Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина: Программирование

Отчет

по курсовому проекту

на тему

«клавиатурный тренажёр»

Выполнила: студент гр. 152002

Игнатенко Андрей Александрович

Проверил: Стаховкий Сергей Михайлович

Минск 2012

СОДЕРЖАНИЕ

1. Задание …………………………………………………………………………3
2. Введение ………………………………..………..………..……...…..………..4
3. Теория ……………………………………………………..……….…....……..6
4. Практика ……….…………………………………..………..…………....…..10
5. Руководство пользователя …………….……………..…………….……......15
6. Заключение ……………………..………..………..…………….……..……..21
7. Литература……………….……………..……………………………………..22
8. Приложение…….………………………..………..………..…………………23
9. Листинг программы………………………………….……………………….23

# Задание

Необходимо разработать и реализовать программу «Клавиатурный тренажёр». Программа должна помочь в обучении слепому методу печати, увеличении скорости набора, уменьшении количества опечаток. Программа поддерживает русскую раскладку.

# Введение

Все мы способны в той или иной степени внятно излагать свои мысли вслух, по крайней мере, у 90 процентов людей с этим проблем не возникает. Но вот когда вы пытаетесь перенести свои мысли на бумагу или страницу веб сайта, то все уже выходит не так гладко. А почему? Мне кажется, что всему виной именно невозможность быстро печатать (или писать на бумаге) с той же скоростью, как формируются и складываются в стройные ряды мысли в вашей голове.

Очень важно во время печати текста не смотреть на клавиатуру в поиске (пусть и подсознательном) нужной клавиши. Ведь у большинства людей нет достаточно большого буфера в памяти, чтобы там хранить все фразы, которые вы еще не успели перенести на бумагу или экран. У кого-то память такой буфер имеет и у них не будет проблем с написанием внятных и лаконичных текстов даже с учетом их не быстрой скорости не слепой печати.

Но большинство из нас не обладает такой природной способностью, а, следовательно, будет наблюдаться явный диссонанс между лаконичных формированием мыслей в голове и неуклюжим их переносом на твердый носитель путем двухпальцевого **не слепого метода не быстрой печати**.

Мысли успевают за это время потеряться, красивые и легко воспринимаемые фразы, пришедшие вам в голову, ускользают и просачиваются сквозь пальцы и вы начинаете нервничать и злиться, а потом вообще забиваете на все попытки стать «чукчей — писателем» и начинаете искать возможности скопировать или купить чужой контент.

А ведь счастье было так близко. Не хватает самой малости — **научиться быстро печатать на клавиатуре**. В название этого слепой десятипальцевой печати сосредоточены все его достоинства — задействуя все десять пальцев вы существенно повышаете теоретически возможную скорость печати, а возможность печатать вслепую, не смотря на клавиатуру при наборе текста, позволят вам как бы это сказать — печатать взглядом, наблюдая на экране материализующиеся в словах мысли и формулировки.

В общем, ответ на вопрос **«как научиться быстро печатать»** позволит вам полностью освобождать мозг от рутины и позволит ему не отвлекаться на такие пошлые вещи как поиск нужных клавиш и постоянный перевод взгляда с клавиатуры на экран монитора.

# Теория

При разработке программы применялся [объектно-ориентированный](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](http://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) **C#**, использующий технологию .Net Framework.

**Язык программирования C#:**

C# — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Java, Delphi, Модула и Smalltalk — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем. (Однако эта закономерность была нарушена с выходом C# 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET.) CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

**.Net Framework**:

.NET Framework — программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является исполняющая среда Common Language Runtime (CLR), способная выполнять как обычные программы, так и серверные веб-приложения. .NET Framework поддерживает создание программ, написанных на разных языках программирования.

Считается, что платформа .NET Framework явилась ответом компании Microsoft на набравшую к тому времени большую популярность платформу Java компании Sun Microsystems (ныне принадлежит Oracle).

Хотя .NET является патентованной технологией корпорации Microsoft и официально рассчитана на работу под операционными системами семейства Microsoft Windows, существуют независимые проекты (прежде всего это Mono и Portable.NET), позволяющие запускать программы .NET на многих других операционных системах.

Разработка платформы началась в 1999 году. Официально о разработке новой технологии было объявлено 13 января 2000 года, в день когда Билл Гейтс официально объявил о передаче поста главы Microsoft Стиву Балмеру. В этот день руководством корпорации была озвучена новая стратегия компании, получившая название Next Generation Windows Services (сокр. NGWS, рус. Новое поколение служб Windows). Новая стратегия должна была объединить в единый набор существующие и будущие разработки Microsoft для предоставления возможности пользователям работать со Всемирной паутиной с беспроводных устройств, обладающих доступом в Интернет, как со стационарных компьютеров.

Концепция:

В феврале 2000 года Стив Балмер, выступая на конференции VBITS 2000 в Сан-Франциско, рассказал о новой версии Visual Basic и Visual C++, являвшихся до той поры краеугольными камнями Visual Studio. Поскольку график выхода версий Visual Studio к тому моменту был изрядно нарушен (Visual Studio 6.0 вышла в сентябре 1998 года, а до этого выпуски входящих в её состав инструментов происходили ежегодно), то информация о нововведениях вызывала изрядный интерес.

Тогда же Балмер и представил концепцию стратегии NGWS (рабочее название которой Балмер счёл весьма ужасным и пообещал сменить его к моменту выпуска) в виде некоей программной платформы, которая позволит пользователям Windows получать доступ к множеству веб-служб. Для этого Microsoft рассчитывала значительно изменить Windows-платформу. Первым шагом к новой версии платформы стала Windows 2000, которую Балмер назвал основой «пользовательского знакомства с Интернет». Windows 2000 являлась одной из частей технологии Windows Distributed interNet Applications (DNA) 2000, в рамках которой была представлена последняя на тот момент схема трёхуровневой модели программирования, впервые представленная на Windows NT 4.0.

На тот момент данная схема состояла из клиентской части (Windows- или веб-приложение), среднего уровня, базировавшегося на компонентах COM+, и базы данных (как правило SQL Server). NGWS должна была сменить путь разработки Windows DNA на исключительное использование XML как стандарта обмена данными. XML был выбран потому, что являлся развитием языка разметки HTML, ставшего основным языком разметки во Всемирной паутине, а кроме того он предоставлял удобный способ описания данных. В модели NGWS каждая часть трёхзвенной структуры (клиентское приложение, промежуточный слой и база данных) должна была взаимодействовать с остальными при помощи XML-пакетов. Для упрощения разработки кода с использованием XML предполагалось, что новая версия Visual Studio будет генерировать XML-код, освобождая разработчиков от необходимости ручного написания XML-кода.

Так, например, планировалось, что разработчики смогут писать веб-сайт для электронной коммерции целиком на новой версии Visual Basic, а благодаря тому, что обмен информацией будет происходить при помощи XML, разработчики смогут создавать клиентские приложения, функционирующие на Linux, Solaris и Mac OS. То есть, для того, чтобы приложение или операционная система могли взаимодействовать друг с другом нужна была лишь поддержка стандарта с их стороны.

Для демонстрации своего замысла Microsoft запустила веб-сайт Passport, зайдя на который, пользователь мог бесплатно завести себе электронный кошелёк (eWallet), упрощавший процесс совершения покупок в интернет-магазинах.

Основной идеей при разработке .NET Framework являлось обеспечение свободы разработчика за счёт предоставления ему возможности создавать приложения различных типов, способные выполняться на различных типах устройств и в различных средах.

Вторым принципом стало ориентирование на системы, работающие под управлением семейства операционных систем Microsoft Windows.

# Практика

В ходе реализации программы «клавиатурный тренажёр» был создан ряд классов для управления программой и пользовательским интерфейсом.

Одним из важнейших компонентов программы является класс **ProgrammEngine**. При его реализации использовался порождающий паттерн Одиночка (Singleton) для обеспечения существования единственного экземпляра данного класса. Класс отвечает за взаимодействие программы с пользователем, обладает такими методами как:

* UserAccount RequestUserAccount() – запросить (создать новый либо выбрать из списка) у пользователя его профиль.
* Level RequestLevel() — запросить (выбрать из списка) у пользователя уровень, который он желает запустить.

Также данный класс отвечает за инициализацию и завершение работы программы. При запуске он загружает из файлов данные пользовательских профилей, а также уровни для прохождения. При завершении работы приложения он сохраняет накопившуюся статистику прохождения уровней.

Движком непосредственно «клавиатурного тренажёра», т.е. среды для набора текста, подсчёта количества опечаток, затраченного времени и другой статистики, является класс **LevelEngine**. В своём конструкторе он получает уровень (Level) с которым будет происходить работа. Его важнейшие методы это:

* Start() – запускает отсчёт времени и создаёт список символов (из текста), которые пользователю необходимо будет ввести.
* Reset() – сбрасывает счётчик времени и ошибок
* bool EnterChar(char symble) – получает символ и возвращает true, если символ верный, либо false, если допущена опечатка

Также важным элементом данного класса является событие Finished, которое срабатывает в момент ввода последнего символа текста, либо когда допущено слишком много ошибок.

При успешном окончании уровня генерируется статистика, включающая скорость печати, затраченное время и количество опечаток. Затраченное время подсчитывается следующим образом: в момент начала печати фиксируется текущее время, а по окончании уровня оно вычитается из времени окончания. Таким образом мы получаем затраченное время. Скорость печати подсчитывается в символах в минуту путём деления количества символов в тексте на затраченное время.

В программе реализована система профилей пользователей, т.е. возможно создать несколько аккаунтов. Таким образом, статистика прохождения ведётся раздельно для каждого пользователя. Информация о пользователях хранится в файле (users.dat) в той же директории, что и исполняемый файл.

Для управления системой профилей используется класс **UserAccountsEngine**. Он также реализует паттерн Одиночка (Singleton). Данный класс используется посредством следующих методов:

* UserAccount CreateNewUser(string userName) – создаёт новый профиль, если данное имя пользователя ещё не занято..
* bool UserExists(string userName) – проверяет, существует ли пользователь с заданным именем.
* Свойство List<UserAccount> UserAccounts – возвращает список профилей пользователей.

Также в программе используется ряд классов, носящих вспомогательный характер:

* Класс **Levels** – представляет собой коллекцию уровней (Level), уровни хранятся в файле (levels.dat). Класс реализует паттерн Одиночка, имеет методы:
  + Add(Level level) – добавление нового уровня (в этот момент ему присваивается номер)
  + Level Get(int levelNumber) – возвращает уровень, в качестве аргумента принимает номер уровня
* Класс **Level** – хранит информацию об уровне и условиях его прохождения:
  + Текст для набора
  + Максимально допустимое количество опечаток
  + Минимально допустимая скорость печати
  + Номер уровня
* Класс **LevelStatistics** - представляет статистику прохождения уровня и хранит:
  + Количество опечаток
  + Скорость печати
  + Номер уровня
  + Уровень, к которому относится данная статистика
* Класс **TypingSpeed** – скорость печати. Хранит скорость в символах в минуту. Реализует интерфейсы сравнения (IComparable<TypingSpeed>, IEquatable<TypingSpeed>), что позволяет легко сравнивать объекты данного типа.
* Класс **UserAccount** – представляет информацию о профиле пользователя. Содержит следующую информацию:
  + Имя профиля
  + Его статистику прохождения (Progress)
* Класс **Progress** – содержит коллекцию уровней и соответствующих им статистику прохождения. Содержит методы добавления и получения статистики.

Порядок использования этих классов в программе следующий: сразу после запуска приложения с помощью класса **ProgrammEngine** загружаются профили пользователей и уровни из файлов, далее у пользователя запрашивается профиль (UserAccount) с помощью метода RequestUserAccount. Далее с помощью метода RequestLevel у пользователя запрашивается уровень для прохождения. С помощью класса **LevelEngine** запускается непосредственно сам уровень. После его завершения у пользователя запрашивается следующий уровень.

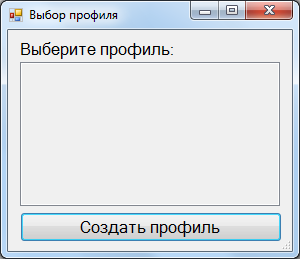
# Руководство пользователя

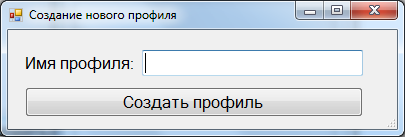
Системные требования:

* Операционная система Windows XP / 7 / 8
* .Net Framework 4.0 или выше
* Удобная клавиатура

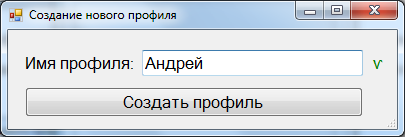
Программа проста в использовании и не требует специальных навыков для работы с ней.

При первом запуске появится окно следующего вида:

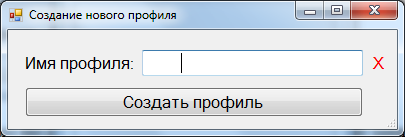


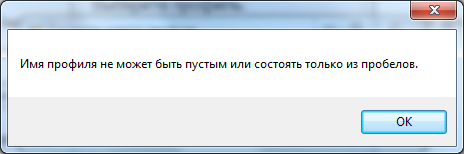
Список профилей пуст, потому необходимо создать новый профиль. Для этого требуется нажать на кнопку «Создать профиль». Откроется окно создания нового профиля:  


Для создания нового профиля требуется ввести лишь его название в окно «Имя профиля». Имя профиля не может быть пустым или состоять только из пробелов. Если введено имя правильного формата, а также оно не было занято ранее другими пользователями, то рядом с введённым именем профиля появится зелёный значок:



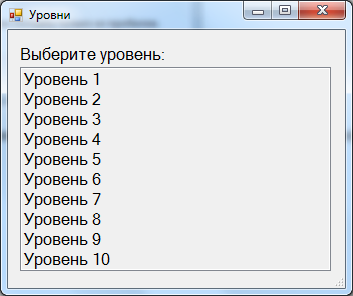
Если же было введено имя только из пробелов, либо оно было занято другими пользователями, то появится красный значок:



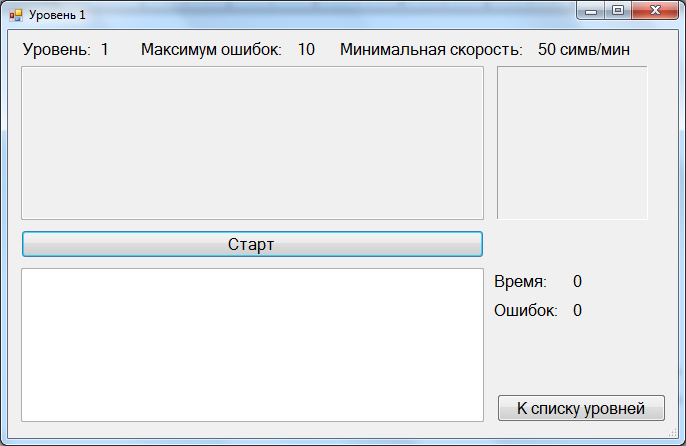
При попытке создания профиля с неправильным именем появляется следующее предупреждение:  


Когда будет введено правильно имя, необходимо нажать кнопку «Создать профиль».

Сразу после этого появится окно с выбором уровня:

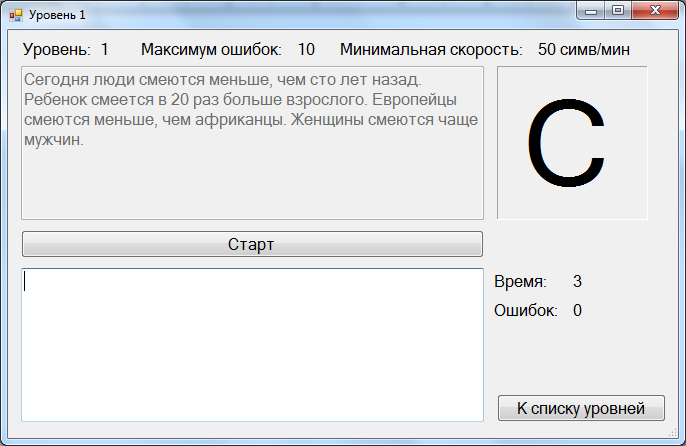


Сложность уровней возрастает от первого к последнему. Тем не менее, можно начать прохождение с любого. Чтобы выбрать уровень, необходимо нажать на его название в списке. Сразу после этого откроется окно этого уровня:

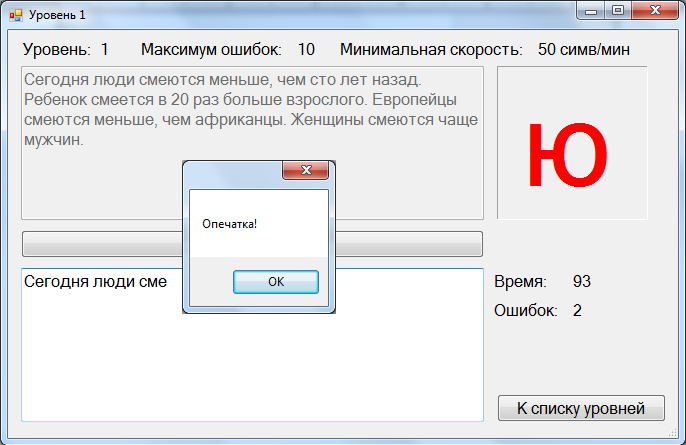


В верху окна вы можете увидеть требования к прохождению данного уровня. Это такие требования как: максимально допустимое количество ошибок и минимальная скорость, с которой необходимо напечатать данный текст, чтобы уровень считался пройденным.

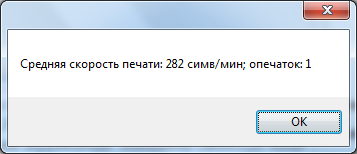
Для начала прохождения надо нажать кнопку «Старт»:



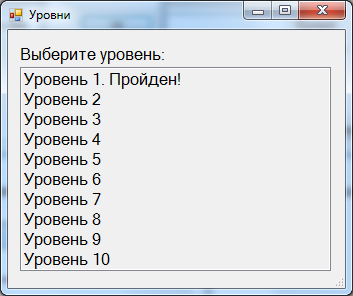
В верхней половине окна появится текст, который необходимо набирать, справа вверху текущая буква. Если при наборе текста будет допущена опечатка, то появится предупреждение, а буква станет красной.



После успешного набора текста, появится окошко со статистикой прохождения:



После этого можно выбрать другой уровень:



Для выхода из программы необходимо просто закрыть это окно. Вся ваша статистика будет сохранена автоматически.

# Заключение

В результате выполнения курсового проекта была разработана программа «Клавиатурный тренажёр». Программа может использоваться в следующих случаях:

* для обучения [слепому методу печати](http://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%B5%D0%BF%D0%BE%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4_%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D0%B8), в частности, задействовать для набора все десять [пальцев рук](http://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BB%D1%8C%D1%86%D1%8B_%D1%80%D1%83%D0%BA)
* для увеличения скорости набора
* для уменьшения количество опечаток

В программе имеется система профилей, предусмотрено сохранение статистики прохождения. Добавлено 10 уровней различной сложности для прохождения.

# Литература

1. Албахари, Дж. C# 3.0. Справочник: Пер. с англ. / Дж. Албахари, Б. Ал-бахари. – 3-е изд. – Спб.: БХВ-Петербург, 2009. – 944 с.: ил.
2. Нэш, Т. C# 2010: ускоренный курс для профессионалов / Т. Нэш. – М.: издательский дом «Вильямс», 2010. – 592 с.
3. Троелсен, Э. Язык программирования C# 2010 и платформа .NET 4.0 / Э. Троелсен. – 5-е изд. – М.: ООО «И.Д. Вильямс», 2011. – 1392 с.: ил.
4. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Дж. Рихтер. – 3-е изд. – Спб.: Питер, 2012. – 928 с.: ил.
5. Фримен, А. LINQ: язык интегрированных запросов в C# 2010 для про-фессионалов / А. Фримен, Дж. С. Раттц-мл. – М.: Издательский дом «Виль-ямс», 2011. – 656 с.
6. Хейлсберг, А. Язык программирования C#. Классика Computers Science. / А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд. – 4-е изд. – Спб.: Питер, 2012. – 784 с.

# Приложение 1. Листинг программы

using System;

using System.Diagnostics;

using System.Windows.Forms;

namespace Typing\_Tutor

{

class ProgrammEngine

{

private static ProgrammEngine \_instance;

private UserAccount \_userAccount;

public Progress Progress { get { return \_userAccount.Progress; } }

protected ProgrammEngine()

{

Initialize();

}

public static ProgrammEngine GetInstance()

{

return \_instance ?? (\_instance = new ProgrammEngine());

}

public void SetUserAccount(UserAccount userAccount)

{

if (userAccount == null)

throw new ArgumentNullException("userAccount");

\_userAccount = userAccount;

}

public UserAccount RequestUserAccount()

{

var userChoiceForm = new UserSelectionForm();

UserAccount userAccount = null;

userChoiceForm.UserAccountSelected += (sender, args) => userAccount = args.EventInfo;

userChoiceForm.ShowDialog();

Debug.Assert(userAccount != null, "Account was not selected.");

return userAccount;

}

public Level RequestLevel()

{

var levelsForm = new LevelsForm();

Level selectedLevel = null;

levelsForm.LevelSelected += (sender, args) => selectedLevel = args.EventInfo;

levelsForm.ShowDialog();

return selectedLevel;

}

public LevelStatistics PlayLevel(Level level)

{

if (level == null)

throw new ArgumentNullException("level");

var levelForm = new LevelForm(level);

LevelStatistics statistics = null;

levelForm.LevelFinished += (sender, args) => statistics = args.EventInfo;

levelForm.ShowDialog();

if (statistics != null && statistics.Complited)

Progress.AddStatistics(statistics);

return statistics;

}

private static void Initialize()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

try

{

UserAccountsEngine.Deserialize();

Levels.Deserialize();

}

catch (Exception)

{

}

}

public static void CloseProgram()

{

try

{

UserAccountsEngine.Serialize();

Levels.Serialize();

}

catch (Exception)

{

}

finally

{

Environment.Exit(0);

}

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Threading.Tasks;

namespace Typing\_Tutor

{

[Serializable]

public class UserAccountsEngine

{

private static UserAccountsEngine \_instance;

private readonly List<UserAccount> \_userAccounts;

public const string DefaultSavingPath = "users.dat";

protected UserAccountsEngine()

{

\_userAccounts = new List<UserAccount>();

}

public static UserAccountsEngine GetInstance()

{

return \_instance ?? (\_instance = new UserAccountsEngine());

}

public List<UserAccount> UserAccounts

{

get { return \_userAccounts; }

}

public UserAccount CreateNewUser(string userName)

{

if (userName == null)

throw new ArgumentNullException("userName");

if (UserExists(userName))

return null;

var userAccount = new UserAccount(userName);

\_userAccounts.Add(userAccount);

return userAccount;

}

public bool UserExists(string userName)

{

if (userName == null)

throw new ArgumentNullException("userName");

return \_userAccounts.Any(account => account.Name == userName);

}

public static void Serialize()

{

Serialize(DefaultSavingPath);

}

public static void Serialize(string path)

{

if (\_instance == null) return;

using (Stream fileStream = File.Create(path))

{

var br = new BinaryFormatter();

br.Serialize(fileStream, \_instance);

}

}

public static void Deserialize()

{

Deserialize(DefaultSavingPath);

}

public static void Deserialize(string path)

{

using (Stream fileStream = File.OpenRead(path))

{

var br = new BinaryFormatter();

\_instance = (UserAccountsEngine)br.Deserialize(fileStream);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

namespace Typing\_Tutor

{

class LevelEngine

{

private readonly Level \_level;

private int \_errorsNumber;

private DateTime \_startTime;

private DateTime \_finishTime;

private bool \_finished;

private Queue<char> \_charsForTyping;

public LevelEngine(Level level)

{

if (level == null)

throw new ArgumentNullException("level");

\_level = level;

}

public Level Level { get { return \_level; } }

public char CurrentChar

{

get { return \_charsForTyping.Count != 0 ? \_charsForTyping.Peek() : ' '; }

}

public int ErrorsNumber { get { return \_errorsNumber; } }

public void Start()

{

\_charsForTyping = new Queue<char>(\_level.Text.ToList());

\_startTime = DateTime.Now;

}

public bool EnterChar(char symble)

{

if (\_finished)

throw new Exception("Level already finished. You need to restart it.");

if (symble == CurrentChar)

{

RightCharEnteret(symble);

return true;

}

WrongCharEntered(symble);

return false;

}

private void RightCharEnteret(char symble)

{

OnCharEntered(new EventArgs<char>(symble));

MoveToNextChar();

}

private void MoveToNextChar()

{

\_charsForTyping.Dequeue();

if (!\_charsForTyping.Any())

Finish();

}

private void WrongCharEntered(char symble)

{

\_errorsNumber++;

OnErrorCharEntered(new EventArgs<char>(symble));

if (\_errorsNumber > \_level.MaxErrorsNumber)

Finish();

}

public void Reset()

{

\_errorsNumber = 0;

\_finished = false;

}

private void Finish()

{

\_finished = true;

\_finishTime = DateTime.Now;

var elapsedTime = \_finishTime - \_startTime;

var typingSpeed = new TypingSpeed(\_level.Text.Length, elapsedTime);

var isSucceed = \_errorsNumber <= \_level.MaxErrorsNumber && typingSpeed >= \_level.MinTypingSpeed;

var levelStatistics = new LevelStatistics(\_level, \_errorsNumber, typingSpeed, isSucceed);

OnFinished(new EventArgs<LevelStatistics>(levelStatistics));

}

public event EventHandler<EventArgs<char>> CharEntered;

public void OnCharEntered(EventArgs<char> e)

{

var handler = CharEntered;

if (handler != null) handler(this, e);

}

public event EventHandler<EventArgs<char>> ErrorCharEntered;

public void OnErrorCharEntered(EventArgs<char> e)

{

var handler = ErrorCharEntered;

if (handler != null) handler(this, e);

}

public event EventHandler<EventArgs<LevelStatistics>> Finished;

public void OnFinished(EventArgs<LevelStatistics> e)

{

var handler = Finished;

if (handler != null) handler(this, e);

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Threading.Tasks;

namespace Typing\_Tutor

{

[Serializable]

class Levels

{

private static Levels \_instance;

private readonly SortedList<int, Level> \_levels;

public const string DefaultSavingPath = "levels.dat";

protected Levels()

{

\_levels = new SortedList<int, Level>();

}

public static Levels GetInstance()

{

return \_instance ?? (\_instance = new Levels());

}

public int LastLevelNumber

{

get

{

return \_levels.Any() ? \_levels.Last().Key : 0;

}

}

public List<Level> LevelList

{

get { return \_levels.Select(pair => pair.Value).ToList(); }

}

public void Add(Level level)

{

if (level == null)

throw new ArgumentNullException("level");

if (\_levels.ContainsValue(level))

throw new Exception("This level is already added.");

level.SetLevelNumber(LastLevelNumber + 1);

\_levels.Add(level.LevelNumber, level);

}

public Level Get(int levelNumber)

{

return \_levels.ContainsKey(levelNumber) ? \_levels[levelNumber] : null;

}

public static void Serialize()

{

Serialize(DefaultSavingPath);

}

public static void Serialize(string path)

{

if (\_instance == null) return;

using (Stream fileStream = File.Create(path))

{

var br = new BinaryFormatter();

br.Serialize(fileStream, \_instance);

}

}

public static void Deserialize()

{

Deserialize(DefaultSavingPath);

}

public static void Deserialize(string path)

{

using (Stream fileStream = File.OpenRead(path))

{

var br = new BinaryFormatter();

\_instance = (Levels)br.Deserialize(fileStream);

}

}

}

}