KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Janne Ripatti

PELINKEHITYSPROSESSI: OTHELLO

Opinnäytetyö
Tammikuu 2022

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tammikuu 2022
Tietojenkäsittelyn koulutusohjelma

Tikkarinne 9
80200 JOENSUU
+358 13 260 600 (vaihde)

Tekijä(t)
Janne Ripatti

Nimeke
Pelinkehitysprosessi: Othello

Toimeksiantaja
-

Tiivistelmä

Toiminnallisen opinnäytetyöni aiheena on digitaalisen version kehittäminen Othello-lautapelistä. Opinnäytetyölläni pyrin vastaamaan seuraaviin kysymyksiin: Millainen on lautapelikonversion kehitysprosessi ideasta valmiiksi tuotteeksi? Millaista on yhden hengen pelinkehitys rajallisella budjetilla? Millaisia vaihtoehtoja PC-pelien digitaaliseen julkaisuun on?

Pelin toteutuksessa käytettiin Unity-pelimoottoria ja C#-ohjelmointikieltä. 3D-grafiikat mallinnettiin käyttämällä Blender-mallinnustyökalua. Pelin julkaisualustana käytettiin Itch.io-verkkosivua.

Pelin suunnittelu alkoi arvioimalla sen rakenne ja vaadittavat graafiset elementit. Myös tekoälyn logiikka suunniteltiin pääpiirteittäin tässä vaiheessa. Pelin toteutus alkoi pelinäkymällä ja sen toiminnallisuudella. Tavoitteena oli, että peliä ylipäätään pystyy pelaamaan ja että se valvoo sääntöjä. Perustoiminnallisuuden toteuttamisen jälkeen vuorossa oli pelin tekoäly, jonka jälkeen toiminnallisuuden osalta vuorossa oli enää alkuvalikon ja pelin lopputuloksen ja voittajan ilmoittavan ruudun tekeminen sekä liittäminen osaksi ohjelman kulkua. Lauta ja pelinappulat mallinnettiin pelin ohjelmoinnin lomassa.

Kokonaisuutena olen tyytyväinen valmiiseen tuotokseen. Pelistä tuli suunnilleen sellainen kuin oli tarkoituskin. Oppimisen kannalta projekti tarjosi paljon.

Kieli
suomi

Sivuja 50

Asiasanat
pelinkehitys, lautapeli, konversio

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
January 2022
Degree Programme in Business Information Technology

Tikkarinne 9
80200 JOENSUU
FINLAND
+ 358 13 260 600 (switchboard)

Author (s)
Janne Ripatti

Title
Game Development Process: Othello

Commissioned by
-

Abstract

The topic of my practice-based thesis is the development of a digital version of Othello board game. With my thesis, I aim to answer the following questions: What is the development process of a board game conversion from an idea to a finished product? What is one-person game development with a limited budget like? What options are there for publishing a PC game digitally?

Implementing the game, Unity game engine and C# programming language were used. The 3D graphics were modeled using Blender modeling tool. Itch.io website was used as the publishing platform.

The planning of the game began by assessing its structure and the required graphical elements. The AI's logic was also preliminarily designed at this time. Implementing the game began with the game view and its functionality. The aim was that the game could be played and that it would keep track of the rules. After the basic functionality, it was time for the game's AI, after which the final stage for the functionality was creating the title menu and the screen announcing the final score and the winner and inserting them in the program's flow. The board and the game pieces were modeled concurrently with the coding.

As a whole, I am pleased with the finished product. The game ended up largely like it was intended to be. Learning-wise the project provided a lot.

Language
Finnish

Pages 50

Keywords
game development, board game, conversion

Sisältö

1 Johdanto

Toiminnallisen opinnäytetyöni aiheena on digitaalisen version kehittäminen Othello-lautapelistä. Tavoitteenani on tehdä, valmista pelimoottoria käyttäen, julkaisukelpoinen peli 3D-grafiikoilla ja alkeellisella tekoälyllä, sekä lopuksi myös julkaista valmis tuotos internetissä. Arvioin tällaisen projektin olevan työmäärältään sopiva yksin tehtäväksi ja testaavan hyvin taitojani suunnittelussa, ohjelmoinnissa ja mallintamisessa.

Johdannon ja loppuyhteenvedon lisäksi tämä raportti koostuu kolmesta luvusta. Toisessa luvussa käsittelen projektin vaatimuksia ja toteutussuunnitelmaa. Kolmannessa luvussa käsittelen Othelloa itseään, käyden läpi sen historiaa ja säännöt. Neljännessä luvussa käyn läpi työkalut, joita tulen käyttämään projektissani ja miksi päädyin valitsemaan ne. Viidennessä luvussa tarkastelen eri julkaisualustoja. Kuudennessa luvussa käyn läpi projektin toteutuksen kohta kohdalta.

2 Projektin vaatimukset

Opinnäytetyöni tavoitteena on tehdä, valmista pelimoottoria ja ilmaisia työkaluja käyttäen, digitaalinen versio Othello-lautapelistä. Tarkoituksena on saada aikaan julkaisukelpoinen tuotos, 3D-grafiikoilla ja jonkinlaisella tekoälyllä, ja lopuksi myös julkaista se internetissä. Opinnäytetyölläni pyrin vastaamaan seuraaviin kysymyksiin: Millainen on lautapelikonversion kehitysprosessi ideasta valmiiksi tuotteeksi? Millaista on yhden hengen pelinkehitys rajallisella budjetilla? Millaisia vaihtoehtoja PC-pelien digitaaliseen julkaisuun on?

Pelin toteutuksessa tulen käyttämään Unity-pelimoottoria ja C#-ohjelmointikieltä. 3D-grafiikat mallinnan käyttämällä

Blender-mallinnustyökalua ja mahdollisesti tarvittavat 2D-grafiikat piirrän Gimp-piirtotyökalulla. Kaikki nämä työkalut ovat ilmaisia. Pelin julkaisualustana tulen käyttämään Itch.io-verkkosivua. Luvussa kaksi mainittuja opinnäytetöitä saatan hyödyntää kehitysprosessin suunnittelussa, mutta aikomuksenani ei ole verrata omaa työtäni aiempiin koska en näe sitä tarpeelliseksi.

Itse kehitysprosessin uskon, pienestä skaalastaan johtuen, kulkevan yksinkertaisella kaavalla, jossa suunnittelen ensin kaiken mahdollisimman yksityiskohtaisesti paperilla ja tämän jälkeen toteutan Unityllä, valmistaen graafiset elementit sitä mukaa kun niitä tarvitaan.

Pyrin opinnäytetyössäni mahdollisimman objektiiviseen näkökulmaan niiltä osin kuin se on mahdollista, tarkoittaen työn teoreettista osuutta. Pelinkehitysprosessia käsittelevä osuus tulee olemaan subjektiivinen, koska kirjoitan sen omien kokemusteni pohjalta. Eettisiä kysymyksiä en näe työlläni olevan.

Tarkoitukseni on toteuttaa peli kesän aikana, niin että se on julkaisuvalmis elokuussa. Tuolloin aloitan myös raportin kirjoittamisen. Lopullisesti opinnäytetyöni tulee siten valmistumaan syksyn aikana. Ulkopuolista rahoitusta opinnäytetyöni ei vaadi.

Toivon mukaan valmis opinnäytetyöni tulee tarjoamaan hyvän kuvauksen pienen skaalan pelinkehityksestä ja julkaisuvaihtoehdoista vuonna 2021. Itse toivon saavani hyvää kokemusta työskentelystä pelinkehitysprojektin parissa sekä valmiin näytteen ohjelmointitaidoistani.

3 Othello

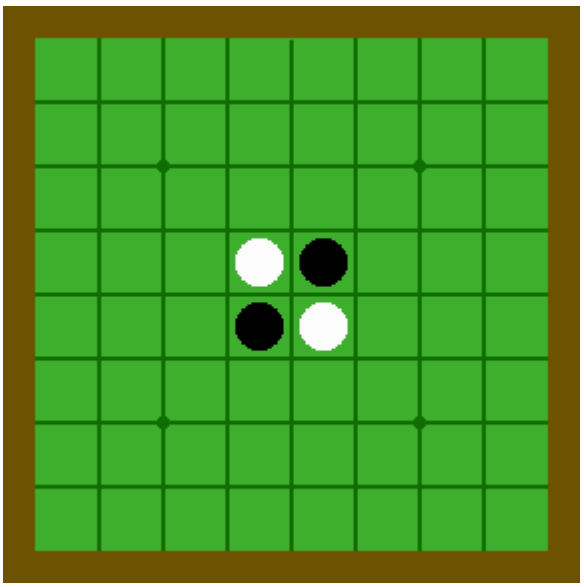
3.1 Esittely

Othello on kahden pelaajan lautapeli. Othello pohjautuu Isossa-Britanniassa 1880-luvulla keksittyyn Reversi-lautapeliin. Japanilainen Goro Hasegawa lisäsi 1960-luvulla siihen alkumuodostelman ja joitain muita peliä parantavia sääntöjä, ja nimesi luomuksensa Othelloksi (Law 2010.)

World Othello Federation (WOF) pyrkii edistämään Othelloa, ylläpitää pelaajien vahvuuslukuja ja järjestää vuosittain Othellon maailmanmestaruusturnauksen. (World Othello Federation 2021a).

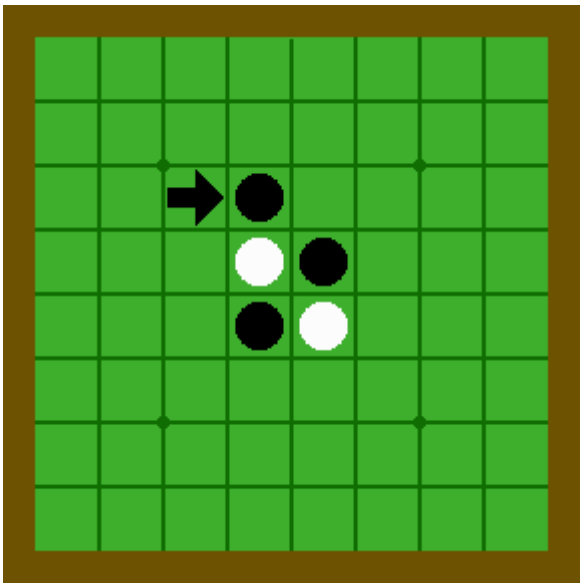
3.2 Säännöt

Othellon pelivälineet ovat 8x8 ruudun lauta ja 64 kiekkoa, jotka ovat toiselta puolelta mustia ja toiselta valkoisia. Pelaajat (musta ja valkoinen) asettavat vuorotellen kiekkoja laudalle, oma väri ylöspäin. Lopussa voittaja on se pelaaja, jonka värisiä kiekkoja laudalla on enemmän. Pelin alkaessa kukin pelaaja asettaa laudalle kaksi kiekkoa kuvassa 1 näkyvään muodostelmaan. (World Othello Federation 2021b.)

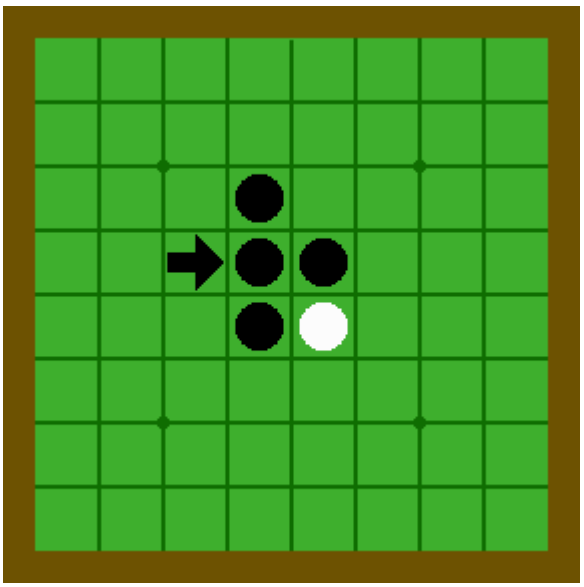


Kuva 1. Aloitus.

Pelaajan asettaessa kiekon laudalle, jokainen asetetun kiekon ja toisen samanvärisen kiekon väliin jäävä kiekko vaihtaa väriä (eli kääntyy ympäri) siirtovuorossa olevan pelaajan väriseksi (World Othello Federation 2021b).

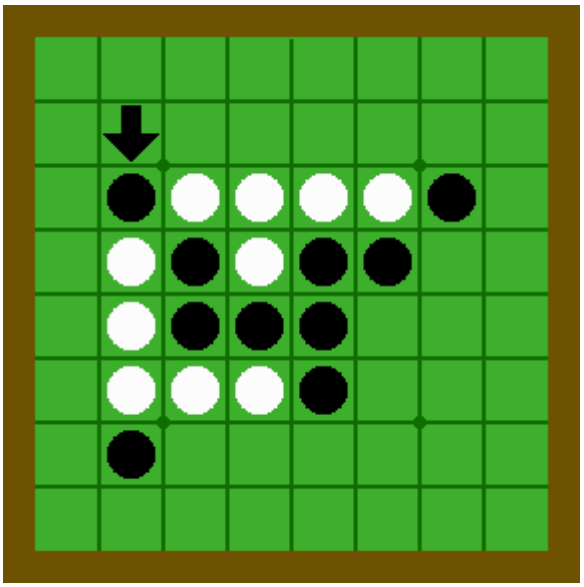


Kuva 2. Musta pelaaja asettaa kiekon nuolen osoittamaan ruutuun.

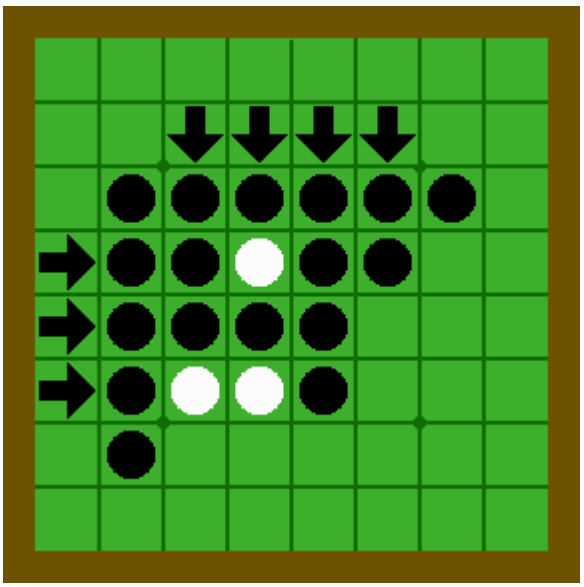


Kuva 3. Nuolen osoittama kiekko vaihtaa väriä.

Kääntyviä kiekkoja voi olla peräkkäin kuinka monta tahansa, vaakasuoraan, pystysuoraan tai viistoon, kuten seuraavassa esimerkissä näemme (World Othello Federation 2021b).

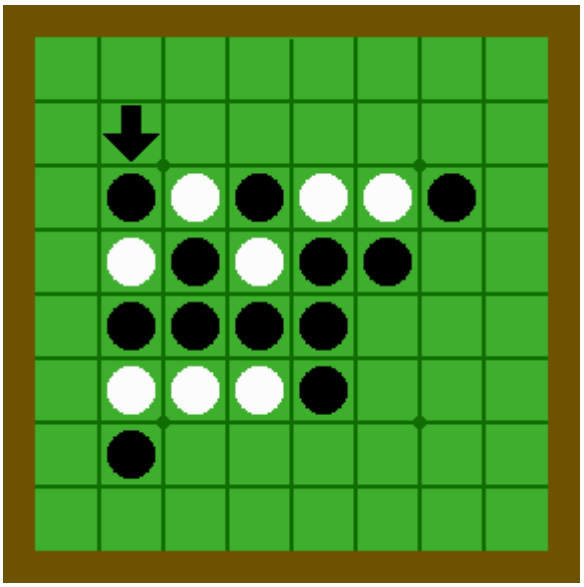


Kuva 4. Musta pelaaja asettaa kiekon nuolen osoittamaan ruutuun.

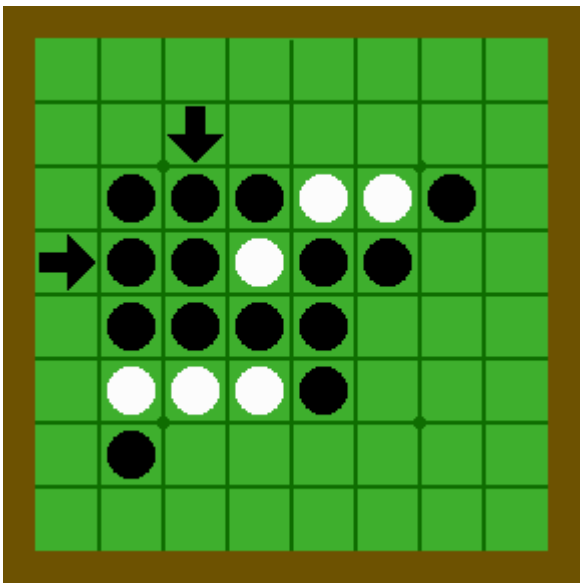


Kuva 5. Seitsemän kiekkoa vaihtaa väriä.

Väriä vaihtavien kiekkojen sarja ei kuitenkaan voi ylittää pelaajan omanvärisiä kiekkoja, kuten seuraava esimerkki näyttää (World Othello Federation 2021b).



Kuva 6. Musta pelaaja asettaa kiekon nuolen osoittamaan ruutuun.



Kuva 7. Tällä kertaa vain kaksi kiekkoa vaihtaa väriä.

3.3 Pelin kulku

Musta pelaaja aloittaa. Jos pelaaja ei voi asettaa kiekkoa niin että se kääntää vastustajan kiekkoja, vuoro siirtyy vastapelaajalle. Vuoroa ei voi vapaaehtoisesti jättää väliin, vaan pelaajan on asetettava kiekko laudalle, jos se vain on mahdollista. (World Othello Federation 2021b.)

Peli päättyy, kun kumpikaan pelaaja ei voi asettaa kiekkoa laudalle. Peli voi siis päättyä ennen kuin lauta on täysi. Pelin lopussa kiekot lasketaan ja voittaja on se pelaaja, jonka värisiä kiekkoja laudalla on enemmän. (World Othello Federation 2021b.)

4 Tietoperusta

Theseuksesta löytyy jo muutama lautapelien digitaalikäännöksiä käsittelevä opinnäytetyö. Näistä Nico Hämäläisen Riichi-mahjong-tekoäly ja Jarkko Kuuselan Tietokoneshakin ohjelmointi JavaScript-kielellä keskittyvät tekoälyohjelmointiin, kun taas Pasi Vilppolan Mobile game prototyping: Programmer’s point of view käsittelee aihetta suunnittelun ja prototyyppauksen näkökulmasta (Hämäläinen 2019; Kuusela 2017; Vilppola 2015). Sekä Teemu Ikosen ja Lauri Lipposen Lautapelistä digitaaliseksi peliksi Case: Muistatko vielä… että Pyry Salosen Tutustumiskorttipeli mobiilimuotoon käsittelevät pelin kääntämisprosessia itseään, vaikkakin jääden edellisessä tapauksessa beta-version ja jälkimmäisessä prototyypin asteelle. (Ikonen & Lipponen 2016; Salonen 2019). Saman tekee Juho-Pekka Pirskasen Lautapelin kääntäminen mobiilialustalle ja konversioiden käytettävyys, joka enemmänkin tarkastelee lautapelikonversioiden historiaa ja tilaa julkaisuvuonnaan (Pirskanen 2014).

Yksikään edellämainituista opinnäytetöistä ei siis kuvaa kääntämisprosessia valmiin tuotteen julkaisemiseen asti. Niinpä oma työni eroaa näistä, koska sen tavoitteena on tehdä julkaisukelpoinen lautapelikäännös ja lopuksi myös julkaista se.

5 Käytetyt ohjelmat ja työkalut

5.1 Pelimoottori

5.1.1 Unity

Unity on Unity Technologies -yhtiön kehittämä grafiikkamoottori, jolla voidaan luoda 2D- ja 3D-pelejä tai muita visuaalisia ohjelmia (Unity Technologies 2021a). Unitylla luotuja ohjelmia voidaan julkaista lukuisille eri alustoille: tietokoneiden lisäksi myös mobiilialustoille, pelikonsoleille ja verkkoselaimille (Unity Technologues 2021b).

5.1.2 Valinta

Valitsin Unityn tätä projektia varten, koska se on ilmainen ja minulla on sen käytöstä aiempaa kokemusta muun muassa harjoittelujaksoni ajalta. Ennen kaikkea se täyttää tarpeeni pelimoottorista, joka kykenee esittämään 3D-grafiikoita.

5.2 Grafiikat

5.2.1 Blender

Blender on Blender Foundation -yhtiön kehittämä ilmainen, avoimeen lähdekoodiin pohjaava 3D-grafiikan luontityökalu. Se mahdollistaa kaikki grafiikkaliukuhihnan vaiheet: mallinnuksen, animoinnin, renderöinnin ja jälkikäsittelyn. Siitä löytyy omat, keskenään yhteensopivat, versiot Windowsille, Macintoshille ja Linuxille. (Blender.org 2021.)

5.2.2 Valinta

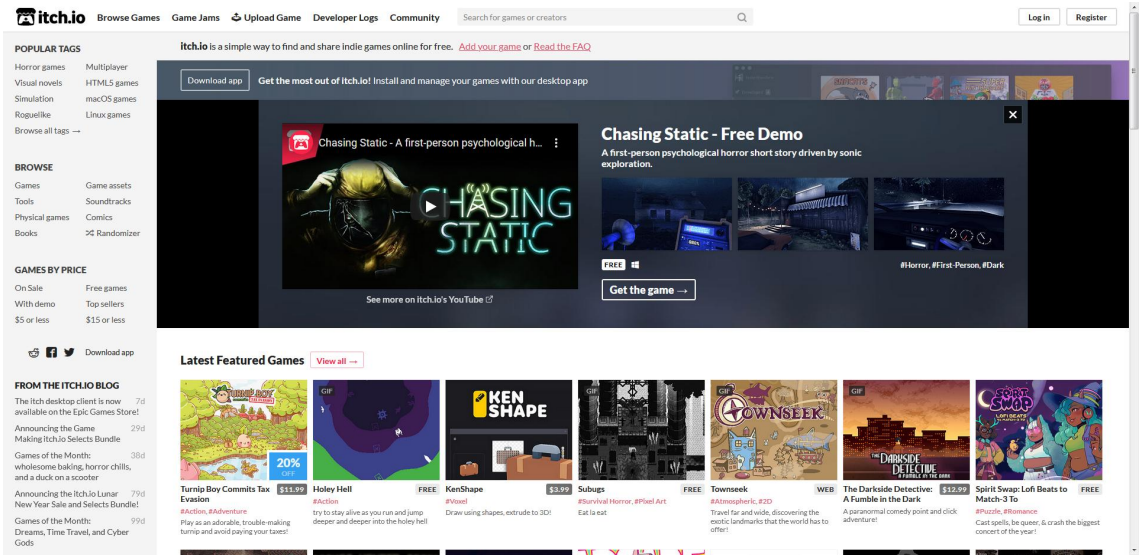
Valitsin Blenderin tätä projektia varten koska se on ilmainen ja minulla on myös sen käytöstä aiempaa kokemusta opintojeni ajalta.

5.3 Plastic SCM

Plastic SCM on Codice Software -yhtiön kehittämä graafinen ja pilvipohjainen versionhallintajärjestelmä, josta löytyy omat, keskenään yhteensopivat versionsa Windowsille, Linuxille ja Macintoshille (Codice Software 2021a). Plastic SCM on ilmainen enintään kolmen käyttäjän tiimeille ja tarjoaa 5 gigatavua ilmaista tallennuskapasiteettia. Ammattilaiskäyttöön siitä löytyy myös kalliimpi Enterprise-versio (Codice Software 2021b).

6 Julkaisualusta

6.1 Itch.io



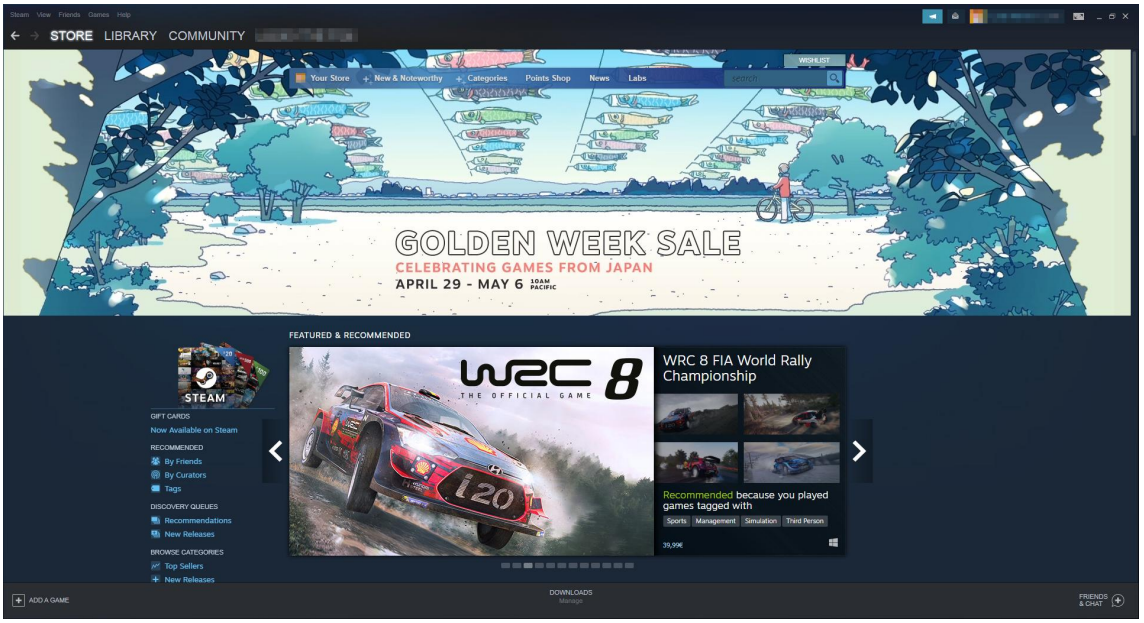
Kuva 8. Itch.io:n verkkosivu.

Itch.io on videopelien julkaisemiseen tarkoitettu verkkosivu. Se on perustettu vuonna 2013 ja nykyisellään sieltä löytyy pelejä yli 380 000 kappaletta. (Itch corp 2021a.)

Pelejä voi julkaista ilmaiseksi tai maksua vastaan. Jälkimmäisessä tapauksessa julkaisija määrittää pelilleen alkuhinnan ja ostaja voi halutessaan maksaa siitä enemmänkin. Lisäksi myyjä voi halutessaan määrittää, kuinka suuri osa hinnasta menee Itch.ion ylläpitäjille ja kehittäjille. Muuten Itch.ion käyttö on kuitenkin täysin ilmaista. (Itch corp 2021b.)

Itse maksaminen tapahtuu Paypalin tai Stripen kautta suoraan julkaisijan itsensä tilille. Vaihtoehtoisesti Itch.io veloittaa pelit itse ja pyydettäessä siirtää tuoton julkaisijan tilille. Tällöin Euroopan Unionin arvonlisävero lisätään hintaan ja maksetaan eteenpäin automaattisesti. Suorassa veloituksessa tämä on julkaisijan itsensä vastuulla. (Itch corp 2021c.)

6.2 Steam

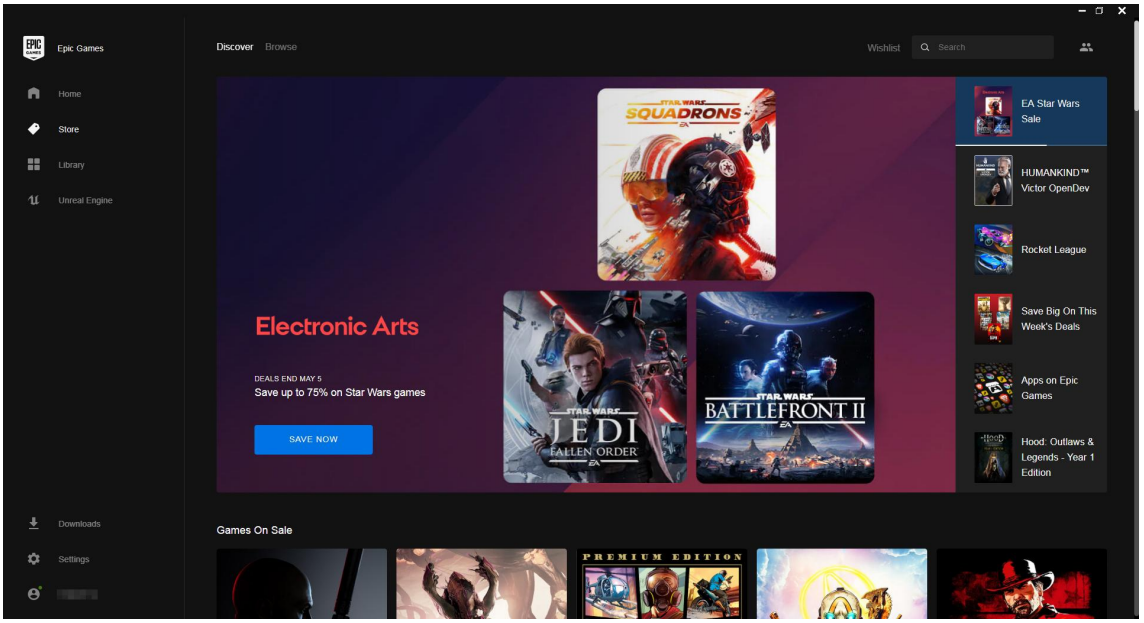


Kuva 9. Steamin kauppanäkymä.

Steam on Valve-yhtiön julkaisualusta. Se on perustettu vuonna 2003 ja sisältää nykyisellään yli 51 000 peliä (Valve Corporation 2021). 75%:n markkinaosuudellaan se on maailman suurin PC-pelien julkaisualusta (Zuckerman 2020).

Pelien julkaiseminen Steamissä maksaa 100 Yhdysvaltain dollaria per peli (Steamworks 2021a). Lisäksi Valve ottaa jokaisen myydyn pelin hinnasta 30% (Xsolla 2020). Tämä osuus tosin pienenee jos peliä on myyty yli miljoonan dollarin edestä (Steamworks 2021b). Myyntitulot Steam tilittää julkaisijalle seuraavan kuukauden lopussa, siis esimerkiksi helmikuun aikana myytyjen pelien tuotto tilitetään 30. maaliskuuta (Steamworks 2021b). Arvonlisäveron lisääminen ja eteenpäin maksaminen hoidetaan julkaisijan puolesta (Steamworks 2021c). Tilitys tapahtuu ainoastaan dollareina (Steamworks, 2021b).

6.3 Epic Game Store



Kuva 10. Epic Game Storen kauppanäkymä.

Epic Games Store on Epic Games -yhtiön vuonna 2018 avattu, ja siten jokseenkin tuore, videopelien julkaisualusta (Backlinko 2021). Pelejä sieltä löytyy nykyisellään yli 470 kappaletta (Backlinko 2021). Epic Games Storen etuna Steamiin verrattuna on se, että Epic Games ottaa vain 12% pelien myyntituloista (Xsolla 2021). Lisäksi sen pelikirjaston suhteellisesta pienuudesta johtuen pelien on siellä helpompi saada näkyvyyttä kuin Steamissä (Xsolla 2021).

Kääntöpuolena Epic Games Store on yhä varsin keskeneräinen ja monet sen ominaisuudet ovat vielä kehityksessä tai lupauksen asteella (Trello 2021). Julkaisijan näkökulmasta erityisen hankalaa on se, että julkaistavaa peliä täytyy tarjota lomakkeen välityksellä Epic Gamesille, joka sitten ottaa sen valikoimaansa tai ei (Epic Games, Inc 2021). Kriteerit tälle valinnalle eivät ole kovinkaan läpinäkyvät ja potentiaaliselle julkaisijalle tarjottu dokumentaatio on muutenkin kovin suppeaa.

6.4 Valinta

Valitsin pelini julkaisualustaksi Itch.io:n, koska siellä julkaiseminen on ilmaista eikä vaadi erillistä hyväksyntää. Aloitteleville ja voittoa tavoittelemattomille julkaisijoille se on siten mielestäni sopivin vaihtoehto. Jos haluaisin myydä peliäni, saattaisi Steam olla parempi vaihtoehto markkinaosuutensa johdosta. Vaikka Epic Game Store tarjoaakin julkaisijalle edullisemmat myyntiehdot, sitä ei voi tällä hetkellä suositella kuin tunnetuille julkaisijoille sen hakemusjärjestelmästä johtuen.

7 Toteutus

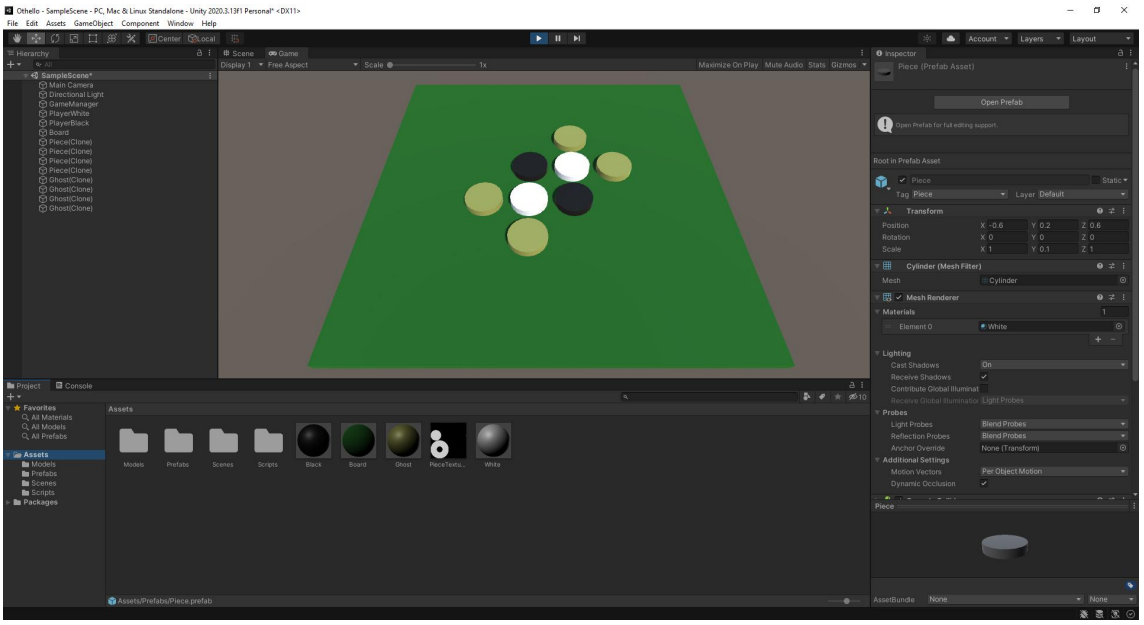
7.1 Suunnittelu

Pelin suunnittelu alkoi arvioimalla sen rakenne ja vaadittavat graafiset elementit. Peli tulisi koostumaan kolmesta erillisestä tilasta: aloitusruutu, peli itse ja pelin päättymisruutu. Kolmiulotteisia graafisia elementtejä tultaisiin tarvitsemaan pelilaudan ja nappuloiden muodossa. Nappuloiden laudalle lisääminen ja kääntyminen tultaisiin lisäksi animoimaan. Lisäksi pohdittiin, tarvittaisiinko aloitus- ja päättymisruutuihin erillistä grafiikkaa, mutta lopulta ne päädyttiin toteuttamaan tekstimuotoisina. Myös tekoälyn logiikka suunniteltiin pääpiirteittäin tässä vaiheessa.

Ohjelmoinnin näkökulmasta erilaisia olioita tulisi olemaan verrattain vähän: kamera pelikuvan näkymiseen, valaistuslähde, lauta ja pelin edetessä luotavat nappulat, mukaan lukien läpikuultaviksi visioimani väliaikaiset nappulat kertomaan pelaajalle mahdollisista siirroista. Näiden lisäksi tultaisiin luomaan pelaajalle näkymätön GameManager-olio valvomaan pelin sääntöjä ja etenemistä.

7.2 Toteutus

Pelin toteutus alkoi pelinäkymällä ja sen toiminnallisuudella. Tavoitteena oli, että peliä ylipäätään pystyy pelaamaan ja että se valvoo sääntöjä. Tässä vaiheessa käytettiin väliaikaisina graafisina elementteinä yksinkertaisia nelikulmio- ja sylinterimuotoja, joita Unityn editori kykenee luomaan itse. Nämä voitaisiin myöhemmin yksinkertaisesti korvata lopullisilla 3D-malleilla ilman että se vaikuttaa pelin toiminnallisuuteen. Pelin koodin jaettiin lauta- ja GameManager-olioiden välille niin että metodit esimerkiksi nappuloiden lisäämistä ja kääntämistä varten ovat liitettyinä lautaan, josta GameManager pystyy kutsumaan niitä pelin edetessä. Pelin käyttöliittymään kuuluvat metodit sisältyvät GameManager-olioon (kuva11).



Kuva 11. Kehitysversio väliaikaisilla grafiikoilla.

Perustoiminnallisuuden jälkeen vuorossa oli pelin tekoäly. Myös tekoälyn sisällytettiin GameManager-olioon, vaikkakin sille olisi voinut luoda myös oman olionsa. Pelin pienestä skaalasta johtuen päädyttiin kuitenkin tähän ratkaisuun. Koska tekoälyn tarkoituksena ei ollut niinkään tarjota todellista haastetta vaan

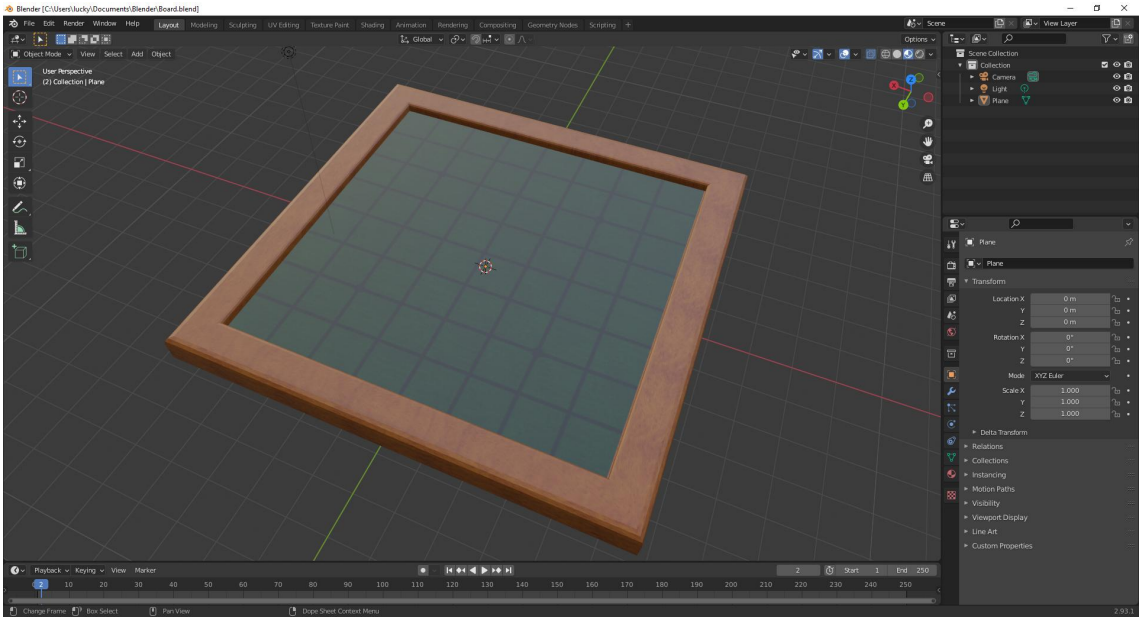
pelkästään vaihtoehto ihmispelaajalle, sovellettiin yksinkertaista kaavaa, jossa jokaiselle pelilaudan ruudulle määritetään arvo sen mukaan, kuinka edukasta pelaajan on saada nappulansa siihen. Ääriesimerkkeinä kulmat ovat kaikkein arvokkaimpia koska niihin asetettua nappulaa ei pysty kääntämään, kun taas näiden viereiset ruudut ovat vähiten arvokkaita, koska ne auttavat vastustajaa saamaan nappulansa edellämainittuihin kulmiin. Tekoäly valitsee nappulalleen aina arvokkaimman saatavilla olevan ruudun.

1	5	3	3	3	3	5	1
5	5	4	4	4	4	5	5
3	4	2	2	2	2	4	3
3	4	2			2	4	3
3	4	2			2	4	3
3	4	2	2	2	2	4	3
5	5	4	4	4	4	5	5
1	5	3	3	3	3	5	1

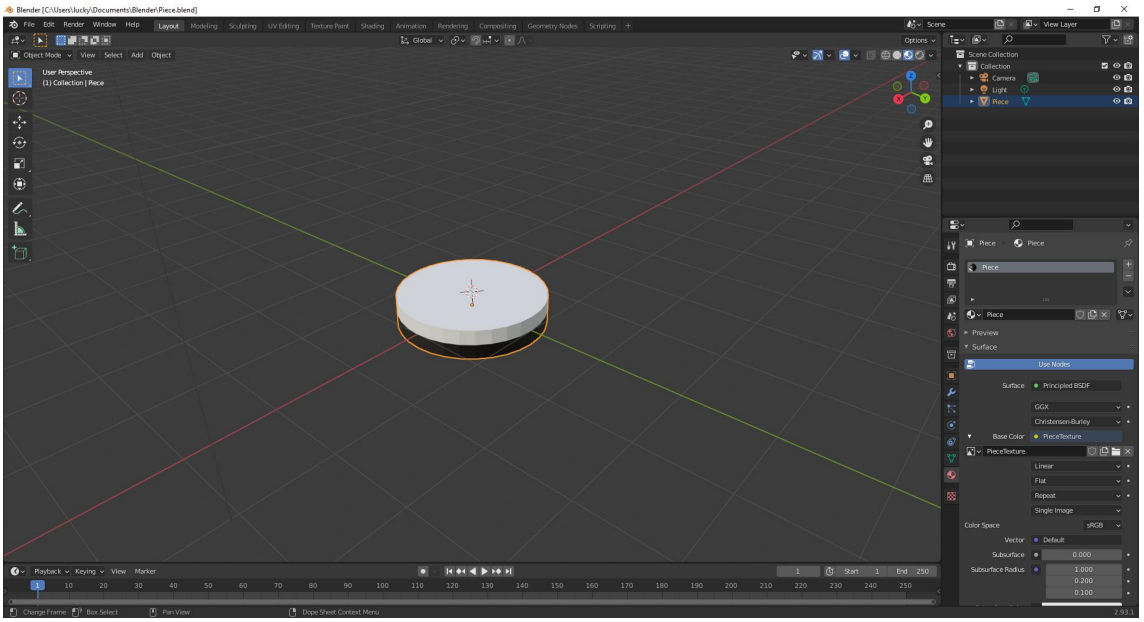
Kuva 12. Tekoälyn logiikka.

Tekoälyohjelmoinnin jälkeen toiminnallisuuden osalta vuorossa oli enää alkuvalikon ja pelin lopputuloksen ja voittajan ilmoittavan ruudun tekeminen, sekä liittäminen osaksi ohjelman kulkua. Koska nämä muodostuisivat lähinnä tekstistä ja alkuvalikossa muutamasta klikattavasta painikkeesta, pystyi ne toteuttamaan yksinkertaisesti käyttämällä Unityn omia 2D-elementtien piirtotyökaluja.

Tässä projektissa se tarkoitti kahden koko ruudun kokoisen overlayn luomista, yhden alkuvalikkoa ja toisen pistenäkymää varten, ja niihin tekstikenttien ja painikkeiden liittämistä. Osan käyttöliittymän toiminnallisuudesta pystyy toteuttamaan ilman koodausta, valitsemalla suoraan editorissa esimerkiksi mitä tapahtuu kun painiketta klikataan. Tässä tapauksessa kuitenkin liitettiin jokaiseen painikkeeseen metodi, joka kutsutaan klikattaessa. Ainoastaan alkuvalikon piilottaminen start-painiketta painettaessa toteutettiin editorin sisällä.



Kuva 13. Pelilaudan malli Blenderissä.

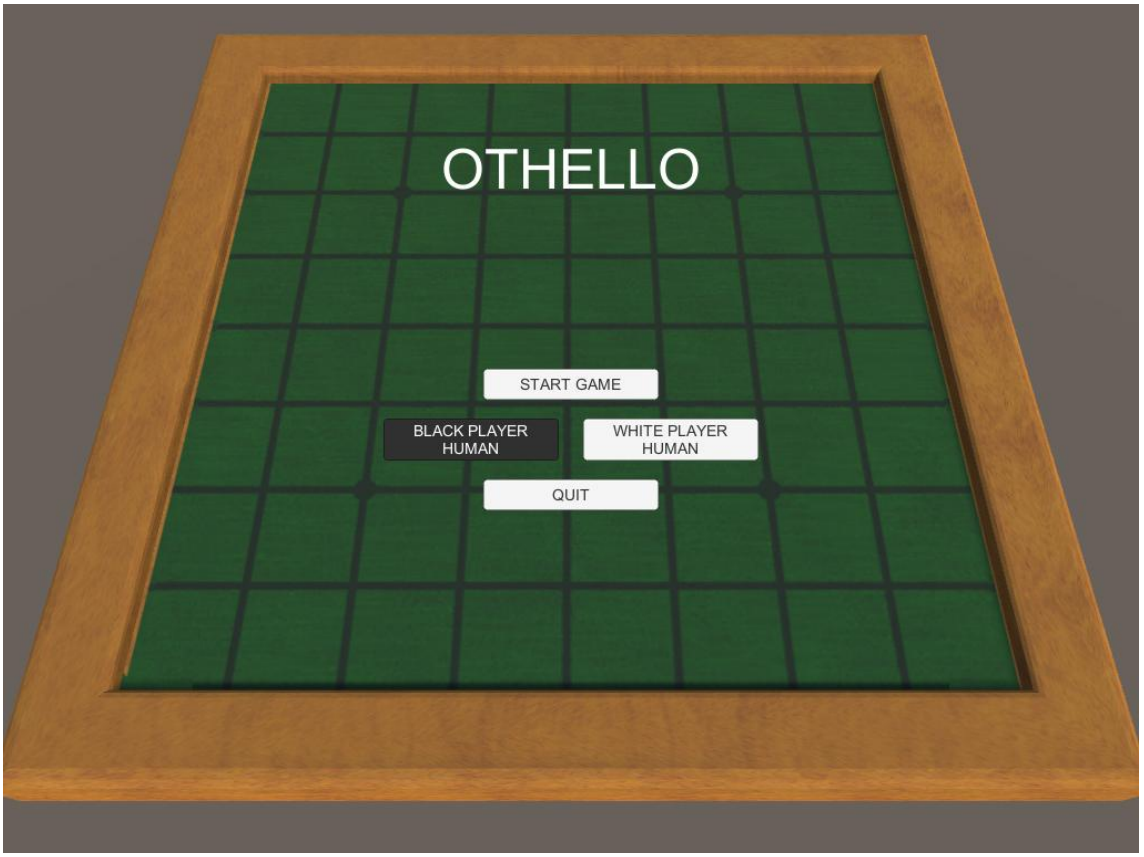


Kuva 14. Pelinappulan malli Blenderissä.

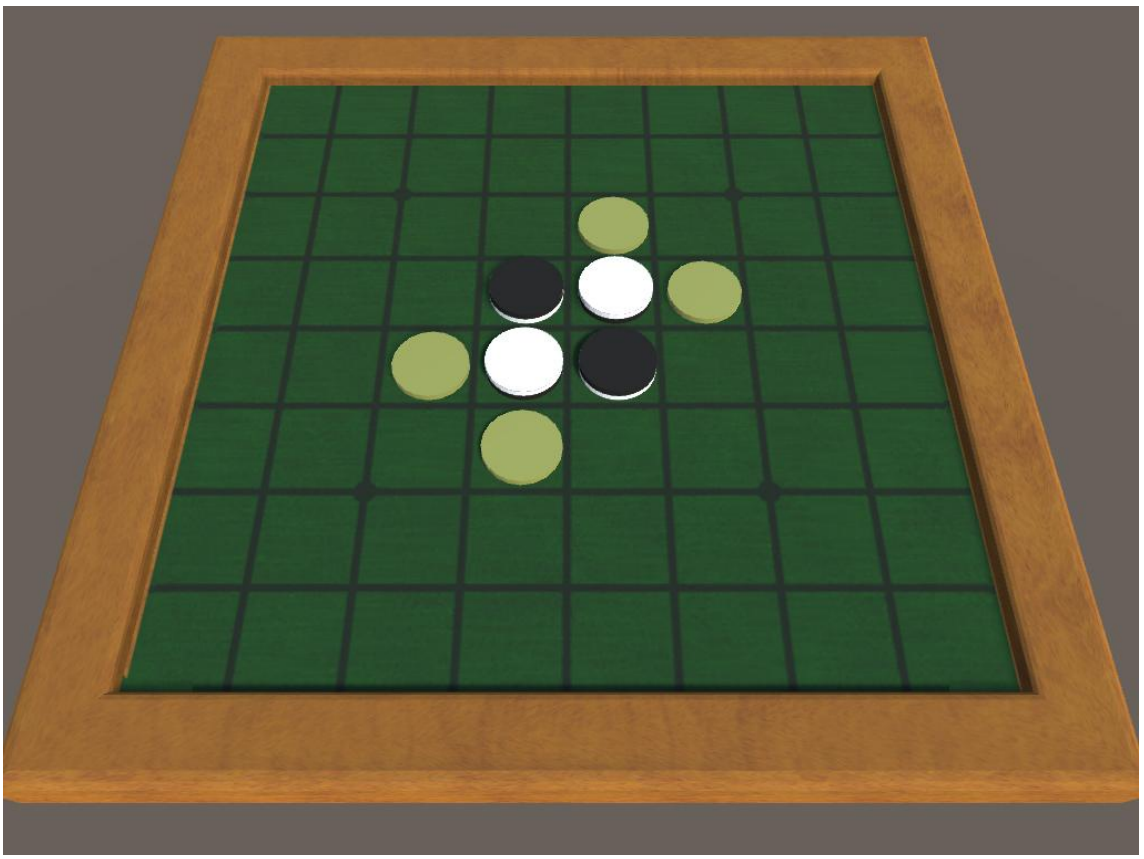
Lauta ja pelinappulat mallinnettiin ohjelmoinnin lomassa. Lauta muotoiltiin nelikulmiosta kaivertamalla. Teksturoinnin tehtiin kahdessa osassa. Laudan reunus maalattiin puukuviolla Blenderissä itsessään, kun taas pelialue liitettiin Photoshopissa suoraan mallin tekstuuritiedostoon. Nappulan mallintaminen oli helpompaa. Yksinkertaisesti mallinnettiin litistetty sylinteri ja jaettiin se keskeltä kahteen osaan, jotka maalattin valkoisella ja mustalla.

Teksturointiin käytettiin internetistä löydettyjä ilmaisia puu- ja samettitekstuureja. Näistä jälkimmäistä muokattiin lisäämällä siihen pelissä tarvittava ruudukko ja vaihtamalla värisävyä. Aikataulukiireiden vuoksi nappuloille ei lopulta tehty minkäänlaisia animaatioita, alkuperäisestä suunnitelmasta poiketen. Myös mahdollisia siirtoja indikoivien aavenappuloiden läpikuultavuudesta luovuttiin, tyytyen jättämään ne vain kellertäviksi sylintereiksi.

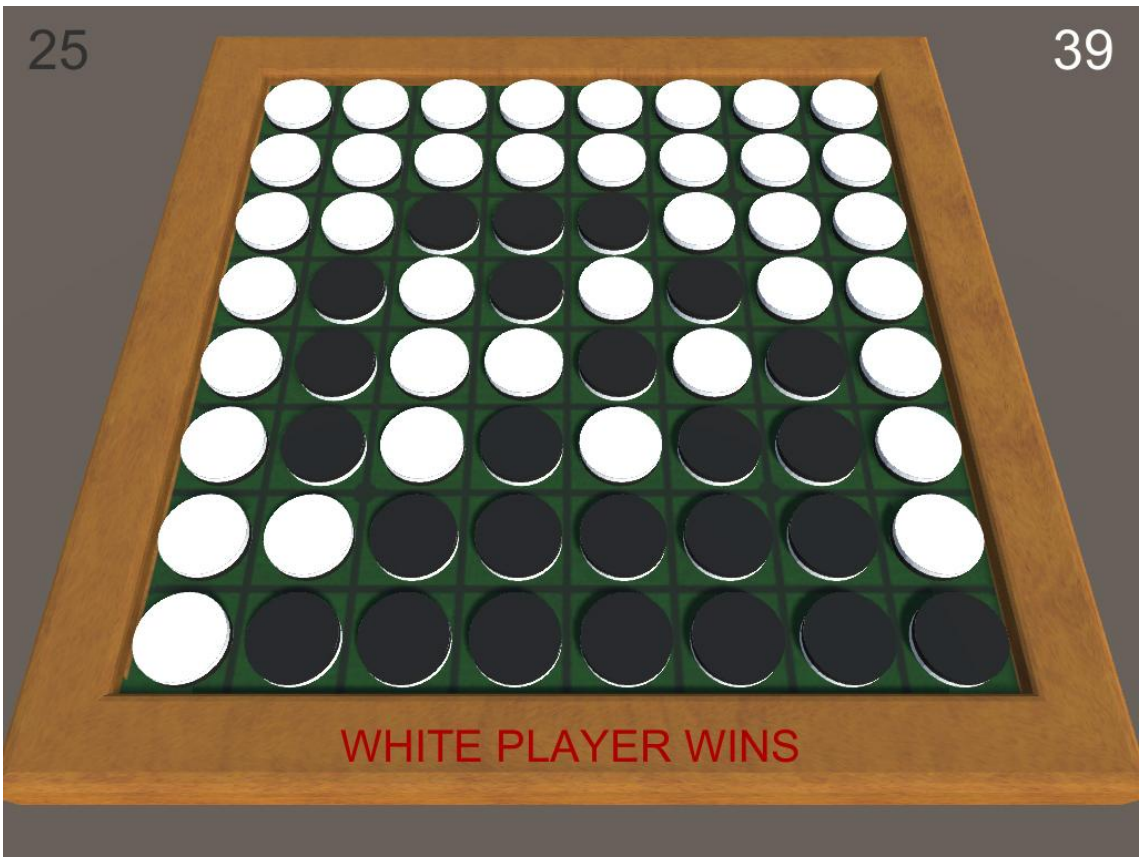
3D-mallit pystyi siirtämään Blenderistä suoraan Unityn ymmärtämään .fbx-muotoon, tekstuurien kulkiessa .png-tiedostoina. Unity tukee 3D-malleissa myös muita tiedostomuotoja. Väliaikaisten mallien vaihtaminen lopullisiin oli odotetun vaivatonta.



Kuva 15. Valmiin version alkuvalikko.



Kuva 16. Valmiin version pelinäkymä.



Kuva 17. Pelin loppuruutu.

7.3 Ohjelmakoodi

7.3.1 Board-luokan metodit

```
void Start()
{
    pieces = new GameObject[8, 8];
}
```

Listaus 1. Start-metodi.

Ajetaan ohjelman käynnistyessä. Luo 8x8 arrayn pelinappuloiden tallentamista varten.

```
public void AddPiece(bool piecePlayer, int pieceX, int pieceY)
{
    // Instantiate the piece and add variables to it
    GameObject piece = Instantiate(piecePrefab);
    piece.GetComponent<Piece>().side = piecePlayer;
    piece.GetComponent<Piece>().x = pieceX;
    piece.GetComponent<Piece>().y = pieceY;
    if (piecePlayer == true)
    {
        piece.transform.Rotate(0f, 0f, 0f);
        piece.transform.position = new Vector3(pieceX * 1.2f, 0.2f, pieceY * 1.2f);
    }
    else
    {
        piece.transform.Rotate(180f, 0f, 0f);
        piece.transform.position = new Vector3(pieceX * 1.2f, 0.4f, pieceY * 1.2f);
    }

    // Add the piece to the pieces array
    pieces[pieceX, pieceY] = piece;
}
```

Listaus 2. AddPiece-metodi.

Luo pelinappulan annettuihin koordinaatteihin vuorossa olevan pelaajan värissä sekä näkyvälle pelilaudalle että pelitilannetta ylläpitävään arrayhin.

```
public void AddGhostPiece(int pieceX, int pieceY, bool left, bool right, bool down, bool up, bool
downleft, bool downright, bool upleft, bool upright)
{
    // Instantiate the piece and add variables to it
    GameObject piece = Instantiate(ghostPiecePrefab);
    piece.transform.position = new Vector3(pieceX * 1.2f, 0.2f, pieceY * 1.2f);
    piece.GetComponent<Ghost>().x = pieceX;
    piece.GetComponent<Ghost>().y = pieceY;
    piece.GetComponent<Ghost>().SetDirections(left, right, down, up, downleft, downright, upleft,
upright);

    // Add the piece to the pieces array
    pieces[pieceX, pieceY] = piece;
}
```

Listaus 3. AddGhostPiece-metodi.

Luo mahdollista siirtoa indikoivan aavenappulan annettuihin koordinaatteihin sekä näkyvälle pelilaudalle että pelitilannetta ylläpitävään arrayhin.

```
public void FlipPiece(int pieceX, int pieceY)
{
    // Find the piece and change its side
    GameObject piece = pieces[pieceX, pieceY];
    if (piece.GetComponent<Piece>().side == false)
    {
        piece.GetComponent<Piece>().side = true;
        piece.transform.Rotate(180f, 0f, 0f);
        piece.transform.position = new Vector3(pieceX * 1.2f, 0.2f, pieceY * 1.2f);
    }
    else
    {
        piece.GetComponent<Piece>().side = false;
        piece.transform.Rotate(180f, 0f, 0f);
        piece.transform.position = new Vector3(pieceX * 1.2f, 0.4f, pieceY * 1.2f);
    }

    // Return the pieces to the pieces array
    pieces[pieceX, pieceY] = piece;
}
```

Listaus 4. FlipPiece-metodi.

Kääntää annetuissa koordinaateissa olevan pelinappulan vuorossa olevan pelaajan väriseksi sekä näkyvällä pelilaudalla että pelitilannetta ylläpitävään arrayhin.

```
public void FlipPieces(bool player, int x, int y, bool goesLeft, bool goesRight, bool goesDown,
bool goesUp, bool goesDownLeft, bool goesDownRight, bool goesUpLeft, bool goesUpRight)
{
    // Pieces on the left
    if (goesLeft == true)
    {
        for (int i = x - 1; i > 1; i--)
        {
            if (pieces[i,y].GetComponent<Piece>().side != player)
            {
                FlipPiece(i, y);
            }
            else
            {
                break;
            }
        }
    }
    // Pieces on the right
    if (goesRight == true)
    {
        for (int i = x + 1; i < 8; i++)
        {
            if (pieces[i, y].GetComponent<Piece>().side != player)
            {
                FlipPiece(i, y);
            }
            else
            {
                break;
            }
        }
    }
    // Pieces down
    if (goesDown == true)
    {
        for (int i = y - 1; i > 0; i--)
        {
            if (pieces[x, i].GetComponent<Piece>().side != player)
            {
                FlipPiece(x, i);
            }
            else
            {
                break;
            }
        }
    }
    // Pieces up
    if (goesUp == true)
    {
        for (int i = y + 1; i > 0; i++)
```

```
    {
        if (pieces[x, i].GetComponent<Piece>().side != player)
        {
            FlipPiece(x, i);
        }
        else
        {
            break;
        }
    }
}
// Pieces down-left
if (goesDownLeft == true)
{
    for (int i = x - 1, j = y - 1; i > 0 && j > 0; i--, j--)
    {
        if (pieces[i, j].GetComponent<Piece>().side != player)
        {
            FlipPiece(i, j);
        }
        else
        {
            break;
        }
    }
}
// Pieces down-right
if (goesDownRight == true)
{
    for (int i = x + 1, j = y - 1; i < 8 && j > 0; i++, j--)
    {
        if (pieces[i, j].GetComponent<Piece>().side != player)
        {
            FlipPiece(i, j);
        }
        else
        {
            break;
        }
    }
}
// Pieces up-left
if (goesUpLeft == true)
{
    for (int i = x - 1, j = y + 1; i > 0 && j < 8; i--, j++)
    {
        if (pieces[i, j].GetComponent<Piece>().side != player)
        {
            FlipPiece(i, j);
        }
        else
        {
```

```
                break;
            }
        }
    }
    // Pieces up-right
    if (goesUpRight == true)
    {
        for (int i = x + 1, j = y + 1; i < 8 && j < 8; i++, j++)
        {
            if (pieces[i, j].GetComponent<Piece>().side != player)
            {
                FlipPiece(i, j);
            }
            else
            {
                break;
            }
        }
    }
}
```

Listaus 5. FlipPieces-metodi.

Kääntää pelinappuloita annettuihin koordinaatteihin lisätyn pelinappulan mukaisesti käyttäen FlipPiece-metodia.

```
public void SetupBoard()
{
    AddPiece(true, 3, 3);
    AddPiece(false, 4, 3);
    AddPiece(false, 3, 4);
    AddPiece(true, 4, 4);
}
```

Listaus 6. SetupBoard-metodi.

Virittää pelilaudan aloitustilanteeseen lisäämällä neljä pelinappulaa niille varattuihin ruutuihin AddPiece-metodia käyttäen.

```
public bool SetupGhostPieces(bool player)
{
    bool movesFound = false;

    for (int x = 0; x < 8; x++)
    {
        for (int y = 0; y < 8; y++)
        {
            if (pieces[x, y] == null)
            {
                bool legalTile = false;
                bool goesLeft = false;
                bool goesRight = false;
                bool goesDown = false;
                bool goesUp = false;
                bool goesDownLeft = false;
                bool goesDownRight = false;
                bool goesUpLeft = false;
                bool goesUpRight = false;

                // Check left
                if (x > 1)
                {
                    if (pieces[x - 1, y] != null && pieces[x - 1, y].tag == "Piece" && pieces[x - 1,
y].GetComponent<Piece>().side != player) // Check if the next piece exists and is opposite color
                    {
                        for (int i = x - 2; i >= 0; i--)
                        {
                            if (pieces[i, y] == null || pieces[i, y].tag != "Piece") // No more pieces
                            {
                                break;
                            }
                            else if (pieces[i, y].GetComponent<Piece>().side == player) // Found a player
piece! Declare a legal tile and end the search
                            {
                                legalTile = true;
                                goesLeft = true;
                                break;
                            }
                        }
                    }
                }
                // Check right
                if (x < 7)
                {
                    if (pieces[x + 1, y] != null && pieces[x + 1, y].tag == "Piece" && pieces[x + 1,
y].GetComponent<Piece>().side != player) // Check if the next piece exists and is opposite color
                    {
                        for (int i = x + 2; i < 8; i++)
                        {
```

```
                    if (pieces[i, y] == null || pieces[i, y].tag != "Piece") // No more pieces
                    {
                        break;
                    }
                    else if (pieces[i, y].GetComponent<Piece>().side == player) // Found a player
piece! Declare a legal tile and end the search
                    {
                        legalTile = true;
                        goesRight = true;
                        break;
                    }
                }
            }
        }
        // Check down
        if (y > 1)
        {
            if (pieces[x, y - 1] != null && pieces[x, y - 1].tag == "Piece" && pieces[x, y -
1].GetComponent<Piece>().side != player) // Check if the next piece exists and is opposite color
            {
                for (int i = y - 2; i >= 0; i--)
                {
                    if (pieces[x, i] == null || pieces[x, i].tag != "Piece") // No more pieces
                    {
                        break;
                    }
                    else if (pieces[x, i].GetComponent<Piece>().side == player) // Found a player
piece! Declare a legal tile and end the search
                    {
                        legalTile = true;
                        goesDown = true;
                        break;
                    }
                }
            }
        }
        // Check up
        if (y < 7)
        {
            if (pieces[x, y + 1] != null && pieces[x, y + 1].tag == "Piece" && pieces[x, y +
1].GetComponent<Piece>().side != player) // Check if the next piece exists and is opposite color
            {
                for (int i = y + 2; i < 8; i++)
                {
                    if (pieces[x, i] == null || pieces[x, i].tag != "Piece") // No more pieces
                    {
                        break;
                    }
                    else if (pieces[x, i].GetComponent<Piece>().side == player) // Found a player
piece! Declare a legal tile and end the search
                    {
                        legalTile = true;
```

```
                        goesUp = true;
                        break;
                    }
                }
            }
        }
        // Check Down-Left
        if (x > 1 && y > 1)
        {
            if (pieces[x - 1, y - 1] != null && pieces[x - 1, y - 1].tag == "Piece" && pieces[x -1, y -
1].GetComponent<Piece>().side != player) // Check if the next piece exists and is opposite color
            {
                for (int i = x - 2, j = y - 2; i >= 0 && j >= 0; i--, j--)
                {
                    if (pieces[i, j] == null || pieces[i, j].tag != "Piece") // No more pieces
                    {
                        break;
                    }
                    else if (pieces[i, j].GetComponent<Piece>().side == player) // Found a player
piece! Declare a legal tile and end the search
                    {
                        legalTile = true;
                        goesDownLeft = true;
                        break;
                    }
                }
            }
        }
        // Check Down-Right
        if (x < 7 && y > 1)
        {
            if (pieces[x + 1, y - 1] != null && pieces[x + 1, y - 1].tag == "Piece" && pieces[x + 1,
y - 1].GetComponent<Piece>().side != player) // Check if the next piece exists and is opposite
color
            {
                for (int i = x + 2, j = y - 2; i < 8 && j >= 0; i++, j--)
                {
                    if (pieces[i, j] == null || pieces[i, j].tag != "Piece") // No more pieces
                    {
                        break;
                    }
                    else if (pieces[i, j].GetComponent<Piece>().side == player) // Found a player
piece! Declare a legal tile and end the search
                    {
                        legalTile = true;
                        goesDownRight = true;
                        break;
                    }
                }
            }
        }
        // Check Up-Left
```

```
            if (x > 1 && y < 7)
            {
                if (pieces[x - 1, y + 1] != null && pieces[x - 1, y + 1].tag == "Piece" && pieces[x - 1, y
+ 1].GetComponent<Piece>().side != player) // Check if the next piece exists and is opposite
color
                {
                    for (int i = x - 2, j = y + 2; i >= 0 && j < 8; i--, j++)
                    {
                        if (pieces[i, j] == null || pieces[i, j].tag != "Piece") // No more pieces
                        {
                            break;
                        }
                        else if (pieces[i, j].GetComponent<Piece>().side == player) // Found a player
piece! Declare a legal tile and end the search
                        {
                            legalTile = true;
                            goesUpLeft = true;
                            break;
                        }
                    }
                }
            }
            // Check Up-Right
            if (x < 7 && y < 7)
            {
                if (pieces[x + 1, y + 1] != null && pieces[x + 1, y + 1].tag == "Piece" && pieces[x + 1,
y + 1].GetComponent<Piece>().side != player) // Check if the next piece exists and is opposite
color
                {
                    for (int i = x + 2, j = y + 2; i < 8 && j < 8; i++, j++)
                    {
                        if (pieces[i, j] == null || pieces[i, j].tag != "Piece") // No more pieces
                        {
                            break;
                        }
                        else if (pieces[i, j].GetComponent<Piece>().side == player) // Found a player
piece! Declare a legal tile and end the search
                        {
                            legalTile = true;
                            goesUpRight = true;
                            break;
                        }
                    }
                }
            }
            // Add a ghost piece if tile is legal
            if (legalTile == true)
            {
                movesFound = true;
                AddGhostPiece(x, y, goesLeft, goesRight, goesDown, goesUp, goesDownLeft,
goesDownRight, goesUpLeft, goesUpRight);
            }
```

```
            }
        }
    }
    return movesFound;
}
```

Listaus 7. SetupGhostPieces-metodi.

Käy läpi jokaisen ruudun, tarkistaen onko se vapaa ja voiko vuorossa oleva pelaaja laittaa siihen nappulansa, lisäten siihen aavenappulan AddGhostPiece-metodia käyttäen mikäli näin on. Lopuksi palauttaa toden mikäli mahdollisia siirtoja on löytynyt.

```
public void EmptyBoard()
{
    foreach (GameObject p in pieces)
    {
        if (p == null)
        {
            continue;
        }
        else if (p.tag == "Piece")
        {
            pieces[p.GetComponent<Piece>().x, p.GetComponent<Piece>().y] = null;
            Destroy(p);
        }
    }
}
```

Listaus 8. EmptyBoard-metodi.

Tyhjentää sekä näkyvän pelilaudan että pelitilannetta ylläpitävän arrayn pelinappuloista.

```
public void RemoveGhostPieces()
{
    foreach (GameObject p in pieces)
    {
        if (p == null)
        {
            continue;
        }
        else if (p.tag == "Ghost")
        {
            pieces[p.GetComponent<Ghost>().x, p.GetComponent<Ghost>().y] = null;
            Destroy(p);
        }
    }
}
```

Listaus 9. RemoveGhostPieces-metodi.

Poistaa aavenappulat sekä näkyvältä pelilaudalta että pelitillannetta ylläpitävästä arraysta.

```
public int CountPieces(bool side)
{
    int count = 0;

    foreach (GameObject p in pieces)
    {
        if (p == null)
        {
            continue;
        }
        else if (p.GetComponent<Piece>().side == side)
        {
            count++;
        }
        else if (p.GetComponent<Piece>().side != side)
        {
            continue;
        }
    }

    return count;
}
```

Listaus 10. CountPieces-metodi.

Laskee laudalla ja annetun pelaajan värissä olevat pelinappulat.

7.3.2 GameManager-luokan metodit

```
void Start()
{
   whitePlayer = true;
   blackPlayer = true;
   gameState = GameState.Menu;
}
```

Listaus 11. Start-metodi.

Merkitsee molemmat pelaajat ihmispelaajiksi ja asettaa ohjelman tilan alkuvalikkoon.

```
void Update()
{
   switch (gameState)
   {
      case GameState.Menu:
         break;

      case GameState.Game:
         // If human player, wait for move
         if ((currentPlayer == false && blackPlayer == true) || (currentPlayer == true &&
whitePlayer == true))
         {
            if (Input.GetButtonDown("Fire1"))
            {
               Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

               RaycastHit hit;

               if (Physics.Raycast(ray, out hit))
               {
                  if (hit.collider.tag == "Ghost")
                  {
                     // Save the ghost piece's variables
                     int x = hit.collider.GetComponent<Ghost>().x;
                     int y = hit.collider.GetComponent<Ghost>().y;
                     bool goesLeft = hit.collider.GetComponent<Ghost>().goesLeft;
                     bool goesRight = hit.collider.GetComponent<Ghost>().goesRight;
                     bool goesDown = hit.collider.GetComponent<Ghost>().goesDown;
                     bool goesUp = hit.collider.GetComponent<Ghost>().goesUp;
                     bool goesDownLeft = hit.collider.GetComponent<Ghost>().goesDownLeft;
                     bool goesDownRight = hit.collider.GetComponent<Ghost>().goesDownRight;
                     bool goesUpLeft = hit.collider.GetComponent<Ghost>().goesUpLeft;
                     bool goesUpRight = hit.collider.GetComponent<Ghost>().goesUpRight;
```

```
                    // Remove ghost pieces and add the new piece
                    board.GetComponent<Board>().RemoveGhostPieces();
                    board.GetComponent<Board>().AddPiece(currentPlayer, x, y);
                    board.GetComponent<Board>().FlipPieces(currentPlayer, x, y, goesLeft,
goesRight, goesDown, goesUp, goesDownLeft, goesDownRight, goesUpLeft, goesUpRight);

                    // Remove turn skipped text
                    overlay.GetComponent<Overlay>().DisplayBottomText("");

                    // Change the player
                    if (currentPlayer == false)
                    {
                        currentPlayer = true;
                    }
                    else
                    {
                        currentPlayer = false;
                    }

                    // Setup ghost pieces. If there are no legal moves, skip turn.
                    if (board.GetComponent<Board>().SetupGhostPieces(currentPlayer) == false)
                    {
                        // Change the player
                        if (currentPlayer == false)
                        {
                            overlay.GetComponent<Overlay>().DisplayBottomText("BLACK TURN
SKIPPED");
                            currentPlayer = true;
                        }
                        else
                        {
                            overlay.GetComponent<Overlay>().DisplayBottomText("WHITE TURN
SKIPPED");
                            currentPlayer = false;
                        }

                        // Setup ghost pieces again. If there are still no legal moves, end game.
                        if (board.GetComponent<Board>().SetupGhostPieces(currentPlayer) ==
false)
                        {
                            EndGame();
                            gameState = GameState.EndGame;
                        }
                    }
                }
            }
        }
    }

    // If AI player, decide move
```

```
        if ((currentPlayer == false && blackPlayer == false) || (currentPlayer == true &&
whitePlayer == false))
        {
            DecideMove();

            // Remove turn skipped text
            overlay.GetComponent<Overlay>().DisplayBottomText("");

            // Change the player
            if (currentPlayer == false)
            {
                currentPlayer = true;
            }
            else
            {
                currentPlayer = false;
            }

            // Setup ghost pieces. If there are no legal moves, skip turn.
            if (board.GetComponent<Board>().SetupGhostPieces(currentPlayer) == false)
            {
                // Change the player
                if (currentPlayer == false)
                {
                    overlay.GetComponent<Overlay>().DisplayBottomText("BLACK TURN
SKIPPED");
                    currentPlayer = true;
                }
                else
                {
                    overlay.GetComponent<Overlay>().DisplayBottomText("WHITE TURN
SKIPPED");
                    currentPlayer = false;
                }

                // Setup ghost pieces again. If there are still no legal moves, end game.
                if (board.GetComponent<Board>().SetupGhostPieces(currentPlayer) == false)
                {
                    EndGame();
                    gameState = GameState.EndGame;
                }
            }
        }
        break;

    case GameState.EndGame:
        if (Input.GetButtonDown("Fire1"))
        {
            ReturnToMenu();
            gameState = GameState.Menu;
        }
        break;
```

```
    }
}
```

Listaus 12. Update-metodi.

Kutsutaan jokaisella framella. Toiminta määräytyy ohjelman tilan mukaan. Tilan ollessa alkuvalikossa ei tee mitään.

Tilan ollessa pelissä ja mikäli on ihmispelaajan vuoro odotetaan pelaavan tekevän siirtonsa klikkaamalla jotain aavenappulaa. Tällöin kyseisen nappulan tiedot tallennetaan muistiin, poistetaan aavenappulat RemoveGhostPieces-metodilla, lisätään uusi nappula AddPiece-metodilla ja käännetään toisen pelaajan nappuloita FlipPieces-metodilla. Tämän jälkeen poistetaan mahdollinen ilmoitus ohitetusta vuorosta DisplayBottomText-metodilla, vaihdetaan vuorossa olevaa pelaajaa ja etsitään tämän mahdolliset siirrot SetupGhostPieces-metodilla. Mikäli mahdollisia siirtoja ei ole, merkitään vuoro ohitetuksi, lisätään tästä ilmoittava teksti ruudulle DisplayBottomText-metodilla ja vaihdetaan vuorossa olevaa pelaajaa uudelleen. Mikäli mahdollisia siirtoja ei vieläkään löydetä, päätetään peli EndGame-metodilla ja vaihdetaan ohjelman tilaksi pelin lopetus.

Tekoälypelaajan vuorolla toimitaan muuten samoin kuin edellä, mutta klikkauksen odottamisen sijaan siirto päätetään käyttämällä DecideMove-metodia.

Tilan ollessa pelin lopetuksessa odotetaan klikkausta, jolloin kutsutaan ReturnToMenu-metodia ja vaihdetaan ohjelman tilaksi alkuvalikko.

```
void DecideMove()
{
    // Check each ghost piece location for each value starting from 1 until one is found
    for (int v = 1; v < 6; v++)
    {
        for (int x = 0; x < 8; x++)
        {
            for (int y = 0; y < 8; y++)
            {
                if (board.GetComponent<Board>().pieces[x, y] == null)
```

```
            {
                continue;
            }
            else if (board.GetComponent<Board>().pieces[x, y].tag == "Ghost")
            {
                if (tiles[x, y] == v)
                {
                    // Save the ghost piece's variables
                    int xx = board.GetComponent<Board>().pieces[x, y].GetComponent<Ghost>().x;
                    int yy = board.GetComponent<Board>().pieces[x, y].GetComponent<Ghost>().y;
                    bool goesLeft = board.GetComponent<Board>().pieces[x,
y].GetComponent<Ghost>().goesLeft;
                    bool goesRight = board.GetComponent<Board>().pieces[x,
y].GetComponent<Ghost>().goesRight;
                    bool goesDown = board.GetComponent<Board>().pieces[x,
y].GetComponent<Ghost>().goesDown;
                    bool goesUp = board.GetComponent<Board>().pieces[x,
y].GetComponent<Ghost>().goesUp;
                    bool goesDownLeft = board.GetComponent<Board>().pieces[x,
y].GetComponent<Ghost>().goesDownLeft;
                    bool goesDownRight = board.GetComponent<Board>().pieces[x,
y].GetComponent<Ghost>().goesDownRight;
                    bool goesUpLeft = board.GetComponent<Board>().pieces[x,
y].GetComponent<Ghost>().goesUpLeft;
                    bool goesUpRight = board.GetComponent<Board>().pieces[x,
y].GetComponent<Ghost>().goesUpRight;

                    // Remove ghost pieces and add the new piece
                    board.GetComponent<Board>().RemoveGhostPieces();
                    board.GetComponent<Board>().AddPiece(currentPlayer, xx, yy);
                    board.GetComponent<Board>().FlipPieces(currentPlayer, xx, yy, goesLeft,
goesRight, goesDown, goesUp, goesDownLeft, goesDownRight, goesUpLeft, goesUpRight);
                    return;
                }
            }
        }
    }
  }
}
```

Listaus 13. DecideMove-metodi.

Tekoälyn ohjelmakoodi. Tarkistaa jokaisen ruudun, joihin on mahdollista asettaa pelinappula ja joka siten on aiemmin merkattu aavenappulalla. Löydettyään korkeimman arvoisen ruudun, koodi tallentaa muistiin tämän ruudun tiedot, poistaa aavenappulat (RemoveGhostPieces-metodi), lisää pelatun nappulan laudalle (AddPiece-metodi) ja kääntää muita nappuloita muistiin tallennettujen tietojen mukaisesti (FlipPieces-metodi.)

```
public void StartGame()
{
    board.GetComponent<Board>().EmptyBoard();
    currentPlayer = false;
    board.GetComponent<Board>().SetupBoard();
    board.GetComponent<Board>().SetupGhostPieces(currentPlayer);
    gameState = GameState.Game;
}
```

Listaus 14. StartGame-metodi.

Aloittaa pelin tyhjentämällä laudan EmptyBoard-metodilla, merkitsemällä vuorossa olevan pelaajan mustaksi, virittämällä laudan alkuasetelmaan SetupBoard-metodilla ja asettamalla vuorossa olevan pelaajan mahdollisia siirtoja indikoivat aavenappulat laudalle SetupGhostPieces-metodilla. Lopuksi asettaa ohjelman pelitilaan.

```
void EndGame()
{
    int blackScore;
    int whiteScore;

    // Count black pieces
    blackScore = board.GetComponent<Board>().CountPieces(false);
    overlay.GetComponent<Overlay>().DisplayScore(false, blackScore);

    // Count white pieces
    whiteScore = board.GetComponent<Board>().CountPieces(true);
    overlay.GetComponent<Overlay>().DisplayScore(true, whiteScore);

    // Declare Winner
    if (blackScore > whiteScore)
    {
        overlay.GetComponent<Overlay>().DisplayBottomText("BLACK PLAYER WINS");
    }
    else if (whiteScore > blackScore)
    {
        overlay.GetComponent<Overlay>().DisplayBottomText("WHITE PLAYER WINS");
    }
    else
    {
        overlay.GetComponent<Overlay>().DisplayBottomText("DRAW");
    }
}
```

Listaus 15. EndGame-metodi.

Laskee molempien pelaajien väreissä olevat nappulat CountPieces-metodilla ja kirjoittaa ne ruudulle DisplayScore-metodilla. Tämän jälkeen ilmoittaa voittajan DisplayBottomText-metodilla.

```
void ReturnToMenu()
{
   // Wipe the UI
   overlay.GetComponent<Overlay>().DisplayBottomText("");
   overlay.GetComponent<Overlay>().HideScores();

   // Empty the board
   board.GetComponent<Board>().EmptyBoard();

   // Turn the menu back on
   menu.SetActive(true);
}
```

Listaus 16. ReturnToMenu-metodi,

Tyhjentää ruudulla näkyvät tekstit DisplayBottomText- ja HideScores-metodeja käyttämällä, tyhjentää laudan EmptyBoard-metodilla ja palauttaa alkuvalikon takaisin näkyviin.

7.3.3 Menu-luokan metodit

```
public void PressStartButton()
{
   this.GetComponentInParent<GameManager>().StartGame();
}
```

Listaus 17. PressStartButton-metodi.

Aloittaa pelin käyttämällä StartGame-metodia.

```
public void PressBlackButton()
{
   // Switch black player to AI if human and vice versa
   if (this.GetComponentInParent<GameManager>().blackPlayer == true)
   {
      this.GetComponentInParent<GameManager>().blackPlayer = false;
      blackButton.GetComponentInChildren<UnityEngine.UI.Text>().text = "BLACK
PLAYER\nAI";

      // If the other player is also AI, make it human
      if (this.GetComponentInParent<GameManager>().whitePlayer == false)
      {
         this.GetComponentInParent<GameManager>().whitePlayer = true;
         whiteButton.GetComponentInChildren<UnityEngine.UI.Text>().text = "WHITE
PLAYER\nHUMAN";
      }
   }
   else
   {
      this.GetComponentInParent<GameManager>().blackPlayer = true;
      blackButton.GetComponentInChildren<UnityEngine.UI.Text>().text = "BLACK
PLAYER\nHUMAN";
   }
}
```

Listaus 18. PressBlackButton-metodi.

Asettaa mustan pelaajan tekoälypelaajasta ihmispelaajaksi tai takaisin. Mikäli molemmat pelaajat ovat tämän jälkeen tekoälypelaajia, asettaa valkoisen pelaajan ihmispelaajaksi, ettei peli joudu pelaamaan itseään vastaan. Päivittää lisäksi valikon painikkeiden tekstit.

```
public void PressWhiteButton()
{
   // Switch white player to AI if human and vice versa
   if (this.GetComponentInParent<GameManager>().whitePlayer == true)
   {
      this.GetComponentInParent<GameManager>().whitePlayer = false;
      whiteButton.GetComponentInChildren<UnityEngine.UI.Text>().text = "WHITE
PLAYER\nAI";

      // If the other player is also AI, make it human
      if (this.GetComponentInParent<GameManager>().blackPlayer == false)
      {
         this.GetComponentInParent<GameManager>().blackPlayer = true;
         blackButton.GetComponentInChildren<UnityEngine.UI.Text>().text = "BLACK
PLAYER\nHUMAN";
      }
   }
```

```
    else
    {
        this.GetComponentInParent<GameManager>().whitePlayer = true;
        whiteButton.GetComponentInChildren<UnityEngine.UI.Text>().text = "WHITE
PLAYER\nHUMAN";
    }
}
```

Listaus 19- PressWhiteButton-metodi.

Asettaa valkoisen pelaajan tekoälypelaajasta ihmispelaajaksi tai takaisin. Mikäli molemmat pelaajat ovat tämän jälkeen tekoälypelaajia, asettaa mustan pelaajan ihmispelaajaksi, ettei peli joudu pelaamaan itseään vastaan. Päivittää lisäksi valikon painikkeiden tekstit.

```
public void PressQuitButton()
{
    Application.Quit();
}
```

Listaus 20. PressQuitButton-metodi.

Sulkee ohjelman.

7.3.4 Overlay-luokan metodit

```
public void DisplayScore(bool side, int count)
{
    if (side == true)
    {
        whiteScore.GetComponent<UnityEngine.UI.Text>().text = count.ToString();
    }
    else
    {
        blackScore.GetComponent<UnityEngine.UI.Text>().text = count.ToString();
    }
}
```

Listaus 21. DisplayScore-metodi.

Kirjoittaa annetun pelaajan pistemäärän ruudulle.

```
public void DisplayBottomText(string result)
{
    bottom.GetComponent<UnityEngine.UI.Text>().text = result;
}
```

Listaus 22. DisplayBottomText-metodi.

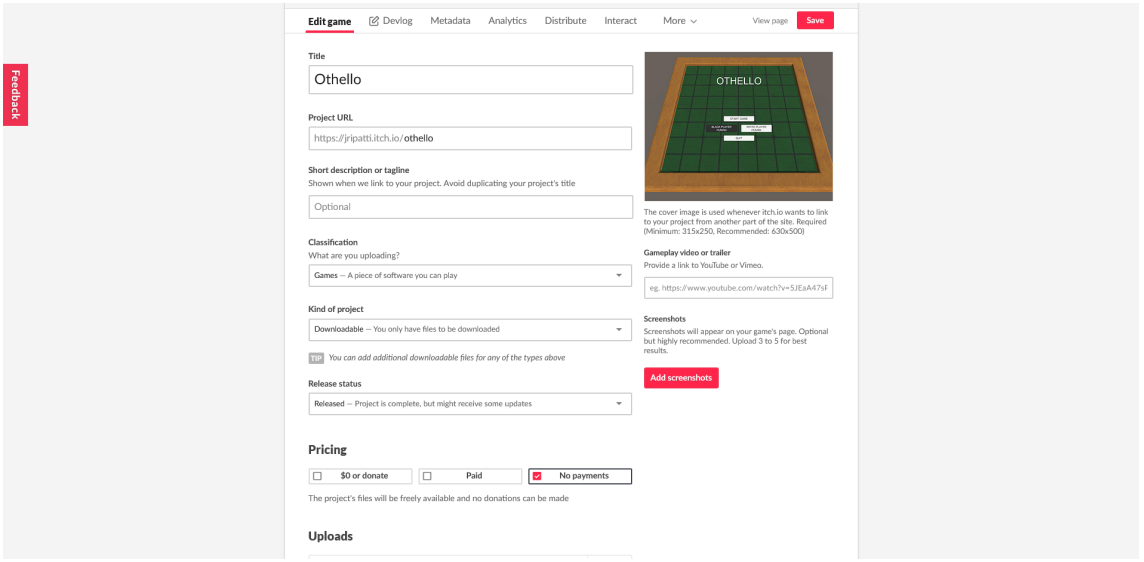
Kirjoittaa ruudun alalaitaan pelin lopputuloksen tai ilmoituksen ohitetusta vuorosta.

```
public void HideScores()
{
    blackScore.GetComponent<UnityEngine.UI.Text>().text = "";
    whiteScore.GetComponent<UnityEngine.UI.Text>().text = "";
}
```

Listaus 23. HideScores-metodi.

Piilottaa pistenäkymän.

7.4 Julkaisu



Kuva 16. Itch.io:n julkaisukäyttöliittymä.

Pelin julkaiseminen Itch.io-verkkosivulla vaatii ainoastaan rekisteröitymisen. Tämän jälkeen riittää, että täyttää pelin tiedot (Kuvaus, alusta, genre, avainsanat, kieli jne.) lomakkeeseen ja lähettää tiedostot Itch.io:n palvelimelle. Mitään erillistä hyväksyntää peli ei siis vaadi julkaisuvaiheessa. Tilaa pelin tiedostoille on vakiona 1 GB, mitä voi nostaa ottamalla yhteyttä ylläpitoon. Peliään voi jakaa ilmaiseksi tai maksua vastaan, kuten aiemmin on mainittu.

Pelilleen voi tehdä sivun Itch.ioon myös ennen kuin se on julkaisuvalmis. Kehitystyön dokumentointia ja muita ilmoituksia varten Itch:io tarjoaa kehityslogin.

8 Loppuyhteenveto

Toiminnallisen opinnäytetyöni aiheena oli Othello-lautapelin kääntäminen Windows-käyttöjärjestelmälle ja julkaiseminen internetissä. Pelimoottorina käytössäni oli Unity, grafiikoiden luomisessa Blender ja versionhallinnassa Plastic SCM. Valitsin nämä pääasiassa niiden maksuttomuuden vuoksi, mutta myös siksi että minulla on niiden käytöstä aiempaa kokemusta. Pelin julkaisin Itch.io-verkkosivulla, koska totesin sen olevan voittoa tavoittelemattomalle pelille sopivampi julkaisualusta kuin Steam tai Epic Game Store.

8.1 Tarkastelu

Kokonaisuutena olen tyytyväinen valmiiseen tuotokseen. Pelistä tuli suunnilleen sellainen kuin oli tarkoituskin. Ainoastaan visuaalinen ilme jäi suunnitellusta, lähinnä puuttuvien animaatioiden vuoksi. Myös aikataulu viivästyi alkuperäisestä, mutta se oli ennalta tiedostettu riski. Toisaalta tekoäly toimi käytännössä paremmin kuin mitä suunnitteluvaiheessa sen oletin toimivan. Koodista olisi varmasti voinut tehdä parempaa, mutta ainakin se ajaa asiansa.

Oppimisen kannalta projekti tarjosi paljon. Minulla ei ollut aikaisempaa kokemusta tämän mittakaavan pelien suunnittelemisesta saati sitten alusta lähtien ohjelmoinnista. Aiempi kokemukseni Unityn käytöstä pohjautui puhtaasti muiden luoman koodin jatkokehittämiseen ja hyödyntämiseen. Tästä huolimatta en kohdannut suurempia odottamattomia ongelmia. Suunnitelma kääntyi valmiiksi peliksi sujuvasti ja eri työkalut toimivat hyvin yhdessä. Koodin kirjoittamisessa vastaan tuli toki ongelmatilanteita, kokemattomuudestani johtuen, mutta sekä C#:n että Unityn käyttöön oli internetissä runsaasti apua tarjolla.

8.2 Kehittämisideat

Jos peliä haluaisi kehittää eteenpäin, selkeimmät osa-alueet ovat tekoälyn ja visuaalisen ulkoasun parantaminen. Nykyisellään tekoäly on hyvin yksinkertainen eikä tarjoa kokeneelle pelaajalle kunnollista vastusta, mitä sen ei toki ole tarkoituskaan. Tekoälyä voisi kehittää ohjelmoimalla sen miettimään useamman siirron eteenpäin. Paremman tekoälyn myötä peliin voisi myös lisätä useampia vaikeustasoja.

Visuaalista puolta taasen voisi kehittää lisäämällä peliin alkuperäisestä suunnitelmasta puuttumaan jääneet animaatiot. Nykyisellään peli ei näytä nappuloiden lisäämistä tai kääntymistä mitenkään, mikä tekee siirtojen lukemisesta tarpeettoman hankalaa. Grafiikoiden parantamisella olisi siis vaikutusta myös pelattavuuden kannalta.

Lähteet

Blender.org. 2021. About. https://www.blender.org/about/. 21.5.2021.

Codice Software. 2021a. Features. https://www.plasticscm.com/features. 14.5.2021.

Codice Software. 2021b. Licensing and Pricing. https://www.plasticscm.com/pricing. 14.5.2021.

Dean, B. 2021. Epic Games Store User Statistics For 2021. Backlinko. https://backlinko.com/epic-games-users. 21.5.2021.

Epic Games, Inc. 2021. About. https://www.epicgames.com/store/en-US/about. 21.5.2021.

Itch corp. 2021a. Top games. https://itch.io/games. 21.5.2021.

Itch corp. 2021b. Creator FAQ. https://itch.io/docs/creators/faq. 21.5.2021.

Itch corp. 2021c. Accepting Payments and Getting Paid. https://itch.io/docs/creators/payments. 21.5.2021.

Law, K. 2010. Origins of 8 classic board games. CNN. http://edition.cnn.com/2010/LIVING/12/26/mf.classic.board.games/index.html. 14.5.2021.

Steamworks. 2021a. Onboarding. https://partner.steamgames.com/doc/gettingstarted/onboarding. 21.5.2021.

Steamworks. 2021b. Reporting and Payments FAQ. https://partner.steamgames.com/doc/finance/payments_salesreporting/faq. 21.5.2021.

Steamworks. 2021c. Taxes FAQ. https://partner.steamgames.com/doc/finance/taxfaq. 21.5.2021.

Trello. 2021. Epic Games Store Roadmap. https://trello.com/b/GXLc34hk/epic-games-store-roadmap. 21.5.2021.

Unity Technologies. 2021a. Unity Platform. https://unity.com/products/unity-platform. 14.5.2021.

Unity Technologies. 2021b. Build once, deploy anywhere. https://unity.com/features/multiplatform. 14.5.2021.

Valve Corporation. 2021. Steam Search. https://store.steampowered.com/search/?category1=998. 21.5.2021.

World Othello Federation. 2021a. About the World Othello Federation. https://www.worldothello.org/about. 14.5.2021.

World Othello Federation. 2021b. Official Rules For The Game Othello. https://www.worldothello.org/about/about-othello/othello-rules/official-rules/english. 21.5.2021.

Xsolla. 2020. How to Get Published on the Epic Games Store. https://xsolla.com/blog/publishing-suite/1926/how-to-get-published-on-the-epic-games-store. 21.5.2021.

Zuckerman. A. 2020. 75 Steam Statistics: 2020/2021 Fact, Market Share & Data Analysis. CompareCamp. https://comparecamp.com/steam-statistics/. 21.5.2021.

Ikonen, J. & Lipponen, L. 2016. Lautapelistä digitaaliseksi peliksi Case: Muistatko vielä…. Laurea-ammattikorkeakoulu. https://www.theseus.fi/handle/10024/105280. 31.5.2021.

Hämäläinen, N. 2019. Riichi-mahjong-tekoäly. Metropolia-ammattikorkeakoulu. https://www.theseus.fi/handle/10024/172646. 31.5.2021.

Kuusela, J. 2017. Tietokoneshakin ohjelmointi JavaScript-kielellä. Haaga-Helia ammattikorkeakoulu. https://www.theseus.fi/handle/10024/137637. 31.5.2021.

Pirskanen, J. 2014. Lautapelin kääntäminen mobiilialustalle ja konversioiden käytettävyys. Karelia-ammattikorkeakoulu. https://www.theseus.fi/handle/10024/71321. 31.5.2021.

Salonen, P. 2019. Tutustumiskorttipeli mobiilimuotoon. Metropolia-ammattikorkeakoulu. https://www.theseus.fi/handle/10024/168425. 31.5.2021.

Vilppola, P. 2015. Mobile game prototyping: Programmer’s point of view. Jyväskylän ammattikorkeakoulu. https://www.theseus.fi/handle/10024/92813. 31.5.2021.