קריית חינוך אורט יד-לבוביץ' נתניה

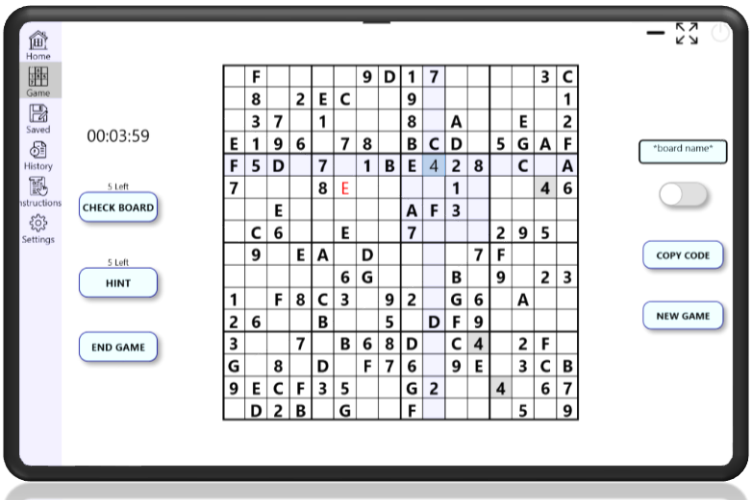
ליזום .להנהיג .להשפיע

סמל ביה"ס: 470096

תכנות מונחה אירועים

פרוייקט מסכם בנושא:

סודוקו



שם התלמיד : איתן אמיר

תעודת זהות : 333183903

כיתה : י17

מורה מלווה : שמעון מכלוף

יוני, 2024, סיון, תשפ"ד

תוכן עניינים

מבוא

מטרות התוכנה

שיפור יכולת החשיבה:

פתרון סודוקו מפתח את החשיבה הלוגית ואת מיומנויות פתרון הבעיות. התכנה מאפשרת למשתמשים לתרגל ולשפר את כישוריהם האנליטיים והלוגיים, מה שיכול להועיל בתחומים רבים אחרים בחיים.

שדרוג חווית פתירת הסודוקו מפתירה על נייר על נייר :
אפליקצית הסודוקו מאפשרת אמצאים ויזואליים העוזרים למשתמש ומשפרים את נוחות פתירת הסודוקו מבלי להגביל את היצירתיות ואת קושי הפאזל. כגון: סימוני תאים באותו טור, שורה או קופסא וסימוני תאים עם טקסט זהה.

חידוש ביחס לאפליקציות אחרות :

אפליקציית הסודוקו שלי מאפשרת יכולות שאפליקציות סודוקו הקיימות בשוק לא מאפשרות, כמו הגדלת כמות התאים בלוח. בנוסף, האפליקציה שלי נותנת לשחקן את החופש לפתור הבעיה בעצמו, כמו חידה שאף אחד לא פתר לפניו (מזכיר פתרון על דף) לעומת אפליקציות רבות המגבילות את המשתמש מלמצוא פתרון משלו ובכך מגבילות את היצירתיות.

בידור ופנאי:

תכנת הסודוקו מעוצבת בצורה המספקת חוויה מהנה ומאתגרת למשתמש. היא משמשת כאמצעי להעברת זמן באופן מהנה, עם מגוון רחב של רמות קושי המתאימות למשתמשים מתחילים ומתקדמים כאחד.

פיתוח אפליקציה מסחרית :

תכנה זו יכולה להיות הבסיס לאפליקציה מסחרית לציבור הרחב הכוללת פונקציות נוספות כמו תחרויות, דירוגים ורשתות חברתיות.

שיפור יכולות התכנות:

פיתוח תכנת סודוקו דורש ידע נרחב בתכנות, כולל תכנון והקמת ממשק משתמש (UI design), תכנות מונחה עצמים (OOP), מבני נתונים, ושימוש במסדי נתונים. הפרויקט יכול לשמש כתירגול מצוין לכתיבת קוד יעיל, קריא ונקי, ולשיפור המיומנויות הטכניות של המפתחים.

יצירת כלים חינוכיים:

תכנת הסודוקו יכולה לשמש ככלי חינוכי ללימוד מתמטיקה, לוגיקה וחשיבה ביקורתית בבתי ספר ובמסגרות לימוד אחרות. היא מספקת תרגול מעשי בניתוח בעיות ופתרונן, ועוזרת לתלמידים לפתח כישורים חשובים בצורה מהנה ומעשירה.

השגת מטרות אלו דורשת השקעה בממשק משתמש ידידותי, בבנייה של לוחות סודוקו מאתגרים ומגוונים, ובהוספת תכונות מתקדמות המעשירות את חווית המשתמש ומגדילות את הערך החינוכי והתפעולי של התכנה.

קהל היעד

קהל היעד של התכנה הוא אנשים הנהנים ממשחקי חשיבה ומחפשים דרך נוחה ופשוטה להפעיל את המוח במהלך היום.

כמו כן, קהל היעד הוא אנשים בכל גיל המאפשר להם שימוש בתכנה - בן דודי בין ה 5 נהנה לשחק בתכנה ואחרי שלימדתי אותו את החוקים, פתר לוחות של 2*2 ברמה בינונית. כמו כן, גם סבתי נהנתה מהתכנה והסתגלה למעבר מפתירה על נייר לפתירה במחשב.

טכנולוגיה

השתמשתי בטכנולוגיה של WPF לפיתוח האפליקציה שלי, שמספקת עיצוב עשיר ויכולות גרפיות מתקדמות בסביבת .NET. ב WPF,-הממשק המשתמש מוגדר באמצעות קבצי XAML (Extensible Application Markup Language), שמאפשרים לי לבנות ולערוך את ה UI-של היישום בצורה ברורה ומבלי לדרוש ידע עמוק בתכנות.

הטכנולוגיה מאפשרת לי ליצור אנימציות ואובייקטים גרפיים מתקדמים, שמשפרים את חוויית המשתמש ומעניקים מראה ותחושה מודרניים ליישום. כמו כן WPF, מספקת תמיכה מובנית בניהול קשרי נתונים, שמאפשרת לי לקשר ולהציג מידע ממקורות שונים כגון מסדי נתונים וקבצי XML בצורה יעילה ונוחה לתחזוקה.

בנוסף, בתכנות מונחה אירועים (Event-Driven Programming), הפעולות ביישום מופעלות בתגובה לאירועים שונים שקורים בממשק המשתמש. כלומר, במקום להריץ קוד בצורה רציפה, היישום יגיב לאירועים כגון לחיצת משתמש על כפתור, שינוי בערך בתיבת טקסט, או כל אירוע אחר שמופעל במהלך השימוש ביישום. זה מאפשר לי לתכנן את היישום כך שהתגובה לפעולות המשתמש תהיה מהירה ויעילה.

השימוש ב WPF-בשילוב עם תכנות מונחה אירועים איפשר לי ליצור UI מודרני ואפקטיבי בצורה יעילה ומותאמת אישית.

תיאור המשחק / לומדה:

משחק הסודוקו שלי הוא אפליקציה מקיפה שמטרתה לספק חוויית סודוקו עשירה ומותאמת אישית לשחקנים בכל הרמות. המשחק כולל מספר עמודים עם הוראות מפורטות, מצבי משחק שונים ותכונות ידידותיות למשתמש המשפרות את חוויית המשחק.

התאמת מימדי לוח אישית:

שחקנים יכולים לבחור את רוחב וגובה המשבצות הגדולות, מה שמאפשר מגוון גדלי רשת וקונפיגורציות. לדוגמה, ניתן לקבל לוח 4*4 עם קופסאות 2*2, לוח 6*6 עם קופסאות 2*3, או כל שילוב אחר שבו גודל הלוח הכולל הוא מכפלת מימדי הקופסא.

עמודי תפריט:

עמוד המשחק: המקום שבו השחקנים יכולים להתחיל משחק חדש עם מימדי רשת ורמות קושי מותאמים אישית. העמוד כולל תכונות כמו רמזים, ביטול וניהול הערות כדי לסייע לשחקנים.

עמוד השמורים: מציג את כל המשחקים השמורים, ומאפשר לשחקנים להמשיך כל משחק מהנקודה בה הפסיקו.

עמוד ההיסטוריה: מציג לוג של המשחקים שהושלמו, כולל ניקוד, זמני השלמה ורמות קושי.

עמוד ההגדרות: מאפשר התאמה אישית של מראה והתנהגות האפליקציה, כמו הפעלת וכיבוי צלילים ומוזיקה, ושינוי ערכות נושא.

משוב קולי

המשחק מספק משוב קולי עבור פעולות שונות. לדוגמה, השמעת צליל כאשר מתבצעת הזנה חוקית או צליל שונה כאשר הזנה לא חוקית חורגת מהמכפלה המקסימלית המותרת של מימדי הרשת.

חוקי המשחק / לומדה:

כל שורה: חייבת להכיל את כל הספרות (או האותיות) מהטווח המוגדר, ללא חזרות.

כל עמודה: חייבת להכיל את כל הספרות (או האותיות) מהטווח המוגדר, ללא חזרות.

כל קופסא: חייבת להכיל את כל הספרות (או האותיות) מהטווח המוגדר, ללא חזרות.

כניסה לאפליקציה:

ניתן לראות מסך כניסה עם תמונה ב DOUBLE ANIMATION של סודוקו. (ראה עמוד פתיחה בדפי הפרוייקט)

הניווט מתבצע באמצעות תפריט מבוסס על RADIO BUTTON דרך MainWindow.

כדי לסגור את האפליקציה או לשנות את גודל והופעתו של המסך יצרתתי 3 כפתורים בצידו העליון מימין.

מבנה הנתונים Database

קשרי הגומלין בין הטבלאות:

לא היה צורך בקשרי גומלין בין טבלאות בעבודתי

טבלאות ב-Database:

tbl_games:

בטבלה זו מאוכסנים הנתונים המתארים משחק.

שם שדה	סוג נתונים	תיאור (אופציונלי)
Id	מספור אוטומטי	
Current	כן/לא	was the game running when app was closed
Solved	כן/לא	Was the game solved
Time	טקסט קצר	The time it took to solve or end the game
BoardCode	טקסט ארוך	An encripted code containing the value of each cell and notes
PuzzleCode	טקסט ארוך	An encripted code containing the solved value of each cell and if it was initial
gameName	טקסט קצר	Name of game given by the user
HintsTaken	מספר	Hints taken in the game till end
ChecksTaken	מספר	Check boards taken in the game till the end
BoxHeight	מספר	Height of boxes
BoxWidth	מספר	Width of boxes
GameDate	טקסט קצר	Date of game
DifficultyLevel	טקסט קצר	Difficulty level of game

tbl_settings:

בטבלה זו מאוכסנים נתונים על ההגדרות של המשתמש.

שם שדה	סוג נתונים	
Id	מספור אוטומטי	
SoundOn	כן/לא	
Theme	טקסט קצר	Mame of color theme
MusicOn	כן/לא	
MarkRelated	כן/לא	
MarkSameText	כן/לא	
AllowNotes	כן/לא	

שימוש במסד הנתונים בתכנה

בתחילת ריצת התכנית, התכנה עוברת על כל השורות בטבלת המשחקים במסד הנתונים ויוצרת לכל שורה אובייקט GameInfo כדי להכיל את המידע.

בהתאם להאם המשחק פתור, היא ממינת את ה אובייקטים לרשימה של המשחקים השמורים ולרשימה של המשחקים בהסטוריה:

```
/// <summary>
/// Sets the saved games pages from the database.
/// </summary>
1 reference
private void SetSavedGamesFromDB()
{
    historyPage = new SaverPage(true); // Initialize history page
    savedPage = new SaverPage(false); // Initialize saved games page

    DataTable dt = DBHelper.GetDataTable("SELECT * FROM tbl_games"); // Retrieve games data from database

    foreach (DataRow dr in dt.Rows)
    {
        // Create GameInfo object from database row
        GameInfo gameInfo = new GameInfo(
                            Convert.ToInt32(dr["Id"]),
                            dr["GameName"].ToString(),
                            dr["BoardCode"].ToString(),
                            dr["PuzzleCode"].ToString(),
                            dr["Time"].ToString(),
                            Convert.ToInt32(dr["HintsTaken"]),
                            Convert.ToInt32(dr["ChecksTaken"]),
                            dr["GameDate"].ToString(),
                            (DificultyLevel)Enum.Parse(typeof(DificultyLevel), dr["DifficultyLevel"].ToString()),
                            Convert.ToBoolean(dr["Solved"]),
                            Convert.ToBoolean(dr["Current"]),
                            Convert.ToInt32(dr["BoxHeight"]),
                            Convert.ToInt32(dr["BoxWidth"])
                            );

        // Determine where to add the game info based on solved and current flags
        if (gameInfo.Solved)
        {
            historyPage.AddItemToList(gameInfo); // Add to history page
        }
        else
        {
            if (gameInfo.Current)
            {
                SaverPage.DeleteGameFromDB(gameInfo); // Remove from database if current game
                gamePage = new GamePage(gameInfo); // Initialize game page
                return; // Exit method after initializing game page
            }
            else
            {
                savedPage.AddItemToList(gameInfo); // Add to saved games page
            }
        }
    }
}
```

מתוך MainWindow.xaml.cs

כמו כן, בתחילה גם ההגדרות נקבעות להגדרות השמורות במסד הנתונים בטבלה tbl_settings :

מתוך MainWindow.xaml.cs

```
/// <summary>
/// Sets the application settings from the database.
/// </summary>
1 reference
private void SetSettingsFromDB()
{
    DataTable dt = DBHelper.GetDataTable("SELECT * FROM tbl_settings");
    DataRow dr = dt.Rows[0]; // Get the first row (assuming single settings row)

    // Load settings values from database
    Settings.markSameText = Convert.ToBoolean(dr[DBConstants.Settings_Parameters.SameText]);
    Settings.markRelated = Convert.ToBoolean(dr[DBConstants.Settings_Parameters.MarkRelated]);
    Settings.soundOn = Convert.ToBoolean(dr[DBConstants.Settings_Parameters.SoundOn]);
    Settings.musicOn = Convert.ToBoolean(dr[DBConstants.Settings_Parameters.MusicOn]);

    // Set theme if valid theme found in database
    if (Enum.TryParse(dr[DBConstants.Settings_Parameters.Theme].ToString(),
        out ColorThemes theme))
    {
        Settings.theme = theme;
        ThemeControl.SetColors(Settings.theme); // Apply theme colors
    }
}
```

בעת סגירת תכנה, אם משחק רץ, התכנה מוסיפה אותו למסד הנתונים:

מתוך MainWindow.xaml.cs

```
/// <summary>
/// Closes the application window and performs necessary cleanup.
/// </summary>
1 reference
private void CloseApp_Click(object sender, RoutedEventArgs e)
{
    // Stop music if playing
    if (Settings.musicOn)
    {
        SoundPlayer.StopMusic();
    }

    // Insert current game into database if game page exists
    if (gamePage != null)
    {
        SaverPage.InsertGame(gamePage.GetGameInfo(false, true));
    }

    UpdateSettingsInDB(); // Update settings in database
    Application.Current.Shutdown(); // Shutdown the application
}
```

בעת סיום משחק, התכנה מוסיפה שורה לרשימות בזיכרון התכנה ולמסד הנתונים עם המידע על המשחק:

מתוך Game.cs

```
/// <summary>
/// Ends the current Sudoku game session.
/// </summary>
/// <param name="isSolved">Indicates whether the game is solved.</param>
/// <param name="toSave">Indicates whether to save the game.</param>
7 references
public void End(bool isSolved, bool toSave)
{
    this.timer.Stop();
    this.inProgress = false;

    var window = (MainWindow)Application.Current.MainWindow;

    if (window.gamePage != null)// wasnt already ended
    {
        if (toSave)
        {
            GameInfo gameInfo = window.gamePage.GetGameInfo(isSolved, false);
            if (isSolved)
            {
                window.historyPage.AddItemToListAndDB(gameInfo);
            }
            else
            {
                window.savedPage.AddItemToListAndDB(gameInfo);
            }
        }

        window.gamePage.DisableGameControls();
        window.gamePage = null;
    }

}
```

בסגירה התכנה גם מעדכנת את ההגדרות

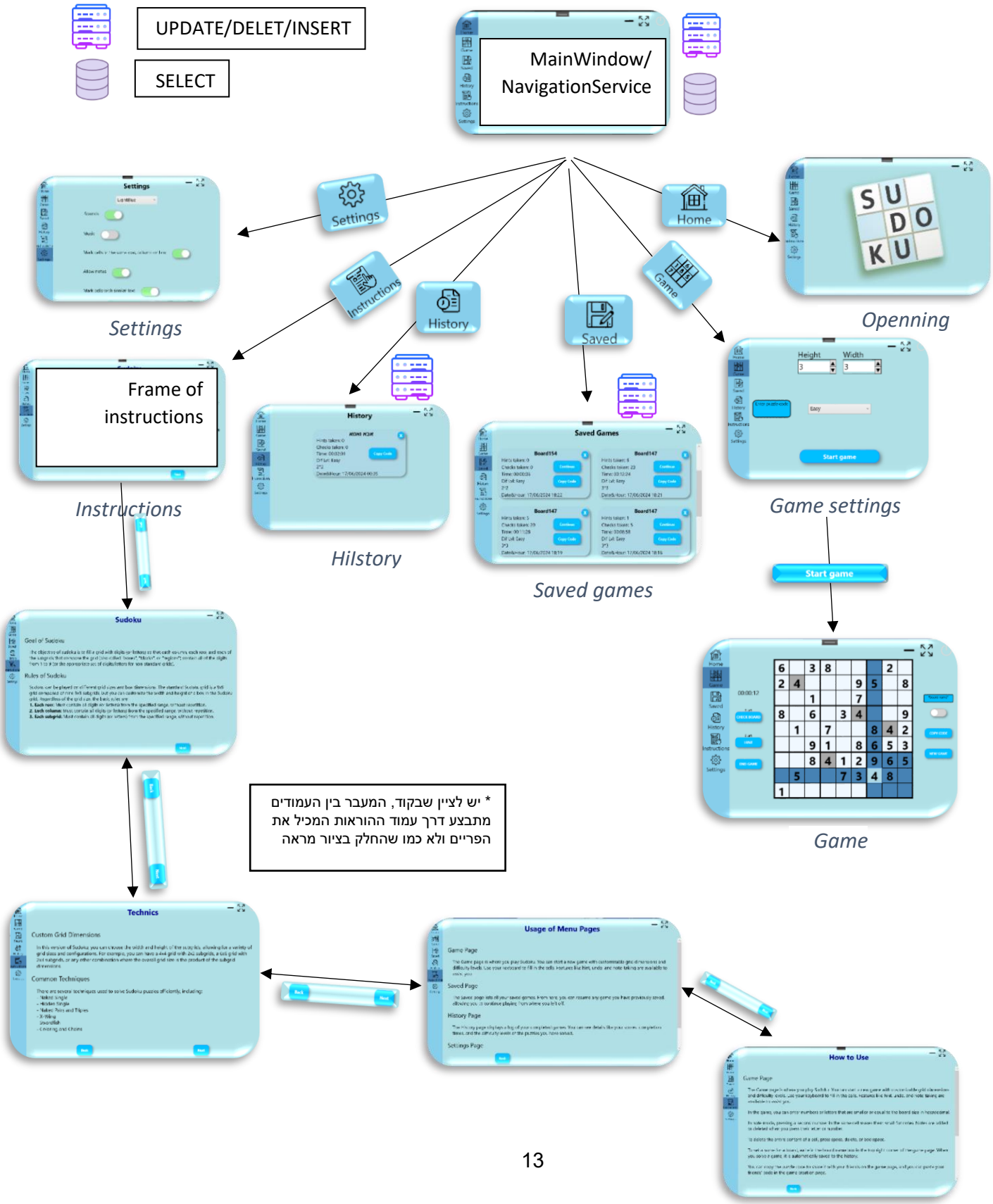
```
public void UpdateSettingsInDB()
{
    string sqlStmt = DBConstants.UpdateSettingsQuary; // SQL statement to update settings

    // Define parameters for the SQL statement
    OleDbParameter[] parameters =
    {
        new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.SameText, Settings.markSameT
        new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.MarkRelated, Settings.markRe
        new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.SoundOn, Settings.soundOn),
        new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.MusicOn, Settings.musicOn),
        new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.Theme, Settings.theme.ToStri
        new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.AllowNotes, Settings.allowNo
    };

    DBHelper.ExecuteCommand(sqlStmt, parameters); // Execute the database command with parameters
}
```

קוד עבור מסד הנתונים

```
using System;
using System.Data;
using System.Data.OleDb;

namespace DAL
{
    public class DBHelper
    {
        /// <summary>
        /// Creates and returns a new OleDbConnection object.
        /// </summary>
        /// <returns>An OleDbConnection object with the connection string set.</returns>
        public static OleDbConnection GetConnection()
        {
            string connString = @"Provider=Microsoft.ACE.OLEDB.12.0;
                                  Data
Source=C:\Users\Cyber_User\source\repos\Sudoku_WPF\DAL\Data\Sudoku_DB1.accdb;
                                  Persist Security Info=True";
            OleDbConnection conn = new OleDbConnection(connString);
            return conn;
        }

        /// <summary>
        /// Creates and returns a new OleDbCommand object.
        /// </summary>
        /// <param name="conn">The OleDbConnection to use for the command.</param>
        /// <param name="sqlstmt">The SQL statement to execute.</param>
        /// <returns>An OleDbCommand object with the specified SQL statement and
connection.</returns>
        public static OleDbCommand GetCommand(OleDbConnection conn, string sqlstmt)
        {
            OleDbCommand cmd = new OleDbCommand(sqlstmt, conn);
            return cmd;
        }

        /// <summary>
        /// Executes the provided SQL statement and returns the results as a DataTable.
        /// </summary>
        /// <param name="sqlstmt">The SQL statement to execute.</param>
        /// <returns>A DataTable containing the results of the SQL query.</returns>
        public static DataTable GetDataTable(string sqlstmt)
        {
            OleDbConnection conn = GetConnection();
            OleDbCommand cmd = GetCommand(conn, sqlstmt);
            OleDbDataAdapter adp = new OleDbDataAdapter();
            adp.SelectCommand = cmd;
            DataTable dt = new DataTable();
            adp.Fill(dt);
            return dt;
        }

        /// <summary>
        /// Executes a non-query SQL statement with the provided parameters.
        /// </summary>
        /// <param name="sqlStmt">The SQL statement to execute.</param>
        /// <param name="parameters">The parameters to add to the SQL command.</param>
        public static void ExecuteCommand(string sqlStmt, params OleDbParameter[]
parameters)
        {
```

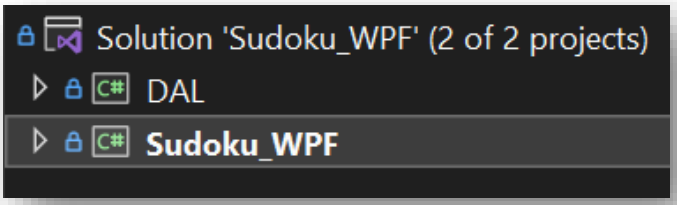
```
            OleDbConnection conn = GetConnection();
            try
            {
                conn.Open();
                OleDbCommand cmd = new OleDbCommand(sqlStmt, conn);
                cmd.Parameters.AddRange(parameters);
                int ret = cmd.ExecuteNonQuery();
            }
            finally
            {
                conn.Close();
            }
        }

        /// <summary>
        /// Retrieves the next available ID for a specified table and column.
        /// </summary>
        /// <param name="tableName">The name of the table to query.</param>
        /// <param name="idColumnName">The name of the ID column to query.</param>
        /// <returns>The next available ID as an integer.</returns>
        public static int GetNextId(string tableName, string idColumnName)
        {
            int nextId = 1; // Default starting point for IDs
            using (OleDbConnection conn = GetConnection())
            {
                string query = $"SELECT MAX([{idColumnName}]) FROM [{tableName}]";
                using (OleDbCommand cmd = new OleDbCommand(query, conn))
                {
                    conn.Open();
                    var result = cmd.ExecuteScalar();
                    if (result != null && result != DBNull.Value)
                    {
                        nextId = Convert.ToInt32(result) + 1;
                    }
                }
            }
            return nextId;
        }
    }
}
```

מפת האפליקציה



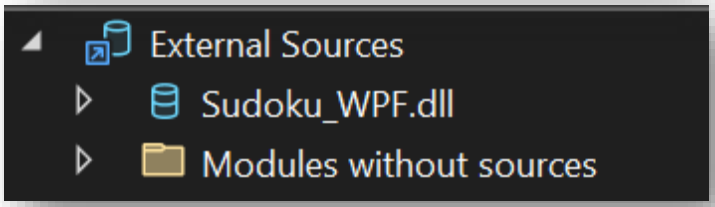
תיאור מסכים

תיאור הקבצים



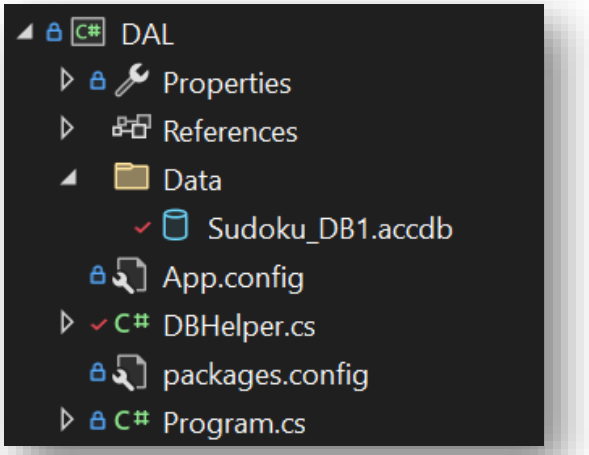
קובץ הsolution מכיל 2 צמתים(nodes):

DAL לטיפול במסד הנתונים והפרויקט סודוקו עצמו.



במצב דיבאג גם מתאפשרת הצפייה בצומת "מקורות חיצוניים":

במקורות החיצוניים נמצאות הספריות החיצוניות כמו dll אשר משומשות בפרויקט ומקורן לא כלול ישירות בפרויקט



הפרויקט DAL (Data Access Layer) מטפל בגישה הנתונים המשומש בפרויקט.

בסיס הנתונים המשומש הוא "Sudoku_DB1" הנמצא בתיקיית "Data" (ראה "מבנה מסד הנתונים").

DBHelper מכיל מחלקה המכילה את השיטות שעוזרות לגשת ולערוך מידע מבסיס הנתונים

(ראה "קוד לבסיס הנתונים").

בנוסף, קיימות גם תיקיות סטנדרטיות שנוצרות אוטומטית כחלק מהפרויקט ב-Visual Studio:

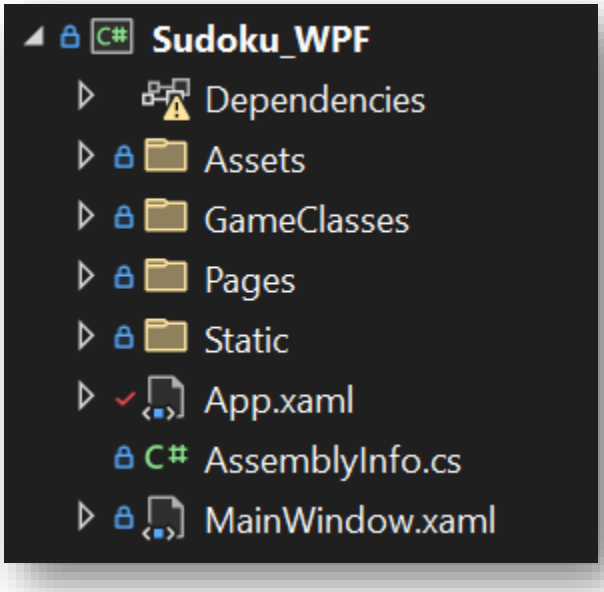
packages.config: קובץ קונפיגורציה של NuGet (ראה רכיבים חיצוניים הסבר).

References: תיקייה עם ההפניות לספריות וקבצים חיצוניים שהפרויקט תלוי בהם.

Program.cs: קובץ הקוד הראשי של הפרויקט, לא מפעיל דבר כיוון שפונקציות ב DBHelper נועדו להיות סטטיות

Properties: מכילה קבצים שמגדירים מאפיינים של הפרויקט, כמו AssemblyInfo.cs, שבו נשמר מידע על ההרכבה (assembly).

App.config: קובץ קונפיגורציה של האפליקציה שמכיל הגדרות כמו חיבורי מסד נתונים, פרמטרים של אפליקציה ועוד.



קובץ הפרויקט מכיל תיקיות רבות המסדרות את קבצי הפרויקט שיצרתי לפי תפקיד (עמוד, מחלקת משחק...) וסוג קובץ (PNG למשל) בצורה אינטואיטיבית, נוחה לשימוש ודיבוג ומסודרת:

Assets: תיקיה המכילה את כל הנכסים של הפרויקט: תמונות, אייקונים, צלילים ,מסיקות רקע , סטיילים ומצבי צבעים.

GameClasses: תיקיה המכילה קבצי cs המחזיקים במחלקות התורמות ליצירת ותפקוד משחק סודוקו בודד.

Pages: תיקיה המכילה את העמודים המשומשים במשחק.

Static: תיקיה המכילה את המחלקות הסטטיות שמטרתן להחזיק מידע מוכר לכולם במהלך ריצת התכנית.

- ראה הסבר מפורט על כל אחת מתיקיות אלו בהמשך.

הקבצים הראשיים בסודוקו:

MainWindow.xaml: קובץ XAML שמגדיר את חלון הראשי של האפליקציה ובו הפריים שמחזיק את העמודים.

תוכן:

```
<Window x:Class="Sudoku_WPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Sudoku_WPF"
        mc:Ignorable="d"
        Title="MainWindow"
        Height="600"
        Width="950"
        AllowsTransparency="True"
        WindowStartupLocation="CenterScreen"
        WindowStyle="None"
        ResizeMode="CanResizeWithGrip"
        Background="Transparent"
        MouseLeftButtonDown="Window_MouseLeftButtonDown"
        MinHeight="330" MinWidth="580"
        Icon="\Assets\Icons\app_Icon.png">



    <Grid>
        <!-- Background Border -->
        <Border Background="{DynamicResource Background}" CornerRadius="20" />

        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>

            <Border Background="{DynamicResource Menu_BG}" CornerRadius="20,0,0,20" />

            <!-- Side Menu -->
            <Grid Grid.Column="0">
                <StackPanel HorizontalAlignment="Left" Margin="0,15,0,0">
                    <!-- Example icon with trigger for hover effect -->
                    <RadioButton Style="{StaticResource MenuButton}"
Tag="/Assets/Icons/home.png" Content="Home" Click="IconButton_Click"/>
                    <RadioButton Style="{StaticResource MenuButton}"
Tag="/Assets/Icons/sudoku.png" Content="Game" Click="IconButton_Click" x:Name
="Game_btn"/>
                    <RadioButton Style="{StaticResource MenuButton}"
Tag="/Assets/Icons/saved.png" Content="Saved" Click="IconButton_Click"/>
                    <RadioButton Style="{StaticResource MenuButton}"
Tag="/Assets/Icons/history.png" Content="History" Click="IconButton_Click"/>
                    <RadioButton Style="{StaticResource MenuButton}"
Tag="/Assets/Icons/instructions.png" Content="Instructions" Click="IconButton_Click"/>
                    <RadioButton Style="{StaticResource MenuButton}"
Tag="/Assets/Icons/settings.png" Content="Settings" Click="IconButton_Click" x:Name
="Settings_btn"/>
                </StackPanel>
            </Grid>

            <!-- Main Content -->
```

```
            <Grid Grid.Column="1">
                <Frame x:Name="MainFrame" NavigationUIVisibility="Hidden" />

            </Grid>
        </Grid>

        <!-- Minimize Button -->
        <Button x:Name="MinimizeApp"
                Style="{StaticResource MinimizeApp}"
                Height="30"
                Width="30"
                HorizontalAlignment="Right"
                VerticalAlignment="Top"
                Margin="0,10,95,0"
                Click="MinimizeApp_Click" />

        <!-- Close Button -->
        <Button x:Name="CloseApp"
    Style="{StaticResource ExitApp}"
    Height="40"
    Width="40"
    HorizontalAlignment="Right"
    VerticalAlignment="Top"
    Margin="0,5,5,0"
    Click="CloseApp_Click" />

        <!-- Resize Toggle Button -->
        <ToggleButton x:Name="Resize"
    Height="40"
    Width="40"
    HorizontalAlignment="Right"
    VerticalAlignment="Top"
    Margin="0,5,50,0"
    Style="{StaticResource ReSize}"
    Checked="TglSizeBtn_Checked"
    Unchecked="TglSizeBtn_Unchecked"/>
    </Grid>

</Window>
```

:MainWindow.xaml.cs

קובץ זה הוא האחראי המרכזי לניווט בין עמודים באפליקציה ואחראי לגישה למסד הנתונים בתחיל הריצה ובסיומה כיוון שהוא הראשון שרץ.

תוכן:

```
 using System;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using DAL;
using Sudoku_WPF.GameClasses;
using Sudoku_WPF.Pages;
using Sudoku_WPF.publico;
using Sudoku_WPF.Themes;
using static Sudoku_WPF.publico.Constants;

namespace Sudoku_WPF
{
    /// <summary>
    /// Interaction logic for the main window of the Sudoku application.
    /// </summary>
    public partial class MainWindow : Window
    {
        public GamePage gamePage; // Game page instance for the Sudoku game
        public SaverPage historyPage; // Page to display history of solved games
        public SaverPage savedPage; // Page to display saved but unsolved games

        /// <summary>
        /// Constructor for initializing the main window of the Sudoku application.
        /// </summary>
        public MainWindow()
        {
```

```
        InitializeComponent();
        MainFrame.Navigate(new OpenningPage()); // Navigate to the opening page
        Resize.Visibility = Visibility.Visible; // Set resize button visibility
        SetSavedGamesFromDB(); // Load saved games from database
        SetSettingsFromDB(); // Load settings from database
        Closing += MainWindow_Closing;

        // Start background music if enabled in settings
        if (Settings.musicOn)
        {
            SoundPlayer.StartMusic(SoundConstants.BACK_MUSIC_NAME);
        }
    }

    /// <summary>
    /// Sets the saved games pages from the database.
    /// </summary>
    private void SetSavedGamesFromDB()
    {
        historyPage = new SaverPage(true); // Initialize history page
        savedPage = new SaverPage(false); // Initialize saved games page

        DataTable dt = DBHelper.GetDataTable("SELECT * FROM tbl_games"); // Retrieve games data from
database

        foreach (DataRow dr in dt.Rows)
        {
            // Create GameInfo object from database row
            GameInfo gameInfo = new GameInfo(
                        Convert.ToInt32(dr["Id"]),
                        dr["GameName"].ToString(),
                        dr["BoardCode"].ToString(),
                        dr["PuzzleCode"].ToString(),
                        dr["Time"].ToString(),
                        Convert.ToInt32(dr["HintsTaken"]),
                        Convert.ToInt32(dr["ChecksTaken"]),
```

```
                    dr["GameDate"].ToString(),
                    (DificultyLevel)Enum.Parse(typeof(DificultyLevel), dr["DifficultyLevel"].ToString()),
                    Convert.ToBoolean(dr["Solved"]),
                    Convert.ToBoolean(dr["Current"]),
                    Convert.ToInt32(dr["BoxHeight"]),
                    Convert.ToInt32(dr["BoxWidth"])
                    );

        // Determine where to add the game info based on solved and current flags
        if (gameInfo.Solved)
        {
            historyPage.AddItemToList(gameInfo); // Add to history page
        }
        else
        {
            if (gameInfo.Current)
            {
                SaverPage.DeleteGameFromDB(gameInfo); // Remove from database if current game
                gamePage = new GamePage(gameInfo); // Initialize game page
                return; // Exit method after initializing game page
            }
            else
            {
                savedPage.AddItemToList(gameInfo); // Add to saved games page
            }
        }
    }
}

/// <summary>
/// Sets the application settings from the database.
/// </summary>
private void SetSettingsFromDB()
{
    DataTable dt = DBHelper.GetDataTable("SELECT * FROM tbl_settings");
```

```
    DataRow dr = dt.Rows[0]; // Get the first row (assuming single settings row)

    // Load settings values from database
    Settings.markSameText = Convert.ToBoolean(dr[DBConstants.Settings_Parameters.SameText]);
    Settings.markRelated = Convert.ToBoolean(dr[DBConstants.Settings_Parameters.MarkRelated]);
    Settings.soundOn = Convert.ToBoolean(dr[DBConstants.Settings_Parameters.SoundOn]);
    Settings.musicOn = Convert.ToBoolean(dr[DBConstants.Settings_Parameters.MusicOn]);

    // Set theme if valid theme found in database
    if (Enum.TryParse(dr[DBConstants.Settings_Parameters.Theme].ToString(),
        out ColorThemes theme))
    {
        Settings.theme = theme;
        ThemeControl.SetColors(Settings.theme); // Apply theme colors
    }
}

private void MainWindow_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    // Stop music if playing
    if (Settings.musicOn)
    {
        SoundPlayer.StopMusic();
    }

    // Insert current game into database if game page exists
    if (gamePage != null)
    {
        SaverPage.InsertGame(gamePage.GetGameInfo(false, true));
    }

    UpdateSettingsInDB(); // Update settings in database
    Application.Current.Shutdown(); // Shutdown the application
}
```

```
/// <summary>
/// Minimizes the application window.
/// </summary>
private void MinimizeApp_Click(object sender, RoutedEventArgs e)
{
    WindowState = WindowState.Minimized; // Minimize the application window
}

/// <summary>
/// Closes the application window and performs necessary cleanup.
/// </summary>
private void CloseApp_Click(object sender, RoutedEventArgs e)
{
    this.Close();
}

/// <summary>
/// Handles toggling the window size between maximized and normal.
/// </summary>
private void TglSizeBtn_Checked(object sender, RoutedEventArgs e)
{
    WindowState = WindowState.Maximized; // Maximize the application window
}

/// <summary>
/// Handles toggling the window size between normal and maximized.
/// </summary>
private void TglSizeBtn_Unchecked(object sender, RoutedEventArgs e)
{
    WindowState = WindowState.Normal; // Restore application window to normal size
}

/// <summary>
/// Handles click events for icon buttons in the application.
```

```
/// </summary>
private void IconButton_Click(object sender, RoutedEventArgs e)
{
    SoundPlayer.PlaySound(SoundConstants.MENU_CLICK); // Play menu click sound

    RadioButton menuBtn = sender as RadioButton; // Get the clicked radio button
    string content = menuBtn.Content.ToString(); // Get the content of the clicked button

    // Navigate to respective pages based on button content
    switch (content)
    {
        case "Home":
            MainFrame.Navigate(new OpenningPage()); // Navigate to opening page
            break;
        case "History":
            if (historyPage == null)
            {
                historyPage = new SaverPage(true); // Initialize history page if not already
            }
            MainFrame.Navigate(historyPage); // Navigate to history page
            break;
        case "Saved":
            if (savedPage == null)
            {
                savedPage = new SaverPage(false); // Initialize saved games page if not already
            }
            MainFrame.Navigate(savedPage); // Navigate to saved games page
            break;
        case "Settings":
            MainFrame.Navigate(new SettingsPage()); // Navigate to settings page
            break;
        case "Game":
            if (gamePage == null)
            {
                Page gsp = new GameSettingsPage();
```

```
                MainFrame.Navigate(gsp); // Navigate to game settings page
            }
            else
            {
                MainFrame.Navigate(gamePage); // Navigate to game page if already initialized
            }
            break;
        case "Instructions":
            MainFrame.Navigate(new InstructionsPage()); // Navigate to instructions page
            break;
        default:
            break;
    }
}

/// <summary>
/// Handles mouse left button down event to enable window dragging.
/// </summary>
private void Window_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
    DragMove(); // Allow dragging of the window
}

/// <summary>
/// Updates the application settings in the database.
/// </summary>
public void UpdateSettingsInDB()
{
    string sqlStmt = DBConstants.UpdateSettingsQuary; // SQL statement to update settings

    // Define parameters for the SQL statement
    OleDbParameter[] parameters =
    {
        new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.SameText,
Settings.markSameText),
```

```
            new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.MarkRelated,
Settings.markRelated),

            new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.SoundOn,
Settings.soundOn),

            new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.MusicOn,
Settings.musicOn),

            new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.Theme,
Settings.theme.ToString()),

            new OleDbParameter(DBConstants.AT + DBConstants.Settings_Parameters.AllowNotes,
Settings.allowNotes)

        };


        DBHelper.ExecuteCommand(sqlStmt, parameters); // Execute the database command with parameters

    }

  }

}
```

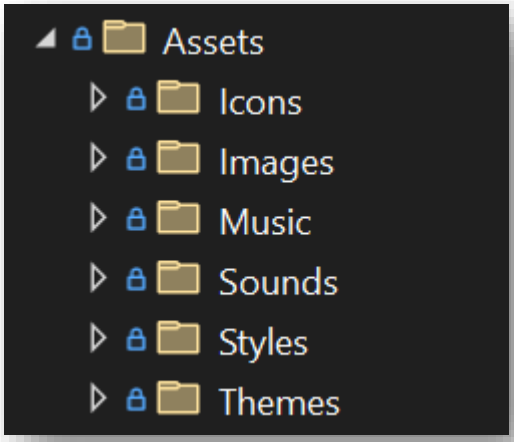
App.xaml :

המחלקה מנגישה את ה resource dictionarys לשאר הפרויקט באמצעות הדרכים היחסיות שלהם.

קוד:

```xml
<Application x:Class="Sudoku_WPF.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">

    <Application.Resources>
        <ResourceDictionary>
            <!-- Merged Resource Dictionaries -->
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Assets/Themes/Dark.xaml"/>

                <ResourceDictionary Source="Assets/Styles/Botton.xaml"/>

                <ResourceDictionary Source="Assets/Styles/TextBlock.xaml"/>

                <ResourceDictionary Source="Assets/Styles/ToggleButton.xaml"/>

                <ResourceDictionary Source="Assets/Styles/RadioButton.xaml"/>

            </ResourceDictionary.MergedDictionaries>


        </ResourceDictionary>
    </Application.Resources>
</Application>
```

נכסי פרויקט



התיקייה מכילה את כל הנכסים של הפרויקט מסודרים בתיקיות:

תמונות - PNG

אייקונים - PNG

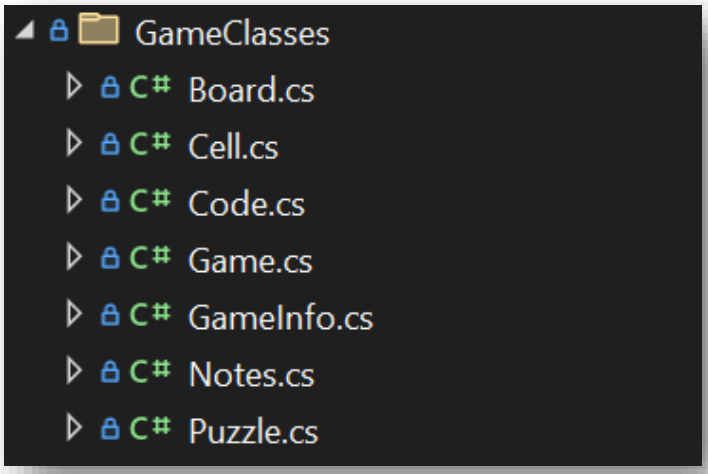
צלילים – WAV

מוסיקות רקע - WAV

סטיילים – XAML (resource dictionary)

נושאי צבעים – XAML (resource dictionary)

קבצי משחק



התיקיה מכילה קבצי cs המחזיקים במחלקות התורמות ליצירת ותפקוד משחק סודוקו בודד.

מחלקת לוח - API

מחלקת לוח מחזיקה באובייקט Puzzle.

תפקידה העיקרי של המחלקה הוא לנהל את לוח המשחק של הסודוקו, הן מבחינת לוגיקת המשחק והן מבחינת הממשק הגרפי. תפקידי המחלקה המרכזיים:

יצירת לוח המשחק: המחלקה יוצרת את ה Grid-המכיל את תאי הסודוקו, מוסיפה שורות, עמודות ומסגרות(Borders) ומציבה את התאים במקומות הנכונים.

אתחול הלוח: המחלקה מאתחלת את הלוח עם הנתונים של הפאזל, מציבה את הערכים ההתחלתיים בתאים ומגדירה אילו תאים ניתנים לעריכה ואילו תאים קבועים.

ניהול לוגיקת המשחק: המחלקה מטפלת באירועים של שינוי ערכים בתאים, בדיקת פתרון הלוח, הצגת האנימציה של פתרון הפאזל והפעלת אירועים כאשר המשחק מסתיים.

הצגת מצב הלוח: המחלקה בודקת את מצב הלוח ומציגה שינויים ויזואליים לפי הערכים שהוזנו בתאים, למשל, שינוי צבעי הרקע והטקסט בהתאם לתקינות הנתונים.

הצגת הפתרון: המחלקה מאפשרת הצגת הפתרון המלא של הפאזל.

אינטראקציה עם המשתמש: המחלקה מספקת פונקציות לניהול הפוקוס על תאים, קבלת התא שנמצא בפוקוס והתמודדות עם שינוי גודל המסגרות.

המחלקה כוללת גם משתנים סטטיים ואירועים המאפשרים לנהל את מצב הלוח ולהגיב לפעולות המשתמש.

קוד מחלקת Board :

```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
```

```
using Sudoku_WPF.publico;
using static Sudoku_WPF.publico.Constants;

namespace Sudoku_WPF.GameClasses
{
    /// <summary>
    /// Represents the Sudoku board consisting of cells and handles game logic.
    /// </summary>
    public class Board
    {
        private static Cell[,] cells; // 2D array of cells representing the Sudoku board
        private Puzzle puzzle; // Puzzle instance associated with the board
        public Grid sudokuGrid; // Grid that contains the Sudoku board UI
        private static bool solutionShown; // Flag indicating if the solution is shown

        public static event EventHandler GameEnded; // Event raised when the game ends

        /// <summary>
        /// Constructor for initializing the board with a given Sudoku grid and puzzle.
        /// </summary>
        /// <param name="sudokuGrid">Grid representing the Sudoku board UI.</param>
        /// <param name="puzzle">Puzzle instance containing the Sudoku puzzle data.</param>
        public Board(Grid sudokuGrid, Puzzle puzzle)
        {
            this.sudokuGrid = sudokuGrid;
            cells = new Cell[GameSettings.BoardSide, GameSettings.BoardSide];
            this.puzzle = puzzle;

            CreateSudokuGrid(sudokuGrid);
            Initialize();
            sudokuGrid.SizeChanged += OnGridSizeChanged;

            solutionShown = false;
        }
```

```
/// <summary>
/// Constructor for initializing the board with a given Sudoku grid, puzzle, and board code.
/// </summary>
/// <param name="sudokuGrid">Grid representing the Sudoku board UI.</param>
/// <param name="puzzle">Puzzle instance containing the Sudoku puzzle data.</param>
/// <param name="boardCode">Encrypted board code string to initialize board state.</param>
public Board(Grid sudokuGrid, Puzzle puzzle, string boardCode)
{
    this.sudokuGrid = sudokuGrid;
    cells = new Cell[GameSettings.BoardSide, GameSettings.BoardSide];
    this.puzzle = puzzle;

    CreateSudokuGrid(sudokuGrid);
    Initialize(boardCode);
    sudokuGrid.SizeChanged += OnGridSizeChanged;

    solutionShown = false;
}

/// <summary>
/// Handles the visual state and validation of the board when a cell's text changes.
/// </summary>
/// <param name="focusCell">The cell currently in focus.</param>
/// <param name="validateForeground">Flag indicating if foreground validation is required.</param>
/// <param name="markText">Optional text to mark in the cells.</param>
public static void VisualizeState(Cell focusCell, bool validateForeground, string markText = "")
{
    if (solutionShown)
        return;
    if (markText == "")
        markText = focusCell.Text;
    if (validateForeground)
        focusCell.SetResourceReference(Control.ForegroundProperty, "Text");

    foreach (Cell cell in cells)
```

```
            {
                if (cell != focusCell)
                {
                    cell.ColorBy(focusCell, markText, validateForeground);
                }
            }

            focusCell.SetResourceReference(Control.BackgroundProperty, "Tbx_Focus");
        }

        /// <summary>
        /// Checks if the Sudoku board is fully solved.
        /// </summary>
        /// <returns>True if the board is fully solved, otherwise false.</returns>
        private static bool IsSolved()
        {
            foreach (Cell cell in cells)
            {
                if (cell.Text == "" || cell.Foreground == (Brush)Application.Current.FindResource("Tbx_WrongForeground")
|| cell.Background == (Brush)Application.Current.FindResource("Tbx_WrongBackground"))
                {
                    return false;
                }
            }
            return true;
        }

        /// <summary>
        /// Shows the solved animation and triggers the game ended event.
        /// </summary>
        private static void ShowSolvedAnimation()
        {
            SoundPlayer.PlaySound(SoundConstants.SOLVED);

            foreach (Cell cell in cells)
            {
```

```
        cell.SetResourceReference(Control.BackgroundProperty, "Tbx_RightBackground");
        cell.IsReadOnly = true;
    }

    GameEnded?.Invoke(null, EventArgs.Empty);
}

/// <summary>
/// Initializes the Sudoku board by setting up cells based on the puzzle.
/// </summary>
public void Initialize()
{
    for (int i = 0; i < GameSettings.BoardSide; i++)
    {
        for (int j = 0; j < GameSettings.BoardSide; j++)
        {
            Cell cell = cells[i, j];
            if (puzzle.IsCellInitial(i, j))
            {
                cell.Text = puzzle.CellValue(i, j).ToString();
                cell.IsReadOnly = true;
                cell.FontWeight = FontWeights.Bold;
            }
            else
            {
                cell.IsReadOnly = false;
                cell.Text = "";
            }

            // Set visual properties using SetResourceReference
            cell.SetResourceReference(Control.BackgroundProperty, ColorConstants.Tbx_Board);
            cell.AttachEventHandlers();
        }
    }
}
```

```
/// <summary>
/// Initializes the Sudoku board with a given encrypted board code.
/// </summary>
/// <param name="boardCode">Encrypted board code string to initialize board state.</param>
public void Initialize(string boardCode)
{
    Initialize();
    boardCode = Code.Unprotect(boardCode);
    string[] cellsStrs = boardCode.Split('|');
    for (int i = 0; i < GameSettings.BoardSide; i++)
    {
        for (int j = 0; j < GameSettings.BoardSide; j++)
        {
            Cell cell = cells[i, j];
            if (cell.IsReadOnly)
            {
                continue;
            }
            string cellStr = cellsStrs[i * GameSettings.BoardSide + j];
            string[] cellParts = cellStr.Split(';');

            cell.Text = cellParts[0];
            string[] notes = cellParts[1].Split(',');
            if (notes[0] != "")
            {
                cell.ShowNotes();

                foreach (string note in notes)
                {
                    if(note.Length == 1)
                    {
                        cell.notesGrid.AddNote(note[0]);
                    }
```

```
            }
        }
    }
  }
}

/// <summary>
/// Creates the Sudoku grid UI by adding rows, columns, cells, and borders.
/// </summary>
/// <param name="mainGrid">Main grid to contain the Sudoku board UI.</param>
private void CreateSudokuGrid(Grid mainGrid)
{
  CreateSeparation(sudokuGrid);
  CreateCells(sudokuGrid);
  AddBorders(sudokuGrid);
}

/// <summary>
/// Creates separation (rows and columns) in the Sudoku grid.
/// </summary>
/// <param name="sudokuGrid">Grid representing the Sudoku board UI.</param>
private void CreateSeparation(Grid sudokuGrid)
{
  for (int i = 0; i < GameSettings.BoardSide; i++)
  {
    sudokuGrid.RowDefinitions.Add(new RowDefinition());
    sudokuGrid.ColumnDefinitions.Add(new ColumnDefinition());
  }
}

/// <summary>
/// Creates cells (text boxes) for the Sudoku board.
/// </summary>
/// <param name="sudokuGrid">Grid representing the Sudoku board UI.</param>
private void CreateCells(Grid sudokuGrid)
```

```
{
    for (int i = 0; i < GameSettings.BoardSide; i++)
    {
        for (int j = 0; j < GameSettings.BoardSide; j++)
        {
            Cell cell = new Cell(i, j);
            sudokuGrid.Children.Add(cell);
            sudokuGrid.Children.Add(cell.notesGrid);
            cells[i, j] = cell;
            cell.InitializeProperties(); // Initialize cell properties with resource references
        }
    }
}

/// <summary>
/// Adds borders (internal and main) to the Sudoku grid.
/// </summary>
/// <param name="sudokuGrid">Grid representing the Sudoku board UI.</param>
private void AddBorders(Grid sudokuGrid)
{
    for (int rows = 0; rows < GameSettings.BoxHeight; rows++)
    {
        for (int cols = 0; cols < GameSettings.BoxWidth; cols++)
        {
            Border internalBorder = new Border();
            internalBorder.SetResourceReference(Border.BorderBrushProperty, "Border");
            Grid.SetRow(internalBorder, cols * GameSettings.BoxHeight);
            Grid.SetRowSpan(internalBorder, GameSettings.BoxHeight);
            Grid.SetColumn(internalBorder, rows * GameSettings.BoxWidth);
            Grid.SetColumnSpan(internalBorder, GameSettings.BoxWidth);
            sudokuGrid.Children.Add(internalBorder);
        }
    }

    // Adding the main border
```

```
        Border gridBorder = new Border();
        gridBorder.SetResourceReference(Border.BorderBrushProperty, "Border");
        Grid.SetRowSpan(gridBorder, GameSettings.BoardSide);
        Grid.SetColumnSpan(gridBorder, GameSettings.BoardSide);
        sudokuGrid.Children.Add(gridBorder);

        sudokuGrid.UpdateLayout(); // Ensure layout is updated immediately
    }

    /// <summary>
    /// Adjusts the border size dynamically based on the Sudoku grid size.
    /// </summary>
    /// <param name="sender">Event sender (Sudoku grid).</param>
    /// <param name="e">Size changed event arguments.</param>
    private void OnGridSizeChanged(object sender, SizeChangedEventArgs e)
    {
        AdjustBorderSize();
    }

    /// <summary>
    /// Adjusts the border thickness of internal and cell borders based on the Sudoku grid size.
    /// </summary>
    private void AdjustBorderSize()
    {
        double newBorderThickness = sudokuGrid.ActualWidth / (GameSettings.BoardSide * 50); // Example
calculation
        foreach (UIElement element in sudokuGrid.Children)
        {
            if (element is Border border)
            {
                if (Grid.GetRowSpan(border) == GameSettings.BoardSide && Grid.GetColumnSpan(border) ==
GameSettings.BoardSide)
                {
                    border.BorderThickness = new Thickness(newBorderThickness *
BoardConstants.EXTERNAL_BORDER_TO_REGULAR);
                }
                else
```

```
                {
                    border.BorderThickness = new Thickness(newBorderThickness *
BoardConstants.INTERNAL_BORDER_TO_REGULAR);
                }
            }
            else if (element is Cell cell)
            {
                cell.BorderThickness = new Thickness(newBorderThickness);
            }
        }
    }

    /// <summary>
    /// Moves focus to the text box of the cell at the specified row and column.
    /// </summary>
    /// <param name="row">Row index of the cell.</param>
    /// <param name="column">Column index of the cell.</param>
    public static void MoveFocusToTextBox(int row, int column)
    {
        Cell nextCell = cells[row, column];
        nextCell.Focus();
    }

    /// <summary>
    /// Shows the solution by filling in all cells and disabling input.
    /// </summary>
    public void ShowSolution()
    {
        solutionShown = true;
        foreach (Cell cell in cells)
        {
            cell.notesGrid.Clear();
            cell.IsEnabled = false;
            cell.Solve();
        }
    }
```

```
/// <summary>
/// Retrieves the currently focused cell on the Sudoku board.
/// </summary>
/// <returns>The currently focused cell, or null if none focused.</returns>
public static Cell FocusedCell()
{
    foreach (Cell cell in cells)
    {
        if (cell.Background == (Brush)Application.Current.FindResource("Tbx_Focus"))
        {
            return cell;
        }
    }
    return null;
}

/// <summary>
/// Disables input on the Sudoku board.
/// </summary>
public void Disable()
{
    sudokuGrid.IsEnabled = false;
}

/// <summary>
/// Checks the current board against the puzzle to validate correct cell entries.
/// </summary>
public void CheckMyBoard()
{
    foreach (Cell cell in cells)
    {
        if (cell.Text == this.puzzle.CellValue(cell.row, cell.column).ToString())
        {
            cell.IsReadOnly = true;
```

```
            cell.SetResourceReference(Control.BackgroundProperty, ColorConstants.Tbx_RightBackground);
            cell.SetResourceReference(Control.ForegroundProperty, ColorConstants.TextFore);
        }
        else if (cell.Text != "")
        {
            cell.SetResourceReference(Control.BackgroundProperty, ColorConstants.Tbx_WrongBackground);
        }
        else
        {
            cell.SetResourceReference(Control.BackgroundProperty, ColorConstants.Tbx_Board);
        }
    }
}

/// <summary>
/// Retrieves a list of cells related to the specified cell based on Sudoku rules.
/// </summary>
/// <param name="cellRow">Row index of the cell.</param>
/// <param name="cellCol">Column index of the cell.</param>
/// <returns>List of cells related to the specified cell.</returns>
public static List<Cell> GetRelatedCells(int cellRow, int cellCol)
{
    List<Cell> relatedCells = new List<Cell>();

    for (int col = 0; col < GameSettings.BoardSide; col++)
    {
        if (col != cellCol)
        {
            relatedCells.Add(cells[cellRow, col]);
        }
    }

    for (int row = 0; row < GameSettings.BoardSide; row++)
    {
        if (row != cellRow)
```

```
        {
            relatedCells.Add(cells[row, cellCol]);
        }
    }

    int boxRowStart = cellRow / GameSettings.BoxHeight * GameSettings.BoxHeight;
    int boxColStart = cellCol / GameSettings.BoxWidth * GameSettings.BoxWidth;

    for (int row = boxRowStart; row < boxRowStart + GameSettings.BoxHeight; row++)
    {
        for (int col = boxColStart; col < boxColStart + GameSettings.BoxWidth; col++)
        {
            if (row != cellRow || col != cellCol)
            {
                relatedCells.Add(cells[row, col]);
            }
        }
    }

    return relatedCells;
}

/// <summary>
/// Generates an encrypted board code representing the current board state.
/// </summary>
/// <returns>Encrypted board code string.</returns>
public string GenerateBoardCode()
{
    string boardCode = "";
    for (int i = 0; i < GameSettings.BoardSide; i++)
    {
        for (int j = 0; j < GameSettings.BoardSide; j++)
        {
            Cell cell = cells[i, j];
            boardCode += $"{cell.Text};";
```

```
                for (int k = 0; k < cell.notesGrid.notes.Count; k++)
                {
                    boardCode += $"{cell.notesGrid.notes[k].Text},";
                }

                boardCode += "|";
            }
        }
        return Code.Protect(boardCode.Substring(0, boardCode.Length - 1));
    }

    /// <summary>
    /// Checks if the game is solved and triggers the solved animation if true.
    /// </summary>
    public static void ForSolvedAnimation()
    {
        if (solutionShown)
            return;
        if (IsSolved())
        {
            ShowSolvedAnimation();
            GameEnded?.Invoke(null, EventArgs.Empty);
        }
    }
  }
}
```

מחלקת הצפנה (code) - API

המחלקה Code מספקת פעולות להצפנה ופענוח של מחרוזות באמצעות ממשק ה- Windows Data Protection API.

שיטת הצפנה (Protect):

שיטה זו מקבלת מחרוזת כקלט ומצפינה אותה באמצעות API(Aplication Interface) של Windows Data Protection.

הקלט ממומר למערך של bytes באמצעות קידוד ASCII.

Entropy (נתוני כניסה לצורך הצפנה) מוגדרים באופן דינמי באמצעות השימוש במידע מזהה של האסמבלי הנוכחית.

תוצאת ההצפנה היא מחרוזת בפורמט Base64 המייצגת את הנתונים המוצפנים.

קוד מחלקת code :

```
using System;
using System.Security.Cryptography;
using System.Text;
using System.Reflection; // for Assembly

namespace Sudoku_WPF.GameClasses
{
    /// <summary>
    /// Provides methods to protect and unprotect data using the Windows data protection
API.
    /// </summary>
    internal class Code
    {
        /// <summary>
        /// Protects a string using Windows data protection API.
        /// </summary>
        /// <param name="str">The string to be protected.</param>
        /// <returns>A base64-encoded string representing the protected data.</returns>
        public static string Protect(string str)
        {
            // Use assembly full name as entropy
            byte[] entropy =
Encoding.ASCII.GetBytes(Assembly.GetExecutingAssembly().FullName);
            byte[] data = Encoding.ASCII.GetBytes(str);
            string protectedData = Convert.ToBase64String(ProtectedData.Protect(data,
entropy, DataProtectionScope.CurrentUser));
            return protectedData;
        }

        /// <summary>
```

```
        /// Unprotects a base64-encoded string using Windows data protection API.
        /// </summary>
        /// <param name="str">The base64-encoded string to be unprotected.</param>
        /// <returns>The original string if successful, otherwise null.</returns>
        public static string Unprotect(string str)
        {
            try
            {
                byte[] protectedData = Convert.FromBase64String(str);
                byte[] entropy =
Encoding.ASCII.GetBytes(Assembly.GetExecutingAssembly().FullName);
                string data =
Encoding.ASCII.GetString(ProtectedData.Unprotect(protectedData, entropy,
DataProtectionScope.CurrentUser));
                return data;
            }
            catch
            {
                return null;
            }
        }
    }
}
```

מחלקת תא(cell) - API

המחלקה מכילה בתוכה אובייקט Notes

המחלקה מייצגת תא בודד בלוח הסודוקו ומשמשת כבסיס לניהול הלוגיקה והאינטראקציה עם תא הסודוקו בUI. היא יורשת מהמחלקה TextBox ומרחיבה אותה בכמה דרכים משמעותיות.

תפקידים עיקריים של המחלקה Cell

אתחול והגדרות תא: קביעת המיקום של התא בלוח הסודוקו, אתחול תכונות חזותיות כמו גודל הפונט, צבעי רקע וגבול, וקביעת הערך הנכון של התא בפתרון הסודוקו.

ניהול אירועים: טיפול באירועים שונים כמו שינוי טקסט, קלט טקסט, לחיצה על מקשים ומעבר פוקוס, על מנת לספק חוויית משתמש חלקה ואינטראקטיבית.

התאמת חזותית: התאמת גודל הפונט של הטקסט בתא בהתאם לגודל התא ולתוכן המוכנס, כדי לשמור על נראות טובה בכל הגדלים.

ניהול והצגת הערות: אפשרות להוסיף ולהציג הערות בתוך התא, המאפשרות לשחקן לרשום הערות קטנות שיסייעו לו בפתרון הפאזל.

בדיקת תקינות: בדיקת הקלט שנכנס לתא ובדיקת תקינות התוכן בתא יחסית לתאים אחרים בלוח, כדי להבטיח שהמשחק יתקדם בצורה נכונה ותקינה.

קוד מחלקת Cell :

```
using Sudoku_WPF.publico;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using static Sudoku_WPF.publico.Constants;


namespace Sudoku_WPF.GameClasses

}

> ///    summary<

 ///    Represents a single cell in the Sudoku board, derived from TextBox.

/> ///    summary<
```

```
    public class Cell : TextBox
}
        public int row, column; // Row and column indices of the cell
        public string solvedValue; // The solved value of the cell
        public Notes notesGrid; // Notes grid associated with the cell

        private List<Cell> relatedCells; // List of cells related to this cell
        private string previousText; // Previous text content of the cell
        private bool notesVisible = false; // Flag indicating if notes are currently visible

> ///        summary<
 ///        Constructor to initialize a cell.
/> ///        summary<
> ///        param name="row">Row index of the cell.</param<
> ///        param name="column">Column index of the cell.</param<
        public Cell(int row, int column)
}
            this.row = row;
            this.column = column;

            notesGrid = new Notes { Visibility = System.Windows.Visibility.Collapsed };

            Grid.SetRow(this, row);
            Grid.SetColumn(this, column);

            Grid.SetRow(notesGrid, row);
            Grid.SetColumn(notesGrid, column);

            solvedValue = Puzzle.CellValueS(row, column).ToString;()

            InitializeProperties;()
{

> ///        summary<
 ///        Clears all content in the cell and notes grid.
```

```
/> ///        summary<

        public void ClearAll()

}

            this.notesGrid.Clear;()

            this.Clear;()

{


> ///        summary<

 ///        Initializes visual properties of the cell.

/> ///        summary<

        public void InitializeProperties()

}

            Text;"" =

            VerticalContentAlignment = System.Windows.VerticalAlignment.Center;

            HorizontalContentAlignment = System.Windows.HorizontalAlignment.Center;

            FontSize = BoardConstants.BOARD_WIDTH * BoardConstants.RELATIVE_FONT_SIZE /
GameSettings.BoardSide;

            SetResourceReference(BorderBrushProperty, "Border"); // Set border brush using resource reference

            SetResourceReference(BackgroundProperty, "Tbx_Board"); // Set background brush using resource
reference

            SetResourceReference(ForegroundProperty, "Text"); // Set foreground brush using resource reference

            CaretBrush = Brushes.Transparent;

{


> ///        summary<

 ///        Attaches event handlers to the cell.

/> ///        summary<

        public void AttachEventHandlers()

}

            TextChanged += TextBox_TextChanged;

//            PreviewTextInput += TextBox_PreviewTextInput;

            PreviewKeyDown += TextBox_PreviewKeyDown;

            GotFocus += TextBox_GotFocus;

            SizeChanged += OnSizeChanged; // Attach SizeChanged event for font size adjustment

            CommandManager.AddPreviewExecutedHandler(this, CommandManager_PreviewExecuted); // Add
PreviewExecuted handler

{
```

```
> ///        summary<
 ///        Detaches event handlers from the cell.
/> ///        summary<
        public void DetachEventHandlers()
}
            TextChanged -= TextBox_TextChanged;
//            PreviewTextInput -= TextBox_PreviewTextInput;
            PreviewKeyDown -= TextBox_PreviewKeyDown;
            GotFocus -= TextBox_GotFocus;
            SizeChanged -= OnSizeChanged; // Detach SizeChanged event


            CommandManager.RemovePreviewExecutedHandler(this, CommandManager_PreviewExecuted); // Remove
PreviewExecuted handler
{


> ///        summary<
 ///        Adjusts font size dynamically based on cell size.
/> ///        summary<
> ///        param name="sender">Event sender.</param<
> ///        param name="e">Size changed event arguments.</param<
        private void OnSizeChanged(object sender, System.Windows.SizeChangedEventArgs e)
}
            AdjustFontSize;()
{


> ///        summary<
 ///        Adjusts the font size of the cell based on its dimensions.
/> ///        summary<
        private void AdjustFontSize()
}
            double fontSize = FontSize;
            double minFontSize = 6; // Minimum font size to avoid being too small
            double maxFontSize = 200; // Maximum font size to avoid being too large


            double width = ActualWidth - Padding.Left - Padding.Right;
```

```
        double height = ActualHeight - Padding.Top - Padding.Bottom;

        if (width <= 0 || height <= 0)
            return;

        System.Windows.Media.FormattedText formattedText = new System.Windows.Media.FormattedText)
            Text,
            System.Globalization.CultureInfo.CurrentCulture,
            System.Windows.FlowDirection.LeftToRight,
            new Typeface(FontFamily, FontStyle, FontWeight, FontStretch),
            fontSize,
            Brushes.Black,
            VisualTreeHelper.GetDpi(this).PixelsPerDip;(

        double scaleFactor = System.Math.Min(width / formattedText.Width, height / formattedText.Height);
        double newFontSize = System.Math.Clamp(fontSize * scaleFactor, minFontSize, maxFontSize);

        FontSize = newFontSize;
{

> ///      summary<
 ///      Colors the cell based on the focus cell and validates the content.
/> ///       summary<
> ///      param name="focusCell">The currently focused cell.</param<
> ///      param name="previewText">Text to preview in the cell.</param<
> ///      param name="validateForeground">Flag indicating if foreground validation is required.</param<
      public void ColorBy(Cell focusCell, string previewText, bool validateForeground)
}
        bool isValid = Foreground == (Brush)FindResource(ColorConstants.TextFore);

        SetResourceReference(BorderBrushProperty, ColorConstants.Border);

        if (IsRelatedTo(focusCell))
}
```

```
            SetResourceReference(BackgroundProperty, Text == focusCell.Text && Text != "" &&
!focusCell.IsReadOnly ? ColorConstants.Tbx_WrongBackground : (Settings.markRelated ? ColorConstants.Tbx_Sign
: ColorConstants.Tbx_Board));


            if (validateForeground)

}

                if (Text == focusCell.previousText && Text != focusCell.Text && Text != "")

}

                    isValid = IsRelativelyValid;()

{

                else if (Text == focusCell.Text && Text != "")

}

                    if (focusCell.Foreground != (Brush)FindResource(ColorConstants.Tbx_WrongForeground))

}

                        SoundPlayer.PlaySound(SoundConstants.WRONG);

{

                    isValid = false;

                    focusCell.SetResourceReference(ForegroundProperty, ColorConstants.Tbx_WrongForeground);

{

                if (!IsReadOnly)

}

                    SetResourceReference(ForegroundProperty, isValid ? ColorConstants.TextFore :
ColorConstants.Tbx_WrongForeground);

{

{

{

          else

}

            SetResourceReference(BackgroundProperty, Text == previewText && Text != "" &&
Settings.markSameText ? ColorConstants.Tbx_SameText : ColorConstants.Tbx_Board);

{

{


> ///        summary<

 ///        Checks if this cell is related to another cell.

/> ///        summary<

> ///        param name="anotherCell">Another cell to compare relation.</param<
```

```
> ///        returns>True if related, otherwise false.</returns<

        private bool IsRelatedTo(Cell anotherCell)

}

            bool sameRow = row == anotherCell.row;

            bool sameColumn = column == anotherCell.column;

            bool sameBox = row / GameSettings.BoxHeight == anotherCell.row / GameSettings.BoxHeight && column /
GameSettings.BoxWidth == anotherCell.column / GameSettings.BoxWidth;

            return sameRow || sameColumn || sameBox;

{


        public void CharEntered(char ch)

}

            if (ch == '\0' || ch == '0')

                return;


            if (IsReadOnly)

}

                Board.VisualizeState(this, false, ch.ToString());

{

            else

}

                if (Text.Length == 1)

}

                    if (Settings.allowNotes)

}

                        ShowNotes(Text[0], ch);

{

                    else

}

                        Text = ch.ToString;()

{

{

                else

}

                    if (notesVisible)

}
```

```
                    SwitchNote(ch);
{
                else
}
                    Text = ch.ToString().ToUpper;()
                    CaretIndex = Text.Length;
{
{
{
{

> ///       summary<
 ///       Solves the cell, displaying its solved value and making it read-only.
/> ///       summary<
> ///       param name="hint">Optional parameter indicating if the solve is a hint.</param<
       public void Solve(bool hint = false)
}
          if (!IsReadOnly)
}
              Text = solvedValue;
              IsReadOnly = true;
{
          Foreground = (Brush)FindResource(ColorConstants.TextFore);
          BorderBrush = (Brush)FindResource(ColorConstants.Border);
          if (!hint)
}
              SetResourceReference(BackgroundProperty, ColorConstants.Tbx_Board);
{
{

> ///       summary<
 ///       Shows notes in the cell for two numbers.
/> ///       summary<
> ///       param name="firstNumber">First number to show as a note.</param<
> ///       param name="secondNumber">Second number to show as a note.</param<
```

```
        private void ShowNotes(char firstNumber, char secondNumber)
}
            ShowNotes;()

            notesGrid.Clear;()
            notesGrid.ManipulateNote(firstNumber);
            notesGrid.ManipulateNote(secondNumber);
{

> ///        summary<
 ///        Shows the notes grid in the cell.
/> ///        summary<
        public void ShowNotes()
}
            if (notesVisible)
                return;

            notesVisible = true;
            notesGrid.Visibility = System.Windows.Visibility.Visible;
            Text;"" =
{

> ///        summary<
 ///        Switches the note in the cell.
/> ///        summary<
> ///        param name="number">Number to manipulate as a note.</param<
        private void SwitchNote(char number)
}
            if (notesGrid != null)
}
                notesGrid.ManipulateNote(number);
                if (notesGrid.IsLastOne())
}
                    Text = notesGrid.LastOne;()
                    notesGrid.Visibility = System.Windows.Visibility.Collapsed;
```

```
                notesVisible = false;
{
{
{

> ///       summary<
 ///       Validates if the input text is valid for the cell.
/> ///       summary<
> ///       param name="input">Input text to validate.</param<
> ///       param name="maxValue">Maximum value allowed.</param<
> ///       returns>True if valid, otherwise false.</returns<
       public static bool IsValidInput(char input)
}
          int value = HexaToInt(input);
          return value >= 1 && value <= GameSettings.BoardSide;
{

> ///       summary<
 ///       Handles the PreviewExecuted event for command interception.
/> ///       summary<
> ///       param name="sender">The event sender.</param<
> ///       param name="e">Event arguments containing information about the executed command.</param<
       private void CommandManager_PreviewExecuted(object sender, ExecutedRoutedEventArgs e)
}
          if (e.Command == ApplicationCommands.Paste)
}
             e.Handled = true; // Cancel paste command
{
{



> ///       summary<
 ///       Handles key down events in the cell.
/> ///       summary<
> ///       param name="sender">Event sender.</param<
```

```
> ///        param name="e">Key event arguments.</param<
        private void TextBox_PreviewKeyDown(object sender, KeyEventArgs e)
}

            switch (e.Key)
}
                case Key.Up:
                    if (row > 0)
}
                        Board.MoveFocusToTextBox(row - 1, column);
{
                    break;
                case Key.Down:
                    if (row < GameSettings.BoardSide - 1)
}
                        Board.MoveFocusToTextBox(row + 1, column);
{
                    break;
                case Key.Left:
                    if (column > 0)
}
                        Board.MoveFocusToTextBox(row, column - 1);
{
                    break;
                case Key.Right:
                    if (column < GameSettings.BoardSide - 1)
}
                        Board.MoveFocusToTextBox(row, column + 1);
{
                    break;
{

            if ((e.Key == Key.Delete || e.Key == Key.Space) && !IsReadOnly)
}
                Text;"" =
```

```
            notesGrid.Clear;()
            e.Handled = true;
{
        else if (e.Key == Key.Back && !IsReadOnly)
}
            Text;"" =
            notesGrid.RemoveNoteByIdx(notesGrid.notes.Count - 1);
            e.Handled = true;
{



        CharEntered(KeyCharValue(e.Key));



        e.Handled = true;
{

> ///      summary<
 ///      Checks if the cell content is relatively valid.
/> ///      summary<
> ///      returns>True if relatively valid, otherwise false.</returns<
      public bool IsRelativelyValid()
}
        if (relatedCells == null)
            relatedCells = Board.GetRelatedCells(row, column);

        foreach (Cell cell in relatedCells)
}
            if (this != cell && cell.Text != "" && Text == cell.Text)
}
                return false;
{
{
        return true;
{
```

```
> ///        summary<
 ///        Checks if the provided key is a valid input for the cell.
/> ///        summary<
> ///        param name="key">The key to validate.</param<
> ///        returns>True if the key represents a valid input, otherwise false.</returns<
        public char KeyCharValue(Key key)
}
            string strKey = key.ToString;()
            char charKey = '\0;'
            if (strKey.Length == 2 && strKey[0] == 'D')
}
                charKey = strKey;[1]
{
            else if(strKey.Length == 1)
}
                charKey = strKey;[0]
{

            return IsValidInput(charKey) ? charKey : '\0;'

{



> ///        summary<
 ///        Checks if the cell content is totally valid.
/> ///        summary<
> ///        returns>True if totally valid, otherwise false.</returns<
        public bool IsTotallyValid()
}
            return Text == Puzzle.CellValueS(row, column).ToString;()
{

> ///        summary<
 ///        Handles text changed event in the cell.
```

```
/> ///        summary<
> ///        param name="sender">Event sender.</param<
> ///        param name="e">Text changed event arguments.</param<
        private void TextBox_TextChanged(object sender, TextChangedEventArgs e)
}
            if (this != null && !IsReadOnly)
}
                AdjustFontSize(); // Adjust font size when text changes
                Board.VisualizeState(this, true, Text);
                Board.ForSolvedAnimation;()

                previousText = Text;
{
{

> ///        summary<
 ///        Handles focus event in the cell.
/> ///        summary<
> ///        param name="sender">Event sender.</param<
> ///        param name="e">Routed event arguments.</param<
        private void TextBox_GotFocus(object sender, RoutedEventArgs e)
}
            CaretIndex = Text.Length;
            Board.VisualizeState(this, false, Text);
//            Background = Foreground;
{



> ///        summary<
 ///        Converts a hexadecimal character to an integer value.
/> ///        summary<
> ///        param name="character">Hexadecimal character to convert.</param<
> ///        returns>Integer value corresponding to the hexadecimal character.</returns<
        private static int HexaToInt(char character)
}
```

```
        if (char.IsDigit(character))
}
            return character - '0;'
{
        else if (char.IsLetter(character))
}
            return char.ToUpper(character) - 'A' + Constants.NUM_DIGITS;
{

        return Constants.ERROR;
{
{
{
```

מחלקת מידע על משחק GameInfo () – API

המחלקה GameInfo אחראית לאחסון מידע על משחק סודוקו מסוים. מידע זה נועד לשמש לניהול ושמירת מצב המשחק ולצורך המשך משחק שמור.

היא מכילה נתונים התואמים את הטורים במסד הנתונים.

קוד מחלקת: GameInfo

```
using static Sudoku_WPF.publico.Constants;
using Sudoku_WPF.GameClasses;
using System.Windows.Controls;
using System.Windows.Threading;
using Sudoku_WPF.publico;


namespace Sudoku_WPF.GameClasses
{
    public class GameInfo
    {
        public string BoardCode { get; set; }
        public int BoxHeight { get; set; }
        public int BoxWidth { get; set; }
        public string PuzzleCode { get; set; }
        public string Name { get; set; }
        public string Time { get; set; }
        public int Hints { get; set; }
        public int Checks { get; set; }
        public string Date { get; set; }
        public bool Solved { get; set; }
        public bool Current { get; set; }
        public DificultyLevel DifficultyLevel { get; set; }

        public int Id { get; set; }

        public GameInfo(int id, string name, string boardCode, string puzzleCode, string
time, int hintsTaken, int checksTaken, string date, DificultyLevel difficultyLevel, bool
solved, bool current, int boxHeight, int boxWidth)
        {
            Name = name;
            BoardCode = boardCode;
            PuzzleCode = puzzleCode;
            Time = time;
            Hints = hintsTaken;
            Checks = checksTaken;
            Date = date;
            Solved = solved;
            Current = current;
            Id = id;
            BoxHeight = boxHeight;
            BoxWidth = boxWidth;
            DifficultyLevel = difficultyLevel;
        }
    }
}
```

מחלקת הערות (Notes)– API

מחלקת הערות Notes (- API)

המחלקה Notes נועדה לניהול של הערות במשחק סודוקו. המחלקה יורשת מ- UniformGrid ומספקת לשחקנים יכולות להוסיף, להסיר ולנקות הערות על לוח המשחק.

תפקידים עיקריים של המחלקה Notes:

אתחול והגדרות ההערות:

יצירת רשימה של TextBlock לייצוג ההערות.

הגדרת ההערות בUI על ידי שימוש ב- UniformGrid.

ניהול הערות:

הוספה והסרה של הערות וניקוי כללי שלהן.

פריסת הערות:

התאמת מספר השורות והעמודות בUniformGrid בהתאם למספר ההערות הקיימות, כך שהפריסה תהיה אסתטית ונוחה לקריאה.

קוד מחלקת Notes :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls.Primitives;
using System.Windows.Controls;
using System.Windows;
using System.Windows.Media;
using Sudoku_WPF.publico;

namespace Sudoku_WPF.GameClasses
{
    /// <summary>
    /// Represents a grid of notes for a Sudoku game.
    /// This class provides methods to manipulate the notes, including adding,
removing, and clearing notes.
    /// </summary>
    public class Notes : UniformGrid
    {
        /// <summary>
        /// List of TextBlock elements representing the notes.
        /// </summary>
        public List<TextBlock> notes;
```

```
        /// <summary>
        /// Initializes a new instance of the Notes class and clears any existing
notes.
        /// </summary>
        public Notes()
        {
            notes = new List<TextBlock>();
            Clear();
        }

        /// <summary>
        /// Adds or removes a note based on its existence.
        /// </summary>
        /// <param name="noteText">The text of the note to manipulate.</param>
        public void ManipulateNote(char noteText)
        {
            if (NoteExist(noteText))
            {
                RemoveNote(noteText);
            }
            else
            {
                AddNote(noteText);
            }
        }

        /// <summary>
        /// Adds a note with the specified text if notes are allowed.
        /// </summary>
        /// <param name="noteText">The text of the note to add.</param>
        public void AddNote(char noteText)
        {
            if (Settings.allowNotes)
            {
                if (noteText != '\0' && noteText != ' ')
                {
                    Viewbox noteContainer = new Viewbox();
                    noteContainer.HorizontalAlignment = HorizontalAlignment.Stretch;
                    noteContainer.VerticalAlignment = VerticalAlignment.Stretch;

                    TextBlock note = NewNote(noteText);

                    // Add the TextBlock to the ViewBox
                    noteContainer.Child = note;

                    notes.Add(note);
                    Children.Add(noteContainer);

                    MyUpdateLayout();
                }
            }
        }

        /// <summary>
        /// Creates a new TextBlock for a note with the specified text.
        /// </summary>
        /// <param name="noteText">The text of the note.</param>
        /// <returns>A TextBlock element with the specified note text.</returns>
        private TextBlock NewNote(char noteText)
        {
```

```
        TextBlock note = new TextBlock()
        {
            Text = noteText.ToString(),
            HorizontalAlignment = HorizontalAlignment.Stretch,
            VerticalAlignment = VerticalAlignment.Stretch,
            TextAlignment = TextAlignment.Center,
            IsHitTestVisible = false
        };

        // Set the Foreground property using resource reference
        note.SetResourceReference(TextBlock.ForegroundProperty, "Text");

        return note;
    }

    /// <summary>
    /// Clears all notes from the grid.
    /// </summary>
    public void Clear()
    {
        notes.Clear();
        Children.Clear();

        MyUpdateLayout();
    }

    /// <summary>
    /// Determines if there is only one note remaining.
    /// </summary>
    /// <returns>True if there is only one note, otherwise false.</returns>
    public bool IsLastOne()
    {
        return notes.Count == 1;
    }

    /// <summary>
    /// Gets the text of the last remaining note.
    /// </summary>
    /// <returns>The text of the last remaining note.</returns>
    public string LastOne()
    {
        return notes[0].Text;
    }

    /// <summary>
    /// Checks if a note with the specified text exists.
    /// </summary>
    /// <param name="noteText">The text of the note to check.</param>
    /// <returns>True if the note exists, otherwise false.</returns>
    public bool NoteExist(char noteText)
    {
        foreach (TextBlock note in notes)
        {
            if (note.Text[0] == noteText)
                return true;
        }
        return false;
    }

    /// <summary>
    /// Removes a note with the specified text.
```

```
        /// </summary>
        /// <param name="noteText">The text of the note to remove.</param>
        private void RemoveNote(char noteText)
        {
            for (int i = 0; i < notes.Count; i++)
            {
                if (notes[i].Text[0] == noteText)
                {
                    RemoveNoteByIdx(i);
                    break;
                }
            }
        }

        /// <summary>
        /// Removes a note at the specified index.
        /// </summary>
        /// <param name="idx">The index of the note to remove.</param>
        public void RemoveNoteByIdx(int idx)
        {
            if (idx < notes.Count && idx >= 0)
            {
                Children.RemoveAt(idx);
                notes.RemoveAt(idx);
                MyUpdateLayout();
            }
        }

        /// <summary>
        /// Updates the layout of the grid based on the number of notes.
        /// </summary>
        private void MyUpdateLayout()
        {
            int noteCount = notes.Count;
            if (noteCount < 4)
            {
                UpdateLayout();
                return;
            }

            Columns = (int)Math.Ceiling(Math.Sqrt(noteCount));
            Rows = (int)Math.Ceiling(noteCount / (double)Columns);

            if (Columns * (Rows - 1) >= noteCount)
            {
                Rows--;
            }
        }
    }
}
```

מחלקת משחק (Game)– API

המחלקה Game אחראית לניהול משחק. היא כוללת את הלוח, הטיימר, והפאזל, ומנהלת את ההתקדמות והסיום של המשחק.

תפקידי המחלקה Game

אתחול משחק: המחלקה מאפשרת התחלת משחק סודוקו חדש עם לוח משחק חדש או ממידע משחק שמור.

ניהול טיימר: המחלקה מטפלת בטיימר של המשחק, מעדכנת את הזמן שחלף ומציגה אותו.

סיום המשחק: המחלקה מטפלת בסיום המשחק, כולל שמירה על המידע המשחק וסיום הטיימר.

אירועים: המחלקה מאזינה ומגיבה לאירוע פתרון המשחק.

קוד למחלקת Game:

```
 using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Threading;
using static Sudoku_WPF.publico.Constants;

namespace Sudoku_WPF.GameClasses
{
    /// <summary>
    /// Represents a Sudoku game session.
    /// </summary>
    public class Game
    {
        private Board board;
        private TextBlock timerTxtB;
        private DispatcherTimer timer;
        private TimeSpan elapsedTime;
        private Puzzle puzzle;
        private bool inProgress;

        /// <summary>
        /// Initializes a new instance of the <see cref="Game"/> class for starting
a new Sudoku game.
        /// </summary>
        /// <param name="sodukoGrid">The grid where the Sudoku game is
displayed.</param>
        /// <param name="timerTxtB">The text block for displaying the game
timer.</param>
        public Game(Grid sodukoGrid, TextBlock timerTxtB)
```

```
        {
            this.timerTxtB = timerTxtB;
            InitTimer();
            this.inProgress = true;
            timerTxtB.Text = TimerConstants.DEFAULT_TIME;

            try
            {
                Mouse.OverrideCursor = Cursors.Wait; // Change cursor to Wait
                puzzle = new Puzzle();
                board = new Board(sodukoGrid, puzzle);
            }
            finally
            {
                Mouse.OverrideCursor = null; // Restore cursor
            }
            Board.GameEnded += OnGameSolved;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Game"/> class for starting
a new Sudoku game with a specific puzzle code.
        /// </summary>
        /// <param name="sodukoGrid">The grid where the Sudoku game is
displayed.</param>
        /// <param name="timerTxtB">The text block for displaying the game
timer.</param>
        /// <param name="puzzleCode">The puzzle code to initialize the Sudoku
game.</param>
        public Game(Grid sodukoGrid, TextBlock timerTxtB, string puzzleCode)
        {
            this.timerTxtB = timerTxtB;
            InitTimer();
            this.inProgress = true;
            timerTxtB.Text = TimerConstants.DEFAULT_TIME;
            try
            {
                Mouse.OverrideCursor = Cursors.Wait; // Change cursor to Wait
                puzzle = new Puzzle(puzzleCode);
                board = new Board(sodukoGrid, puzzle);
            }
            finally
            {
                Mouse.OverrideCursor = null; // Restore cursor
            }
            Board.GameEnded += OnGameSolved;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Game"/> class for starting
a new Sudoku game from saved game information.
        /// </summary>
        /// <param name="sodukoGrid">The grid where the Sudoku game is
displayed.</param>
        /// <param name="timerTxtB">The text block for displaying the game
timer.</param>
        /// <param name="info">The saved game information to initialize the Sudoku
game.</param>
        public Game(Grid sodukoGrid, TextBlock timerTxtB, GameInfo info)
        {
            this.timerTxtB = timerTxtB;
```

```
            InitTimer(info.Time);
            this.inProgress = true;
            try
            {
                Mouse.OverrideCursor = Cursors.Wait; // Change cursor to Wait
                puzzle = new Puzzle(info.PuzzleCode);
                board = new Board(sodukoGrid, puzzle, info.BoardCode);
            }
            finally
            {
                Mouse.OverrideCursor = null; // Restore cursor
            }

            timerTxtB.Text = info.Time;
            Board.GameEnded += OnGameSolved;
        }

        /// <summary>
        /// Handles the event when the Sudoku game is solved.
        /// </summary>
        /// <param name="sender">The event sender.</param>
        /// <param name="e">The event arguments.</param>
        private void OnGameSolved(object sender, EventArgs e)
        {
            End(true, true);
        }

        /// <summary>
        /// Ends the current Sudoku game session.
        /// </summary>
        /// <param name="isSolved">Indicates whether the game is solved.</param>
        /// <param name="toSave">Indicates whether to save the game.</param>
        public void End(bool isSolved, bool toSave)
        {
            this.timer.Stop();
            this.inProgress = false;

            var window = (MainWindow)Application.Current.MainWindow;

            if (window.gamePage != null)// wasnt already ended
            {
                if (toSave)
                {
                    GameInfo gameInfo = window.gamePage.GetGameInfo(isSolved,
false);
                    if (isSolved)
                    {
                        window.historyPage.AddItemToListAndDB(gameInfo);
                    }
                    else
                    {
                        window.savedPage.AddItemToListAndDB(gameInfo);
                    }
                }

                window.gamePage.DisableGameControls();
                window.gamePage = null;
            }

        }
```

```
/// <summary>
/// Checks if a Sudoku game is currently in progress.
/// </summary>
/// <returns>True if a game is in progress; otherwise, false.</returns>
public bool IsInGame()
{
    return inProgress;
}

/// <summary>
/// Retrieves the puzzle code associated with the current Sudoku game.
/// </summary>
/// <returns>The puzzle code string.</returns>
public string GetPuzzleCode()
{
    return puzzle.GetCode();
}

/// <summary>
/// Initializes the game timer.
/// </summary>
private void InitTimer()
{
    // Initialize the timer
    timer = new DispatcherTimer();
    timer.Interval = TimeSpan.FromSeconds(1); // Tick every second
    timer.Tick += Timer_Tick;
}

/// <summary>
/// Initializes the game timer with a specified starting time.
/// </summary>
/// <param name="startTime">The starting time for the game timer.</param>
private void InitTimer(string startTime)
{
    // Initialize the timer
    timer = new DispatcherTimer();
    timer.Interval = TimeSpan.FromSeconds(1); // Tick every second
    timer.Tick += Timer_Tick;

    // Parse the string and convert it to a TimeSpan
    if (TimeSpan.TryParse(startTime, out TimeSpan parsedTime))
    {
        elapsedTime = parsedTime;
    }
}

/// <summary>
/// Event handler for the game timer tick event.
/// </summary>
/// <param name="sender">The event sender.</param>
/// <param name="e">The event arguments.</param>
private void Timer_Tick(object sender, EventArgs e)
{
    // Update the elapsed time and display it
    elapsedTime = elapsedTime.Add(TimeSpan.FromSeconds(1));
    timerTxtB.Text = GetTime();
}

/// <summary>
/// Retrieves the formatted string representation of the elapsed game time.
```

```
        /// </summary>
        /// <returns>The formatted game time string.</returns>
        public string GetTime()
        {
            return elapsedTime.ToString(TimerConstants.FORMAT);
        }

        /// <summary>
        /// Accessor for retrieving the Sudoku game board.
        /// </summary>
        public Board Board => board;

        /// <summary>
        /// Accessor for retrieving the game timer.
        /// </summary>
        public DispatcherTimer Timer => timer;
    }
}
```

מחלקת פאזל API- (puzzle)

המחלקה Puzzle מייצגת פאזל סודוקו.

ייצוג פאזל : המחלקה מכילה משתנים לשמירה על הפאזל פתור (solvedPuzzle) ואילו תאים לא ניתנים לשינוי (initialCells). זה מאפשר את הלוגיקה של חידת הסודוקו (הלוח במצבו ההתחלתי).

ייצירת פאזל:

יש למחלקה אפשרות ליצור פאזל חדש או מקוד פאזל קיים.

פאזל חדש נוצר באמצעות יצירת לוח פתור ב BachTrecking באמצעות רקורסיה (ראה דברים מעניינים בפרויקט).

וקוד פאזל קיים מחזיק מידע על הפאזל הפתור ועל התאים ההתחלתיים בקוד בפרוטוקול שפיתחתי. הקוד מגיע מוצפן (ראה מחלקת Code).

פעולות על תאים: המחלקה מציעה שיטות לבדיקה האם תא מסוים הוא תא התחלתי, ולקבלת ערך של תא בפאזל הפתור.

ניהול קוד פאזל: ישנם שיטות שמייצגות את הפאזל באמצעות קוד, ומאפשרות ייבוא ויצוא של מצב הפאזל בין מצבים שונים.

הקוד למחלקת Puzzle:

```
 using System;
using Sudoku_WPF.publico;
using static Sudoku_WPF.publico.Constants;

namespace Sudoku_WPF.GameClasses
{
    /// <summary>
    /// Represents a Sudoku puzzle and provides methods for puzzle generation, management, and access.
    /// </summary>
    public class Puzzle
    {
        public static bool[,] initialCells; // Array to track which cells are initial (pre-filled)
        public static char[,] solvedPuzzle; // Array to store the solved puzzle
        public string code = "NO CODE YET"; // Code representing the puzzle state
        private static Random rnd = new Random(); // Random number generator for shuffling and selecting initial cells
```

```
/// <summary>
/// Constructor to create a new Sudoku puzzle instance with a random arrangement of initial cells.
/// </summary>
public Puzzle()
{
    initialCells = new bool[GameSettings.BoardSide, GameSettings.BoardSide];
    solvedPuzzle = new char[GameSettings.BoardSide, GameSettings.BoardSide];

    SetFullCells();
    this.code = CreatePuzzleCode();
}

/// <summary>
/// Constructor to import a Sudoku puzzle from a given code representation.
/// </summary>
/// <param name="code">The code representing the puzzle state.</param>
public Puzzle(string code)
{
    ImportPuzzleCode(code);
    this.code = CreatePuzzleCode();
}

/// <summary>
/// Checks if a specified cell is initially filled (pre-filled) in the puzzle.
/// </summary>
/// <param name="row">The row index of the cell.</param>
/// <param name="col">The column index of the cell.</param>
/// <returns>True if the cell is an initial cell; otherwise, false.</returns>
public bool IsCellInitial(int row, int col)
{
    return initialCells[row, col];
}

/// <summary>
```

```
/// Retrieves the value of a specified cell in the solved puzzle.
/// </summary>
/// <param name="row">The row index of the cell.</param>
/// <param name="col">The column index of the cell.</param>
/// <returns>The character value at the specified cell position in the solved puzzle.</returns>
public static char CellValueS(int row, int col)
{
    return solvedPuzzle[row, col];
}

/// <summary>
/// Retrieves the value of a specified cell in the solved puzzle.
/// </summary>
/// <param name="row">The row index of the cell.</param>
/// <param name="col">The column index of the cell.</param>
/// <returns>The character value at the specified cell position in the solved puzzle.</returns>
public char CellValue(int row, int col)
{
    return solvedPuzzle[row, col];
}

/// <summary>
/// Retrieves the code representing the current state of the puzzle.
/// </summary>
/// <returns>The code representing the puzzle state as a string.</returns>
public string GetCode() => code;

/// <summary>
/// Generates a new Sudoku puzzle based on the current game settings and difficulty level.
/// </summary>
private void SetFullCells()
{
    double fullCellsPercent;

    switch (GameSettings.dificultyLevel)
```

```
    {
        case DificultyLevel.Easy:
            fullCellsPercent = GameSettingsConstants.FULL_CELLS_EASY;
            break;
        case DificultyLevel.Medium:
            fullCellsPercent = GameSettingsConstants.FULL_CELLS_MEDIUM;
            break;
        case DificultyLevel.Hard:
            fullCellsPercent = GameSettingsConstants.FULL_CELLS_HARD;
            break;
        default:
            fullCellsPercent = 0;
            break;
    }
    GenerateSolvedPuzzle();
    SelectInitialCells((int)(GameSettings.BoardSide * GameSettings.BoardSide * fullCellsPercent));
}

/// <summary>
/// Generates a solved Sudoku puzzle based on the current state of the board.
/// </summary>
private void GenerateSolvedPuzzle()
{
    FillCells(0, 0);
}


/// <summary>
/// Recursively fills the remaining cells in the board to complete the Sudoku puzzle.
/// </summary>
/// <param name="row">The current row index for filling cells.</param>
/// <param name="col">The current column index for filling cells.</param>
/// <returns>True if the board is successfully filled, otherwise false.</returns>
private bool FillCells(int row, int col)
{
```

```
    if (row == GameSettings.BoardSide - 1 && col == GameSettings.BoardSide)
        return true;
    if (col == GameSettings.BoardSide)
    {
        row++;
        col = 0;
    }
    if (solvedPuzzle[row, col] != '\0')
        return FillCells(row, col + 1);

    char[] nums = GenerateShuffledNumbers();

    foreach (char num in nums)
    {
        if (IsSafeToPlace(row, col, num))
        {
            solvedPuzzle[row, col] = num;
            if (FillCells(row, col + 1))
                return true;
            solvedPuzzle[row, col] = '\0';
        }
    }
    return false;
}

/// <summary>
/// Generates an array of characters (numbers or letters) shuffled randomly.
/// </summary>
/// <returns>An array of characters representing shuffled numbers or letters.</returns>
private char[] GenerateShuffledNumbers()
{
    char[] nums = new char[GameSettings.BoardSide];
    for (int i = 0; i < GameSettings.BoardSide; i++)
    {
        nums[i] = ConvertToHexa(i + 1);
```

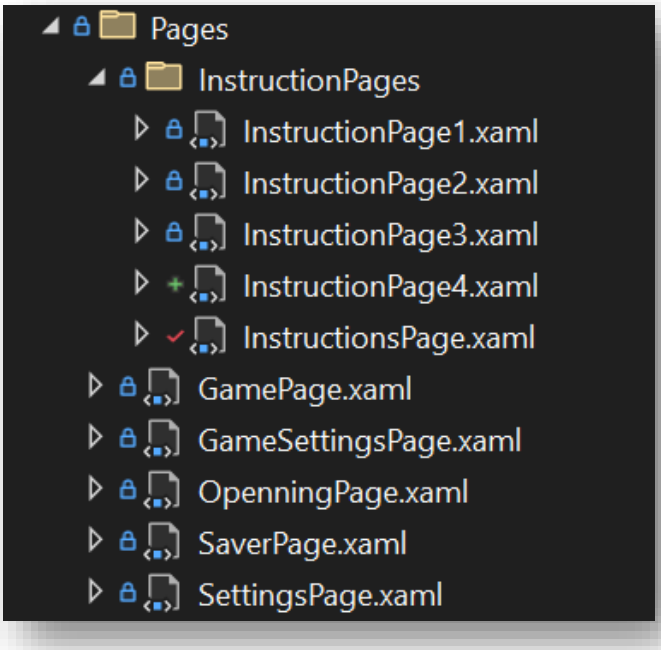
```
    }
    ShuffleArray(nums);
    return nums;
}

/// <summary>
/// Shuffles an array of characters in place using a simple random swap technique.
/// </summary>
/// <param name="array">The array of characters to be shuffled.</param>
private void ShuffleArray(char[] array)
{
    int arrayLength = array.Length;
    for (int currentIndex = 0; currentIndex < arrayLength; currentIndex++)
    {
        int randomIndex = rnd.Next(arrayLength);
        char temporary = array[currentIndex];
        array[currentIndex] = array[randomIndex];
        array[randomIndex] = temporary;
    }
}


/// <summary>
/// Checks if it's safe to place a number in the specified cell based on Sudoku rules.
/// </summary>
/// <param name="row">The row index of the cell.</param>
/// <param name="col">The column index of the cell.</param>
/// <param name="num">The number (character) to be placed in the cell.</param>
/// <returns>True if placing the number is safe, otherwise false.</returns>
private bool IsSafeToPlace(int row, int col, char num)
{
    // Check row
    for (int colC = 0; colC < GameSettings.BoardSide; colC++)
        if (solvedPuzzle[row, colC] == num)
            return false;
```

```
    // Check column
    for (int rowC = 0; rowC < GameSettings.BoardSide; rowC++)
        if (solvedPuzzle[rowC, col] == num)
            return false;

    // Check box
    int boxStartRow = (row / GameSettings.BoxHeight) * GameSettings.BoxHeight;
    int boxStartCol = (col / GameSettings.BoxWidth) * GameSettings.BoxWidth;

    for (int rowC = 0; rowC < GameSettings.BoxHeight; rowC++)
        for (int colC = 0; colC < GameSettings.BoxWidth; colC++)
            if (solvedPuzzle[boxStartRow + rowC, boxStartCol + colC] == num)
                return false;

    return true;
}

/// <summary>
/// Converts a number (1-15) to its corresponding hexadecimal character ('0'-'F').
/// </summary>
/// <param name="num">The number to be converted.</param>
/// <returns>The corresponding hexadecimal character.</returns>
private char ConvertToHexa(int num)
{
    if (num < Constants.NUM_DIGITS)
        return (char)('0' + num);
    return (char)('A' + num - Constants.NUM_DIGITS);
}

/// <summary>
/// Randomly selects a specified number of initial cells to be pre-filled in the puzzle.
/// </summary>
/// <param name="amountOfInitCells">The number of initial cells to select.</param>
private void SelectInitialCells(int amountOfInitCells)
```

```
{
    Random rnd = new Random();
    while (amountOfInitCells != 0)
    {
        int i = rnd.Next(GameSettings.BoardSide);
        int j = rnd.Next(GameSettings.BoardSide);

        if (!initialCells[i, j])
        {
            amountOfInitCells--;
            initialCells[i, j] = true;
        }
    }
}

/// <summary>
/// Generates a code representing the current state of the puzzle.
/// </summary>
/// <returns>A string representing the puzzle state code.</returns>
private string CreatePuzzleCode()
{
    string puzzleCode = $"{GameSettings.BoxHeight},{GameSettings.BoxWidth}:";
    for (int i = 0; i < solvedPuzzle.GetLength(0); i++)
    {
        for (int j = 0; j < solvedPuzzle.GetLength(1); j++)
        {
            puzzleCode += $"{solvedPuzzle[i, j]},";
            puzzleCode += initialCells[i, j] ? "V" : "X";
            puzzleCode += "|";
        }
    }
    return Code.Protect(puzzleCode.Substring(0, puzzleCode.Length - 1));
}

/// <summary>
```

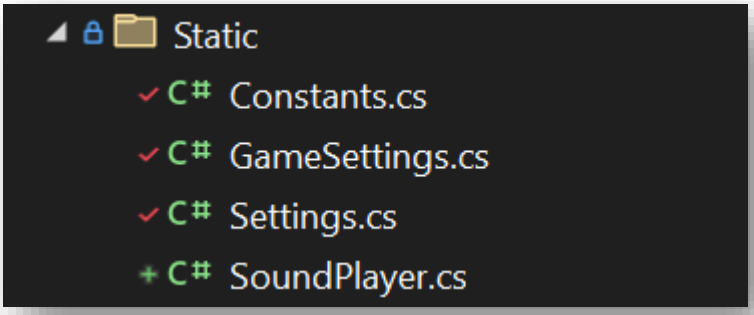
```
        /// Imports a puzzle code and initializes game settings, initial cells, and solved puzzle matrix.
        /// </summary>
        /// <param name="puzzleCode">The puzzle code to import.</param>
        private void ImportPuzzleCode(string puzzleCode)
        {
            // Unprotect puzzle code if necessary
            puzzleCode = Code.Unprotect(puzzleCode);

            // Determine indices for parsing settings
            int settingEnd = puzzleCode.IndexOf(":");
            int separator = puzzleCode.IndexOf(",");

            // Parse game settings
            GameSettings.BoxHeight = int.Parse(puzzleCode.Substring(0, separator));
            GameSettings.BoxWidth = int.Parse(puzzleCode.Substring(separator + 1, settingEnd - separator - 1));
            GameSettings.BoardSide = GameSettings.BoxWidth * GameSettings.BoxHeight;

            // Initialize arrays for initial cells and solved puzzle
            initialCells = new bool[GameSettings.BoardSide, GameSettings.BoardSide];
            solvedPuzzle = new char[GameSettings.BoardSide, GameSettings.BoardSide];
            // Parse initial cells and solved puzzle from puzzle code
            int startIdx = settingEnd + 1;
            for (int i = 0; i < GameSettings.BoardSide; i++)
            {
                for (int j = 0; j < GameSettings.BoardSide; j++)
                {
                    separator = puzzleCode.IndexOf(',', startIdx + 1);
                    solvedPuzzle[i, j] = puzzleCode[startIdx];
                    initialCells[i, j] = puzzleCode[separator + 1] == 'V';
                    startIdx = puzzleCode.IndexOf('|', startIdx + 1) + 1;
                }
            }
        }
    }
}
```

עמודים



התיקייה מכילה את כל העמודים המשומשים בפרוייקט ראה את העמודים, את תפקידם ואת קודם ראה ב " דפים ". ראה שיטת ניווט ב "מפת הדפים"

מחלקות סטטיות



תיקיה המכילה את המחלקות הסטטיות שמטרתן להחזיק מידע מוכר לכולם במהלך ריצת התכנית.

Constants – קובץ המכיל מחלקת קבועים שבתוכה תת מחלקות של קבועים.

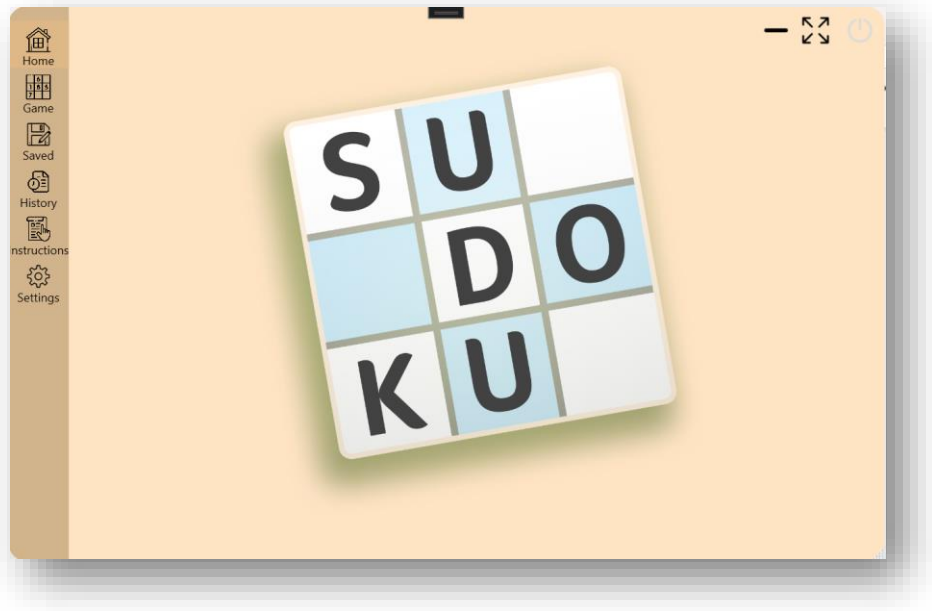
SoundPlayer - מחלקה שמטפלת בענייני שמע כמו צלילים ומוזיקה.

GameSettings – מחלקה השומרת את הגדרות המשחק הנוחכי

Settings – מחלקה השומרת את ההגדרות הנוכחיות

דפים

דף בית



העמוד מכיל לוגו גדול של סודוקו למטרות יופי שעולה ויורד בלופ בעזרת (double animation).

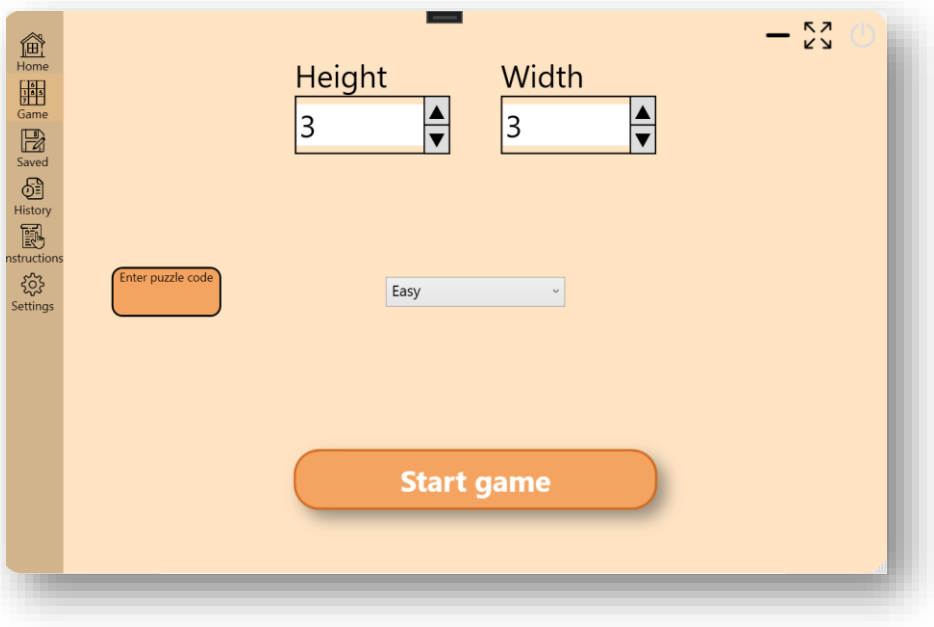
עמוד פתיחה – CS

```
using System.Windows.Controls;


namespace Sudoku_WPF
{
    /// <summary>
    /// Interaction logic for OpenningPage.xaml
    /// </summary>
    public partial class OpenningPage : Page
    {
        public OpenningPage()
        {
            InitializeComponent();
        }


    }
}
```

עמוד פתיחה - XAML

```
<Page x:Class="Sudoku_WPF.OpenningPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:Sudoku_WPF"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="OpenningPage">


    <Grid HorizontalAlignment="center" VerticalAlignment="Center" MaxHeight="700"
MaxWidth="700">
        <Viewbox  Stretch="Uniform">
            <Grid >
                <Image Source="\Assets\Images\sudoku_pic.png"  Grid.Column="1"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
                    <Image.Style>
                        <Style TargetType="Image">
                            <Setter Property="RenderTransformOrigin" Value="0.5,0.5"/>
                            <Setter Property="RenderTransform">
                                <Setter.Value>
                                    <TranslateTransform/>
                                </Setter.Value>
                            </Setter>
                            <Style.Triggers>
                                <EventTrigger RoutedEvent="Loaded">
                                    <BeginStoryboard>
                                        <Storyboard RepeatBehavior="Forever">
                                            <DoubleAnimation
Storyboard.TargetProperty="(Image.RenderTransform).(TranslateTransform.Y)"
                                         To="-7" Duration="0:0:0.5" AutoReverse="True"/>
                                        </Storyboard>
                                    </BeginStoryboard>
                                </EventTrigger>
                            </Style.Triggers>
                        </Style>
                    </Image.Style>
                </Image>
            </Grid>
        </Viewbox>



    </Grid>
</Page>
```

דף יצירת משחק



מטרתו לקבוע את הגדרות המשחק:

העמוד מאפשר יצירת משחק בשתי דרכים, יצירת משחק חדש בגודל הנקבע לפי ה NumericUpDown ורמת קושי הנקבעת בערך המוכנס ב Combo box.

הוא מאפשר גם הדבקת קוד פאזל (אפשר לשלוח בווצאפ) להתחלת משחק שנוצר בעבר מחדש. הקוד מוצפן לכן אין כל סכנה שבהדבקה המשתמש ישיג מידע על פתרון הלוח.

```
using ControlLib;
using Sudoku_WPF.GameClasses;
using Sudoku_WPF.publico;
using Sudoku_WPF.Themes;
using System;
using System.Collections.Generic;
using System.Diagnostics.Metrics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using static Sudoku_WPF.publico.Constants;

namespace Sudoku_WPF
{
    /// <summary>
    /// Interaction logic for SettingsPage.xaml
    /// </summary>
    public partial class GameSettingsPage : Page
    {
        public Page gamePage;

        /// <summary>
        /// Constructor for initializing the GameSettingsPage.
        /// Sets initial theme, max values for number up down controls, and event
handlers.
        /// </summary>
        public GameSettingsPage()
        {
            InitializeComponent();

            SetInitialTheme(GameSettings.dificultyLevel.ToString());

            NUD_boxHeight.Value = GameSettings.BoxHeight;
            NUD_boxWidth.Value = GameSettings.BoxWidth;



            gamePage = null;
        }

        /// <summary>
        /// Sets the initial theme based on the provided theme name.
        /// </summary>
        /// <param name="themeName">The name of the theme to set.</param>
        private void SetInitialTheme(string themeName)
        {
            // Find the ComboBoxItem with the specified theme name and set it as
selected
            var selectedItem = DificultyLevel_CMBB.Items
                                .OfType<ComboBoxItem>()
                                .FirstOrDefault(item => item.Content.ToString() ==
themeName);
```

```
            if (selectedItem != null)
            {
                DificultyLevel_CMBB.SelectedItem = selectedItem;
            }
        }

        private void codeTextBox_TextChanged(object sender, TextChangedEventArgs e)
        {
            TextBox textBox = sender as TextBox;

            if (textBox == codeTxtBox)
            {
                codePlaceholder.Visibility = string.IsNullOrWhiteSpace(codeTxtBox.Text)
? Visibility.Visible : Visibility.Hidden;
            }
        }


        /// <summary>
        /// Event handler for selection changes in the difficulty level combo box.
        /// Updates the game settings with the selected difficulty level.
        /// </summary>
        private void DificultyLevel_CMBB_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            string option = "";
            ComboBoxItem cbi = (ComboBoxItem)DificultyLevel_CMBB.SelectedValue;
            option = cbi.Content.ToString();

            if (Enum.TryParse(option, out DificultyLevel difLvl))
            {
                GameSettings.dificultyLevel = difLvl;
            }
        }

        /// <summary>
        /// Event handler for clicking the game starter button.
        /// Plays a sound and initiates the game based on user input or puzzle code.
        /// </summary>
        private void GameStarterBtn_Click(object sender, RoutedEventArgs e)
        {
            SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);

            var window = (MainWindow)Application.Current.MainWindow;
            if (codeTxtBox.Text == "" || codeTxtBox.Text == "Enter puzzle code")
            {
                GameSettings.BoxHeight = Convert.ToInt32(NUD_boxHeight.Value);
                GameSettings.BoxWidth = Convert.ToInt32(NUD_boxWidth.Value);
                GameSettings.BoardSide = GameSettings.BoxWidth * GameSettings.BoxHeight;
                window.gamePage = new GamePage();
                NavigationService.Navigate(window.gamePage);
                //window.Settings_btn.Visibility = Visibility.Collapsed;
            }
            else if (IsValidCode(Code.Unprotect(codeTxtBox.Text)))
            {
                window.gamePage = new GamePage(codeTxtBox.Text);
                NavigationService.Navigate(window.gamePage);
            }
            else
            {
                MessageBox.Show("Puzzle code invalid! Enter it again");
                this.codeTxtBox.Text = "";
            }
        }
```

```
        /// <summary>
        /// Checks if the provided puzzle code is valid.
        /// </summary>
        /// <param name="code">The puzzle code to validate.</param>
        /// <returns>True if the puzzle code is valid; otherwise, false.</returns>
        private bool IsValidCode(string code)
        {
            if (code == null)
            {
                return false;
            }

            // Check for the presence of the colon and commas
            int settingEnd = code.IndexOf(":");
            int firstSeparator = code.IndexOf(",");

            if (settingEnd == -1 || firstSeparator == -1 || firstSeparator >=
settingEnd)
            {
                return false;
            }

            // Check if box height and width are integers and positive
            if (!int.TryParse(code.Substring(0, firstSeparator), out int boxHeight) ||
boxHeight <= 0)
            {
                return false;
            }

            if (!int.TryParse(code.Substring(firstSeparator + 1, settingEnd -
firstSeparator - 1), out int boxWidth) || boxWidth <= 0)
            {
                return false;
            }

            int boardSide = boxWidth * boxHeight;

            // Check the puzzle data length and format
            string puzzleData = code.Substring(settingEnd + 1);
            string[] puzzleParts = puzzleData.Split('|');

            if (puzzleParts.Length != boardSide * boardSide)
            {
                return false;
            }

            // Check each puzzle part for valid characters and format
            foreach (string part in puzzleParts)
            {
                int commaIndex = part.IndexOf(',');
                if (commaIndex == -1 || commaIndex != 1 || (part[commaIndex + 1] != 'V'
&& part[commaIndex + 1] != 'X'))
                {
                    return false;
                }

                char puzzleChar = part[0];
                if (!Cell.IsValidInput(puzzleChar) && puzzleChar != '.')
                {
                    return false;
                }
            }

            return true;
```

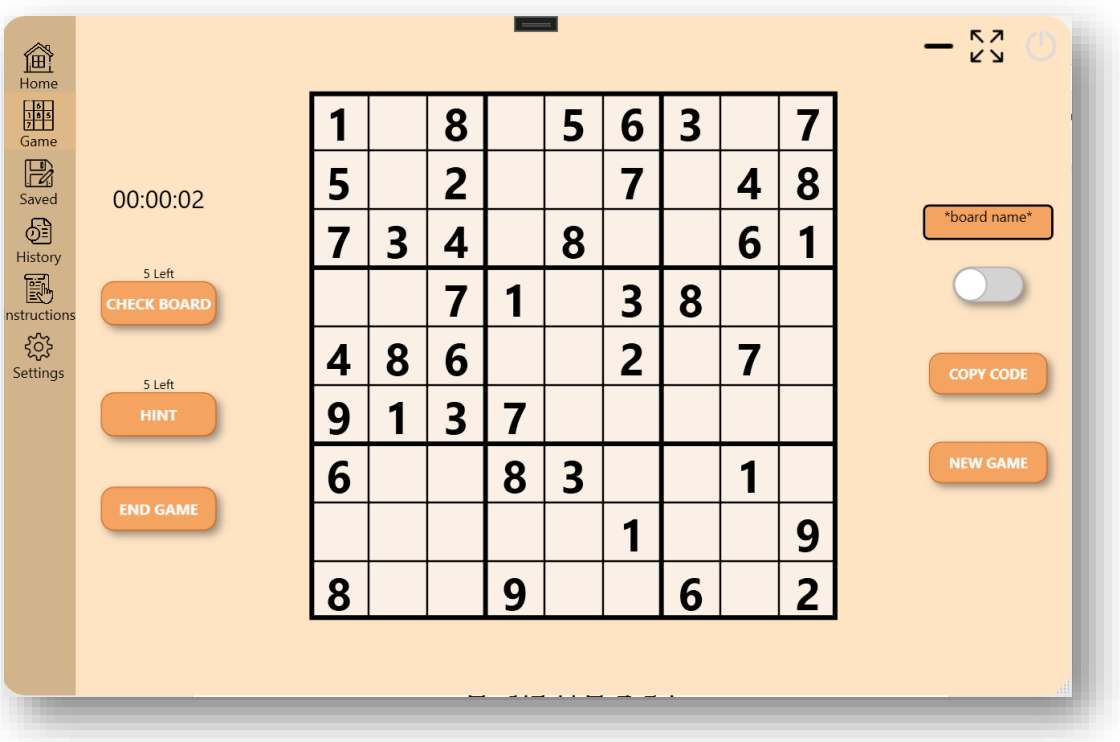
```
        }
    }
}
```

XAML – עמוד הגדרות משחק

```
<Page x:Class="Sudoku_WPF.GameSettingsPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:Sudoku_WPF"
      xmlns:lib= "clr-namespace:ControlLib;assembly=NumericUpDown"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="SettingsPage">

    <Grid >
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>

        <Viewbox HorizontalAlignment="Center" VerticalAlignment="Center"
Grid.Column="2" MaxHeight="100" Margin="10">
            <StackPanel Orientation="Vertical">
                <TextBlock Text="Width" Foreground="{DynamicResource Text}"
FontSize="20" />

                <lib:NumericUpDown x:Name="NUD_boxWidth" MinValue="2" MaxValue="4"
             FontSize="20" />
            </StackPanel>
        </Viewbox>

        <Viewbox HorizontalAlignment="Center" VerticalAlignment="Center"
Grid.Column="1" MaxHeight="100" Margin="10">
            <StackPanel Orientation="Vertical"    Grid.Column="1">
                <TextBlock Text="Height" Foreground="{DynamicResource Text}"
FontSize="20" />

                <lib:NumericUpDown x:Name="NUD_boxHeight" MinValue="2" MaxValue="5"
FontSize="20"/>
            </StackPanel>

        </Viewbox>



        <ComboBox  Name="DificultyLevel_CMBB"
SelectionChanged="DificultyLevel_CMBB_SelectionChanged"  Height="30" Width="180"
                   Grid.Column="1" Grid.Row="1" Grid.ColumnSpan="2"
HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="15">
            <ComboBoxItem Content="Easy" />
            <ComboBoxItem Content="Medium" />
            <ComboBoxItem Content="Hard" />

        </ComboBox>

        <Viewbox Grid.Column="1" Grid.Row="2" Grid.ColumnSpan="2">
            <Button x:Name = "btn_gameStarterBtn" Width="150" Height="25"
Style="{DynamicResource RoundedButtonStyle}" Click = "GameStarterBtn_Click"  Content
="Start game"  HorizontalAlignment="Center" VerticalAlignment="Center"  Margin="10"/>
```

```
        </Viewbox>



        <Border Background="{DynamicResource Botton_BG}"
                BorderBrush="{DynamicResource Border}"
                CornerRadius="10"
                Grid.Row="1"
                Grid.Column="0"
                BorderThickness="2"
                Height="50"
                Width="110">

            <Grid >
                <TextBox x:Name="codeTxtBox"
                     Foreground="{DynamicResource Text}"
                     TextChanged="codeTextBox_TextChanged"
                     Background="Transparent"
                     BorderBrush="Transparent"
                     FontSize="12"
                     TextAlignment="Center"/>
                <TextBlock x:Name="codePlaceholder"
                       Text="Enter puzzle code"
                       Foreground="{DynamicResource Text}"
                       IsHitTestVisible="False"
                       Background="Transparent"
                       FontSize="12"
                       TextAlignment="Center"/>

            </Grid>


        </Border>




    </Grid>
</Page>
```

דף משחק



עמוד משחק בו משחקים. מאפשר:

הכנסת שם לוח

עצירה זמנית של המשחק

העתקה אוטומטית של קוד

יצירת משחק חדש (ושמירת הלוח הזה)

סיום לוח והצגת הפתרון (ושמירת הלוח הזה)

תצוגת הזמן שעבר עד כה

בדיקת לוח

רמז (קבלת פתרון המשבצת בפוקוס)

עמוד משחק – CS

```
using Sudoku_WPF.publico;
using System;
using System.Data.OleDb;
using System.Diagnostics.Eventing.Reader;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Navigation;
using DAL;
using Sudoku_WPF.GameClasses;
using static Sudoku_WPF.publico.Constants;

namespace Sudoku_WPF
{
    /// <summary>
    /// Interaction logic for GamePage.xaml
    /// </summary>
    public partial class GamePage : Page
    {
        public Game game;
        private bool isLoaded = true;
        private int hintsLeft;
        private int checksLeft;

        // Default constructor for XAML
        public GamePage()
        {
            InitializeComponent();
            Init();
            game = new Game(SudokuGrid, timerTxtB);
        }

        /// <summary>
```

```
/// Constructor for initializing GamePage with a specific puzzle code.
/// </summary>
/// <param name="puzzleCode">The puzzle code to initialize the game with.</param>
public GamePage(string puzzleCode)
{
    InitializeComponent();
    Init();
    game = new Game(SudokuGrid, timerTxtB, puzzleCode);
}

/// <summary>
/// Constructor for initializing GamePage with game information.
/// </summary>
/// <param name="gameInfo">The game information to initialize the game with.</param>
public GamePage(GameInfo gameInfo)
{
    InitializeComponent();
    Init();
    this.nameTxtB.Text = gameInfo.Name;
    this.nameTxtB.IsReadOnly = true;

    this.hintsLeft -= gameInfo.Hints;
    this.checksLeft -= gameInfo.Checks;

    if(hintsLeft == 0)
    {
        btn_hint.IsEnabled = false;
    }
    if (checksLeft == 0)
    {
        btn_checkBoard.IsEnabled = false;
    }

    hintsTxtB.Text = hintsLeft.ToString() + GameConstants.REMEINING_STR;
```

```
    checksTxtB.Text = checksLeft.ToString() + GameConstants.REMEINING_STR;

    game = new Game(SudokuGrid, timerTxtB, gameInfo);
}

/// <summary>
/// Retrieves the current game information.
/// </summary>
/// <param name="solved">Whether the game is solved or not.</param>
/// <param name="current">Whether the game is the current game or not.</param>
/// <returns>The GameInfo object representing the current game state.</returns>
public GameInfo GetGameInfo(bool solved, bool current)
{
    int Id = DBHelper.GetNextId("tbl_games", DBConstants.Games_Parameters.Id);
    nameTxtB.Text = string.IsNullOrEmpty(nameTxtB.Text) ? "Board" + Id.ToString() : nameTxtB.Text;

    GameInfo gameInfo = new GameInfo(
        Id,
        nameTxtB.Text,
        this.game.Board.GenerateBoardCode(),
        this.game.GetPuzzleCode(),
        this.game.GetTime(),
        GameConstants.HINTS - hintsLeft,
        GameConstants.CHECKS - checksLeft,
        DateTime.Now.ToString(),
        GameSettings.dificultyLevel,
        solved,
        current,
        GameSettings.BoxHeight,
        GameSettings.BoxWidth
        );
    return gameInfo;
}
```

```
/// <summary>
/// Initializes the GamePage by setting up initial values and event handlers.
/// </summary>
private void Init()
{
    this.hintsLeft = GameConstants.HINTS;
    this.checksLeft = GameConstants.CHECKS;

    hintsTxtB.Text = hintsLeft.ToString() + GameConstants.REMEINING_STR;
    checksTxtB.Text = checksLeft.ToString() + GameConstants.REMEINING_STR;

    this.Loaded += Page_Loaded;
    this.Unloaded += Page_Unloaded;
}

/// <summary>
/// Event handler when the GamePage is loaded into the visual tree.
/// Starts the game timer and subscribes to navigation events.
/// </summary>
private void Page_Loaded(object sender, RoutedEventArgs e)
{
    // Subscribe to the Navigating event
    if (NavigationService != null)
    {
        NavigationService.Navigating += OnNavigatingFrom;
        if(Board.FocusedCell() != null)
        {
            Board.FocusedCell().Focus();
        }

        game.Timer.Start();
    }
}
```

```
/// <summary>
/// Event handler when the GamePage is unloaded from the visual tree.
/// Stops the game timer and unsubscribes from navigation events.
/// </summary>
private void Page_Unloaded(object sender, RoutedEventArgs e)
{
    // Unsubscribe from the Navigating event
    if (NavigationService != null)
    {
        NavigationService.Navigating -= OnNavigatingFrom;
        Board.FocusedCell().Focus();
        game.Timer.Stop();
    }
}

/// <summary>
/// Event handler for navigating away from the GamePage.
/// Stops the game timer based on navigation actions.
/// </summary>
/// <param name="sender">The sender of the event.</param>
/// <param name="e">The event arguments.</param>
private void OnNavigatingFrom(object sender, NavigatingCancelEventArgs e)
{
    // Check if the current content is GamePage and we're navigating away
    if (NavigationService != null && NavigationService.Content is GamePage && isLoaded)
    {
        isLoaded = false;
        game.Timer.Stop();
    }
    if (e.Content is GamePage)
    {
        isLoaded = true;
        game.Timer.Start();
    }
```

```
    }

    /// <summary>
    /// Disables the game controls and stops certain functionalities.
    /// </summary>
    public void DisableGameControls()
    {
        this.game.Board.Disable();
        this.btn_pause.Visibility = Visibility.Collapsed;
        this.btn_hint.IsEnabled = false;
        this.btn_checkBoard.IsEnabled = false;
        this.timerTxtB.IsEnabled = false;
        this.nameTxtB.IsEnabled = false;
        this.btn_endGame.Visibility = Visibility.Collapsed;
    }

    /// <summary>
    /// Event handler for ending the game.
    /// Shows solution if to end
    /// </summary>
    private void EndGame_Click(object sender, RoutedEventArgs e)
    {
        SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);

        MessageBoxResult msbxRes = MessageBox.Show("Are you sure you want to end this game? It won't be saved", "End
Game", MessageBoxButton.YesNo, MessageBoxImage.Warning);
        if (msbxRes == MessageBoxResult.Yes)
        {
            try
            {
                Mouse.OverrideCursor = Cursors.Wait; // Change cursor to
                game.End(false, false);
                game.Board.ShowSolution();
            }
            finally
```

```
        {
            Mouse.OverrideCursor = null; // Restore cursor
        }
    }
}

/// <summary>
/// Event handler for using a hint during the game.
/// Plays a sound and applies a hint to the focused cell.
/// </summary>
private void Hint_Click(object sender, RoutedEventArgs e)
{
    if (hintsLeft == 1)
    {
        btn_hint.IsEnabled = false;
    }
    Cell focusCell = Board.FocusedCell();
    if (focusCell != null && !focusCell.IsReadOnly)
    {
        if(focusCell.Text != focusCell.solvedValue)
        {
            SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);
            Cell cell =  Board.FocusedCell();
            cell.Solve(true);
            cell.notesGrid.Clear();
            RemoveHint();
        }
        else
        {
            focusCell.IsReadOnly = true;
            SoundPlayer.PlaySound(SoundConstants.WRONG);
        }
    }
    else
```

```
        {

            SoundPlayer.PlaySound(SoundConstants.WRONG);
        }
    }

    /// <summary>
    /// Event handler for starting a new game.
    /// Asks to save the current game and navigates to game settings.
    /// </summary>
    private void NewGame_Click(object sender, RoutedEventArgs e)
    {
        SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);

        if (this.game.IsInGame())
        {
            MessageBoxResult msbxRes = MessageBox.Show("Do you want to save this game?", "Save Game",
MessageBoxButton.YesNoCancel);

            if (msbxRes == MessageBoxResult.Yes || msbxRes == MessageBoxResult.No)
            {
                game.End(false, msbxRes == MessageBoxResult.Yes);
                NavigationService.Navigate(UriConstants.GAME_SETTINGS_PAGE);
            }
        }
        else
        {
            NavigationService.Navigate(UriConstants.GAME_SETTINGS_PAGE);
            return;
        }
    }

    /// <summary>
    /// Removes a hint and updates the remaining hints count.
    /// </summary>
```

```
private void RemoveHint()
{
    this.hintsLeft--;
    hintsTxtB.Text = hintsLeft.ToString() + GameConstants.REMEINING_STR;
}

/// <summary>
/// Removes a check and updates the remaining checks count.
/// </summary>
private void RemoveCheck()
{
    this.checksLeft--;
    checksTxtB.Text = checksLeft.ToString() + GameConstants.REMEINING_STR;
}

/// <summary>
/// Copies the current puzzle code to the clipboard.
/// </summary>
private void CopyPuzzleCode_Click(object sender, RoutedEventArgs e)
{
    SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);

    Clipboard.SetText(game.GetPuzzleCode());
    MessageBox.Show(GameConstants.COPIED_STR);
}

/// <summary>
/// Limits the text length in a TextBox to 20 characters.
/// </summary>
private void nameTextBox_TextChanged(object sender, TextChangedEventArgs e)
{
    TextBox textBox = sender as TextBox;

    // If the length exceeds the limit, truncate the text
```

```
    if (textBox.Text.Length > 20)
    {
        textBox.Text = textBox.Text.Substring(0, 20);

        // Set the caret to the end of the truncated text
        textBox.CaretIndex = textBox.Text.Length;
    }

    if (textBox == nameTxtB)
    {
        nameTextPlaceholder.Visibility = string.IsNullOrWhiteSpace(nameTxtB.Text) ? Visibility.Visible : Visibility.Hidden;
    }
}

/// <summary>
/// Event handler for checking the current board configuration.
/// Plays a sound and checks the board with the remaining checks count.
/// </summary>
private void CheckBoard_Click(object sender, RoutedEventArgs e)
{
    SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);

    if (checksLeft == 1)
    {
        btn_checkBoard.IsEnabled = false;
    }


    this.game.Board.CheckMyBoard();

    RemoveCheck();
}


/// <summary>
```

```
/// Handles the toggled state of the pause button, pausing and resuming the game.
/// </summary>
private void Pause_Checked(object sender, RoutedEventArgs e)
{
    SoundPlayer.PlaySound(SoundConstants.ON_OFF);

    game.Timer.Stop();
    DisablePauseControls();
}

private void Pause_Unchecked(object sender, RoutedEventArgs e)
{
    SoundPlayer.PlaySound(SoundConstants.ON_OFF);
    game.Timer.Start();
    EnablePauseControls();
}

/// <summary>
/// Disables all game controls when the game is paused.
/// </summary>
private void DisablePauseControls()
{
    SudokuGrid.IsEnabled = false;
    btn_showPuzzleCode.IsEnabled = false;
    btn_checkBoard.IsEnabled = false;
    btn_hint.IsEnabled = false;
}

/// <summary>
/// Enables all game controls when the game is resumed.
/// </summary>
private void EnablePauseControls()
{
    SudokuGrid.IsEnabled = true;
```

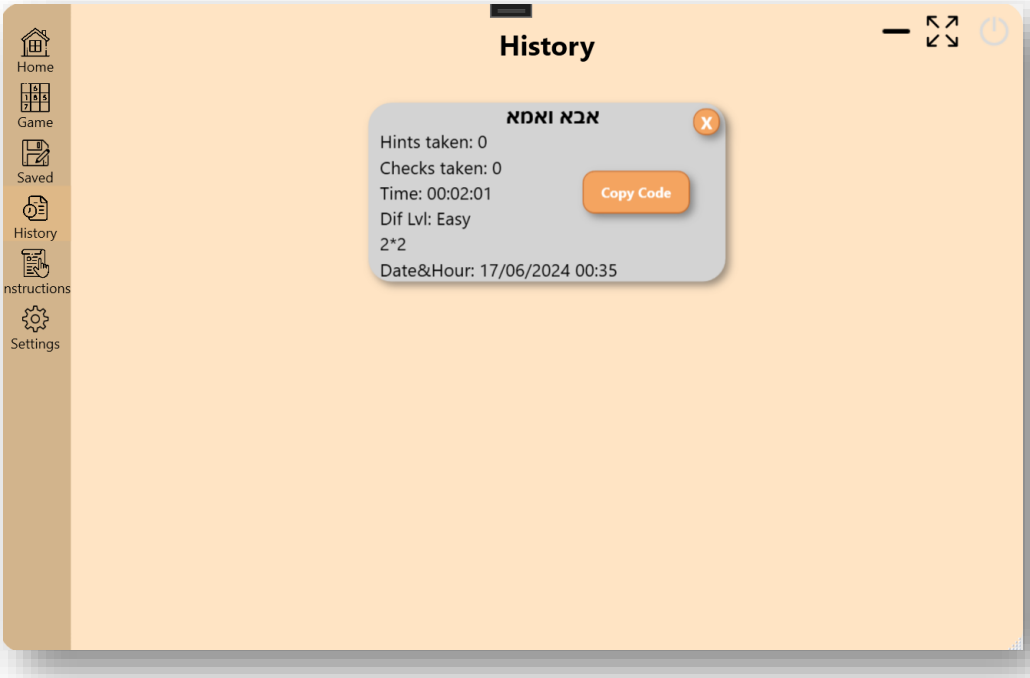
```
            btn_showPuzzleCode.IsEnabled = true;
            btn_checkBoard.IsEnabled = true;
            btn_hint.IsEnabled = true;
        }

        /// <summary>
        /// Property to access the Game instance associated with this GamePage.
        /// </summary>
        public Game Game => this.game;
    }
}
```

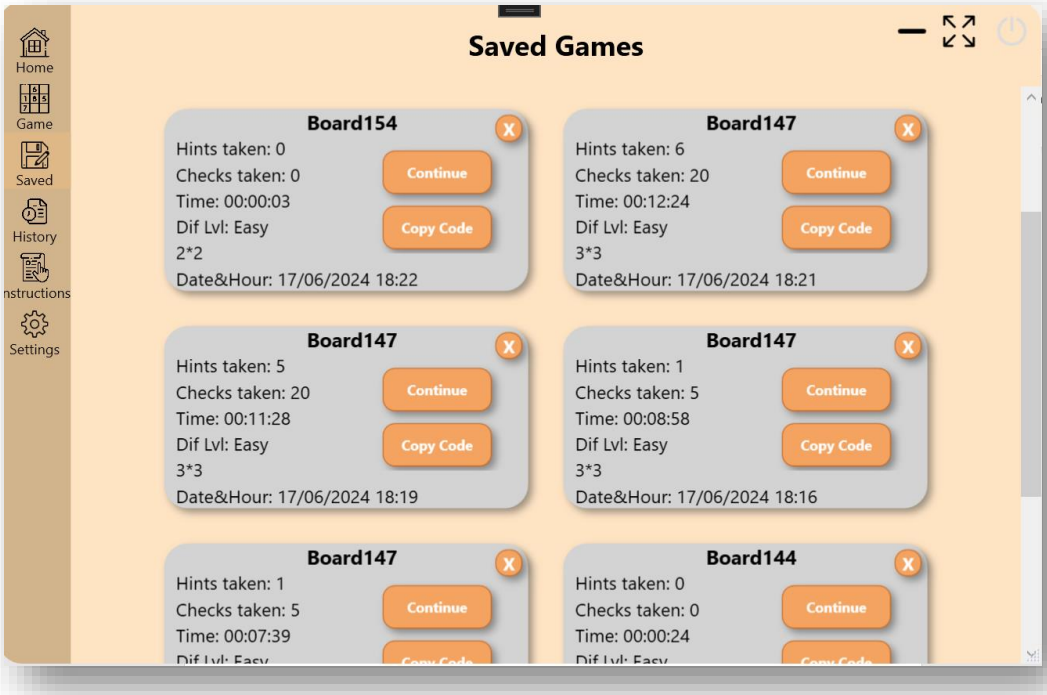
עמוד משחק - XAML

```
<Page x:Class="Sudoku_WPF.GamePage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:Sudoku_WPF"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="GamePage"
      KeepAlive="True">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition Width="4*"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition Height="7*"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Grid Grid.Column="1" Grid.Row="1">
            <Viewbox Stretch="Uniform">
                <Grid Width="500" Height="500" x:Name="SudokuGrid"/>
            </Viewbox>
        </Grid>
        <Viewbox Stretch="Uniform" Grid.Column="2" Grid.Row="1">
            <StackPanel Orientation="Vertical" HorizontalAlignment="Stretch">
                <Border Background="{DynamicResource Botton_BG}"
                        BorderBrush="{DynamicResource Border}"
                        CornerRadius="4"
                        BorderThickness="2"
                        Height="30"
                        Width="110">
                    <Grid>
                        <TextBox x:Name="nameTxtB"
                                 Foreground="{DynamicResource Text}"
                                 TextChanged="nameTextBox_TextChanged"
                                 Background="Transparent"
                                 BorderBrush="Transparent"
                                 FontSize="12"
                                 TextAlignment="Center"/>
                        <TextBlock x:Name="nameTextPlaceholder"
                                   Text="*board name*"
                                   Foreground="{DynamicResource Text}"
                                   IsHitTestVisible="False"
                                   Background="Transparent"
                                   FontSize="12"
                                   TextAlignment="Center"/>
                    </Grid>
                </Border>
                <ToggleButton Style="{StaticResource DAToggleButton}"
x:Name="btn_pause" Checked="Pause_Checked" Unchecked="Pause_Unchecked"
Content="PAUSE" Width="100" Margin="20" Height="35" HorizontalAlignment="Stretch"/>
```

```
                <Button Style="{StaticResource RoundedButtonStyle}"
x:Name="btn_showPuzzleCode" Click="CopyPuzzleCode_Click" Margin="20" Content="COPY
CODE" Height="35" HorizontalAlignment="Stretch"/>
                <Button Style="{StaticResource RoundedButtonStyle}"
x:Name="btn_newGame" Click="NewGame_Click" Margin="20" Content="NEW GAME"
Height="35" HorizontalAlignment="Stretch"/>
            </StackPanel>
        </Viewbox>
        <Viewbox Stretch="Uniform" Grid.Column="0" Grid.Row="1">
            <StackPanel Orientation="Vertical" HorizontalAlignment="Stretch">
                <TextBlock x:Name="timerTxtB" Foreground="{DynamicResource Text}"
FontSize="20" Margin="20" HorizontalAlignment="Center"/>
                <StackPanel Margin="20">
                    <TextBlock x:Name="checksTxtB" Text="0 left"
Foreground="{DynamicResource Text}" FontSize="10" HorizontalAlignment="Center"/>
                    <Button Style="{StaticResource RoundedButtonStyle}"
x:Name="btn_checkBoard" Click="CheckBoard_Click" Content=" CHECK BOARD " Height="35"
HorizontalAlignment="Stretch"/>
                </StackPanel>
                <StackPanel Margin="20">
                    <TextBlock x:Name="hintsTxtB" Text="0 left"
Foreground="{DynamicResource Text}" FontSize="10" HorizontalAlignment="Center"/>
                    <Button Style="{StaticResource RoundedButtonStyle}"
x:Name="btn_hint" Click="Hint_Click" Content="HINT" Height="35"
HorizontalAlignment="Stretch"/>
                </StackPanel>
                <Button Style="{StaticResource RoundedButtonStyle}"
x:Name="btn_endGame" Click="EndGame_Click" Content="END GAME" Margin="20"
Height="35" HorizontalAlignment="Stretch"/>
            </StackPanel>
        </Viewbox>
    </Grid>
</Page>
```

דף הסטוריה ודף משחקים שמורים





העמודים מציגים משחקים שנשמרו (בזיכרון של התכנה ובמסד הנתונים).

עמוד ההסטוריה מראה את המידע על המשחקים שפתרו בהצלחה ואפשרות להעתיק את קודם.

עמוד המשחקים השמורים מציג את המידע על המשחקים השמורים ומאפשר להמשיכם, למחוקם ולהעתיק קודם.

עמוד ההסטוריה והשמירה הם מופעים שונים של אותו עצם (עמוד) ולהם אותו קוד,

לכן אקרא להם "עמוד שומר".

עמוד שומר - CS

```
 using DAL;
using Sudoku_WPF.GameClasses;
using Sudoku_WPF.publico;
using System;
using System.Collections.Generic;
using System.Data.OleDb;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Effects;
using static Sudoku_WPF.publico.Constants;

namespace Sudoku_WPF.Pages
{
    /// <summary>
    /// Interaction logic for SavedPage.xaml
    /// </summary>
    public partial class SaverPage : Page
    {
        private List<GameInfo> games = new List<GameInfo>(); // List to store game information
        private List<Border> Items = new List<Border>(); // List to store UI border elements
        private bool isHistory; // Flag indicating if the page is showing history or saved games

        /**
         * Constructor for SaverPage class.
         * Initializes a new instance of the SaverPage.
         *
         * input: isHistory - Flag indicating if the page displays history or saved games.
         * output: None
         */
        public SaverPage(bool isHistory)
        {
            InitializeComponent();
```

```
    this.isHistory = isHistory;
    title_Txb.Text = isHistory ? "History" : "Saved Games";

    for (int i = 0; i < Items.Count; i++)
    {
        Border item = Items[i];
        item.Tag = i;
        UpdateButtonTags(item, i); // Update button tags in the UI
    }
}

/**
 * Adds a game item to the list and database.
 *
 * input: gameInfo - GameInfo object containing game information to add.
 * output: None
 */
public void AddItemToListAndDB(GameInfo gameInfo)
{
    AddItemToList(gameInfo);
    InsertGame(gameInfo);
}

/**
 * Adds a game item to the list.
 *
 * input: gameInfo - GameInfo object containing game information to add.
 * output: None
 */
public void AddItemToList(GameInfo gameInfo)
{
    games.Add(gameInfo);

    Border border = new Border
    {
        Margin = new Thickness(SaverConstants.MARGIN),
        Height = SaverConstants.HEIGHT,
```

```
    Width = SaverConstants.HEIGHT * 2,
    CornerRadius = new CornerRadius(SaverConstants.CORNER_RADIUS),
    Tag = games.Count - 1 // Store index of gameInfo in games list
};

// Add shadow effect
border.Effect = new DropShadowEffect
{
    Color = Colors.Gray,
    BlurRadius = 10,
    ShadowDepth = 5,
    Opacity = 0.5
};

border.SetResourceReference(BackgroundProperty, ColorConstants.HistoryItem_BG);

Grid grid = new Grid();
grid.ColumnDefinitions.Add(new ColumnDefinition());
grid.ColumnDefinitions.Add(new ColumnDefinition());

Button deleteGameBtn = new Button
{
    Style = FindResource("RoundedButtonStyle") as Style,
    Height = 23,
    Width = 23,
    Content = "X",
    HorizontalAlignment = HorizontalAlignment.Right,
    VerticalAlignment = VerticalAlignment.Top,
    FontSize = 15,
    Margin = new Thickness(0, 5, 5, 0)
};
Grid.SetColumn(deleteGameBtn, 1);
Grid.SetRowSpan(deleteGameBtn, 3);
grid.Children.Add(deleteGameBtn);

deleteGameBtn.Click += DeleteGame_Click;
deleteGameBtn.Tag = border.Tag; // Set tag to match index in games list
```

```
        double fontSize = SaverConstants.RELATIVE_FONT_SIZE * border.Height;

        AddTextBlockToGrid(grid, gameInfo.Name.ToString(), fontSize, 2, true);
        AddTextBlockToGrid(grid, "Hints taken: " + gameInfo.Hints.ToString(), fontSize);
        AddTextBlockToGrid(grid, "Checks taken: " + gameInfo.Checks.ToString(), fontSize);
        AddTextBlockToGrid(grid, "Time: " + gameInfo.Time, fontSize);
        AddTextBlockToGrid(grid, "Dif Lvl: " + gameInfo.DifficultyLevel.ToString(), fontSize);
        AddTextBlockToGrid(grid, gameInfo.BoxHeight.ToString() + "*" + gameInfo.BoxWidth.ToString(), fontSize);
        AddTextBlockToGrid(grid, "Date&Hour: " + (gameInfo.Date.ToString().Substring(0,
gameInfo.Date.ToString().Length - 3)), fontSize, 2);

        StackPanel btnPanel = CreateBtnPanel(isHistory, gameInfo);
        Grid.SetRowSpan(btnPanel, grid.RowDefinitions.Count - 2);
        Grid.SetRow(btnPanel, 1);
        Grid.SetColumn(btnPanel, 1);
        grid.Children.Add(btnPanel);

        border.Child = grid;
        ItemsWrapPanel.Children.Insert(0, border);

        Items.Add(border); // Add border to Items list
    }

    /**
     * Creates a button panel with specific buttons based on game history status.
     *
     * input: isHistory - Flag indicating if the page displays history or saved games.
     *        gameInfo - GameInfo object containing game information.
     * output: StackPanel containing buttons for the UI.
     */
    private StackPanel CreateBtnPanel(bool isHistory, GameInfo gameInfo)
    {
        StackPanel btnPanel = new StackPanel
        {
            HorizontalAlignment = HorizontalAlignment.Center,
            VerticalAlignment = VerticalAlignment.Center,
```

```
            Orientation = Orientation.Vertical,
            Margin = new Thickness(10) // Adjust margin as needed
        };

        if (!isHistory)
        {
            Button continueBtn = new Button
            {
                Height = SaverConstants.BTN_HEIGHT,
                Width = SaverConstants.BTN_WIDTH, // Adjust width as needed
                Content = "Continue",
                Margin = new Thickness(5),
                Visibility = Visibility.Visible,
                Style = FindResource("RoundedButtonStyle") as Style,
            };

            // Handle the click event for the "Continue" button
            continueBtn.Click += (sender, e) =>
            {
                SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);

                var window = Application.Current.MainWindow as MainWindow;

                // Check if gamePage is already instantiated
                if (window.gamePage == null)
                {
                    window.Game_btn.IsChecked = true;
                    window.gamePage = new GamePage(gameInfo);
                    window.MainFrame.Navigate(window.gamePage);
                    DeleteGame(gameInfo);  // Remove game from the list and database
                    return;
                }

                MessageBoxResult result = MessageBox.Show("Do you want to save the running game?", "Save Game",
MessageBoxButton.YesNoCancel);

                if (result == MessageBoxResult.Yes)
```

```
            {
                window.gamePage.game.End(false, true);  // Save the current game state
                DeleteGame(gameInfo);  // Remove game from the list and database
            }
            else if (result == MessageBoxResult.No)
            {
                window.gamePage.game.End(false, false);  // Do not save the current game state
                DeleteGame(gameInfo);  // Remove game from the list and database
            }

            // Navigate to the new game page
            window.Game_btn.IsChecked = true;
            window.gamePage = new GamePage(gameInfo);
            window.MainFrame.Navigate(window.gamePage);
        };

        btnPanel.Children.Add(continueBtn);
    }

    Button copyPuzzleCodeBtn = new Button
    {
        Height = SaverConstants.BTN_HEIGHT,
        Width = SaverConstants.BTN_WIDTH,
        Content = "Copy Code",
        Margin = new Thickness(5),
        Visibility = Visibility.Visible,
        Style = FindResource("RoundedButtonStyle") as Style,
        Tag = Items.Count
    };
    copyPuzzleCodeBtn.Click += CopyPuzzleCode_Click;
    btnPanel.Children.Add(copyPuzzleCodeBtn);

    return btnPanel;
}

/**
 * Event handler for copying the puzzle code to the clipboard.
```

```
 *
 * input: sender - The object that raised the event.
 *        e - The event arguments.
 * output: None
 */
private void CopyPuzzleCode_Click(object sender, RoutedEventArgs e)
{
    SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);

    Button btn = sender as Button;
    Clipboard.SetText(games[(int)btn.Tag].PuzzleCode);
    MessageBox.Show(GameConstants.COPIED_STR);
}

/**
 * Event handler for deleting a game.
 *
 * input: sender - The object that raised the event.
 *        e - The event arguments.
 * output: None
 */
private void DeleteGame_Click(object sender, RoutedEventArgs e)
{
    SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);

    Button btn = sender as Button;
    int index = (int)btn.Tag;

    MessageBoxResult result = MessageBox.Show("Are you sure you want to delete this game?", "Delete Game",
MessageBoxButton.YesNo, MessageBoxImage.Warning);

    if (result == MessageBoxResult.Yes)
    {
        DeleteGame(games[index]);
    }
}
```

```
/**
 * Deletes a game from the list and database.
 *
 * input: gameToRemove - GameInfo object containing the game information to remove.
 * output: None
 */
private void DeleteGame(GameInfo gameToRemove)
{
    DeleteGameFromDB(gameToRemove);

    int indexToRemove = games.IndexOf(gameToRemove);

    games.RemoveAt(indexToRemove);
    Border borderToRemove = Items[indexToRemove];

    Items.RemoveAt(indexToRemove);
    ItemsWrapPanel.Children.Remove(borderToRemove);

    // Update tags after deletion
    for (int i = indexToRemove; i < Items.Count; i++)
    {
        Border item = Items[i];
        item.Tag = i; // Update tag to match index in games list
        UpdateButtonTags(item, i); // Update button tags in the UI
    }
}

/**
 * Updates the tags of buttons inside a border element.
 *
 * input: border - Border element containing buttons to update.
 *        newTag - New tag value to assign to the buttons.
 * output: None
 */
private void UpdateButtonTags(Border border, int newTag)
{
    Grid grid = border.Child as Grid;
```

```
    foreach (UIElement element in grid.Children)
    {
        if (element is Button button)
        {
            button.Tag = newTag;
        }
    }
}

/**
 * Deletes a game from the database.
 *
 * input: gameInfo - GameInfo object containing the game information to delete.
 * output: None
 */
public static void DeleteGameFromDB(GameInfo gameInfo)
{
    string sqlStmt = DBConstants.DeletGameQuary;
    OleDbParameter parameter = new OleDbParameter("@Id", gameInfo.Id);
    DBHelper.ExecuteCommand(sqlStmt, parameter);
}

/**
 * Inserts a new game into the database.
 *
 * input: gameInfo - GameInfo object containing the game information to insert.
 * output: None
 */
public static void InsertGame(GameInfo gameInfo)
{
    string sqlStmt = DBConstants.InsertGameQuary;

    OleDbParameter[] parameters =
    {
        new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.Current, gameInfo.Current),
        new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.Solved, gameInfo.Solved),
        new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.Time, gameInfo.Time),
```

```
            new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.GameDate, gameInfo.Date),
            new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.BoardCode, gameInfo.BoardCode),
            new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.PuzzleCode, gameInfo.PuzzleCode),
            new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.GameName, gameInfo.Name),
            new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.HintsTaken, gameInfo.Hints),
            new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.ChecksTaken, gameInfo.Checks),
            new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.BoxHeight, gameInfo.BoxHeight),
            new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.BoxWidth, gameInfo.BoxWidth),
            new OleDbParameter(DBConstants.AT + DBConstants.Games_Parameters.DifficultyLevel, gameInfo.DifficultyLevel.ToString())
        };

        DBHelper.ExecuteCommand(sqlStmt, parameters);
    }

    /**
     * Adds a text block to a grid.
     *
     * input: grid - Grid to add the text block.
     *        text - Text content of the text block.
     *        fontSize - Font size of the text block.
     *        columnSpan - Number of columns the text block spans.
     *        title - Flag indicating if the text block is a title.
     * output: None
     */
    private void AddTextBlockToGrid(Grid grid, string text, double fontSize, int columnSpan = 1, bool title = false)
    {
        grid.RowDefinitions.Add(new RowDefinition());
        TextBlock textBlock = new TextBlock
        {
            Text = text,
            VerticalAlignment = VerticalAlignment.Center,
            HorizontalAlignment = title ? HorizontalAlignment.Center : HorizontalAlignment.Left,
            FontSize = title ? fontSize * 1.1 : fontSize,
            Margin = new Thickness(10, 0, 0, 0),
```

```
            FontWeight = title ? FontWeights.Bold : FontWeights.Normal,
        };

        textBlock.SetResourceReference(ForegroundProperty, ColorConstants.TextFore);

        Grid.SetRow(textBlock, grid.RowDefinitions.Count - 1);
        Grid.SetColumn(textBlock, 0);
        Grid.SetColumnSpan(textBlock, columnSpan);
        grid.Children.Add(textBlock);
    }
  }
}
```

עמוד שומר - XAML

```
<Page x:Class="Sudoku_WPF.Pages.SaverPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:Sudoku_WPF.Pages"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <!-- Make the title row height Auto -->
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <TextBlock x:Name="title_Txb" Grid.Row="0" Foreground="{DynamicResource Text}"
VerticalAlignment="Center" FontSize="24" FontWeight="Bold" Margin="20"
HorizontalAlignment="Center" Text="Saver Page Title"/>

        <ScrollViewer VerticalScrollBarVisibility="Auto"
HorizontalScrollBarVisibility="Disabled" Grid.Row="1">
            <WrapPanel x:Name="ItemsWrapPanel" HorizontalAlignment="Center"/>
        </ScrollViewer>
    </Grid>
</Page>
```

דף הצגת הוראות

מאפשר מעבר בין עמודי הוראות וקריאתם.

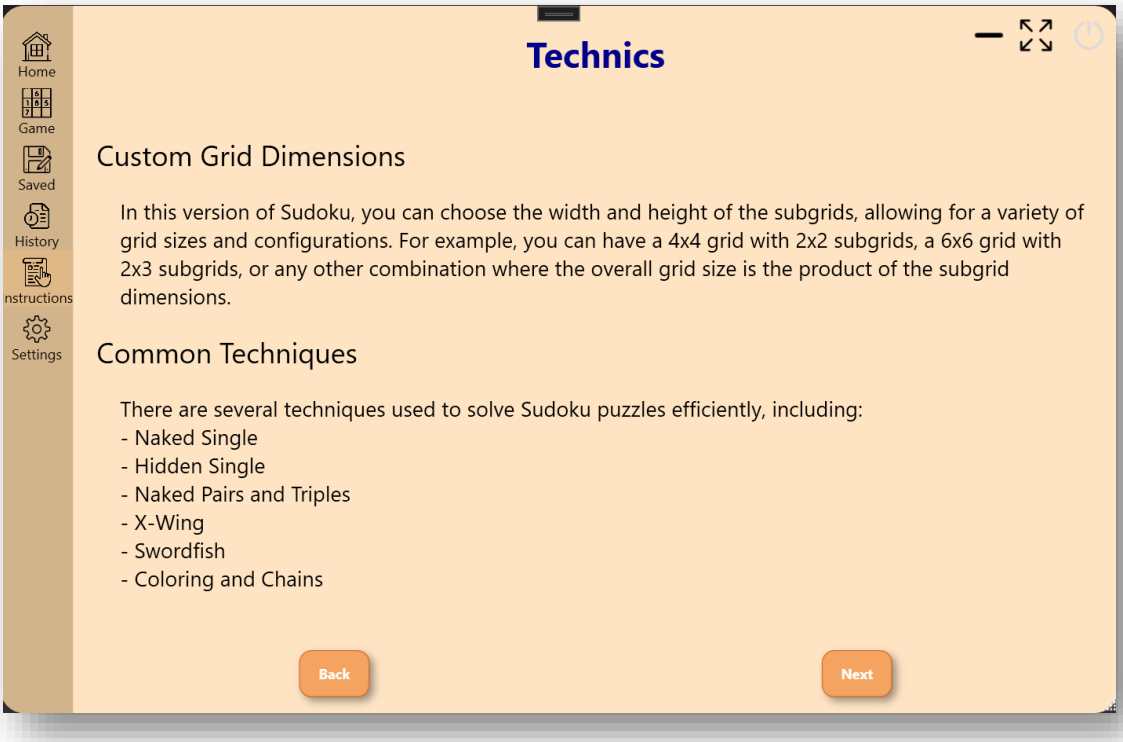
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Sudoku_WPF.Pages.InstructionPages;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using static Sudoku_WPF.publico.Constants;

namespace Sudoku_WPF
{
    /// <summary>
    /// Interaction logic for InstructionsPage.xaml
    /// </summary>
    public partial class InstructionsPage : Page
    {
        private int currentPageIndex = 0;
        private readonly Page[] instructionPages;

        /// <summary>
        /// Constructor for initializing the InstructionsPage.
        /// Initializes the instruction pages array and loads the first page.
        /// </summary>
        public InstructionsPage()
        {
            InitializeComponent();

            // Initialize the pages
            instructionPages = new Page[]
            {
                new InstructionPage1(),
                new InstructionPage2(),
                new InstructionPage3(),
                new InstructionPage4()

            };

            // Load the first page
            LoadCurrentPage();
        }

        /// <summary>
        /// Loads the current instruction page into the InstructionFrame.
        /// Updates the state of the navigation buttons.
        /// </summary>
        private void LoadCurrentPage()
        {
            InstructionFrame.Navigate(instructionPages[currentPageIndex]);
            UpdateButtonStates();
        }

        /// <summary>
```

```
        /// Updates the enabled state of the navigation buttons based on the current
page index.
        /// </summary>
        private void UpdateButtonStates()
        {
            BackButton.IsEnabled = currentPageIndex > 0;
            BackButton.Visibility = currentPageIndex > 0 ? Visibility.Visible :
Visibility.Collapsed;
            NextButton.IsEnabled = currentPageIndex < instructionPages.Length - 1;
            NextButton.Visibility = currentPageIndex < instructionPages.Length - 1 ?
Visibility.Visible : Visibility.Collapsed;
        }

        /// <summary>
        /// Event handler for the Back button click.
        /// Plays a button click sound and navigates to the previous instruction page if
available.
        /// </summary>
        private void BackButton_Click(object sender, RoutedEventArgs e)
        {
            SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);

            if (currentPageIndex > 0)
            {
                currentPageIndex--;
                LoadCurrentPage();
            }
        }

        /// <summary>
        /// Event handler for the Next button click.
        /// Plays a button click sound and navigates to the next instruction page if
available.
        /// </summary>
        private void NextButton_Click(object sender, RoutedEventArgs e)
        {
            SoundPlayer.PlaySound(SoundConstants.BOTTON_CLICK);

            if (currentPageIndex < instructionPages.Length - 1)
            {
                currentPageIndex++;
                LoadCurrentPage();
            }
        }
    }
}
```

עמוד הוראות - XAML

```
<Page x:Class="Sudoku_WPF.InstructionsPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="InstructionPage">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="8*" />
            <RowDefinition />
        </Grid.RowDefinitions>

        <Frame x:Name="InstructionFrame" NavigationUIVisibility="Hidden" Grid.Row="0" />

        <Grid Grid.Row="1" >
            <Grid.ColumnDefinitions>
                <ColumnDefinition />
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Button Style="{StaticResource RoundedButtonStyle}" x:Name="BackButton"
Height="33" Width="55" HorizontalAlignment="Center" VerticalAlignment="Center"
Content="Back" Click="BackButton_Click" Grid.Column="0"/>
            <Button Style="{StaticResource RoundedButtonStyle}" x:Name="NextButton"
Height="33" Width="55" HorizontalAlignment="Center"  VerticalAlignment="Center"
Content="Next" Click="NextButton_Click" Grid.Column="1"/>
        </Grid>
    </Grid>
</Page>
```

עמוד הוראות לדוגמה

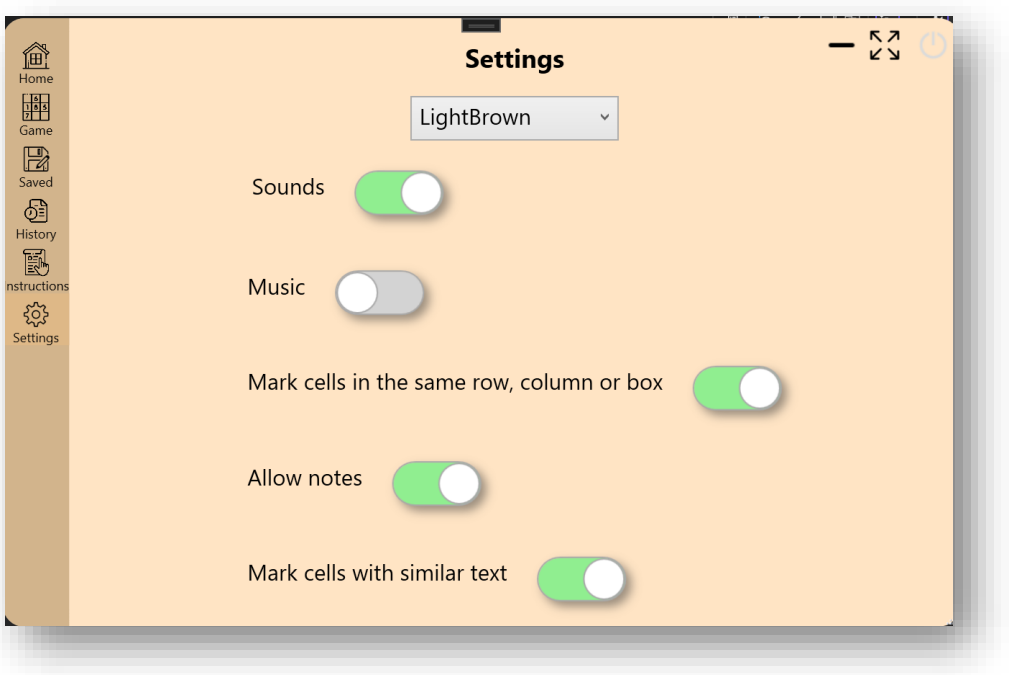


מכיל את תוכן ההערות בלבד. הכפתורים חלק מעמוד הצגת ההוראות

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Sudoku_WPF.Pages.InstructionPages
{
    /// <summary>
    /// Interaction logic for InstructionPage2.xaml
    /// </summary>
    public partial class InstructionPage2 : Page
    {
        public InstructionPage2()
        {
            InitializeComponent();
        }
    }
}
```

```xml
<Page x:Class="Sudoku_WPF.Pages.InstructionPages.InstructionPage2"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:Sudoku_WPF.Pages.InstructionPages"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="InstructionPage2">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!-- Header -->
        <TextBlock Grid.Row="0" Text="Technics" Style="{StaticResource
HeaderTextBlockStyle}"/>

        <!-- Content -->
        <ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">
            <StackPanel Margin="20">
                <!-- Custom Grid Dimensions -->
                <TextBlock Style="{StaticResource SubheaderTextBlockStyle}">Custom Grid
Dimensions</TextBlock>
                <TextBlock Style="{StaticResource ContentTextBlockStyle}"
TextWrapping="Wrap">
                    In this version of Sudoku, you can choose the width and height of
the subgrids, allowing for a variety of grid sizes and configurations. For example, you
can have a 4x4 grid with 2x2 subgrids, a 6x6 grid with 2x3 subgrids, or any other
combination where the overall grid size is the product of the subgrid dimensions.
                </TextBlock>

                <!-- Common Techniques -->
                <TextBlock Style="{StaticResource SubheaderTextBlockStyle}">Common
Techniques</TextBlock>
                <TextBlock Style="{StaticResource ContentTextBlockStyle}"
TextWrapping="Wrap">
                    There are several techniques used to solve Sudoku puzzles
efficiently, including:
                    <LineBreak/>
                    - Naked Single
                    <LineBreak/>
                    - Hidden Single
                    <LineBreak/>
                    - Naked Pairs and Triples
                    <LineBreak/>
                    - X-Wing
                    <LineBreak/>
                    - Swordfish
                    <LineBreak/>
                    - Coloring and Chains
                </TextBlock>
            </StackPanel>
        </ScrollViewer>
    </Grid>
</Page>
```

דף הגדרות תכנה



עמוד הגדרות המאפשר הגדרות תכנה ושינוי מצב הצבע בכל התכנה.

```
using static Sudoku_WPF.publico.Constants;
using Sudoku_WPF.publico;
using Sudoku_WPF.Themes;
using System;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;

namespace Sudoku_WPF
{
    /// <summary>
    /// Interaction logic for SettingsPage.xaml
    /// </summary>
    public partial class SettingsPage : Page
    {
        /// <summary>
        /// Constructor for initializing the SettingsPage.
        /// Sets up initial UI state based on current settings.
        /// </summary>
        public SettingsPage()
        {
            InitializeComponent();
            SetInitialTheme(Settings.theme.ToString());

            // Initialize toggle button states based on Settings
            sound.IsChecked = Settings.soundOn;
            music.IsChecked = Settings.musicOn;
            markRelated.IsChecked = Settings.markRelated;
            markSameText.IsChecked = Settings.markSameText;
            allowNotes.IsChecked = Settings.allowNotes;
        }

        /// <summary>
        /// Event handler for toggle button clicks.
        /// Updates corresponding setting values based on button state.
        /// </summary>
        /// <param name="sender">The toggle button that was clicked.</param>
        /// <param name="e">Event arguments.</param>
        private void ToggleButton_Click(object sender, RoutedEventArgs e)
        {
            SoundPlayer.PlaySound(SoundConstants.ON_OFF);

            ToggleButton toggleButton = sender as ToggleButton;
            if (toggleButton != null)
            {
                string buttonName = toggleButton.Tag as string;
                if (!string.IsNullOrEmpty(buttonName))
                {
                    // Handle the logic based on the button name
                    switch (buttonName)
                    {
                        case DBConstants.Settings_Parameters.SoundOn:
                            Settings.soundOn = toggleButton.IsChecked ?? false;
                            break;
                        case DBConstants.Settings_Parameters.MusicOn:
                            Settings.musicOn = toggleButton.IsChecked ?? false;
                            if (Settings.musicOn)
                            {
                                SoundPlayer.StartMusic(SoundConstants.BACK_MUSIC_NAME);
                            }
```

```
                            else
                            {
                                SoundPlayer.StopMusic();
                            }
                            break;
                        case DBConstants.Settings_Parameters.MarkRelated:
                            Settings.markRelated = toggleButton.IsChecked ?? false;
                            break;
                        case DBConstants.Settings_Parameters.AllowNotes:
                            Settings.allowNotes = toggleButton.IsChecked ?? false;
                            break;
                        case DBConstants.Settings_Parameters.SameText:
                            Settings.markSameText = toggleButton.IsChecked ?? false;
                            break;
                        default:
                            break;
                    }
                }
            }
        }

        /// <summary>
        /// Sets the initial theme selection in the ColorMode_CMBB ComboBox.
        /// </summary>
        /// <param name="themeName">The name of the theme to select initially.</param>
        private void SetInitialTheme(string themeName)
        {
            // Find the ComboBoxItem with the specified theme name and set it as
selected
            var selectedItem = ColorMode_CMBB.Items
                               .OfType<ComboBoxItem>()
                               .FirstOrDefault(item => item.Content.ToString() ==
themeName);

            if (selectedItem != null)
            {
                ColorMode_CMBB.SelectedItem = selectedItem;
            }
        }

        /// <summary>
        /// Event handler for the ColorMode_CMBB ComboBox selection change.
        /// Applies the selected theme and updates the Settings.
        /// </summary>
        /// <param name="sender">The ComboBox that triggered the event.</param>
        /// <param name="e">Event arguments.</param>
        private void ColorMode_CMBB_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            string option = "";
            ComboBoxItem cbi = (ComboBoxItem)ColorMode_CMBB.SelectedValue;
            option = cbi.Content.ToString();

            if (Enum.TryParse(option, out ColorThemes theme))
            {
                ThemeControl.SetColors(theme);
                Settings.theme = theme;
            }
        }
    }
}
```

```
<Page x:Class="Sudoku_WPF.Pages.SettingsPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="SettingsPage">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <!-- Title row -->
            <RowDefinition Height="*"/>
            <!-- Content row -->
        </Grid.RowDefinitions>

        <!-- Title TextBlock -->
        <TextBlock Text="Settings" FontSize="24" FontWeight="Bold" Margin="20"
HorizontalAlignment="Center" Foreground="{DynamicResource Text}" Grid.Row="0"/>

        <Viewbox Stretch="Uniform" Grid.Row="1" >
            <StackPanel Orientation="Vertical" HorizontalAlignment="Center"
Margin="50, 0,50,0">
                <!-- Combo Box for Color Mode Selection -->
                <ComboBox Name="ColorMode_CMBB"
SelectionChanged="ColorMode_CMBB_SelectionChanged" Height="30" Width="140"
FontSize="15">
                    <ComboBoxItem Content="Light" Foreground="Black"
Background="White"/>
                    <ComboBoxItem Content="Dark" Foreground="Black"
Background="White"/>
                    <ComboBoxItem Content="Red" Foreground="Red"
Background="White"/>
                    <ComboBoxItem Content="LightBrown" Foreground="SaddleBrown"
Background="White"/>
                    <ComboBoxItem Content="LightBlue" Foreground="LightBlue"
Background="White"/>
                    <ComboBoxItem Content="Pink" Foreground="Pink"
Background="White"/>
                </ComboBox>


                <!-- Sounds Toggle Button -->
                <StackPanel Orientation="Horizontal" Margin="20">
                    <TextBlock FontSize="15" Text="Sounds"
Foreground="{DynamicResource Text}" Margin="0,0,20,0"/>
                    <ToggleButton Tag="SoundOn" Style="{StaticResource
DAToggleButton}" x:Name="sound" Click="ToggleButton_Click" Content="Toggle Button"/>
                </StackPanel>

                <!-- Music Toggle Button -->
                <StackPanel Orientation="Horizontal" Margin="17">
                    <TextBlock Text="Music" FontSize="15"
Foreground="{DynamicResource Text}" Margin="0,0,20,0"/>
                    <ToggleButton x:Name="music" Tag="MusicOn"
Style="{StaticResource DAToggleButton}" Click="ToggleButton_Click"/>
                </StackPanel>

                <!-- Mark Related Toggle Button -->
                <StackPanel Orientation="Horizontal" Margin="17">
```

```
                    <TextBlock Text="Mark cells in the same row, column or box"
FontSize="15" Foreground="{DynamicResource Text}" Margin="0,0,20,0"/>
                    <ToggleButton x:Name="markRelated" Tag="MarkRelated"
Style="{StaticResource DAToggleButton}" Click="ToggleButton_Click"/>
                </StackPanel>

                <!-- Mark Related Toggle Button -->
                <StackPanel Orientation="Horizontal" Margin="17">
                    <TextBlock Text="Allow notes" FontSize="15"
Foreground="{DynamicResource Text}" Margin="0,0,20,0"/>
                    <ToggleButton x:Name="allowNotes" Tag="AllowNotes"
Style="{StaticResource DAToggleButton}" Click="ToggleButton_Click"/>
                </StackPanel>

                <!-- Mark Same Text Toggle Button -->
                <StackPanel Orientation="Horizontal" Margin="17">
                    <TextBlock Text="Mark cells with similar text" FontSize="15"
Foreground="{DynamicResource Text}" Margin="0,0,20,0"/>
                    <ToggleButton x:Name="markSameText" Tag="MarkSameText"
Style="{StaticResource DAToggleButton}" Click="ToggleButton_Click"/>
                </StackPanel>

            </StackPanel>
        </Viewbox>
    </Grid>
</Page>
```

אלגוריתמים ויכולות מיוחדות

יצירת לוח פתור

קריאה לפונקציה הרקורסיבית במקום הראשון בלוח

```
/// <summary>
/// Generates a solved Sudoku puzzle based on the current state of the board.
/// </summary>
1 reference
private void GenerateSolvedPuzzle()
{
    FillCells(0, 0);
}
```

סיום בסוף הלוח

```
if (row == GameSettings.BoardSide - 1 && col == GameSettings.BoardSide)
    return true;
```

ירידת שורה או מעבר לתא הבא במקרה הצורך

```
if (col == GameSettings.BoardSide)
{
    row++;
    col = 0;
}
if (solvedPuzzle[row, col] != '\0')
    return FillCells(row, col + 1);
```

בדיקת איזה מספר אפשר להכניס בתא כך שהלוח ישאר נכון, והאם בכלל אפשר

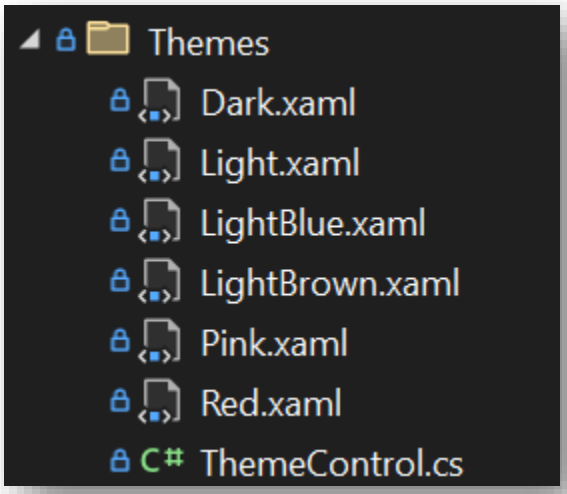
```
char[] nums = GenerateShuffledNumbers();

foreach (char num in nums)
{
    if (IsSafeToPlace(row, col, num))
    {
        solvedPuzzle[row, col] = num;
        if (FillCells(row, col + 1))
            return true;
        solvedPuzzle[row, col] = '\0';
    }
}
return false;
```

אם מחזיר false התא נשאר null וכאן בעצם הולכים אחורה חזרה בBack Tracking

Color themes

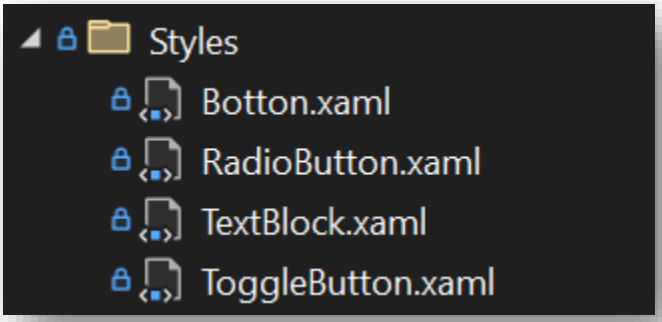
התכנה מחליפה את נושא הצבע שלה בצורה ריספונסיבית עזרת רפרנסים ל dynamic resources של הThemes

ה Themes הם resource dictionarys המכילים את שמות הצבעים כמפתחות ואת הצבעים ב RGB כערכים.



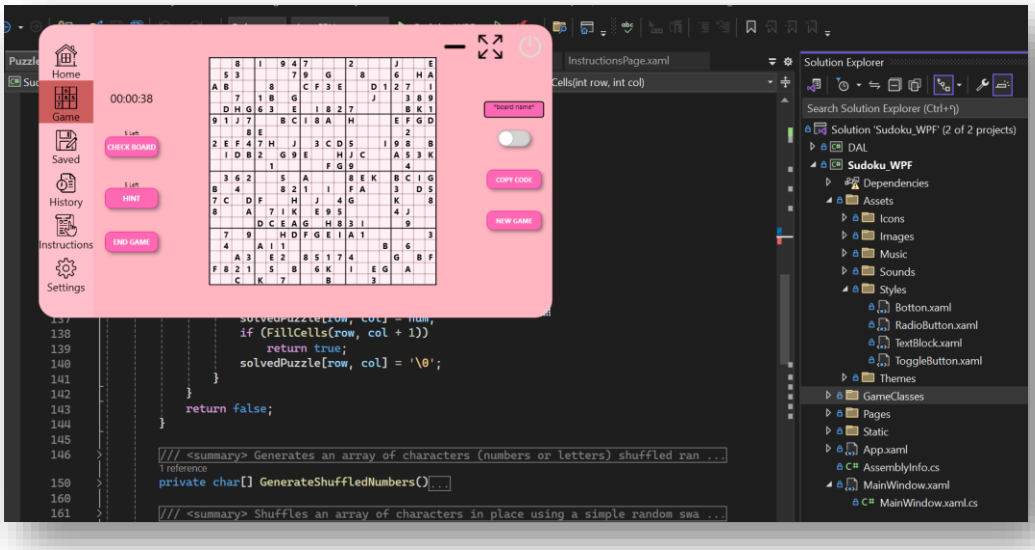
חזרה על קוד עיצובי

בתכנה אין הרבה חזרה על קוד עיצובי משום שלמדתי להשתמש ב styles לכפתורים מסודים שונים:



רספונסיביות ודינמיות מסך

חלון האפליקציה דינאמי כמעט חלוטין, הוא מאפשר שינוי גודל לכל הכיוונים עם עיווט מינימלי של התכנה. עשיתי זאת ללא ViewBox על כל החלון כי זה לא התאפשר בשל שקיפות החלון עצמו.



רכיבים חיצוניים

צירפתי חבילת Nuget בשם System.Data.OleDb.8.0.0

כדי לטפל בבאג שהיה לי עם הגישה לדאטה בייסים.

צירפתי גם חבילת Nuget בשם NumericUpDown

שמספקת אובייקת לבחירת ערך מספרי בצורה נוחה (ראה עמוד הגדרות משחק)

רפלקציה

בעיות במהלך הדרך

- היו לי אינספור בעיות ושגיאות במהלך הדרך ועל כולן הצלחתי להתגבר. המרכזיות שבהן היו:
- התקשיתי בחיבור הפרויקט DAL לפרויקט שלי – הפתרון היה התקנת Nuget Pakage.
- לקח לי זמן רב להשיג את האלגוריתם ליצירת הלוחות הפתורים.
- התקשיתי בהפיכת חלון האפליקציה לריספונסיבי ועלו בעיות תצוגה רבות עד שהגעתי לתוצאה הסופית.
- נושאי הצבעים לא עבדו כראוי במהלך משחק כי תחילה ניסיתי לקבוע אותם דרך משתנים בCS המחזיקים את ערכי הצבעים – הפתרון היה לקשור את תכונת האובייקת לresource reference עם השם המתאים כדי להבטיח שינוי צבע מיידי ורספונסיבי.

דברים שלמדתי

במהלך ביצוע פרויקט הסודוקו, למדתי המון נושאים חשובים שהעשירו את הידע שלי בתחום התכנות והפיתוח. למדתי כיצד ליישם רקורסיה כדי לפתור בעיות מתמטיות כמו הפיתרון של לוחות הסודוקו בצורה יעילה ומובנית. גם למדתי על ירושה וכיצד להשתמש בה במבנה הפרויקטים שלי, כולל הגדרת מחלקות ושימוש בהם באופן מקורי ויעיל. ניסיתי גם להבין וליישם מבני נתונים מורכבים יותר וטכניקות ניהול בסיסי נתונים, כמו שימוש ב-SQL לשאילתות לבסיסי נתונים של ACCESS ועדכונים.

בנוסף, למדתי על פיתוח תוכנה מונחה אירועים ב-WPF. כמו כן, ניסיתי לשפר את יכולות העיצוב והסגנון באמצעות יישום סטיילים ונגישות למשתמש באמצעות עיצוב והתאמת פרטים קטנים כגון שינוי גודל של חלונות.

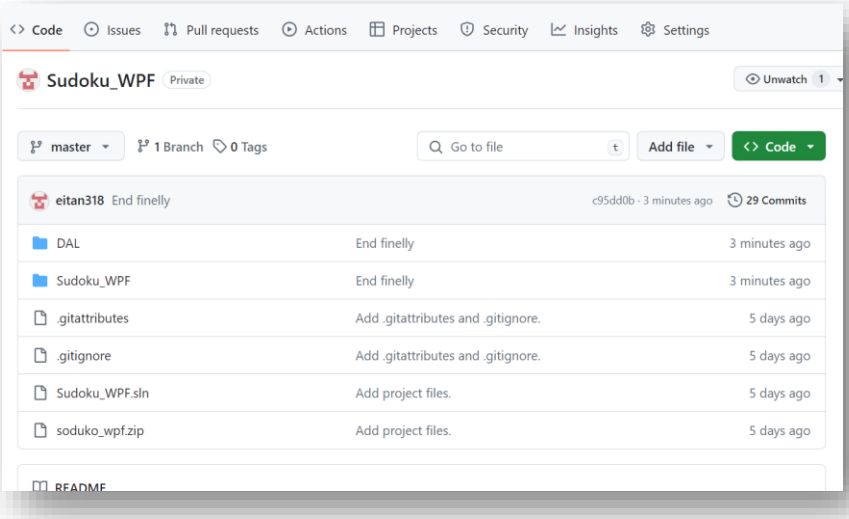
הפרויקט עזר לי להתפתח כמתכנת באופן משמעותי ולרכוש כישורים חשובים בעולם התכנות ואני בטוח שפרויקט זה הפך אותי למתכנת טוב יותר.

תהליך עבודה

בתחילת העבודה תכננתי בקפידה מה אני הולך לעשות כשהכנתי הצעת פרויקט. עבדתי על הפרויקט כמה ימים והיה לי לוח המתפקד כראוי. אז עשיתי הכל במחלקה אחת אבל כשהגיעה ל 800 שורות החלטתי שזה הולך להיות הפרויקטם הראשון שלי שאממש ב OOP. לקח קצת זמן, אך לבסוף המרתי את כולו למחלקות משחק קטנות.

מאז אני עבדתי על הפרויקט מסודר – אלגוריתם אחד בכל פעם והשתדלתי לשמור על עקרונות קוד טוב: חלוקה לפונקציות, minimal API, אבסטרקציה, KISS, אנקפסולציה ופולימורפיזם.

את ניהול הגרסאות שלי ביצעתי לראשונה ב github וזה עזר לי מאוד.



לקראת תהליך ההגשה עלו לי רעיונות חדשים בכל פעם, אך החלטתי שיותר מדיי זה לא טוב ושאני צריך לממש רק את הדברים המאתגרים והחשובים ביותר.

אני מאוד שמח מהתוצאה ואני חושב שהפרויקט תרם מאוד לידע ולניסיון שלי משום שהוא הפרויקט הגדול והמורכב ביותר שעשיתי עד כה.

תוספות אפשריות

- יצירת גרסה ל/mobile
- טורנירים בין שחקנים
- חידה יומית
- רשת חברתית המאפשרת יצירת חברות סודוקו ותחרויות מאורגנות