МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**МОДЕЛИРОВАНИЕ АВТОМАТОВ С ТЬЮРИНГОВЫМ ОБОРУДОВАНИЕМ**

Работу выполнил И.А. Михеев

(подпись)

Направление подгтовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность (профиль) Технология программирования

Научный руководитель

доц., канд. физ.-мат. наук С.А. Жуков

(подпись)

Нормоконтролер

ст. преп., канд. техн. наук Е.Е. Полупанова

(подпись)

Краснодар

2018

**СОДЕРЖАНИЕ**

Введение 3

1 Машина Тьюринга 4

1.1 Обычная машина Тьюринга 4

1.1.1 Компоненты и неформальное описание 4

1.1.2 Формальное определение детерминированной машины Тьюринга 5

1.2 Разновидности машин Тьюринга 6

1.2.1 Многоленточная машина Тьюринга 7

1.2.2 Многоголовочная машина Тьюринга 8

2 Используемые инструменты 9

3 Имитатор машины Тьюринга 11

3.1 Возможности приложения 11

3.2 Интерфейс приложения 11

3.3 Описание языка программ 13

3.4 Реализация 15

3.4.1 Лента 15

3.4.2 Машина 17

3.4.3 Окна приложения 18

3.5 Описание работы приложения 20

Заключение 22

Список использованных источников 23

Приложение А Листинг программы 24

**ВВЕДЕНИЕ**

Абстрактные вычислительные машины имеют важное значение в таких областях, как теория алгоритмов и теория сложности вычислений. Они позволяют, например, анализировать сложность алгоритмов или доказывать неразрешимость или труднорешаемость проблем. Поэтому, если при решении некоторой проблемы установлено, что для нее не существует эффективного алгоритма, можно либо искать эффективные алгоритмы для более частной проблемы, либо изменить постановку задачи, наложив дополнительные ограничения [1].

Одной из таких машин является машина Тьюринга (МТ). Эту модель предложил А. М. Тьюринг в 1936 году [2]. Преимуществом МТ является то, что она достаточно проста и хорошо подходит для доказательства связанных с вычислимостью проблем. МТ является формальной моделью вычислителя. Согласно тезису Черча-Тьюринга, любая алгоритмически вычислимая функция может быть вычислена с помощью машины Тьюринга. Такое свойство вычислителя называется полнотой по Тьюрингу. Это означает, что любое физическое вычислительное устройство не способно реализовать функцию, которая не могла бы быть вычислена с помощью МТ. Язык программирования, являющийся полным по Тьюрингу, теоретически способен решить любую задачу, вычислимую с помощью компьютера. Почти все языки программирования являются полными по Тьюрингу. Для доказательства полноты некоторой системы необходимо показать, что с помощью нее можно моделировать другую полную по Тьюрингу систему. Чаще всего, наиболее просто показать, что система может моделировать универсальную машину Тьюринга – машину Тьюринга, которая может моделировать любую другую машину Тьюринга.

Целью работы является реализация программы, которая позволяет имитировать вычисления машины Тьюринга. Также, в частности, таких её разновидностей, как многоленточная МТ и многоголовочная МТ.

**1 Машина Тьюринга**

**1.1 Обычная машина Тьюринга**

**1.1.1 Компоненты и неформальное описание**

Перед тем как дать формальное определение, начнем с неформального описания. Машина Тьюринга содержит три компонента.

1. Конечное управление состоянием. Этот компонент находится в одном из конечного числа состояний в данный момент времени, и связан с ленточной головкой.

2. Ленточная головка. Этот компонент сканирует (обозревает) одну из клеток ленты в данный момент, и связан с конечным управлением. Головка может читать с ленты символы и записывать их на неё, а также перемещаться влево и вправо по ленте.

3. Лента. Этот компонент состоит из бесконечного числа клеток, каждая из которых может содержать один из конечного числа ленточных символов в данный момент времени. Лента является бесконечной в обе стороны.

На рисунке 1.1 изображены компоненты МТ и то, как они связаны между собой.

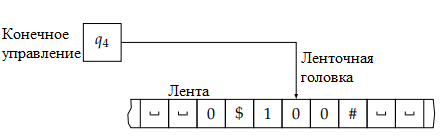


Рисунок 1.1 – Связь компонентов машины Тьюринга

Идея состоит в том, что поведение МТ в данный момент определяется состоянием, в котором находится конечное управление, и символом на той клетке ленты, которую читает головка. Таким образом, существует конечное число возможных действий: по одному для каждой пары состояние-символ. Всего есть три действия, который способна выполнять МТ на каждом шаге. Она может изменить состояние конечного управления, записать новый символ в читаемую головкой клетку, переместить головку влево или вправо по ленте. После этих действий, на следующем шаге, конечное управление получает новое состояние, головка читает новый символ, и процесс продолжается [3]. Машина остановится, если она перешла в особое состояние, называемое терминальным, или если переход для текущей пары состояние-символ не задан. Символы на ленте после остановки МТ могут рассматриваться как результат работы МТ.

**1.1.2 Формальное определение детерминированной машины Тьюринга**

Детерминированной машиной Тьюринга называется семёрка

(1.1)

где

Q – конечное непустое множество состояний;

Σ – множество символов, называемое входным алфавитом;

Γ – множество ленточных символов, при этом Σ ∪ {B} ⊆ Γ;

δ – функция переходов;

q0 – начальное состояние, в котором находится конечное управление в начальный момент времени, q0 ∈ Q;

B – пустой символ, B ∈ Γ и B ∉ Σ;

F – множество терминальных состояний, F ⊆ Q.

Функция переходов определяется следующим образом.

(1.2)

где

Q, F, Γ – множества из формулы (1.1);

D – направление сдвига головки, может иметь значения «влево» (L), «вправо» (R).

Интерпретировать функцию переходов, описываемую формулой (1.2), можно следующим образом. Предположим, что машина Тьюринга находится в состоянии q ∈ Q, σ – символ на ленте в клетке, обозреваемой головкой, а также задан переход δ(q, σ) = (r, τ, D). В этом случае МТ выполнит следующие действия. Состояние конечного управления сменится на r, головка перезапишет символ на ленте в текущей клетке на τ и переместится на одну клетку в направлении D. Для состояний из множества F переходы не задаются, так как машина останавливается при достижении состояния из этого множества.

**1.2 Разновидности машин Тьюринга**

Машина Тьюринга может определяться по-разному. В различных источниках определение МТ может слегка отличаться в некоторых местах. В рассмотренной ранее машине Тьюринга предполагается, что лента бесконечна в обоих направлениях. Но иногда удобно выбрать модель с лентой, бесконечной только в одном направлении. В этом случае появляется крайняя клетка ленты, и необходимо определить поведение МТ в такой ситуации. Возможен также вариант, когда головка может не перемещаться по ленте, а оставаться на месте. Тогда множество D из формулы (1.2) будет содержать еще один элемент «N», означающий отсутствие сдвига.

**1.2.1 Многоленточная машина Тьюринга**

Многоленточная МТ работает подобно обычной МТ, однако имеет она k головок, работающих независимо с k лентами, k – фиксированное положительное целое число. Рисунок 1.2 иллюстрирует многоленточную МТ с тремя лентами.

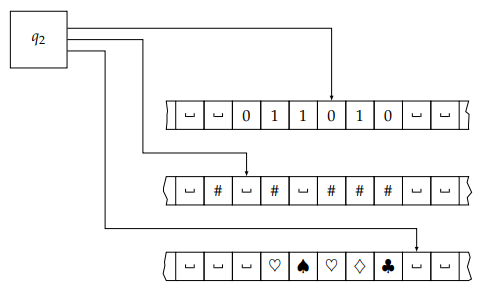


Рисунок 1.2 – Многоленточная машина Тьюринга с тремя лентами

Подобно обычной МТ, многоленточная определяется формулой (1.1), однако имеет другую формулу переходов, определенную следующим образом.

. (1.3)

Также здесь рассматривается вариант МТ, головки которой могут оставаться на месте.

Интерпретировать формулу (1.3) можно следующим образом. Пусть задан переход δ(p, (a1, …, ak)) = (q, (b1, …, bk), (d1, …, dk)). Если МТ находится в состоянии p и читает символы a1, …, ak на k лентах, то новым состоянием станет q, символы b1, …, bk запишутся на k лент, перезаписывая a1, …, ak, а головка с номером j сдвинется в направлении hj, j = 1, …, k.

**1.2.2 Многоголовочная машина Тьюринга**

Многоголовочная машина Тьюринга может рассматриваться как МТ с одной лентой и одним конечным управлением состоянием, но со многими независимыми головками, которые могут записывать и считывать. Однако в каждом состоянии активна может быть только одна головка [4][5]. Функция переходов для МТ с k головками определяется следующим образом.

(1.3)

где

Q = Q1 ∪ Q2 ∪ … ∪ Qk, Qi – множество состояний головки с номером i, k – фиксированное положительное целое число.

На рисунке 1.3 показан вид многоголовочной МТ при k = 3.

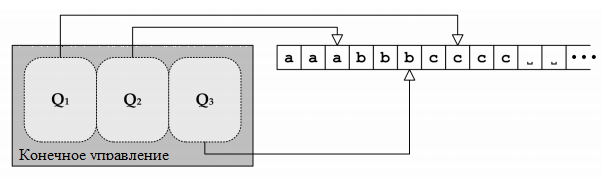


Рисунок 1.3 – Многоголовочная МТ с тремя головками

**2 Используемые инструменты**

Проект реализован в среде Microsoft Visual Studio на языке программирования C# с использованием системы Windows Presentation Foundation (WPF). WPF – это современная графическая система для Windows, является частью платформы .NET. В ее основе лежит мощная инфраструктура, основанная на DirectX.

WPF – один из лучших инструментов создания приложений для Windows Vista, Windows 7, Windows 8 и Windows 10, поскольку имеет ряд ключевых достоинств. За счет аппартного ускорения обеспечивается высокая производительность. Качество внешнего вида остается высоким даже при низком разрешении экрана. Используя web-подобную модель компоновки элементов, WPF поддерживает гибкий интерфейс, при котором размер и положение элементов может динамически меняться в зависимости от содержимого и размеров окна. WPF предоставляет возможность отображения стилизованного текста в любой части интерфейса.

WPF поддерживает декларативное определение пользовательского интерфейса с помощью языка разметки XAML (Extensible Application Markup Language), основанного на XML. Достоинство такого подхода состоит в том, что интерфейс может быть отделен от программного кода, а благодаря поддержке стилей можно легко задать новый стиль любому элементу [6].

Для работы с WPF необходим любой совместимый с .NET язык. Самым распространенным из них является язык C#. Язык C# наряду с такими языками, как C++, Java, PHP, относится к семейству языков программирования, основанному на C. Кроме того, он смоделирован на основе различных аспектов языков Visual Basic и C++. Вследствие чего, C# можно назвать гибридом из нескольких языков: такой же простой, как VB, и такой же мощный и гибкий, как C++.

Для C# характерны следующие особенности. В программах на C# нет необходимости использовать указатели напрямую, хотя это возможно. Управление памятью осуществляется автоматически при помощи сборки мусора. В C# есть возможность создания обобщенных типов, используя которые, можно создавать безопасный к типам и в то же время очень мощный код [7].

**3 Имитатор машины Тьюринга**

**3.1 Возможности приложения**

Приложение позволяет имитировать работу машины Тьюринга. Возможна имитация как обычной МТ, так и таких её вариаций, как многоленточная МТ и многоголовочная МТ.

Программы для машины Тьюринга записываются в текстовом формате в виде набора правил перехода и специальных описаний, которые могут задавать, например, начальное состояние. Программы можно сохранять и загружать в виде текстового файла.

Исполнение программы может происходить пошагово или автоматически с некоторой заданной скоростью. Оно может быть приостановлено в любой момент.

Панель с лентой содержит поля – клетки с символами. Эти поля можно изменять, задавая таким образом конфигурацию ленты МТ. Каждое такое поле обозначено числом – номером клетки на ленте. Также панель содержит кнопки прокрутки ленты, что позволяет задавать конфигурацию ленты неограниченной длины.

На информационной панели отображаются начальное и терминальные состояния. А в каждый момент времени указывается последний выполненный машиной переход.

Кроме того, есть возможность автоматически «очистить» всю ленту, то есть заполнить её пустыми символами, и сбросить текущее состояние машины на начальное.

**3.2 Интерфейс приложения**

На рисунках 3.1 – 3.3 показан интерфейс приложения для различных вариантов машины Тьюринга.

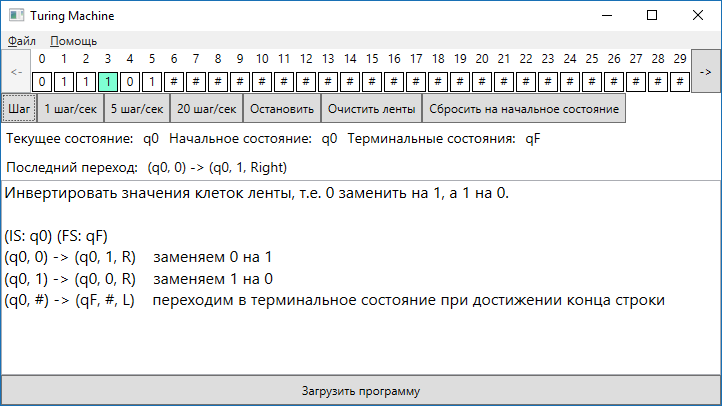


Рисунок 3.1 – Интерфейс приложения с одной лентой

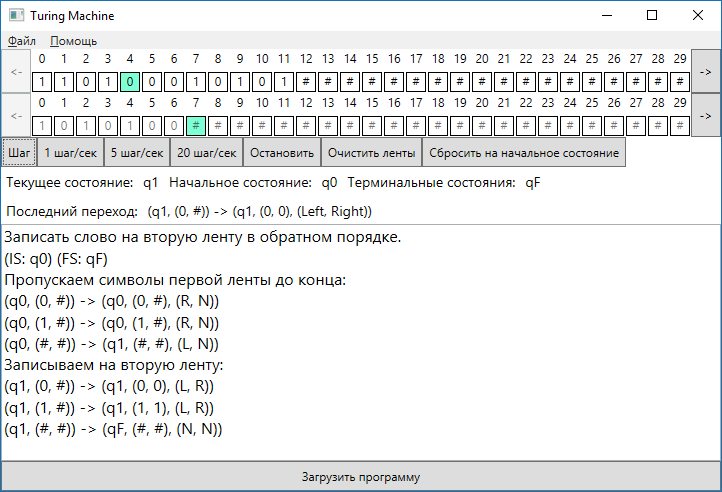


Рисунок 3.2 – Интерфейс приложения с двумя лентами

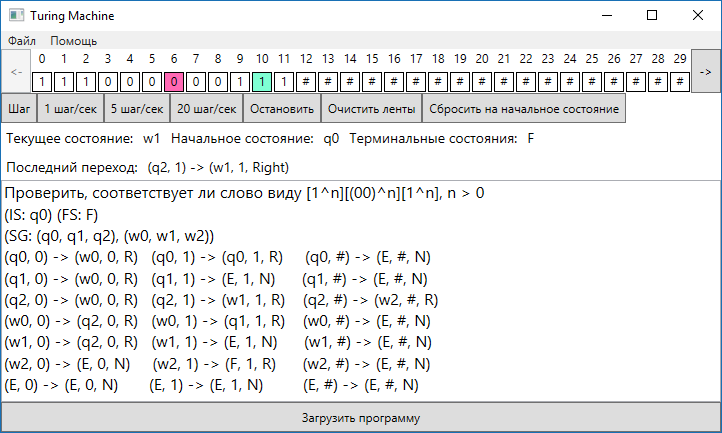


Рисунок 3.3 – Интерфейс приложения с двумя головками

**3.3 Описание языка программ**

Приложение принимает тексты программ для машины Тьюринга, которые написаны в соответствии с формулами 3.1 – 3.7.

Правила перехода задаются с помощью следующего описания.

(3.1)

где

state1 – текущее состояние;

state2 – новое состояние;

a – символ в обозреваемой клетке;

b – новый символ для записи в обозреваемую клетку;

D – направление сдвига головки (L, l – влево, R, r – вправо, N, n – продолжать обозревать текущую клетку).

В случае нескольких лент описание правил перехода принимает следующий вид.

(3.2)

где

ak, b­­k – символы k-й ленты;

Dk – направление сдвига k-й головки.

Также доступны специальные описания, позволяющие уточнить правила работы машины Тьюринга.

1) Задание начальной конфигурации ленты.

(3.3)

где

ak – символ в k-й клетке ленте.

2) Задание начального состояния.

(3.4)

где

state – начальное состояние.

3) Задание терминальных состояний.

(3.5)

где

state1, state2, …, stateN – терминальные состояния.

4) Задание пустого символа.

(3.6)

где

# - пустой символ.

5) Задание групп состояний головок для многоголовочной МТ.

(3.7)

где

sk1, sk2, …, skl – состояния k-й головки.

Кроме того, в любом месте текста программы вне описаний допустим любой другой текст, что позволяет писать комментарии и описывать назначение программы.

**3.4 Реализация**

**3.4.1 Лента**

Каждая панель ленты (рисунок 3.4) является объектом класса TapePanel.



Рисунок 3.4 – Панель ленты

Этот класс отвечает за отображение ленты, как части графического интерфейса. В конструкторе этого класса создаются элементы пользовательского интерфейса, и привязывается объект класса Tape, отвечающего за логику ленты. Данный класс содержит методы, позволяющие взаимодействовать с интерфейсом: вводить ленточные символы, прокручивать ленту, считывать символы из текстовых полей и записывать их.

Класс Tape содержит список символов ленты, а также поля и методы для работы с ним. Этот класс позволяет получить или изменить обозреваемый символ ленты и его позицию для каждой из головок, очистить ленту, узнать номер активной в текущий момент головки, задать пустой символ и другое.

Сама лента представлена в виде списка символов: List<char> tape. При создании объекта этот список заполняется пустыми символами. Значение пустого символа хранит поле char BlankSymbol. Количество таких пустых символов на ленте задается константой int initialLength. Эта константа используется объектом класса TapePanel, чтобы создавать только необходимое количество элементов интерфейса для клеток ленты. Кроме того, заполняется список позиций головок на ленте – List<int> positions. Позиция каждой головки в начальный момент времени равна 0.

Основным методом класса является метод Move(int head, char symbol, Shift shift), который позволяет совершить сдвиг головки с номером head, записав в обозреваемую ей клетку символ char и переместить головку в направлении shift. Если головка находится в позиции, соответствующей значению 0 в списке позиций, то позиция остается равной 0, но увеличивается значение переменной Offset, назначение которой – хранить номер элемента в списке символов ленты, соответствующего клетке на ленте с номером 0. Таким образом, клетки ленты, соответствующие символам в списке символов с номером, меньшим Offset, имеют отрицательный индекс. В остальных случаях значение позиции головки просто изменяется на соответствующее значение.

Методы Position и CurrentSymbol возвращают соответственно позицию и обозреваемый символ активной головки, номер которой хранит поле ActiveHead.

Общее количество головок на ленте задается значением поля NumberOfHeads.

**3.4.2 Машина**

Класс Machine отвечает за большую часть работы имитатора. Этот класс содержит информацию о текущем (поле string CurrentState), начальном (поле string InitialState) и терминальных (список List<string> FinalStates) состояниях машины, об остановке машины (поле bool Stopped), о последнем выполненном переходе (поле string LastTransition), количество лент (поле int NumberOfTapes), список самих лент (List<Tape> tapes), тип машины MachineType MType (многоленточная или многоголовочная).

Для случая многоголовочной МТ есть поле List<List<string>> statesGroups. Элемент этого списка – это список состояний головки с соответствующим номером.

Правила перехода хранятся в виде словаря, имеющего следующую структуру. Dictionary<Tuple<string, string>, Tuple<string, List<char>, List<Shift>>> transitions. Здесь каждой двойке, содержащей текущее состояние и строку символов всех лент, обозреваемых в данный момент, соответствует тройка, содержащая новое состояние, список новых символов для записи на ленту и список направлений сдвига головок.

Метод Step() считывает символы ленты, вычисляет функцию переходов, то есть извлекает данные из словаря transitions, если это возможно (если такая запись есть в словаре), и совершает необходимые действия, чтобы машина выполнила один шаг своей работы, если она еще не остановилась. Здесь сохраняется информация о переходе для вывода пользователю и для каждой ленты вызывается метод Move() со значениями, полученными из словаря правил перехода. И останавливается, если не был задан переход, или было достигнуто терминальное состояние.

Также этот класс содержит метод UpdateProgramText(string text), который обрабатывает текст программы для машины Тьюринга. При обработке осуществляется поиск написанных по определенным правилам описаний, описанных в разделе 3.3, с помощью регулярных выражений и выполнение соответствующих действий. Например, при нахождении правила перехода происходит добавление соответствующей записи в словарь переходов transitions. Здесь же происходит вычисление количества лент и определение вида машины. Если задано описание групп состояний, то это многоголовочная МТ, и лента одна. В противном случае, МТ – многоленточная, и количество лент определяется по первому описанию правила перехода.

**3.4.3 Окна приложения**

Выполнение программы начинается с конструктора класса MainWindow.

Этот класс инициализирует приложение. Он содержит элементы графического интерфейса главного окна и методы, связывающие эти элементы с классами, отвечающими за логику работы приложения.

Сначала создаются экземпляр класса Machine и пустой список панелей ленты. При загрузке новой программы для МТ происходит сброс всех прошлых изменений, добавление необходимого числа панелей ленты и их обновление (это необходимо, когда задана начальная конфигурация ленты). Дальше управление приложением осуществляется с помощью элементов интерфейса (например, кнопок), для каждого из которых описаны соответствующие методы.

Например, при нажатии на кнопку «Шаг» вызывается метод step(), который выполняет следующие действия. Если машина остановилась, то будет выведено сообщение об этом. Если нет, то происходит считывание значений клеток ленты, вызов метода Step() класса Machine, который выполнит все необходимые вычисления, после чего панели лент будут обновлены, то есть будут выведены новые значения клеток, а головки сдвинутся в нужном направлении. На информацинной панели отобразится новое состояние и информация о последнем переходе.

Кроме того, главное окно имеет меню, имеющее пункты для сохранения и загрузки программы (рисунок 3.5), а также для открытия окна со справкой.

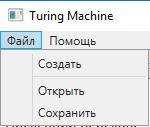


Рисунок 3.5 – Меню приложения

Окно со справкой описано в классе Help. Оно содержит текстовое поле, содержащую краткое описание языка программ для МТ (рисунок 3.6).

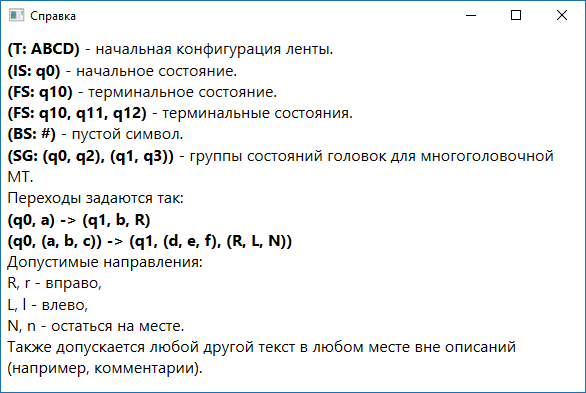


Рисунок 3.6 – Окно справки

**3.5 Описание работы приложения**

Главное окно приложения состоит из нескольких панелей.

Панели ленты содержат поля ввода, соответствующие клеткам ленты. Поля первой ленты можно редактировать. При вводе символа фокус переходит к следующему полю. Если это поле последнее, то лента прокручивается на одну клетку вправо. При нажатии на клавишу Backspace символ в клетке заменяется на пустой, и фокус переходит к предыдущему полю. Кроме того, перемещаться по клеткам ленты с помощью клавиш-стрелок «Влево» и «Вправо». Для остальных лент поля ввода недоступны.

Клетка, обозреваемая головкой в данный момент, подсвечивается цветом. В случае многоголовочной МТ каждая головка выделяется своим цветом.

Над полями ввода находятся поля с текстом, содержащие номер соответствующей клетки ленты.

Справа и слева на панели находятся кнопки прокрутки ленты. При нажатии происходит смещение отображаемой части ленты на одну клетку в соответствующую сторону. Изначально кнопка прокрутки влево недоступна, так как записывать символы на ленту слева может только головка машины.

Ниже панелей лент находится панель с кнопками. Кнопка «Шаг» позволяет выполнить один шаг работы машины Тьюринга. Если машина достигла терминального состояния, то при нажатии появляется сообщение об этом. Если нет, то происходит поиск соответствующего перехода в словаре. Если он найден, то этот переход будет выполнен. В противном случае появится соответствующее сообщение.

Кнопки «1 шаг/сек», «5 шаг/сек», «20 шаг/сек» позволяют автоматически выполнять программу с соответствующей скоростью при помощи таймера. Кнопка «Остановить» останавливает автоматическое выполнение программы. Кнопка «Очистить ленты» заменяет все символы всех лент на пустой символ. Кнопка «Сбросить на начальное состояние» устанавливает текущее состояние равным начальному.

Информационная панель выводит информацию о машине.

Панель ввода программы содержит поле ввода для текста программы и кнопку «Загрузить программу», при нажатии на которую происходит загрузка новой программы, текст которой содержится в поле выше.

Меню содержит пункты «Сохранить» и «Загрузить», которые позволяют выполнить соответствующие действия. Приложение работает с текстовыми файлами. По умолчанию предлагается расширение «.turm», однако оно не является обязательным. Пункт «Справка» открывает окно, содержащее информацию о языке для описания программ.

**ЗАКЛЮЧЕНИЕ**

В результате работы было разработано приложение, позволяющее имитировать работу машины Тьюринга, в том числе, многоленточной и многоголовочной. Кроме того, были изучены основы работы с языком C# и системой Windows Presentation Foundation.

Хотя приложение обладает всеми необходимыми функциями для имитации машины Тьюринга, оно может быть улучшено за счет добавления многих других полезных функций. В частности, было бы полезно, если бы приложение позволяло работать с диаграммами и таблицами переходов.

Приложение может быть использовано, например, в экспериментах, а также в учебном процессе.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 Michael R. Garey, David S. Johnson. Computers and Intractability; A Guide to the Theory of NP-Completeness. – W. H. Freeman & Co., 1979. – 338 с.

2 John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. Introduction to Automata Theory, Languages, and Computation (3rd ed.). – Pearson, 2013. – 488 с.

3 John Watrous. Introduction to the Theory of Computing [Электронный ресурс]. – URL: <https://cs.uwaterloo.ca/~watrous/CS360.Spring2017/> (дата обращения 15.10.2018 г.).

4 Rajendra Kumar. Theory of Automata Languages and Computation. – Tata McGraw-Hill, 2010. – 421 с.

5 ECS120: Introduction to the Theory of Computation, Discussion notes 11/15/06 [Электронный ресурс]. – URL: <https://www.cs.usfca.edu/~sjengle/ucdavis/ecs120-f06/notes.html> (дата обращения 23.10.2018 г.).

6 Matthew MacDonald. Pro WPF 4.5 in C#: Windows Presentation Foundation in .NET 4.5. – Apress, 2012. – 850 с.

7 Andrew Troelsen, Philip Japikse. C# 6.0 and the .NET 4.6 Framework. – Apress, 2015. – 1625 с.

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

MainWindow.xaml.cs:

using System;

using System.Collections.Generic;

using System.Windows;

using Microsoft.Win32;

using System.IO;

using System.Windows.Threading;

namespace TuringMachine

{

enum Shift

{

Left,

Right,

No

}

enum MachineType

{

Multitape,

Multihead

}

public partial class MainWindow : Window

{

private Machine machine = new Machine();

private List<TapePanel> tapePanels = new List<TapePanel>();

DispatcherTimer timer = new DispatcherTimer();

private MachineType type = MachineType.Multihead;

public MainWindow()

{

InitializeComponent();

initializeTapes();

timer.Tick += new EventHandler(step);

}

private void initializeTapes()

{

tapePanels.Clear();

tapesPanel.Children.Clear();

for (int i = 0; i < machine.NumberOfTapes; i++)

{

tapePanels.Add(new TapePanel(machine.GetTape(i), i != 0));

tapesPanel.Children.Add(tapePanels[i].GetGrid());

tapePanels[i].update();

}

}

private void updateTapes()

{

for (int i = 0; i < machine.NumberOfTapes; i++)

{

tapePanels[i].update();

}

}

private void updateProgram()

{

machine.UpdateProgramText(programText.Text);

initializeTapes();

machine.ResetState();

lastTransitionLabel.Content = "";

stateLabel.Content = machine.CurrentState;

initialStateLabel.Content = machine.InitialState;

finalStatesLabel.Content = string.Join(", ", machine.FinalStates);

}

private void clearTapes()

{

for (int i = 0; i < machine.NumberOfTapes; i++)

{

machine.GetTape(i).Clear();

}

initializeTapes();

updateTapes();

}

private void step(object state = null, EventArgs e = null)

{

if (machine.Stopped)

{

if (timer.IsEnabled)

timer.Stop();

else MessageBox.Show("Переход невозможен, машина достигла терминального состояния.");

}

else if (machine.NumberOfTapes > 0)

{

tapePanels[0].ReadTape();

machine.Step();

updateTapes();

stateLabel.Content = machine.CurrentState;

lastTransitionLabel.Content = machine.LastTransition;

}

}

private void setTimer(int stepsPerSec)

{

timer.Interval = new TimeSpan(0, 0, 0, 0, 1000/stepsPerSec);

timer.Start();

}

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

{

clearTapes();

}

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

{

step();

}

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

{

updateProgram();

}

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

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Turing Machine (\*.turm)|\*.turm|All files (\*.\*)|\*.\*";

if (openFileDialog.ShowDialog() == true)

programText.Text = File.ReadAllText(openFileDialog.FileName);

updateProgram();

}

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

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "Turing Machine (\*.turm)|\*.turm|All files (\*.\*)|\*.\*";

if (saveFileDialog.ShowDialog() == true)

File.WriteAllText(saveFileDialog.FileName, programText.Text);

updateProgram();

}

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

{

programText.Text = "";

updateProgram();

}

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

{

setTimer(1);

}

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

{

timer.Stop();

}

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

{

setTimer(5);

}

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

{

setTimer(20);

}

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

{

machine.ResetState();

stateLabel.Content = machine.CurrentState;

}

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

{

Help helpWindow = new Help();

helpWindow.Show();

}

}

}

MainWindow.xaml:

<Window x:Class="TuringMachine.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TuringMachine"

mc:Ignorable="d"

Title="Turing Machine" Height="550" Width="736" MaxWidth="736" MinWidth="736">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition Height="\*"></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

</Grid.RowDefinitions>

<DockPanel>

<Menu DockPanel.Dock="Top">

<MenuItem Header="\_Файл">

<MenuItem Header="\_Создать" Name="createMenuItem" Click="createMenuItem\_Click"/>

<Separator/>

<MenuItem Header="\_Открыть" Name="openMenuItem" Click="openMenuItem\_Click"/>

<MenuItem Header="\_Сохранить" Name="saveMenuItem" Click="saveMenuItem\_Click"/>

</MenuItem>

<MenuItem Header="\_Помощь">

<MenuItem Header="\_Справка" Name="helpMenuItem" Click="helpMenuItem\_Click"/>

</MenuItem>

</Menu>

</DockPanel>

<StackPanel Name="tapesPanel" Grid.Row="1">

</StackPanel>

<StackPanel Orientation="Horizontal" Grid.Row="2">

<Button Name="stepBtn" Click="stepBtn\_Click" Padding="6">Шаг</Button>

<Button Name="oneStepPerSecBtn" Click="oneStepPerSecBtn\_Click" Padding="6">1 шаг/сек</Button>

<Button Name="fiveStepPerSecBtn" Click="fiveStepPerSecBtn\_Click" Padding="6">5 шаг/сек</Button>

<Button Name="twentyStepPerSecBtn" Click="twentyStepPerSecBtn\_Click" Padding="6">20 шаг/сек</Button>

<Button Name="stopBtn" Click="stopBtn\_Click" Padding="6">Остановить</Button>

<Button Name="clearTapesBtn" Click="clearTapesBtn\_Click" Padding="6">Очистить ленты</Button>

<Button Name="resetStateBtn" Click="resetStateBtn\_Click" Padding="6">Сбросить на начальное состояние</Button>

</StackPanel>

<StackPanel Orientation="Vertical" Grid.Row="3">

<StackPanel Orientation="Horizontal">

<Label FontSize="14">Текущее состояние:</Label>

<Label FontSize="14" Name="stateLabel"></Label>

<Label FontSize="14">Начальное состояние:</Label>

<Label FontSize="14" Name="initialStateLabel"></Label>

<Label FontSize="14">Терминальные состояния:</Label>

<Label FontSize="14" Name="finalStatesLabel"></Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label FontSize="14">Последний переход:</Label>

<Label FontSize="14" Name="lastTransitionLabel"></Label>

</StackPanel>

</StackPanel>

<TextBox Grid.Row="4" FontSize="16" VerticalAlignment="Stretch" Name="programText" TextWrapping="NoWrap" AcceptsReturn="True" VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Auto" xml:space="preserve"></TextBox>

<Button Name="updateProgramTextBtn" Click="updateProgramTextBtn\_Click" Grid.Row="6" Padding="6">Загрузить программу</Button>

</Grid>

</Window>

Help.xaml.cs:

using System.Windows;

namespace TuringMachine

{

public partial class Help : Window

{

public Help()

{

InitializeComponent();

}

}

}

Help.xaml:

<Window x:Class="TuringMachine.Help"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TuringMachine"

mc:Ignorable="d"

Title="Справка" Height="400" Width="600">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="\*"></RowDefinition>

</Grid.RowDefinitions>

<TextBlock Grid.Row="0" xml:space="preserve" TextWrapping="Wrap" Padding="6" FontSize="16"><Bold>(T: ABCD)</Bold> - начальная конфигурация ленты.

<Bold>(IS: q0)</Bold> - начальное состояние.

<Bold>(FS: q10)</Bold> - терминальное состояние.

<Bold>(FS: q10, q11, q12)</Bold> - терминальные состояния.

<Bold>(BS: #)</Bold> - пустой символ.

<Bold>(SG: (q0, q2), (q1, q3))</Bold> - группы состояний головок для многоголовочной МТ.

Переходы задаются так:

<Bold>(q0, a) -> (q1, b, R)</Bold>

<Bold>(q0, (a, b, c)) -> (q1, (d, e, f), (R, L, N))</Bold>

Допустимые направления:

R, r - вправо,

L, l - влево,

N, n - остаться на месте.

Также допускается любой другой текст в любом месте вне описаний (например, комментарии).

</TextBlock>

</Grid>

</Window>

Machine.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Text.RegularExpressions;

namespace TuringMachine

{

class Machine

{

public string InitialState { get; set; }

public List<string> FinalStates { get; set; } = new List<string>();

public string CurrentState { get; private set; }

private List<Tape> tapes = new List<Tape>();

public int NumberOfTapes { get; private set; } = 0;

private Dictionary<Tuple<string, string>, Tuple<string, List<char>, List<Shift>>> transitions = new Dictionary<Tuple<string, string>, Tuple<string, List<char>, List<Shift>>>();

public string LastTransition { get; private set; }

public bool Stopped { get; private set; } = false;

public MachineType MType { get; private set; } = MachineType.Multihead;

private List<List<string>> statesGroups = new List<List<string>>();

public Machine()

{

}

public void Step()

{

if (!Stopped)

{

int head = 0;

if (MType == MachineType.Multihead)

{

for (var j = 0; j < statesGroups.Count; j++)

{

if (statesGroups[j].Contains(CurrentState))

{

head = j;

}

}

tapes[0].ActiveHead = head;

}

var tapesSymbols = new List<char>();

foreach (Tape tape in tapes)

{

tapesSymbols.Add(tape.CurrentSymbol);

}

Tuple<string, List<char>, List<Shift>> transition;

if (transitions.TryGetValue(Tuple.Create(CurrentState, string.Concat(tapesSymbols)), out transition))

{

if (NumberOfTapes > 1)

LastTransition = "(" + CurrentState + ", (" + string.Join(", ", tapesSymbols) + ")) -> (" + transition.Item1 + ", (" + string.Join(", ", transition.Item2) + "), (" + string.Join(", ", transition.Item3) + "))";

else

LastTransition = "(" + CurrentState + ", " + tapesSymbols[0] + ") -> (" + transition.Item1 + ", " + transition.Item2[0] + ", " + transition.Item3[0] + ")";

for (var i = 0; i < NumberOfTapes; i++)

{

tapes[i].Move(head, transition.Item2[i], transition.Item3[i]);

}

CurrentState = transition.Item1;

}

else

{

LastTransition = "-";

MessageBox.Show("Переход не задан.");

Stopped = true;

}

if (FinalStates.Contains(CurrentState))

{

MessageBox.Show("Достигнуто конечное состояние (" + CurrentState + ").");

Stopped = true;

}

}

}

public void UpdateProgramText(string text)

{

Regex initialStateRegex = new Regex(@"\(IS: (\w+)\)");

Match initialStateMatch = initialStateRegex.Match(text);

InitialState = initialStateMatch.Groups[1].Value;

if (InitialState == "") InitialState = "S0";

Regex finalStatesRegex = new Regex(@"\(FS: (\w+(?:, \w+){0,})\)");

Match finalStatesMatch = finalStatesRegex.Match(text);

FinalStates.Clear();

FinalStates.AddRange(finalStatesMatch.Groups[1].Value.Split(new string[] { ", " }, StringSplitOptions.None));

Regex blankSymbolRegex = new Regex(@"\(BS: (.)\)");

Match blankSymbolMatch = blankSymbolRegex.Match(text);

char emptySymbol = '#';

if (blankSymbolMatch.Groups[1].Value != "") emptySymbol = blankSymbolMatch.Groups[1].Value[0];

Regex initialTapeRegex = new Regex(@"\(T: (\w+)\)");

Match initialTapeMatch = initialTapeRegex.Match(text);

string initialTapeSymbols = initialTapeMatch.Groups[1].Value;

Regex statesGroupsRegex = new Regex(@"\(SG: \((\w+(?:, \w+){0,})\)(?:, \((\w+(?:, \w+){0,})\)){0,}\)");

Match statesGroupsMatch = statesGroupsRegex.Match(text);

if (statesGroupsMatch.Groups.Count > 1)

{

MType = MachineType.Multihead;

statesGroups.Clear();

statesGroups.Add(statesGroupsMatch.Groups[1].Value.Split(new string[] { ", " }, StringSplitOptions.None).ToList());

if (statesGroupsMatch.Groups[2].Value != "")

{

foreach (Capture capture in statesGroupsMatch.Groups[2].Captures)

{

statesGroups.Add(capture.Value.Split(new string[] { ", " }, StringSplitOptions.None).ToList());

}

}

NumberOfTapes = 1;

}

else MType = MachineType.Multitape;

transitions.Clear();

Regex regex = new Regex(@"\((\w+), (?:\((.(?:, .){0,})\)|(.))\) -> \((\w+), (?:\((.(?:, .){0,})\)|(.)), (?:\(([LlRrNn](?:, [LlRrNn]){0,})\)|([LlRrNn]))\)");

MatchCollection matches = regex.Matches(text);

if (MType == MachineType.Multitape)

{

if (matches.Count > 0)

NumberOfTapes = (matches[0].Groups[3].Value != "") ? 1 : matches[0].Groups[2].Value.Split(new string[] { ", " }, StringSplitOptions.None).Length;

else NumberOfTapes = 0;

}

foreach (Match match in matches)

{

string state1 = match.Groups[1].Value;

string symbols1 = (match.Groups[3].Value != "") ? match.Groups[3].Value : string.Concat(match.Groups[2].Value.Split(new string[] { ", " }, StringSplitOptions.None));

string state2 = match.Groups[4].Value;

List<char> symbols2 = new List<char>();

symbols2.AddRange((match.Groups[6].Value != "") ? match.Groups[6].Value : string.Concat(match.Groups[5].Value.Split(new string[] { ", " }, StringSplitOptions.None)));

List<Shift> shifts = new List<Shift>();

string dirs = (match.Groups[8].Value != "") ? match.Groups[8].Value.ToLower() : match.Groups[7].Value.ToLower();

for (int i = 0; i < dirs.Length; i += 3)

{

if (dirs[i] == 'r') shifts.Add(Shift.Right);

else if (dirs[i] == 'l') shifts.Add(Shift.Left);

else shifts.Add(Shift.No);

}

if (symbols1.Length == NumberOfTapes && symbols2.Count == NumberOfTapes && shifts.Count == NumberOfTapes)

transitions.Add(Tuple.Create(state1, symbols1), Tuple.Create(state2, symbols2, shifts));

}

tapes.Clear();

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

{

if (MType == MachineType.Multihead)

tapes.Add(new Tape(statesGroups.Count, emptySymbol, initialTapeSymbols));

else if (i == 0) tapes.Add(new Tape(1, emptySymbol, initialTapeSymbols));

else tapes.Add(new Tape(1, emptySymbol));

}

}

public void ResetState()

{

Stopped = false;

CurrentState = InitialState;

}

public Tape GetTape(int number)

{

return tapes[number];

}

}

}

TapePanel.cs:

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

namespace TuringMachine

{

class TapePanel

{

private Grid grid = new Grid();

private Tape tape;

private StackPanel labelsPanel = new StackPanel() { Orientation = Orientation.Horizontal };

private StackPanel cellsPanel = new StackPanel() { Orientation = Orientation.Horizontal };

private Button moveLeftButton;

private Button moveRightButton;

private int startSymbol = 0;

private char cellSymbol;

private static readonly Brush[] colors = { Brushes.Aquamarine, Brushes.HotPink, Brushes.DarkViolet, Brushes.Tomato, Brushes.CornflowerBlue };

public TapePanel(Tape tape, bool disabled = false)

{

this.tape = tape;

cellSymbol = tape.BlankSymbol;

grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = new GridLength(30) });

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

grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = new GridLength(30) });

grid.RowDefinitions.Add(new RowDefinition());

grid.RowDefinitions.Add(new RowDefinition());

Grid.SetRow(labelsPanel, 0);

Grid.SetColumn(labelsPanel, 1);

grid.Children.Add(labelsPanel);

Grid.SetRow(cellsPanel, 1);

Grid.SetColumn(cellsPanel, 1);

grid.Children.Add(cellsPanel);

moveLeftButton = new Button()

{

Content = "<-"

};

Grid.SetRow(moveLeftButton, 0);

Grid.SetColumn(moveLeftButton, 0);

Grid.SetRowSpan(moveLeftButton, 2);

moveLeftButton.Click += MoveLeftButton\_Click;

grid.Children.Add(moveLeftButton);

moveRightButton = new Button()

{

Content = "->"

};

Grid.SetRow(moveRightButton, 0);

Grid.SetColumn(moveRightButton, 2);

Grid.SetRowSpan(moveRightButton, 2);

moveRightButton.Click += MoveRightButton\_Click;

grid.Children.Add(moveRightButton);

for (int i = 0; i < Tape.initialLength; i++)

{

labelsPanel.Children.Add(new Label() {

Width = 20,

Height = 20,

Padding = new Thickness(0),

Margin = new Thickness(1),

HorizontalContentAlignment = HorizontalAlignment.Center

});

cellsPanel.Children.Add(new Border()

{

BorderBrush = Brushes.Black,

BorderThickness = new Thickness(1),

Width = 20,

Height = 20,

Margin = new Thickness(1),

Child = new TextBox

{

TextAlignment = TextAlignment.Center,

MaxLength = 1,

BorderThickness = new Thickness(0),

Background = Brushes.Transparent,

CaretBrush = Brushes.Transparent

}

});

TextBox box = getTextBox(i);

if (disabled) box.IsEnabled = false;

box.GotFocus += cell\_GotFocus;

box.LostFocus += cell\_LostFocus;

box.KeyUp += cell\_KeyUp;

}

}

protected void moveRight()

{

ReadTape();

if (startSymbol + Tape.initialLength - 1 == tape.Top) tape.Increase();

startSymbol++;

update();

}

protected void moveLeft()

{

ReadTape();

if (startSymbol > tape.Bottom) startSymbol--;

update();

}

public void ReadTape()

{

for (int i = 0; i < Tape.initialLength; i++)

{

tape.SetSymbol(i + startSymbol, (getTextBox(i).Text != "") ? getTextBox(i).Text[0] : cellSymbol);

}

}

public void update()

{

for (int j = 0; j < Tape.initialLength; j++)

{

(cellsPanel.Children[j] as Border).Background = Brushes.White;

getTextBox(j).Text = tape.GetSymbol(j + startSymbol).ToString();

(labelsPanel.Children[j] as Label).Content = (j + startSymbol).ToString();

}

for (var i = 0; i < tape.NumberOfHeads; i++)

{

var index = tape.GetPosition(i) - tape.Offset - startSymbol;

if (index >= 0 && index < Tape.initialLength) (cellsPanel.Children[index] as Border).Background = colors[i % 5];

}

if (startSymbol == tape.Bottom) moveLeftButton.IsEnabled = false;

else moveLeftButton.IsEnabled = true;

}

protected TextBox getTextBox(int index)

{

return (cellsPanel.Children[index] as Border).Child as TextBox;

}

public Grid GetGrid()

{

return grid;

}

protected void MoveLeftButton\_Click(object sender, RoutedEventArgs e)

{

moveLeft();

}

protected void MoveRightButton\_Click(object sender, RoutedEventArgs e)

{

moveRight();

}

protected void cell\_KeyUp(object sender, KeyEventArgs e)

{

TextBox box = sender as TextBox;

FocusNavigationDirection direction;

switch (e.Key)

{

case Key.Back:

direction = FocusNavigationDirection.Left;

box.Text = tape.BlankSymbol.ToString();

break;

case Key.Left:

direction = FocusNavigationDirection.Left;

box.Text = cellSymbol.ToString();

break;

case Key.Right:

direction = FocusNavigationDirection.Right;

box.Text = cellSymbol.ToString();

break;

default:

direction = FocusNavigationDirection.Right;

break;

}

if (box.Text != "")

{

TraversalRequest request = new TraversalRequest(direction);

UIElement elementWithFocus = Keyboard.FocusedElement as UIElement;

if (elementWithFocus != null)

{

if (elementWithFocus.PredictFocus(direction) is TextBox)

{

elementWithFocus.MoveFocus(request);

}

else

{

if (direction == FocusNavigationDirection.Right) moveRight();

else if (direction == FocusNavigationDirection.Left) moveLeft();

(elementWithFocus as TextBox).Text = "";

}

}

}

}

protected void cell\_GotFocus(object sender, RoutedEventArgs e)

{

TextBox box = sender as TextBox;

cellSymbol = (box.Text.Length == 1) ? box.Text[0] : tape.BlankSymbol;

box.Background = new SolidColorBrush(Color.FromArgb(127, 200, 200, 200));

box.Text = "";

}

protected void cell\_LostFocus(object sender, RoutedEventArgs e)

{

TextBox box = sender as TextBox;

box.Background = Brushes.Transparent;

if (box.Text == "") box.Text = cellSymbol.ToString();

}

}

}

Tape.cs:

using System.Collections.Generic;

namespace TuringMachine

{

class Tape

{

public char BlankSymbol { get; set; }

public const int initialLength = 30;

protected List<char> tape = new List<char>();

public int Position { get { return positions[ActiveHead]; } private set { positions[ActiveHead] = value; } }

public char CurrentSymbol { get { return tape[positions[ActiveHead]]; } set { tape[positions[ActiveHead]] = value; } }

public int Top { get { return tape.Count - 1 - Offset; } }

public int Bottom { get { return -Offset; } }

public int Offset { get; private set; } = 0;

public int NumberOfHeads;

private List<int> positions = new List<int>();

public int ActiveHead { private get; set; } = 0;

public Tape(int numberOfHeads, char emptySymbol = '#', string strTape = "")

{

BlankSymbol = emptySymbol;

NumberOfHeads = numberOfHeads;

tape.AddRange(string.Concat(strTape));

for (var i = tape.Count; i < initialLength; i++) tape.Add(BlankSymbol);

for (var i = 0; i < NumberOfHeads; i++)

{

positions.Add(0);

}

}

public void Clear()

{

tape.Clear();

for (var i = tape.Count; i < initialLength; i++) tape.Add(BlankSymbol);

for (var i = 0; i < NumberOfHeads; i++) positions[i] = 0;

Offset = 0;

}

public void Move(int head, char symbol, Shift shift)

{

CurrentSymbol = symbol;

ActiveHead = head;

switch (shift)

{

case Shift.Left:

if (Position == 0)

{

tape.Insert(0, BlankSymbol);

Offset++;

}

else Position--;

break;

case Shift.Right:

if (Position == tape.Count - 1)

Increase();

Position++;

break;

}

}

public int GetPosition(int head)

{

if (head < NumberOfHeads)

{

return positions[head];

}

else return 0;

}

public void Increase()

{

tape.Add(BlankSymbol);

}

public char GetSymbol(int pos)

{

if (pos >= Bottom && pos <= Top)

return tape[pos + Offset];

else return BlankSymbol;

}

public void SetSymbol(int pos, char symbol)

{

tape[Offset + pos] = symbol;

}

}

}