Міністерство освіти і науки України

Запорізька державна інженерна академія

Факультет автоматизованих систем управління виробництвом

Кафедра програмного забезпечення автоматизованих систем

**КУРСОВА РОБОТА**

**з курсу Об’єктно-орієнтоване програмування**

тема: Програмне забезпечення табло на станції метро

Спеціальність 6.050103 „Програмна інженерія”

Виконав студент

групи сп-14-1д

Барабаш Андрій

Керівник роботи

Доцент Попівщий В.І.

Оцінка

Члени комісії: доц. Попівщий В.І.

доц. Безверхий А.І.

ст. викл. Кривуляк В.В.

Запоріжжя

2015

**Зміст**

[Розділ 1.Вступ 3](#_Toc438659856)

[Розділ 1.1 Глосарій 4](#_Toc438659857)

[Розділ 1.2 Опис предметної області](#_Toc438659858) 5

[Розділ 1.3 Неформальна постановка задачі](#_Toc438659859) 6

[Розділ 1.4 План робіт](#_Toc438659861) 7

[Розділ 2. Вимоги до оточення](#_Toc438659862) 8

[Розділ 2.1 Вимоги до апаратного забезпечення](#_Toc438659863) 8

[Розділ 2.2 Вимоги до програмного забезпечення](#_Toc438659864) 8

[Розділ 2.3 Вимоги до користувача](#_Toc438659866) 8

[Розділ 3. Архітектура системи](#_Toc438659867) 9

[Розділ 3.1. Діаграма класів](#_Toc438659868) 10

[Розділ 3.2. Основні атрибути і методи. Їх короткий опис](#_Toc438659869) 11

[Розділ 3.3. Відношення між класами](#_Toc438659881) 14

[Розділ 3.4.Варіанти користування](#_Toc438659882) 15

[Розділ 4. Проект програмної системі](#_Toc438659883) 16

[Розділ 4.1. Засобі реалізації](#_Toc438659884) 16

[Розділ «4.2 Проект інтерфейсу»](#_Toc438659885) 16

[Розділ 4.3. Реалізація та тестування](#_Toc438659886) 18

[Висновки](#_Toc438659887) 19

[Література](#_Toc438659888) 20

[Додаток](#_Toc438659889) 21

[Додаток 1](#_Toc438659890) 21

# Розділ 1.Вступ

Завдання даної курсової роботи - розробити програмну модель, в якості предметної області якої виступала розробка програмного забезпечення роботи табло станції метро.

Реалізацію необхідно було виконати, використовуючи об'єктно-орієнтоване програмування, засвоївши і закріпивши цю парадигму і складові принципи.

Об'єктно-орієнтоване програмування (ООП) — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. В ній використано декілька технологій від попередніх парадигм, зокрема успадкування, модульність, поліморфізм та інкапсуляцію.

Оскільки C# - це об'єктно-орієнтована мова програмування, то вона володіє трьома найбільш загальними властивостями ООП:

• Інкапсуляція - це механізм програмування, що об'єднує разом код

і дані, якими він маніпулює, виключаючи як втручання ззовні, так і неправильне використання даних. Основною одиницею інкапсуляції в С # є клас, який визначає форму об'єкта.

• Поліморфізм - ця властивість, дозволяє одному інтерфейсу отримати доступ до загального класу дій. Вибрати конкретну дію (метод) в кожному окремому випадку - це завдання компілятора.

• Наслідування – це процес, в ході якого один об'єкт набуває властивостей іншого об'єкта. Завдяки механізму

спадкування один об'єкт стає окремим екземпляром більш загального класу.

Оскільки даний застосунок реалізовано на С#, то йому притаманні властивості ООП.

## Розділ 1.1 Глосарій

Клас – визначає абстрактні характеристики деякої сутності, включаючи характеристики самої сутності (її атрибути або властивості) та дії, які вона здатна виконувати.

Об’єкт – окремий екземпляр класу.

Метод – можливості об’єкта.

Інкапсуляція - властивість системи, що дозволяє об'єднати дані і методи, які працюють з ними, в класі і приховати деталі реалізації від користувача.

Поліморфізм - властивість, яка дозволяє одне і теж ім'я використовувати для розв’язання двох або більше схожих, але технічно різних задач.

Наслідування - клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватися цілі дерева успадкування. Підкласи успадковують атрибути та поведінку своїх батьківських класів, і можуть вводити свої власні.

Абстрагування - спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми.

Предметна область – область, на основі якої створюється програмна модель.

Таймер - засіб, що відраховує інтервал(и) часу.

Метро (Subway) - вид міського громадського транспорту, фактично варіант залізниці. Лінії метро, на відміну від трамвая, завжди прокладають окремо від вуличного транспорту і зазвичай — в тунелях, або рідше — на естакадах.

Табло (Board) - пристрій відображення інформації, зображення на якому формується електронним способом.

## Розділ 1.2 Опис предметної області

1. Голуб Б.М. C#. Концепція та синтаксис, Львів: Видавничий центр ЛНУ ім. Івана Франка, 2006 – 136с.

2. Джесс Ліберті. Програмування на C#, Символ-Плюс, 2003 – 777с.

Метро – міський вид транспорту, схожий на залізницю. Складається з декількох ліній (віток), які зазвичай прокладають під землею, рідше – на естакадах. По лініям рухаються потяги з різним інтервалом часу. Лінії містять станції, які розташовані на деякій відстані одна від одної. На кожній станції повинно знаходитись табло, яке вказую назву станції, поточний час, та іншу інформацію про слідування потягів, рекламу та інше.

## Розділ 1.3 Неформальна постановка задачі

Потрібно розробити модель програмного забезпечення табло для інформаційної служби метрополітену. Табло розташовані на кожній станції метро. Вони працюють під керуванням єдиного пункту управління (ПУ) інформаційної служби метро.

Табло відображає поточний час (години, хвилини, секунди) і час, що минув з моменту відправлення останнього потягу (хвилини, секунди). Момент прибуття і відправлення потягу визначається за допомогою датчиків, що встановлюються на лініях. Усі табло метро синхронізовано, поточний час відраховується і встановлюється з центральної служби часу, що знаходиться на ПУ. На табло висвічується кінцева станція призначення потягу, що прибуває. Ці дані містяться в розкладі руху поїздів, яке зберігається в пам'яті табло і періодично оновлюється з ПУ.

У «рухомому рядку» табло відображається рекламна інформація. Пам'ять табло зберігає до 10 рекламних повідомлень. Повідомлення відображаються один за одним з невеликими паузами, циклічно. Зміст рекламних повідомлень надходить з ПУ. Додаткова функція табло - за запитом з ПУ вона пересилає дані про порушення розкладу (передчасних відправленнях потягів або запізнення).

Пояснення: в завданні потрібно розробити модель ПЗ тільки для табло, але не для пункту управління інформаційної служби.

Також, з власної ініціативи, я додав до завдання такі вимоги:

Відтворити приблизну симуляцію роботи метрополітену. Реалізувати створення ліній метро, станцій для кожної лінії, та потягів. Відобразити рух потягів між станціями, в обидва кінці віток. Користувач має змогу розташовувати станції на уявній карті, задавати кількість потягів для кожної лінії, час їх відправлення, та кількість етапів. Користувач може видаляти потяги, станції, лінії за бажанням. Кожна лінія, станція повинна мати назву, яка відображається як на полі, так і в табло. Також присутня допомога користувачеві у вигляді правил користування програмою.

## Розділ 1.4 План робіт

Термін розробки ~ 6 тижнів:

• Вивчення літератури – 7 днів;

• Проектування та розробка інтерфейсу – 10 днів;

• Реалізація та написання коду ~ 20 днів;

• Оформлення документації – 3 дні.

# Розділ 2. Вимоги до оточення

# Розділ 2.1 Вимоги до апаратного забезпечення

Мінімальні вимоги до апаратного забезпечення:

**Оперативна пам’ять** ємністю 128 Мб.

**Звукова карта** або звукові виходи.

**Монітор** з розподільною здатністю 800х600.

Рекомендовані вимоги до апаратного забезпечення:

**Процесор** з тактовою частотою не менше 1 ГГц.

**Оперативна пам’ять** ємністю 1024Мб.

**Жорсткий диск** об’ємом 100Мб

**Графічний процесор** 128 Мб пам'яті (адаптер з підтримкою DirectX 9), драйвери WDDM версії 1.0 і старше.

**Звукова карта** або звукові виходи.

**Монітор** з розподільною здатністю 800х600.

# Розділ 2.2 Вимоги до програмного забезпечення

# Операційна Windows 7 та вище, а також встановлений .Net Framework версії не менше за 4.5.

# Розділ 2.3 Вимоги до користувача

Користувач повинен пройти базовий курс користування операційною системою Windows. В цьому випадку програмне забезпечення не потребує спеціальних навичок користування.

# Розділ 3. Архітектура системи

Проаналізувавши постановку завдання можна виділити наступні класи:

* CommandCentre;(Пункт управління, головне вікно)
* Subway;(Метро)
* Railway;(Лінія метро)
* Station;(Станція метро)
* Train;(Потяг)
* Board;(Табло метро)

## Розділ 3.1. Діаграма класів

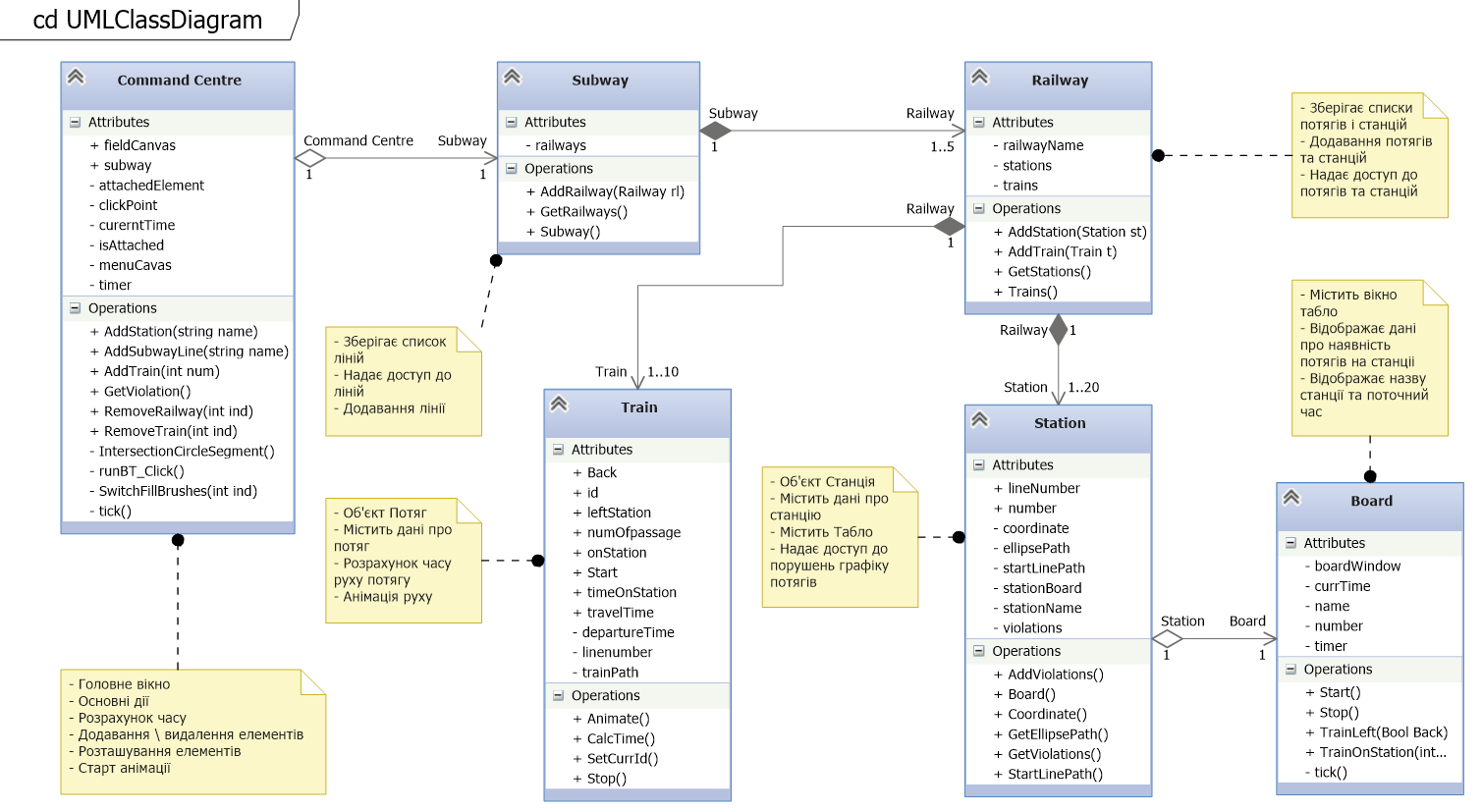


Рисунок 1 - Діаграма класів

## Розділ 3.2. Основні атрибути і методи. Їх короткий опис

**Клас CommandCentre(MainWindow)**

Атрибути:

Canvas fieldCanvas – поле, на якому розміщуються лінії метро.

Canvas menuCanvas – поле, на якому розміщується меню програми.

Subway subway – об’єкт метро.

DispatcherTimer timer – головній таймер.

TimeSpan currentTime – поточний час.

FrameworkElement attachedElement – вибраний елемент на полі.

Методи:

private void Add\_Station\_Ellipse\_On\_Field(object sender, RoutedEventArgs e)

- додає станцію на поле.

private void Add\_Station\_Ellipse\_On\_Field(object sender, RoutedEventArgs e)

- відповідає за переміщення елементів по полю за допомогою миші.

private void Delete\_Station\_Click(object sender, RoutedEventArgs e)

- видалення станції з поля.

private bool IntersectionCircleSegment(double x1, double y1, double x2, double y2, double xC, double yC, double R) – перевіряє перетин кола та прямої.

private void runBT\_Click(object sender, RoutedEventArgs e) – запускає роботу метрополітену.

public bool RemoveTrain(int num) – видаляє потяг за номером.

private void HelpBT\_Click(object sender, RoutedEventArgs e) – викликає допомогу для користувача.

private void tick(object sender, EventArgs e) – відраховує час.

**Клас Subway**

Атрибути:

private List<Railway> \_railways – список ліній метро.

Методи:

public List<Railway> GetRailways – надає доступ до ліній метро.

public void addRailway(Railway rl) – додає лінію до списку.

**Клас Station**

Атрибути:

private TextBlock \_stationName – і’мя станції.

private Point \_coordinate – координати станції на полі.

private Path \_ellipsePath – зображення станції на полі.

private Board \_stationBoard – табло станції.

public int Number – номер станції.

private List<String> violationsSchedules – порушення графіку на станції.

Методи:

public void AddViolations(String str) – додати порушення.

**Клас Train**

Атрибути:

private TimeSpan departureTime – час прибуття на станцію.

private Path trainPath – зображення потягу на полі.

public int travelTime – час в дорозі.

public bool Back – чи їде потяг в зворотньому напрямку.

public int numOfPassage – кількість етапів.

public int timeOnStation – час проведений на станції.

public bool Start – старт руху потягу.

public int id – номер станції, на якій знаходиться потяг.

public bool onStation – потяг знаходиться на станції.

public bool leftStation – потяг покинув станцію.

Методи:

internal void Stop(Canvas field, Subway subway) – зупиняє рух потягу.

internal void SetCurrId(Subway subway) – встановлює номер поточної станції.

internal void calcTime(Subway subway, int i, int j) – розраховує час подорожі потягу.

internal void Animate(Subway subway) – анімація руху потяга.

**Клас Board**

Атрибути:

private BoardWindow bWin – вікно табло.

private TimeSpan \_currentTime – поточний час.

public int StationNumber – номер станції, на якій знаходиться табло.

DispatcherTimer timer – таймер.

Методи:

public void Start(int h, int m, int s) – запуск таймеру.

public void Stop() – зупинка таймеру.

private void tick(object sender, EventArgs e) – розрахунок часу.

public void TrainOnStation(int num, bool back) – відображає повідомлення, якщо потяг на станції.

public void TrainLeft(bool back) – відображає повідомлення, якщо потяг покину станцію.

## Розділ 3.3. Відношення між класами

Клас CommandCentre являється одночасно і головним вікном проекту. Він містить об’єкт класу Subway, над яким виконуються операції. В класі присутні обробники подій кнопок, а також методи, які відповідають за правильне розташування станцій на полі та головний таймер, який відраховує щосекундно час.

У класі Subway міститься список ліній – об’єктів класу Railway, та є можливість додавати нові лінії.

Клас Railway відповідає лінії метро - в ньому знаходяться списки об’єктів класів Train та Station.

Клас Train містить дані потяга, такі як час відправки, номер потягу, кількість етапів та інше. Також містить метод Animate() який відтворює анімацію потягу на полі.

У класі Station знаходиться інформація про станцію, а також об’єкт класу Board. У цьому класі можливо отримати інформацію про порушення графіку руху потягів.

І нарешті клас Board є власне табло, яке знаходиться на кожній станції, воно має полем вікно BoardWindow, на якому знаходиться інформація про станцію та потяги, які повинні прибути.

## Розділ 3.4.Варіанти користування

На діаграмі показана модель співвідношення ролей розробника і користувача до модулів та функціоналу програми.

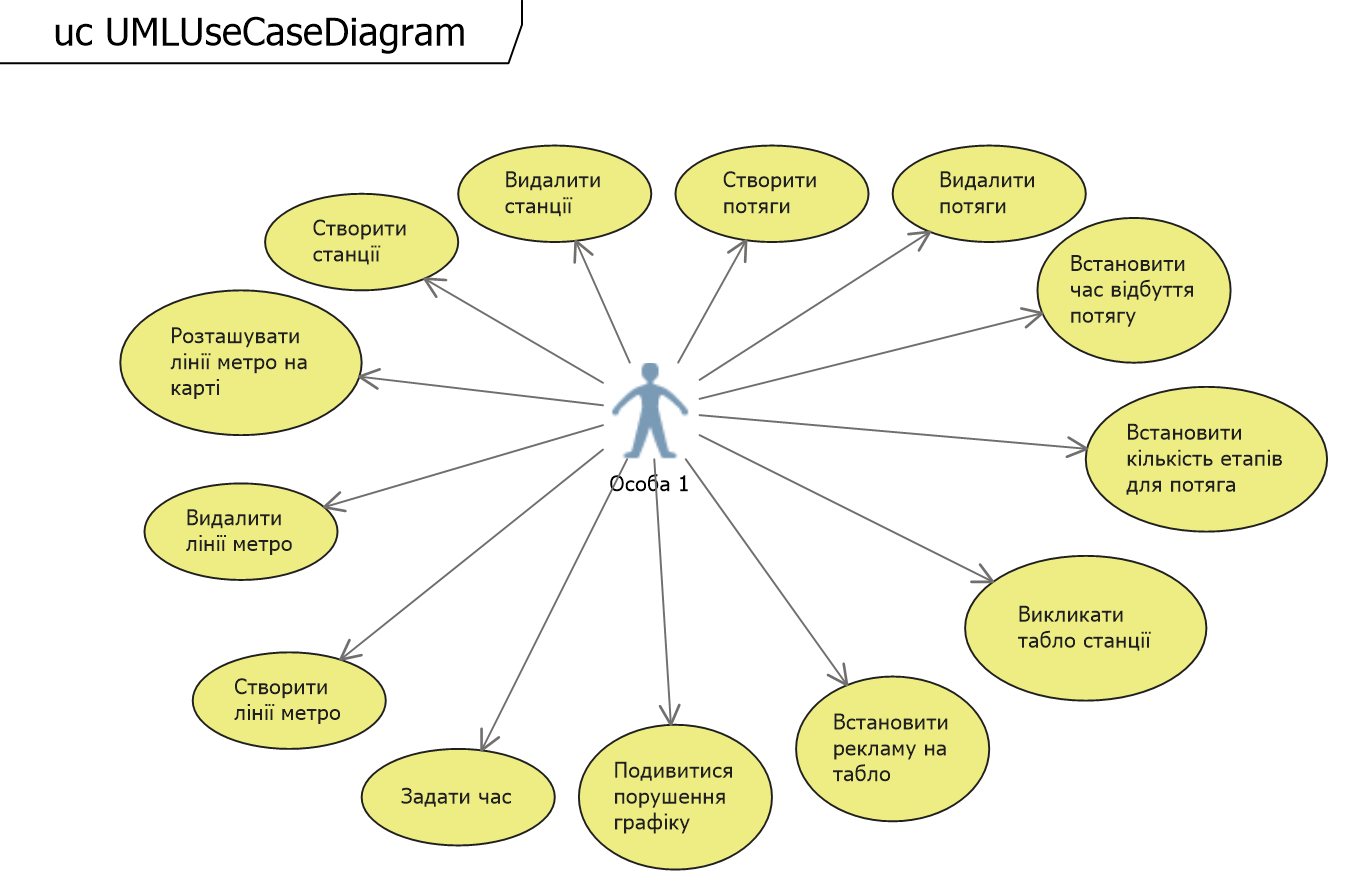


Рисунок 2 - Варіанти використання

# Розділ 4. Проект програмної системі

## Розділ 4.1. Засоби реалізації

Модель програмного забезпечення була реалізована засобами мови С #. Середа розробки: Microsoft Visual Studio 2015 з використанням системи для побудови клієнтських додатків WPF.

## Розділ «4.2 Проект інтерфейсу»

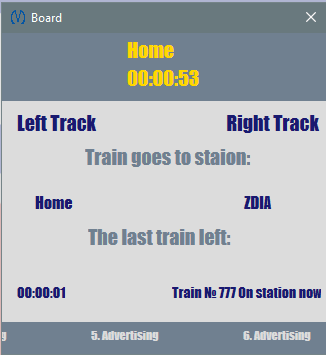


Рисунок 3 – Табло метро

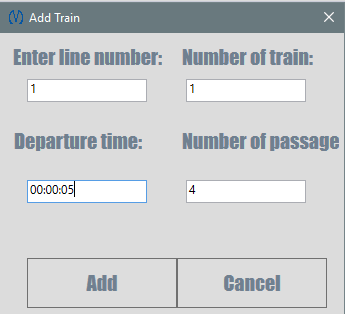


Рисунок 4 - Додавання потягу

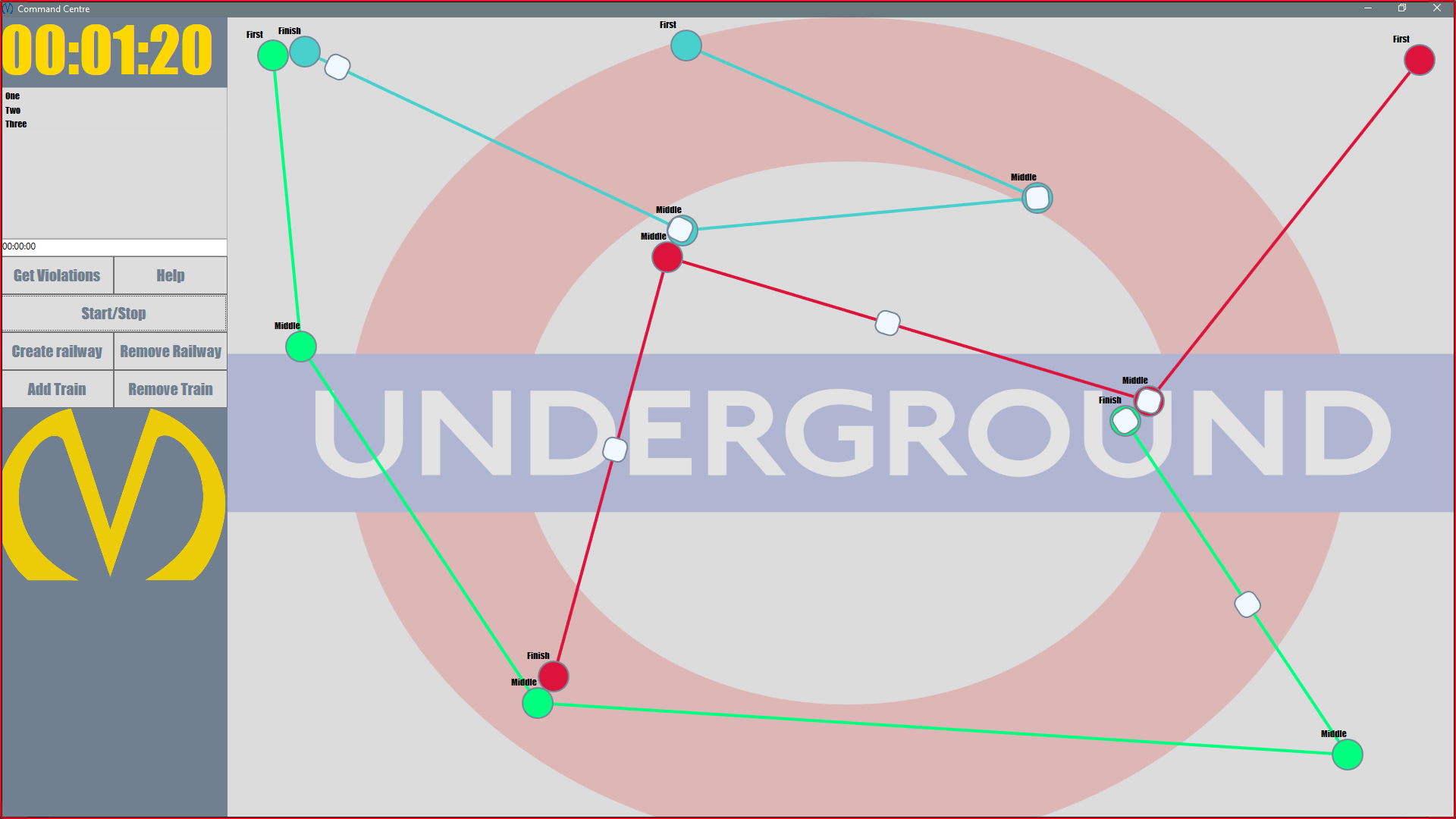


Рисунок 5 - Програма в дії

## Розділ 4.3. **Реалізація та тестування**

Фізичні характеристики даної системи:

1) Об’єм коду в строках– 1500.

2) Об’єм програми в КБ– 2000.

3) Кількість форм – 8.

4) Витрати оперативної пам'яті МБ ~ 50мб.

5) Затрати часу на обробку основних операцій~ 0.7 сек.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Опис тесту | Версія програми | Дата | Результат |
| 1 | Додавання 3 ліній на поле та видалення | 0.1 | 20.11.15 | Пройдений |
| 2 | Додавання 12 станцій на поле та видалення | 0.3 | 25.11.15 | Пройдений |
| 3 | Додавання 10 потягів та їх видалення | 0.5 | 17.12.15 | Пройдений |
| 4 | Анімація руху потягів | 0.7 | 20.12.15 | Пройдений |
| 5 | Відображення порушення графіку потягів | 0.8 | 20.12.15 | Пройдений |
| 6 | Відображення табло на станціях | 0.9 | 25.12.15 | Пройдений |
| 7 | Переміщення елементів по полю | 0.9 | 25.12.15 | Пройдений |

# **Висновки**

* У процесі виконання курсової роботи було розроблено модель програмного забезпечення табло метро засобами мови С #
* Було спроектовано об'єктно-орієнтовану модель програмного забезпечення: виділено основні класи, встановлені залежності між ними і побудовано діаграму класів. Засвоєні принципи об'єктно - орієнтованого програмування (успадкування, інкапсуляція, поліморфізм), залежність між класами.
* Отримано безцінний досвід роботи з відносно великим проектом, розробкою його архітектури, та реалізацією.
* Закріплено знання побудови користувальницьких застосунків за допомогою технології WPF.
* Було виконане додаткове завдання, яке полягало в створенні імітації роботи метрополітену.

# Література

1.Веселов А.І., Попівщий В.І. Методичні вказівки до виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування», Запоріжжя. ЗДІА, 2010.

2. Оформлення і захист курсових, дипломних та кваліфікаційних робіт Методичні вказівки для студ. ЗДІА спец. 080403 "ПЗАС" ден. та заоч. форм навчання / А. І. Безверхий, Н. П. Полякова, В. І. Попівщий, І. А. Скрипник ; ЗДІА. - Запоріжжя : ЗДІА, 2007.

3. Гайсарян С.С. Объектно-ориентированное программирование. М.: ЦИТ, 2002.

4.Проценко В.С., Чаленко П.Й., Ставровський А.Б. Техніка програмування мовою Сі. – К.: Либідь, 1993.

5. MSDN Online Library: http://msdn.microsoft.com/

# Додаток

## Додаток 1

Класс MainWindow

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.ComponentModel;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.IO;

namespace CW\_Underground

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

fieldCanvas.MouseMove += Mouse\_Move\_On\_Field;

fieldCanvas.MouseLeftButtonUp += Field\_Mouse\_Left\_Up;

fieldCanvas.MouseLeave += Field\_Mouse\_Leave;

ContextMenu menu = new ContextMenu();

MenuItem create = new MenuItem();

create.Header = "Add station";

menu.Items.Add(create);

create.Click += Add\_Station\_Ellipse\_On\_Field;

fieldCanvas.ContextMenu = menu;

}

private void On\_Load(object sender, RoutedEventArgs e)

{

//set the size of the canvas relative to the maximum window size

if (this.WindowState == WindowState.Maximized)

{

fieldCanvas.Width = mainWindow.ActualWidth - 300;

menuCanvas.Height = fieldCanvas.Height = mainWindow.ActualHeight;

ImageBrush ib = new ImageBrush(new BitmapImage(new Uri("back.png", UriKind.Relative)));

fieldCanvas.Background = ib;

}

}

private void On\_Closing(object sender, CancelEventArgs e)

{

foreach (Railway rl in subway.GetRailways)

{

foreach (Station st in rl.GetStations)

{

st.Board.Stop();

if (st.Board.BoardWindow != null)

{

st.Board.BoardWindow.Close();

}

}

}

timer.Stop();

Application.Current.Shutdown();

}

private void Mouse\_Left\_Button\_Down\_menuCanvas(object sender, MouseEventArgs e)

{

SubwayLinesLBox.SelectedIndex = -1;

if (fieldCanvas.Children.Count > 0 && fieldCanvas.Children[fieldCanvas.Children.Count - 1] is Menu)

{

//clear right click menu in field

fieldCanvas.Children.RemoveAt(fieldCanvas.Children.Count - 1);

}

}

internal Subway subway = new Subway();

private void Add\_Station\_Ellipse\_On\_Field(object sender, RoutedEventArgs e)

{

if (on)

{

if (fieldCanvas.Children.Count > 0 && fieldCanvas.Children[fieldCanvas.Children.Count - 1] is Menu)

{

//clear right click menu in field

fieldCanvas.Children.RemoveAt(fieldCanvas.Children.Count - 1);

}

if (SubwayLinesLBox.SelectedIndex != -1)

{

double x = Mouse.GetPosition(fieldCanvas).X;

double y = Mouse.GetPosition(fieldCanvas).Y;

//if too near for window borders

if (x < 20 || y < 20 ||

x > (fieldCanvas.Width - 20) || y > (fieldCanvas.Height - 20))

{ return; }

Point point = new Point(x, y);

foreach (Railway rl in subway.GetRailways)

{

foreach (Station st in rl.GetStations)

{

//if too near for 2 any station

if (Math.Sqrt((st.Coordinate.X - point.X) \* (st.Coordinate.X - point.X) + (st.Coordinate.Y - point.Y) \* (st.Coordinate.Y - point.Y)) < 40)

{

TooNearStationPP.IsOpen = true;

return;

}

}

}

foreach (Railway rl in subway.GetRailways)

{

//if too near for station and railway

if (rl.GetStations.Count > 1)

{

for (int i = 0; i < rl.GetStations.Count - 1; i++)

{

if (IntersectionCircleSegment

(rl.GetStations.ElementAt(i).Coordinate.X, rl.GetStations.ElementAt(i).Coordinate.Y,

rl.GetStations.ElementAt(i + 1).Coordinate.X, rl.GetStations.ElementAt(i + 1).Coordinate.Y,

x, y, 30))

{

TooNearPathPP.IsOpen = true;

return;

}

}

}

}

foreach (Station st in subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations)

{

//if too near for 2 station at 1 railway

if (Math.Sqrt((st.Coordinate.X - point.X) \* (st.Coordinate.X - point.X) +

(st.Coordinate.Y - point.Y) \* (st.Coordinate.Y - point.Y)) < 120)

{

TooNearStationPP.IsOpen = true;

return;

}

}

foreach (Railway rl in subway.GetRailways)

{

int count = (subway.GetRailways.IndexOf(rl) == SubwayLinesLBox.SelectedIndex) ? rl.GetStations.Count - 1 : rl.GetStations.Count;

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

{

//if the railway crosses stations

if (subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count > 0)

{

if (IntersectionCircleSegment

(subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.

ElementAt(subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count - 1).Coordinate.X,

subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.

ElementAt(subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count - 1).Coordinate.Y,

point.X, point.Y, rl.GetStations.ElementAt(i).Coordinate.X, rl.GetStations.ElementAt(i).Coordinate.Y, 30))

{

CrossesStationPP.IsOpen = true;

return;

}

}

}

}

AddStationWindow win = new AddStationWindow(this);

win.Owner = this;

win.ShowDialog();

if (DialogResultOk)

{

DialogResultOk = false;

//Paint railway and station

if (subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count > 1)

{

System.Windows.Shapes.Path linePath = new System.Windows.Shapes.Path();

linePath.Stroke = SwitchFillBrushes(SubwayLinesLBox.SelectedIndex);

linePath.StrokeThickness = 4;

linePath.HorizontalAlignment = HorizontalAlignment.Center;

linePath.VerticalAlignment = VerticalAlignment.Center;

List<Point> pointsList = new List<Point>();

pointsList.Add(point);

pointsList.Add(point);

pointsList.Add(point);

PathFigure lineFigure = new PathFigure();

Point ps = subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.

ElementAt(subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count - 2).Coordinate;

lineFigure.StartPoint = ps;

lineFigure.Segments.Add(new PolyBezierSegment(pointsList, true));

PathGeometry lineGeometry = new PathGeometry();

lineGeometry.Figures.Add(lineFigure);

linePath.Data = lineGeometry;

linePath.MouseLeftButtonDown += Line\_Mouse\_Left\_Down;

linePath.MouseLeftButtonUp += Line\_Mouse\_Left\_Up;

Canvas.SetZIndex(linePath, -1);

subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.

ElementAt(subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count - 2).startLinePath = linePath;

fieldCanvas.Children.Add(linePath);

linePath = new System.Windows.Shapes.Path();

lineFigure = new PathFigure();

pointsList = new List<Point>();

pointsList.Add(ps);

pointsList.Add(ps);

pointsList.Add(ps);

lineFigure.StartPoint = point;

lineFigure.Segments.Add(new PolyBezierSegment(pointsList, true));

lineGeometry = new PathGeometry();

lineGeometry.Figures.Add(lineFigure);

linePath.Data = lineGeometry;

subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.

ElementAt(subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count - 1).endLinePath = linePath;

}

System.Windows.Shapes.Path ellipsePath = new System.Windows.Shapes.Path();

ellipsePath.Stroke = Brushes.LightSlateGray;

ellipsePath.Fill = SwitchFillBrushes(SubwayLinesLBox.SelectedIndex);

ellipsePath.StrokeThickness = 2;

ellipsePath.HorizontalAlignment = HorizontalAlignment.Center;

ellipsePath.VerticalAlignment = VerticalAlignment.Center;

EllipseGeometry myEllipseGeometry = new EllipseGeometry();

myEllipseGeometry.Center = new Point(x, y);

myEllipseGeometry.RadiusX = 20;

myEllipseGeometry.RadiusY = 20;

ellipsePath.Data = myEllipseGeometry;

ellipsePath.MouseLeftButtonDown += Ellipse\_Mouse\_Left\_Down;

ellipsePath.MouseLeftButtonUp += Ellipse\_Mouse\_Left\_Up;

ellipsePath.MouseRightButtonDown += Ellipse\_Mouse\_Right\_Down;

subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.

ElementAt(subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count - 1).ellipsePath = ellipsePath;

ContextMenu menu = new ContextMenu();

MenuItem delete = new MenuItem();

delete.Header = "Delete station";

menu.Items.Add(delete);

delete.Click += Delete\_Station\_Click;

MenuItem board = new MenuItem();

board.Header = "Open board";

menu.Items.Add(board);

board.Click += Open\_Board\_Click;

ellipsePath.ContextMenu = menu;

fieldCanvas.Children.Add(ellipsePath);

TextBlock stationName = new TextBlock();

stationName.FontFamily = new FontFamily("Impact");

stationName.FontWeight = FontWeights.Light;

stationName.Text = subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.

ElementAt(subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count - 1).stationName.Text;

subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.

ElementAt(subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count - 1).stationName = stationName;

stationName.Margin = new Thickness(x - 35, y - 35, x, y);

fieldCanvas.Children.Add(stationName);

Canvas.SetZIndex(stationName, 100);

//Added station coordinates

subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.

ElementAt(subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count - 1).

Coordinate = new Point(x, y);

}

}

}

}

private bool isAttached = false;

private FrameworkElement attachedElement;

private FrameworkElement attachedElementText;

private FrameworkElement attachedElementLine;

private Point clickPoint;

private void Line\_Mouse\_Left\_Down(object sender, MouseEventArgs e)

{

isAttached = true;

attachedElementLine = sender as FrameworkElement;

foreach (Railway rl in subway.GetRailways)

{

//need for offset lines

for (int i = 0; i < rl.GetStations.Count - 1; i++)

{

if (rl.GetStations.ElementAt(i).startLinePath.Equals(sender as System.Windows.Shapes.Path))

{

attachedElementLine = rl.GetStations.ElementAt(i).startLinePath as FrameworkElement;

clickPoint = rl.GetStations.ElementAt(i + 1).Coordinate;

}

}

}

}

private void Line\_Mouse\_Left\_Up(object sender, MouseEventArgs e)

{

isAttached = false;

}

private void Field\_Mouse\_Leave(object sender, RoutedEventArgs e)

{

isAttached = false;

}

private void Ellipse\_Mouse\_Left\_Down(object sender, MouseEventArgs e)

{

isAttached = true;

attachedElement = sender as FrameworkElement;

clickPoint = e.GetPosition(sender as FrameworkElement);

foreach (Railway rl in subway.GetRailways)

{

//need for offset station name

foreach (Station st in rl.GetStations)

{

if (st.ellipsePath.Equals(sender as System.Windows.Shapes.Path))

{

attachedElementText = st.stationName as FrameworkElement;

}

}

}

}

private void Ellipse\_Mouse\_Left\_Up(object sender, MouseEventArgs e)

{

attachedElement = null;

isAttached = false;

}

private void Field\_Mouse\_Left\_Up(object sender, MouseEventArgs e)

{

isAttached = false;

}

private void Mouse\_Move\_On\_Field(object sender, MouseEventArgs e)

{

if (on)

{

bool block = false;

EllipseGeometry currentEllipse = new EllipseGeometry();

PathGeometry ellipseLine = new PathGeometry();

PathFigure startLineFigure = new PathFigure();

PathFigure endLineFigure = new PathFigure();

PathFigure backLineFigure = new PathFigure();

PathFigure backendLineFigure = new PathFigure();

Station st = new Station();

if (isAttached && attachedElement != null)

{

foreach (Railway rl in subway.GetRailways)

{

for (int i = 0; i < rl.GetStations.Count; i++)

{

if (rl.GetStations[i].ellipsePath == attachedElement as System.Windows.Shapes.Path)

{

currentEllipse = rl.GetStations[i].ellipsePath.Data as EllipseGeometry;

st = rl.GetStations[i];

if (rl.GetStations[i].startLinePath != null)

{

ellipseLine = rl.GetStations[i].startLinePath.Data as PathGeometry;

startLineFigure = ellipseLine.Figures.ElementAt(0);

if (i + 1 < rl.GetStations.Count)

{

backLineFigure = (rl.GetStations[i + 1].endLinePath.Data as PathGeometry).Figures.ElementAt(0);

}

}

if (i > 0)

{

ellipseLine = rl.GetStations[i - 1].startLinePath.Data as PathGeometry;

endLineFigure = ellipseLine.Figures.ElementAt(0);

backendLineFigure = (rl.GetStations[i].endLinePath.Data as PathGeometry).Figures.ElementAt(0);

}

}

}

}

Point p = new Point(Mouse.GetPosition(fieldCanvas).X, Mouse.GetPosition(fieldCanvas).Y);

foreach (Railway rl in subway.GetRailways)

{

foreach (Station s in rl.GetStations)

{

if (s.ellipsePath != st.ellipsePath)

{

if (s.lineNumber == st.lineNumber)

{

if (Math.Sqrt((s.Coordinate.X - Mouse.GetPosition(fieldCanvas).X) \* (s.Coordinate.X - Mouse.GetPosition(fieldCanvas).X) + (s.Coordinate.Y - Mouse.GetPosition(fieldCanvas).Y) \* (s.Coordinate.Y - Mouse.GetPosition(fieldCanvas).Y)) < 100)

{

block = true;

}

}

else

{

if (Math.Sqrt((s.Coordinate.X - Mouse.GetPosition(fieldCanvas).X) \* (s.Coordinate.X - Mouse.GetPosition(fieldCanvas).X) + (s.Coordinate.Y - Mouse.GetPosition(fieldCanvas).Y) \* (s.Coordinate.Y - Mouse.GetPosition(fieldCanvas).Y)) < 40)

{

block = true;

}

}

}

}

}

if (!block)

{

double X = 35;

double Y = 30;

if (Mouse.GetPosition(fieldCanvas).X > 30)

{

X = Mouse.GetPosition(fieldCanvas).X;

}

if (Mouse.GetPosition(fieldCanvas).Y > 30)

{

Y = Mouse.GetPosition(fieldCanvas).Y;

}

currentEllipse.Center = new Point(X, Y);

st.Coordinate = currentEllipse.Center;

st.stationName.Margin = new Thickness(currentEllipse.Center.X - 35, currentEllipse.Center.Y - 35, currentEllipse.Center.X, currentEllipse.Center.Y);

startLineFigure.StartPoint = currentEllipse.Center;

backendLineFigure.StartPoint = currentEllipse.Center;

if (endLineFigure.Segments.Count > 0)

{

endLineFigure.Segments.RemoveAt(0);

}

if (backLineFigure.Segments.Count > 0)

{

backLineFigure.Segments.RemoveAt(0);

}

List<Point> pointsList = new List<Point>();

pointsList.Add(currentEllipse.Center);

pointsList.Add(currentEllipse.Center);

pointsList.Add(currentEllipse.Center);

endLineFigure.Segments.Add(new PolyBezierSegment(pointsList, true));

backLineFigure.Segments.Add(new PolyBezierSegment(pointsList, true));

}

}

}

}

private void Delete\_Station\_Click(object sender, RoutedEventArgs e)

{

if (on)

{

for (int i = 0; i < subway.GetRailways.Count; i++)

{

Railway rl = subway.GetRailways.ElementAt(i);

for (int j = 0; j < subway.GetRailways.ElementAt(i).GetStations.Count; j++)

{

Station st = rl.GetStations.ElementAt(j);

if (st.ellipsePath == currElleipse)

{

if (st.Number == rl.GetStations.Count - 1)

{

fieldCanvas.Children.Remove(st.stationName);

fieldCanvas.Children.Remove(currElleipse);

if (st.Number > 0)

{

fieldCanvas.Children.Remove(rl.GetStations.ElementAt(j - 1).startLinePath);

}

rl.GetStations.RemoveAt(rl.GetStations.Count - 1);

}

else

{

MessageBox.Show("You can delete only the last station!");

}

}

}

}

}

}

private System.Windows.Shapes.Path currElleipse;

private void Ellipse\_Mouse\_Right\_Down(object sender, RoutedEventArgs e)

{

currElleipse = sender as System.Windows.Shapes.Path;

}

private void Open\_Board\_Click(object sender, RoutedEventArgs e)

{

Station station = new Station();

String nameLeft = "";

string nameRight = "";

foreach (Railway rl in subway.GetRailways)

{

foreach (Station st in rl.GetStations)

{

if (st.ellipsePath.Equals(currElleipse))

{

station = st;

nameLeft = rl.GetStations.ElementAt(rl.GetStations.Count - 1).stationName.Text;

nameRight = rl.GetStations.ElementAt(0).stationName.Text;

break;

}

}

}

if (station.Board.BoardWindow == null || !station.Board.BoardWindow.IsLoaded)

{

station.Board.BoardWindow = new BoardWindow();

station.Board.BoardWindow.lastLeftLB.Content = nameLeft;

station.Board.BoardWindow.lastRightLB.Content = nameRight;

station.Board.BoardWindow.Show();

}

}

private bool IntersectionCircleSegment(double x1, double y1, double x2, double y2, double xC, double yC, double R)

{

x1 -= xC;

y1 -= yC;

x2 -= xC;

y2 -= yC;

double dx = x2 - x1;

double dy = y2 - y1;

//составляем коэффициенты квадратного уравнения на пересечение прямой и окружности.

//если на отрезке [0..1] есть отрицательные значения, значит отрезок пересекает окружность

double a = dx \* dx + dy \* dy;

double b = 2.0 \* (x1 \* dx + y1 \* dy);

double c = x1 \* x1 + y1 \* y1 - R \* R;

//а теперь проверяем, есть ли на отрезке [0..1] решения

if (-b < 0)

return (c < 0);

if (-b < (2.0 \* a))

return ((4.0 \* a \* c - b \* b) < 0);

return (a + b + c < 0);

}

internal bool DialogResultOk

{

get;

set;

}

private System.Windows.Media.SolidColorBrush SwitchFillBrushes(int index)

{

switch (index)

{

case 0:

return System.Windows.Media.Brushes.SpringGreen;

case 1:

return System.Windows.Media.Brushes.MediumTurquoise;

case 2:

return System.Windows.Media.Brushes.Crimson;

case 3:

return System.Windows.Media.Brushes.Gold;

default:

return System.Windows.Media.Brushes.Orchid;

}

}

private void Create\_Subway\_Line\_BT\_Click(object sender, RoutedEventArgs e)

{

if (on)

{

AddSubwayLinesWindow win = new AddSubwayLinesWindow(this);

win.Owner = this;

win.ShowDialog();

}

}

internal void AddSubwayLine(String name)

{

if (on)

{

Railway rl = new Railway(name);

subway.addRailway(rl);

SubwayLinesLBox.Items.Add(name);

}

}

internal void AddStation(String name)

{

if (on)

{

Station st = new Station(name);

st.lineNumber = SubwayLinesLBox.SelectedIndex;

st.Number = subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).GetStations.Count;

subway.GetRailways.ElementAt(SubwayLinesLBox.SelectedIndex).addStation(st);

}

}

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

{

if (on)

{

AddTrainWindow win = new AddTrainWindow(this);

win.Owner = this;

win.ShowDialog();

}

}

public bool addTrain(Train t, int i, int num, String time)

{

try

{

if (subway.GetRailways.ElementAt(i).GetStations.Count > 1)

{

int hour, min, sec;

String[] arr = time.Split(':');

hour = Convert.ToInt32(arr[0]);

min = Convert.ToInt32(arr[1]);

sec = Convert.ToInt32(arr[2]);

TimeSpan tp = new TimeSpan(hour, min, sec);

foreach (Railway rl in subway.GetRailways)

{

foreach (Train tr in rl.Trains)

{

TimeSpan tps = tr.DepartureTime.Subtract(tp);

if (tps < new TimeSpan(0, 0, 0))

{

tps = -tps;

}

if (tps < new TimeSpan(0, 0, 10) && tr.LineNumber == t.LineNumber)

{

MessageBox.Show("Min interval across trains - 15 seconds");

return false;

}

if (t.Number == tr.Number)

{

MessageBox.Show("Already exists with this number!");

return false;

}

}

}

t.DepartureTime = tp;

t.TrainPath.Stroke = Brushes.LightSlateGray;

t.TrainPath.Fill = Brushes.AliceBlue;

t.TrainPath.StrokeThickness = 2;

t.TrainPath.HorizontalAlignment = HorizontalAlignment.Center;

t.TrainPath.VerticalAlignment = VerticalAlignment.Center;

RectangleGeometry myEllipseGeometry = new RectangleGeometry();

Point p = subway.GetRailways.ElementAt(i).GetStations.ElementAt(0).Coordinate;

myEllipseGeometry.Rect = new Rect(p.X - 15, p.Y - 15, 30, 30);

myEllipseGeometry.RadiusX = 10;

myEllipseGeometry.RadiusY = 10;

t.TrainPath.Data = myEllipseGeometry;

t.numOfPassage = num + 1;

t.passage = num + 1;

subway.GetRailways.ElementAt(i).addTrain(t);

return true;

}

else

{

MessageBox.Show("On the railway should be more than one station");

return false;

}

}

catch (Exception)

{

MessageBox.Show("No this railway");

return false;

}

}

System.Windows.Threading.DispatcherTimer timer = new System.Windows.Threading.DispatcherTimer();

private bool on = true;

private int hour = 0, min = 0, sec = 0;

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

{

if (on)

{

try

{

String[] arr = timeTBox.Text.Split(':');

hour = Convert.ToInt32(arr[0]);

min = Convert.ToInt32(arr[1]);

sec = Convert.ToInt32(arr[2]);

timeTBlock.Text = new TimeSpan(hour, min, sec).ToString();

}

catch (Exception)

{

MessageBox.Show("Enter Time!");

return;

}

timer.Tick += new EventHandler(tick);

timer.Interval = new TimeSpan(0, 0, 0, 1);

timer.Start();

on = false;

foreach (Railway rl in subway.GetRailways)

{

foreach (Station st in rl.GetStations)

{

st.Board.Start(hour, min, sec);

}

}

}

else

{

timer.Stop();

on = true;

foreach (Railway rl in subway.GetRailways)

{

foreach (Train t in rl.Trains)

{

t.Stop(fieldCanvas, subway);

}

foreach (Station st in rl.GetStations)

{

st.Board.Stop();

}

}

timer.Tick -= new EventHandler(tick);

}

}

private TimeSpan currentTime;

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

{

if (on)

{

RemoveSubwayLineWindow win = new RemoveSubwayLineWindow(this);

win.Owner = this;

win.ShowDialog();

}

}

public void RemoveRailWay(int num)

{

try

{

if (subway.GetRailways.ElementAt(num - 1).GetStations.Count == 0)

{

SubwayLinesLBox.Items.RemoveAt(num - 1);

}

else

{

MessageBox.Show("It is necessary to remove all stations!");

}

}

catch (Exception)

{

MessageBox.Show("Wrong number!");

}

}

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

{

if (on)

{

RemoveTrainWindow win = new RemoveTrainWindow(this);

win.Owner = this;

win.ShowDialog();

}

}

public bool RemoveTrain(int num)

{

foreach (Railway rl in subway.GetRailways)

{

foreach (Train t in rl.Trains)

{

if (t.Number == num)

{

rl.Trains.Remove(t);

return true;

}

}

}

return false;

}

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

{

ViolationWindow win = new ViolationWindow(this);

win.Owner = this;

win.ShowDialog();

}

public string GetViolation()

{

string str = "";

foreach (Railway rl in subway.GetRailways)

{

foreach (Station st in rl.GetStations)

{

foreach (string s in st.GetViolations)

{

str += s;

str += '\n';

}

}

}

return str;

}

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

{

string strLine; StringBuilder str = new StringBuilder();

try

{

FileStream aFile = new FileStream("Help.txt", FileMode.Open);

StreamReader sr = new StreamReader(aFile);

strLine = sr.ReadLine();

while (strLine != null)

{

str.Append(strLine);

str.Append(" ");

strLine = sr.ReadLine();

}

sr.Close();

}

catch (IOException)

{

MessageBox.Show("IO ERROR");

}

MessageBox.Show(str.ToString());

}

//private void backgroundBT\_Click(object sender, RoutedEventArgs e)

//{

// ImageBrush ib = new ImageBrush(new BitmapImage(new Uri("map1.jpg", UriKind.Relative)));

// fieldCanvas.Width = mainWindow.ActualWidth - 300 + ib.ImageSource.Width;

// menuCanvas.Height = fieldCanvas.Height = mainWindow.ActualHeight + ib.ImageSource.Height;

// fieldCanvas.Background = ib;

//}

private void AddTick(ref int hour, ref int min, ref int sec)//add one second

{

sec++;

if (sec > 59) { sec = 0; min++; }

if (min > 59) { min = 0; hour++; }

if (hour > 23) { hour = 0; }

currentTime = new TimeSpan(hour, min, sec);

timeTBlock.Text = new TimeSpan(hour, min, sec).ToString();

}

private void tick(object sender, EventArgs e)

{

Railway rl = new Railway();

Train t = new Train();

for (int i = 0; i < subway.GetRailways.Count; i++)

{

rl = subway.GetRailways.ElementAt(i);

for (int j = 0; j < subway.GetRailways.ElementAt(i).Trains.Count; j++)

{

t = rl.Trains.ElementAt(j);

if (rl.GetStations.Count > 1)

{

////////////////////////

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Start && subway.GetRailways.ElementAt(i).Trains.ElementAt(j).travelTime == 0 &&

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation != 0 || !subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Start &&

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).DepartureTime == currentTime)

{

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).DepartureTime == currentTime)

{

if (!fieldCanvas.Children.Contains(subway.GetRailways.ElementAt(i).Trains.ElementAt(j).TrainPath))

{

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Stop(fieldCanvas, subway);

fieldCanvas.Children.Add(subway.GetRailways.ElementAt(i).Trains.ElementAt(j).TrainPath);

}

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Start = true;

}

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).onStation = true;

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation--;

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation != 0)

{

Random r = new Random(DateTime.Now.Millisecond);

if (r.Next(1, 100) % 55 == 0)

{

String str = "Train number: " + subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Number + " went ahead by " + subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation + " seconds at the station: " + subway.GetRailways.ElementAt(i).Trains.ElementAt(j).id;

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation = 0;

subway.GetRailways.ElementAt(i).GetStations.ElementAt(subway.GetRailways.ElementAt(i).Trains.ElementAt(j).id).AddViolations(str);

}

}

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).away && subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation == 0)

{

if (t.Back)

{

rl.GetStations.ElementAt(t.id).Board.TrainLeft(true);

}

else

{

rl.GetStations.ElementAt(t.id).Board.TrainLeft(false);

}

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Back = !subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Back;

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).numOfPassage != 1)

{

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation = 3;

}

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).currStationInd = 0;

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).away = false;

}

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation == 0)

{

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).onStation = false;

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).leftStation = true;

}

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).SetCurrId(subway);

}

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).numOfPassage == 1 &&

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).travelTime == 0 && subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation == 0)

{

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Start = false;

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Stop(fieldCanvas, subway);

}

/////////////////////////////////////

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).travelTime != 0 && subway.GetRailways.ElementAt(i).Trains.ElementAt(j).travelTime > 0)

{

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).travelTime--;

}

///////////////////////////

if (t.onStation)

{

if (t.Back)

{

rl.GetStations.ElementAt(t.id).Board.TrainOnStation(t.Number, true);

}

else

{

rl.GetStations.ElementAt(t.id).Board.TrainOnStation(t.Number, false);

}

}

if (t.leftStation)

{

if (t.Back)

{

rl.GetStations.ElementAt(t.id).Board.TrainLeft(true);

}

else

{

rl.GetStations.ElementAt(t.id).Board.TrainLeft(false);

}

t.leftStation = false;

}

//////////////////////////

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Start && subway.GetRailways.ElementAt(i).Trains.ElementAt(j).travelTime == 0 && subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation == 0)

{

if (!fieldCanvas.Children.Contains(subway.GetRailways.ElementAt(i).Trains.ElementAt(j).TrainPath))

{

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Stop(fieldCanvas, subway);

fieldCanvas.Children.Add(subway.GetRailways.ElementAt(i).Trains.ElementAt(j).TrainPath);

}

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).calcTime(subway, i, j);

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Animate(subway);

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).currStationInd < subway.GetRailways.ElementAt(i).GetStations.Count - 2)

{

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).currStationInd++;

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation = 3;

}

else

{

if (subway.GetRailways.ElementAt(i).Trains.ElementAt(j).numOfPassage != 1)

{

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).numOfPassage--;

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).away = true;

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).currStationInd++;

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation = 3;

}

else

{

subway.GetRailways.ElementAt(i).Trains.ElementAt(j).timeOnStation = 3;

}

}

}

//////////////////////////////////////////

}

}

}

AddTick(ref hour, ref min, ref sec);

}

}

}

Класс Subway

using System.Collections.Generic;

namespace CW\_Underground

{

class Subway

{

private List<Railway> \_railways = new List<Railway>();

public List<Railway> GetRailways

{

get { return \_railways; }

}

public void addRailway(Railway rl)

{

\_railways.Add(rl);

}

public Subway(){ }

}

}

Класс Railway

using System.Collections.Generic;

namespace CW\_Underground

{

class Railway

{

private string \_railwayName;

private List<Station> \_stations = new List<Station>();

public List<Station> GetStations

{

get { return \_stations; }

}

private List<Train> \_trains = new List<Train>();

public List<Train> Trains

{

get { return \_trains; }

set { \_trains = value; }

}

public void addTrain(Train t)

{

\_trains.Add(t);

}

public void addStation(Station st)

{

\_stations.Add(st);

}

public Railway()

{

\_railwayName = "Railway";

}

public Railway(string name)

{

\_railwayName = name;

}

}

}

Класс Station

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Shapes;

using System.Windows.Controls;

namespace CW\_Underground

{

class Station

{

private TextBlock \_stationName;

private Point \_coordinate;

private Path \_ellipsePath;//for station ellipse on field

private Path \_startLinePath;//for railway in station

private Board \_stationBoard;

public int lineNumber;

public int Number;

private Path \_endLinePath;//for railway in station

private List<String> violationsSchedules = new List<string>();

public Path ellipsePath

{

get { return \_ellipsePath; }

set { \_ellipsePath = value; }

}

public List<string> GetViolations

{

get { return violationsSchedules; }

set { violationsSchedules = value; }

}

public void AddViolations(String str)

{

violationsSchedules.Add(str);

}

public Board Board

{

get { return \_stationBoard; }

set { \_stationBoard = value; }

}

public Path startLinePath

{

get { return \_startLinePath; }

set { \_startLinePath = value; }

}

public Path endLinePath

{

get { return \_endLinePath; }

set { \_endLinePath = value; }

}

public TextBlock stationName

{

get { return \_stationName; }

set { \_stationName = value; }

}

public Point Coordinate

{

get { return \_coordinate; }

set { \_coordinate = value; }

}

public Station(string name)

{

\_stationName = new TextBlock();

\_stationName.Text = name;

Board = new Board(Number, stationName.Text);

}

public Station()

{

Board = new Board(Number, "name");

}

}

}

Класс Train

using System;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Shapes;

using System.Windows.Media;

using System.Windows.Media.Animation;

namespace CW\_Underground

{

public class Train

{

private TimeSpan departureTime;

private int lineNumber;

private Path trainPath = new Path();

public int travelTime = 1;

public bool Back;

public int passage;

public int numOfPassage;

public int timeOnStation = 3;

public bool Start = false;

public int currStationInd = 0;

public int id = 0;

public int Number;

public bool away = false;

public bool onStation = false;

public bool leftStation = false;

private Storyboard pathAnimationStoryboard = new Storyboard();

internal void Stop(Canvas field, Subway subway)

{

pathAnimationStoryboard.Children.Clear();

field.Children.Remove(trainPath);

RectangleGeometry rc = trainPath.Data as RectangleGeometry;

trainPath.RenderTransform = null;

travelTime = 0;

timeOnStation = 3;

Start = false;

onStation = false;

leftStation = false;

numOfPassage = passage;

currStationInd = 0;

id = 0;

Back = false;

away = false;

if (subway.GetRailways.ElementAt(lineNumber).GetStations.Count > 1)

{

if (subway.GetRailways.ElementAt(lineNumber).GetStations.ElementAt(0) != null)

{

Station st = subway.GetRailways.ElementAt(lineNumber).GetStations.ElementAt(0);

rc.Rect = new Rect(st.Coordinate.X - 15, st.Coordinate.Y - 15, 30, 30);

}

else

{

trainPath = null;

}

}

}

internal void SetCurrId(Subway subway)

{

if (Back)

{

id = subway.GetRailways.ElementAt(lineNumber).GetStations.Count - currStationInd - 1;

}

else

{

id = currStationInd;

}

}

internal void calcTime(Subway subway, int i, int j)

{

Point ps = subway.GetRailways.ElementAt(i).GetStations.ElementAt(id).Coordinate;

Point p = new Point();

if (Back)

{

p = subway.GetRailways.ElementAt(i).GetStations.ElementAt(id - 1).Coordinate;

}

else

{

if (id + 1 < subway.GetRailways.ElementAt(i).GetStations.Count)

{

p = subway.GetRailways.ElementAt(i).GetStations.ElementAt(id + 1).Coordinate;

}

}

int time = (int)(Math.Sqrt((p.X - ps.X) \* (p.X - ps.X) + (p.Y - ps.Y) \* (p.Y - ps.Y))) / 50;

Random r = new Random(DateTime.Now.Millisecond);

if (r.Next(1, 30) % 20 == 0)

{

int temp = time / 10;

if (temp == 0)

{

temp = 1;

}

time += temp;

String str = "Train number: " + subway.GetRailways.ElementAt(i).Trains.ElementAt(j).Number + " was late by " + temp + " seconds at the station: " + id;

subway.GetRailways.ElementAt(i).GetStations.ElementAt(id).AddViolations(str);

}

travelTime = time;

}

internal void Animate(Subway subway)

{

pathAnimationStoryboard = new Storyboard();

Start = true;

MatrixTransform buttonMatrixTransform = new MatrixTransform();

this.TrainPath.RenderTransform = buttonMatrixTransform;

NameScope.SetNameScope(this, new NameScope());

RectangleGeometry rc = TrainPath.Data as RectangleGeometry;

EllipseGeometry e = subway.GetRailways.ElementAt(lineNumber).GetStations.ElementAt(id).ellipsePath.Data as EllipseGeometry;

Point c = e.Center;

rc.Rect = new Rect(c.X, c.Y, 30, 30);

this.RegisterName("ButtonMatrixTransform", buttonMatrixTransform);

PathGeometry animationPath = new PathGeometry();

if (Back)

{

animationPath = subway.GetRailways.ElementAt(lineNumber).GetStations.ElementAt(id).endLinePath.Data as PathGeometry;

}

else

{

if (subway.GetRailways.ElementAt(lineNumber).GetStations.ElementAt(id).startLinePath != null)

{

animationPath = subway.GetRailways.ElementAt(lineNumber).GetStations.ElementAt(id).startLinePath.Data as PathGeometry;

}

}

Point p = subway.GetRailways.ElementAt(lineNumber).GetStations.ElementAt(id).Coordinate;

rc.Rect = new Rect(rc.Rect.X - p.X - 15, rc.Rect.Y - p.Y - 15, 30, 30);

MatrixAnimationUsingPath matrixAnimation =

new MatrixAnimationUsingPath();

matrixAnimation.PathGeometry = animationPath;

matrixAnimation.Duration = TimeSpan.FromSeconds(travelTime);

matrixAnimation.DoesRotateWithTangent = true;

Storyboard.SetTargetName(matrixAnimation, "ButtonMatrixTransform");

Storyboard.SetTargetProperty(matrixAnimation,

new PropertyPath(MatrixTransform.MatrixProperty));

pathAnimationStoryboard.Children.Add(matrixAnimation);

pathAnimationStoryboard.Begin(this);

}

public Path TrainPath

{

set { trainPath = value; }

get { return trainPath; }

}

public int LineNumber

{

set { lineNumber = value; }

get { return lineNumber; }

}

public TimeSpan DepartureTime

{

set { departureTime = value; }

get { return departureTime; }

}

public Train(int i, int j)

{

lineNumber = i;

Number = j;

Back = false;

}

public Train() { }

}

}

Класс Board

using System;

namespace CW\_Underground

{

class Board

{

private BoardWindow bWin;

private TimeSpan \_currentTime;

private int hour, min, sec, rhour = 0, rmin = 0, rsec = 0, lhour = 0, lmin = 0, lsec = 0;

public int StationNumber { get; set; }

public string StationName { get; set; }

private bool leftTimer = false;

private bool rightTimer = false;

private System.Windows.Threading.DispatcherTimer timer = new System.Windows.Threading.DispatcherTimer();

public void Start(int h, int m, int s)

{

timer.Stop();

timer.Tick -= new EventHandler(tick);

timer.Tick += new EventHandler(tick);

timer.Interval = new TimeSpan(0, 0, 0, 1);

hour = h;

min = m;

sec = s;

timer.Start();

}

public void Stop() {

rhour = 0; rmin = 0; rsec = 0;

lhour = 0; lmin = 0; lsec = 0;

leftTimer = false;

rightTimer = false;

bWin.lastTrainRightLB.Content = "nope";

bWin.lastTrainLeftLB.Content = "nope";

timer.Stop();

timer.Tick -= new EventHandler(tick);

}

private void AddTick(ref int hour, ref int min, ref int sec)//add one second

{

sec++;

if (sec > 59) { sec = 0; min++; }

if (min > 59) { min = 0; hour++; }

if (hour > 23) { hour = 0; }

CurrentTime = new TimeSpan(hour, min, sec);

}

private void tick(object sender, EventArgs e)

{

AddTick(ref hour, ref min, ref sec);

if (bWin != null && bWin.IsLoaded)

{

bWin.currTimeLB.Content = CurrentTime;

bWin.NameLB.Content = StationName;

}

if (leftTimer)

{

AddTick(ref lhour, ref lmin, ref lsec);

bWin.lastTrainLeftLB.Content = new TimeSpan(lhour, lmin, lsec);

}

if (rightTimer)

{

AddTick(ref rhour, ref rmin, ref rsec);

bWin.lastTrainRightLB.Content =" " + new TimeSpan(rhour, rmin, rsec);

}

}

public void TrainOnStation(int num, bool back)

{

string str = "Train № " + num + " On station now!";

if (bWin != null && bWin.IsLoaded)

{

if (back)

{

rightTimer = false;

bWin.lastTrainRightLB.Content = str;

}

else

{

leftTimer = false;

bWin.lastTrainLeftLB.Content = str;

}

}

}

public void TrainLeft(bool back)

{

if (back)

{

rhour = 0; rmin = 0; rsec = 0;

rightTimer = true;

}

else

{

lhour = 0; lmin = 0; lsec = 0;

leftTimer = true;

}

}

public TimeSpan CurrentTime

{

get { return \_currentTime; }

set { \_currentTime = value; }

}

public BoardWindow BoardWindow

{

get { return bWin; }

set { bWin = value; }

}

public Board(int num, string n)

{

bWin = new BoardWindow();

StationNumber = num;

StationName = n;

}

}

}