Sveučilište u Rijeci

Fakultet informatike i digitalnih tehnologija

Objektno Programiranje

ZAVRŠNI RAD IZ KOLEGIJA   
OBJEKTNO PROGRAMIRANJE

Izrada igre Peg Solitaire korištenjem koncepata objektnog programiranja u C#-u.

Nastavnik: Doc. dr. sc. Miran Pobar

Student: Zara Čubranić

Rijeka, rujan 2023.

Contents

[1. UVOD 3](#_Toc144676921)

[2. OPĆI PREGLED APLIKACIJE 4](#_Toc144676922)

[2.1 Cilj/Ideja 4](#_Toc144676923)

[2.2 Funkcionalnost 4](#_Toc144676924)

[3. Opis strukture programa 6](#_Toc144676925)

[3.1. Klase aplikacije 6](#_Toc144676926)

[3.2. Korištene biblioteke 7](#_Toc144676927)

[3.3. UML Dijagrami 7](#_Toc144676928)

[3.3.1. Dijagram klasa 8](#_Toc144676929)

[3.3.2. Dijagram aktivnosti 8](#_Toc144676930)

[3.3.4. Dijagram slijeda 9](#_Toc144676931)

[4. Upute za buildanje programa 11](#_Toc144676932)

[5. Osnovne upute za korištenje programa 12](#_Toc144676933)

[6. Zaključak 13](#_Toc144676934)

[Kod 14](#_Toc144676935)

[MainWindow.xaml 14](#_Toc144676936)

[MainWindow.xaml.cs 14](#_Toc144676937)

[Cell.cs 18](#_Toc144676938)

[GameLogic.cs 19](#_Toc144676939)

# UVOD

Peg Solitaire, ili Solo Noble, je drevna logička igra s jasnim ciljem - ostaviti samo jedan peg na ploči nakon serije valjanih poteza. Iako je igra u svojoj srži jednostavna, tokom vremena je postala izazov za brojne ljubitelje slagalica. Dok je tradicionalna verzija Peg Solitaire-a igrana na drvenoj ploči s drvenim pegovima, dolaskom digitalne ere otvorila se mogućnost njene adaptacije za moderne platforme.

U ovom dokumentu predstavlja se digitalna adaptacija klasične igre Peg Solitaire. Njena svrha je da pruži sveobuhvatan pregled digitalne adaptacije klasične igre, fokusirajući se na njen dizajn, razvoj i funkcionalnosti.

Predstavljena aplikacija je skromnog obima, dizajnirana s osnovnim funkcionalnostima koje omogućuju igračima da dožive izazov Peg Solitaire-a u digitalnom okruženju. Unatoč svojoj osnovnoj strukturi, aplikacija efikasno prenosi srž same igre i pruža sve potrebne mehanike za njen užitak.

U ovom dokumentu detaljno će se razmotriti osnovna ideja aplikacije, njena struktura i tehnologije koje su korištene u njenom razvoju, te upute za njen rad i korištenje.

# OPĆI PREGLED APLIKACIJE

Aplikacija Peg Solitaire ima zadaću donijeti tradicionalno igračko iskustvo u suvremeno digitalno doba. Dok je u svojoj originalnoj formi igra bila popularna u određenim krugovima i kulturama, digitalna adaptacija pruža šansu za obnovljenom popularnošću i dostupnošću.

## 2.1 Cilj/Ideja

S glavnim ciljem da se klasična slagalica prilagodi modernim računalima, ovaj projekt je rođen iz želje da se omogući pristup igri Peg Solitaire široj publici. Digitalizacija igre osigurava da može biti dijeljena, preuzeta i igrana bez potrebe za fizičkom pločom ili komponentama. Osim toga, digitalna verzija pruža dodatne pogodnosti poput praćenja rezultata, automatske provjere valjanosti poteza i grafičkog prikaza igre.

## 2.2 Funkcionalnost

Na temelju ideje o prilagodbi ove tradicionalne igre, aplikacija donosi sljedeće ključne funkcionalnosti:

* Igranje Peg Solitaire-a:

Korisniku se pruža jasno vizualizirana ploča s pegovima koji odražavaju tradicionalnu postavu igre. Interakcija je jednostavna - igrač može kliknuti na peg kako bi ga odabrao, a potom odabrati ciljnu poziciju za premještanje.

* Automatska provjera poteza:

Kako bi se osiguralo da se igra igra prema pravilima, aplikacija automatski provjerava valjanost svakog poteza, sprječavajući korisnika u izvođenju nevažećih akcija.

* Praćenje i bilježenje rezultata:

Nakon što je partija završena, ako igrač pobijedi, aplikacija bilježi rezultate. Time se igračima pruža prilika da prate svoj napredak i nastoje poboljšati svoje rezultate u budućim partijama.

* Jednostavnost korištenja:

Osmišljena s naglaskom na intuitivnost, aplikacija ne zahtijeva nikakvo prethodno iskustvo, već samo upoznavanje s pravilima igre.

Ova aplikacija, iako osnovna u svom dizajnu, nastoji ponuditi čistu i bezvremensku zabavu koju Peg Solitaire nudi, ali s modernim obratom koji osigurava širu dostupnost i interaktivno iskustvo.

# Opis strukture programa

## 3.1. Klase aplikacije

1. **MainWindow**:

*MainWindow* je ključna klasa aplikacije, jer predstavlja središte interakcije s korisnikom. Kroz ovu klasu odvijaju se svi vizualni elementi igre i korisnički ulazi/izlazi.

*MainWindow* komunicira s *GameLogic* klasom kako bi interpretirao korisničke akcije u kontekstu pravila igre. Također, surađuje s *Cell* klasom za vizualizaciju svake pojedinačne ćelije i s *DatabaseHelper* klasom za zapisivanje rezultata.

* **Ključne funkcionalnosti**:
  + Inicijalizacija ploče (**InitializeBoard()**)
  + Reagiranje na korisnikov odabir ćelija (**Cell\_Clicked()**)
  + Provjera završetka igre (**CheckGameEnd()**)
  + Zapisivanje trajanja igre (**RecordGameTime()**)

1. **GameLogic**:

*GameLogic* je esencijalna klasa koja čuva pravila Peg Solitaire igre. Ova klasa osigurava da se svaki potez odvija u skladu s pravilima igre.

*GameLogic* koristi informacije iz *Cell* klase kako bi odredio valjanost svakog poteza.

* **Ključne funkcionalnosti**:
  + Provjera valjanosti poteza (**IsValidMove()**)
  + Izvršavanje poteza (**MakeMove()**)
  + Detekcija mogućih poteza (**AreMovesPossible()**)

1. **Cell**:

*Cell* klasa predstavlja osnovnu jedinicu igre. Svaka ćelija može biti prazna, sadržavati peg ili biti neaktivna. Ova klasa omogućuje vizualnu reprezentaciju i logiku svake ćelije.

*Cell* klasa koristi se unutar *MainWindow* i *GameLogic* klasa za interpretaciju i vizualizaciju igre.

* **Ključne funkcionalnosti i svojstva**:
  + Praćenje promjena u svojstvima ćelije (**OnPropertyChanged()**)
  + Reprezentacija stanja, boje i pozicije ćelije (**State**, **Color**, **Position**)

1. **DatabaseHelper**:

*DatabaseHelper* omogućava interakciju s bazom podataka. Omogućuje pohranu, dohvaćanje i manipulaciju podacima igre, poput rezultata i vremena igre.

*DatabaseHelper* koristi se unutar *MainWindow* klase kako bi se zabilježili rezultati igre.

* **Ključne funkcionalnosti**:
  + Inicijalizacija veze s bazom (**DatabaseHelper()**)
  + Pohrana rezultata (**AddScore()**)
  + Dohvaćanje rezultata (**GetScores()**)

## Korištene biblioteke

Jedna od ključnih biblioteka koje je ovaj projekat inkorporirao je SQLite. SQLite je izuzetno lagana baza podataka koja je posebno dizajnirana za integraciju unutar aplikacija. Glavna prednost SQLite-a je što ne zahtijeva konfiguraciju servera niti složenu instalaciju. Umjesto toga, svi podaci se čuvaju unutar jedne datoteke koja je lako prenosiva.

U kontekstu ove aplikacije, SQLite se koristi za pristup informacijama o igri. Specifično, čuva se povijest rezultata igrača, kao što su broj poteza koji su potrebni za dovršetak igre i ukupno trajanje igre. Implementacija SQLite-a omogućava brzi i pouzdani pristup tim podacima bez potrebe za teškim i složenim sustavima upravljanja bazama podataka.

Osim što pruža jednostavnost i efikasnost, SQLite donosi i dodatnu fleksibilnost. Na primjer, ako bi se u budućnosti željela proširiti funkcionalnost aplikacije da uključuje višekorisnički mod ili online rangiranje, SQLite bi se lako mogao zamijeniti složenijom bazom podataka ili se čak koristiti u kombinaciji s drugim rješenjima. U ovom trenutnom obliku, SQLite se pokazao kao idealan izbor za potrebe aplikacije, pružajući ravnotežu između jednostavnosti i funkcionalnosti.

## UML Dijagrami

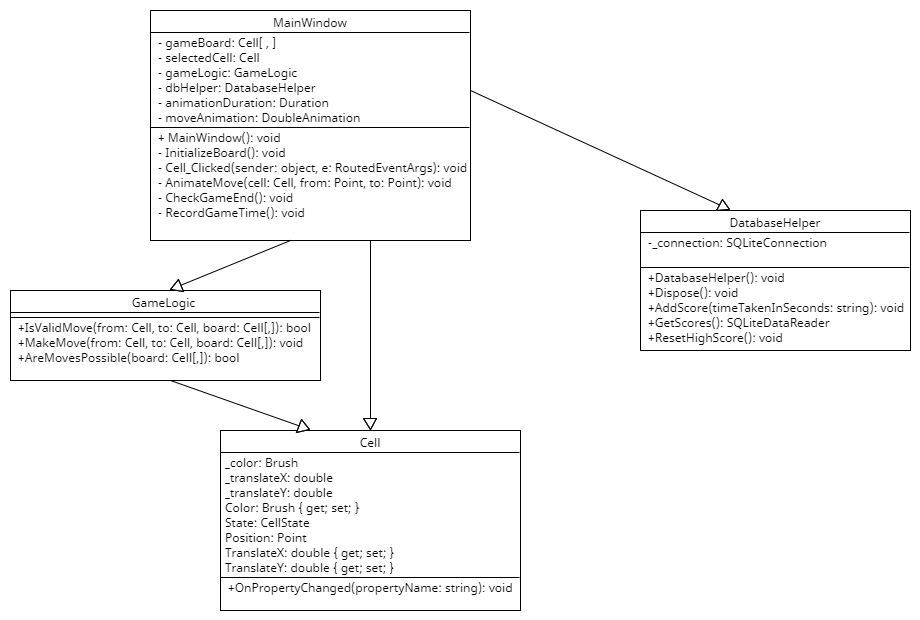
UML (Unified Modeling Language) dijagrami pružaju grafički prikaz strukture, ponašanja, komunikacije i interakcije unutar sistema. U ovom projektu, koristili smo četiri različita UML dijagrama kako bismo precizno predstavili funkcionalnosti i strukturu igre Peg Solitaire.

### Dijagram klasa

Dijagram klasa ilustrira strukturu klasa unutar aplikacije i njihove međusobne odnose. Slijede ključne klase i njihove funkcionalnosti:

* **MainWindow**: Srž aplikacije, ova klasa je zadužena za korisničko sučelje i integrira sve ključne komponente igre. Uz prikaz igre, upravlja interakcijama korisnika, animacijama, provjerama završetka igre i pohranom rezultata.
* **GameLogic**: Fokusirana na pravila i logiku igre, ova klasa provodi valjanost poteza i obnavlja stanje ploče nakon svakog poteza.
* **Cell**: Predstavlja osnovnu jedinicu igre - ćeliju ili "peg". Sadrži informacije o trenutnom stanju i omogućuje dinamičke promjene kao što su kretanja i ažuriranja.
* **DatabaseHelper**: Ova klasa omogućava interakciju s SQLite bazom podataka. Pruža funkcionalnosti kao što su dodavanje novog rezultata ili dohvaćanje prethodnih rezultata.

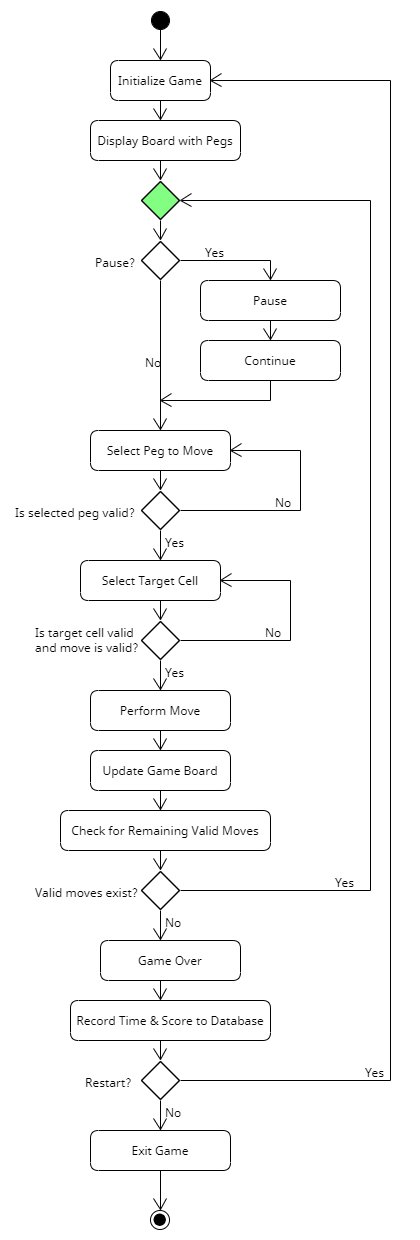
Nadalje, dijagram klasa pokazuje kako su klase međusobno povezane. Primjerice, **MainWindow** koristi funkcionalnosti iz klase **GameLogic**, **Cell** i **DatabaseHelper**.



. Dijagram klasa

### Dijagram aktivnosti

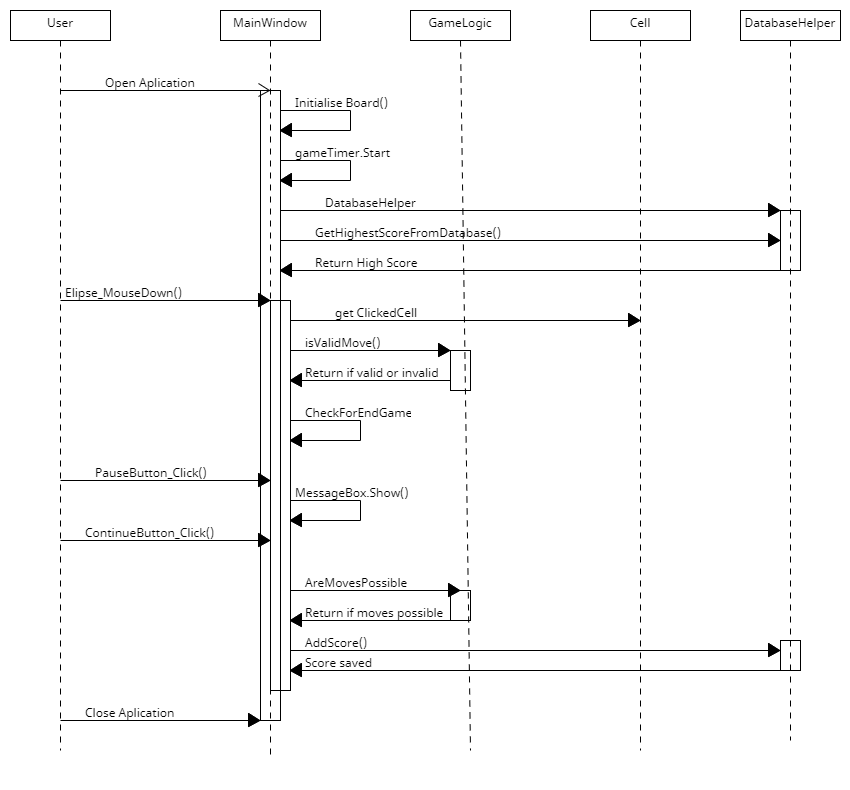
Dijagram aktivnosti prikazuje tok igre Peg Solitaire. Od inicijalizacije igre, odabira pega za pomicanje, provjere valjanosti poteza, sve do završetka igre i pohrane rezultata. Ovaj dijagram služi kao vodič kroz osnovne korake igrača i njihove moguće odluke tijekom igre.



. Dijagram Aktivnosti

### 3.3.3. Dijagram slijeda

Dijagram slijeda se koristi za prikaz komunikacije između različitih klasa tijekom igre. To pomaže razumjeti kako se informacije prenose i obrađuju između glavnih komponenti aplikacije. U kontekstu igre Peg Solitaire, dijagram slijeda ilustrira interakciju između glavne prozorske klase (**MainWindow**), logike igre (**GameLogic**), osnovne jedinice igre (**Cell**) i klase za pomoć s bazom podataka (**DatabaseHelper**).



. Dijagram Slijeda

# 4. Upute za buildanje programa

Prije nego što započnete s buildanjem programa, važno je osigurati da su sve potrebne biblioteke i alati pravilno instalirani i konfigurirani. Posebnu pažnju treba obratiti na nestandardne biblioteke koje naša aplikacija koristi, poput SQLite-a, kako biste osigurali nesmetan rad i optimalne performanse.

**Instalacija SQLite-a**:

1. **Preuzimanje**:
   * Posjetite web stranicu SQLite-a.
   * Odaberite odgovarajući paket za preuzimanje koji je kompatibilan s vašim operativnim sustavom (Windows, macOS, Linux itd.).
   * Preuzmite željeni paket i spremite ga na poznatu lokaciju na vašem računalu.
2. **Instalacija**:
   * Navigirajte do prethodno spremljene lokacije preuzimanja i pokrenite instalacijski program.
   * Slijedite upute koje se prikazuju na ekranu. Defaultne postavke bit će sasvim prikladne.
   * Kada završi instalacija, možda ćete trebati ponovno pokrenuti svoje razvojno okruženje ili čak računalo kako biste osigurali pravilno prepoznavanje i funkcionalnost SQLite-a.
3. **Verifikacija**:
   * Da biste provjerili je li SQLite ispravno instaliran, otvorite terminal ili naredbeni redak i upišite: **sqlite3 --version**. Trebali biste vidjeti verziju SQLite-a koju ste upravo instalirali.
   * Ako se pojave bilo kakve greške ili ako ne možete dobiti verziju, ponovno provjerite korake instalacije ili potražite online pomoć i resurse vezane za SQLite.

Nakon uspješne instalacije SQLite-a, možete nastaviti s buildanjem i pokretanjem programa Peg Solitaire.

# 5. Osnovne upute za korištenje programa

Korištenje programa "Peg Solitaire" koncipirano je tako da bude što intuitivnije, omogućujući korisnicima svih nivoa tehničkog znanja da se lako snađu. Evo kako da maksimalno uživate u iskustvu igranja:

1. **Pokretanje Igre**:
2. **Odabir Pega za Pomicanje**:
   * Na igračkoj ploči prikazat će se različiti pegovi. Da biste odabrali peg kojim želite napraviti potez, jednostavno kliknite na njega. Kada je peg odabran, trebao bi biti vizualno istaknut ili označen kako biste znali da je spreman za pomicanje.
3. **Odabir ciljanog polja za odabrani peg**:
   * Nakon što ste odabrali željeni peg, trebate odabrati ciljno polje na koje želite da ga premjestite. Kliknite na ciljno polje. Ako je polje dostupno i pomicanje je validno, peg će automatski preći na odabranu poziciju.
4. **Pravila Pomicanja**:
   * Peg se može pomicati samo ako postoji susjedni peg između odabranog pega i ciljnog polja, a ciljno polje mora biti prazno. Susjedni peg će biti uklonjen s ploče prilikom pomicanja.
5. **Cilj Igre**:
   * Glavni cilj "Peg Solitaire" igre je da kroz seriju poteza ostavite samo jedan peg na ploči. Ako ste uspjeli smanjiti broj pegova na jedan, uspješno ste završili igru.
6. **Ponovno Pokretanje ili Prekid Programa**:
   * Ako želite ponovno pokrenuti igru i pokušati ispočetka, postoji opcija "Restart" unutar sučelja programa. Ako želite prekinuti igru i zatvoriti program, zatvorite prozor ili koristite odgovarajuću komandu unutar programa.

Dok igrate, preporučujemo da se povremeno odmorite i razmislite o sljedećim koracima kako biste poboljšali svoju strategiju. Uživajte u izazovu!

# 6. Zaključak

Projekt "Peg Solitaire" digitalna je inačica tradicionalne pločne igre, prilagođena suvremenom okruženju. Dokumentacija je omogućila detaljan prikaz svih bitnih elemenata projekta. Kroz ovu analizu, dobio se jasan uvid u kako program funkcionira i kako ga koristiti. Ideja ovog projekta bila je spajanje tradicionalnog i modernog kroz igru, a kroz ovu dokumentaciju dano je sve potrebno za razumijevanje njegove izvedbe.

# Kod

### MainWindow.xaml

<!-- Deklaracija glavnog prozora za igru Peg Solitaire -->

<Window x:Class="PegSolitaire.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Peg Solitaire" Height="350" Width="300">

<!-- Osnovna mreža (layout) za organizaciju sadržaja -->

<Grid>

<!-- Vertikalni StackPanel za organizaciju sadržaja -->

<StackPanel VerticalAlignment="Center" HorizontalAlignment="Center">

<!-- Oznaka (Label) za prikaz proteklog vremena igre -->

<Label x:Name="timeLabel" Content="00:00:00" HorizontalAlignment="Center" VerticalAlignment="Top" Margin="10" FontSize="16" FontWeight="Bold"/>

<!-- Kontrola ploče za igru -->

<ItemsControl x:Name="BoardItemsControl" ItemsSource="{Binding Cells}">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<!-- Definiranje 7x7 mreže za izgled ploče za igru -->

<UniformGrid Rows="7" Columns="7"/>

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

<!-- Predložak podataka za svaku ćeliju -->

<ItemsControl.ItemTemplate>

<DataTemplate>

<!-- Oblik elipse predstavlja svaki peg; njegova boja mijenja se prema stanju igre -->

<Ellipse x:Name="PegEllipse" Fill="{Binding Color}" Width="30" Height="30" Margin="5" MouseDown="Ellipse\_MouseDown">

<Ellipse.RenderTransform>

<TranslateTransform X="{Binding TranslateX}" Y="{Binding TranslateY}"/>

</Ellipse.RenderTransform>

</Ellipse>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

<!-- Horizontalni StackPanel za tipke pauze i nastavka -->

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" Margin="5">

<Button x:Name="pauseButton" Content="Pause" Click="PauseButton\_Click" Margin="5"/>

<Button x:Name="continueButton" Content="Continue" Click="ContinueButton\_Click" Margin="5" Visibility="Collapsed"/>

</StackPanel>

<!-- Oznaka za prikaz najboljeg rezultata -->

<Label x:Name="highScoreLabel" Content="High Score: " HorizontalAlignment="Center" VerticalAlignment="Top" Margin="10" FontSize="16" FontWeight="Bold"/>

</StackPanel>

</Grid>

</Window>

### MainWindow.xaml.cs

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Shapes;

using System.Windows.Threading;

namespace PegSolitaire

{

public partial class MainWindow : Window

{

// Listi ćelija na ploči za igru

public List<Cell> Cells { get; set; }

// 2D niz koji predstavlja ploču igre

private Cell[,] BoardArray;

// Trenutno odabrana ćelija (peg)

private Cell SelectedCell;

// Instanca logike igre

private GameLogic Logic = new GameLogic();

// Štoperica za mjerenje proteklog vremena igre

private System.Diagnostics.Stopwatch stopwatch;

// Indikator da li je timer pokrenut

private bool isTimerRunning = true;

// Timer igre

private DispatcherTimer gameTimer;

public MainWindow()

{

InitializeComponent();

InitializeBoard();

DataContext = this;

// Inicijalizacija i pokretanje štoperice

stopwatch = new System.Diagnostics.Stopwatch();

stopwatch.Start();

// Postavljanje timera

gameTimer = new DispatcherTimer();

gameTimer.Interval = TimeSpan.FromSeconds(1);

gameTimer.Tick += GameTimer\_Tick;

gameTimer.Start();

// Dohvaćanje najboljeg rezultata iz baze podataka

DatabaseHelper dbHelper = new DatabaseHelper();

string highScore = GetHighestScoreFromDatabase();

highScoreLabel.Content = $"High Score: {highScore}";

}

// Ažuriranje prikaza vremena svake sekunde

private void GameTimer\_Tick(object sender, EventArgs e)

{

timeLabel.Content = stopwatch.Elapsed.ToString(@"hh\:mm\:ss");

}

// Pauziranje igre

private void PauseButton\_Click(object sender, RoutedEventArgs e)

{

// Ako je igra trenutno aktivna, pauzira se

if (isTimerRunning)

{

// Pauziranje štoperice i timera

stopwatch.Stop();

isTimerRunning = false;

gameTimer.Stop();

// Poruka s opcijom nastavka igre

MessageBoxResult result = MessageBox.Show("Game paused. Do you want to continue?", "Pause", MessageBoxButton.YesNo, MessageBoxImage.Question);

if (result == MessageBoxResult.Yes)

{

// Nastavak štoperice i timera

stopwatch.Start();

gameTimer.Start();

isTimerRunning = true;

}

else

{

continueButton.Visibility = Visibility.Visible;

}

}

}

// Nastavak igre

private void ContinueButton\_Click(object sender, RoutedEventArgs e)

{

if (!isTimerRunning)

{

// Nastavak štoperice i timera

stopwatch.Start();

gameTimer.Start();

isTimerRunning = true;

continueButton.Visibility = Visibility.Collapsed;

}

}

// Inicijalizacija ploče za igru

private void InitializeBoard()

{

Cells = new List<Cell>();

BoardArray = new Cell[7, 7];

for (int i = 0; i < 7; i++)

{

for (int j = 0; j < 7; j++)

{

Cell cell;

if ((i < 2 && j < 2) || (i < 2 && j > 4) || (i > 4 && j < 2) || (i > 4 && j > 4))

cell = new Cell { Color = Brushes.Gray, State = Cell.CellState.Inactive, Position = new Point(j, i) };

else if (i == 3 && j == 3)

cell = new Cell { Color = Brushes.White, State = Cell.CellState.Empty, Position = new Point(j, i) };

else

cell = new Cell { Color = Brushes.Blue, State = Cell.CellState.Peg, Position = new Point(j, i) };

Cells.Add(cell);

BoardArray[i, j] = cell;

}

}

BoardItemsControl.ItemsSource = null;

BoardItemsControl.ItemsSource = Cells;

}

// Event handler za klik na peg

private void Ellipse\_MouseDown(object sender, MouseButtonEventArgs e)

{

Ellipse ellipse = sender as Ellipse;

Cell clickedCell = ellipse.DataContext as Cell;

if (SelectedCell == null)

{

if (clickedCell.State == Cell.CellState.Peg)

{

SelectedCell = clickedCell;

SelectedCell.Color = Brushes.Red; // Highlight the selected peg.

}

}

else

{

if (SelectedCell == clickedCell) // Deselecting the peg

{

SelectedCell.Color = Brushes.Blue;

SelectedCell = null;

}

else if (Logic.IsValidMove(SelectedCell, clickedCell, BoardArray))

{

Logic.MakeMove(SelectedCell, clickedCell, BoardArray);

SelectedCell = null;

CheckForEndGame();

}

else

{

SelectedCell.Color = Brushes.Blue; // Reset the color of the previously selected peg.

SelectedCell = null;

}

}

}

private int CountRemainingPegs()

{

int count = 0;

for (int y = 0; y < 7; y++)

{

for (int x = 0; x < 7; x++)

{

if (BoardArray[y, x].State == Cell.CellState.Peg)

{

count++;

}

}

}

return count;

}

private void CheckForEndGame()

{

if (!Logic.AreMovesPossible(BoardArray))

{

stopwatch.Stop(); // Stop timer

int remainingPegs = CountRemainingPegs();

if (remainingPegs == 1 && BoardArray[3, 3].State == Cell.CellState.Peg)

{

MessageBox.Show("Congratulations! You won!", "End of Game", MessageBoxButton.OK, MessageBoxImage.Asterisk);

var timeTaken = stopwatch.Elapsed.TotalSeconds;

DatabaseHelper dbHelper = new DatabaseHelper();

dbHelper.AddScore(timeTaken.ToString()); // Save the score only if the user wins

}

else

{

MessageBox.Show($"Game Over! You have {remainingPegs} pegs remaining.", "End of Game", MessageBoxButton.OK, MessageBoxImage.Exclamation);

}

var result = MessageBox.Show("Would you like to restart?", "Restart Game", MessageBoxButton.YesNo, MessageBoxImage.Question);

if (result == MessageBoxResult.Yes)

{

InitializeBoard();

stopwatch.Restart(); // Restart the stopwatch for the new game

isTimerRunning = true;

continueButton.Visibility = Visibility.Collapsed;

// Refresh the high score

string highScore = GetHighestScoreFromDatabase();

highScoreLabel.Content = $"High Score: {highScore}";

}

}

}

private string GetHighestScoreFromDatabase()

{

using (var dbHelper = new DatabaseHelper())

{

var scoresReader = dbHelper.GetScores();

double? bestScore = null;

while (scoresReader.Read())

{

double currentScore;

if (double.TryParse(scoresReader["TimeTaken"].ToString(), out currentScore))

{

if (!bestScore.HasValue || currentScore < bestScore.Value)

{

bestScore = currentScore;

}

}

}

return bestScore.HasValue ? FormatTime((int)bestScore.Value) : "N/A";

}

}

private string FormatTime(int seconds)

{

TimeSpan timeSpan = TimeSpan.FromSeconds(seconds);

return timeSpan.ToString(@"hh\:mm\:ss");

}

}

}

### Cell.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Media;

using System.Windows;

namespace PegSolitaire

{

public class Cell : INotifyPropertyChanged

{

// Enumeracija za različite stanja ćelije

public enum CellState

{

Empty, // Prazna ćelija

Peg, // ćelija s peg-om

Inactive // Neaktivna ćelija

}

// Privatna varijabla za boju ćelije

private Brush \_color;

// Javna svojstva za boju ćelije koja koristi INotifyPropertyChanged za obavijest promjena

public Brush Color

{

get => \_color;

set

{

\_color = value;

OnPropertyChanged(nameof(Color));

}

}

// Trenutno stanje (Empty, Peg, Inactive)

public CellState State { get; set; }

// Pozicija na ploči

public Point Position { get; set; }

// Event koji se okida kada se neko svojstvo promijeni

public event PropertyChangedEventHandler PropertyChanged;

// Metoda koja okida PropertyChanged event

protected virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

// Privatna varijabla za horizontalno pomicanje ćelije

private double \_translateX;

// Javna svojstva za horizontalno pomicanje ćelije koja koristi INotifyPropertyChanged za obavijest promjena

public double TranslateX

{

get => \_translateX;

set

{

\_translateX = value;

OnPropertyChanged(nameof(TranslateX));

}

}

// Privatna varijabla za vertikalno pomicanje ćelije

private double \_translateY;

// Javna svojstva za vertikalno pomicanje ćelije koja koristi INotifyPropertyChanged za obavijest promjena

public double TranslateY

{

get => \_translateY;

set

{

\_translateY = value;

OnPropertyChanged(nameof(TranslateY));

}

}

}

}

### GameLogic.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace PegSolitaire

{

public class GameLogic

{

// Provjerava je li potez s 'from' na 'to' valjan prema pravilima Peg Solitaire igre.

public bool IsValidMove(Cell from, Cell to, Cell[,] board)

{

// Ako početna ćelija nema peg ili ciljna ćelija nije prazna, potez nije valjan

if (from.State != Cell.CellState.Peg || to.State != Cell.CellState.Empty)

return false;

// Izračunava razliku u X i Y koordinatama između početne i ciljne ćelije

int dx = (int)(to.Position.X - from.Position.X);

int dy = (int)(to.Position.Y - from.Position.Y);

// Provjerava je li pomicanje dvije ćelije u vodoravnom ili okomitom smjeru

if ((Math.Abs(dx) == 2 && dy == 0) || (dx == 0 && Math.Abs(dy) == 2))

{

// Pronalazi ćeliju između početne i ciljne ćelije

Cell midCell = board[(int)(from.Position.Y + dy / 2), (int)(from.Position.X + dx / 2)];

// Potez je valjan ako ćelija između ima peg

return midCell.State == Cell.CellState.Peg;

}

return false;

}

// Izvršava potez od 'from' do 'to' i ažurira stanje ploče

public void MakeMove(Cell from, Cell to, Cell[,] board)

{

// Izračunava razliku u X i Y koordinatama između početne i ciljne ćelije

int dx = (int)(to.Position.X - from.Position.X);

int dy = (int)(to.Position.Y - from.Position.Y);

// Pronalazi ćeliju između početne i ciljne ćelije

Cell midCell = board[(int)(from.Position.Y + dy / 2), (int)(from.Position.X + dx / 2)];

// Ažurira stanje ćelija

from.State = Cell.CellState.Empty;

midCell.State = Cell.CellState.Empty;

to.State = Cell.CellState.Peg;

// Ažurira boje ćelija

from.Color = Brushes.White;

midCell.Color = Brushes.White;

to.Color = Brushes.Blue;

}

// Provjerava postoji li moguć potez na ploči

public bool AreMovesPossible(Cell[,] board)

{

for (int y = 0; y < 7; y++)

{

for (int x = 0; x < 7; x++)

{

Cell currentCell = board[y, x];

// Ako trenutna ćelija ima peg, provjerava se svaki od mogućih smjerova skakanja: gore, dolje, lijevo, desno

if (currentCell.State == Cell.CellState.Peg)

{

foreach (var direction in new[] { (0, -2), (0, 2), (-2, 0), (2, 0) })

{

int newY = y + direction.Item2;

int newX = x + direction.Item1;

// Ako su nove koordinate unutar granica ploče

if (newY >= 0 && newY < 7 && newX >= 0 && newX < 7)

{

Cell targetCell = board[newY, newX];

// Ako je moguć valjan potez, igra nije gotova

if (IsValidMove(currentCell, targetCell, board))

{

return true;

}

}

}

}

}

}

// Ako je petlja završila bez povratne vrijednosti, nema preostalih poteza

return false;

}

}

}

DatabaseHelper.cs  
using System;

using System.Data.SQLite;

using System.IO;

public class DatabaseHelper : IDisposable

{

private SQLiteConnection \_connection;

// Konstruktor koji inicijalizira ili otvara bazu podataka

public DatabaseHelper()

{

const string DbName = "PegSolitaire.db";

// Provjera postoji li baza podataka

if (!File.Exists(DbName))

{

// Ako ne postoji, kreiraj novu bazu podataka

SQLiteConnection.CreateFile(DbName);

\_connection = new SQLiteConnection($"Data Source={DbName};Version=3;");

\_connection.Open();

// Kreiraj tabelu za pohranu rezultata

using (var command = new SQLiteCommand("CREATE TABLE Scores (Date TEXT, TimeTaken TEXT)", \_connection))

{

command.ExecuteNonQuery();

}

}

else

{

// Ako baza podataka već postoji, samo je otvori

\_connection = new SQLiteConnection($"Data Source={DbName};Version=3;");

\_connection.Open();

}

}

// Implementacija IDisposable sučelja za čišćenje resursa

public void Dispose()

{

\_connection?.Dispose();

}

// Dodaje novi rezultat u bazu podataka

public void AddScore(string timeTakenInSeconds)

{

// Upit za umetanje novog reda u tabelu s trenutnim datumom i vremenom

using (var command = new SQLiteCommand($"INSERT INTO Scores (Date, TimeTaken) VALUES (DateTime('now'), '{timeTakenInSeconds}')", \_connection))

{

command.ExecuteNonQuery();

}

}

// Dohvaća sve rezultate iz baze podataka

public SQLiteDataReader GetScores()

{

using (var command = new SQLiteCommand("SELECT \* FROM Scores", \_connection))

{

return command.ExecuteReader();

}

}

// Briše sve rezultate iz baze podataka

public void ResetHighScore()

{

using (var command = new SQLiteCommand("DELETE FROM Scores", \_connection))

{

command.ExecuteNonQuery();

}

}

}