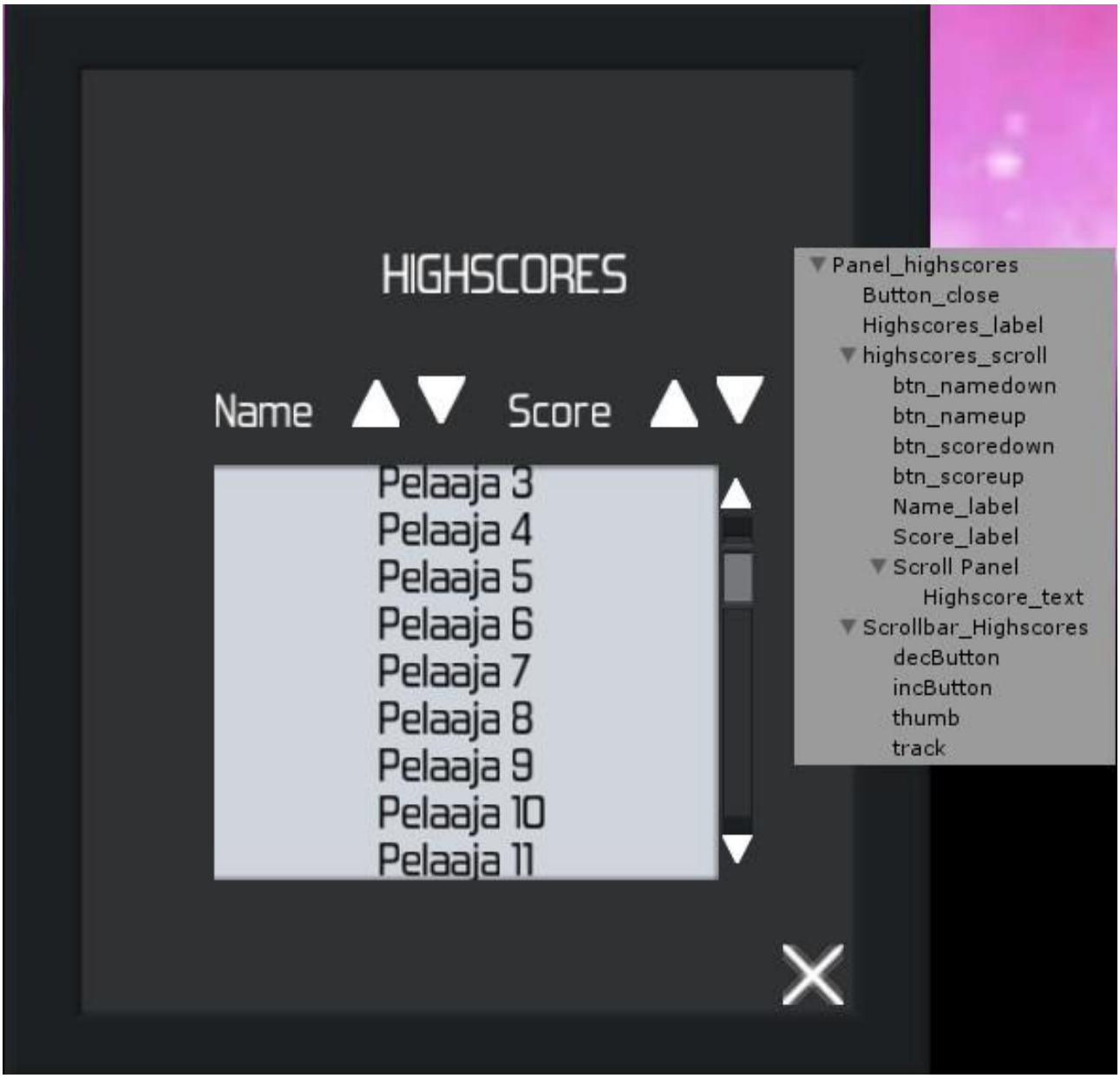
Join Server-painikkeesta ajetaan koodi valitulle serverille liittymiseen.

```
void JoinServer()
{
    dfListbox listBox = GameObject.Find("Listbox_servers").GetComponent<dfListbox>();
    GUIServerList serverlist = GameObject.Find("Listbox_servers").GetComponent<GUIServerList>();

    if (listBox.SelectedIndex != -1)
    {
        NetworkConnectionError err = Network.Connect( serverlist.srvList[listBox.SelectedIndex].address, guiScript.GetGamePort());
        switch( err )
        {
        case NetworkConnectionError.NoError:
            Debug.Log("Waiting for reply...");
            break;
        default:
            Debug.Log("Could not connect: " + err );
            break;
        }
    }
}
```

Kuva: Serverille liittymisen koodia.

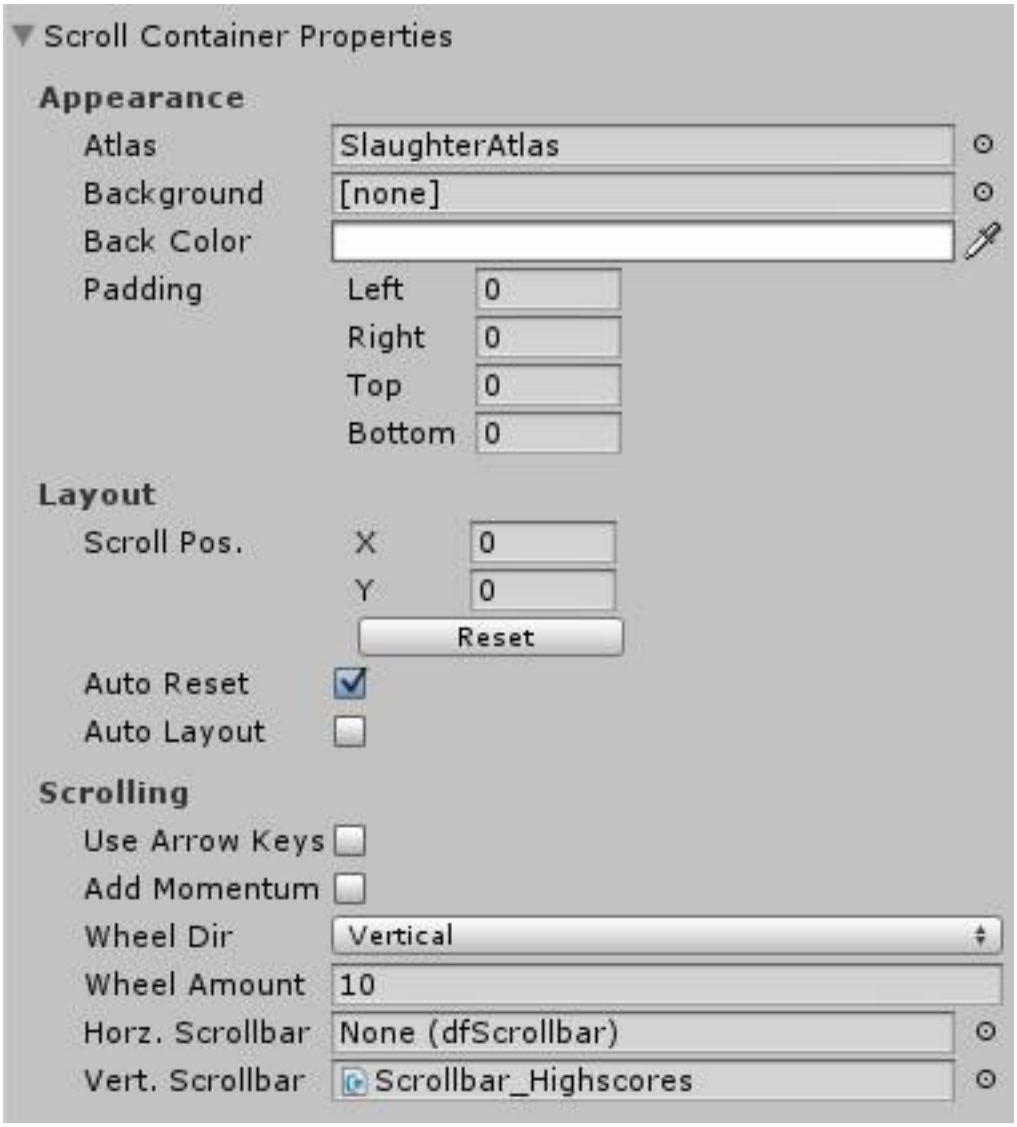
6.3.3 Highscore

Highscore-paneelissa peli hakee serveriltä parhaat pisteet ja laittaa ne näkyville. Pisteitä pystyy järjestämään nimen tai pisteiden mukaan nousevaan tai laskevaan järjestykseen. Pisteiden näyttäminen toteutetaan ScrollPanel-elementillä, jotta pelaaja voi selata varatun alueen ulkopuolelle jääviä pisteitä.



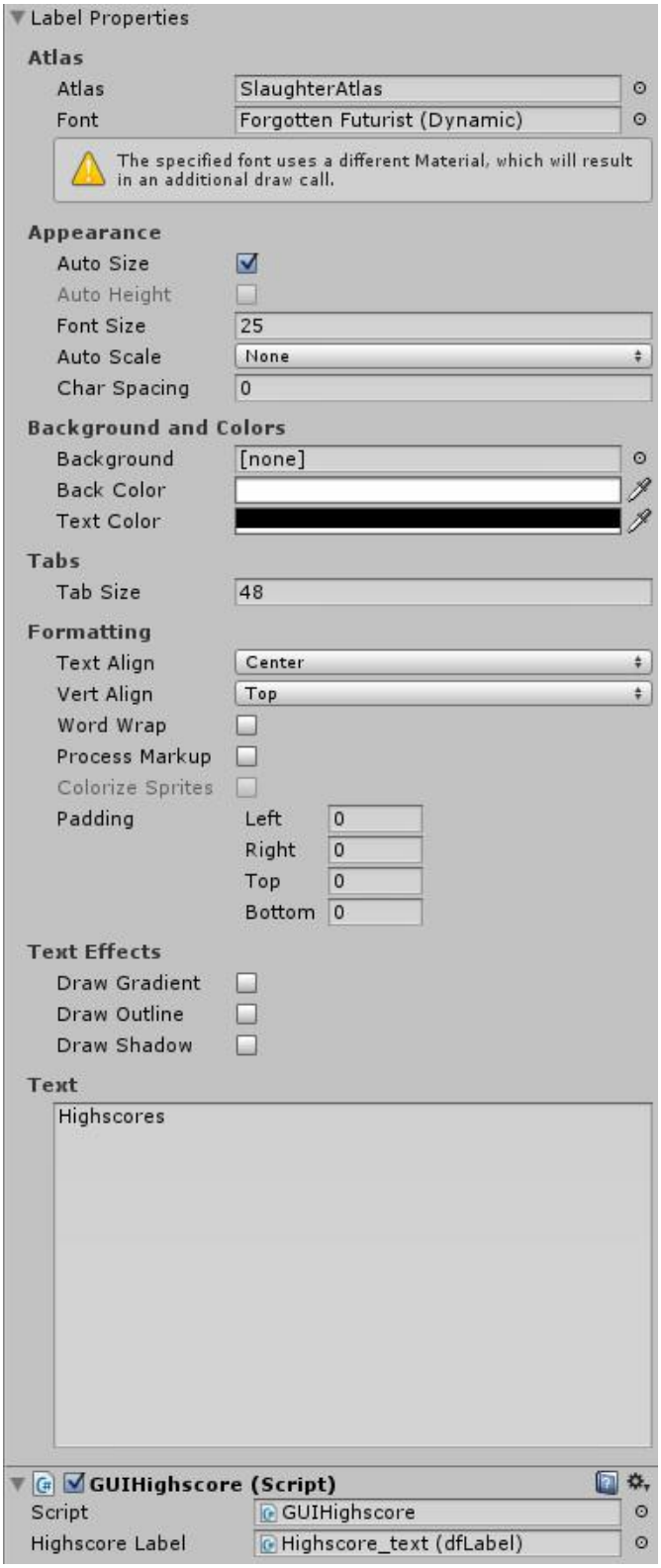
Kuva: Highscore-paneelin rakenne.

Vieritettävä Scroll Panel- elementti toteutetaan DfScrollPanel-skriptillä. Se lisätään klikkaamalla hiiren oikealla napilla Scene-näkymässä ja valitsemalla Add control -> Containers -> Scrollable Panel. Skriptissä säädetään ulkonäköasetukset halutunlaisiksi. Taustakuvaa ei tässä tapauksessa laiteta tähän skriptiin, vaan seuraavassa kohdassa tehtävään Label-objektiin. Vierittämistä varten asetetaan käytettävä vierityspalkki ja sen asetukset.



Kuva: ScrollPanel-elementin asetuksia.

Scroll Panelin alle luodaan Label-objekti. Sen tehtävä on hakea ja näyttää highscore-tiedot. GUIHighscore-skriptissä toteutetaan tietojen haku ja DfLabel-skripti näyttää ne.



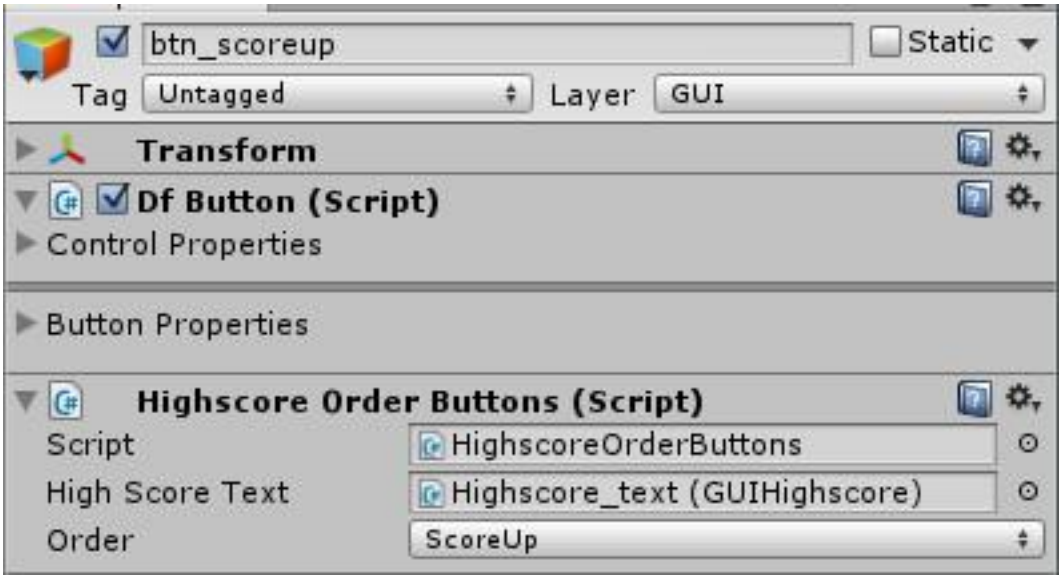
Kuva: Label-elementin asetuksia.

GUIHighscore-skriptissä haetaan highscore-tiedot ja asetetaan ne Label-skriptiin.

```
public class GUIHighscore : MonoBehaviour {

    public dfLabel highscoreLabel;
    private HighScoreEntry[] highScoreList = new HighScoreEntry[2];

    public enum Order
    {
        NameUp,
        NameDown,
        ScoreUp,
        ScoreDown
    }

    void OnEnable ()
    {
        highscoreLabel.Text = getOrderedHighscore(Order.ScoreUp);
    }

    public string getOrderedHighscore(Order order)
    {
        string hsText = "";
        HighScore highScore = new HighScore();
        highScoreList = new HighScoreEntry[2];
        highScoreList[0] = new HighScoreEntry();
        if ( highScore != null )
        {
            highScoreList = highScore.GetHighScores("SlaughterGame",0, -1);
        }

        if ( highScoreList != null )
        {
            switch (order)
            {
            case Order.NameUp:
                Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                    return entry1.Name.CompareTo(entry2.Name);
                });
                break;
            case Order.NameDown:
                Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                    return entry2.Name.CompareTo(entry1.Name);
                });
                break;
            case Order.ScoreDown:
                Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                    return entry1.Score.CompareTo(entry2.Score);
                });
                break;
            case Order.ScoreUp:
                Array.Sort(highScoreList, delegate(HighScoreEntry entry1, HighScoreEntry entry2) {
                    return entry2.Score.CompareTo(entry1.Score);
                });
                break;
            }

            foreach(HighScoreEntry hs in highScoreList )
            {
                hsText = hsText + hs.Name + " " + hs.Score + "\n";
            }
        }
        return hsText;
    }
```

Kuva: GUIHighscore-skriptin koodia.

Pisteet voidaan järjestää eri järjestyksiin pelin ollessa käynnissä. Järjestyspainikkeisiin on lisätty skripti, joka kutsuu yllä olevaa getOrderedHighscore-metodia asetetulla parametrilla.

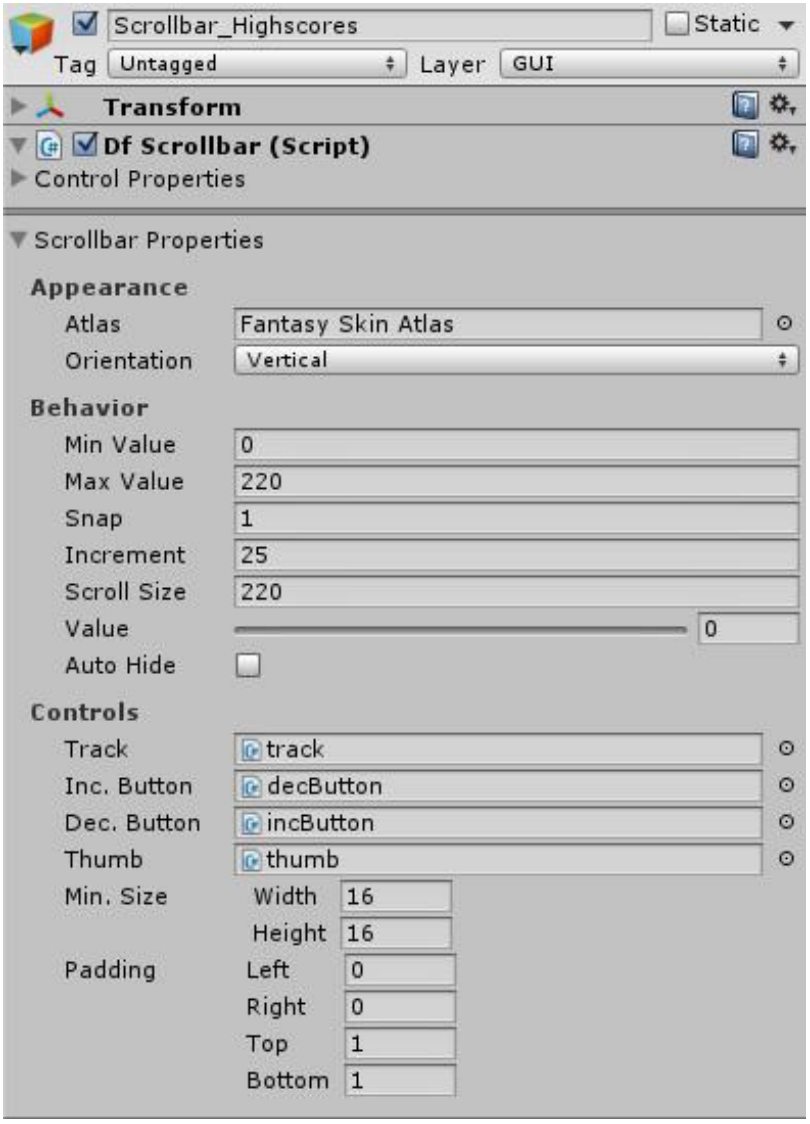


Kuva: Pisteiden järjestystä muuttavan painikkeen rakenne.

```
public class HighscoreOrderButtons : MonoBehaviour {

    public GUIHighscore highScoreText;
    public GUIHighscore.Order order;

    void OnClick()
    {
        GameObject.Find("Highscore_text").GetComponent<dfLabel>().Text = highScoreText.getOrderedHighscore(order);
    }
}
```

Kuva: Painikkeen skriptin koodia.

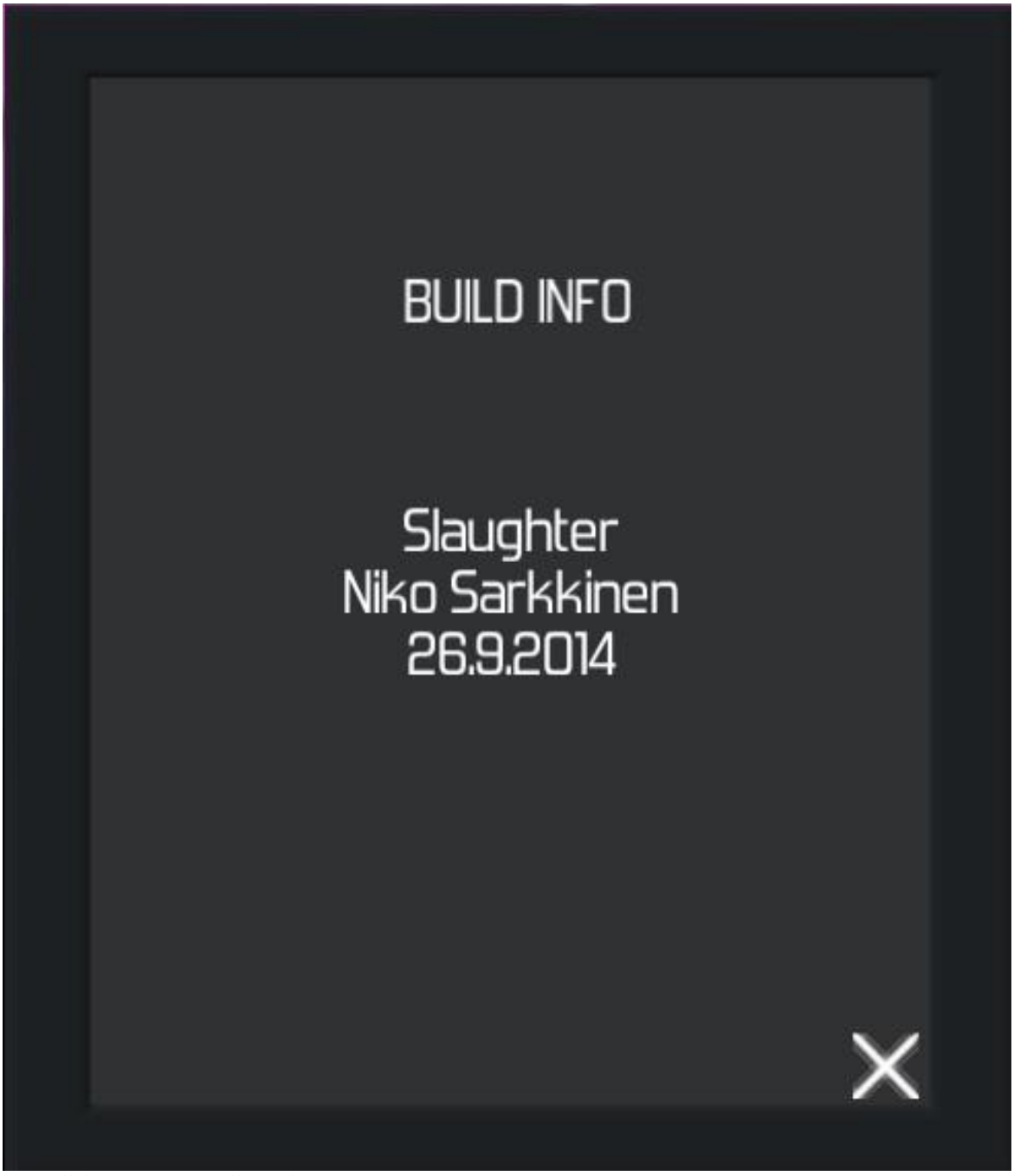
Vierityspalkki koostuu neljästä osasta, joita hallitsee DfScrollbar-skripti. Vierityspalkki luodaan klikkaamalla Scene-näkymässä hiiren oikealla napilla ja valitsemalla Add Control -> Scrollbar. Vierityspalkista tehdään tässä tapauksessa pystysuuntainen, eli Orientation kohtaan asetetaan Vertical. Vierityspalkin graafiset elementit asetetaan Controls-kohtaan. Track ja Thumb -elementit ovat toteutettu DfSlicedSprite-skriptillä. Se eroaa tavallisesta Sprite-skriptistä sillä, että siinä voi näyttää vain osan tekstuurista. Inc. ja Dec. Button -kohdat eivät ole pakollisia, mutta niihin voi lisätä Button-elementit, jos vierityspalkkia halutaan ohjata klikkailemalla.



Kuva: Scrollbar-elementin asetuksia.

6.3.4 Build Info

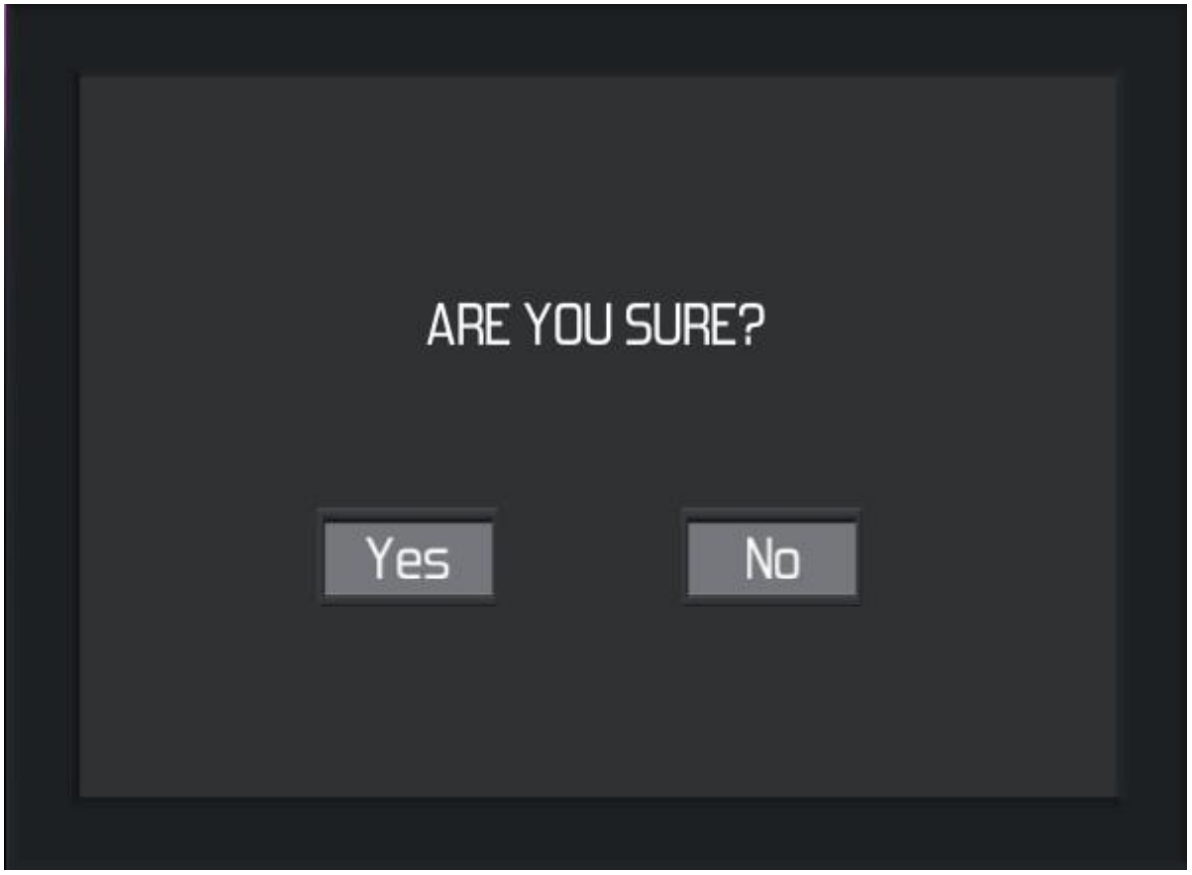
Build Info -valikossa pelaaja saa tietoa buildin nimestä ja tekijästä sekä näkee ajankohdan milloin buildi on käännetty. Tässä paneelissa kaikki näytettävä tieto on muuttumatonta tekstidataa DfLabel-elementeissä, joten omaa koodia ei tarvita laisinkaan.



Kuva: Build Info-paneelin rakenne.

6.3.5 Quit

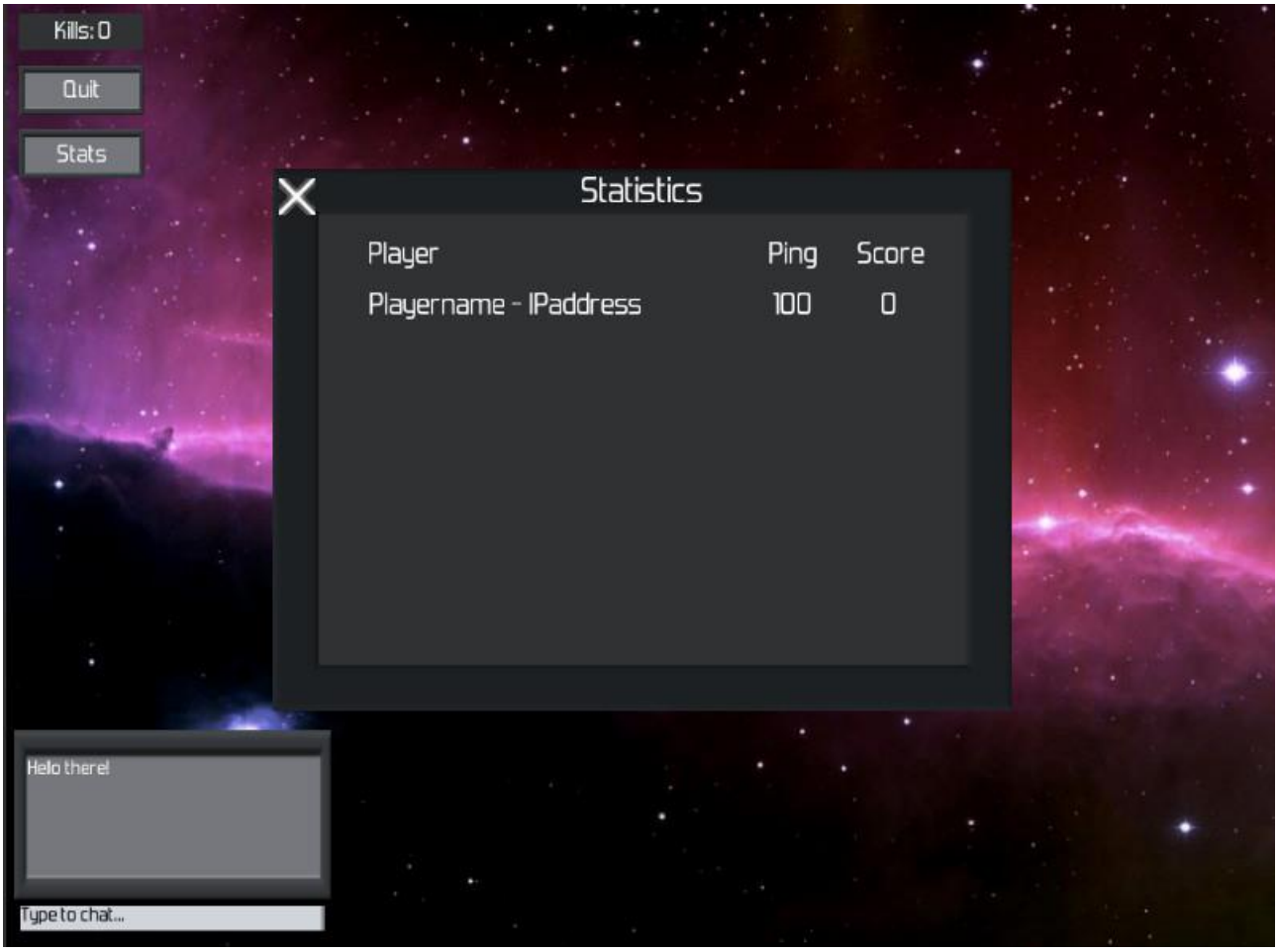
Quit-painikkeesta avautuu ikkuna, jossa pelaajalta varmistetaan haluaako hän poistua pelistä. Yes-painikkeessa on skripti missä kutsutaan Application.Quit-metodia. No-painikkeesta paneeli suljetaan ja palataan aloitusvalikkoon.



Kuva: Pelistä poistumisen varmistusikkuna.

6.4 Pelinäkymä

Pelinäkymässä on näkyvissä erilaista tietoa pelin tilastoista, chat ikkuna sekä painike pelistä poistumiseen.



Kuva: Pelinäkymä.

6.4.1 Kill Score

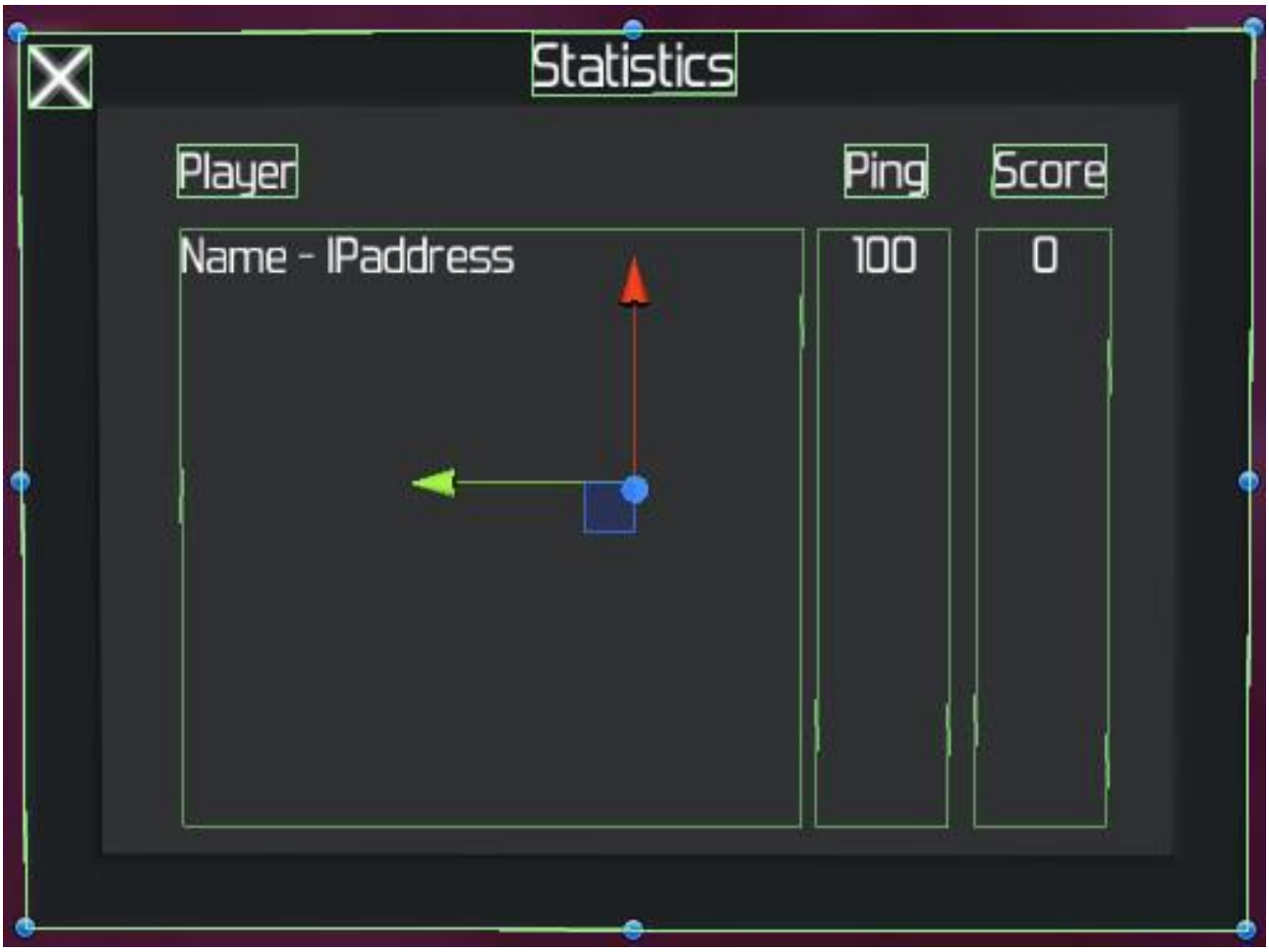
Kill Score on hyvin yksinkertainen GUI-elementti, joka on koko ajan näkyvissä peliä pelatessa. Se toteutetaan DfLabel-elementillä. Koodissa tarvitsee päästä käsiksi Label-elementin value-arvoon, ja asettaa siihen oikea pistemäärä.

```
if ( other.tag == "Ship" )
{
    // your own dog won't bite you.
    if ( other.GetComponent<ShipController>().ownerId != ownerId )
    {
        // Blow up
        GameObject.Find("NetworkGUI").GetComponent<BattleScript>().KillPlayer(other.gameObject.networkView.viewID);
        GameObject.Find("NetworkGUI").GetComponent<GUIScript>().numberOfKills++;
        GameObject.Find("Killscore").GetComponent<dfLabel>().Text = "Kills: " +
            GameObject.Find("NetworkGUI").GetComponent<GUIScript>().numberOfKills;

        //  might disappear faster
    }
}
```

Kuva: Pelin pistetilanteen päivittämisen koodia.

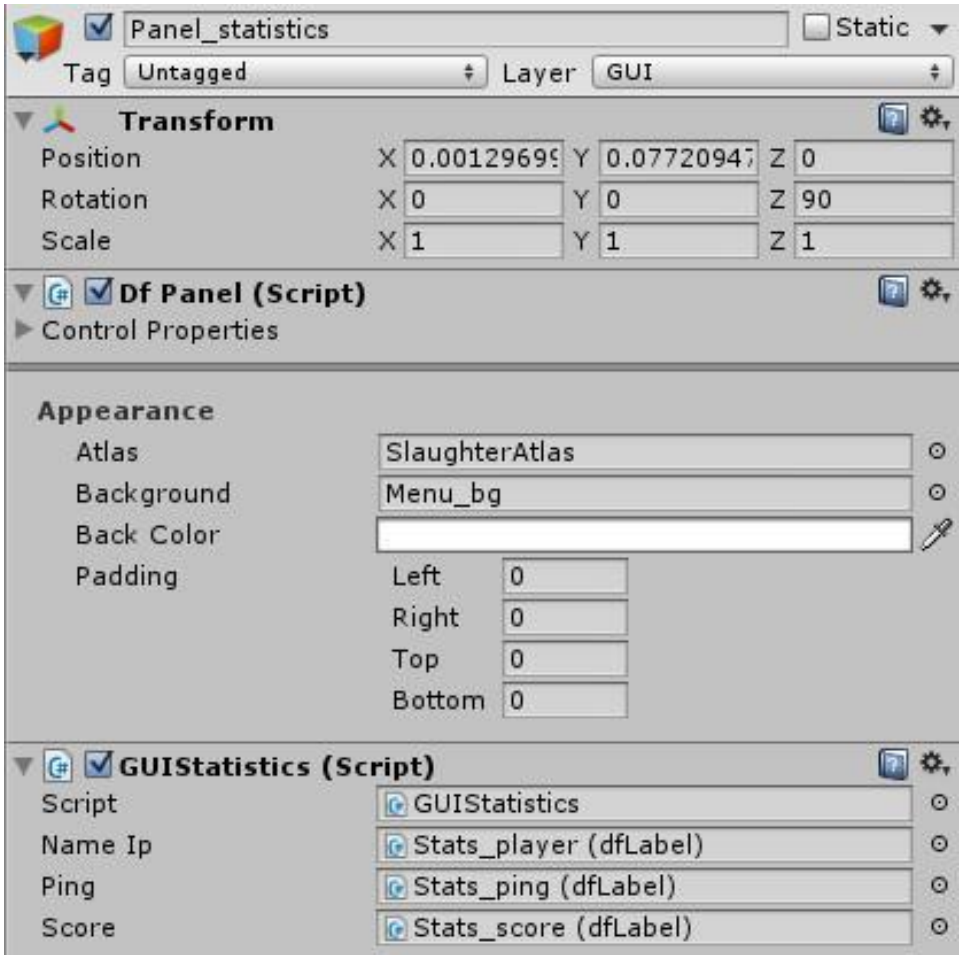
6.4.2 Statistics

Pelin tilasto-paneelistä pääsee katsomaan pelaajiin liittyvää tietoa kuten ip-osoite, ping-vasteaika ja pisteet. Paneeliin tehdään Label-elementit tilastojen otsikoille. Tilastot tehdään myös Label-elementteihin. Label-elementit on jaettu kolmeen sarakkeeseen helpomman sijoittelun vuoksi.



Kuva: Statistics-paneelin rakenne.

Paneeli avautuu Stats-painikkeesta. Siihen liitetään skripti, joka hakee tiedot, kun paneeli aktivoidaan. Skriptissä on julkisina muuttujina paikat tiedot näyttäville Label-objekteille.



Kuva: Statistics-paneelin asetuksia.

Paneelissa kiinni oleva skripti hakee tiedot OnEnable-metodissa.

```
public class GUIStatistics : MonoBehaviour {

    GUIScript guiScript;

    public dfLabel nameIp;
    public dfLabel ping;
    public dfLabel score;

    void OnEnable()
    {

        if (guiScript == null)
        {
            guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
        }

        ConnectedPlayerStats[] connected = guiScript.GetPlayerStats();

        string nameiptext = "";
        string pingtext = "";
        string scoretext= "";

        foreach (ConnectedPlayerStats player in connected)
        {
            nameiptext = nameiptext + player.name + " - " + player.address + "\n";
            pingtext = pingtext + player.ping.ToString() + "\n";
            scoretext = scoretext + player.score.ToString() + "\n";
        }

        nameIp.Text = nameiptext;
        ping.Text = pingtext;
        score.Text = scoretext;
    }
}
```

Kuva: GUIStatistics-skripti hakee ja asettaa tilastot Label-elementteihin.

GUIScriptissä oleva funktio hoitaa tilastojen hakemisen verkon yli ja palauttaa tarvittavat tiedot.

```
public struct ConnectedPlayerStats
{
    public string name;
    public string address;
    public int ping;
    public int score;
}
```

Kuva: Tilastot tallennetaan Structiin.

```
public ConnectedPlayerStats[] GetPlayerStats()
{
    ConnectedPlayerStats[] players = new ConnectedPlayerStats[Network.connections.Length+1];

    players[0].name = "Server";
    players[0].address = ipAddress;
    players[0].ping = 0;

    for (int i = 1; i <= players.Length; i++)
    {
        players[i].name = "Player " + i;
        players[i].address = Network.connections[i].ipAddress;
        players[i].ping = Network.GetAveragePing(Network.connections[i]);
    }

    // Score
    for(int i = 0; i < score.Length; i++)
    {
        for(int j = 0; j < score.Length; j++)
        {
            if (int.Parse(Network.connections[i].guid) == score[j].id)
            {
                players[i].score = score[j].id;
            }
        }
    }

    return players;
}
```

Kuva: GUIScript-skripti hakee ja palauttaa tiedot.

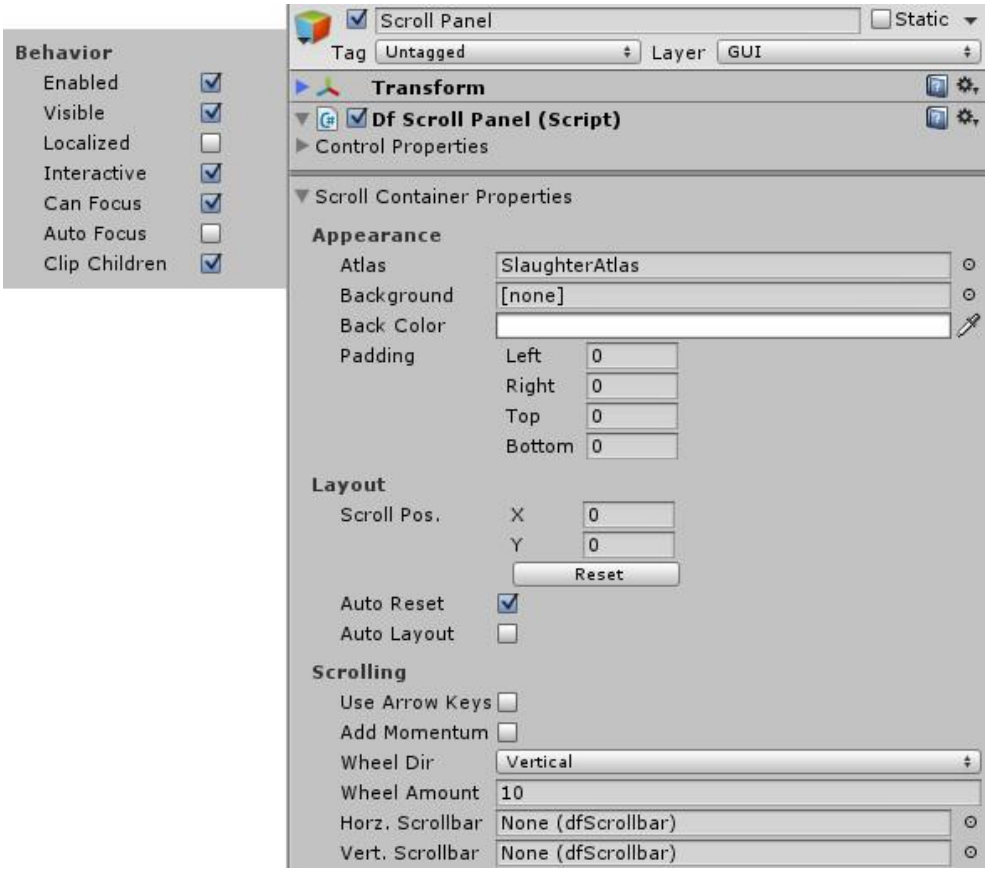
6.4.3 Chat-ikkuna

Chat-ikkuna toteutetaan asettamalla MessageQueuen sisältö Label-elementtiin. Vieritettävä elementti tehdään DfScrollPanel-elementillä. Tekstinsyöttö-elementtiin kirjoitettu tekstijono lisätään MessageQueueen Enter-nappia painamalla.

Kuva: Chat-ikkuna.

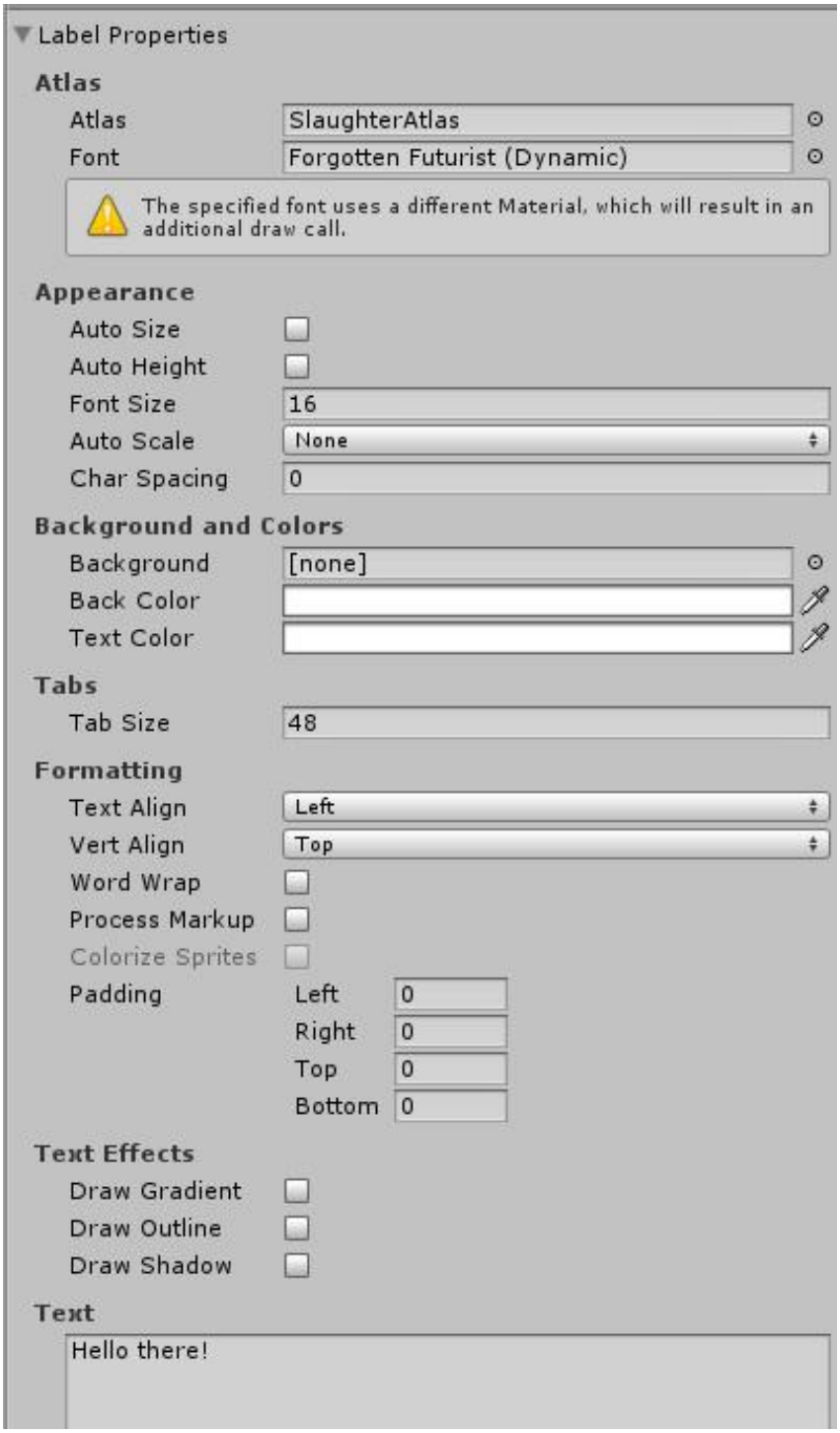
6.4.3.1 Viestialue

Chatin ylin objekti on Sprite-tekstuuri. Tekstuurin piirtämisen lisäksi se helpottaa chatin hallintaa ja siirtämistä. Vieritettävä tekstialue toteutetaan DfScrollPanel-elementillä. Siihen määritellään veritysalueen ominaisuudet.



Kuva: Viestialueen asetuksia.

Scroll Panelin alle luodaan DfLabel-elementti. Sen tehtävä on näyttää MessageQueuen sisältämä teksti. Scroll Panelilla on ongelmia näyttää alhaaltapäin alkavaa tekstiä oikein, joten toisista lisäosista poiketen teksti vierii tässä ylhäältä alas.



Kuva: Label-elementin asetuksia.

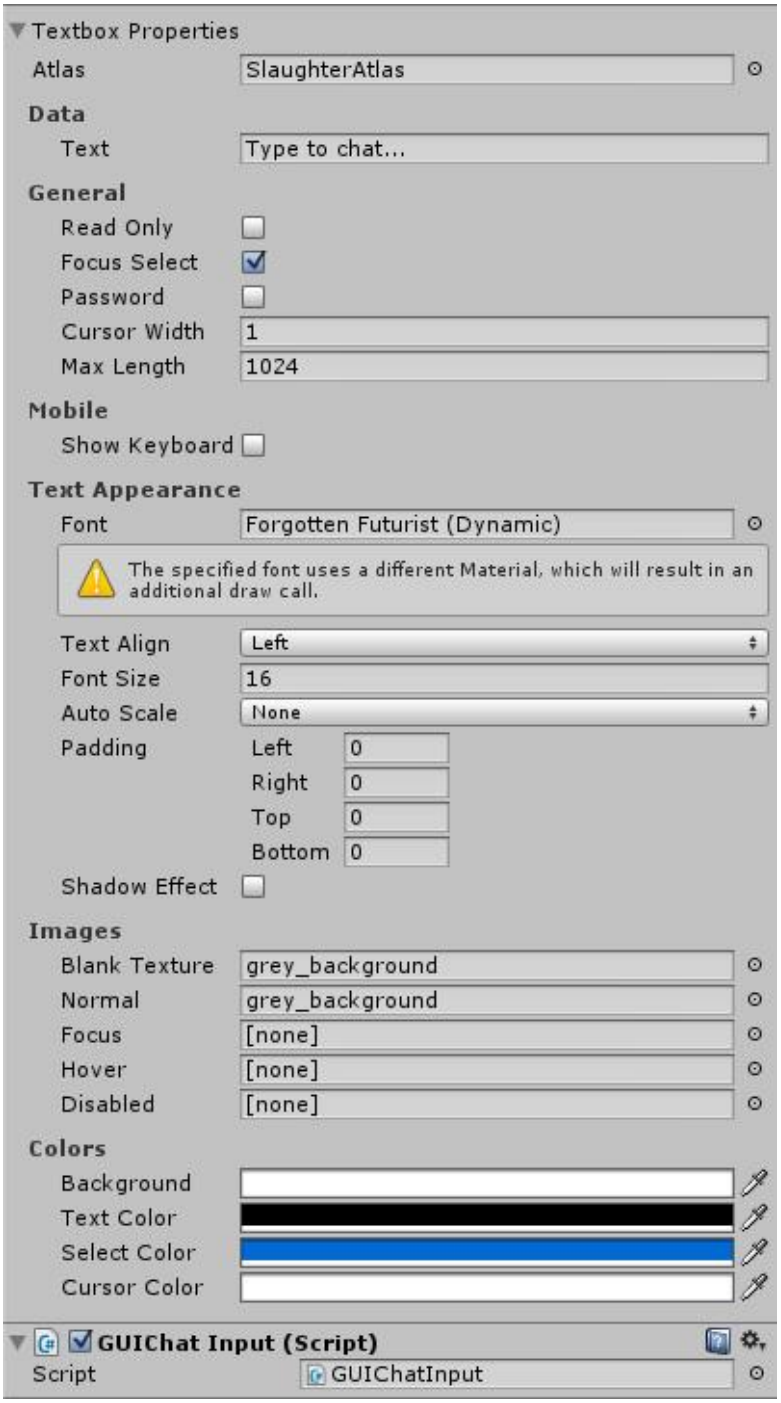
Koodissa Label-elementin text-muuttujan arvoa muokataan MessageQueuessa. Uuden viestin käsittelyn yhteydessä parsetetaan MessageQueuen sisältö merkkijonoksi ja asetetaan se text-muuttujaan.

```
public new void Enqueue(string msg)
{
    if ( Count > m_MaxMessagesInQueue ) Dequeue();
    base.Enqueue(msg);

    GameObject chatArea = GameObject.Find("ChatArea");
    if(chatArea != null)
    {
        chatArea.GetComponent<dfLabel>().Text = this.ToString();
    }
}

public string ToString()
{
    string msg = "";
    foreach(string s in this.ToArray())
    {
        msg = s + "\n" +msg;
    }
    return msg;
}
```

Kuva: MessageQueue asettaa Label-elementin arvoksi sisältämänsä tekstidatan.

6.4.3.2 Syöttökenttä

Chatin tekstinsyöttö toteutetaan DfTextbox-elementillä. Skriptille asetetaan vakioteksti, joka kertoo pelaajalle miten viestien kirjoittamisen voi aloittaa. Kun Focus Select -kohta on ruksittu, maalautuu vakioteksti, ja kirjoittamisen alottaminen on pelaajalle helpompaa.



Kuva: Textbox-elementin asetuksia.

GUIChatInput-skriptissä luetaan Textboxin sisältämä viesti, ja se lähetetään GUIScriptille.

```
[RequireComponent(typeof(dfTextbox))]
public class GUIChatInput : MonoBehaviour {

    GUIScript guiScript;
    dfTextbox mInput;

    void Update()
    {
        if(Input.GetKeyDown(KeyCode.Return))
        {
            enterLine();
        }
    }

    public void enterLine()
    {
        if (guiScript == null)
        {
            guiScript = GameObject.Find("NetworkGUI").GetComponent<GUIScript>();
        }


        mInput = GetComponent<dfTextbox>();
        guiScript.ChatMessage(mInput.Text);
        mInput.Text = "";
    }
}
```

Kuva: GUIChatInput-skriptin koodi.

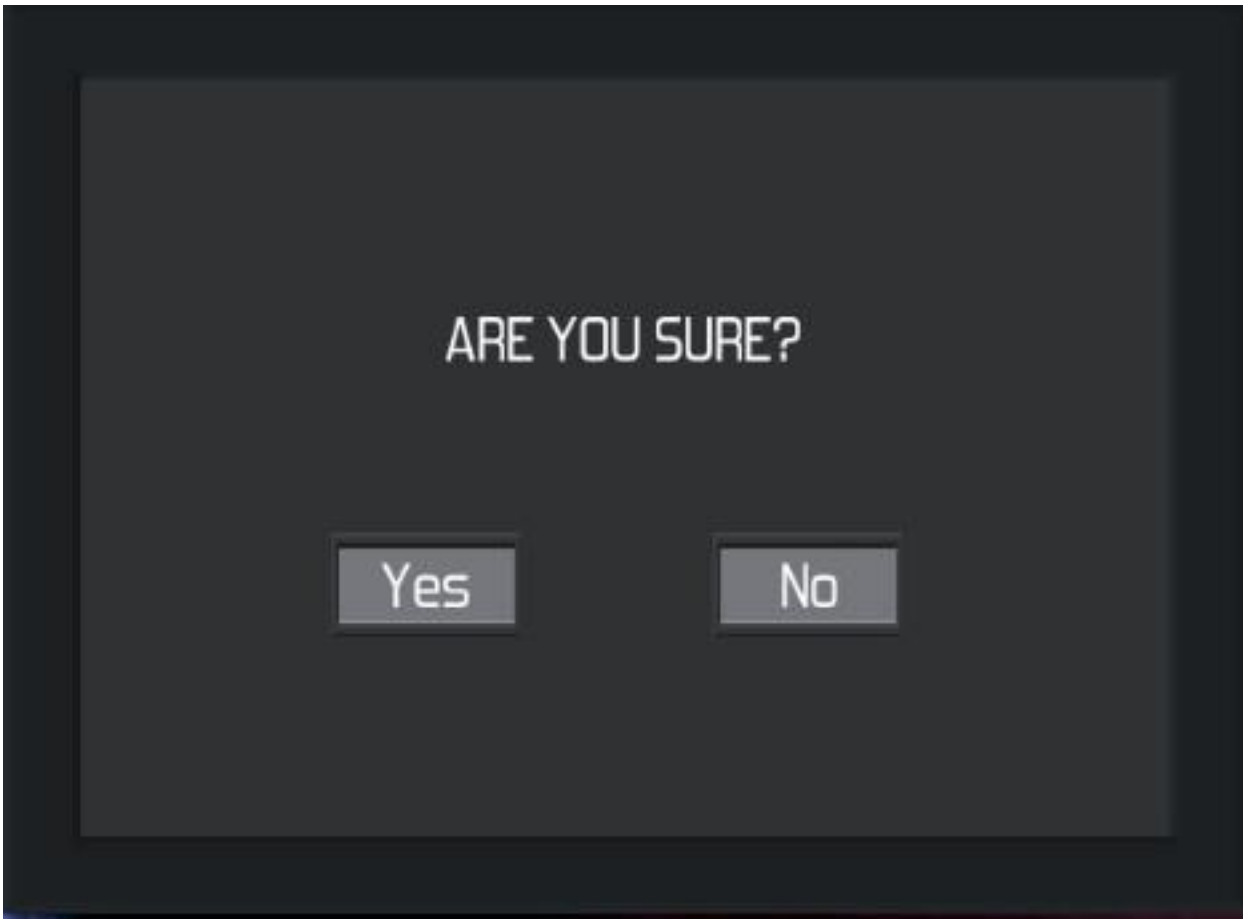
GUIScriptissä käsitellään viestin lähettäminen verkon yli kaikille pelaajille.

```
public void ChatMessage(string msg)
{
    object[] arr = new object[1];
    arr[0] = networkView.owner.ipAddress + ": " + msg;
    networkView.RPC("Chat", RPCMode.All, arr);
}

[RPC]
public void Chat( string msg )
{
    msgQueue.Enqueue(msg);
}
```

Kuva: ChatMessage-metodi.

6.4.4 Quit

Quit-painikkeesta avautuu paneeli, jossa pelaajalta varmistetaan, haluaako hän poistua pelistä. Yes-napista pelin ajaminen suljetaan ja No-napista paneeli sulkeutuu.



Kuva: Pelin sulkemisen varmistava paneeli.