Московский Авиационный институт

(Национальный исследовательский университет)

Институт №8

«Компьютерные науки и прикладная математика»

Кафедра 806

«Вычислительная математика и программирование»

Курсовая работа по дисциплине
«Теория автоматов и формальных языков»

Тема: «Конечный автомат для моделирования поведения хомяка»

Студент: Буреева П.С.

Группа: М80-411Б-19

Преподаватель: Лемтюжникова Д.В.

Оценка:

Дата:

Москва 2022

Оглавление

Введение 3

Теоретическая часть 4

Практическая часть 6

- **Техническое задание** 6
- **Описание состояний** 6
- **Граф состояний** 7
- **Описание условий перехода между состояниями** 8
- **Визуализация** 8

Вывод 12

Список использованные источников 13

Приложение 14

Введение

Теория автоматов как научная дисциплина возникла в пределах теории управляющих систем (теоретической кибернетики) в середине XX века, в период начавшегося бурного развития средств электронной вычислительной техники и соответствующих областей математического знания. Потребовалась разработка теоретической базы для решения проблем, возникавших при проектировании реальных цифровых ЭВМ, а также в процессе построения и исследования гипотетических систем, таких как нейронные сети. В качестве моделей последних рассматривались конечные автоматы. Развитие информационных технологий вывело сферу приложения теории автоматов далеко за рамки моделирования аппаратных средств цифровой электроники, расширив ее до фундаментальных основ современной теоретической информатики.

Сегодня абстракции и модели, разработанные в теории автоматов, востребованы такими научными дисциплинами как теория формальных грамматик, математическая лингвистика, теория логических моделей, математическая логика и формальные аксиоматические системы, теория кодирования, теория вычислительной сложности и другими. Наиболее тесно теория автоматов связана с теорией алгоритмов и, в частности, с таким ее разделом как теория абстрактных машин.

Теоретическая часть

Конечный автомат (или попросту FSM — Finite-state machine) это модель вычислений, основанная на гипотетической машине состояний. В один момент времени только одно состояние может быть активным. Следовательно, для выполнения каких-либо действий машина должна менять свое состояние.

Конечные автоматы обычно используются для организации и представления потока выполнения чего-либо. Это особенно полезно при реализации ИИ в играх.

Абстрактным автоматом называют модель, описываемую пятиместным кортежем:

$$A = (X, Y, S, f_y, f_s),$$

где первые три компонента – непустые множества:

- X – множество входных сигналов АА,
- Y – множество выходных сигналов АА,
- S – множество состояний АА.

Два последних компонента кортежа – характеристические функции:

- f_y – функция выходов;
- f_s – функция переходов АА из одного состояния в другое.

Если множества X, Y, S – конечные, то такой АА называют конечным автоматом (КА).

Если хотя бы одно из перечисленных множеств не является конечным, то такой АА называют бесконечным.

Общую схему автомата можно представить в виде некоторого «черного ящика», осуществляющего преобразование вектора входных сигналов в вектор выходных сигналов (рис. 1):

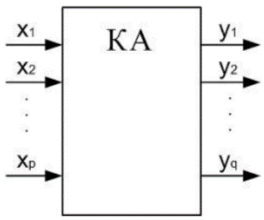

Рисунок 1 Общая схема конечного автомата

Таким образом, можно записать: $Y = f_y(X, t)$.

Фактор времени в приведенном уравнении учитывается введением вектора состояний S, как своего рода «памяти о прошлом». На один и тот же набор входных сигналов (значений компонентов вектора X) автомат будет выдавать разные выходные сигналы (значения компонентов вектора Y) в зависимости от состояния, в котором он находится в данный момент времени. Текущее состояние, в свою очередь, определяется алгоритмом функционирования автомата.

Рассмотрим характеристические функции автомата.

Функция f_s реализует бинарное отношение вида $S \times X \rightarrow S$, то есть каждой паре «состояние – входной сигнал» ставит в однозначное соответствие определенное состояние из множества S. Аналогично, бинарное отношение для функции f_y имеет вид $S \times X \rightarrow Y$, то есть каждой паре «состояние – входной сигнал» ставится в соответствие конкретный выходной сигнал – элемент множества Y.

Таким образом, характеристические функции определяют, в какое состояние $s \in S$ перейдет автомат в следующий, (t+1)-й момент времени и каково будет значение выходного сигнала $y \in Y$ в текущий момент времени t:

$$s(t+1) = f_s\,(x(t), s(t))$$

$$y(t) = f_y\,(x(t), s(t))$$

Из приведенных уравнений видно, что аргументами характеристических функций являются текущее значение входного сигнала и текущее состояние. Конечный автомат, заданный парой уравнений, называется автоматом I рода или, по имени автора модели, автоматом Мили (Mealy).

На практике часто встречаются автоматы, выходные сигналы которых в момент времени t однозначно определяются текущим состоянием автомата и не зависят от компонентов вектора входных сигналов:

$$s'(t+1) = f_s'(x(t+1),$$

$$s'(t))\, y(t) = f_y'(s'(t))$$

Автомат, заданный парой уравнений, называют автоматом II рода или автоматом Мура (Moore). Штрих введен в обозначения для отличия записи функций и состояний автомата Мура от автомата Мили. Заметим, что автомат Мили по отношению к автомату Мура «запаздывает» на один дискретный момент времени по входному сигналу.

Автоматы I и II рода являются двумя базовыми моделями, изучаемыми теорией автоматов.

Практическая часть

Техническое задание

Создать автомат состояний, характеризующих поведение хомяка при условии входных сигналов, отражающих уровень бодрости, сытости и досуга.

Для реализации необходимо определить:

1. Набор состояний, в которых может находится хомяк в каждый момент времени;

2. Условия перехода между состояниями согласно входным сигналам.

Описание состояний

Определим некоторые переменные для дальнейшей работы автомата:

1. Satiety – шкала сытости хомяка.
2. Leisure – шкала досуга хомяка.
3. Sleep – шкала бодрости хомяка.
4. Dislike – переменная, отображающая желание хомяка сбежать.
5. Scares – количество раз, которое хозяин пугал хомяка.

Для выполнения технического задания определим состояния, в которых может находиться хомяк:

1. Initial – базовое и входное состояние хомяка. В нем он оказывается, когда не совершает никакого действия.
2. Eat – в данном состоянии хомяк оправляется к миске и начинает есть. Шкала сытости satiety полностью заполняется до 100. По окончании действия возвращается в состояние Initial.
3. Pat – в данном состоянии хомяк отправляется на крышу домика, где хозяин гладит его в течение 3 секунд реального времени. Шкала досуга leisure увеличивается на 40. Переменная dislike уменьшается на 1. По окончании действия возвращается в состояние Initial.
4. Sleep – в данном состоянии хомяк отправляется спать на 5 секунд реального времени. Шкала бодрости полностью заполняется до 100. По окончании действия возвращается в состояние Initial.
5. Wheel – в данном состоянии хомяк отправляется к колесу и бегает там в течение 3 секунд реального времени. Шкала досуга leisure

увеличивается на 40. По окончании действия возвращается в состояние Initial.

6. Scare – состояние, в котором хомяк боится. Переменные dislike и scare увеличиваются на 1.
   - Если переменная scare = 1, то хомяк бегает по клетке.
   - Если переменная scare = 2, то хомяк бегает прячется в домике.
   - Если переменная scare = 3, то хомяк очень грустит и боится.
7. Hungry – в данном состоянии хомяк просит еды. Переменная dislike увеличивается на 1.
8. Escape – состояние, в котором хомяк сбегает из клетки и игра заканчивается.
9. Dead – состояние смерти. Игра заканчивается.

Для каждого состояния имеется соответствующее изображение, которое отображает его.

Граф состояний

Построим систему S с дискретными состояниями $S_1, \ldots, S_n$. Будем изображать каждое состояние кругом, а возможные переходы из состояния в состояние — стрелками, над которыми будет указано условие перехода (Рис. 2):

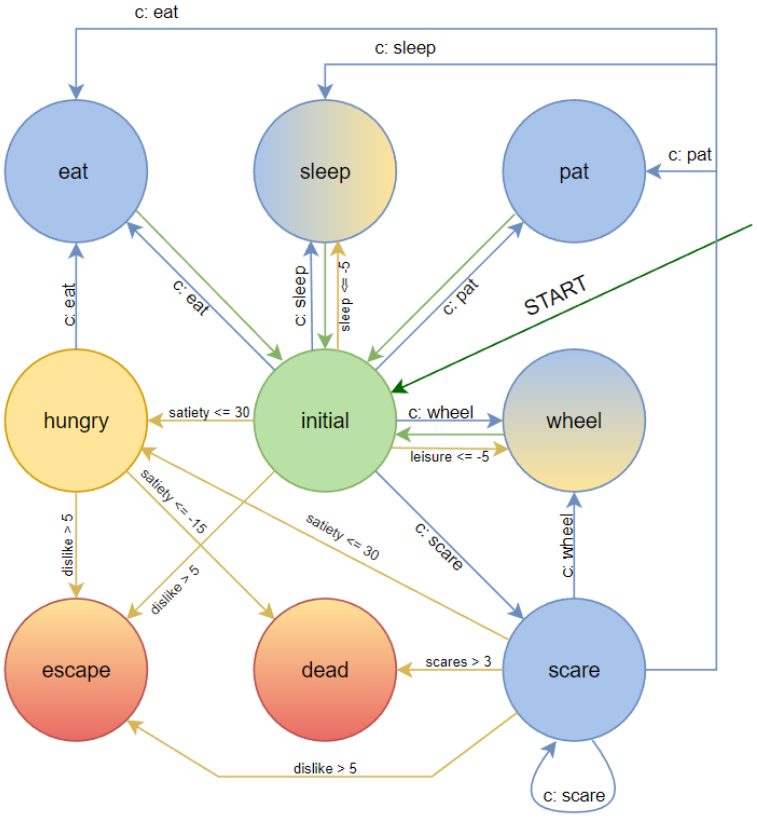

Рисунок 2 Граф состояний Хомяка

Описание условий перехода между состояниями

На графе, изображенном на Рисунке 2 зеленые стрелочки обозначают переход в состоянии Initial по окончании действия, описанном в состоянии. Такой переход может осуществиться только из состояний Eat, Sleep, Pat и Wheel.

Синие стрелочки обозначают команды, которые доступны пользователю приложения:

1. Eat – команда, при выполнении которой хомяк отправляется в состояние Eat. Может быть вызвана только из состояний Initial, Hungry и Scare.
2. Sleep – команда, при выполнении которой хомяк отправляется спать в состояние Sleep. Может быть вызвана только из состояний Initial и Scare.
3. Pat – команда, позволяющая погладить хомяка, он оправляется в состояние Pat. Может быть вызвана из состояний Initial и Scare.
4. Wheel – команда перехода в состояние Wheel из состояний Initial или Scare.
5. Scare – команда, при выполнении которой хомяк пугается и попадает в состояние Scare. Может быть вызвана из состояний Initial и Scare.

Желтые стрелочки обозначают автоматические переходы (не зависящие от пользователя приложения):

1. Если значение шкалы sleep <= -5, совершается переход из состояния Initial в Sleep.
2. Если значение шкалы satiety <= 30, совершается переход из состояния Initial в Hungry или из состояния Scare в Hungry.
3. Если значение шкалы leisure <= -5, совершается переход из состояния Initial в Wheel.
4. Если значение шкалы satiety <= -15, совершается переход из состояния Hungry в Dead.
5. Если значение переменной scares > 3, хомяк переходит в состояние Dead.
6. Если значение переменной dislike > 5, то совершается переход в состояние Escape из одного из трех состояний: Hungry, Initial, Scare.

Визуализация

Разработки пользовательского приложения велась на языке программирования C# с помощью средства создания графических интерфейсов WPF.

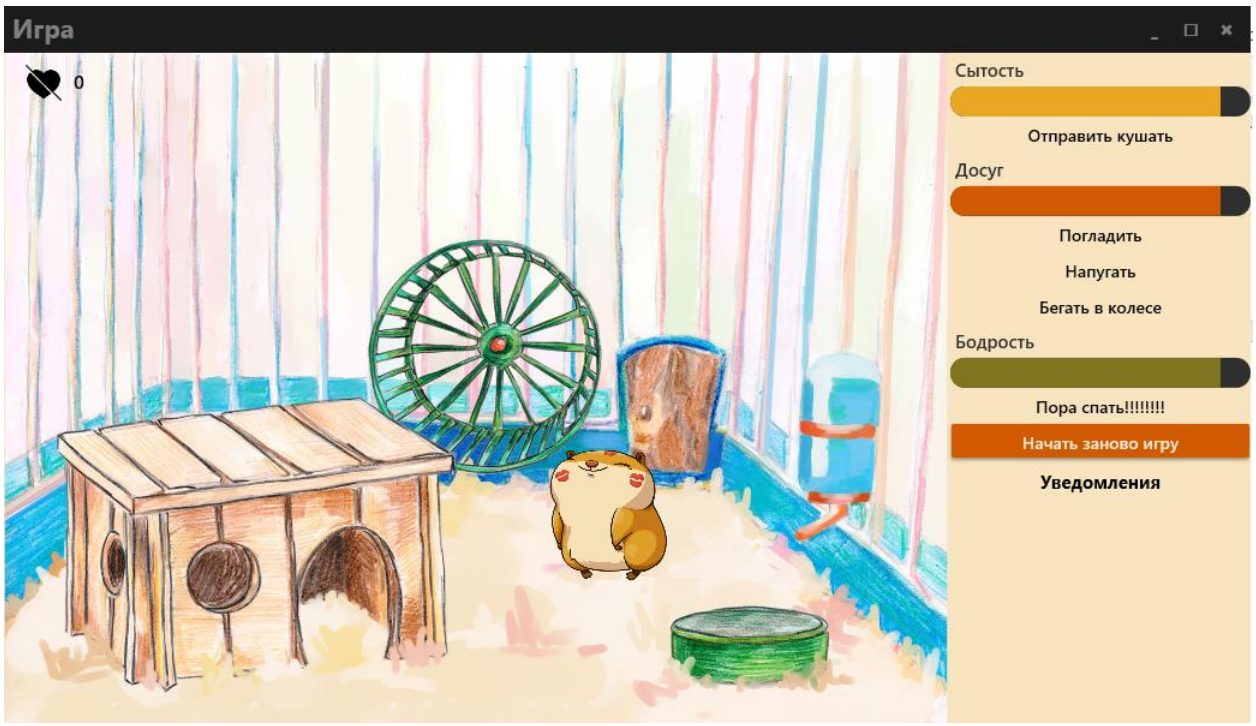

Рисунок 3 Стартовый интерфейс приложения

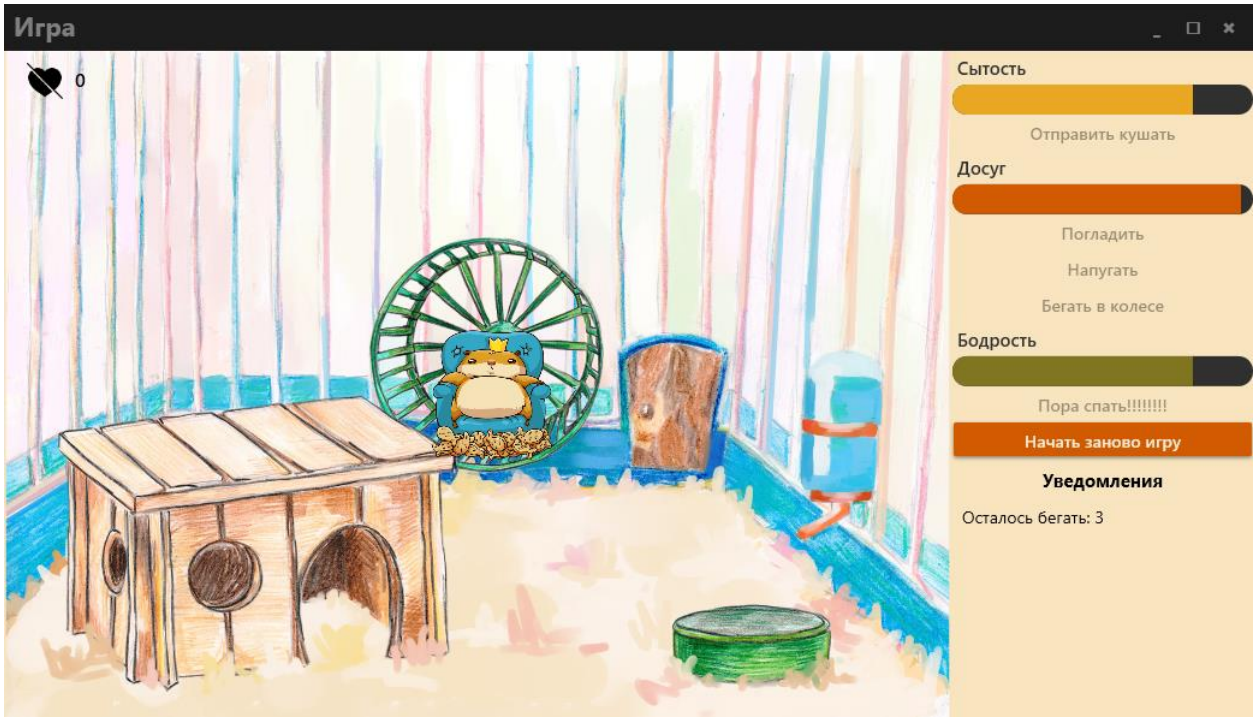

Рисунок 4 Состояние Wheel

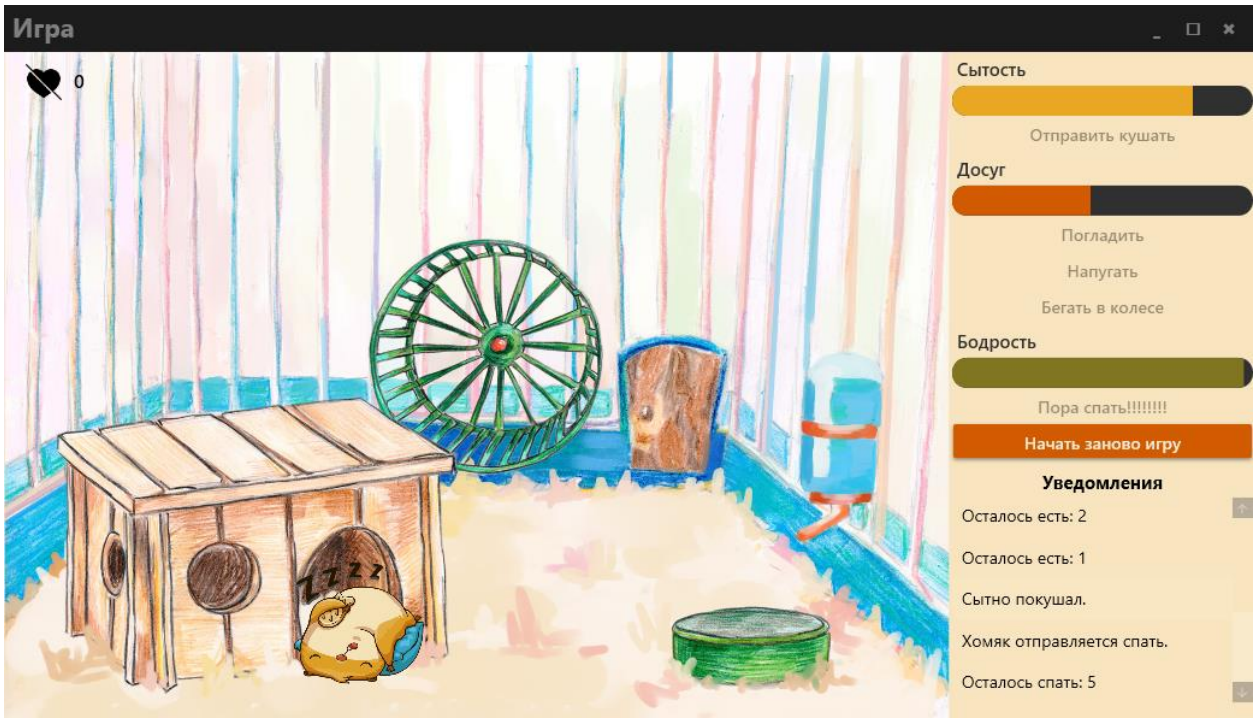

Рисунок 5 Состояние Sleep

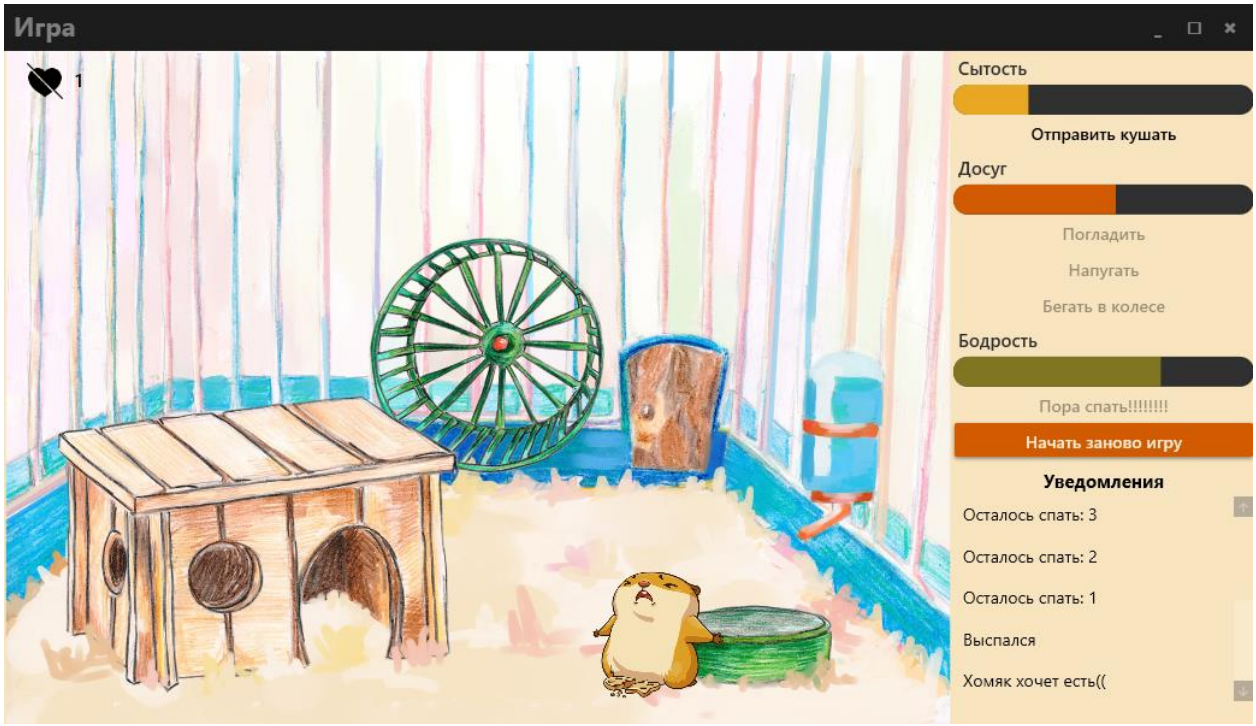

Рисунок 6 Состояние Hungry

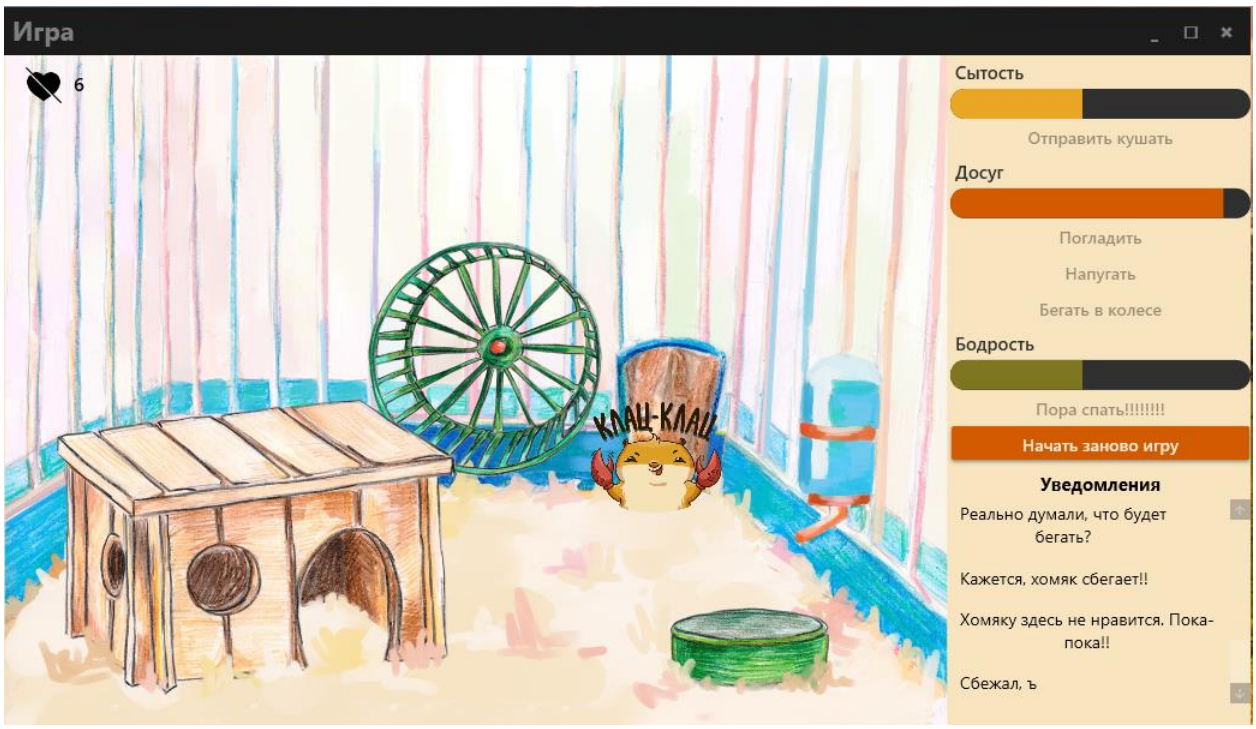

Рисунок 5 Состояние Escape

Вывод

В результате работы над курсовой работой была создана рабочая версия приложения, иллюстрирующего поведение домашнего хомяка.

Были изучены принципы построения и работы автоматов Милли и Мура, а также разработана основная концепция поведения и перемещения хомяка.

Список использованные источников

1. Е.Н. Ишакова Теория формальных языков, грамматик и автоматов [Текст] / Е.Н. Ишакова — Оренбург: Оренбургский государственный университет, 2005 — 55 с
2. Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман. Дискретная математика. — 2-е изд. — Вильямс, 2002. — 528 с. — (Алгоритмы и методы. Искусство программирования).
3. Паттерны объектно-ориентированного проектирования. — СПб.: Питер, 2020. — 448 с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-4461-1595-2
4. Гуренко, В. В. Введение в теорию автоматов [Текст] / В. В. Гуренко — Москва: МГТУ имени Н.Э. Баумана, 2013 — 63 с.

Приложение

HamsterState.cs

```
using System.Threading.Tasks;
using System.Timers;
using coursavt.UserControls;

namespace coursavt.MVVM.Model;

public abstract class HamsterState
{
    public Hamster Hamster { get; set; }
    public abstract Task Enter();

    protected abstract Task Logic();

    public abstract void Exit();
    protected Timer _timer;

    protected string Status
    {
        set { Hamster.InvokeInUiThread(() => Hamster.MessageControls.Add(new
MessageControl(value))); }
    }

    protected HamsterState(Hamster hamster)
    {
        Hamster = hamster;
        _timer = new Timer(1000);
    }

    private int GetY(int x, System.Drawing.Point a, System.Drawing.Point c)
    {
        double k = (double)(a.Y - c.Y) / (double)(a.X - c.X);
        double b = (double)c.Y - (double)(k * c.X);
        return (int)(k * x + b);
    }

    protected async Task Move(System.Drawing.Point currentPoint, System.Drawing.Point
destinationPoint, int speed = 2)
    {
        if (currentPoint.X < destinationPoint.X)
        {
            for (var i = currentPoint.X; i <= destinationPoint.X; i += speed)
            {
                Hamster.Location = new System.Drawing.Point(i, GetY(i, currentPoint,
destinationPoint));
                //await Task.Run(() => { Thread.Sleep(10); });
                await Task.Delay(1);
            }
        }
        else if (currentPoint.X == destinationPoint.X)
        {
            for (var i = currentPoint.Y; i >= 0; i--)
            {
                Hamster.Location = new System.Drawing.Point(currentPoint.X, i);
                //await Task.Run(() => { Thread.Sleep(10); });
                await Task.Delay(1);
            }
```

```
        }
        else
        {
            for (var i = currentPoint.X; i > destinationPoint.X; i -= speed)
            {
                Hamster.Location = new System.Drawing.Point(i, GetY(i, currentPoint,
destinationPoint));
                //await Task.Run(() => { Thread.Sleep(10); });
                await Task.Delay(1);
            }
        }
    }
}
```

Hamster.cs

```
using System;
using System.Collections.ObjectModel;
using System.Timers;
using System.Windows;
using System.Windows.Threading;
using coursavt.Core;
using coursavt.MVVM.Model.Consts;
using coursavt.MVVM.Model.States;
using coursavt.UserControls;
using Point = System.Drawing.Point;

namespace coursavt.MVVM.Model;

public class Hamster : ObservableObject
{
    private Point _location;
    private string _imagePath;
    public System.Timers.Timer Timer;
    private int _hungryPBar;
    private int _sleepPBar;
    private int _leisurePBar;
    private bool _buttonEnabled;
    public byte CountFright;
    private HamsterState _currentState;
    private bool _eatButtonEnabled;
    private bool _isAlive;
    private byte _countEscape;

    public ObservableCollection<MessageControl> MessageControls { get; set; } =
        new ObservableCollection<MessageControl>();

    public byte CountEscape
    {
        get => _countEscape;
        set
        {
            _countEscape = value;
            OnPropertyChanged();
        }
    }

    public bool IsAlive
    {
        get => !_isAlive;
        set
        {
            _isAlive = value;
```

```
            OnPropertyChanged();
        }
    }

    public bool ButtonEnabled
    {
        get => _buttonEnabled;
        set
        {
            _buttonEnabled = value;
            OnPropertyChanged();
        }
    }

    public bool EatButtonEnabled
    {
        get => _eatButtonEnabled;
        set
        {
            _eatButtonEnabled = value;
            OnPropertyChanged();
        }
    }

    public Point Location
    {
        get => _location;
        set
        {
            _location = value;
            OnPropertyChanged();
        }
    }

    public string ImagePath
    {
        get => _imagePath;
        set
        {
            _imagePath = value;
            OnPropertyChanged();
        }
    }

    public int LeisurePBar
    {
        get => _leisurePBar;
        set
        {
            _leisurePBar = value;
            OnPropertyChanged();
        }
    }

    public int HungryPBar
    {
        get => _hungryPBar;
        set
        {
            _hungryPBar = value;
            OnPropertyChanged();
```

```
    }
}

public int SleepPBar
{
    get => _sleepPBar;
    set
    {
        _sleepPBar = value;
        OnPropertyChanged();
    }
}

public Hamster()
{
    ImagePath = HamsterImages.InitImage;
    CountEscape = 0;
    CountFright = 0;
    SleepPBar = 100;
    HungryPBar = 100;
    LeisurePBar = 100;
    Location = new Point(240, 420);
    IsAlive = true;
    _eatButtonEnabled = true;
    Timer = new System.Timers.Timer(500);
    TransitionToState(new InitState(this));
    VitalSigns();
}

private void VitalSigns()
{
    Timer.Elapsed += timer_Tick;
    Timer.Start();
}

private void timer_Tick(Object source, ElapsedEventArgs e)
{
    HungryPBar--;
    SleepPBar--;
    LeisurePBar--;
}

public void TransitionToState(HamsterState state)
{
    _currentState?.Exit();
    _currentState = state;
    _currentState.Enter();
}

/// <summary>
/// Invoke <see cref="Action"/> in default UI thread.
/// </summary>
/// <param name="action"></param>
public void InvokeInUiThread(Action action)
{
    Application.Current.Dispatcher.Invoke(() =>
    {
        try
        {
            action();
        }
```

```
            catch (Exception ex)
            {
                return;
            }
        }, DispatcherPriority.Normal);
    }
}
```

HamsterStateMachine.cs

```
using System;
using coursavt.MVVM.Model.States;

namespace coursavt.MVVM.Model;

public class HamsterStateMachine
{
    public Hamster Hamster { get; set; }

    public HamsterStateMachine(Hamster hamster)
    {
        Hamster = hamster;
    }

    public void ChangeState(HamsterCommand command)
    {
        switch (command)
        {
            case HamsterCommand.EAT:
                GoEat();
                break;
            case HamsterCommand.PAT:
                GoPat();
                break;
            case HamsterCommand.SCARE:
                GoScare();
                break;
            case HamsterCommand.SLEEP:
                GoSleep();
                break;
            case HamsterCommand.WHEEL:
                GoWheel();
                break;
            default:
                throw new ArgumentOutOfRangeException();
        }
    }

    private void GoEat()
    {
        Hamster.TransitionToState(new EatState(Hamster));
    }

    private void GoPat()
    {
        Hamster.TransitionToState(new PatState(Hamster));
    }

    private void GoScare()
    {
        Hamster.TransitionToState(new ScareState(Hamster));
    }
```

```
    private void GoSleep()
    {
        Hamster.TransitionToState(new SleepState(Hamster));
    }

    private void GoWheel()
    {
        Hamster.TransitionToState(new WheelState(Hamster));
    }
}
```

EatState.cs

```
using System;
using System.Threading.Tasks;
using System.Timers;
using coursavt.MVVM.Model.Consts;

namespace coursavt.MVVM.Model.States;

public class EatState : HamsterState
{
    public EatState(Hamster hamster) : base(hamster)
    {
        _eatTime = 3;
    }

    private byte _eatTime;

    public override async Task Enter()
    {
        Hamster.ButtonEnabled = false;
        Hamster.EatButtonEnabled = false;
        Status = "Хомяк отправился кушоц.";
        await Logic();
    }

    protected override async Task Logic()
    {
        Hamster.ImagePath = HamsterImages.InitImage;
        await Move(Hamster.Location, HamsterPoints.EatPoint);
        Hamster.ImagePath = HamsterImages.EatImage;
        _timer.Start();
        _timer.Elapsed += timer_Tick;
        Hamster.HungryPBar = 100;
        Hamster.CountFright = 0;
    }

    public override void Exit()
    {
        Status = "Сытно покушал.";
        Hamster.ButtonEnabled = true;
        Hamster.EatButtonEnabled = true;
    }

    private void timer_Tick(Object source, ElapsedEventArgs e)
    {
        Status = $"Осталось есть: {_eatTime}";
        _eatTime--;
        if (_eatTime != 0) return;
        _timer.Stop();
        _timer.Dispose();
        Hamster.TransitionToState(new InitState(Hamster));
```

```
    }
}
```

EscapeState.cs

```
using System.Threading.Tasks;
using coursavt.MVVM.Model.Consts;

namespace coursavt.MVVM.Model.States;

public class EscapeState : HamsterState
{
    public EscapeState(Hamster hamster) : base(hamster)
    {
    }

    public override async Task Enter()
    {
        Hamster.ButtonEnabled = false;
        Hamster.EatButtonEnabled = false;
        Status = "Кажется, хомяк сбегает!!";
        await Logic();
    }

    protected override async Task Logic()
    {
        Status = "Хомяку здесь не нравится. Пока-пока!!";
        Hamster.Timer.Stop();
        Hamster.IsAlive = false;
        Hamster.ImagePath = HamsterImages.DoorImage;
        await Move(Hamster.Location, HamsterPoints.DoorPoint, 2);
        Hamster.ImagePath = HamsterImages.NibbleImage;
        Exit();
    }

    public override void Exit()
    {
        Status = "Сбежал, ъ";
        Status = "Игра закончена";
    }
}
```

HungryState.cs

```
using System;
using System.Threading.Tasks;
using System.Timers;
using coursavt.MVVM.Model.Consts;

namespace coursavt.MVVM.Model.States;

public class HungryState : HamsterState
{
    public HungryState(Hamster hamster) : base(hamster)
    {
    }

    private string _status = null;

    public override async Task Enter()
    {
        Status = "Хомяк хочет есть((";
        Hamster.ButtonEnabled = false;
        Hamster.EatButtonEnabled = true;
```

```
        Hamster.CountFright = 0;
        Hamster.CountEscape++;
        await Logic();
    }

    protected override async Task Logic()
    {
        if (Hamster.CountEscape > 5)
        {
            _status = "О, нет..";
            Hamster.TransitionToState(new EscapeState(Hamster));
            return;
        }

        _status = "Ура, еда!!!!";
        await Move(Hamster.Location, HamsterPoints.HungryPoint);
        _timer.Start();
        _timer.Elapsed += timer_Tick;
        Hamster.ImagePath = HamsterImages.HungryImage;
    }

    public override void Exit()
    {
        _timer.Stop();
        _timer.Dispose();
        Status = _status;
    }

    private void timer_Tick(Object source, ElapsedEventArgs e)
    {
        if (Hamster.HungryPBar > -10) return;
        _timer.Stop();
        _timer.Dispose();
        Hamster.TransitionToState(new OtkinylsiaState(Hamster));
    }
}
```

InitState.cs

```
using System;
using System.Threading.Tasks;
using System.Timers;
using coursavt.MVVM.Model.Consts;

namespace coursavt.MVVM.Model.States;

public class InitState : HamsterState
{
    public InitState(Hamster hamster) : base(hamster)
    {
        _timer = new Timer(500);
    }

    public override async Task Enter()
    {
        Hamster.ButtonEnabled = false;
        Hamster.EatButtonEnabled = false;
        await Logic();
    }

    protected override async Task Logic()
    {
        Hamster.ImagePath = HamsterImages.InitImage;
```

```
        await Move(Hamster.Location, HamsterPoints.InitPoint);
        _timer.Start();
        _timer.Elapsed += timer_Tick;
        Hamster.ButtonEnabled = true;
        Hamster.EatButtonEnabled = true;
    }

    public override void Exit()
    {
        _timer.Dispose();
    }

    private void timer_Tick(Object source, ElapsedEventArgs e)
    {
        if (Hamster.CountEscape > 5)
        {
            _timer.Stop();
            Hamster.TransitionToState(new EscapeState(Hamster));
            return;
        }

        switch (Hamster.SleepPBar)
        {
            case <= -5:
                _timer.Stop();
                Hamster.TransitionToState(new SleepState(Hamster));
                return;
            case 30:
                Status = "Кажется, пора спать...";
                break;
        }

        if (Hamster.HungryPBar <= 30)
        {
            _timer.Stop();
            Hamster.TransitionToState(new HungryState(Hamster));
            return;
        }

        switch (Hamster.LeisurePBar)
        {
            case <= 5:
                _timer.Stop();
                Hamster.CountEscape++;
                Hamster.TransitionToState(new WheelState(Hamster));
                return;
            case 30:
                Status = "Кажется, хомяку скучно...";
                break;
        }
    }
}
```

OtkinylsiaState.cs

```
using System.Threading.Tasks;
using coursavt.MVVM.Model.Consts;

namespace coursavt.MVVM.Model.States;

public class OtkinylsiaState : HamsterState
{
    public OtkinylsiaState(Hamster hamster) : base(hamster)
```

```
    {
    }

    public override async Task Enter()
    {
        Hamster.ButtonEnabled = false;
        Hamster.EatButtonEnabled = false;
        Status = "Вы не уследили за хомяком и он ушел :(";
        await Logic();
    }

    protected override async Task Logic()
    {
        Hamster.Timer.Stop();
        Hamster.IsAlive = false;
        Hamster.ImagePath = HamsterImages.OtkinylsiaImage;
        await Move(Hamster.Location, Hamster.Location with { Y = 0 });
        Exit();
    }

    public override void Exit()
    {
        Status = "Игра закончена";
    }
}
```

PatState.cs

```
using System;
using System.Threading.Tasks;
using System.Timers;
using coursavt.MVVM.Model.Consts;

namespace coursavt.MVVM.Model.States;

public class PatState : HamsterState
{
    public PatState(Hamster hamster) : base(hamster)
    {
        _patTime = 3;
    }

    private byte _patTime;

    public override async Task Enter()
    {
        Hamster.ButtonEnabled = false;
        Hamster.EatButtonEnabled = false;
        await Logic();
    }

    protected override async Task Logic()
    {
        await Move(Hamster.Location, HamsterPoints.PatPoint);
        Hamster.ImagePath = HamsterImages.PatImage;
        Hamster.LeisurePBar = Hamster.LeisurePBar + 40 > 100 ? 100 :
Hamster.LeisurePBar + 40;
        Hamster.CountFright = 0;
        _timer.Start();
        _timer.Elapsed += timer_Tick;
    }

    public override void Exit()
```

```
    {
        Status = "Хомяк тоже Вас любит.";
        Hamster.CountEscape = (byte)(Hamster.CountEscape - 1 < 0 ? 0 :
Hamster.CountEscape - 1);
    }

    private void timer_Tick(Object source, ElapsedEventArgs e)
    {
        _patTime--;
        if (_patTime != 0) return;
        _timer.Stop();
        _timer.Dispose();
        Hamster.TransitionToState(new InitState(Hamster));
    }
}
```

ScareState.cs

```
using System;
using System.Threading.Tasks;
using System.Timers;
using coursavt.MVVM.Model.Consts;

namespace coursavt.MVVM.Model.States;

public class ScareState : HamsterState
{
    public ScareState(Hamster hamster) : base(hamster)
    {
        _timer = new Timer(500);
    }

    public override async Task Enter()
    {
        Hamster.ButtonEnabled = false;
        Hamster.EatButtonEnabled = false;
        Status = Hamster.CountFright switch
        {
            0 => "Зачем пугаете малышарика?",
            1 => "Бедняга....",
            2 => "Он же боится...",
            _ => "Зря-зря..."
        };
        Hamster.CountEscape++;
        _timer.Start();
        _timer.Elapsed += timer_Tick;
        await Logic();
    }

    protected override async Task Logic()
    {
        if (Hamster.CountEscape > 5)
        {
            Hamster.TransitionToState(new EscapeState(Hamster));
            return;
        }
        Hamster.ImagePath = HamsterImages.FrightImage1;
        switch (Hamster.CountFright)
        {
            case 0:
                await Move(Hamster.Location, HamsterPoints.HungryPoint, 5);
                await Move(Hamster.Location, HamsterPoints.HomePoint, 5);
                await Move(Hamster.Location, HamsterPoints.EatPoint, 5);
```

```
                await Move(Hamster.Location, HamsterPoints.WheelPoint, 5);
                await Move(Hamster.Location, HamsterPoints.PatPoint, 5);
                Hamster.ButtonEnabled = true;
                Hamster.EatButtonEnabled = true;
                break;
            case 1:
                await Move(Hamster.Location, HamsterPoints.HomePoint, 4);
                Hamster.ButtonEnabled = true;
                Hamster.EatButtonEnabled = true;
                break;
            case 2:
                Hamster.ImagePath = HamsterImages.FrightImage2;
                Hamster.ButtonEnabled = true;
                Hamster.EatButtonEnabled = true;
                break;
            default:
                Hamster.TransitionToState(new OtkinylsiaState(Hamster));
                break;
        }
    }

    public override void Exit()
    {
        Hamster.CountFright++;
    }

    private void timer_Tick(Object source, ElapsedEventArgs e)
    {
        if (Hamster.HungryPBar > 30) return;
        _timer.Stop();
        Hamster.TransitionToState(new HungryState(Hamster));
    }
}
```

SleepState.cs

```
using System;
using System.Threading.Tasks;
using System.Timers;
using coursavt.MVVM.Model.Consts;

namespace coursavt.MVVM.Model.States;

public class SleepState : HamsterState
{
    public SleepState(Hamster hamster) : base(hamster)
    {
        _sleepTime = 5;
    }

    private byte _sleepTime;

    public override async Task Enter()
    {
        Hamster.ButtonEnabled = false;
        Hamster.EatButtonEnabled = false;
        Status = "Хомяк отправляется спать.";
        await Logic();
    }

    protected override async Task Logic()
    {
        await Move(Hamster.Location, HamsterPoints.HomePoint);
```

```
        Hamster.ImagePath = HamsterImages.SleepImage;
        Hamster.SleepPBar = 100;
        _timer.Start();
        _timer.Elapsed += timer_Tick;
        Hamster.CountFright = 0;
    }

    public override void Exit()
    {
        Status = "Выспался";
    }

    private void timer_Tick(Object source, ElapsedEventArgs e)
    {
        Status = $"Осталось спать: {_sleepTime}";
        _sleepTime--;
        if (_sleepTime != 0) return;
        _timer.Stop();
        _timer.Dispose();
        Hamster.TransitionToState(new InitState(Hamster));
    }
}
```

WheelState.cs

```
using System;
using System.Threading.Tasks;
using System.Timers;
using coursavt.MVVM.Model.Consts;

namespace coursavt.MVVM.Model.States;

public class WheelState : HamsterState
{
    public WheelState(Hamster hamster) : base(hamster)
    {
        _wheelTime = 3;
    }

    private byte _wheelTime;

    public override async Task Enter()
    {
        Hamster.ButtonEnabled = false;
        Hamster.EatButtonEnabled = false;
        await Logic();
    }

    protected override async Task Logic()
    {
        await Move(Hamster.Location, HamsterPoints.WheelPoint);
        Hamster.ImagePath = HamsterImages.WheelImage;
        Hamster.LeisurePBar = Hamster.LeisurePBar + 40 > 100 ? 100 :
Hamster.LeisurePBar + 40;
        Hamster.CountFright = 0;
        _timer.Start();
        _timer.Elapsed += timer_Tick;
    }

    public override void Exit()
    {
        Status = "Реально думали, что будет бегать?";
    }
```

```
    private void timer_Tick(Object source, ElapsedEventArgs e)
    {
        Status = $"Осталось бегать: {_wheelTime}";
        _wheelTime--;
        if (_wheelTime != 0) return;
        _timer.Stop();
        _timer.Dispose();
        Hamster.TransitionToState(new InitState(Hamster));
    }
}
```