SAVONIA
ammattikorkeakoulu

PROJEKTIN LOPPURAPORTTI - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

RISTINOLLA-PELLI

Klassinen peli

TEKIJÄ/T Valerii Reutskyi

26.12.2023

SISÄLTÖ

1 JOHDANTO

Tämä raportti on laadittu työn raportointia varten. Löydät täältä käyttöoppaan, perinteisen ristinollapelin toteutusprosessit ja voit tutustua myös ohjelmaan liittyviin syvällisempiin prosesseihin. Tässä raportissa on lisätty dokumentaatio siitä alustasta, jolla projekti luotiin, sekä itse alustan luomisen historia.

Käyttöoppaassa löydät kaikki tämän ohjelman hallinnan hienoudet sekä tutustut liitännäispelin sääntöihin. Tässä luvussa on linkkejä muihin lähteisiin, joista voit syventyä tarkemmin tämän pelityypin historian tutkimiseen ja perehtyä tarkemmin ristinollapelin sääntöjen syntymiseen ja lajityypin kehitykseen.

Puhuttaessa .Net MAUI:n ja Visual Studio 2022:n historiasta, joiden avulla tämä projekti toteutettiin, voimme syventyä ohjelmoinnin historiaan ja ymmärtää paremmin prosesseja tässä vaiheessa. Siirtyessäsi luomisprosessiin voit tutustua tämän projektin logiikkaan, tuntea kaikki hienoudet ja ymmärtää, kuinka luoda vastaavia pelejä.

2 HISTORIA VISUAL STUDIO:N JA .NET MAUI:N LUOMISESTA

2.1 Visual Studio ja sen historia

Microsoft Visual Studio (VS) on integroitu kehitysympäristö (IDE), jonka on kehittänyt Microsoft. Visual Studion historia ulottuu useiden vuosikymmenten päähän ja liittyy Microsoftin tarjoamien kehitystyökalujen evoluutioon.

Johdanto Visual Studioon:

1980-luku: 1980-luvun alussa Microsoft julkaisi erilaisia kääntäjiä ja kehitysympäristöjä ohjelmointikielille, kuten C ja C++. Kuitenkin nämä olivat erillisiä tuotteita eivätkä muodostaneet yhtenäistä integroitua ympäristöä.

1) Visual Basic ja Visual C++:

1990-luku: 1990-luvun puolivälissä Microsoft esitteli Visual Basicin ja Visual C++, joista kumpikin tarjosi integroidun kehitysympäristön vastaaville kielille. Nämä tuotteet olivat askel kohti kehityksen integrointia Microsoftissa.

2) Visual Studio 97:

1997: Visual Studio 97:n julkaisu yhdisti kehitystyökalut yhdeksi integroiduksi ympäristöksi. Tämä julkaisu sisälsi Visual Basicin, Visual C++:n ja Visual FoxPro:n. Visual Studio 97 oli ensimmäinen versio, joka kutsuttiin "Visual Studio"ksi.

3) Päivitykset ja uudet versiot:

Поздние 1990-е и 2000-е годы: В последующие годы Microsoft регулярно выпускала новые версии Visual Studio, добавляя поддержку для новых языков программирования, технологий и фреймворков. Visual Studio.NET, выпущенная в 2002 году, была крупным обновлением, представившим новую архитектуру и введя поддержку для платформы .NET.

2.2 Visual Studio -versiot ja niiden kielet

Microsoft Visual Studiolla on lukuisia versioita, ja kullakin uudella versiolla on yleensä parannuksia, uusia ominaisuuksia ja tukea uusimmille teknologioille. Alla on lyhyt yhteenveto joistakin tärkeimmistä Visual Studio -versioista ja niissä tuetuista ohjelmointikielistä:

Visual Studio 6.0 (1998):

Kielet: Visual Basic 6.0, Visual C++ 6.0, Visual J++ 6.0 ja muut.

Visual Studio .NET (2002):

Kielet: .NET-alustan käyttöönotto. Tuki C#:lle, Visual Basic.NET:lle, C++:lle, J#:lle, ASP.NET:lle.

Visual Studio 2005:

Kielet: Tuki C#:lle, Visual Basic.NET:lle, C++:lle, J#:lle, F#:lle.

Visual Studio 2008:

Kielet: Tuki C#:lle, Visual Basic.NET:lle, C++:lle, J#:lle, F#:lle.

Visual Studio 2010:

Kielet: Tuki C#:lle, Visual Basic.NET:lle, C++:lle, F#:lle, J#:lle, IronPythonille, IronRubylle.

Visual Studio 2012:

Языки: Поддержка C#, Visual Basic.NET, C++, F#, JavaScript, Python.

Visual Studio 2013:

Kielet: Tuki C#:lle, Visual Basic.NET:lle, C++:lle, F#:lle, JavaScriptille, Pythonille.

Visual Studio 2015:

Kielet: Tuki C#:lle, Visual Basic.NET:lle, C++:lle, F#:lle, JavaScriptille, Pythonille, TypeScriptille.

Visual Studio 2017:

Kielet: Tuki C#:lle, Visual Basic.NET:lle, C++:lle, F#:lle, JavaScriptille, Pythonille, TypeScriptille.

Visual Studio 2019:

Kielet: Tuki C#:lle, Visual Basic.NET:lle, C++:lle, F#:lle, JavaScriptille, Pythonille, TypeScriptille.

Visual Studio 2022:

Kielet: Tuki C#:lle, Visual Basic.NET:lle, C++:lle, F#:lle, JavaScriptille, Pythonille, TypeScriptille.

Jokainen Visual Studio -versio sisältää yleensä työkalut erilaisten sovellustyyppien kehittämiseen, kuten Windows-sovellukset, web-sovellukset, mobiilisovellukset, palvelut ja muut. Lisäksi Visual Studio tukee monipuolisesti erilaisia teknologioita, kuten .NET Framework, .NET Core, ASP.NET, Xamarin ja monet muut.

2.3 .Net MAUI ja sen luomisen historia

.NET MAUI (Multi-platform App UI) on moderni kehys krossi-alustaisten mobiili- ja työpöytäsovellusten luomiseen, ja se on esitelty Microsoftin toimesta. Tutustutaan .NET MAUI:n luomisen historiaan:

Xamarinin juuret:

Vuonna 2011 perustettiin Xamarin-niminen yritys. He kehittivät alustaa krossi-alustaista mobiilikehitystä varten, mahdollistaen C# -ohjelmointikielen ja .NETin käytön sovellusten luomiseen eri mobiilialustoille, kuten iOS, Android ja Windows.

Xamarinin hankinta Microsoftin toimesta:

Vuonna 2016 Microsoft hankki Xamarinin. Tämä hankinta vahvisti Microsoftin strategista suuntautumista tukemaan krossi-alustaista sovelluskehitystä C#:n ja .NETin avulla.

.NET 5 ja projektien yhdistäminen:

Vuonna 2020 Microsoft yhdisti eri .NET-projektit yhdeksi .NET 5:ksi. Tämä sisälsi .NET Coren, .NET Frameworkin ja Xamarinin. Näiden alustojen yhdistäminen mahdollisti kehittäjille yhtenäisen teknologiapinon käytön sovellusten kehityksessä eri kohdealustoille.

.NET MAUI:

Build 2020 -konferenssissa Microsoft esitteli .NET MAUI:n. Tämä kehys edustaa seuraavaa kehitysvaihetta Xamarin.Formsista ja mahdollistaa krossi-alustaisten sovellusten luomisen iOS:lle, Androidille, Windowsille ja macOS:lle yhteistä koodia käyttäen.

.NET MAUI:n ominaisuudet:

Yhteinen koodi: Kehittäjät voivat kirjoittaa yhteistä koodia liiketoimintalogiikalle, käyttöliittymälle ja muille komponenteille, ja käyttää sitten alustakohtaisia moduuleita tarvittaessa.

Yhtenäinen käyttöliittymä: .NET MAUI sisältää uuden käyttöliittymämallin, joka tarjoaa yhtenäisemmän käyttöliittymän eri alustoilla.

Monialustatuki: Kyky luoda sovelluksia iOS:lle, Androidille, Windowsille ja macOS:lle käyttäen samaa koodia.

Jatkuva kehitys:

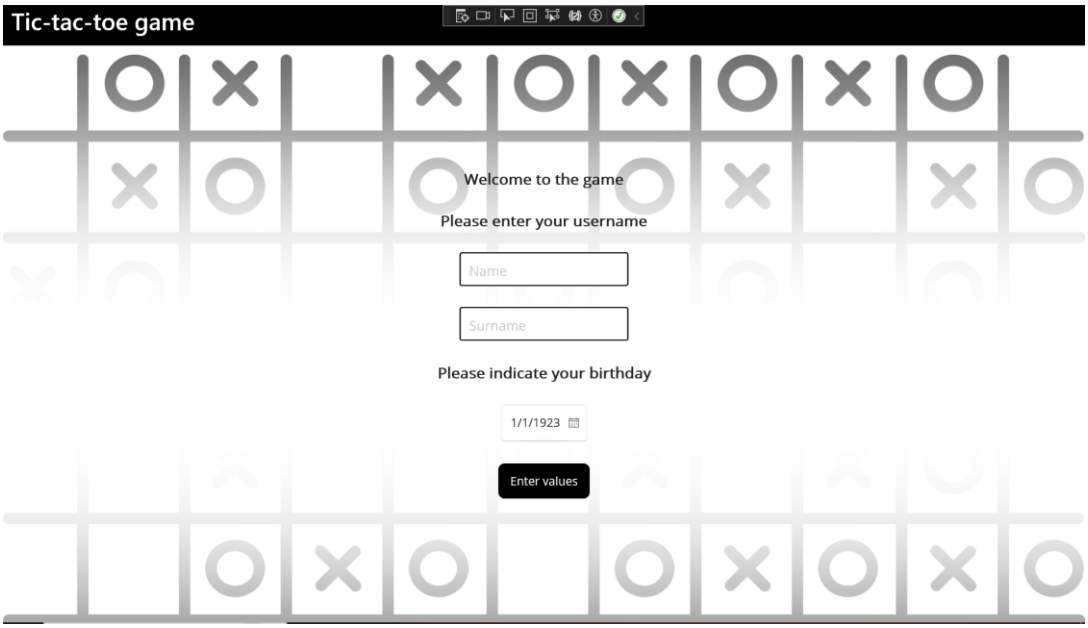
Microsoft jatkaa .NET MAUI:n aktiivista kehittämistä ja tarjoaa päivityksiä pitääkseen tuen uusille alustaversioille ja tarjotakseen kehittäjille uusia toimintoja.

.NET MAUI on merkittävä askel krossi-alustaisten sovellusten kehityksessä .NET-alustalla, tarjoten kehittäjille helppoutta ja tehokkuutta sovellusten luomisessa eri laitteille ja alustoille.

3 KÄYTTÖOPAS

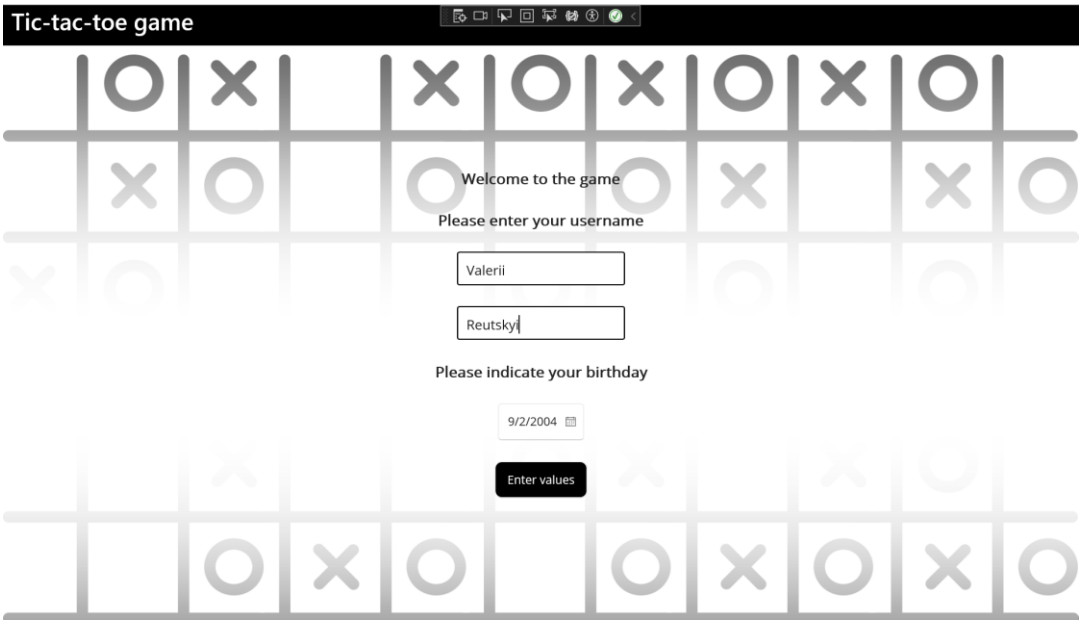
3.1 Ohjelman hallinta

3.1.1 Sovelluksen ensimmäisessä avauksessa käyttäjä kohtaa tämän sovelluksen etupuolen (Label)

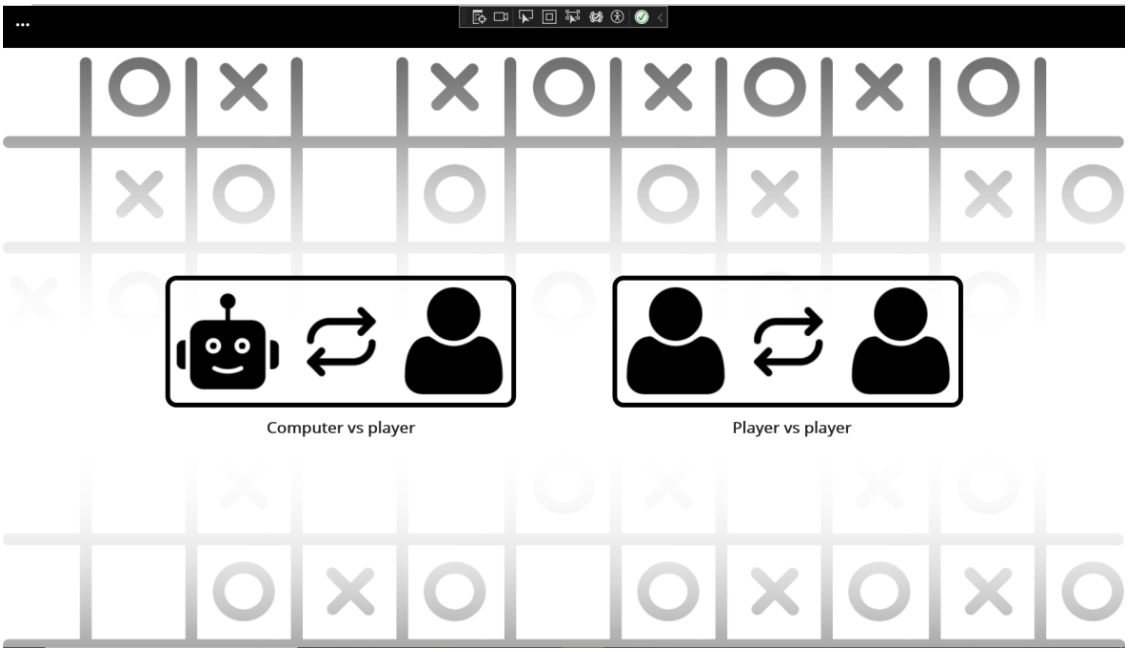


(Label)

Missä vaiheessa meidän tulee ilmoittaa nimi, sukunimi ja syntymäaika, ja näitä tietoja käytetään vain sovelluksessa eikä missään muualla. Kun olemme antaneet tiedot, voimme painaa 'Syötä arvot' -painiketta.



3.1.2 Kun painat painiketta, se tallentaa tietomme profiiliin ja siirtää meidät seuraavalle sivulle (GameLabel).



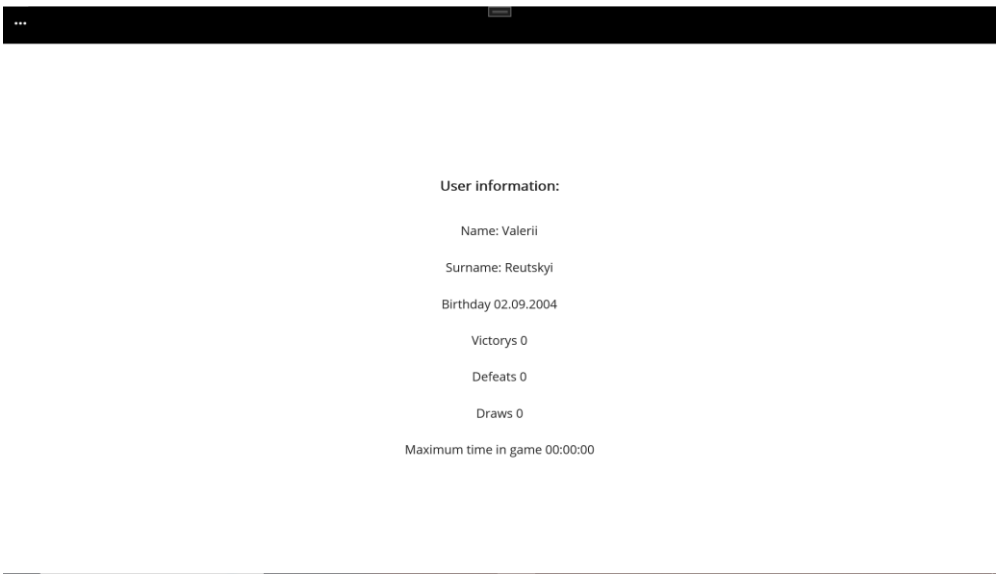
(GameLabel)

Tässä vaiheessa meitä tervehtivät kaksi painiketta keskellä näyttöä ja kolme pistettä näytön vasemmassa yläkulmassa.

3.1.3 Tässä vaiheessa meitä tervehtivät kaksi painiketta näytön keskellä ja kolme pistettä näytön vasemmassa yläkulmassa. Tässä vaiheessa voimme siirtyä profiiliin painamalla kolmea pistettä näytön vasemmassa yläkulmassa, minkä jälkeen avautuu kaksi painiketta: 'Profiili' ja 'Lopeta', kuva (Navig.1).

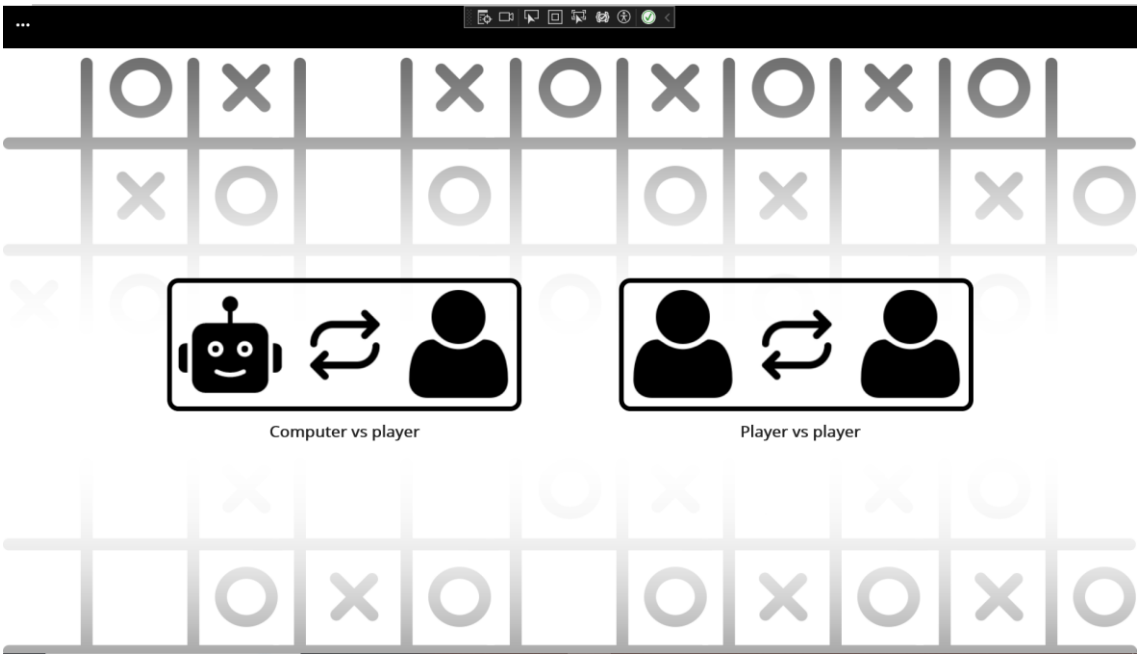
(Navig.1)

Profiilin painiketta painamalla siirrymme profiiliin, kun taas 'Quit'-painikkeen avulla poistumme sovelluksesta. Jos haluamme siirtyä profiiliin, seuraamme edellisen kohdan 4 ohjeita ja siirrymme Profiiliin.



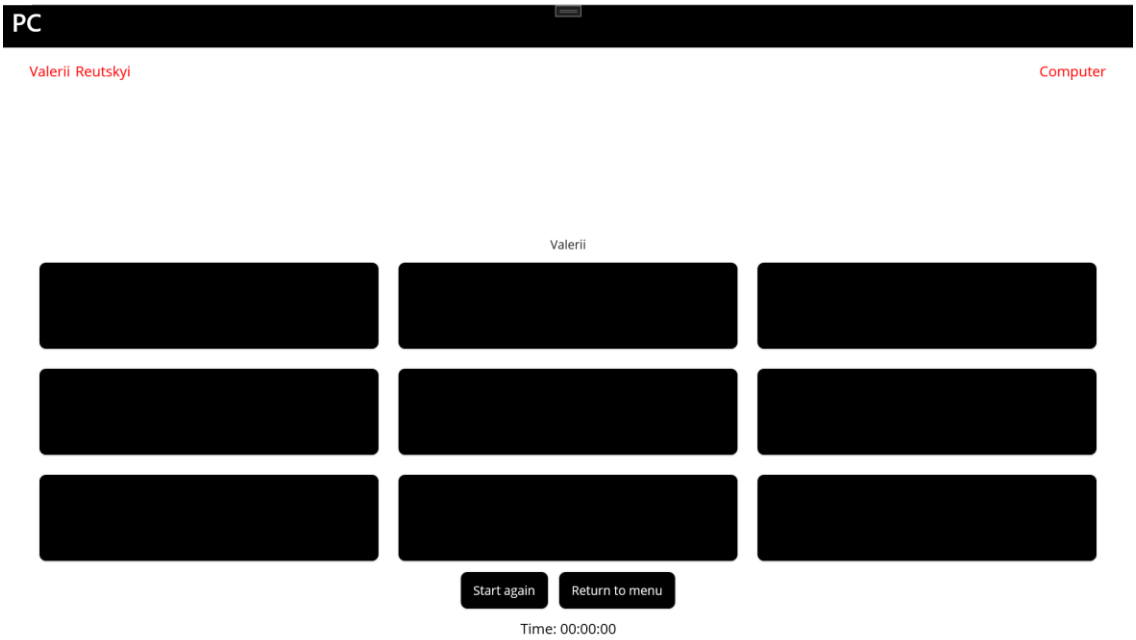
Täällä voimme löytää tiedot aiemmin syöttämästämme käyttäjästä, mukaan lukien kaikkien pelattujen pelien tilastot ja pisimmän koskaan pelatun pelin enimmäiskesto.

3.1.4 Palatessamme takaisin valikkoon (katso kuva 'GameLabel'), voimme nähdä alareunassa painikkeet, jotka on merkitty nimellä 'Computer vs Player' ja 'Player vs Player'.



(GameLabel)

3.1.5 Kun painamme 'Computer vs Player' -painiketta, siirrymme 'PC'-sivulle, missä voimme aloittaa peliprosessin.

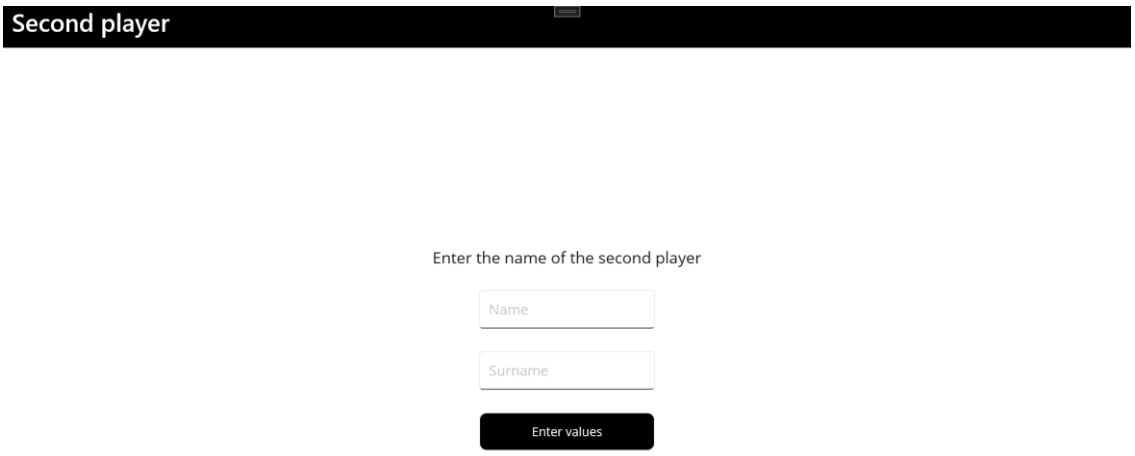


Myös pelikentän jälkeen meillä on kaksi painiketta: 'Aloita uudelleen' ja 'Palaa valikkoon'.

'Aloita uudelleen' aloittaa pelin alusta.

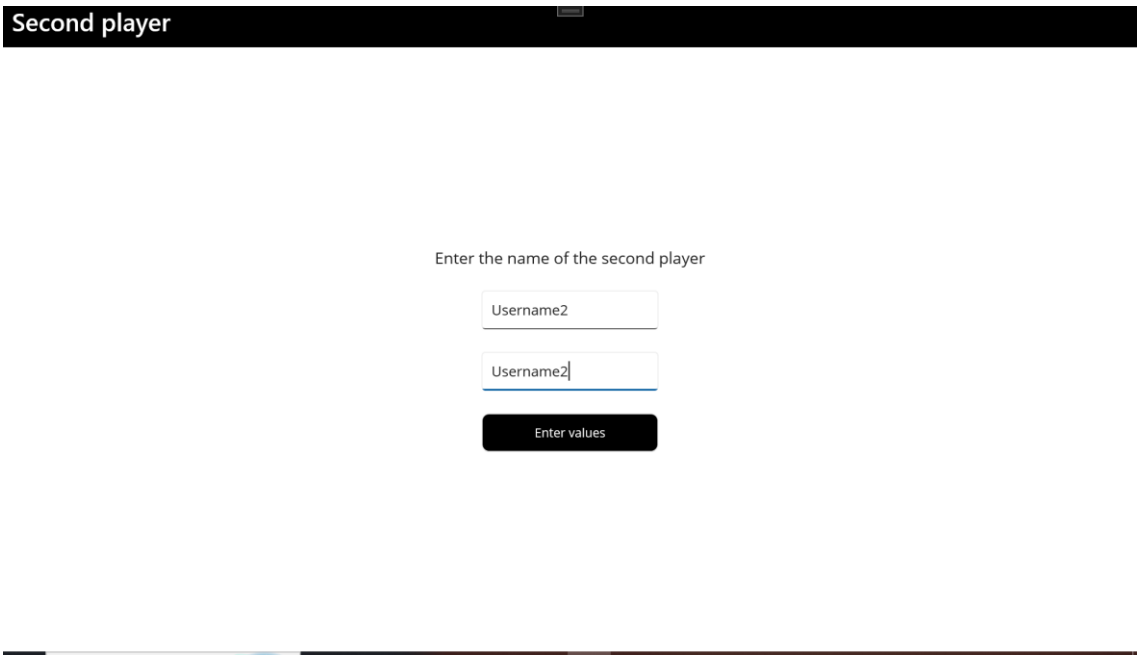
'Palaa valikkoon' vie meidät takaisin valikkoon.

3.1.6 Kun painamme 'Pelaaja vastaan pelaaja' -painiketta, siirrymme 'Toinen pelaaja' -sivulle, jossa voimme aloittaa peliprosessin.

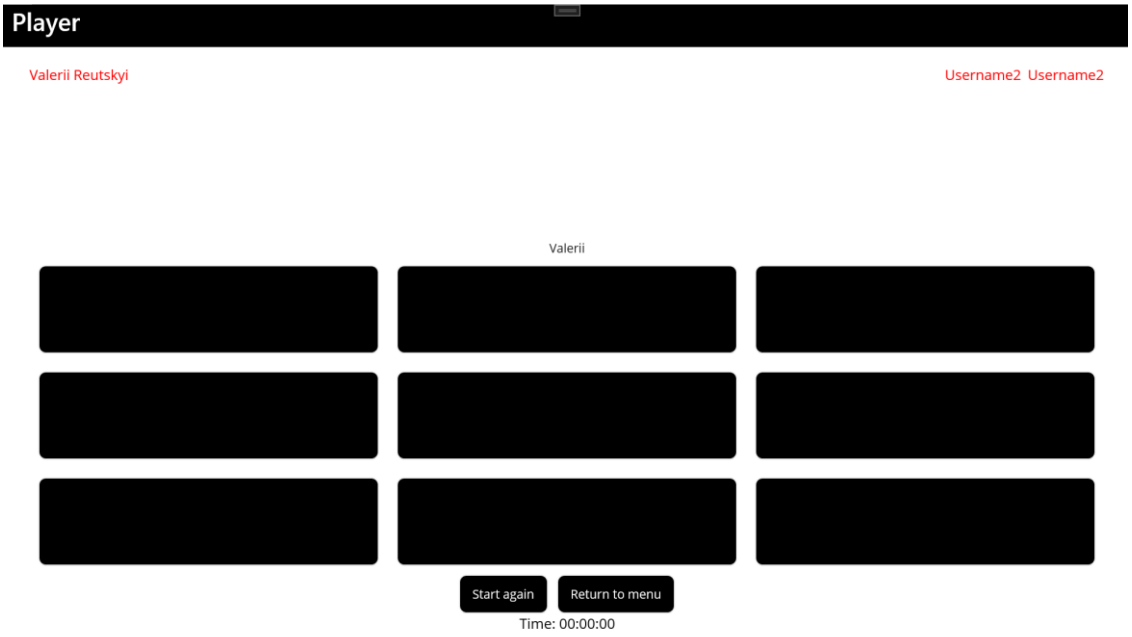


(Second player)

Sen jälkeen kun olemme antaneet toisen pelaajan tiedot, painamme 'Syötä arvot' -painiketta.



Lisättyämme toisen pelaajan siirrymme sivulle 'Pelaaja', missä voimme aloittaa peliprosessin.

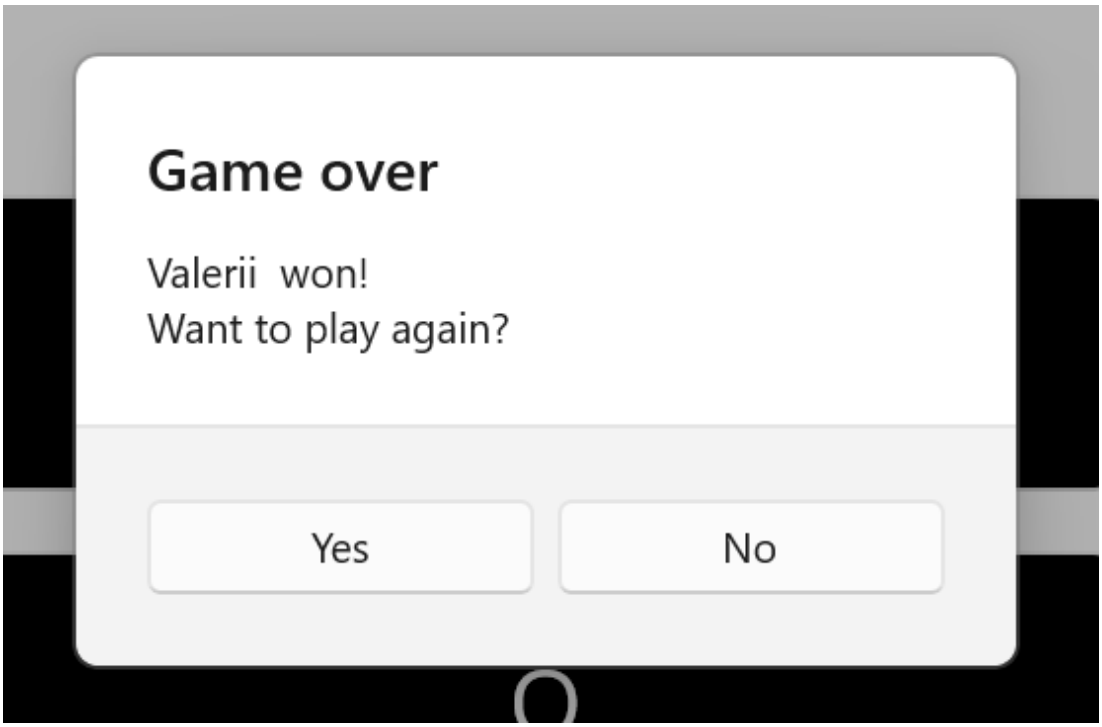


Myös pelikentän jälkeen meillä on kaksi painiketta: 'Aloita uudelleen' ja 'Palaa valikkoon'.

'Aloita uudelleen' aloittaa pelin alusta.

'Palaa valikkoon' vie meidät takaisin valikkoon.

3.1.7 Lisäksi pelin päätyttyä ilmestyy ikkuna pelin tuloksista, jonka jälkeen pelaajalle esitetään kysymys haluaako hän pelata uudelleen. Painamalla 'Kyllä' peli käynnistyy uudelleen, kun taas 'Ei'-painikkeella siirrymme takaisin valikkoon.



3.2 Pelisäännöt

Pelitavoite:

Pelitavoite on rakentaa rivi kolmea omaa symbolia (ristejä tai nollia) peräkkäin vaaka-, pysty- tai vinottaisesti pelilaudalla.

Pelilauta:

Laudan koko: Tavanomainen pelilauta on 3x3-ruudukko.

Ruutujen merkintä: Jokainen pelilaudan ruutu merkitään yksilöllisellä koordinaatilla, kuten A1, B2, C3 jne.

Pelimerkit:

Ensimmäinen pelaaja (X): Pelaaja, joka käyttää ristejä, yleensä tekee ensimmäisen siirron.

Toinen pelaaja (O): Pelaaja, joka käyttää nollia, tekee siirtonsa ensimmäisen pelaajan jälkeen.

Pelin kulku:

Vuorottaiset siirrot: Pelaajat tekevät vuorotellen siirtojaan asettaen symbolinsa mihin tahansa tyhjään ruutuun laudalla.

Siirtojen rajoitukset: Ei ole sallittua asettaa symbolia jo varattuun ruutuun.

Voitto:

Kolmen symbolin rivi: Pelaaja voittaa, jos hän onnistuu rakentamaan rivin kolmea omaa symboliaan vaaka-, pysty- tai vinorivillä.

Tasapeli:

Jos kaikki ruudut pelilaudalla ovat täynnä eikä kumpikaan pelaajista ole onnistunut muodostamaan kolmen symbolin riviä, peli päättyy tasapeliin.

Pelin päättyminen:

Peli päättyy, kun jompikumpi pelaajista voittaa tai peli päättyy tasapeliin. Pelistä voi sopia jatkamisesta tai se voidaan lopettaa.

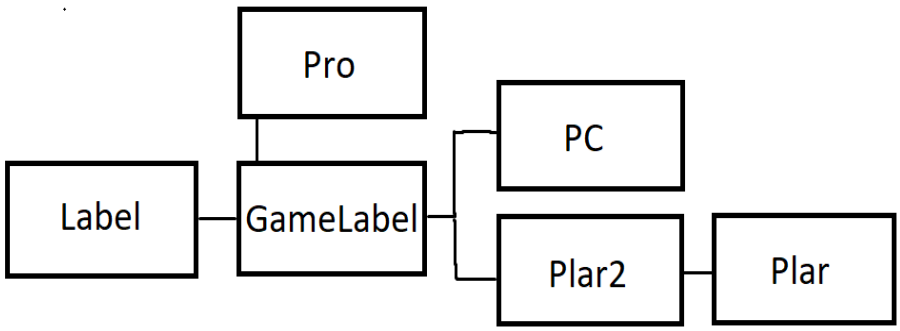
4 PELIN KULKU

4.1 Sivut ja niiden yhdistäminen

Aloittakaamme luettelemalla kaikki sivut, jotka ovat mukana tässä työssä:

1) Label – sivu henkilötietojen syöttämiseen (ensimmäinen sivu, joka tulee vastaan).
2) GameLabel – sivu pelimuodon valitsemiseen ja siirtymiseen profiiliin.
3) Pro – sivu henkilötietojen ja pelattujen pelien tilastojen näyttämiseen.
4) PC – sivu peliin perustuvan tekoälyn kanssa pelaamiseen.
5) Plar2 – sivu toisen pelaajan lisäämiseen.
6) Plar – sivu peliin toisen pelaajan kanssa (samalla laitteella).

Kaaviokuvaus kaikkien sivujen keskinäisestä yhdistämisestä (Kartta):



(Map)

Samaan tapaan, mikä auttaa yhdistämään ja välittämään tietoa toistensa välillä, on Player-instanssi, joka sijaitsee NewPage1:ssä.

4.2 NewPage1

```
Ссылок: 7
public class NewPage1 : ContentPage
{

    Ссылок: 9
    public class Player
    {

        Ссылок: 16
        public string FirstName { get; set; }
        Ссылок: 4
        public string LastName { get; set; }
        Ссылок: 8
        public string Firstperson { get; set; }
        Ссылок: 2
        public string Lastperson { get; set; }
        Ссылок: 2
        public string BirthYear { get; set; }
        Ссылок: 3
        public int Wins { get; set; }
        Ссылок: 3
        public int Losses { get; set; }
        Ссылок: 3
        public int Draws { get; set; }
        Ссылок: 5
        public TimeSpan TotalGameTime { get; set; }


    }

    Ссылок: 2
    public static void SetTotalGameTime(Player player, TimeSpan totalTime)
    {
        if (totalTime > player.TotalGameTime)
        {
            player.TotalGameTime = totalTime;
        }
    }

}
```

Tässä tapauksessa instanssi ottaa vastaan ja antaa arvot, jotka siihen on tallennettu. Se välittää nimen, sukunimen ja syntymäpäivän, jotka olemme tallentaneet siihen Label-vaiheessa (ks. kartta 'Map') painamalla 'Syötä arvot' -painiketta. Tämän jälkeen sitä käytetään painettaessa 'Profiili'-painiketta GameLabelissa, ja arvot NewPage1:stä siirtyvät Profiiliin. Samoin tapahtuu painettaessa 'Tietokone vs Pelaaja' -painiketta, ja arvot NewPage1:stä siirtyvät PC-sivulle. Sama toistuu myös painettaessa 'Syötä arvot' -painiketta Plar2:ssa.

Alla on kaikki rivit Player-instanssin käyttöä ja tietojen siirtoa varten NewPage1:stä:

Tietojen tallennus Player-instanssiin Labelista:

```
private async void OnCounterClicked(object sender, EventArgs e)
{
    try
    {
        PlayerData.Instance.Player = new Player
        { FirstName = FirstName.Text,
          LastName = SecondName.Text,
          BirthYear = DateOfBirthString };
        await Navigation.PushAsync(new ContentPage1());
    }
    catch (Exception ex)
    {
        _ = ex.Message;

    }


}
```

NewPage1:stä tietojen tallentaminen Pro-instanssiin:

```
private async void Profil(object sender, EventArgs e)
{
    try
    {
        var player = PlayerData.Instance.Player;
        await Navigation.PushAsync(new Pro(player));

    }
    catch (Exception ex)
    {
        _ = ex.Message;

    }
}
```

NewPage1:stä tietojen tallentaminen PC-instanssiin:

```
private async void Computer1(object sender, EventArgs e)
{
    try
    {
        var player = PlayerData.Instance.Player;
        await Navigation.PushAsync(new PC(player, "Player", ""));

    }
    catch (Exception ex)
    {
        _ = ex.Message;

    }
}
```

Plar2:sta tietojen tallentaminen Player-instanssiin:

```
private async void OnCounterClickedtogame(object sender, EventArgs e)
{
    try
    {
        PlayerData.Instance.Player.Firstperson = FirstName2per.Text;
        PlayerData.Instance.Player.Lastperson = SecondName2per.Text;

        var plarPage = new Plar(PlayerData.Instance.Player, FirstName2per.Text, SecondName2per.Text);
        await Navigation.PushAsync(plarPage);

    }
    catch (Exception ex)
    {
        _ = ex.Message;

    }
}
```

NewPage1:stä tietojen tallentaminen Player-instanssiin Plar:ssa ja samoin tietojen siirto NewPage1:stä Plar:iin:

```
private async void OnCounterClickedtogame(object sender, EventArgs e)
{
    try
    {
        PlayerData.Instance.Player.Firstperson = FirstName2per.Text;
        PlayerData.Instance.Player.Lastperson = SecondName2per.Text;

        var plarPage = new Plar(PlayerData.Instance.Player, FirstName2per.Text, SecondName2per.Text);
        await Navigation.PushAsync(plarPage);

    }
    catch (Exception ex)
    {
        _ = ex.Message;

    }
}
```

4.3 Perustason tekoäly

Alla oleva koodi toteuttaa perustason tekoälyn "risti-nolla"-peliin ihmisen ja tietokoneen välillä. Tässä toteutuksessa tietokone pystyy tekemään päätöksiä omasta siirrostaan perustuen strategiaan, joka sisältää mahdollisten voittavaa siirtoa sisältävien paikkojen etsimisen ja vastustajan siirtojen estämisen.

Tässä on toteutettu tekoäly myös muilla menetelmillä:

```
private async Task ComputerMove()
{
    if (_cells.Cast<Button>().Any(c => string.IsNullOrEmpty(c.Text)))
    {
        ComputerLoadingIndicator.IsRunning = true;
        ComputerLoadingIndicator.IsVisible = true;
        await Task.Delay(2000);

        var winningMove = FindWinningMove("X");
        var blockingMove = FindWinningMove("O");

        if (winningMove != null)
        {

            MakeComputerMove(winningMove);
        }
        else if (blockingMove != null)
        {

            MakeComputerMove(blockingMove);
        }
        else
        {

            MakeRandomComputerMove();
        }

        if (!_gameStopwatch.IsRunning)
        {
            StartGameTimer();
        }

        ComputerLoadingIndicator.IsRunning = false;
        ComputerLoadingIndicator.IsVisible = false;

        if (!CheckForWinner() && !_cells.Cast<Button>().Any(c => string.IsNullOrEmpty(c.Text)))
        {

        }
    }
}
```

Kaikki muut toiminnot, joiden perusteella tietokoneen liike logiikka toimii, sekä muut tärkeät toiminnot koko ohjelman toiminnalle:

1) Voittohahmon etsimismetodi:

```
private Button FindWinningMove(string symbol)
{
    // Check rows
    for (int i = 0; i < 3; i++)
    {
        if (CountSymbolsInLine(_cells[i, 0].Text, _cells[i, 1].Text, _cells[i, 2].Text, symbol) == 2)
        {
            return GetEmptyCellInLine(_cells[i, 0], _cells[i, 1], _cells[i, 2]);
        }
    }

    // Check columns
    for (int i = 0; i < 3; i++)
    {
        if (CountSymbolsInLine(_cells[0, i].Text, _cells[1, i].Text, _cells[2, i].Text, symbol) == 2)
        {
            return GetEmptyCellInLine(_cells[0, i], _cells[1, i], _cells[2, i]);
        }
    }

    // Check diagonals
    if (CountSymbolsInLine(_cells[0, 0].Text, _cells[1, 1].Text, _cells[2, 2].Text, symbol) == 2)
    {
        return GetEmptyCellInLine(_cells[0, 0], _cells[1, 1], _cells[2, 2]);
    }

    if (CountSymbolsInLine(_cells[0, 2].Text, _cells[1, 1].Text, _cells[2, 0].Text, symbol) == 2)
    {
        return GetEmptyCellInLine(_cells[0, 2], _cells[1, 1], _cells[2, 0]);
    }

    return null;
}
```

2) Metodi symbolien laskemiseksi rivissä:

```
private int CountSymbolsInLine(string symbol1, string symbol2, string symbol3, string targetSymbol)
{
    int count = 0;

    if (symbol1 == targetSymbol)
        count++;

    if (symbol2 == targetSymbol)
        count++;

    if (symbol3 == targetSymbol)
        count++;

    return count;
}
```

3) Metodi tyhjän solun saamiseksi rivillä:

```
private Button GetEmptyCellInLine(Button cell1, Button cell2, Button cell3)
{
    if (string.IsNullOrEmpty(cell1.Text))
        return cell1;

    if (string.IsNullOrEmpty(cell2.Text))
        return cell2;

    if (string.IsNullOrEmpty(cell3.Text))
        return cell3;

    return null;
}
```

4) Metodi tietokoneen siirron suorittamiseksi:

```
private void MakeComputerMove(Button cell)
{
    cell.Text = "X";

    _lastMoveRow = Grid.GetRow(cell);
    _lastMoveColumn = Grid.GetColumn(cell);

    if (CheckForWinner())
    {
        DisplayWinner("Computer");
    }
    else
    {
        _isPlayer1Turn = !_isPlayer1Turn;
        TurnLabel.Text = _isPlayer1Turn ? _player.FirstName : "Computer";
    }
}
```

5) Metodi tietokoneen satunnaisen siirron suorittamiseksi:

```
private void MakeRandomComputerMove()
{
    Random random = new Random();

    var emptyCells = _cells.Cast<Button>().Where(c => string.IsNullOrEmpty(c.Text)).ToArray();
    if (emptyCells.Length > 0)
    {
        var randomCell = emptyCells[random.Next(emptyCells.Length)];
        randomCell.Text = "X";

        _lastMoveRow = Grid.GetRow(randomCell);
        _lastMoveColumn = Grid.GetColumn(randomCell);

        if (CheckForWinner())
        {
            DisplayWinner("Computer");
        }
        else
        {
            _isPlayer1Turn = !_isPlayer1Turn;
            TurnLabel.Text = _isPlayer1Turn ? _player.FirstName : "Computer";
        }
    }
}
```

Alustus: Peli alkaa määrittämällä, aloittaako ihminen vai tietokone.

Ihmisen vuoro: Klikattaessa "O"-merkkiä sisältävää ruutua käsitellään ihmisen siirto. Jos voittoehdot täyttyvät, näytetään viesti ihmisen voitosta, muuten siirto siirtyy tietokoneelle.

Tietokoneen vuoro: Tietokone tekee päätöksen omasta siirrostaan. Jos mahdollisuus voittoon (kaksi merkkiä linjassa) on olemassa, se tekee voittoisan siirron. Jos on tarpeen estää vastustajaa (kaksi vastustajan merkkiä linjassa), tietokone estää sen. Muussa tapauksessa tietokone tekee satunnaisen siirron.

Voiton tai tasapelin tarkistus: Jokaisen siirron jälkeen tarkistetaan, onko voittajaa tai tasapeliä. Jos ehdot täyttyvät, näytetään asiaankuuluva viesti ja tarjotaan mahdollisuutta aloittaa uusi peli.

Ajastin: Peliaikaa lasketaan.

4.4 Logiikka peliin ihmisen kanssa

Alla oleva koodi toteuttaa "risti-nolla"-pelin logiikan kahdelle pelaajalle. Tässä toteutuksessa pelaajat tekevät siirtojaan vuorotellen, ja voittaa se, joka ensimmäisenä saa aikaan rivin kolmea omaa symboliaan (ristejä tai nollia) vaaka-, pysty- tai vinorivillä. Tässä on toteutettu logiikka peliin ihmisen kanssa (samalla koneella).

```
public Plar(Player player, string extraFirstName, string extraLastName)
{
    _player = player;
    _extraFirstName = extraFirstName;
    _extraLastName = extraLastName;

    InitializeComponent();
    _gameStopwatch = new Stopwatch();
    InitializeGame();
    FirstNameSecond.Text = $" {_player.Firstperson}  ";
    SecondNameSecond.Text = $" {_player.Lastperson} ";
    First.Text = $"{_player.FirstName}";
    Second.Text = $"{_player.LastName}";



}
```

Tässä on pelin logiikka pelaajien välillä:

```
private void InitializeGame()
{
    _isPlayer1Turn = GetRandomName() == _player.FirstName;

    TurnLabel.Text = _isPlayer1Turn ? _player.FirstName : _player.Firstperson;
    _cells = new Button[,]
    {
        { Cell00, Cell01, Cell02 },
        { Cell10, Cell11, Cell12 },
        { Cell20, Cell21, Cell22 }
    };

    foreach (var cell in _cells)
    {
        cell.Text = "";
        cell.Clicked += OnCellClicked;
    }
    _gameStopwatch.Reset();
}

Ссылок: 1
private void OnCellClicked(object sender, EventArgs e)
{
    var cell = (Button)sender;
    if (string.IsNullOrEmpty(cell.Text))
    {
        cell.Text = _isPlayer1Turn ? "X" : "O";


        _lastMoveRow = Grid.GetRow(cell);
        _lastMoveColumn = Grid.GetColumn(cell);


        if (!_gameStopwatch.IsRunning)
        {
            StartGameTimer();
        }

        if (CheckForWinner())
        {
            DisplayWinner(_isPlayer1Turn ? _player.FirstName : _player.Firstperson);
        }
        else
        {
            _isPlayer1Turn = !_isPlayer1Turn;
            TurnLabel.Text = _isPlayer1Turn ? _player.FirstName : _player.Firstperson;


            if (_cells.Cast<Button>().All(c => !string.IsNullOrEmpty(c.Text)))
            {
                DisplayWinner("Draw");
            }
        }
    }
}
```

Metodi voittajan tarkistamiseen:

```
private bool CheckForWinner()
{
    for (int i = 0; i < 3; i++)
    {
        if (_cells[i, 0].Text == _cells[i, 1].Text && _cells[i, 1].Text == _cells[i, 2].Text && !string.IsNullOrEmpty(_cells[i, 0].Text))
        {
            return true;
        }
    }

    for (int i = 0; i < 3; i++)
    {
        if (_cells[0, i].Text == _cells[1, i].Text && _cells[1, i].Text == _cells[2, i].Text && !string.IsNullOrEmpty(_cells[0, i].Text))
        {
            return true;
        }
    }

    if (_cells[0, 0].Text == _cells[1, 1].Text && _cells[1, 1].Text == _cells[2, 2].Text && !string.IsNullOrEmpty(_cells[0, 0].Text))
    {
        return true;
    }

    if (_cells[0, 2].Text == _cells[1, 1].Text && _cells[1, 1].Text == _cells[2, 0].Text && !string.IsNullOrEmpty(_cells[0, 2].Text))
    {
        return true;
    }


    return false;
}
```

Metodi voiton ilmoittamiseen ja voittajan määrittämiseen:

```
private async void DisplayWinner(string winner)
{
    string player1Name = _player.FirstName;
    string player2Name = _player.Firstperson;

    string outcomeMessage;

    if (winner == "Draw")
    {
        outcomeMessage = "It's a draw\nWant to play again?";
        _player.Draws++;
    }
    else if (_isPlayer1Turn)
    {

        _player.Wins++;
        outcomeMessage = $"{player1Name} won!\nWant to play again?";
    }
    else
    {
        _player.Losses++;
        outcomeMessage = $"{player2Name} won!\n{player1Name} lost!\nWant to play again?";
    }
    StopGameTimer();
    var result = await DisplayAlert("Game over", outcomeMessage, "Yes", "No");

    if (result)
    {
        RestartGame();
    }
    else
    {
        Ongout();
    }

}
```

4.4 Yhteenveto ja logiikka

Haluan korostaa, kuinka logiikka toimii pelin alustuksessa molemmissa tapauksissa, kun päätetään, kuka pelaajista aloittaa ensimmäisenä. Tämä tapahtuu käyttämällä metodia GetRandomName(), joka valitsee satunnaisesti yhden pelaajan nimen. Valitun pelaajan nimi tulee nykyiseksi pelaajaksi, joka tekee ensimmäisen siirron.

```
private string GetRandomName()
{
    Random random = new Random();
    int randomIndex = random.Next(2);

    return randomIndex == 0 ? $" {_player.Firstperson}  " : $"{_player.FirstName}";
}


Ссылок: 3
private void InitializeGame()
{
    _isPlayer1Turn = GetRandomName() == _player.FirstName;

    TurnLabel.Text = _isPlayer1Turn ? _player.FirstName : _player.Firstperson;
```

Lisäksi on syytä huomata logiikka pelatessa tietokonetta vastaan, missä valinta pelaajan ja tietokoneen välillä määritellään satunnaisesti. Tässä tapauksessa päätetään, kumpi aloittaa ensimmäisenä - pelaaja vai tietokone.

```
private void MakeRandomComputerMove()
{
    Random random = new Random();

    var emptyCells = _cells.Cast<Button>().Where(c => string.IsNullOrEmpty(c.Text)).ToArray();
    if (emptyCells.Length > 0)
    {
        var randomCell = emptyCells[random.Next(emptyCells.Length)];
        randomCell.Text = "X";

        _lastMoveRow = Grid.GetRow(randomCell);
        _lastMoveColumn = Grid.GetColumn(randomCell);

        if (CheckForWinner())
        {
            DisplayWinner("Computer");
        }
        else
        {
            _isPlayer1Turn = !_isPlayer1Turn;
            TurnLabel.Text = _isPlayer1Turn ? _player.FirstName : "Computer";
        }
    }
}
```

Kun napsautat pelikentän solua (joka on esitetty Button-tyyppisenä objektina), tapahtumankäsittelijä OnCellClicked käynnistyy. Tässä metodissa tarkistetaan, onko solu tyhjä, ja jos on, siihen asetetaan nykyisen pelaajan symboli ("X" tai "O"). Tämän jälkeen tarkistetaan voittajan olemassaolo.

```
private void OnCellClicked(object sender, EventArgs e)
{
    var cell = (Button)sender;
    if (string.IsNullOrEmpty(cell.Text))
    {
        cell.Text = _isPlayer1Turn ? "X" : "O";


        _lastMoveRow = Grid.GetRow(cell);
        _lastMoveColumn = Grid.GetColumn(cell);


        if (!_gameStopwatch.IsRunning)
        {
            StartGameTimer();
        }

        if (CheckForWinner())
        {
            DisplayWinner(_isPlayer1Turn ? _player.FirstName : _player.Firstperson);
        }
        else
        {
            _isPlayer1Turn = !_isPlayer1Turn;
            TurnLabel.Text = _isPlayer1Turn ? _player.FirstName : _player.Firstperson;


            if (_cells.Cast<Button>().All(c => !string.IsNullOrEmpty(c.Text)))
            {
                DisplayWinner("Draw");
            }
        }
    }
}
```

Kun pelaaja napsauttaa solua tietokoneen ollessa aktiivinen (_isComputerThinking tarkistetaan), jotta vältetään tietokoneen siirto pelaajan vuoron aikana. Tämän jälkeen pelaajan siirron jälkeen tarkistetaan voittaja tai tasapeli. Jos peli ei ole päättynyt, siirto siirtyy tietokoneelle.

```
private async void OnCellClicked(object sender, EventArgs e)
{

    if (_isComputerThinking)
    {
        return;
    }

    var cell = (Button)sender;
    if (string.IsNullOrEmpty(cell.Text))
    {
        cell.Text = "O";
        _lastMoveRow = Grid.GetRow(cell);
        _lastMoveColumn = Grid.GetColumn(cell);


        if (!_gameStopwatch.IsRunning)
        {
            StartGameTimer();
        }

        if (CheckForWinner())
        {
            DisplayWinner(_isPlayer1Turn ? _player.FirstName : "Computer");
        }
        else
        {
            _isPlayer1Turn = !_isPlayer1Turn;
            TurnLabel.Text = _isPlayer1Turn ? _player.FirstName : "Computer";

            if (!_isPlayer1Turn)
            {

                _isComputerThinking = true;
                await ComputerMove();
                _isComputerThinking = false;


            }
        }
        if (!CheckForWinner() && !_cells.Cast<Button>().Any(c => string.IsNullOrEmpty(c.Text)))
        {

            DisplayWinner("Draw");

        }
    }
}
```

Seuraavaksi voiton tarkistus suoritetaan käyttämällä metodia CheckForWinner(). Tämä metodi käy läpi kaikki vaakasuorat, pystysuorat ja vinot pelikentän rivit tarkistaen, sisältävätkö ne samoja symboleja. Jos tällainen rivi löytyy, pelaajaa, jonka symboli on kyseisellä rivillä, pidetään voittajana. Tässä tapauksessa tietokoneen toiminta logiikalla on täsmälleen samanlainen.

```
private bool CheckForWinner()
{
    for (int i = 0; i < 3; i++)
    {
        if (_cells[i, 0].Text == _cells[i, 1].Text && _cells[i, 1].Text == _cells[i, 2].Text && !string.IsNullOrEmpty(_cells[i, 0].Text))
        {
            return true;
        }
    }

    for (int i = 0; i < 3; i++)
    {
        if (_cells[0, i].Text == _cells[1, i].Text && _cells[1, i].Text == _cells[2, i].Text && !string.IsNullOrEmpty(_cells[0, i].Text))
        {
            return true;
        }
    }

    if (_cells[0, 0].Text == _cells[1, 1].Text && _cells[1, 1].Text == _cells[2, 2].Text && !string.IsNullOrEmpty(_cells[0, 0].Text))
    {
        return true;
    }

    if (_cells[0, 2].Text == _cells[1, 1].Text && _cells[1, 1].Text == _cells[2, 0].Text && !string.IsNullOrEmpty(_cells[0, 2].Text))
    {
        return true;
    }


    return false;
}
```

Jokaisen siirron jälkeen tarkistetaan voittaja. Jos voittaja löytyy, kutsutaan metodia DisplayWinner, joka näyttää voittoviestin ja ehdottaa pelaajille uuden pelin aloittamista tai palaamista päävalikkoon.

```
private async void DisplayWinner(string winner)
{
    string player1Name = _player.FirstName;
    string player2Name = _player.Firstperson;

    string outcomeMessage;

    if (winner == "Draw")
    {
        outcomeMessage = "It's a draw\nWant to play again?";
        _player.Draws++;
    }
    else if (_isPlayer1Turn)
    {

        _player.Wins++;
        outcomeMessage = $"{player1Name} won!\nWant to play again?";
    }
    else
    {
        _player.Losses++;
        outcomeMessage = $"{player2Name} won!\n{player1Name} lost!\nWant to play again?";
    }
    StopGameTimer();
    var result = await DisplayAlert("Game over", outcomeMessage, "Yes", "No");

    if (result)
    {
        RestartGame();
    }
    else
    {
        Ongout();
    }
}
```

Metodi RestartGame() käynnistää pelin uudelleen, ja metodi OnQuit() palaa päävalikkoon vastaavasti.

```
private async void Ongout()
{
    await Navigation.PushAsync(new ContentPage1());

}

Ссылок: 0
private void OnRestartClicked(object sender, EventArgs e)
{

    InitializeGame();

}
```

Sen jälkeen, kun tietokoneen vuoro on tulossa, on luotava perustavanlaatuinen tekoäly. Kun pelaajan ja tietokoneen välillä on tehty satunnainen valinta, ensimmäinen vaihe on tehdä satunnainen valinta solun joukosta {Cell00, Cell01, Cell02}, {Cell10, Cell11, Cell12}, {Cell20, Cell21, Cell22}. Sitten pelaajan siirron jälkeen astuu voimaan logiikka voittavan vaihtoehdon löytämiseksi. Kuitenkin kehittäessäni perustavanlaatuista tekoälyä huomasin, että aiempi logiikkani oli liian älykästä. Siksi tässä vaihtoehdossa loogiset kyvyt ovat rajoitetumpia.

```
private Button FindWinningMove(string symbol)
{
    // Check rows
    for (int i = 0; i < 3; i++)
    {
        if (CountSymbolsInLine(_cells[i, 0].Text, _cells[i, 1].Text, _cells[i, 2].Text, symbol) == 2)
        {
            return GetEmptyCellInLine(_cells[i, 0], _cells[i, 1], _cells[i, 2]);
        }
    }

    // Check columns
    for (int i = 0; i < 3; i++)
    {
        if (CountSymbolsInLine(_cells[0, i].Text, _cells[1, i].Text, _cells[2, i].Text, symbol) == 2)
        {
            return GetEmptyCellInLine(_cells[0, i], _cells[1, i], _cells[2, i]);
        }
    }

    // Check diagonals
    if (CountSymbolsInLine(_cells[0, 0].Text, _cells[1, 1].Text, _cells[2, 2].Text, symbol) == 2)
    {
        return GetEmptyCellInLine(_cells[0, 0], _cells[1, 1], _cells[2, 2]);
    }

    if (CountSymbolsInLine(_cells[0, 2].Text, _cells[1, 1].Text, _cells[2, 0].Text, symbol) == 2)
    {
        return GetEmptyCellInLine(_cells[0, 2], _cells[1, 1], _cells[2, 0]);
    }

    return null;
}
```

Ja sitten keräämme kaiken yhteen tehtävässä ComputerMove() tietokonelogiikassa.

```
private async Task ComputerMove()
{
    if (_cells.Cast<Button>().Any(c => string.IsNullOrEmpty(c.Text)))
    {
        ComputerLoadingIndicator.IsRunning = true;
        ComputerLoadingIndicator.IsVisible = true;
        await Task.Delay(2000);

        var winningMove = FindWinningMove("X");
        var blockingMove = FindWinningMove("O");

        if (winningMove != null)
        {

            MakeComputerMove(winningMove);
        }
        else if (blockingMove != null)
        {

            MakeComputerMove(blockingMove);
        }
        else
        {

            MakeRandomComputerMove();
        }

        if (!_gameStopwatch.IsRunning)
        {
            StartGameTimer();
        }

        ComputerLoadingIndicator.IsRunning = false;
        ComputerLoadingIndicator.IsVisible = false;

        if (!CheckForWinner() && !_cells.Cast<Button>().Any(c => string.IsNullOrEmpty(c.Text)))
        {

        }
    }
}
```

Tietysti ajastin seuraa peliaikaa ja päivittää sen näyttämön näytöllä.

```
private void StartGameTimer()
{
    _gameStopwatch.Start();
    Device.StartTimer(TimeSpan.FromSeconds(1), () =>
    {
        UpdateTimerLabel();
        return true;
    });
}


Ссылок: 1
private void StopGameTimer()
{
    _gameStopwatch.Stop();
    UpdateTimerLabel();
    TimeSpan elapsedTime = _gameStopwatch.Elapsed;
    SetTotalGameTime(_player, elapsedTime);
}
Ссылок: 2
private void UpdateTimerLabel()
{
    Device.BeginInvokeOnMainThread(() =>
    {
        TimeSpan elapsed = _gameStopwatch.Elapsed;
        TimerLabel.Text = $"Time: {elapsed.Hours:D2}:{elapsed.Minutes:D2}:{elapsed.Seconds:D2}";
    });
}
```

4.6 Pelaajaprofiili

Alla oleva koodi toteuttaa pelaajaprofiilin, se ottaa tiedot NewPage1:stä. Tässä on määritelty luokka Pro, joka on peritty ContentPage:sta. Tämän luokan konstruktori ottaa Player-olion (luultavasti luokka, jossa on pelaajan tiedot). Konstruktorissa suoritetaan sivun komponenttien alustus (InitializeComponent()) ja asetetaan tekstiarvot käyttöliittymän elementeille (kuten FirstNameLabel, WinsNameLabel jne.) pelaajan tiedoista _player.

```
Ссылок: 5
public partial class Pro : ContentPage
{

    private Player _player;
    Ссылок: 1
    public Pro(Player player)
    {
        InitializeComponent();
        _player = player;
        FirstNameLabel.Text = $"Name: {_player.FirstName}";
        SecondNameLabel.Text = $"Surname: {_player.LastName}";
        BirthNameLabel.Text = $"Birthday {_player.BirthYear}";
        WinsNameLabel.Text = $"Victorys {_player.Wins}";
        LossesNameLabel.Text = $"Defeats {_player.Losses}";
        DrawsNameLabel.Text = $"Draws {_player.Draws}";
        string formattedTime = $"{_player.TotalGameTime.Hours:D2}:{_player.TotalGameTime.Minutes:D2}:{_player.TotalGameTime.Seconds:D2}";
        TotalGameTime1NameLabel.Text = $"Maximum time in game {formattedTime}";
    }


    Ссылок: 0
    private async void gooutProfil(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new ContentPage1());
    }

    Ссылок: 0
    private  void Out(object sender, EventArgs e)
    {
        Application.Current.Quit();
    }
}
```

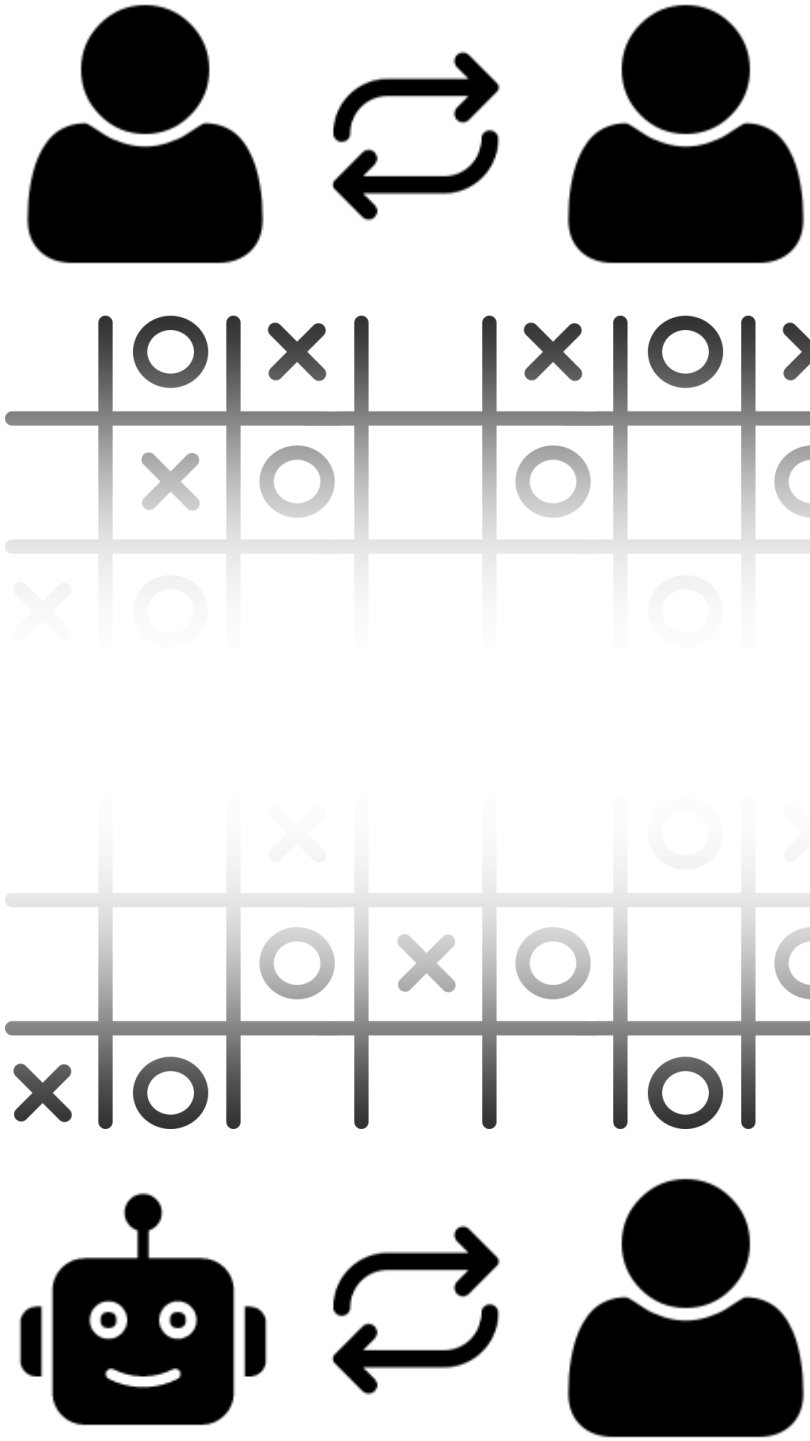
Metodia gooutProfil käytetään siirtymiseen päävalikkoon. Tämän metodin kutsun aikana tapahtuu navigointi ContentPage1-instanssiin. Metodi Out sammuttaa sovelluksen.

```
private async void gooutProfil(object sender, EventArgs e)
{
    await Navigation.PushAsync(new ContentPage1());
}

Ссылок: 0
private  void Out(object sender, EventArgs e)
{
    Application.Current.Quit();
}
```

4.7 Suunnittelu ja asettelu

Tässä vaiheessa on mainittava sovelluksen suunnittelusta. Aloitetaan tarvittavien kuvien lisäämisestä itse sovellukseen. Itse lisäsin kaikki valokuvat Images-kansioon, joka sijaitsee Resources-kansiossa.



Tämä suunnittelu oli henkilökohtainen luomukseni, mutta pääidea otettiin ystävältäni, joka opiskelee suunnittelualalla.

ImageButtonin avulla luotiin kuvapainikkeita.

```
<StackLayout Orientation="Horizontal" HorizontalOptions="Center" VerticalOptions="Center" Margin="0,20,0,0">

    <StackLayout>
    <ImageButton  Source="robothuman.png"  WidthRequest="400" HeightRequest="150" x:Name="Computer" Clicked="Computer1" BorderColor="Black" BorderWidth="5"  CornerRadius="10" Margin="0,0,110,0" />
        <Label
            Text="Computer vs player" FontAttributes="Bold"  FontSize="18"  HorizontalOptions="Center" Margin="0,10,110,0" />
    </StackLayout>
    <StackLayout>
    <ImageButton Source="humanhuman.png" WidthRequest="400" HeightRequest="150" x:Name="Player" Clicked="Player1" BorderColor="Black" BorderWidth="5"  CornerRadius="10" Margin="0,0,0,0" />
        <Label
            Text="Player vs player" FontAttributes="Bold"  FontSize="18"  HorizontalOptions="Center" Margin="10,10,0,0" />
    </StackLayout>
</StackLayout>
```

Käyttämällä <Image Source="my_piv.png" Aspect="AspectFill" Opacity="0.7" Margin="0,10,0,0" /> sovelluksen tausta luotiin.

5 ITSEARVIOINTI

Tässä osiossa haluaisin mainita, mitä opin näiden töiden tekemisen aikana. Rehellisesti sanottuna tein suuren virheen painikkeiden navigoinnissa, ja rehellisesti sanottuna käytin yli 5 tuntia tämän ongelman ratkaisemiseen. Luulin, että ongelma oli logiikassa, mutta kävi ilmi, että en jopa käynnistänyt tietokonelogiikkaa. Ongelma oli siinä, että kehittäessäni logiikkaa ihmisen ja ihmisen pelaamiseen, määritin 'Computer vs Player' -painikkeelle avautumisen samalla logiikalla kuin 'Player vs Player', jotta vältettäisiin virhe. Nyt muistellessani sitä hymyilen. Paljon hermoja meni ymmärtämättömyyden takia.

Voisin sanoa pari sanaa myös suunnittelusta. Opin uuden tavan käyttää ImageButton-painikkeita. En tiennyt siitä, mutta onnistuin oppimaan sen, tosin internetin avulla.

Projektin alussa oli suuri kysymys siitä, miten luoda perustavanlaatuinen tekoäly. Kuitenkin tämän projektin tekemisen ansiosta sain arvokasta kokemusta sovelluksen luomisessa. Joten rehellisesti sanottuna, jos voisin palata taaksepäin ja olettaa, että olen unohtanut, miten sen tein, tekisin sen uudelleen.

Mielestäni koodini ei ehkä ole täysin puhdas, mutta se toimii, kuten sanotaan - 'Jos se toimii, älä koske siihen.