Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Егор Евгеньевич Зайнуллин

Реализация образовательной среды программирования исполнителей на основе платформы REAL.NET

Бакалаврская работа

Научный руководитель:
доцент кафедры СП СПбГУ, к. т. н. Ю. В. Литвинов

Рецензент:
педагог дополнительного образования, учитель информатики ГБОУ лицей № 419
Санкт-Петербурга М. М. Киселев

Санкт-Петербург
2020

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Egor Zainullin

Realisation of education system of perfomers' programming on REAL.NET platform

Graduation Thesis

Scientific supervisor:
Docent, C.Sc. Yurii Litvinov

Reviewer:
additional education teacher, informatics teacher, Lyceum 419 Michail Kiselev

Saint-Petersburg
2020

Оглавление

Введение

С недавнего времени школьники сдают Единый государственный экзамен, используя компьютер для программирования части заданий. Возникла идея о создании среды программирования, с помощью которой обучающиеся смогли бы изучать программирование, начиная с младших классов и визуального программирования, а заканчивая старшей школой уже с использованием текстовых языков вплоть до самого экзамена. Это упрощает обучение, так как учащимся не надо знакомиться с новыми программами для разработки. С просьбой сделать такую среду обратились к нам преподаватели Лицея №419. Первым шагом для реализации этой задачи является создание системы для обучения детей младшего школьного возраста.

Хорошо себя зарекомендовала для обучения школьников среднего школьного возраста среда TRIK Studio [7]. Она нужна для программирования роботов, в качестве языка для разработки использует визуальный язык. Это позволяет снизить порог вхождения, быстро обучиться синтаксису языка благодаря его простоте, а также позволит детям наглядно понять, как работает тот или иной алгоритм на примере робота, его выполняющего. Но TRIK Studio представляет сложность для детей младшего школьного возраста и работает только лишь с роботами. По этой причине возникла идея о создании похожей системы, которая бы позволила работать с абстрактным роботами, умеющими выполнять небольшой набор команд. Таких роботов не надо собирать и настраивать, они перемещаются по экрану компьютера, таким образом, визуальный язык также будет содержать малое количество операторов, что позволит уже программировать ученикам младшей школы. Такие роботы называются исполнителями. Такие исполнители, как «Черепашка», придуманная выдающимся математиком, программистом и педагогом Сеймуром Пейпертом, а также «Робот» много лет применяются для обучения азам программирования. Было доказано, что они эффективно справляются с данной задачей.

В данной работе рассматриваются указанные два исполнителя.

1. Исполнитель «Черепашка». Данный исполнитель умеет непрерывно перемещаться по плоскости, а также рисовать за собой линии. Поддерживает следующие команды (рис. 1).

    - Движение вперед и назад на произвольное расстояние.
    - Поворот вокруг своей оси на определенный градус влево или вправо.
    - Опускание и поднятие пера. Если перо поднято, то исполнитель не рисует, если опущено, то при движении «Черепашка» оставляет за собой след в виде линии.

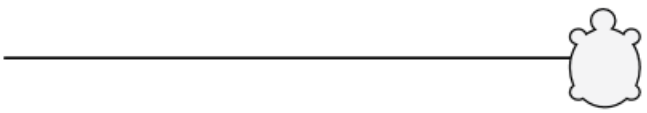

Рис. 1: Исполнитель «Черепашка»

2. Исполнитель «Робот». Данный исполнитель передвигается по клетчатому прямоугольнику со стенами – их пересекать он не может. Поддерживает следующие команды (рис. 2).

    - Движение вперед и назад на одну клетку.
    - Поворот налево и направо на прямой угол.

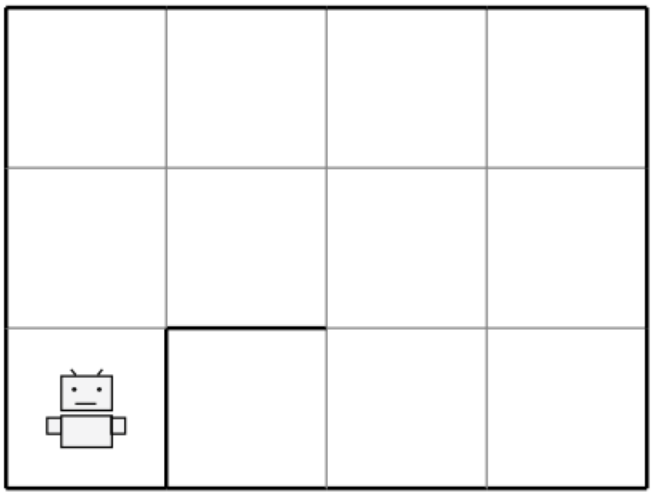

Рис. 2: Исполнитель «Робот»

1. Постановка задачи

Целью данной работы является реализация среды программирования образовательных исполнителей, то есть создание нескольких графических языков, интерпретаторов для них, а также визуализаторов выполнения программ. Для реализации этой цели были поставлены следующие задачи.

1. Выполнить обзор существующих образовательных сред программирования исполнителей.
2. Разработать набор визуальных языков для программирования исполнителей.
3. Разработать архитектуру системы.
4. Реализовать первого исполнителя, умеющего ездить по плоскости и рисовать.
5. Реализовать второго исполнителя, который ездит по клетчатому прямоугольнику со стенами.
6. Провести тестирование и апробацию созданных средств.

2. Обзор

2.1. Существующие среды программирования исполнителей

На данный момент существует несколько образовательных сред программирования исполнителей. Рассмотрим некоторые из тех, которые работают с исполнителями, рассматриваемыми в данной работе, а также используются для обучения школьников в данный момент. Интересуют в первую очередь следующие критерии в рассматриваемых системах.

- Удобство редактора.
- Язык интерфейса.
- Наглядность языка для программирования исполнителей.
- Лицензия системы.

2.1.1. Вычислительная среда FMSLogo

FMSLogo [1] — легковесный редактор, компилятор и визуализатор для программирования исполнителя «Черепашка», разработанный Массачусетским технологическим институтом, популярен в англоязычных странах (рис. 3).

1. Имеет простой редактор для написания программы.
2. Поддерживает многопоточность, добавление нескольких черепашек.
3. Содержит небольшую документацию с примерами на английском языке.
4. Кроссплатформенный.
5. Поддерживает локализацию самих программ с помощью расширений.

6. С открытым исходным кодом.

Основные недостатки FMSLogo заключаются в отсутствии у редактор подсветки синтаксиса, полноценной локализации, что осложняет обучение российских школьников.

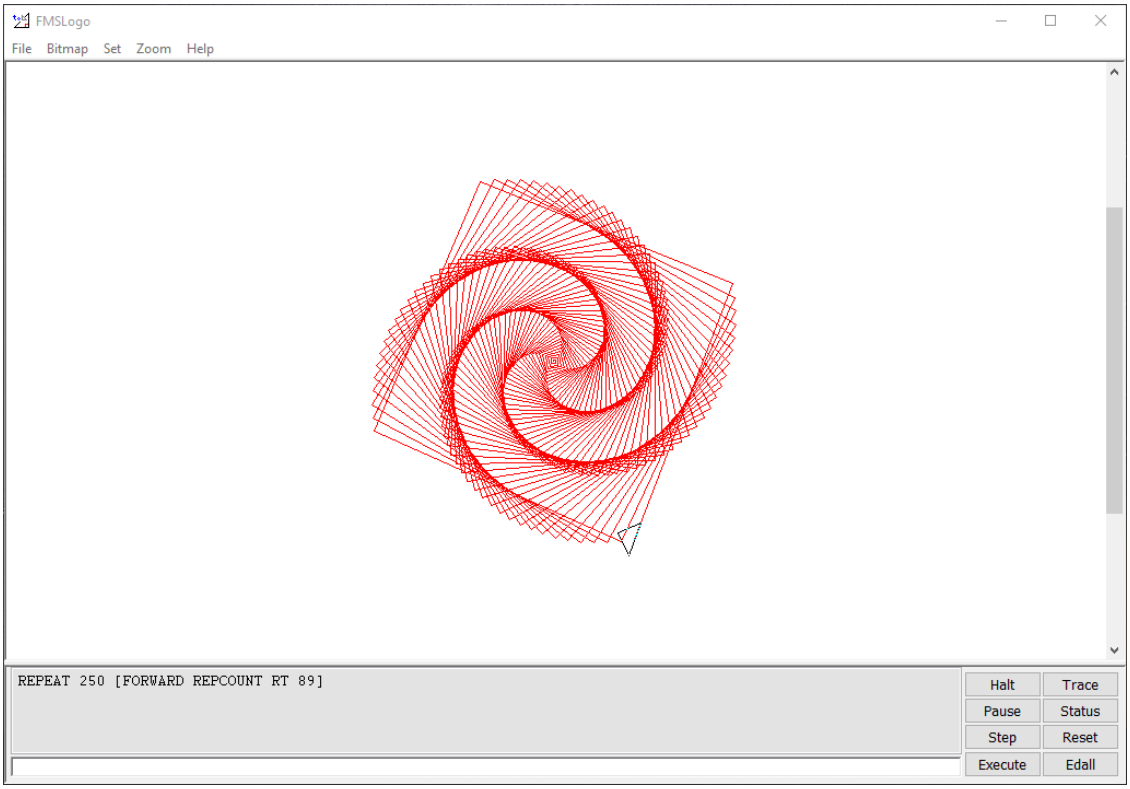

Рис. 3: FMS Logo

2.1.2. Интегрированная творческая среда ЛогоМиры

ЛогоМиры [9] — интегрированная творческая среда для программирования на языке Лого (рис. 4).

1. Является полноценной интегрированной системой разработки: подсвечивает синтаксис, позволяет пошагово отлаживать программу.

2. Можно программировать полноценное приложение, позволяет добавлять кнопки, анимацию, поддерживает многопоточность, одновременное выполнение программы несколькими черепашками.

3. Есть подробная документация и справка.

4. Кроссплатформенная, полностью на русском языке.

5. Проприетарная лицензия.

Ключевым недостатком данной среды является то, что данный проект является коммерческим.

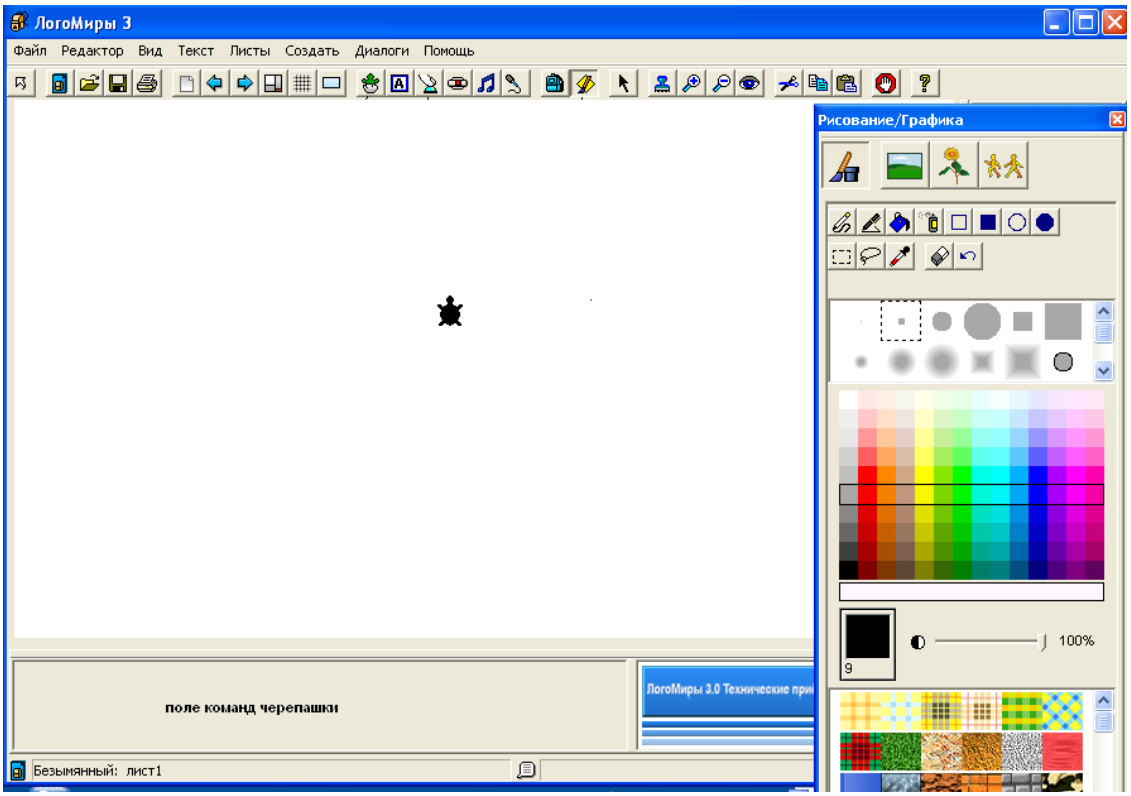

Рис. 4: ЛогоМиры

2.1.3. Язык и система программирования КуМир

КуМир [6] — система программирования исполнителей, являющаяся стандартом обучения школьников в России азам программирования (рис. 5).

1. Проект с открытом исходным кодом, разрабатываемый РАН.

2. Кроссплатформенный.

3. Имеет свой собственный полноценный алгоритмический язык на русском языке, подсвечивает синтаксис.

4. Поддерживает множество исполнителей, в частности: «Черепашку», «Робота», «Чертежника». Более того, позволяет добавлять своих исполнителей и расширять язык новыми конструкциями.

5. Имеет очень хорошую документацию на русском языке с примерами.

Основной недостаток заключается в медленной интерпретации программы, а также многословности алгоритмического языка. Также КуМир не рекомендуется для обучения школьников младших классов.

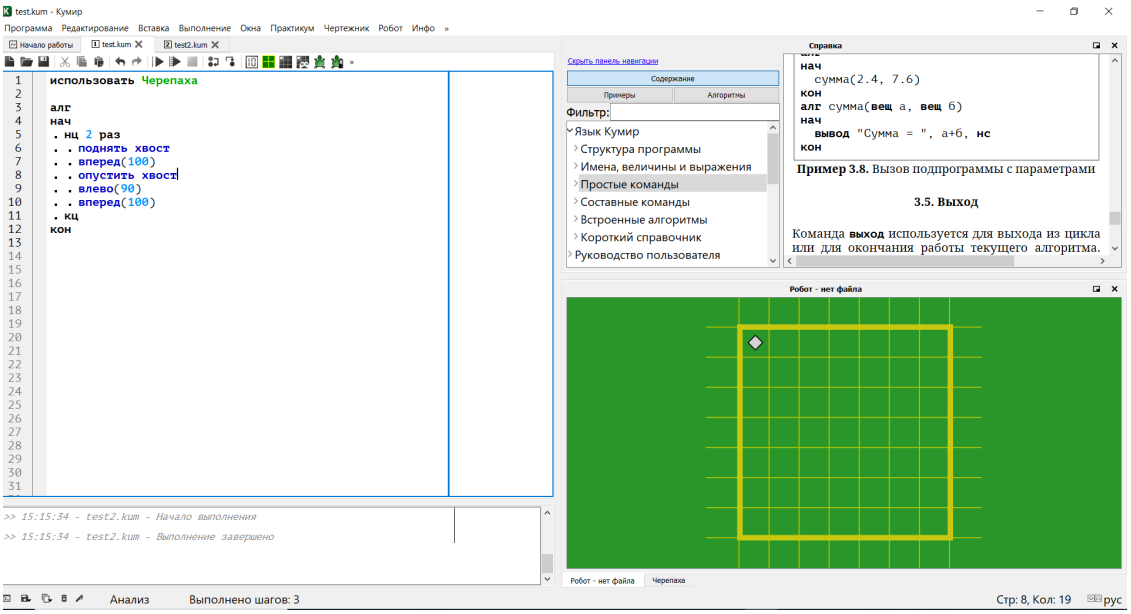

Рис. 5: КуМир

Таким образом, всё ещё существует потребность в создании среды программирования для российских школьников младших классов.

2.2. Предметно-ориентированные языки

Рассмотрим какую-нибудь предметную область, например, бытовую технику. Насколько бы упростило жизнь, если бы пользователь мог бы настраивать режим стиральной машины, задавал бы путь робота-пылесоса, писал бы сценарии для умного дома. Для того, чтобы это

стало реальностью необходимо было бы использовать такой язык для программирования техники, который бы не требовал особых знаний, был наглядным и простым. По этой причине разумно использовать именно предметно-ориентированный язык, то есть язык который создается конкретно для данной предметной области. Это упрощает его: теперь им может пользоваться не профессиональный программист, а тот, кто разбирается в этой области. Для того, чтобы сделать язык более наглядным, используется визуальное моделирование. Имеется в виду, что программа становится не текстовой, а диаграммой.

Традиционным способом представления программы на визуальном языке является граф: вершины представляют операторы, ребра указывают последовательность их выполнения. У вершин есть атрибуты — поля, в которых указаны свойства данного элемента. В качестве модели исполнения программы можно рассматривать сеть Петри с одним маркером, указывающим на узел, который выполняется в данный момент.

Сам язык также описывается с помощью модели, такая модель называется метамоделью.

Но стоит отметить, что создание языка для конкретного устройства — это часто неоправданные траты, фирма, которая создала данное устройство, не будет тратить деньги на его создание с нуля. По этой причине для реализации языка используется специальный инструмент — уже готовые библиотеки, пакеты и технологии, которые позволяют создать новый язык за разумное время и за приемлемые деньги. Такой инструмент называется Domain-Specific modelling платформа, или DSM-платформа [5].

2.3. Используемые технологии

2.3.1. REAL.NET

В качестве основы проекта взята DSM-платформа REAL.NET [10]. Данная система позволяет создать визуальный язык, а также предоставляет редактор для программирования на созданном языке. Модель — граф на сцене, справа от которой — палитра с конструкциями язы-

ка. Внизу представлена консоль, справа снизу находятся атрибуты конкретного элемента на сцене (рис. 6).

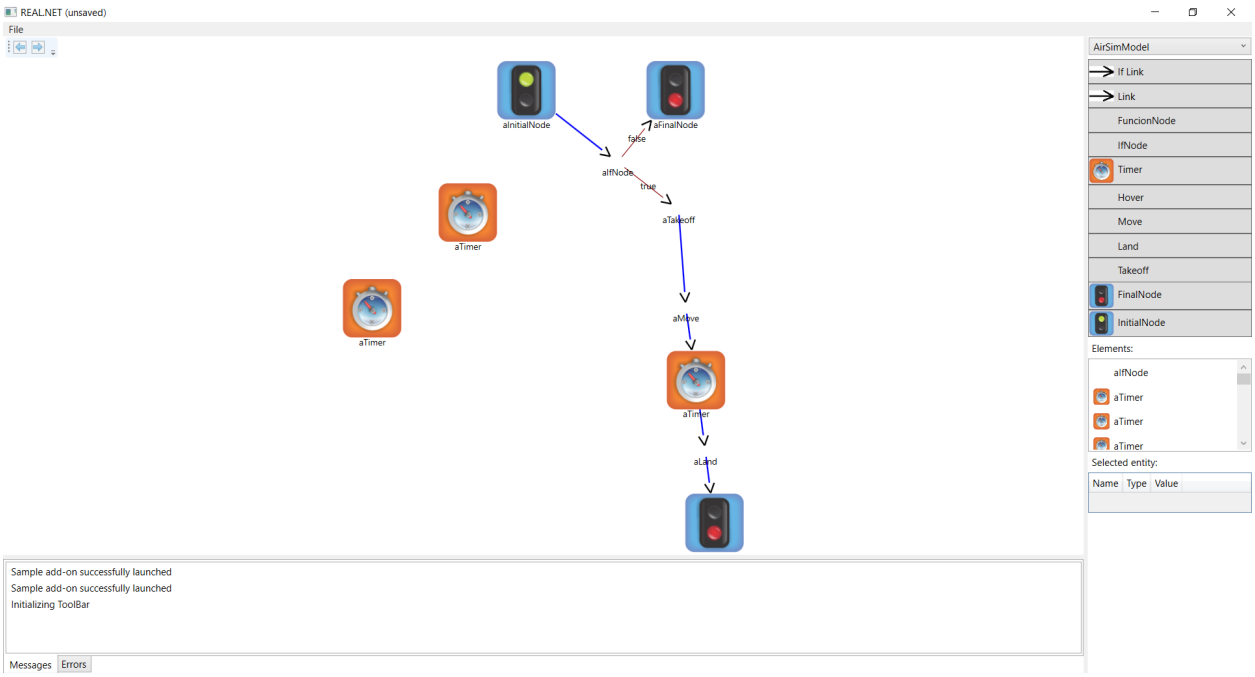

Рис. 6: Редактор REAL.NET

На рис. 7 представлена диаграмма компонентов данной системы. На диаграмме изображены непосредственно сам редактор, библиотека, которая описывает графические элементы, составляющие редактор, а также репозиторий, в котором хранятся все созданные модели.

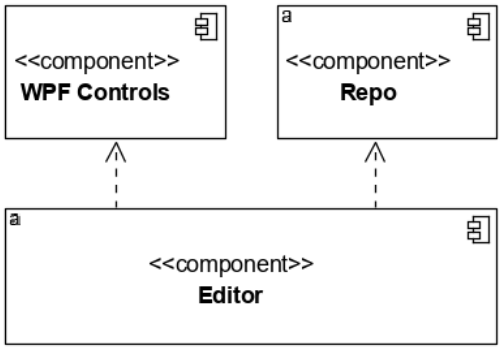

Рис. 7: Компоненты REAL.NET

2.3.2. Windows Presentation Foundation

REAL.NET написана на .NET, а для построения графического интерфейса используется подсистема Windows Presentation Foundation (WPF).

При использовании WPF Microsoft рекомендует придерживаться паттерна[1] Model-View-ViewModel [11]. Графические элементы описываются на специальном языке разметки XAML, основанном на XML – эта часть называется View. Бизнес-логика приложения, то есть Model, содержится в коде на C#. Для того, чтобы связать модель и графическое представление, создается промежуточная составляющая ViewModel – класс, предоставляющий данные View для отображения и обновляющий модель при необходимости.

[1]Подход к решению задачи по проектированию

3. Визуальный язык

3.1. Требования к языку

Необходимо, чтобы каждый из языков исполнителей удовлетворял следующим требованиям.

1. Поддерживал циклы, условные выражения.
2. Поддерживал арифметические выражения, позволял создавать переменные.

Сами арифметические выражения могут состоять из таких конструкций.

1. Переменных и одномерных массивов.
2. Бинарных арифметических операторов, таких как: сложение, вычитание, умножение и деление.
3. Бинарных логических операторов, таких как: конъюнкция и дизъюнкция.
4. Бинарных операторов сравнения, таких как: больше, меньше, нестрого больше, нестрого меньше.
5. Унарных, таких как: взятие противоположного и отрицания.
6. Вызовов функций.
7. Инициализаций массивов.

3.2. Описание общих конструкций

Для того, чтобы каждый из создаваемых языков для программирования исполнителей удовлетворял указанным требованиям, в каждый язык были добавлены следующие операторы (рис. 8).

- InitialNode и FinalNode — операторы начала и конца программы.

- IfElse — оператор условия, который проверяет значение своего атрибута ExpressionValue, если оно истинно, то следующий оператор определяется по ребру типа Link, ложно — по ребру типа ElseLink. Link — синего цвета, ElseLink — ребро зеленого цвета.

- Repeat — оператор цикла, который выполняет тело цикла такое число раз, какое указано в значении ExpressionValue. После выполнения цикла выполняется команда, указанная специальным ребром.

- Expression — оператор, который позволяет задать новую переменную или присвоить уже существующей значение.

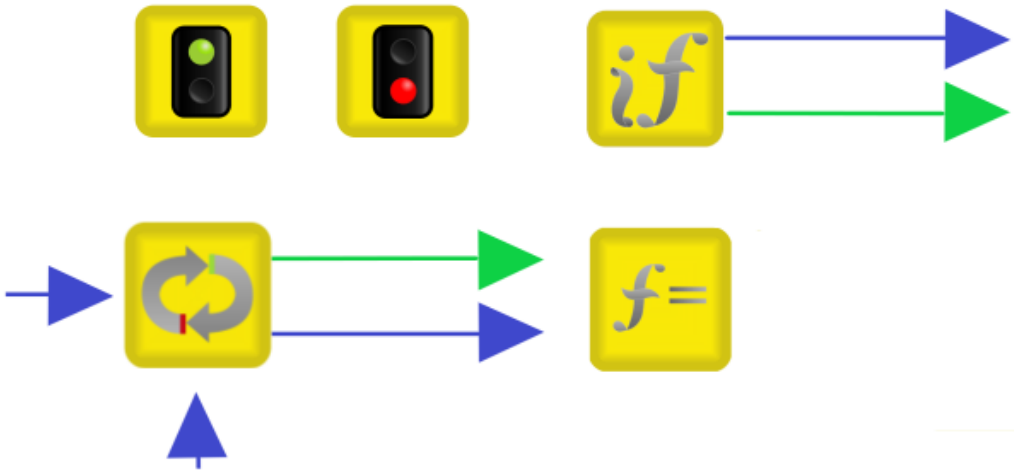

Рис. 8: Общие операторы языка

В качестве атрибутов можно использовать арифметические выражения, удовлетворяющие вышеуказанным требованиям. Они могут быть записаны с помощью следующего синтаксиса.

- Переменная объявляется следующим образом: указывается её имя и начальное значение. Тип не указывается, он определяется автоматически по правой части присваивания. Ровно тот же синтаксис используется для того, чтобы присвоить новое значение уже существующей переменной. Если выражений несколько, они разделяются точкой с запятой.

```
a = 12;
b = 1.0;
c = true;
d = ''hello, world!'';
```

- Можно использовать арифметические и логические операторы. Поддерживаются и скобки.

```
(a + b) * c / d - (-2)

true || x && !y
```

- Массив можно создать таким образом. В квадратных скобках указывается количество элементов в массиве. В фигурных скобах записываются значения для инициализации. Если значение одно, массив инициализирует им все свои элементы.

```
a = new int[4]{1};
b = new bool[2]{true, false};
c = new double[3 + 2];
d = new string[10];
```

- Чтобы обратиться к элементу массива или присвоить ему новое значение, надо использовать квадратные скобки, в которых следует написать индекс искомого элемента.

```
d = a[5];
c[7] = 8.0;
```

На рис. 9 представлена диаграмма классов языка для программирования исполнителей, именно таким образом он хранится в репозитории. Все общие операторы, описанные выше, наследуется от AbstractNode. У данной вершины есть ассоциация с самой собой: это означает, что редактору разрешено соединять ребрами любую вершину с любой другой. От AbstractCommand наследуются команды, специфичные для конкретного исполнителя.

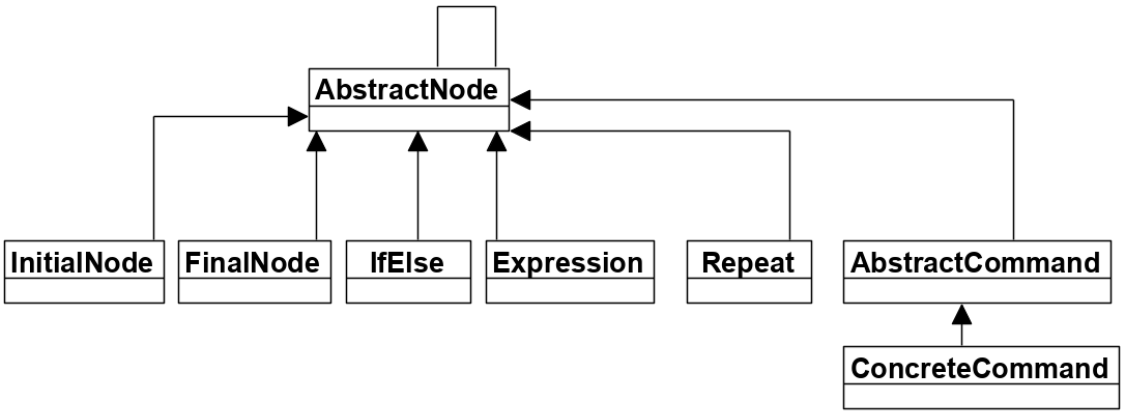

Рис. 9: Диаграмма классов базового языка

3.3. Язык первого исполнителя

В языке первого исполнителя, кроме общих операторов, также присутствуют следующие конструкции (рис. 10).

- Forward и Backward — операторы движения вперед и назад на расстояние, указанное в атрибуте Distance.
- Right и Left — операторы поворота направо и налево на угол, указанный в атрибуте Degrees.
- PenUp и PenDown — операторы, которые соответственно поднимают и опускают перо, которое рисует линию за исполнителем.

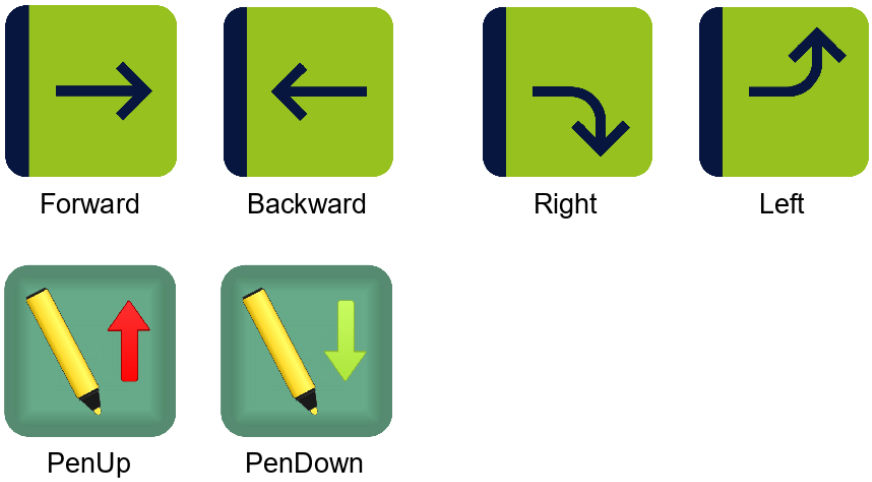

Рис. 10: Команды первого исполнителя

3.4. Язык второго исполнителя

Кроме операторов, общих для всех исполнителей, в данном языке есть следующие операторы.

- Forward — оператор движения на одну клетку вперед.
- Backward — оператор движения на одну клетку назад.
- Right — оператор поворота направо на прямой угол.
- Left — оператор поворота налева на прямой угол.

Рис. 11: Команды второго исполнителя

4. Архитектура системы

4.1. Компоненты системы

На рис. 12 изображена диаграмма компонентов всей системы. Компонент Editor отвечает за редактор. Редактор состоит из сцены с программой, панели атрибутов, панели инструментов. Используя его, пользователь меняет модель. После завершения редактирования пользователь запускает выполнение программы, нажимая соответствующую кнопку на панели инструментов. Запуском и выполнением программы занимается компонент ProgramRunner, он забирает модель из репозитория, определяет ее тип, выбирает интерпретатор, который будет пошагово исполнять полученную модель. На каждом шагу выделяется команда, которая передается соответствующему визуализатору, который уже отображает ее на экране. После выполнения одной команды, начинается следующий шаг интерпретации.

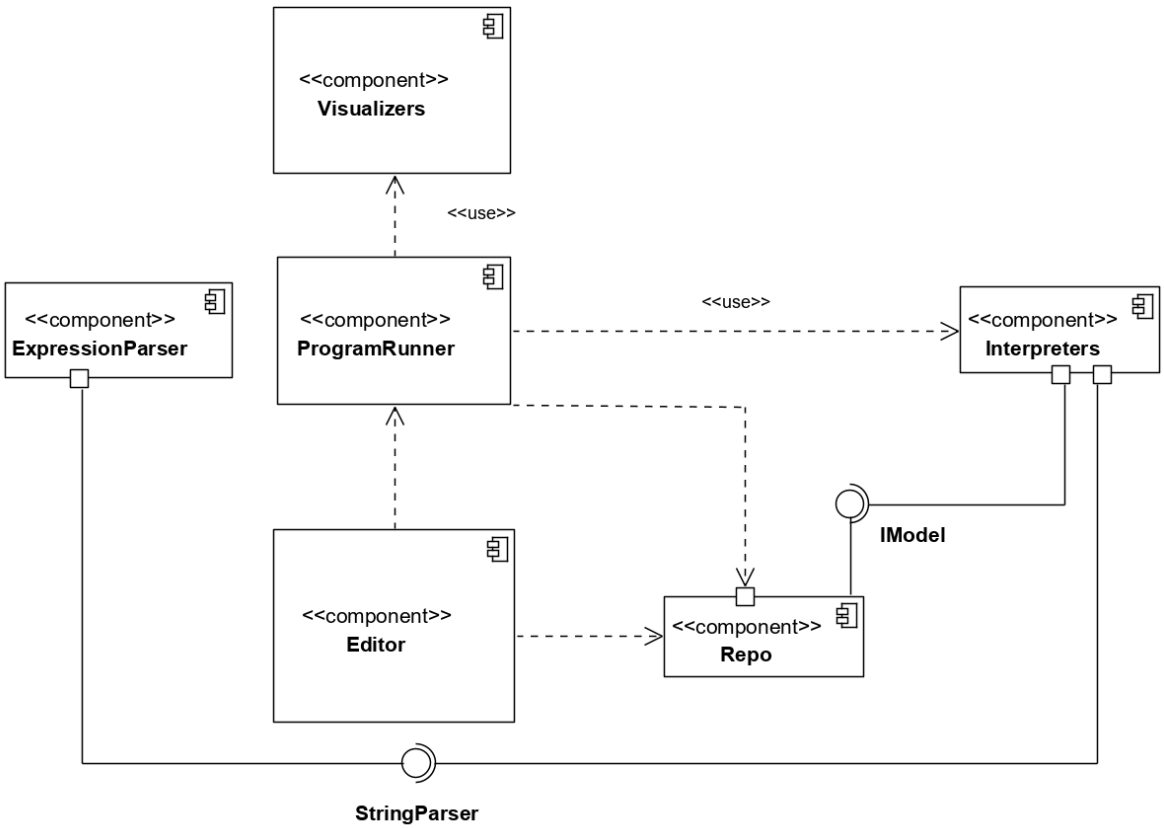

Рис. 12: Компоненты системы

4.2. Архитектура интерпретатора

На рис. 13 проиллюстрирована архитектура интерпретатора. Каждый оператор языка разбирается отдельной функцией, они комбинируются в одну, чтобы разбирались все возможные конструкции. На каждом шагу интерпретатор создает объект типа Parsing, передает его функции разбора, она на основании полученных данных возвращает новый объект того же типа, только с другими полями. В данных полях содержится следующая информация.

1. Множество переменных.
2. Контекст — данные, специфичные для языка исполнителя. Например, здесь может храниться положение «Робота» в лабиринте, конфигурация лабиринта, команды, которые необходимо выполнить.
3. Элемент для интерпретации. Во время выполнения интерпретации должен определиться оператор, к которому нужно обратиться следующим.

Такой подход позволяет просто менять интерпретатор при добавлении нового оператора в язык: достаточно добавить новую функцию разбора для него.

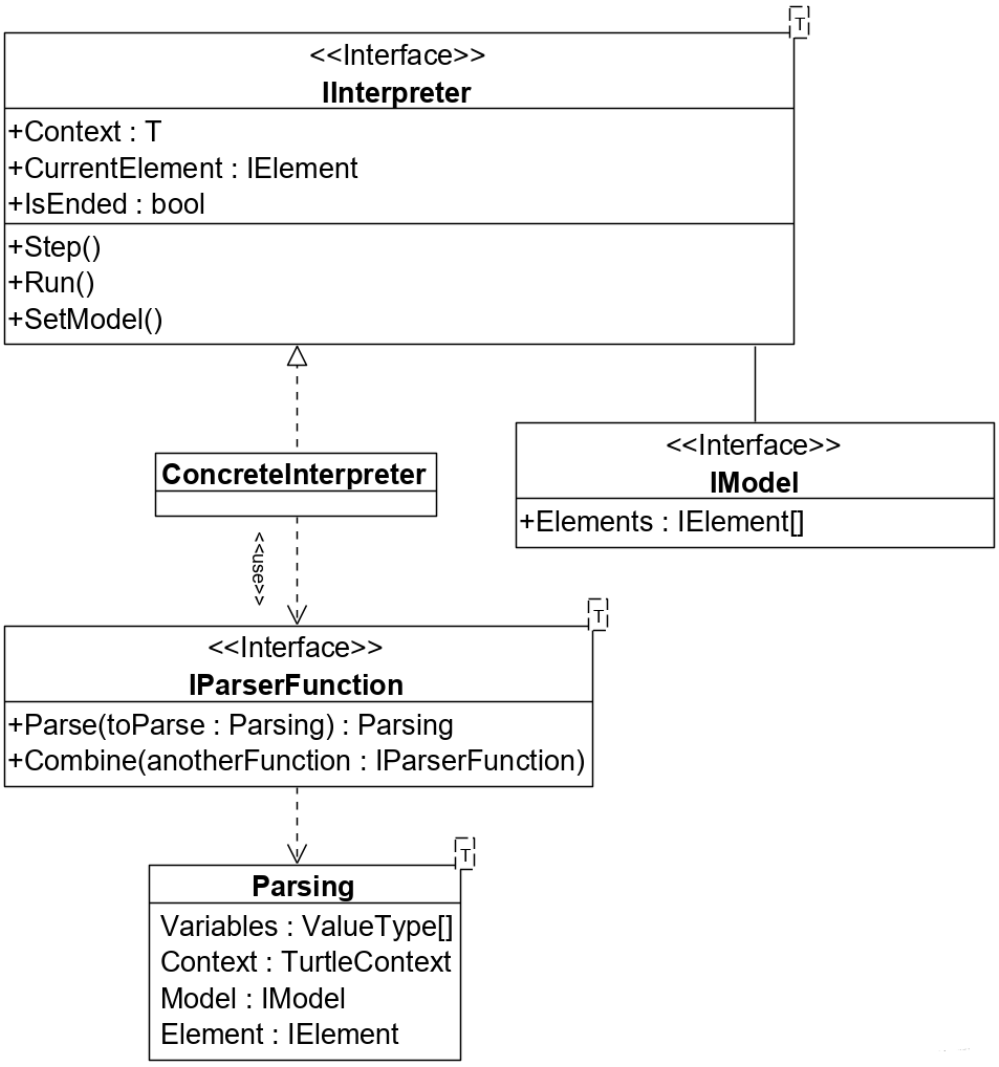

Рис. 13: Диаграмма классов интерпретатора

5. Реализация первого исполнителя

5.1. Реализация интерпретатора

На рис. 14 и рис. 15 представлен принцип работы интерпретатора исполнителя «Черепашка». В начале шага интерпретации интерпретатор создает объект типа Parsing, в который сохраняет информацию о том, какой элемент необходимо разбирать, список переменных, контекст, в котором в данном случае хранятся команды, которые необходимо выполнить исполнителю. Разбор делегируется объекту, который хранит в себе функции разбора всех операторов языка, он выбирает подходящую, она преобразует Parsing и возвращает новое значение. Интер-

претатор выделяет соответствующую команду, которую получит позже ProgramRunner для дальнейшей передачи визуализатору.

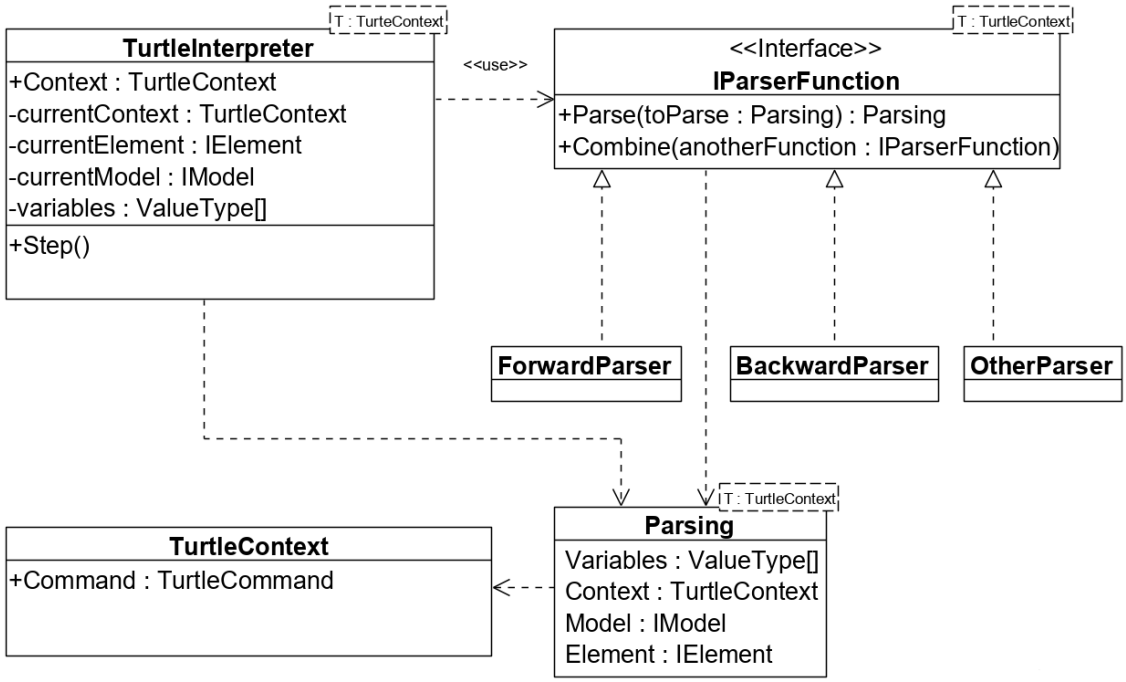

Рис. 14: Диаграмма классов интерпретатора первого исполнителя

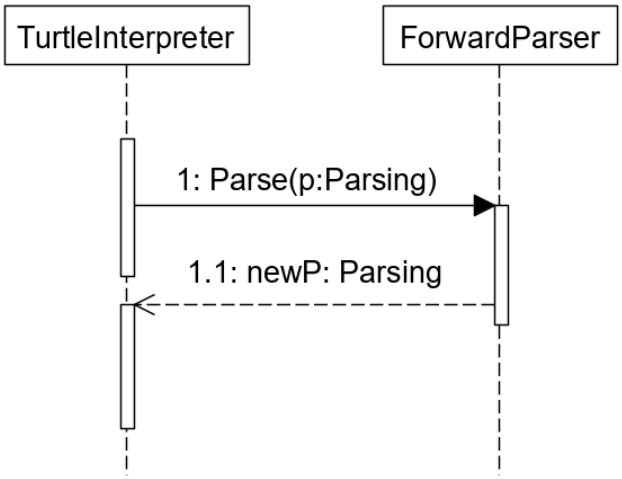

Рис. 15: Диаграмма последовательностей для интерпретатора

На рис. 16 и рис. 17 проиллюстрировано взаимодействие ProgramRunner с интерпретатором. ProgramRunner вызывает метод Step интерпретатора, после выполнения которого получает тип команды, которую надо выполнить. Если при интерпретации возникла ошибка, ProgramRunner отправляет об этом сообщение в консоль.

Вызывается один из методов класса, реализующего интерфейс управления исполнителем ITurtleCommander, то есть для движения используется MoveForward или MoveBackward, для поворота — RotateRight или RotateLeft, для управления пером — PenUp и PenDown. Это инициирует событие, которое уведомляет визуализатор о том, что ему нужно нарисовать.

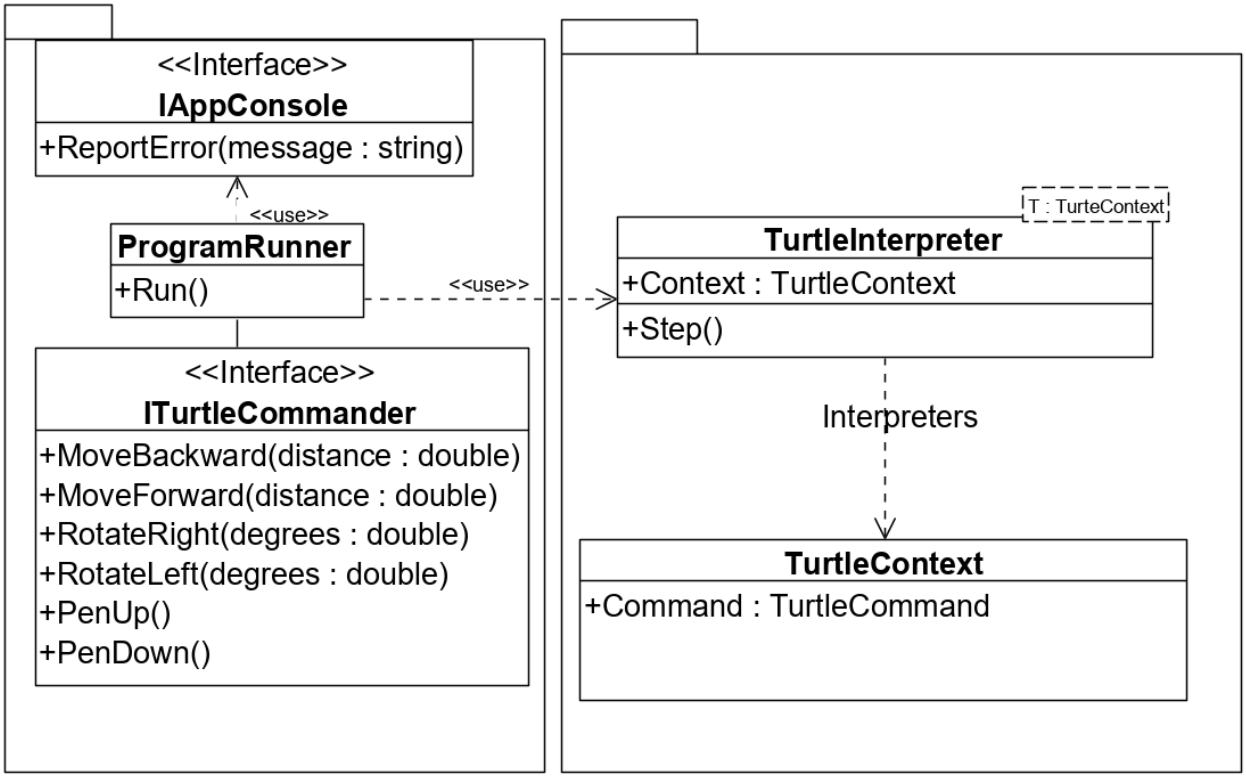

Рис. 16: Использование интерпретатора

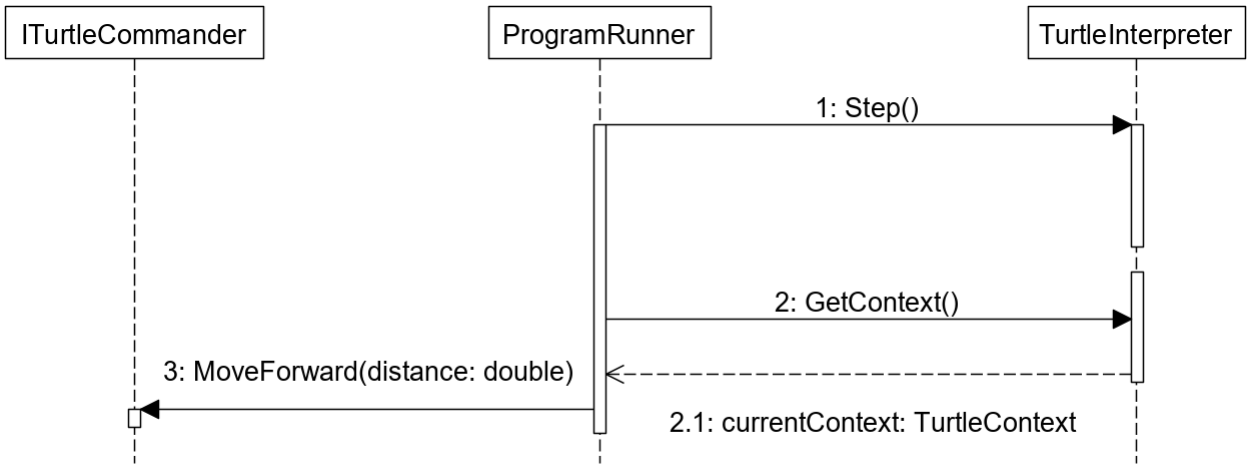

Рис. 17: Взаимодействие с интерпретатором

5.2. Реализация визуализатора

На рис. 18 показано, каким образом работает визуализация движения исполнителя на сцене. ProgramRunner получает данные о том, что необходимо нарисовать на сцене, после выполнения шага интерпретации у TurtleCommander вызывается один из методов интерфейса ITurtleCommander, об этом получает уведомление сцена с исполнителем DrawingSceneViewModel, она узнает новое положение исполнителя, его поворот относительно первоначального положения, необходимо ли рисовать линию. Выставляются привязанные значения начальной и конечной точки движения, свойство IsAnimationStarted является триггером [4] для начала анимации [8]. Одновременно начинается две анимации: сам исполнитель, а также линия, которую он рисует, если перо поднято, то данная линия спрятана от глаз пользователя. После окончания движения на сцену добавляется «нарисованная» линия, а также вызывается метод NotifyActionDone у TurtleCommander: TurtleCommander обновляет поля исполнителя и посылает уведомление о том, что анимация закончена.

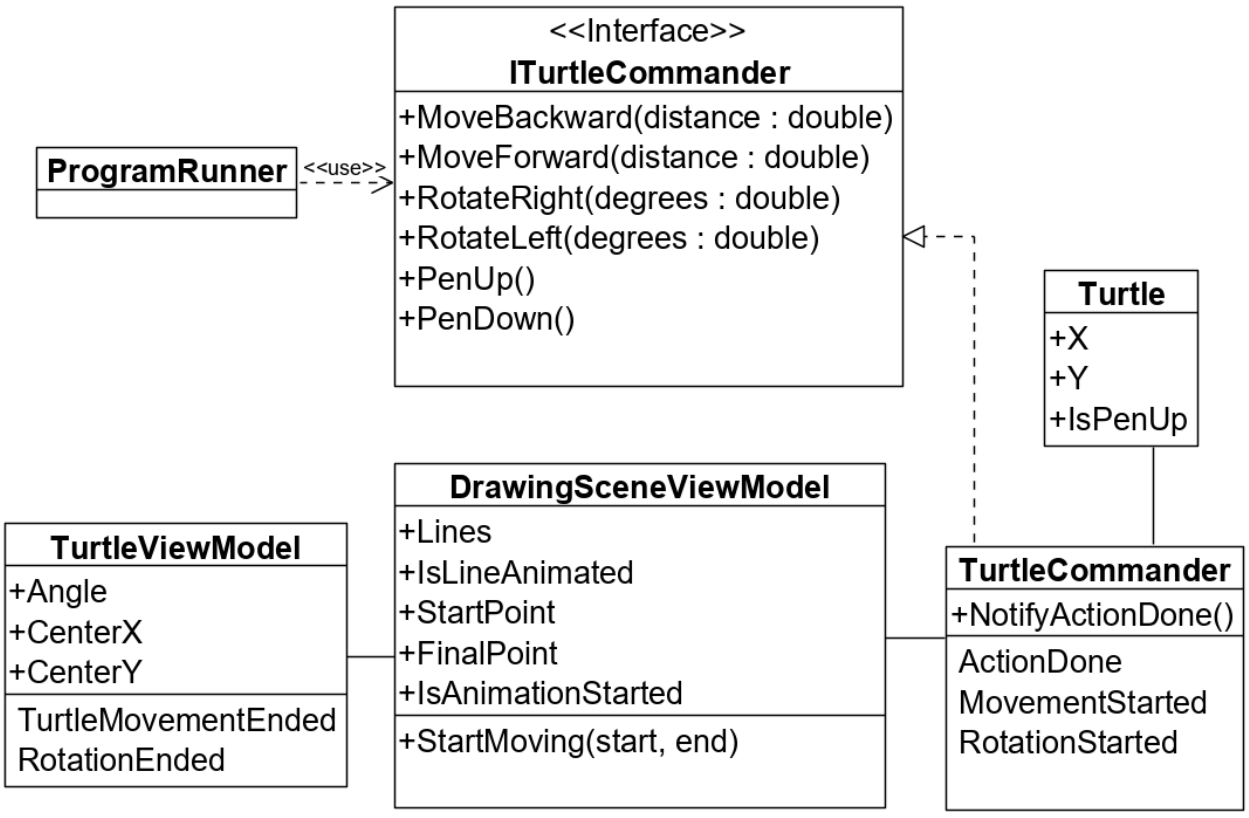

Рис. 18: Диаграмма классов визуализатора

6. Реализация второго исполнителя

6.1. Реализация интерпретатора

На рис. 19 и рис. 20 представлен принцип работы интерпретатора исполнителя «Робот». В начале шага интерпретации создается объект типа Parsing, в который сохраняется информация о том, какой элемент необходимо разбирать, список переменных, контекст, в котором хранятся конфигурация лабиринта, данные об исполнителе: положение и направление, команды, которые необходимо выполнить исполнителю. Разбор делегируется объекту, который хранит в себе функции разбора всех операторов языка, он выбирает ту, которая может разобрать данный элемент, она обрабатывает Parsing и возвращает новое значение. Если эта функция разбирает движение, например, ForwardParser, она также проверяет, что исполнитель не врезается в стенку, если же новое состояние некорректно, инициируется исключение, которое обраба-

тывает интерпретатор. Выделяется соответствующая команда, которая далее будет должна выполниться визуализатором.

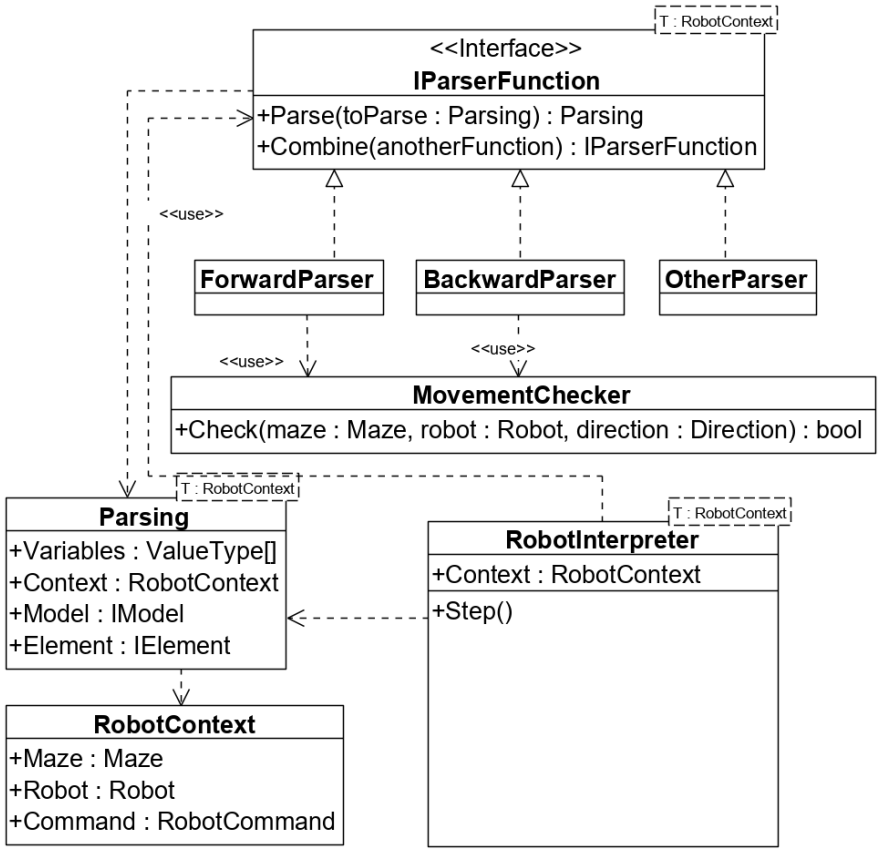

Рис. 19: Диаграмма классов интерпретатора второго исполнителя

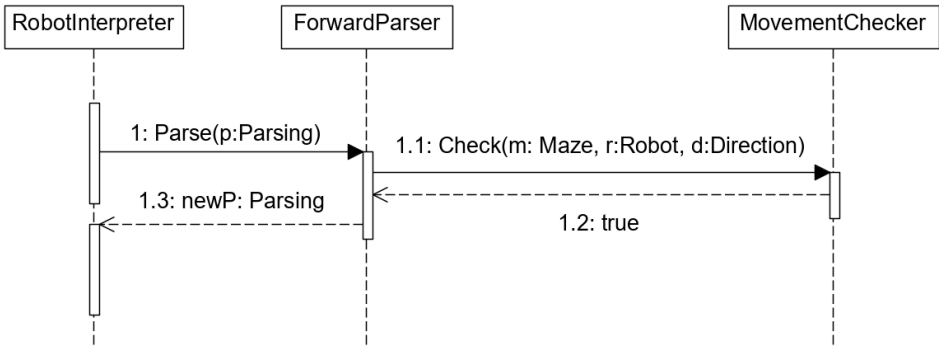

Рис. 20: Диаграмма последовательностей для интерпретатора

На рис. 21 и рис. 22 показано, как происходит взаимодействие ProgramRunner с интерпретатором. ProgramRunner вызывает метод Step интерпретатора, после выполнения которого получает команду, которую надо будет отобразить визуализатору. Если при интерпретации возникает ошибка, ProgramRunner уведомляет об это пользователя, отправляя соответствующее сообщение в консоль.

Вызывается один из методов класса, реализующего интерфейс для управления исполнителем IRobotCommander, то есть при для движения вперед используется MoveForward, назад — MoveBackward, для поворота направо — RotateRight, налево – RotateLeft. Это, в свою очередь, вызывает событие, которое позволяет визуализатору узнать, что надо начать выполнять.

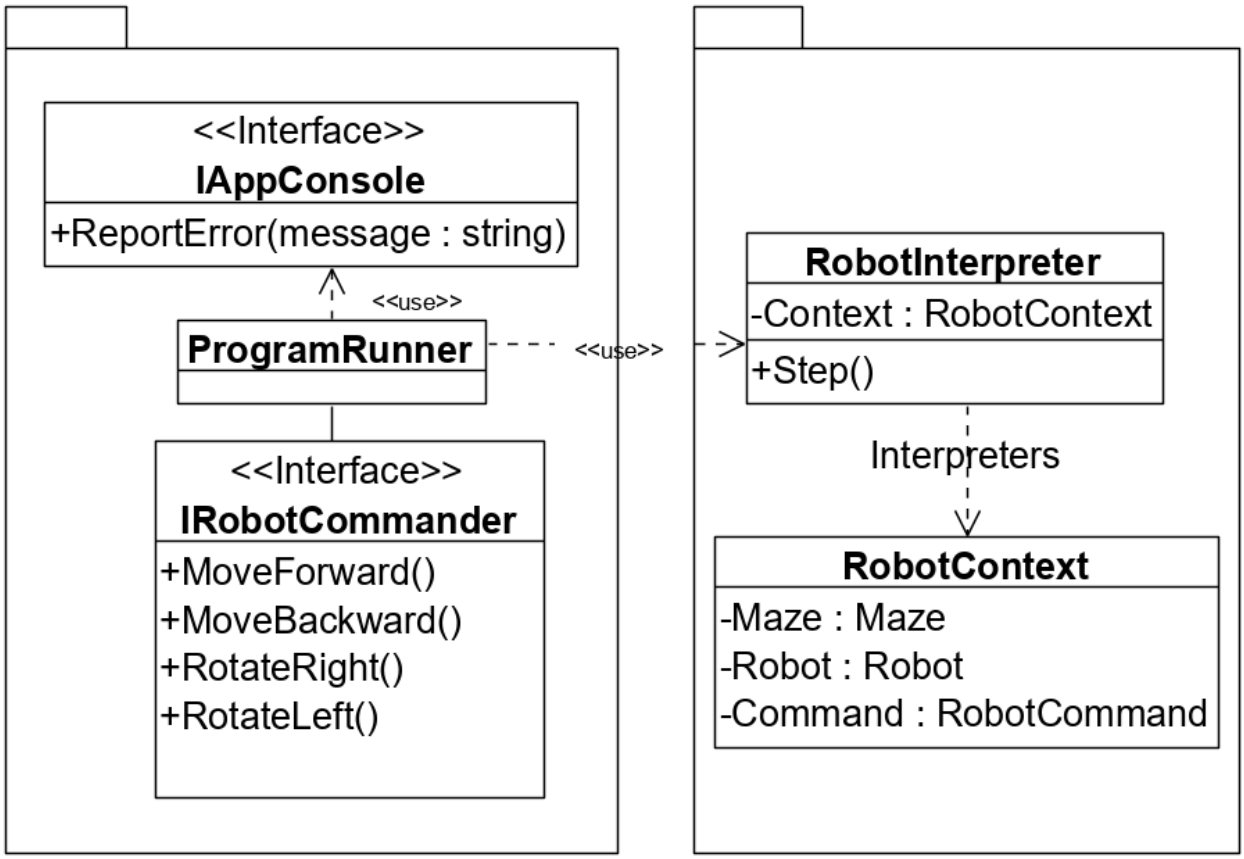

Рис. 21: Использование интерпретатора

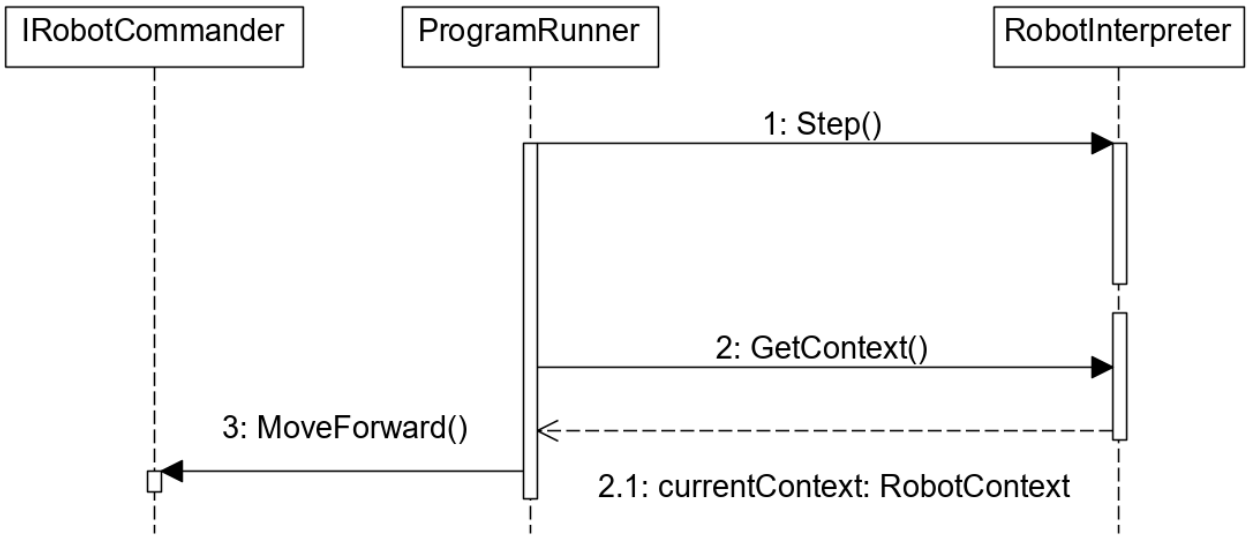

Рис. 22: Взаимодействие с интерпретатором

6.2. Реализация визуализатора

На рис. 23 представлена архитектура визуализатора. ProgramRunner получает информацию о команде, которую надо отобразить, после RobotCommander инициирует событие, благодаря которому сцена с исполнителем RobotSceneViewModel понимает, что необходимо начать анимацию, она узнает новое положение исполнителя, его направление первоначального положения. Эти данные пересчитываются, так как сцена использует вещественные координаты и угол. Выставляются привязанные значения, откуда начинается движение и где оно заканчивается. Свойство IsAnimationStarted является триггером [4] для начала анимации [8]. После окончания движения вызывается метод NotifyActionDone у RobotCommander: поля исполнителя обновляются.

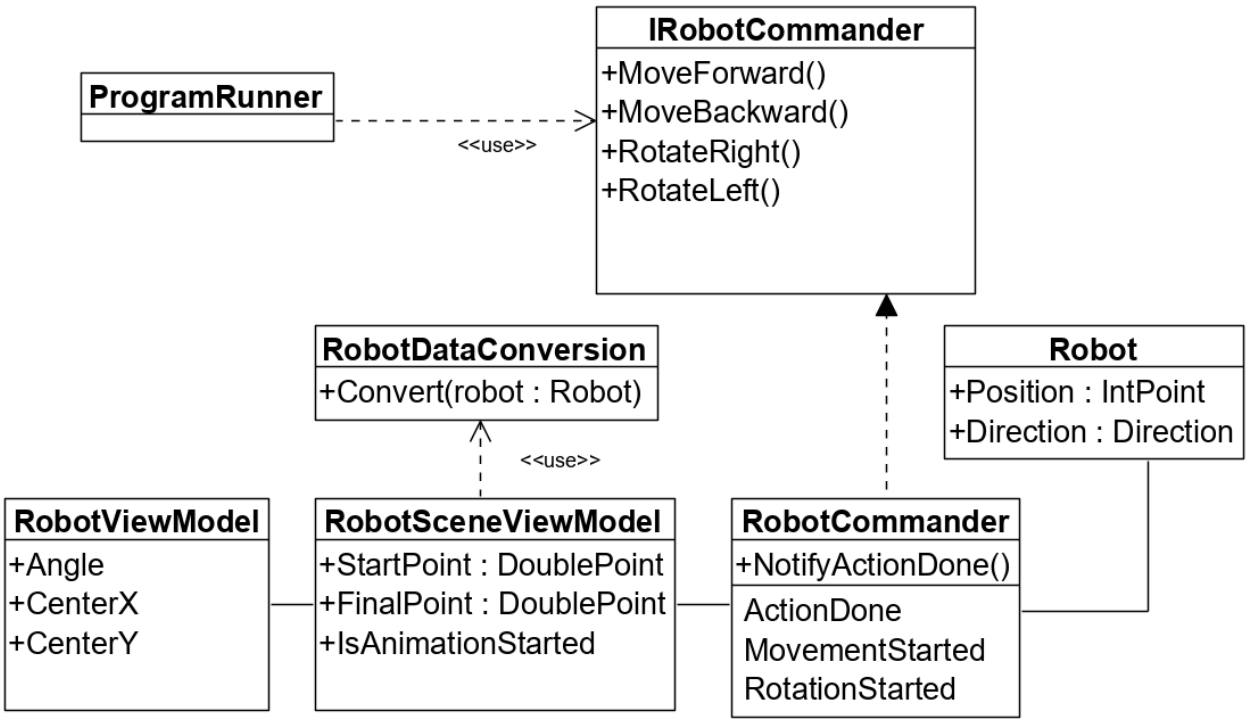

Рис. 23: Диаграмма классов визуализатора

7. Тестирование и апробация

7.1. Тестирование первого исполнителя

Для тестирования интерпретатора первого исполнителя выполнялись следующие сценарии.

1. Разбирались каждые операторы по отдельности: проверялась каждая функция разбора.

2. Для комплексного тестирования было создано несколько автоматических тестов, которые запускали интерпретатор на нескольких программах, включающих все возможные операторы. После выполнения интерпретации тест проверял, что результат выполнения программы совпадал с ожидаемым, то есть список команд, порожденный интерпретатором, сравнивался с эталонным.

Для тестирования визуализатора выполнялись следующие сценарии.

1. Проверялось, что каждая команда исполнителя визуализируется.

2. Проверено было, что при движении с опущенным пером линия рисовалась, а при поднятом — она отсутствовала.

3. Для комплексного тестирования было нарисовано несколько геометрических фигур.

В качестве интеграционных тестов были выполнены несколько программ с рисованием фигур.

7.2. Тестирование второго исполнителя

Для тестирования интерпретатора выполнялись следующие тесты

1. Разбирался каждый оператор своей функцией разбора. Для операторов движения проверялось, что они вызывают исключение при попытке проехать через стену.

2. Для комплексного тестирования были запущены автоматические тесты на программах, включающих все возможные операторы. После того, как интерпретатор заканчивал свою работу, проверялось, что выданные им команды и позиция исполнителя совпадали с эталонными.

Для тестирования визуализатора выполнялись следующие сценарии.

1. Было проверено, что все команды движения визуализируются.

2. Было провизуализировано движение по ломаной.

В качестве интеграционного теста была выполнена программа, в которой исполнитель спустя несколько шагов врезается в стену, то есть программа выполнялась ровно до момента столкновения, было проверено, что консоль предупредила о данной ошибке.

7.3. Апробация

Апробация проводилась в два этапа.

1. Апробация проводилась на учащихся младших классов Лицея №419, занимающихся в кружке по программированию. В качестве ознакомительного материала школьникам была предоставлена инструкция с описанием операторов, редактора, а также алгоритмом запуска программы. В качестве задания на занятие опрашиваемым были выданы задачи. Участники опроса до этого уже имели опыт работы с исполнителями.

   В первую очередь сложность вызвал редактор, так как на данный момент редактор не позволяет использовать drag-and-drop для ребер. Также у участников возникли вопросы по использованию панели атрибутов, так как ее изначально сложно обнаружить. Функциональность приложения устроил как и школьников, так и преподавателей.

   В целом продукт оставил благоприятное впечатление.

2. Апробация также проводилась путем опроса шести учеников средней школы разной степени подготовки. Каждому была выдана та же инструкция, что и в предыдущем этапе, а также было дано словесное описание способа использования редактора. Сложности возникли с рисованием программ. По шкале удобства использования системы (system usablity scale) приложение набрало 75,4 балла, что соответствует оценке B — «хорошо» [2].

   На рис. 24 представлены средние значения по каждому из вопросов. Наименьшие баллы набрали вопросы под номерами 1 и 4: «я буду часто использовать эту систему» и «мне понадобится помощь, чтобы научиться пользоваться этой системой». Низкий балл первого вопроса объясняется тем, что участники опроса не занимаются программированием исполнителей, второго — недостаточно хорошей документации и отсутствием опыта у опрашиваемых. Наивысший балл набрал вопрос под номером 7: «большая

часть людей очень быстро научится пользоваться этой системой». Из этого можно сделать вывод о простоте системы.

Рис. 24: Среднее значение по вопросам

В целом апробация показала, что система достаточно проста, можно быстро обучиться ее использовать, что позволяет ее использовать для преподавания школьникам программирования.

Заключение

В ходе данной работы получены следующие результаты.

- Сделан обзор существующих сред программирования исполнителей: FMSLogo, ЛогоМиры, КуМир.
- Созданы визуальные языки для первого и второго исполнителей.
- Разработана архитектура системы: спроектирован интерпретатор, а также механизм взаимодействия сервисных компонентов с интерпретатором.
- Создан интерпретатор команд первого исполнителя, его визуализатор был спроектирован и реализован.
- Создан интерпретатор команд второго исполнителя, был спроектирован и реализован его визуализатор.
- Проведено тестирование полученных средств, а также апробация на реальных пользователях.

Доклад по данной работе был представлен на конференции «Современные технологии в теории и практике программирования», сборник с тезисами доклада готовится к публикации. Исходный код проекта опубликован в GitHub [3].

Список литературы

[1] FMSLogo // Internet. — Access mode: http://fmslogo.sourceforge.net/manual/ (online; accessed: 14.12.2019).

[2] Sauro Jeff. MEASURING USABILITY WITH THE SYSTEM USABILITY SCALE. — Access mode: https://measuringu.com/sus/ (online; accessed: 24.05.2020).

[3] Домашняя страница REAL.NET. — Режим доступа: https://github.com/yurii-litvinov/REAL.NET (дата обращения: 13.12.2019).

[4] Класс DataTrigger. — Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.datatrigger?view=netframework-4.8 (дата обращения: 15.03.2020).

[5] Кознов Д.В. Основы визуального моделирования. — БИНОМ, 2008. — ISBN: 978-5-94774-823-9. — https://math.spbu.ru/user/dkoznov/papers/vmbook.pdf (дата обращения: 19.05.2020).

[6] Леонов А. Г., Первин Ю. А. Переход от непосредственного управления исполнителями к составлению программ в пропедевтическом курсе информатики // Ярославский педагогический вестник. — 2013. — Т. 3. — С. 20–30.

[7] Мордвинов Д.А. Среда программирования роботов TRIK Studio. — Режим доступа: https://dspace.spbu.ru/bitstream/11701/5456/1/Mordvinov_TRIKStudioTechnicalIntroduction.pdf (дата обращения: 22.02.2020).

[8] Общие сведения о Storyboard. — Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/graphics-multimedia/storyboards-overview (дата обращения: 15.12.2019).

[9] Пирогова Е.Ю. Создание проектов в интерактивной творческой среде ЛогоМиры. — ВИРО, 2013. — ISBN: 978-5-87590-398-4. — http://viro.edu.ru/attachments/article/2495/LogoMiry.pdf (дата обращения: 18.04.2020).

[10] Литвинов Ю.В., Кузьмина Е.В., Небогатиков И.Ю., Алымова Д.А. Среда предметно-ориентированного программирования REAL.NET // Список. — 2017. — Режим доступа: https://github.com/yurii-litvinov/articles/blob/master/2017-realNet/realNet.pdf (дата обращения: 09.12.2018).

[11] Шаблон Model-View-ViewModel. — Режим доступа: https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/enterprise-application-patterns/mvvm (дата обращения: 25.12.2018).