

Projektuppgift

DT071 C#

Projektrapport

Tony Johansson



Mittuniversitetet

MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. Campus Sundsvall Holmgatan 10, SE-851 70 Sundsvall.

Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.

Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

MITTUNIVERSITETET
Avdelningen för informationssystem och -teknologi

Författare: Tony Johansson, tojo8500@student.miun.se
Utbildningsprogram: Webbutveckling, 120 hp
Huvudområde: Datateknik
Termin, år: 02, 2021

Sammanfattning

Rapporten behandlar mitt projekt KnightTour som jag har valt att göra samt vilka funktioner som projektet ska implementera.

Jag beskriver vad KnightTour är för något lite kortfattat samt lite grundläggande teori om den eller de tekniker som jag har valt att göra projektet i. Tar även upp de verktyg och metoder som jag har valt att använda i projektet. Jag tar även upp och beskriver lite om hur koden fungerar och är uppbyggt kring GUI och algoritmen. Jag beskriver algoritmen med bilder för att ge läsaren en tydlig och lättförståelig uppfattning om hur KnightTour är och hur man implementerar algoritmen. Då projektet använder en databas beskriver jag hur man har skapat denna samt några centrala metoder som projektet använder. Vidare tar jag upp några viktiga saker om connectionstring och app.config. Slutligen tar jag upp lite om hur själva DatabasAPI fungerar när man ska skriva respektive läsa från en databas. Jag gör en summering kring resultatet av projektet samt lägger till lite egna synpunkter på projektet.

Innehållsförteckning

Sammanfattning.....	iii
Terminologi.....	v
1 Introduktion / Inledning.....	1
1.1 Bakgrund och problemmotivering.....	1
1.2 Detaljerad problembeskrivning.....	1
2 Teori / Bakgrundsmaterial.....	3
2.1 Win forms.....	3
2.2 Knight Tour.....	3
2.3 Databas.....	3
3 Metod.....	4
3.1 Verktyg och metoder.....	4
4 Konstruktion / Lösningalternativ.....	5
4.1 Översikt.....	5
4.1.1 GUI.....	5
4.1.2 Validering och val av position.....	7
4.1.3 Algoritmer.....	7
4.1.4 Databas och DatabasAPI.....	9
4.1.5 Klasser.....	12
5 Resultat.....	14
6 Slutsatser / Analys / Diskussion.....	15
Källförteckning.....	16
7 Bilagor.....	17
7.1 Databasdiagram.....	17
7.2 Flödesschema.....	18

Terminologi

En eventuell förteckning över termer, förkortningar och variabelnamn med korta förklaringar placeras efter innehållsförteckningen. Observera att man måste förklara begrepp och förkortningar första gången de används i den löpande texten, även om rapporten har ett terminologiavsnitt.

Akronymer/Förkortningar

VS	Visual studio
----	---------------

1 Introduktion / Inledning

1.1 Bakgrund och problemmotivering

I denna studie undersöker jag hur man löser springarvandringen rent programmeringssmässigt.

Jag fastnade för att implementera en springarvandring i en desktop win form applikation genom att använda programmeringsspråket C#. Springarvandring går ofta under namnet Knight Tour. Jag har hört talas om springarvandring tidigare och det är när man ska göra 64 drag med en springare på ett schackbräde men den får aldrig gå på samma ruta två gånger.

Detta känns som en utmanande och spännande uppgift att söka rätt på rätt algoritm som stöder en springarvandring. Jag vill även kunna spara alla körda springarvandringar och sedan titta på sekvensen av drag vid ett senare tillfälle. För att få den användbar ska det även vara möjligt att kunna ange en godtycklig koordinat genom att ange rad och kolumn samt namnet på den som kör springarvandringen. Jag kommer att ha en databas kopplad där jag kan spara personens namn, sekvensen av drag samt datum och tid då springarvandringen kördes.

1.2 Detaljerad problembeskrivning

Följande uppgifter ska denna springarvandring stödja.

1. Kunna ange namn på den personen som kör springarvandringen
2. Kunna ange koordinater för start position för springaren. Om ingen position anges gäller den default position som är angiven.
3. Tydligt visa sekvensen av alla 64 dragen som man gör med springaren på ett schackbräda genom att ange ett löpnummer på varje ruta på schackbrädet.
4. Rita upp ett snyggt GUI schackbräda med angivna kolumner [1-h] och rader [1-8] så det är lätt att ange koordinater.
5. Man ska kunna rensa schackbrädet så det blir tomt
6. Alla dragen som ingår i en springarvandring ska sparas i en databas tillsammans med datum, tid samt namnet som angavs.
7. Alla körda springarvandringar ska visas i en kontroll med datum, tid och namnet på den som körde springarvandringen där man kan välja en av dessa.

8. Vid val av någon av dessa ska springarvandringens alla 64 drag visas med [rad,kolumn]
9. Det ska finnas en tabell Person med id(pk), namn och Created och en annan tabell KnightTour med id(pk), Row, Col och PersonID(fk).
Vi har en relation en till många mellan Person och KnightTour
10. Connection string ska ligga i app.config
11. Säkerställer koden så man undviker SQL Injection.

2 Teori / Bakgrundsmaterial

2.1 Win forms

Detta är en teknik för att skapa windows program. Man använder kontroller som tex. knappar, label och textboxar och placerar dessa på skärmen där man vill ha dem.

Man skapar alltså ett grafiskt gränssnitt som man kan interagera med genom att använda händelsehanterare som t.ex. när man klickar på en knapp ska något hända. Klassen som win form bygger på heter Form och denna har två olika visningslägen dels View Designer och dels View Code.

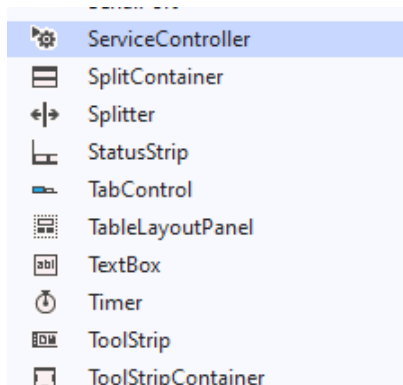


Fig 1 Toolbox

I läge View Designer visas de kontroller som finns och View Code visas den kod som ligger bakom denna form applikation. Här i Fig 1 Toolbox kan man se en utdrag av några kontroller. Här man kan t.ex. se TextBox kontrollen.[2]

2.2 Knight Tour

Du placerar en springare på en godtycklig ruta på ett schackbräda och ska sedan hoppa runt på alla 64 rutorna(8x8) men får aldrig hamna på samma ruta två gånger.[1]

2.3 Databas

Jag kommer att använda en relationsdatabas där man har relationer mellan olika tabeller. I en tabell sparas data på ett strukturerat sätt med rader och kolumner. Varje tabell måste ha en kolumn som är primärnyckel och där varje värde är unikt. Om man har en relation en till många kommer det att finnas en främmande nyckel på många sidan som pekar på primärnyckeln på parent sidan. Här ett litet exempel på en sekvens med insert

3 Metod

3.1 Verktyg och metoder

Jag kommer här att beskriva vilka verktyg och metoder som jag har använd för att utföra en springarvandring.

1. Visual Studio 2019 som IDE
2. Windows forms
3. SQL Server som databas hanterare
4. app.config för att lagra databasens connectionstring

4 Konstruktion / Lösningalternativ

4.1 Översikt

För att lösa springarvandringen har jag delat upp konstruktion i följande delar.

1. GUI. Här kommer jag beskriva lite om statiska kontroller och dynamiska kontroller.
2. Validering och val
3. Algoritmer för att lösa springavandringen
4. DatabasAPI
5. Klasser som jag använder

4.1.1 GUI

För att skapa den interaktiva delen Fig 2 Kontroll Panel har jag helt enkelt använt mig av färdiga kontroller som TextBox, Button, Label osv. Dessa kan man bara dra in från ToolBox. Dessa kontroller är okomplicerade och enkla att jobba med.

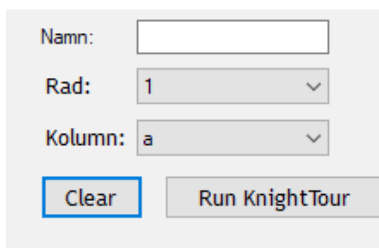


Fig 2 Kontroll panel

Sedan har vi schackbrädet Fig 3 som ser ut så här.

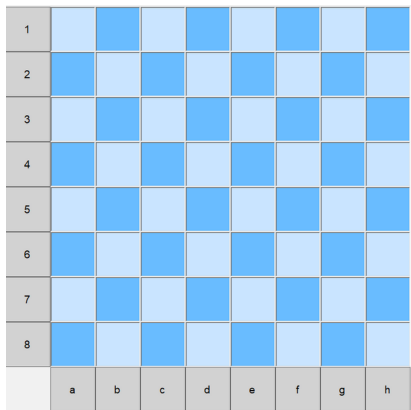


Fig 3 Schackbrädet

Det finns ju ingen färdig schackbräda kontroll som t.ex. TextBox som man bara kan ta och dra in från Toolbox. Detta schackbräda bygger man upp dynamiskt genom att använda andra kontroller. Dessa kontroller fungerar som typ en byggsats som man kan använda för att bygga det färdiga schackbrädet. Har t.ex. använd två hjälpmetoder AddRowToH för att skapa raden a-h längs ned och AddColTo8 för att skapa kolumnen till vänster där det står 1-8. Dessa två påminner lite om varandra. Jag visar en av dessa eftersom den är kort. Dessa hjälpmetoder anropas för varje varv i loopen.

```
private void AddRowToH(int n)
{
    Label lbl2 = new Label();
    lbl2.AutoSize = false;
    lbl2.TextAlign = ContentAlignment.MiddleCenter;
    lbl2.Text = aToh[n];
    lbl2.Location = new Point(80 * n + 200, 640 + 180);
    lbl2.Font = new Font("Arial", 14, FontStyle.Regular);
    lbl2.Size = new Size(tileSize, tileSize);
    lbl2.BackColor = System.Drawing.ColorTranslator.FromHtml("#cfcfcf");
    lbl2.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
    Controls.Add(lbl2);
}
```

Fig 4 Metoden AddRowToH

En central del i metoden CreateChessBoard är denna

```
// Create new Panel control which will be one cell in the chessboard
var newPanel = new Panel
{
    Size = new Size(tileSize, tileSize),
    Location = new Point(tileSize * n + 200, tileSize * m + 100)
};
```

Fig 5 Ny panel instans

Denna newpanel som visas i Fig 5 skapas i varje varv och läggs sedan till i kontroll collection på detta sättet Controls.Add(newPanel)

4.1.2 Validering och val av position

För att få numeriska data för a-h drar jag bort 96. Om jag t.ex. har valt default position [1,a] kommer programmet att jobba med a som kolumn 1.

```
int col = cboCol.Text[0]-96; // ger 1 om vi har a
```

Om namn är tomt ges ett felmeddelande om att namn måste anges. Validerar genom metoden `string.IsNullOrEmpty(value);`

4.1.3 Algoritmer

För att lösa springarvandring måste man använda någon form av algoritm. Det finns flera att välja på men jag har valt den som kallas Warnsdorff rule [1].

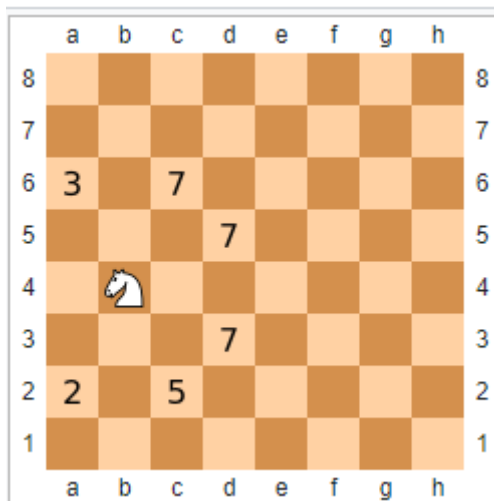


Fig 6 Warnsdorff rule

Här en grafisk beskrivning av Warnsdorffs regel. I schackrutorna i Fig 6 står det ett heltal och det betyder att från denna plats kan man göra så många drag med en springare. Det står 3,7,5 och 2. Warnsdorffs regel säger följande att man ska flytta springaren till det ställe där det finns minst antal fria ställen att flytta till. Så i detta fallet kommer nästa drag att flytta springaren till position [2,a] eftersom från denna position finns bara 2 ställen att flytta till.

I min kod är metoden `ProcessKnightTour` själva motorn i min applikation.

```
// We go clockwise starting at top
CheckCountMin(GoToAllEightPositions(rad - 2, col - 1), rad - 2, col - 1); //Up 2, Left 1
CheckCountMin(GoToAllEightPositions(rad - 2, col + 1), rad - 2, col + 1); //Up 2, Right 1
CheckCountMin(GoToAllEightPositions(rad - 1, col + 2), rad - 1, col + 2); //Up 1, Right 2
CheckCountMin(GoToAllEightPositions(rad + 1, col + 2), rad + 1, col + 2); //Down 1, Right 2
CheckCountMin(GoToAllEightPositions(rad + 2, col + 1), rad + 2, col + 1); //Down 2, Right 1
CheckCountMin(GoToAllEightPositions(rad + 2, col - 1), rad + 2, col - 1); //Down 2, Left 1
CheckCountMin(GoToAllEightPositions(rad + 1, col - 2), rad + 1, col - 2); //Down 1, Left 1
CheckCountMin(GoToAllEightPositions(rad - 1, col - 2), rad - 1, col - 2); //Up 1, Left 1
```

Fig 7 Kod från `ProcessKnightTour`

I Fig 7 kan man se att jag jag anropar dessa rader 8 gånger och det representerar 8 st drag med springaren. En springare kan alltid göra 8 st drag men ibland hamnar ett drag utanför schackbrädet som man måste kontrollera. Sedan i metoden `GoToAllEightPositions` testar jag alla 8 dragen från den position som anges med rad och kolumn. Om man jämför med Fig 6 ovan så i `GoToAllEightPositions` går man först till [a,6] och beräknar hur många platser kan man gå till (3 st lediga) sedan provar man med position [c,6] sedan med position [d,5] osv. Som man kan se i Fig 7 finns det med en metod som heter `CheckCountMin` denna metoden sparar undan det minsta värdet så till denna metod kommer det först en 3 sedan en 7 och sedan 5 och slutligen en 2 som blir det slutliga positionen som springaren ska gå till.

Metoden `PositionHorseOnBoard` används för att uppdatera GUI och den ser ut så här.

```
private void PositionHorseOnBoard(int row, int col)
{
    Label lbl = new Label();
    lbl.Font = new Font("Verdana", 16, FontStyle.Regular);
    lbl.Text = (++sekvensCounter).ToString();
    chessBoardPanels[col, row].Controls.Add(lbl);
}
```

Fig 8 `PositionHorseOnBoard`

Som man kan se i Fig 8 `PositionHorseOnBoard` skickar jag in rad och kolumn och uppdaterar sedan GUI med en siffra + lägger på lite större text (16px). Sedan kan man också se sekvensräknaren som skriver ned vilket sekvensnummer som man hade. För att validera om jag håller mig inom schackbrädets ramar använder jag metoden `ValidateRC(int rad, int col)`

```
{
    return rad >= MIN && rad <= MAX && col >= MIN && col <= MAX;
}
```

För att tömma schackbrädet använder jag metoden `ClearChessBoard` som visas i Fig 9

```
private void ClearChessBoard()
{
    for (var n = 1; n < gridSize; n++) //Columns
    {
        for (var m = 1; m < gridSize; m++) //Rows
        {
            chessBoardPanels[n, m].Controls.Clear();
        }
    }
}
```

Fig 9 `ClearChessBoard`

4.1.4 Databas och DatabasAPI

För att kunna spara data på ett smidigt och bra sätt använder jag en databas. Jag skapar en ny databas genom Add =>NewItem => Service-based Database och döpte denna till Database. Extension för denna typ av database blir mdf. Jag skapar två tabeller KnightTour som innehåller själva dragen och Person som kör själva springarvandringen samt de kolumner med typer som behövs. Dessa två har relationen 1 till många dvs en person gör många drag. Connection string lägger jag i app.config och ger den namnet KnightTour som man kan se i Fig 10 App.config

```
<connectionStrings>
  <add name="KnightTour"
        connectionString="Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Programme
</connectionStrings>
```

Fig 10 App.config

Att lägga connectionstring i själva koden är en dålig lösning men genom att lägga den i app.config kan man enkelt ändra på sökvägen om man vill kunna ansluta till den utan att från ett annan path utan att behöva kompilera om koden. För att kunna få en tydlig och bra separation mellan databasen och applikationen har jag valt att placera alla databasaccessmetoder i en egen klass som jag kallar för DatabasAPI. Jag använder databasen dels för att skriva till den men också för att läsa data från den.

1. Skriva till den sker genom metoden SaveKnightTour och denna visas i Fig 11 SaveKnightTour

```
private void Saveknighttour()
{
    DatabaseAPI dbapi = new DatabaseAPI();
    dbapi.InsertIntoDb(txtNamn.Text, chessMoves);
}
```

Fig 11 SaveKnightTour

Denna metod anropar metoden InsertIntoDb i DatabasAPIet och skickar med namnet på personen som körde springarvandringen samt alla 64 dragen som argument.

2. Har två metoder som används för att läsa från databasen. Den första metoden som kallas GetKnightTourPerson hämtar alla data som behövs för att ladda comboBox som visas i Fig 12

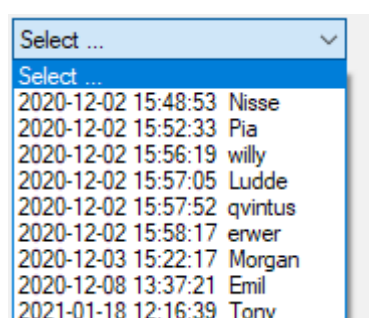


Fig 12 Combobox

```
private void GetKnightTourPerson()
{
    DatabaseAPI dbapi = new DatabaseAPI();
    DataTable dt = dbapi.GetKnightTourPerson();
    cboPersons.Items.Clear();

    if (dt.Rows.Count > 0)
    {
        cboPersons.Items.Add("Select ...");
        for (int i = 0; i < dt.Rows.Count; i++)
        {
            cboPersons.Items.Add(new ComboBoxItem(dt.Rows[i]["id"].ToString(), dt.Rows[i]["
        }
        cboPersons.SelectedIndex = 0;
    }
}
```

Fig 12 GetKnightTourPerson

Som man kan se i Fig 12 GetKnightTourPerson loopar man runt och instansierar en ny comboBoxItem för varje rad och bifogar de data som man fick från databasen. Man kan också se att jag rensar combo boxen innan jag lägger till nya. Som man kan se i fig 12 använder jag ToString för att skriva ned en representation av klassen ComboBoxItem.

Min andras metod heter GetKnightToursMoves se Fig 13. Denna hämtar alla drag som man har gjort. Denna metod är snarlik Fig 12 fast här använder jag kontrollen ListBox i stället. Argumentet id är personens id som körde springarvandringen som man vill ha kör sekvensen för.

```
private void GetKnightToursMoves(int id)
{
    DatabaseAPI dbapi = new DatabaseAPI();
    DataTable dt = dbapi.GetKnightToursMoves(id);
    lstMoves.Items.Clear();
    for (int i = 0; i < dt.Rows.Count; i++)
    {
        lstMoves.Items.Add(i+1 + " " + dt.Rows[i]["Row"].ToString() + ", "+
    }
}
```

Fig 13 GetKnightToursMoves

När man klickar på en rad i Combo boxen körs ebent hanteraren som visas i Fig 14

```
private void cboPersons_SelectedIndexChanged(object sender, EventArgs e)
{
    //Clear the array that keep all the moves
    chessMoves.Clear();
    if (!cboPersons.SelectedItem.ToString().StartsWith("Select"))
    {
        //Get the id for selected person you want to see all the moves
        int idValue = int.Parse(((ComboBoxItem)cboPersons.SelectedItem).HiddenValue);
        GetKnightToursMoves(idValue);
    }
}
```

Fig 14 cboPersons_SelectedItemChanged

Som man kan se här anropar jag ett HiddenValue property för att få tag på vilket id som denna person har som jag sedan använder när jag anropar metoden GetKnightToursMoves som jag visade förut.

Slutligen ska jag nämna lite om själva DatabasAPIet.

I konstruktorn för DatabasAPI hämtar jag connectionstringen från app.config så här.

```
connectionString = ConfigurationManager.ConnectionStrings["Knight-Tour"].ConnectionString;
```

Som man kan se här anger jag namnet på connection string som är KnightTour

```
using (var connection = new SqlConnection(connectionString))
{
    connection.Open();
    string query = "INSERT into Person(name, created) VALUES (@name, @cre";
    SqlCommand cmd = new SqlCommand(query, connection);
    cmd.Parameters.AddWithValue("@Name", name);
    cmd.Parameters.AddWithValue("@created", DateTime.Now);
    int insertedID = Convert.ToInt32(cmd.ExecuteScalar());
}
```

Fig 15 InserIntoDB

Principen för att hämta och skriva till databasen är snarlika. Som man kan se i Fig 15 används några olika metoder.

Lite förklaring

1. Man börjar med att Instansiera en SqlConnection
2. Öppna connection till db
3. Skapa sql frågan
4. Instansiera SqlCommand med query och connection
5. Stoppa in parametrar(Detta sätt säkerställer att man undviker Sql Injection)
6. Exekvera sql frågan och få tillbaka aktuellt id för denna person.

Här metoden GetKnightTourPerson Fig 16 som hämtar data från databasen

```
public DataTable GetKnightTourPerson()
{
    try
    {
        using (var connection = new SqlConnection(connectionStri
        {
            connection.Open();
            string query = "select * from Person";
            SqlCommand cmd = new SqlCommand(query, connection);
            SqlDataAdapter adapter = new SqlDataAdapter(query,
            DataTable persons = new DataTable();
            adapter.Fill(persons);
            return persons;
        }
    }
    catch (SqlException e )
    {
    }
```

Fig 16 GetKnightTourPerson

Denna metod är en smula annorlunda.

1. Man börjar med att Instansiera en SqlConnection
2. Öppna connection till db
3. Skapa sql frågan
4. Instansiera SqlCommand med query och connection
5. Instansiera en SqlDataAdapter
6. Instansiera en DataTable
7. Kör frågan som laddar DataTable med data och denna returneras
8. Har även med ett catch block ifall något går snett.

4.1.5 Klasser

Har är de klasser som jag har använd

1. ChessMove. Detta är en liten klass och representerar ett drag på ett schackbräda.

2. ComboBoxItem

Denna klass används för att representera varje rad i Fig 17 combobox

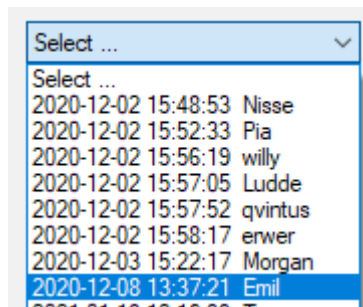


Fig 17 ComboBox

3. DatabasAPI klassen representerar all accessmetoder som man använder mot databasens
4. Form1 klassen är windows form som driver hela applikationen.
5. Program klassen. Används för att starta applikationen och anropas från runtime

5 Resultat

Jag har löst allt som finns specificerat under detaljerad problembeskrivning. Jag har använd en hel del bilder för att tydligt kunna visa exempel för att förstärka förståelsen. Jag kan notera att springarvandringen fungerar och att varje ruta på schackbrädet har en siffra från 1 till 64. Jag har gjort relevanta kommentarer i koden så att alla ska förstå denna. Där så är möjligt har jag kedjat uttryck i stället för att använda en massa extra variabler.

```
int idValue = int.Parse(((ComboBoxItem)cboPersons.SelectedItem).HiddenValue);  
if (chessBoardPanels[col + 1, rad + 2].Controls.Count == 0)
```

Allt access mot databasen fungerar som det ska både att skriva till databas tabellen Person och tabellen KnightTour och att läsa från dessa tabeller. Att läsa connection string från app.config fungerar också utan problem. För att fånga eventuella fel på ett bra sätt som kan förekomma när man jobbat mot en databas har jag använt try...catch. För att åstadkomma implicit close av SqlConnection string har använt using statement.

6 Slutsatser / Analys / Diskussion

KnightTour var en intressant och givande uppgift där jag bl.a. har förstärkt mina kunskaper i C#. När jag skriver kod strävar jag efter att skriva kort kod som ändå ska den vara läsbar. Ett exempel på detta är när jag använder ternary operatoren som är smidig och bra att använda men ändå lätt att förstå. Jag är helnöjd men min uppgift och tycker att GUI blev riktigt snyggt med nyanser av blått för schackbrädet och allt fungerar klockrent.

Det som är särskilt intressant är att algoritmen som jag använder är riktigt trivial och lätt att implementera i koden. Jag tycker att denna uppgift skulle passa bra att som en laboration i t.ex. js.

Springarvandringen är ett välkänt problem och har anor från 800-talet e.Kr. Det som också är intressant är att springarvandringen kan lösas med några olika algoritmer där warnsdorffs regel är den enklaste och lättast att förstå och den som jag har använt. Om jag skulle göra samma sak igen skulle jag göra på exakt samma sätt. Det kanske skulle gå att förkorta koden på några ställen med då förmodligen på bekostad av tydligheten och läsbarheten.

Jag vill gärna ha en kommentars block före varje metod som kort beskriver metoden samt eventuella argument som används. Jag tycker jag har strukturerat upp koden och använt klasser på ett bra sätt som t.ex..DatabasAPI där jag har kapslar in accesskod som tillhör Databasen. En annan som jag tänker speciellt på är connectionstring som alltid ska ligga i app.config och inte inuti koden och skvalpa. Jag har använt rikligt med bilder så att läsaren ska få en bra förståelse för min lösning.

Källförteckning

Här följer exempel på hur en källförteckning kan utformas enligt Vancouver-systemet. Den är automatiserad enligt metoden numrerad lista och korsreferenser, som beskrivs i kapitel Fel: Det gick inte att hitta referenskällan.

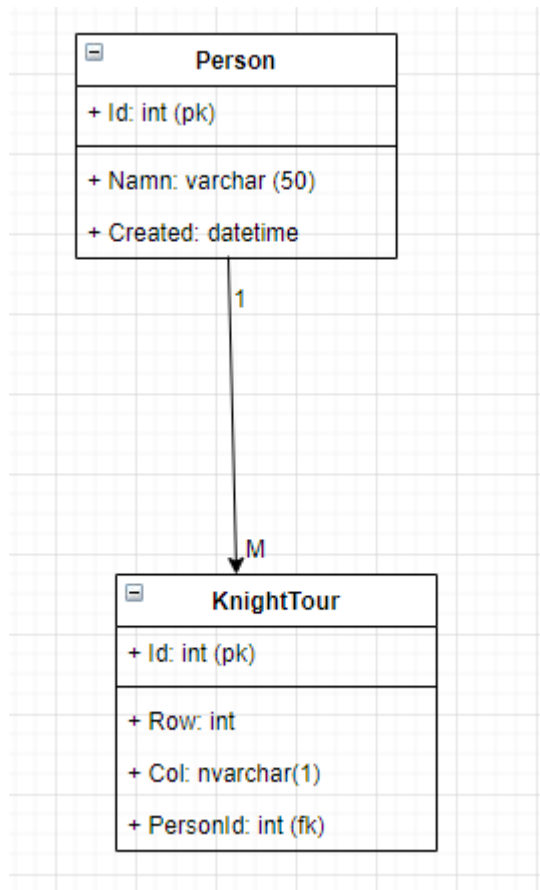
- [1] https://en.wikipedia.org/wiki/Knight's_tour
Knight's tour
Publiserad 2020-12-24

- [2] <https://csharpskolan.se/article/windows-forms-del-1/>
Windows Form
Publiserad Hämtad 2021-01-18

- [3] <https://www.c-sharpcorner.com/UploadFile/5d065a/tutorial-1sql-server-database-connection-in-window-form/>
SQL SERVER Database connection in Windowss FORMS
Publiserad Hämtad 2021-01-18

7 Bilagor

7.1 Databasdiagram



7.2 Flödesschema

