

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий  
Кафедра Информационных систем и технологий  
Специальность 1-40 01 01 «Программное обеспечение информационных технологий»  
Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированное программирование»  
Тема «Разработка симулятора диспетчера руления»

Исполнитель  
студент (ка) 2 курса группы 8 Иванов И.А.  
(Ф.И.О.)

Руководитель  
доцент, канд. техн. наук Пацей Н.В.  
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой \_\_\_\_\_  
Руководитель Пацей Н.В.  
(подпись)

Минск 2017

## **Реферат**

Пояснительная записка курсового проекта содержит 25 страниц, 23 рисунка, 7 источников литературы, 4 приложения.

СИМУЛЯТОР, АНИМАЦИЯ, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС, C#, ООП, WPF, VISUAL STUDIO.

Основными целями курсового проекта являются: разработка симулятора диспетчера руления.

Пояснительная записка состоит из введения, трёх разделов, заключения.

Во введении представлена общая информация, дающая представление о предстоящей работе, определены цели.

В первом разделе рассматривается архитектура приложения и структура XML-файла с расписанием.

Во втором разделе представлен процесс и результаты разработки итогового приложения.

В третьем разделе представлено руководство пользователя.

В заключении представлены итоги курсового проектирования и задачи, которые были решены в ходе разработки приложения.

## Содержание

Введение .....	4
1 Архитектура приложения.....	5
2 Разработка приложения.....	7
2.1 Создание изображений.....	7
2.2 Разработка GUI.....	8
2.3 Реализация класса Airport .....	9
2.4 Перечисления .....	10
2.5 Реализация класса Flight.....	11
2.6 Реализация класса FlightTimetable .....	11
2.7 Реализация анимирования.....	12
2.8 Реализация класса-обёртки для DateTime .....	13
2.9 Реализация формы добавления рейса .....	13
2.10 Реализация формы поиска, изменения, удаления рейсов .....	14
2.11 Реализация класса Logger.....	14
3 Руководство пользователя .....	15
3.1 Вид приложения.....	15
3.2 Добавление рейса .....	16
3.3 Форма поиска и изменения рейса.....	17
3.4 Задержка рейса .....	18
3.5 Изменение рабочей полосы .....	18
3.6 Изменение названия аэропорта, позиций УВД.....	18
3.7 Изменение количества строк в таблицах прилёта/вылета.....	19
3.8 Сохранение расписания.....	19
3.9 Загрузка расписания .....	19
3.10 Система логгирования.....	19
3.11 Просмотр информации о выполняемом рейсе.....	20
3.12 Выполнение рейса.....	20
Заключение .....	22
Список использованных источников.....	23
ПРИЛОЖЕНИЕ А. Листинг функции, управляющей анимированием .....	24
ПРИЛОЖЕНИЕ Б. Фрагмент листинга класса Flight .....	25
ПРИЛОЖЕНИЕ В. Листинг класса DateValue .....	26
ПРИЛОЖЕНИЕ Г. Листинг функции анимирования посадки .....	29

## **Введение**

На сегодняшний день мощность вычислительной техники позволяет разрабатывать и использовать программы, требовательные к ресурсам. Симуляторы требовательны к ресурсам вычислительной техники. Под симулятором понимают устройства и/или программы, которые имитируют управление каким-либо процессом, аппаратом или транспортным средством. Существуют компьютерно-механические симуляторы, абсолютно точно воспроизводящие кабину пилотов реального самолёта. Это позволяет симулировать аварийную ситуацию с целью обучения пилота правильным действиям, и при этом не подвергать угрозе жизни людей.

Проблема безопасности выполнения полётов является актуальной на сегодняшний день. Диспетчеры, осуществляющие управление воздушным движением, должны корректно выполнять отведенные им обязанности.

Главной целью данного курсового проекта является разработка симулятора диспетчера руления. Для достижения поставленной цели необходимо:

### **1. Разработать:**

- визуальное представление аэропорта;
- визуальное представление трёх типов летательных аппаратов (ЛА);
- пути для анимирования ЛА (посадка, руление, буксировка, взлёт);
- GUI, позволяющий осуществлять управление ЛА.

### **2. Реализовать:**

- расписание рейсов;
- возможность сохранять/загружать расписание;
- возможность задерживать рейсы;
- возможность изменять рабочую полосу;
- систему логгирования с возможностью сохранения в файл.

# 1 Архитектура приложения

Архитектура приложения представлена на рисунке 1.1.



Рисунок 1.1 – Архитектура приложения

Хранение расписания рейсов осуществляется при помощи XML. Структура XML-файла следующая: корневым элементом является элемент FLIGHTS, который содержит внутри себя элементы FLIGHT, каждый из которых описывает один рейс.

Элемент FLIGHT содержит следующие элементы:

1. DAYS – дни выполнения рейса;
2. CATEGORY – тип ЛА, на котором выполняется данный рейс;
3. ARRIVAL\_TIME – время прилёта;
4. DEPARTURE\_TIME – время вылета;
5. FROM – город или аэропорт, откуда выполняется рейс;
6. DEST – город или аэропорт, куда выполняется рейс;
7. OPERATOR – эксплуатант, т.е. авиакомпания, которой принадлежит ЛА, выполняющий данный рейс;
8. ARRIVAL\_FN – номер рейса при прилёте;
9. DEPARTURE\_FN – номер рейса при вылете.

Пример сохранённого расписания представлен на рисунке 1.2.

```
<?xml version="1.0" encoding="utf-8"?>
<FLIGHTS>
  <FLIGHT>
    <DAYS>Вс,Пн,Вт,Ср,Чт,Пт,Сб</DAYS>
    <CATEGORY>Medium</CATEGORY>
    <ARRIVAL_TIME>12:30:00</ARRIVAL_TIME>
    <DEPARTURE_TIME>13:45:00</DEPARTURE_TIME>
    <FROM>OMAA</FROM>
    <DEST>OMAA</DEST>
    <OPERATOR>Etihad</OPERATOR>
    <ARRIVAL_FN>ETD61</ARRIVAL_FN>
    <DEPARTURE_FN>ETD62</DEPARTURE_FN>
  </FLIGHT>
  <FLIGHT>
    <DAYS>Вс,Пн,Ср,Пт</DAYS>
    <CATEGORY>Heavy</CATEGORY>
    <ARRIVAL_TIME>17:20:00</ARRIVAL_TIME>
    <DEPARTURE_TIME>18:50:00</DEPARTURE_TIME>
    <FROM>ZBAA</FROM>
    <DEST>LROP</DEST>
    <OPERATOR>Air China</OPERATOR>
    <ARRIVAL_FN>CCA721</ARRIVAL_FN>
    <DEPARTURE_FN>CCA722</DEPARTURE_FN>
  </FLIGHT>
</FLIGHTS>
```

Рисунок 1.2 – Сохранённое расписание

## 2 Разработка приложения

Приложение будет реализовано на языке C# с использованием Windows Presentation Foundation в среде разработки Visual Studio 2015. Для создания изображений аэропорта, подложки (травы) и ЛА используется Adobe Photoshop Creative Cloud 2015.

### 2.1 Создание изображений

Приложение будет включать три самолёта разных типов: Heavy (тяжёлый), Medium (средний), Light (лёгкий) в соответствии с категориями турбулентности ICAO [1]. Изображение самолётов, которое будет использовано в программе, представлено на рисунке 2.1:



Рисунок 2.1 – Самолёты

Изображение аэропорта содержит полосу, рулёжные дорожки, стоянки, терминал. Аэропорт включает 8 стоянок: с 1 по 3 включительно для самолётов типа Light, с 4 по 8 включительно для самолётов типа Medium и Heavy. При помощи отдельных текстур создается композиция, представляющая изображение аэропорта. Готовое изображение аэропорта представлено на рисунке 2.2:

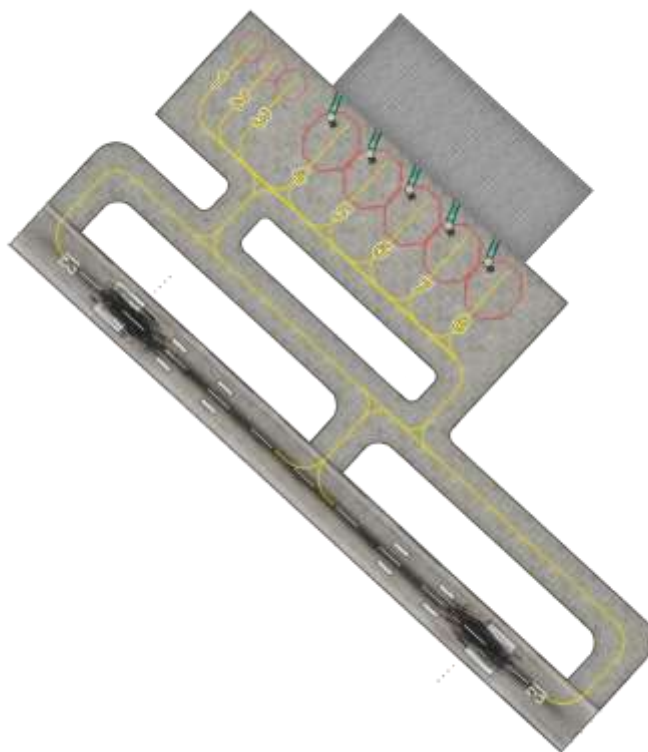


Рисунок 2.2 – Аэропорт

## 2.2 Разработка GUI

Для создания пользовательского интерфейса используется Windows Presentation Foundation. В левой части окна программы расположен аэропорт, в правой – таблицы вылетов/прилётов. В нижней части окна расположен логгер. На рисунке 2.3 представлен внешний вид приложения.



Рисунок 2.3 – Общий вид приложения

Левая часть GUI представлена элементом Canvas, на котором расположено изображение аэропорта. В качестве фона выступает повторяющееся изображение травы.

В правой части GUI сверху расположена таблица прилётов, снизу – вылетов. Данные таблицы выполнены при помощи элемента DataGridView. В заголовках таблиц используется шрифт Arial, а в ячейках – шрифт LCD Phone. Каждая таблица содержит 4 колонки:

1. Time (время прилёта/вылета соответственно);
2. From/To (аэропорт (город) прилёта/вылета соответственно). На рисунке 2.3 использованы ICAO коды аэропортов [2];
3. Flight (номер рейса);
4. Status (статус рейса).

В левой верхней части каждой таблицы расположено поле, в котором отображено, сколько всего рейсов имеется в очереди на прилёт/вылет. На рисунке 2.3 данное поле в обеих таблицах имеет значение «Всего 13».



В правой верхней части таблицы «Прилёты» расположено поле, значением которого является день недели и время. На рисунке 2.3 значением данного поля является «Пн, 00:26».

Аэропорт отделен от табло вылетов/прилётов при помощи элемента GridSplitter. Табло вылета/прилёта так же отделены друг от друга аналогичным элементом. Логгер отделен от остальной части UI при помощи GridSplitter.

В верхней части GUI расположено меню со следующей иерархией:

1. Файл:
  - 1.1. Загрузить расписание;
  - 1.2. Сохранить расписание;
  - 1.3. Сохранить лог;
  - 1.4. Выход.
2. Редактирование:
  - 2.1. Названия – редактирование названия аэропорта, наименования текущей позиции управления воздушным движением (УВД) и вышестоящей позиции УВД;
  - 2.2. Количество строк – изменение количества отображаемых строк в таблице прилётов/вылетов;
  - 2.3. Добавить рейс;
  - 2.4. Задержать рейс;
  - 2.5. Поиск и изменение рейсов – так же позволяет удалить рейс из расписания;
  - 2.6. Изменить рабочую полосу.
3. О программе.

## 2.3 Реализация класса Airport

Класс Airport представляет собой главный класс, реализованный с использованием паттерна Singleton. Данный класс содержит:

- название аэропорта;
- наименование позиции управления воздушным движением (УВД);
- наименование вышестоящей позиции УВД;
- данные для таблиц прилёта/вылета;
- очередь прилётов;
- очередь руления на стоянку;
- очередь вылетов (буксировки);
- очередь руления на предварительный старт (позиция у торца взлётно-посадочной полосы);
- очередь взлётов;
- булевы поля, отображающие, свободен ли перрон, разрешен ли взлёт/посадка, разрешено ли руление по перрону;
- список стоянок.

В классе Airport расположена функция, которая управляет запуском анимации. Листинг данной функции представлен в приложении А.

## 2.4 Перечисления

В программе определены перечисления `AirplanesCategory`, `FlightStatus` и `PlaneActions`. Данные перечисления представлены на рисунке 2.4.

```
public enum AirplanesCategory // категория самолета
{
    H, // Heavy
    M, // Medium
    L  // Light
}

public enum FlightStatus // статус выполнения рейса
{
    Error,
    OnTime, // вовремя
    Landed, // совершил посадку
    CheckIn, // регистрация
    Boarding, // загрузка в самолет
    Departured, // вылетел
    Delayed, // отложен
    Cancelled // отменен
}

public enum PlaneActions // действия, которые выполняет рейс
{
    InFlight, // в полёте
    Landing, // посадка
    TaxiToStand, // руление на стоянку
    OnStand, // на стоянке
    Pushback, // буксировка
    TaxiToHr, // руление на предварительный
    Takeoff // взлёт
}
```

Рисунок 2.4 – Перечисления

В перечислении `AirplanesCategory` определены категории летательных аппаратов. Категории задаются при добавлении рейса (подраздел 2.9) и могут быть изменены в форме изменения рейса (подраздел 2.10).

В перечислении `FlightStatus` определены статусы выполнения рейса, которые изменяются во время анимирования. Так, например, после освобождения полосы и выдачи разрешения руления на стоянку, статус рейса в таблице прилётов меняется с `OnTime` (согласно расписанию) на `Landed` (совершил посадку). В момент завершения руления на стоянку, статус рейса в таблице вылетов изменяется с `CheckIn` (регистрация) на `Boarding` (загрузка пассажиров в самолёт).

В перечислении `PlaneActions` определены действия, которые может выполнять летательный аппарат. По умолчанию имеет значение `InFlight` (в полёте). Данное поле содержится в объектах типа `Flight`, изменяется при анимировании. Так, например, при начале анимирования посадки значение изменяется с `InFlight` на `Landing` (посадка).

## 2.5 Реализация класса **Flight**

Класс **Flight** представляет собой рейс. Данный класс реализует интерфейс **ICloneable**.

В классе содержатся следующие статические поля:

1. **DateValue MinimumTimeOnGround** – минимальное время нахождения ЛА на стоянке. По умолчанию имеет значение 50 минут;
2. **DateValue DelayValue** – время, по прошествии которого рейс помечается как «Delayed» (задержан). По умолчанию имеет значение 1 час;
3. **Dictionary<AirplanesCategory, string> Categories**, представляющий собой ключ типа **AirplanesCategory** (подраздел 2.4) и значение типа **string** (строковое представление ключа);
4. **Dictionary<FlightStatus, string> Status**, представляющий собой ключ типа **FlightStatus** (подраздел 2.4) и значение типа **string**, являющееся строковым представлением ключа.

Класс **Flight** содержит 16 **public**-полей. Фрагмент листинга, в котором представлены данные поля, представлен в приложении Б.

Стоит обратить внимание на поля **DateValue DepartureTimetable** (время вылета) и **DateValue DepartureTime** (реальное время вылета). Так, **DepartureTimetable** является значением времени вылета, которое отображается в таблице вылета, округлённое с точностью до 5 минут. **DepartureTime** генерируется при помощи добавления случайного числа из интервала [0; 30] к значению минут в **DepartureTimetable**. Генерация **DepartureTime** происходит после остановки летательного аппарата на стоянке. Значение данного поля означает реальное время вылета, при наступлении которого рейс запросит буксировку со стоянки.

## 2.6 Реализация класса **FlightTimetable**

Класс **FlightTimetable** представляет собой **static** класс, который хранит расписание и содержит методы для работы с расписанием. Расписание представлено словарём, в котором ключом выступает тип **string**, значением – тип **Flight**. Ключ формируется при помощи конкатенации двух строк: номера рейса при прилёте и номера рейса при вылете.

Данный класс содержит следующие функции:

1. **bool Add (Flight f)** – добавляет рейс **f** в расписание. В случае успешного добавления рейса возвращает **true**;
2. **bool ChangeFlight (Flight f1, Flight f2)** – заменяет рейс **f1** на рейс **f2**. Возвращает **true** в случае успешного добавления рейса. Если рейс **f1** отсутствует, то рейс **f2** всё равно будет добавлен;
3. **bool Delete (Flight f)** – удаляет рейс **f**. Возвращает **true** в случае успешного удаления рейса из расписания;
4. **IEnumerable<Flight> GetCurrentDayTimetable ()** – возвращает все рейсы на данный день. Данная функция вызывается при загрузке расписания и каждый раз при наступлении полуночи (по времени приложения).

## 2.7 Реализация анимирования

Анимирование выполняется в классе Main, содержащем Canvas и UI (подраздел 2.2). Для анимирования предназначены следующие поля:

1. Storyboard storyboard – временная шкала;
2. Dictionary<string, PathGeometry> paths – пути для анимирования. В качестве ключа выступает строка, являющаяся однозначным идентификатором пути, а в качестве значения – путь анимирования, который строится по точкам, заданным в файле XAML. Пример инициализации переменной paths приведен на рисунке 2.5.

```
paths = new Dictionary<string, System.Windows.Media.PathGeometry>
{
    { "LightLandingR13", LightLandingR13Path.Data.GetFlattenedPathGeometry() },
    { "LandingR13", LandingR13Path.Data.GetFlattenedPathGeometry() },
    { "LightLandingR31", LightLandingR31Path.Data.GetFlattenedPathGeometry() },
    { "LandingR31", LandingR31Path.Data.GetFlattenedPathGeometry() }
};
```

Рисунок 2.5 – Пример инициализации переменной paths

3. Dictionary<string, int> imageOffsetValues – смещения для ЛА по типам (для рисунка 2.1). В зависимости от типа ЛА необходимо вырезать определенную часть данного рисунка. Инициализация данной переменной представлена на рисунке 2.6.

```
imageOffsetValues = new Dictionary<string, int>
{
    { "Heavy", 0},
    { "Medium", 80},
    { "Light", 160}
};
```

Рисунок 2.6 – Инициализация переменной imageOffsetValues

4. Dictionary<int, UIElement> planesDictionary – хранит картинки добавленных самолётов (используется при анимировании). В качестве ключа используется GetHashCode рейса, а в качестве значения – изображение самолёта;
5. Dictionary<string, int> animationDurations – время анимирования. В качестве ключа выступает строка, однозначно идентифицирующая путь анимирования, а в качестве значения – целочисленное значение, которое является временем анимирования и измеряется в секундах. Пример инициализации переменной animationDurations представлен на рисунке 2.7.

```
animationDurations = new Dictionary<string, int>
{
    { "HeavyLanding", 5 },
    { "MediumLanding", 5 },
    { "LightLanding", 5 }
}
```

Рисунок 2.7 – Пример инициализации переменной animationDurations

Листинга функции анимирования посадки представлен в приложении Г.

## 2.8 Реализация класса-обёртки для DateTime

Класс DateValue является обёрткой для DateTime, а так же реализует интерфейс IComparable. Данный класс предоставляет пользователю день недели, часы и минуты.

В классе перегружены операции сравнения (==, !=, <, >, <=, >=), операции сложения, вычитания. Реализованы функции для добавления часов или минут к текущему значению времени. Реализована функция bool Between(DateValue t1, DateValue t2), которая возвращает true, если текущее значение времени находится в интервале [t1; t2]. Листинг класса DateValue представлен в приложении В.

## 2.9 Реализация формы добавления рейса

Для добавления рейса в расписание в программе предусмотрено диалоговое окно, реализованное при помощи Windows Forms. Его внешний вид представлен на рисунке 2.8.

Добавить рейс

Номер рейса (прилет): AFL1830

Номер рейса (вылет): AFL1831

Прилет из: UUEE

Отправление в: UUEE

Эксплуатант: Aeroflot

Категория: Medium

Время прилета: 10:00

Время вылета: 10:55

Дни выполнения рейса:

- ☒ Вс
- ☒ Пн
- ☒ Вт
- ☒ Ср
- ☒ Чт
- ☒ Пт
- ☒ Сб

Добавить

Рисунок 2.8 – Форма добавления рейса

Все поля на форме обязательны для заполнения. Выпадающий список «Категория» имеет значения Heavy (по умолчанию), Medium, Light.

В группе флажков «Дни выполнения рейсов» должен быть отмечен как минимум один checkbox.

При добавлении рейса происходит проверка полей на недопустимые значения. Недопустимо добавление рейса, если рейс с таким номером уже существует. Для изменения рейса необходимо воспользоваться формой изменения рейса (подраздел 2.10). Так же проверяется значение времени прилёта и времени вылета (прилёт должен быть раньше), между прилётом и вылетом должно пройти минимум 50 минут (подраздел 2.5 – MinimumTimeOnGround – минимальное время нахождения на стоянке).

## 2.10 Реализация формы поиска, изменения, удаления рейсов

Форма для поиска, изменения, удаления рейсов реализована при помощи Windows Forms. Внешний вид данной формы представлен на рисунке 2.9.

Эксплуатант	№рейса (прилет)	№рейса (вылет)	Из	В	Категория	День выполнения	Время(при)	Время(выл)	Изменить	Удалить
Etihad	ETD61	ETD62	OMAA	OMAA	Medium	Вс.Пн.Вт...	12:30:00	13:45:00	Изменить	Удалить
Vueling	VLG7692	VLG7693	LEBZ	LEBZ	Medium	Вс.Чт	05:10:00	06:30:00	Изменить	Удалить
Azerbaijan	AHJ8701	AHJ8702	UBBB	UBBB	Medium	Вс.Вт	10:10:00	11:40:00	Изменить	Удалить
LOT	LOT706	LOT705	EPWA	EPWA	Medium	Вс.Вт.Чт	14:05:00	14:55:00	Изменить	Удалить
LOT	LOT707	LOT708	EPWA	EPWA	Medium	Пн.Пт	17:50:00	19:20:00	Изменить	Удалить
Austrian	AUA687	AUA688	LOWW	LOWW	Medium	Вс.Пн.Вт...	12:40:00	14:05:00	Изменить	Удалить
Motor Sic	MSI317	MSI318	UKDE	UKDE	Medium	Вс.Пн.Ср...	11:15:00	15:00:00	Изменить	Удалить
Ukraine I	AUJ891	AUJ892	UKBB	UKBB	Medium	Вс.Вт.Сб	10:25:00	11:15:00	Изменить	Удалить
Aeroflot	AFL1830	AFL1831	UUEE	UUEE	Medium	Вс.Пн.Вт...	10:00:00	10:55:00	Изменить	Удалить
UTair	UTA835	UTA836	UUWW	UUWW	Medium	Вс.Пн.Вт...	11:20:00	13:20:00	Изменить	Удалить
UTair	UTA775	UTA776	UUWW	UUWW	Medium	Вс.Пн.Вт...	21:20:00	23:20:00	Изменить	Удалить
Air China	CCA721	CCA722	ZBAA	LROP	Heavy	Вс.Пн.Ср...	17:20:00	18:50:00	Изменить	Удалить
Turkish A	THY283	THY284	LTBA	LTBA	Medium	Пн.Ср.Чт...	16:15:00	17:10:00	Изменить	Удалить
Uzbekist	ANY709	ANY710	UTTT	UTTT	Medium	Вт.Чт	11:10:00	12:40:00	Изменить	Удалить
Arkia Isra	AIZ317	AIZ318	LLBG	LLBG	Medium	Пн.Ср.Пт	18:15:00	19:35:00	Изменить	Удалить
Lufthansa	DLH1486	DLH1487	EDDF	EDDF	Medium	Пн.Ср.Пт...	13:40:00	14:35:00	Изменить	Удалить
Belavia	BRU993	BRU950	UDDD	UMKK	Medium	Вс.Пн.Вт...	19:15:00	20:50:00	Изменить	Удалить
Belavia	BRU946	BRU735	ULLI	UGTB	Medium	Вс.Пн.Вт...	21:10:00	22:50:00	Изменить	Удалить

Рисунок 2.9 – Форма поиска, изменения, удаления рейсов

В левой части данной формы расположены поля, позволяющие установить критерии для поиска (на рисунке 2.9 заполнены значениями по умолчанию). В правой части расположен DataGridView, в который выводятся результаты поиска. Каждый рейс, отображаемый в результате поиска, имеет собственные кнопки «Изменить» и «Удалить».

## 2.11 Реализация класса Logger

Класс Logger представляет собой класс, реализующий интерфейс INotifyPropertyChanged. Данный класс содержит:

1. Очередь сообщений private readonly Queue<string> messages;
2. Свойство private int Limiter { get; } = 50, значением которого является количество сообщений, которое может содержать очередь сообщений;
3. Свойство public string Messages, для которого результатом выполнения блока get является строковое представление очереди сообщений (пункт 1);
4. Функцию public bool Add (string s), предназначенную для добавления строки s в очередь сообщений. Возвращает true в случае успешного добавления сообщения в очередь. Так же в случае добавления сообщения в очередь, будет вызван метод protected virtual void OnPropertyChanged([CallerMemberName] string propertyName);
5. Метод public async void Save(), предназначенный для сохранения очереди сообщений в файл.



## 3 Руководство пользователя

### 3.1 Вид приложения

Внешний вид приложения представлен на рисунке 3.1.



Рисунок 3.1 – Внешний вид приложения

В левой части приложения находится аэропорт, в правой – расписание рейсов, внизу – логгер.

Аэропорт содержит одну взлётно-посадочную полосу, рулёжные дорожки и 8 пронумерованных стоянок. Каждая стоянка способна принимать лишь свой тип летательных аппаратов (ЛА).

Верхняя часть расписания – это таблица прилётов, нижняя – таблица вылетов. Каждая из этих таблиц разбита на 4 колонки:

1. Time – время прилёта/вылета соответственно;
2. From/To – откуда прилёт/куда вылет;
3. Flight – номер рейса;
4. Status – статус выполнения рейса.

В логгер выводится информация, относящаяся к выполнению рейсов, изменения в настройках программы. Содержимое логгера может быть сохранено. Данная возможность будет рассмотрена в следующих разделах.

В верхней части приложения расположено горизонтальное меню, которое позволяет выполнять различные действия, которые будут рассмотрены в следующих разделах.

### 3.2 Добавление рейса

Для добавления рейса необходимо выбрать пункт меню Редактирование -> Добавить рейс. В результате данного действия будет открыто диалоговое окно, внешний вид которого представлен на рисунке 3.2.

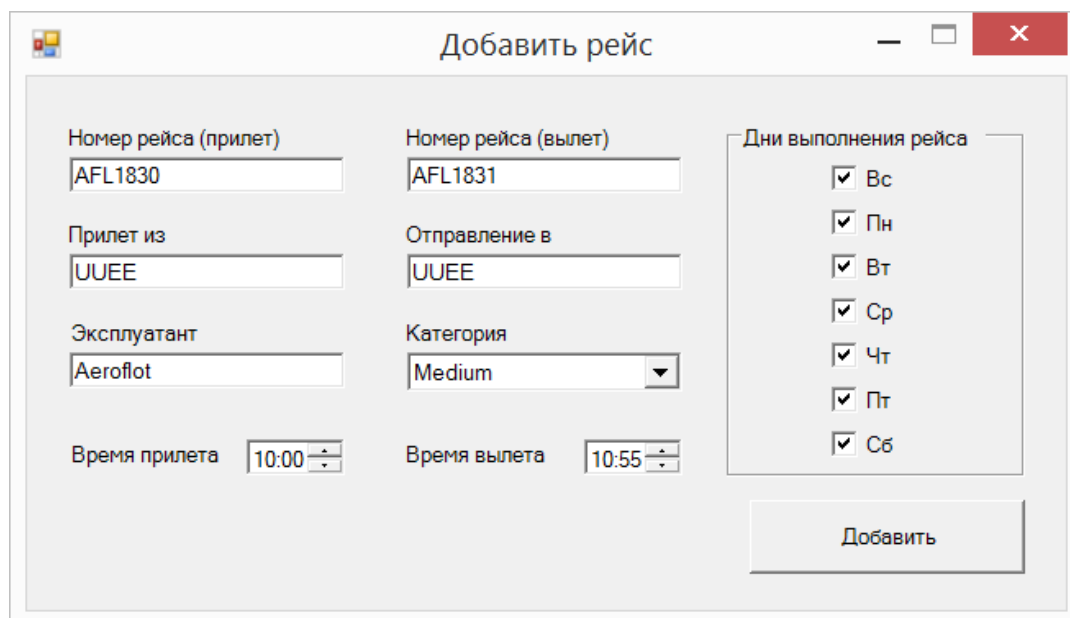


Рисунок 3.2 – Форма добавления рейса

Введённые в данную форму данные будут отображаться в таблицах прилёта/вылета. На форме расположены следующие элементы:

1. Номер рейса (прилёт);
2. Номер рейса (вылет);
3. Прилёт из – аэропорт либо город, из которого выполняется рейс;
4. Отправление в – аэропорт либо город, из которого выполняется рейс;
5. Эксплуатант – авиакомпания, которой принадлежит самолёт, который выполняет данный рейс;
6. Категория. В выпадающем меню есть три элемента: Heavy, Medium, Light, т.е. Тяжёлый, Средний, Лёгкий соответственно. При помощи данного выбранного значения задаётся тип самолёта, который будет выполнять рейс. Внешний вид самолётов представлен на рисунке 3.3;
7. Время прилёта;
8. Время вылета. Между прилётом и вылетом должно пройти как минимум 50 минут;
9. Дни выполнения рейса. Набор из 7 флажков, для успешного добавления рейса должен быть выбран как минимум один флажок.



Рисунок 3.3 – Самолёты



### 3.3 Форма поиска и изменения рейса

Для выполнения поиска или изменения рейса необходимо выбрать пункт меню Редактирование -> Поиск и изменение рейсов. Внешний вид формы, которая будет открыта после данного действия, представлен на рисунке 3.4.

Поиск и изменение рейсов

Обязательные поля оставить не тронутыми  
\*регистрация

Эксплуатант:   
Номер рейса:   
Из:   
В:   
Категория:   
День выполнения:   
Прилет в период: с  по   
Вылет в период: с  по   
Найти Сбросить

Эксплуатант	№рейса (прилет)	№рейса (вылет)	ИЗ	В	Категория	Дни выполнения	Время(при)	Время(выл)		
Ethad	ETD61	ETD62	OMAA	OMAA	Medium	Вс,Пн,Вт...	12:30:00	13:45:00	Изменить	Удалить
Vueling	VLG7692	VLG7693	LEBZ	LEBZ	Medium	Вс,Чт	05:10:00	06:30:00	Изменить	Удалить
Azerbaijan A...	AHU8701	AHU8702	UBBB	UBBB	Medium	Вс,Вт	10:10:00	11:40:00	Изменить	Удалить
LOT	LOT706	LOT705	EPWA	EPWA	Medium	Вс,Вт,Чт	14:05:00	14:55:00	Изменить	Удалить
LOT	LOT707	LOT708	EPWA	EPWA	Medium	Пн,Пт	17:50:00	19:20:00	Изменить	Удалить
Austrian Airli...	AUA687	AUA688	LOW	LOWW	Medium	Вс,Пн,Вт...	12:40:00	14:05:00	Изменить	Удалить
Motor Sich A...	MSI317	MSI318	UKDE	UKDE	Medium	Вс,Пн,Ср...	11:15:00	15:00:00	Изменить	Удалить
Ukraine Inter...	AUI891	AUI892	UKBB	UKBB	Medium	Вс,Вт,Сб	10:25:00	11:15:00	Изменить	Удалить
Aeroflot	AFL1830	AFL1831	UUUE	UUUE	Medium	Вс,Пн,Вт...	10:00:00	10:55:00	Изменить	Удалить
UTair	UTA835	UTA836	UU	UUWW	Medium	Вс,Пн,Вт...	11:20:00	13:20:00	Изменить	Удалить
UTair	UTA775	UTA776	UU	UUWW	Medium	Вс,Пн,Вт...	21:20:00	23:20:00	Изменить	Удалить
Air China	CCA721	CCA722	ZBAA	LROP	Heavy	Вс,Пн,Ср...	17:20:00	18:50:00	Изменить	Удалить
Turkish Airlin...	THY283	THY284	LTBA	LTBA	Medium	Пн,Ср,Чт...	16:15:00	17:10:00	Изменить	Удалить
Uzbekistan ...	AHY709	AHY710	UTTT	UTTT	Medium	Вт,Чт	11:10:00	12:40:00	Изменить	Удалить
Arkia Israel	AZ317	AZ318	LLBG	LLBG	Medium	Пн,Ср,Пт	18:15:00	19:35:00	Изменить	Удалить
Lufthansa	DHL1486	DHL1487	EDDF	EDDF	Medium	Пн,Ср,Пт...	13:40:00	14:35:00	Изменить	Удалить
Bolavia	BRU993	BRU950	UDDO	UMKK	Medium	Вс,Пн,Вт...	19:15:00	20:50:00	Изменить	Удалить
Bolavia	BRU946	BRU735	ULLI	UGTB	Medium	Вс,Пн,Вт...	21:10:00	22:50:00	Изменить	Удалить

Рисунок 3.4 – Форма поиска и изменения рейсов

В левой части формы представлены поля, значения которых служит критерием поиска. Поиск может производиться по эксплуатанту, номеру рейса, аэропорту (или городу) прибытия, аэропорту (или городу) отправления, категории, дням выполнения, времени прилёта и времени вылета.

При нажатии на кнопку «Найти» результаты поиска будут выведены в таблицу в правой части окна. Строка представляет собой рейс. У каждого рейса есть кнопки «Изменить» и «Удалить». При нажатии на кнопку «Изменить» будет открыто диалоговое окно, представленное на рисунке 3.5.

Редактирование

Номер рейса (прилет):   
Номер рейса (вылет):   
Прилет из:   
Отправление в:   
Эксплуатант:   
Категория:   
Время прилета:   
Время вылета:   
Дни выполнения рейса:  
☒ Вс  
☐ Пн  
☒ Вт  
☐ Ср  
☐ Чт  
☐ Пт  
☐ Сб  
Изменить

Рисунок 3.5 – Форма изменения рейса

Как видно из рисунка 3.5, форма изменения рейса заполнена в соответствии с выбранным рейсом. Для удаления рейса необходимо нажать кнопку «Удалить» на соответствующей строке таблицы.

### 3.4 Задержка рейса

Для задержки рейса на некоторый промежуток времени необходимо выбрать пункт меню Редактирование -> Задержать рейс. Если на стоянке нет ни одного самолёта, то будет выдана ошибка. Иначе будет открыта форма, представленная на рисунке 3.6.

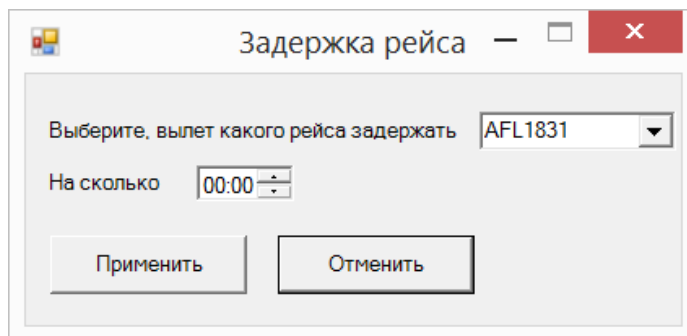


Рисунок 3.6 – Форма задержки рейса

В выпадающем меню, значением которого на рисунке 3.6 является «AFL1831», перечислены все рейсы, находящиеся на стоянках. При нажатии «Применить» для выбранного рейса устанавливается задержка вылета.

### 3.5 Изменение рабочей полосы

Для изменения рабочей полосы необходимо выбрать пункт меню Редактирование -> Изменить рабочую полосу. В случае, если в аэропорту нет перемещающихся воздушных судов, рабочая полоса будет изменена сразу же. Иначе – через некоторое время, когда в аэропорту не будет движения.

### 3.6 Изменение названия аэропорта, позиций УВД

Для изменения названия аэропорта либо позиции УВД необходимо выбрать пункт меню Редактирования -> Названия. При выборе данного пункта будет открыто окно, представленное на рисунке 3.7.

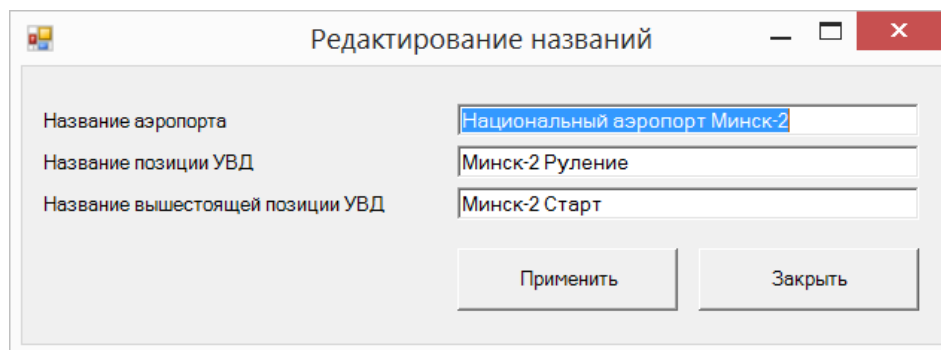


Рисунок 3.7 – Изменение названий

### 3.7 Изменение количества строк в таблицах прилёта/вылета

Для изменения количества отображаемых строк в таблицах прилёта/вылета необходимо выбрать пункт меню Редактирование -> Количество строк. После выбора данного пункта меню будет открыто окно, представленное на рисунке 3.8.

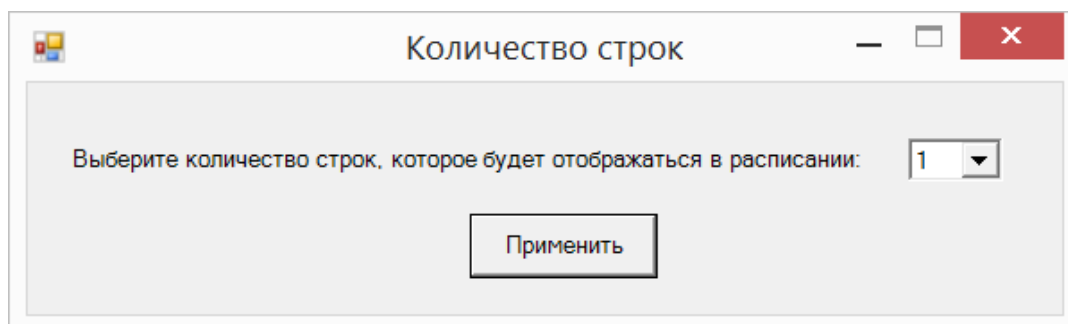


Рисунок 3.8 – Изменение количества строк

После нажатия кнопки «Применить» в таблицах прилёта/вылета будет отображаться количество строк, не превышающее указанного. В таблицах прилёта/вылета слева сверху отображается, сколько всего записей на данный момент в таблице. Пример приведен на рисунке 3.9.

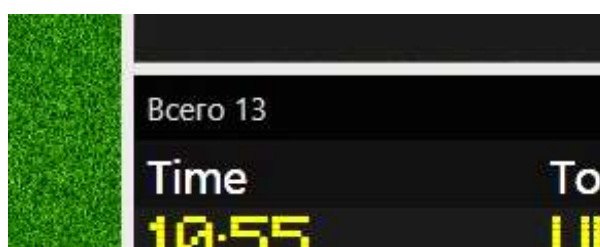


Рисунок 3.9 – Количество строк в таблице

### 3.8 Сохранение расписания

Для сохранения созданного Вами расписания необходимо выбрать пункт меню Файл -> Сохранить расписание, затем выбрать путь для сохранения и нажать кнопку «Сохранить».

### 3.9 Загрузка расписания

Для загрузки расписания необходимо выбрать пункт меню Файл -> Открыть, затем выбрать желаемый XML файл, содержащий расписание. В случае, если файл содержит неправильную структуру или не является расписанием, то будет выдана ошибка.

### 3.10 Система логгирования

Система логгирования регистрирует действия пользователя, а так же события, связанные с рейсами. Вы можете сохранить информацию, находящуюся в логгере, при помощи выбора пункта меню Файл -> Сохранить лог. Пример логгера представлен на рисунке 3.10.

Пн, 00:34: Рабочая полоса изменена на R13  
Пн, 01:00: Рейс AFL1830 выполняет посадку на R13 полосу  
Пн, 01:05: Рейс AFL1830 из UUEE произвёл посадку на R13 полосу  
Пн, 01:06: Рейс AFL1830 рулит на стоянку 4  
Пн, 01:15: Рейс AFL1830 на стоянке  
Пн, 01:20: Рейс AUA687 выполняет посадку на R13 полосу  
Пн, 01:25: Рейс AUA687 из LOWW произвёл посадку на R13 полосу  
Пн, 01:26: Рейс AUA687 рулит на стоянку 5  
Пн, 01:26: Рейс MSI317 выполняет посадку на R13 полосу

Рисунок 3.10 – Логгер

### 3.11 Просмотр информации о выполняемом рейсе

В момент выполнения рейса, т.е. когда рейс находится в аэропорту, Вы можете просмотреть информацию, кликнув по изображению самолёта. При клике на самолёт будет отображено окно (заполненное информацией в соответствии с выбранным рейсом), аналогичное представленному на рисунке 3.11.

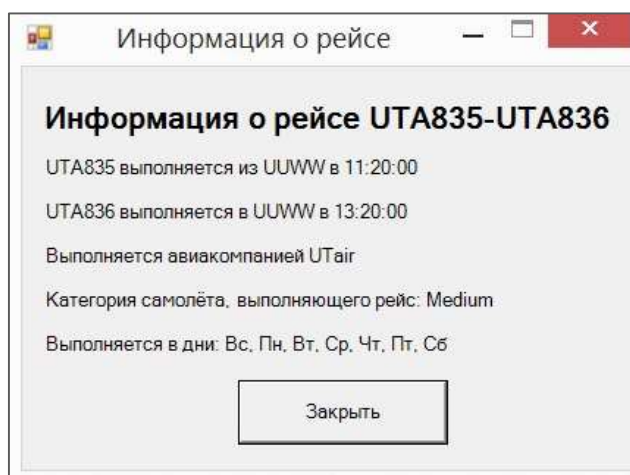


Рисунок 3.11 – Информация о рейсе

### 3.12 Выполнение рейса

Выполнение рейса разделено на следующие пункты:

1. Посадка. В логгер выводится сообщение о выполнении посадки с указанием рейса. После посадки в логгер выводится сообщение об успешном выполнении посадки;

2. Руление на стоянку. После успешной посадки выводится диалоговое окно, предлагающее пользователю выбрать стоянку для ЛА. Пример такого окна приведен на рисунке 3.12.

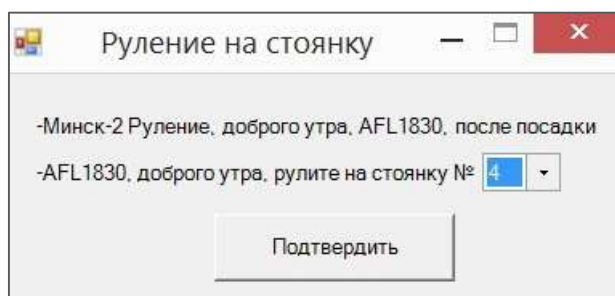


Рисунок 3.12 – Окно «После посадки»

После выбора стоянки в логгер выводится сообщение о рулении. После окончания руления в логгер будет выведено сообщение об этом;

3. Буксировка. В момент наступления времени вылета в логгер выводится сообщение о том, что рейс готов к вылету. Выполняется буксировка ЛА;

4. Руление на предварительный старт. После окончания буксировки выводится сообщение об этом в логгер, а затем выполняется руление на предварительный старт – позицию у торца ВПП, на которой ЛА ждут разрешения на взлёт. В момент занятия предварительного старта происходит передача ЛА вышестоящей позиции УВД;

5. Взлёт.

## Заключение

Был разработан симулятор диспетчера руления, который включает:

1. Визуальное представление аэропорта, которое включает 1 взлётно-посадочную полосу, рулёжные дорожки, 8 стоянок. Каждая стоянка способна принимать только свой тип летательного аппарата;
2. Визуальное представление трёх типов самолётов – Heavy, Medium и Light. В зависимости от заданного в рейсе типа летательного аппарата будет загружено соответствующее изображение;
3. 44 пути для анимирования, а именно: пути для посадок, руления на стоянку, буксировки, руления на предварительный старт, пути для взлёта. В зависимости от рабочей полосы будет выбран корректный путь для анимирования посадки, взлёта;
4. GUI для осуществления управления воздушным движением. GUI включает: аэропорт, таблицы прилёта и вылета, логгер, меню;
5. Класс-обёртка для DateTime, который предоставляет минуты, часы и дни недели;
6. Расписание рейсов. Был разработан отдельный класс, который хранит расписание рейсов. При добавлении рейса и при наступлении полуночи по игровому времени происходит загрузка актуального расписания;
7. Возможность сохранять и загружать готовое расписание. Для хранения готового расписания используется XML;
8. Возможность задерживать рейсы;
9. Возможность изменять рабочую полосу;
10. Систему логгирования с возможностью сохранения в файл. Сохранение производится в файл с расширением txt.

## Список использованных источников

1. Категории турбулентности летательных аппаратов в соответствии со стандартами ICAO [Электронный ресурс] / Википедия. – Режим доступа: [https://en.wikipedia.org/wiki/Wake\\_turbulence#Wake\\_vortex\\_separation](https://en.wikipedia.org/wiki/Wake_turbulence#Wake_vortex_separation). Дата доступа 24.02.2016.
2. Код аэропорта ICAO [Электронный ресурс] / Википедия. – Режим доступа: [https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4\\_%D0%B0%D1%8D%D1%80%D0%BE%D0%BF%D0%BE%D1%80%D1%82%D0%B0\\_%D0%98%D0%9A%D0%90%D0%9E](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%B0%D1%8D%D1%80%D0%BE%D0%BF%D0%BE%D1%80%D1%82%D0%B0_%D0%98%D0%9A%D0%90%D0%9E). Дата доступа 26.02.2016.
3. Пацей, Н.В. Курс лекций по языку программирования C# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.
4. Руководство по WPF [Электронный ресурс] / Metanit.com. – Режим доступа: <http://metanit.com/sharp/wpf/>. Дата доступа: 28.02.2016.
5. Введение в WPF в Visual Studio 2015 [Электронный ресурс] / Msdn. – Режим доступа: <https://msdn.microsoft.com/ru-ru/library/aa970268%28v=vs.110%29.aspx>. Дата доступа 28.02.2016.
6. Анимации в WPF [Электронный ресурс] / Habrahabr.ru. – Режим доступа: <https://habrahabr.ru/post/111126/>. Дата доступа 10.03.2016.
7. Графика и анимации WPF [Электронный ресурс] / Professorweb.ru. – Режим доступа: [http://professorweb.ru/my/WPF/graphics\\_and\\_animation/level12/graph\\_animation\\_index.php](http://professorweb.ru/my/WPF/graphics_and_animation/level12/graph_animation_index.php). Дата доступа 10.03.2016.

## ПРИЛОЖЕНИЕ А.

### Листинг функции, управляющей анимированием

```
public static void CheckNextFlightActions()
{
    // смена полосы
    if (IsNeedToChangeRunway && IsRunwayChangeAvailable())
        ChangeActiveRunway();

    // удаление из отмененных рейсов
    if (IsNeedToDeleteCancelledFlight())
        CancelledDictionary.Remove(CancelledDictionary.First().Key);

    // анимирование руления на стоянку
    if (IsClearedForTaxiToStand())
        AnimationEvent(PlaneActions.TaxiToStand, TaxiToStandQueue.First());

    // анимирование руления на предварительный
    if (IsClearedForTaxiToHp())
        AnimationEvent(PlaneActions.TaxiToHp, TaxiToHpQueue.First());

    // анимирование посадки
    if (IsClearedForLand())
    {
        // если есть свободная подходящая стоянка - анимируем. Иначе - отмена рейса
        if (IsAnyAvailableStand(ArrivalList.First()))
            AnimationEvent(PlaneActions.Landing, ArrivalList.First());
        else
        {
            ArrivalList.First().ArrivalStatus = FlightStatus.Cancelled;
            ArrivalList.First().DepartureStatus = FlightStatus.Cancelled;
            CancelledDictionary.Add(
                ArrivalList.First(),
                Time.CurrentDate + TimeSpan.FromHours(2));
            AddToMsgBox($"Рейс {ArrivalList.First().ArrivalFlightNumber} ОТМЕНЕН");
            ArrivalList.RemoveFirst();
            PerformTimetablesUpdate();
        }
    }

    // анимирование взлёта
    if (IsClearedForTakeoff())
        AnimationEvent(PlaneActions.Takeoff, TakeoffQueue.First());

    // анимирование буксировки
    if (IsClearedForPushback())
        AnimationEvent(PlaneActions.Pushback, PushbackQueue.First());

    // каждые 30 минут проверять на задержанные рейсы
    if (Time.CurrentTime.Minute % 30 == 0)
        CheckForDelays();
}
```



## ПРИЛОЖЕНИЕ Б.

### Фрагмент листинга класса **Flight**

```
public Runways Runway { get; set; } // выбранная полоса для взлета/посадки
public PlaneActions CurrentAction { get; set; } // текущая анимация (действие)
public FlightStatus ArrivalStatus { get; set; } // статус прибывающего рейса
public FlightStatus DepartureStatus { get; set; } // статус вылетающего рейса
public AirplanesCategory Category { get; } // категория самолета (Heavy, Medium, Light)
public DateValue ArrivalTimetable { get; } // время прилета по расписанию
public DateValue DepartureTimetable { get; set; } // время вылета
public DateValue DepartureTime { get; set; } // реальное время вылета
public bool Blocked { get; set; } // при установке отложенного вылета true
public short Stand { get; set; } // номер стоянки
public string AircraftOperator { get; } // авиакомпания
public string ArrivalFlightNumber { get; } // номер рейса при прибытии
public string DepartureFlightNumber { get; } // номер рейса при отбытии
public string From { get; } // откуда
public string Destination { get; } // куда
public DateList Days { get; } // дни прилета
```

## ПРИЛОЖЕНИЕ В.

### Листинг класса DateValue

```
public class DateValue : IComparable
{
    public static DateList WeekDays { get; } =
        new DateList { "Бс", "Пн", "Вт", "Ср", "Чт", "Пт", "Сб" };

    public int WeekDay => Day % 7;
    public int Day => dateTime.Day;
    public int Hour => dateTime.Hour;
    public int Minute => dateTime.Minute;
    public double TotalMinutesToday => Hour * 60 + Minute;

    private DateTime dateTime; // обертываемый объект
    private readonly int hashCode;

    private static readonly Random Rnd = new Random();

    public DateValue()
    {
        hashCode = Rnd.Next(int.MinValue, int.MaxValue);
        dateTime = new DateTime();
    }

    public DateValue(DateTime t)
    {
        hashCode = Rnd.Next(int.MinValue, int.MaxValue);
        dateTime = t;
    }

    public DateValue(int hours, int minutes)
    {
        hashCode = Rnd.Next(int.MinValue, int.MaxValue);
        dateTime = new DateTime();
        dateTime = dateTime.AddHours(hours).AddMinutes(minutes);
    }

    public DateValue(int days, int hours, int minutes)
    {
        hashCode = Rnd.Next(int.MinValue, int.MaxValue);
        dateTime = new DateTime();
        dateTime = dateTime.AddDays(days).AddHours(hours).AddMinutes(minutes);
    }

    public string GetTime() => $"{Hour}:{Minute}";

    public string GetDate() => GetWeekDay() + ", " + GetTime();

    public string GetWeekDay() => WeekDays[WeekDay];

    public DateValue AddMinutes(int m) => new DateValue(dateTime.AddMinutes(m));

    public DateValue AddHours(int h) => new DateValue(dateTime.AddHours(h));

    public DateValue Duration(DateValue other) => this > other ? this - other : other - this;

    public static DateValue Duration(DateValue t1, DateValue t2)
    {
        t1 > t2 ? t1 - t2 : t2 - t1;
    }

    public static DateValue Parse(string s)
```

```

{
    var t = TimeSpan.Parse(s);
    return new DateValue(t.Days, t.Hours, t.Minutes);
}

public static DateValue operator -(DateValue t1, DateValue t2)
{
    var buff = t1.dateTime.Subtract(t2.dateTime);
    return new DateValue(new DateTime().AddTicks(buff.Ticks));
}

public static DateValue operator +(DateValue t1, DateValue t2)
{
    DateTime rc = t1.dateTime.AddMinutes(t2.Minute).AddHours(t2.Hour);
    return new DateValue(rc);
}

public static DateValue operator +(DateValue t1, TimeSpan t2)
{
    DateTime rc = t1.dateTime.AddMinutes(t2.Minutes).AddHours(t2.Hours);
    return new DateValue(rc);
}

public static bool operator <=(DateValue t1, DateValue t2)
{
    if (t1?.Day > t2?.Day) return false;
    if (t1?.Day < t2?.Day) return true;
    if (t1?.Hour > t2?.Hour) return false;
    if (t1?.Hour < t2?.Hour) return true;
    if (t1?.Minute > t2?.Minute) return false;
    return true;
}

public static bool operator >=(DateValue t1, DateValue t2)
{
    if (t1?.Day < t2?.Day) return false;
    if (t1?.Day > t2?.Day) return true;
    if (t1?.Hour < t2?.Hour) return false;
    if (t1?.Hour > t2?.Hour) return true;
    if (t1?.Minute < t2?.Minute) return false;
    return true;
}

public static bool operator <(DateValue t1, DateValue t2)
{
    if (t1?.Day > t2?.Day) return false;
    if (t1?.Day < t2?.Day) return true;
    if (t1?.Hour > t2?.Hour) return false;
    if (t1?.Hour < t2?.Hour) return true;
    if (t1?.Minute > t2?.Minute) return false;
    if (t1?.Minute < t2?.Minute) return true;
    return false;
}

public static bool operator >(DateValue t1, DateValue t2)
{
    if (t1?.Day < t2?.Day) return false;
    if (t1?.Day > t2?.Day) return true;
    if (t1?.Hour < t2?.Hour) return false;
    if (t1?.Hour > t2?.Hour) return true;
    if (t1?.Minute < t2?.Minute) return false;
    if (t1?.Minute > t2?.Minute) return true;
    return false;
}

public static bool operator ==(DateValue t1, DateValue t2)

```

```

=> (t1?.Day == t2?.Day) && (t1?.Hour == t2?.Hour) && (t1?.Minute == t2?.Minute);

public static bool operator !=(DateValue t1, DateValue t2) => !(t1 == t2);

public override string ToString() => new TimeSpan(Hour, Minute, 0).ToString();

public override int GetHashCode() => hashCode;

// возвращает true, если текущее время (ЧЧ:ММ) находится в промежутке [t1, t2]
public bool Between(DateValue t1, DateValue t2)
{
    if (Hour > t1.Hour && Hour < t2.Hour) return true;
    if (Hour < t1.Hour || Hour > t2.Hour) return false;
    if (Minute >= t1.Minute && Minute <= t2.Minute) return true;
    return false;
}

public int CompareTo(object obj)
{
    DateValue other = obj as DateValue;
    if (other == null) throw new ArgumentException("obj не является DateValue.");
    if (dateTime > other.dateTime) return 1;
    if (dateTime < other.dateTime) return -1;
    return 0;
}

protected bool Equals(DateValue other) => this == other;

public override bool Equals(object obj)
{
    if (ReferenceEquals(null, obj)) return false;
    if (ReferenceEquals(this, obj)) return true;
    if (obj.GetType() != GetType()) return false;
    return Equals((DateValue)obj);
}
}

```

## ПРИЛОЖЕНИЕ Г.

### Листинг функции анимирования посадки

```
private void PerformLandingAnimation(Flight flight)
{
    Airport.IsRunwayClearForLand = false;
    Airport.IsRunwayClearForTakeoff = false;
    flight.CurrentAction = PlaneActions.Landing;
    flight.Runway = Airport.RunwayInUse;
    flight.DepartureStatus = FlightStatus.CheckIn;

    var planeImg = new Image(); // изображение самолета
    var matrixTransform = new MatrixTransform(); // для перемещения картинки самолета
    var animationPath = GetPath(flight); // устанавливаем путь для анимирования посадки

    animationPath.Freeze(); // "замораживаем" анимацию для лучшей производительности

    // создаем NameScope, который понадобится для использования Storyboard
    NameScope.SetNameScope(this, new NameScope());

    // устанавливаем картинку самолёта и ее размеры, соответствующие типу
    SetImg(ref planeImg, ref matrixTransform, flight);

    // регистрируем matrixTransform, чтобы затем его использовать в Storyboard
    RegisterName("flight", matrixTransform);

    // создаем MatrixAnimationUsingPath для передвижения самолета
    var animation = new MatrixAnimationUsingPath
    {
        PathGeometry = animationPath,
        Duration = TimeSpan.FromSeconds(GetDurationForLandingOrTakeoff(flight)),
        DecelerationRatio = 0.9, // замедление анимации с этого момента (весь путь равен 1)
        DoesRotateWithTangent = true // вращение анимируемого элемента за направлением пути
    };

    animation.Completed += (completedSender, eSender) =>
    {
        if ((animation != null) && storyboard.Children.Contains(animation))
        {
            Airport.IsRunwayClearForTakeoff = true;
            storyboard.Children.Remove(animation);
            AddToLogger($"Пейс {Airport.ArrivalList.First().ArrivalFlightNumber} из {Airport.ArrivalList.First().From} произвёл посадку на {Airport.ArrivalList.First().Runway} полосу");
            Airport.TaxiToStandQueue.Enqueue(Airport.ArrivalList.First());
            Airport.ArrivalList.RemoveFirst();
            animation = null;
        }
    };

    // устанавливаем элемент(свойство), к которому будет применена анимация
    Storyboard.SetTargetProperty(animation, new PropertyPath(MatrixTransform.MatrixProperty));
    Storyboard.SetTargetName(animation, "flight"); // задаем присоединяемый к анимации элемент

    AddToLogger($"Пейс {flight.ArrivalFlightNumber} выполняет посадку на {flight.Runway.ToString("f")} полосу");
    storyboard.Children.Add(animation);
    planeImg.BeginStoryboard(storyboard); // запускаем анимацию
}
```