МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Омский государственный университет
им. Ф.М. Достоевского

Кафедра «Математической логики и логического программирования»

ДИПЛОМНАЯ РАБОТА
на тему: «Разработка модулей для web-системы обучения программированию»

ВЫПОЛНИЛ:
Студент гр. ММС-902-о
очной формы обучения
Притужалов Евгений

НАУЧНЫЙ РУКОВОДИТЕЛЬ:
Ашаев И. В.

Омск – 2014

Содержание

Для обучения детей младшего и среднего возраста широко применяются графические исполнители. Программист при помощи команд управления неким роботом, который может передвигаться по плоскости и чертить на ней рисунки. Для этого используется простой язык программирования, включающий систему команд непосредственного управления роботом, т.е. команды перемещения и рисования, а также управляющие конструкции, которые позволяют организовать повторения, ветвления, выделить какие–то действия в подпрограмму или процедуру.

В качестве такого языка часто используется Logo [13]. В Омской физико–математической школе № 64 в 90-е годы прошлого века в течение долгого времени использовалась программа MSW Logo от Softronics Home Page [14]. К сожалению развитие этого продукта остановилось, и даже интерфейс продукта выглядит несколько инородно в современных версиях OS Windows. На данный момент лицей № 64 уже давно не использует этот продукт. Сейчас там используется Scratch-проект Массачусетского технологического института [15].

ООО «Образование IT» одна из омских организаций, которая занимается обучением школьников программированию в рамках проекта «Школа программиста». Её не устроили известные графические исполнители, реализованные в виде десктопных приложений, в силу громоздкости интерфейса, сложности установки, ограниченности поддерживаемых платформ. С одной стороны, хотелось иметь продукт, который бы не требовал установки вовсе, или устанавливался бы очень легко, а так же поддерживался всеми популярными платформами: Windows, OS X, Linux. Выбор упал на web-приложение, ведь для его работы необходим только современный браузер, который с большой вероятностью используется на компьютере в учебных заведениях и дома у учеников. С другой стороны, хотелось получить удобство и гибкость в создании и поддержке методических материалов, а так же иметь независимость от наличия интернета, чтобы иметь возможность заниматься в летних выездных школах в оздоровительных лагерях, а так же в рядовых городских образовательных заведениях, где перебои с интернетом-вещь весьма обычная. Поэтому «Школу программиста» не устроили существующие, на данный момент, Web аналоги [1–3, 10, 17–23], которые не дают удобной возможности организовывать библиотеку методических материалов, а так же, как правило, требуют наличие соединения с сервером. Поэтому было принято решение создать собственного графического исполнителя, который обладал бы всеми необходимыми свойствами: работал бы в браузере, позволял организовывать методические материалы, не был бы привязан к интернету. Так появился проект «Тортуга».

«Тортуга» уже использовался для обучения школьников в течение весны 2013 года. В рамках этого проекта передо мной стояла задача по расширению функциональных возможностей программы. Описанные в данной работе результаты на данный момент включены в релизную версию программы, которая использовалась преподавателями «Школы программиста» в летних выездных и при школьных лагерях в июне 2013 года.

Глава 1

О проекте «Тортуга»

1.1 Описание

Программа «Тортуга» является проектом компании «Образование IT», которая ставит перед собой цель обучение школьников основам программирования. Программа представляет собой Web–приложение, построенное на языке JavaScript, с использованием технологий HTML5 и CSS3. Благодаря применению данных технологий, нет необходимости в постоянном доступе к сети Интернет. Использовать «Тортугу» можно, скопировав на флешку или любой другой цифровой носитель страницу сайта, где расположено данное приложение.

Один из сценариев работы с «Тортугой» следующий. Преподаватель на странице constructor.html создает урок: формулирует тексты и название задачи. Информация об уроке особым образом кодируется и формируется ссылка с текстом этого урока, пройдя по которой, откроется страница index.html с текстом задания. Для выполнения задачи ученику потребуется создать объект Черепашка и, управляя ее действиями через команды, достигнуть поставленной задачи.

Часто учителям не удается заинтересовать школьников предметом лишь потому, что не могут найти хорошего задания. По этому в «Тортуге» любой человек может самостоятельно построить именно такой урок, который будет ему необходим, что важно для применения в школах. Вариант готового урока можно увидеть на рис. 1.

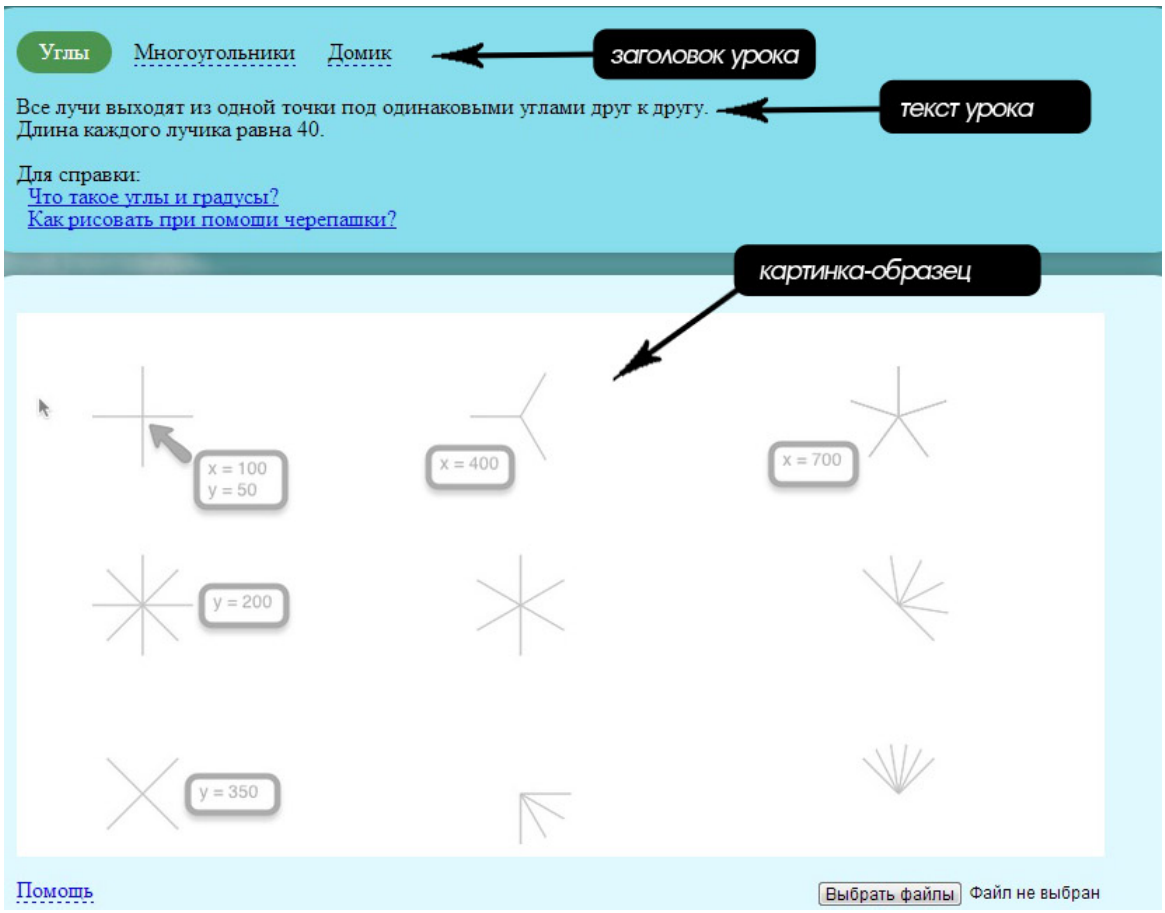

Рисунок 1.1: Пример урока.

Для создания урока на странице constructor.html в текстовое окно вводится текст урока и, при нажатии кнопки «Создать», содержимое текстового поля извлекается, сжимается, кодируется base64 и создается ссылка на урок.

1.2 Механизм работы

Команды управления черепахой пишутся в консоли браузера, или в файле *.js, который добавляется на страницу для обработки с помощью кнопки «открыть файл». Для программирования черепахи используется преобразованный синтаксис языка JavaScript. Например: *t = createTortoise(); t.go(20); t.rotate(90);* Однако написав что либо на привычном нам JavaScript, к примеру *alert();* или объявив функцию, интерпретатор выполнит требуемое. Это происходит потому, что после разбора и преобразования пользовательского кода на выходе получается обычный JavaScript.

Рассмотрим более подробно этот пример. Команды введенные пользователем для управления черепахой в виде последовательности символов подвергаются лексическому анализу для распознания и выделения лексем используемого формального языка. За этот процесс отвечает лексический анализатор. Далее полученные объекты - лексемы преобразуются в синтаксическое

дерево, отображающее синтаксическую структуру переданной последовательности. Эту работу выполняет синтаксический анализатор. Так как команды черепахи представляют из себя отдельный язык, то во время прохода по дереву они интерпретируются в JavaScript, и исполняются, используя доступ к интерфейсу рисования и интерфейсу черепашки. За разбор синтаксического дерева отвечает интерпретатор.

За построение необходимых html - элементов, установку между ними связей и подписывание на события отвечает функция *initApp()*. Так же она жестко прописывает логику зависимостей всех модулей Тортуги. Это приводит к наличию большого объема кода и неразрывной связности компонентов, тем самым лишая приложение гибкости, усложняя Unit - тестирование. А это говорит о необходимости изменения архитектуры на более гибкую и независимую.

Глава 2

Развитие функционала.

2.1 Сокращение ссылки

Как было сказано выше, сценарий работы с «Тортугой» – это работа с приготовленными уроками, которые открываются по ссылке вида:

http://tortuga/index.html?b4/wMsYxLAwL/+Ml5/HjEw (1)

Урок содержит тексты задачи, ссылки на картинки–обазцы, поэтому ссылка получается довольно длинной, например:

http://tortuga/index.html?7VcxTkMxDL2K9eeqB+gMGys
TZTCJWywlTkhshIS4O04LN2AA5O3nx35+fk+OkvdNWQ
tth+2uDarAfVqF3EobMFkBK+kOUpNJSUltAGbuPDmxn
IEK++6k7BlAbLO2fBSl2j2dJXHmbKJgCgWfvACQXsEJ
Kp4FAQu/GO7hXoGEq6ND5fXx6kusu6O8GE+QNnVYBnq
jkVhRuQlYKVhTu0KvIKe1Sl0wuXswEDr36qzaUS5NeD

Hdw83CRFMCHuZcrv2ywKA+6Jkk0/Dm/cdrK9a9Hjkhb
xZoTnIoLuVbJu/J4GRnRgVZlKDj8IWNPdy+JepKtrRG
VpKSMnjknXOqCujyVH6aJxJlpRLLS+brHQE7wLa6eRa
I2SaNNZubWURwSUSuyLzS1ur+7Dwz1u47TZ28ed2eHi
P0fxXvs6Rwss4ZsPCsDAs/JET1cPS4L70iZM1xjIs/A
1j+bGLi2uMZFgYb4/wMsYxLAwL/+Ml5/HjEw== (2)

Делиться ссылкой такого размера, отправлять по почте, размещать в соц–сетях, использовать в презентациях весьма неудобно. В связи с чем появилась необходимость укорачивания ссылки до 20 знаков и менее. Уже довольно давно существуют способы их сокращения при помощи сайтов–сокращателей таких как: *www.bitly.com, www.gg.gg, www.goo.gl* и др. Необходимо зайти на соответствующую страницу и после ввода в текстовое окно длинной ссылки, создастся короткая, вида:

http://goo.gl/fbsS (3)

при переходе по которой происходит перенаправление на длинную. Но такое ручное использование удобно только для единичных вариантов. Что бы создателю уроков не приходилось каждый раз выполнять все эти рутинные операции, возникла задача автоматизировать процесс сокращения ссылок. Для этого было решено воспользоваться API, которое предоставляет Google. Была найдена и использована сторонняя библиотека Jsonlib [24] (автор Девид Бау/David Bau). С ее помощью можно обмениваться запросами в формате JSON технологией AJAX не напрямую, а через сторонний сервер, который осуществляет обращения и возвращает ответ. В связи с чем были созданы функции *getShortenURL* и *parseShortenedResponse*. Одна из которых отправляет запрос с передачей длинной ссылки, а вторая, получив ответ в формате JSON, и, распарсив, выводит на страницу constructor.html в текстовое окно. Если по какой-либо причине запрос не был осуществлен или ответ не был получен, на страницу выводится длинный URL (см. приложение A).

2.2 Очистка экрана

При выполнении урока ошибки в алгоритмах рисования неизбежны, и единственный способ очистки экрана-это обновление страницы. Но сраница долго перегружается и приходится заново создавать черепаху с соответствующими настройками положения на экране, ее цветом и толщиной рисования, что отвлекает и создает неудобства. Решением проблемы была бы

возможность очистки экрана без обновления страницы. Для этого в библиотеку функций была добавлена еще одна публичная функция *clearCanvas()*, очищающая прямоугольник размером с Canvas (см. приложение А). Помимо решения такой проблемы очистка экрана дает дополнительные возможности в программировании, например создание анимации.

2.3 Изменение толщины рисования

Дополнительное разнообразие достигается при возможности изменения толщины рисования черепахи. Для реализации этого функционала в объект, представляющий черепаху, добавлено свойство width, а в прототип (объект, представляющий свойства и методы, общие для всех черепах) была добавлена функция *setWidth*, внутри которой передаваемая в качестве параметра толщина присваивается конкретной черепахе. Так как доступ к графическому объекту Canvas доступен через графические методы, с помощью которых манипуляции с черепахой отражаются на Canvas, то добавлена открытая функция *setWidth* (см.приложение А) [7].

2.4 Редактирование урока

Во время использования созданных уроков возможно выявление неточностей, ошибок в орфографии и других недочетов, которые требуют исправления. Для этого функционал приложения был дополнен инструментом для редактирования уроков. Для его создания на страницу создания уроков *constructor.html* было добавлено текстовое поле *lessonInput*, в которое вставляется ссылка на уже имеющийся урок. При нажатии на кнопку «Изменить урок» данные из *lessonInput* передаются в свойство *givLessonArea* объекта Tortuga для вызова закрытой функции *updateArea*, которая, в свою очередь, извлекает текст из ссылки, разархивирует, парсит, и вставляет в текстовое окно в том виде, в котором он был введен. После чего текст урока можно править и далее, при нажатии кнопки «создать урок», формируется новая ссылка на исправленный урок. (см. приложение А) [11, 12].

2.5 Вид концов толстых линий

У большинства детей слово «программирование» ассоциируется с чем-то сложным, неизвестным, пугающим. Из-за этого интерес к обучению теряется, и задача показать, что программирование это не скучное и нудное, а творческое занятие со множеством решений. Ребенок всегда пытается провести аналогию с тем, что когда либо видел или трогал, и помочь в изучении мы

можем дав ему подсказку. Например, показав что рисовать картины можно не только с помощью кисти, но и программируя действия черепахи, сопоставляя программирование и кисть- как инструменты для создания.

Важным элементом рисования является закрашивание объектов, и для этого иногда удобнее использовать линии с прямыми концами, а не скругленными. Например линии с квадратными концами будут создавать видимые изломы на поворотах. А значит, при такой основе как графическое взаимодействие, важно иметь возможность изменять концы линий в ходе работы программы. Это добавит гибкости в построении заданий и мышлении ребенка.

Для этого мною. В лексический анализатор добавлены функции, которые может использовать пользователь для управления черепахой, также пополнил синтаксический анализатор и интерпретатор команд функциями *capsSquare / capsRound и runCapsRound / runCapsSquare* соответственно [приложение А7] и написал вызов команд, работающих с холстом Canvas.

2.6 Перестроение урока без перезагрузки страницы

Формирование страниц Тортуги происходит динамически выделением из адресной строки переданных данных, после анализа и обработки которых генерируется страница. Например

http://trtg.org/index.html?001nVRNbxJRFP0rL7N /Fw==#2

Если же после построения страницы адресная строка в ходе каких-либо действий изменяется, то для отображения изменений необходимо вновь обработать URL, и перестроить страницу. Единственным готовым решением является перезагрузка страницы, но это приводит к сбрасыванию изменений в canvas и дополнительному обращению к серверу, что негативно сказывается на процессе выполнения заданий.

При изучении данной проблемы оказалось, что современные браузеры отслеживают такое событие как *onhashchange* - изменение хештега. На основе этого события можно изменять атрибуты уроков без обновления страницы, однако для браузеров не поддерживающих *onhashchange* event пришлось использовать таймеры проверки изменений тега. См. приложение[]

2.7 Изменение координат и угла поворота

Во время выполнения урока такой инструмент как черепаха приходится перемещать по всему полю в разные его части, а такие действия возможны

только указаниями угла поворота и количестве шагов которые необходимо выполнить. Очевидно этот способ управления требует точности, чем весьма не удобен, и требует отвлекающих подсчетов. Например при начале рисования из определенной точки. По этому было поставлено задание по реализации перемещения черепахи в указанные координаты , а так же поворотом относительно оси OX.

Решена данная проблема следующим образом: на каждом уровне взаимодействия с черепахой были добавлены функции *setX()* и *setY()*, принимающие координаты точки нового местоположения черепахи по X и по Y соответственно. А для изменения угла поворота были добавлены методы *setAngle(angle)* и *getAngle()*, устанавливающие и возвращающие угол поворота черепахи в градусах относительно оси OX (против часовой стрелки).

2.8 Обработка событий мыши

Компьютер стал пользовательским в тот момент, когда появилось взаимодействие с интерфейсом через мышь. Благодаря этому, действия стали интуитивно понятными, что снизило уровень необходимых знаний для использования ЭВМ. Именно по этому большинство детей даже дошкольного возраста пользуются компьютером без чьей либо помощи. Видя этот пример из истории, хотелось бы использовать аналогичное решение для проблемы высокого умственного порога для вхождения в ряды программистов.

Основой такого шага вперед является возможность отслеживания мышиных событий на canvas, и уже исходя из них должен вызываться им соответствующий механизм обработки. Для этого в файле *DrawingSystem.js* объекту добавлен метод *convertCoordsTortugaToCanvas(x, y)*, который получает координаты в Тортужной системе координат, а возвращает объект: *{x: 20, y: 40}* - координаты в системе координат canvas, и аналогичный метод *convertCoordsCanvasToTortuga(x, y)* поступающий в обратном порядке. Это необходимо для преобразования системы координат canvas в привычную Декартову. В файле *mouse.js* создал публичный метод *Tortuga.initMouse(drawingSystem)*, и внутри него подписался на все события мыши объекта canvas так, что бы если в глобальном поле *Tortuga.Events* объявлена реакция на произошедшее событие, то оно запускалось.

Обработка событий мыши на объекте canvas является инструментом широкого применения для визуализации программы и ее создания при помощи *Drug&Dropa*.

2.9 Обрабатывание перемещенных файлов

На данный момент выполнение скриптов тротуги выполняется добавлением файла с помощью кнопки «Выбрать файлы» или вводом выполняемого кода непосредственно через консоль браузера. Однако перенос мышкой может заменить целую последовательность кликов. И, самое главное, он упрощает внешний вид интерфейса. Было бы весьма удобно обрабатывать скрипты в виде файлов js и просто выделенного куска кода , перемещая их на страницу тортуги, тем самым ускоряя процесс, не отвлекая школьника от основного занятия.

Современные браузеры отслеживают событие *e.dataTransfer*, использующееся при сохранении данных, перетаскиваемых в его области. Используя API браузера имеем возможность доступа к нему и его обработки. Так как необходимо обрабатывать как файлы, так и выделенный код, перенесенный в специальную область, то в зависимости от типа получаемой информации происходит ее подготовка, и передача в обработчик скриптов.

Глава 3

Рефакторинг архитектуры

3.1 Автоматизация сборки

На рынке веб-приложений важно иметь высокую скорость загрузки, а тем более при работе с детьми. Ведь внимание детей трудно удерживать, а пока ожидается реакция браузера ребенок может увлечься посторонними мыслями и потерять как нить рассуждений, так и желание продолжать обучение. Во время работы программист использует табуляцию, пробелы и перевод строки, которые пропускаются компилятором при обработке, но важны для создания удобочитаемого и воспринимаемого человеком кода. Это увеличивает размер передаваемых файлов, но допускает возможность их дальнейшей доработки и поиску возникающих ошибок. А значит возникает выбор между уменьшением времени загрузки страницы и удобочитаемостью.

Для ускорения загрузки страницы передаваемые скрипты и css можно минифицировать, но тем самым осложняется отладка продукта, что говорит о

неудобстве разработчиков. по этому решением было бы разделение проекта на девелоп и релиз версии.

По этому передо мной стояла задача по автоматизации сборки проекта по общему файлу-шаблону для всех страниц с подключением к каждой странице как стандартного набора скриптов, так и индивидуальных js-файлов соответствующих урокам. Так же иметь возможность подключать как только минифицированные файлы js и css для клиентской release -версии приложения, так и только исходный удобочитаемый набор файлов для debug-версии разработчиков. Помимо этого уметь сжимать необходимые js-файлы в один файл *min.js* и css-файлы в один *min.css*.

Большинство задач, с которыми приходится сталкиваться программистам, уже давным-давно решены другими членами нашего сообщества. Классическим сборщиком является Make, до сих пор являющийся стандартом де факто при сборке программного обеспечения для Linux из исходников. Это простой, старый и при этом могучий инструмент, до сих пор не утративший своей актуальности, однако требующий умения писать shell-скрипты и пользоваться утилитами Unix, и , что не маловажно, быстро увеличивает рост сложности проекта. В отличие от Make, утилита Ant (Another Neat Tool) полностью независима от платформы, требуется лишь наличие на применяемой системе установленной рабочей среды Java — JRE. Отказ от использования команд операционной системы и формат XML обеспечивают переносимость сценариев. Имеет встроенные задачи, но их очень мало для фронтенда. Gradle-построен на принципах Apache Ant и Apache Maven, но предоставляющая Предметно-ориентированный язык на основе языка Groovy вместо традиционной XML-образной формы представления конфигурации проекта.

Взрыв инструментов языка JavaScript, помогающих оптимизировать рабочий процесс front-end-разработчиков перешел далеко за свои пределы. Все основные инструменты, которые используются для тестирования и разработки на JavaScript активно поддерживаются Node.js. Одним из современных продуктов сборки является Grunt. В отличии от других ему подобных инструментов Grunt создавался специально для фронтенд-разработчиков. И сам Грант, и расширения для него, и даже конфиг написаны на знакомом им языке-JavaScript. Этот инструмент пользуется большой популярностью у веб-разработчиков по всему миру и позволяет не только экономить время, избавляя от монотонных задач, но и выполнять то, что раньше автоматически было делать невозможно. В первую очередь нам интересны такие возможности как:

- проверка javascript-кода на ошибки;
- склеивание и минификация исходников в один файл;
- запуск unit-тестов;

- отслеживание изменений исходных файлов и автоматический перезапуск необходимых задач;

Расширения Grunt продолжают разрабатываться и сегодня, увеличивая его функциональность. Разнообразие его возможностей позволяет подобрать необходимые инструменты и наладить их наиболее подходящим для нашей ситуации образом. Ведь каждое решение по проектированию, которое принимается, будет иметь некоторый набор результатов, в который, как минимум, должно быть включено удовлетворительное решение поставленной задачи. И от этого набора полученных результатов зависит дальнейшая перспектива развития продукта Тортуга. В качестве сборщика был выбран Grunt.

В роли шаблонизатора имеется возможность использовать Assemble, поскольку он позволяет осуществить настройку таким образом, что сборка страниц может происходить с подключением шаблонов, присущих только некоторому набору страниц, а так же передавать файлы json формата.

Подключение js и css-файлов происходит из разделов src/js и src/css благодаря расширению *sails-linker*, имеющий возможность подключать файлы любого формата в любом шаблоне, и к тому же имеет разные настройки для debug и release версий.

copy осуществляет копирование файлов по необходимым разделам , и к тому же имеет возможность изменения

uglify минимизирует и склеивает исходные файлы в min.js и min.css.

clean целиком очищает раздел, в который организована сборка проекта.

watch автоматически перезапускает сборку проекта при изменении выбранного набора файлов.

Вместе они позволили настроить сборщик Grunt таким образом, что бы при внесении изменений автоматически собирать как release-версии с минифицированными файлами, так и debug-версии.

3.2 Рефакторинг архитектуры

Изначально рассчитывалось, что Тортуга будет небольшим приложением, но со временем требования к нему росли. Необходимые усовершенствования и доработка функционала сделали ее несколько более сложной и взаимосвязанной. А значит, что при дальнейшем росте зависимость между ее компонентами будет расти, что повышает актуальность проблемы создания гибкой расширяемой архитектуры.

Передовые программисты уже не раз сталкивались с подобными задачами, решение которых проанализировано, объяснено, проверено на практике и сформированы общепринятые паттерны проектирования для создания такой

архитектуры. Необходимые определения и способы взаимоотношения компонентов воссозданы во фреймворках, и в качестве подходящего мы выбрали AngularJS.

Раньше структура приложения Тортуга была связной. Вызывалась функция initApp(), которая устанавливала зависимости между существующими html-документами, создавала нужные данные и вспомогательные html-элементы, устанавливала обработчики событий. Другими словами вносила логику и устанавливала связи между распределенными по разным файлам частями. Основными компонентами были: контейнер поля с черепахой и контейнер списка задач урока и текстом. Команды, вызванные пользователем либо из консоли, либо из файла, попадают в лексический анализатор, который формирует из них набор лексем, и передает в синтаксический анализатор, перерабатывающий их и строящий дерево программного кода. Команды управления черепахой представляют из себя отдельный язык, не похожий на javascript, по этому дерево кода передается в интерпретатор команд черепахи и с помощью интерфейса рисования отображаются результаты на экране.

Рисунок 3.1: Структура была.

В следствии рефакторинга структуры модули приложения Тортуга были переработаны и строго разделены по смыслу на данные, директивы, контроллеры и сервисы. Благодаря стандартизации AngularJS наш сложный initApp() с описанием действий по вызову модулей и их связям отсутствует. Все ком-

поненты передаются друг в друга за пределами Тортуги, а сама передача осуществляется с помощью фреймворка, который создает экземпляры и связывает их исходя из зависимостей, описанных в конфигурации.

В шаблоне страницы содержится тег <tbox_tortoise_canvas/>, являющийся директивой и отвечающий за внешние преобразования, в том числе за размещение дополнительных html-элементов. Эта директива напрямую связана с контроллером, обеспечивающим связь между такими сервисами как Менеджер Мышиных Команд и Диспетчер Сообщений. Менеджер отвечает за вызов функций, подписанных на события мыши в области canvas, а Диспетчер Сообщений за поддержку связи сервисов - модулей Тортуги, с контроллером.

TortoiseGlobals-интерфейс пользователя. Он регистрирует все глобальные функции, которые может вызывать пользователь. Полученные им команды передаются в синтаксический анализатор, который строит дерево команд и передает его в интерпретатор - исполнитель дерева команд черепахи, а он, в свою очередь, связан только с Диспетчером Сообщений, в который отправляет переработанные команды в виде элементарных графических операций.

В Диспетчер Сообщений, следуя настройкам конфигурации, фреймворк передает в качестве параметра TortoiseCanvasBlock, который получает сообщения о графических операциях, и исполняет их.

Рисунок 3.2: Структура стала.

Описав такую зависимость в конфигурационном файле Ангуляр сам объявляет и запускает необходимые компоненты, и отвечает за передачу данных

между ними. Использование AngularJS в проекте стандартизировало компоненты и связи между ними с точки зрения фреймворка. Таким образом у нас внедрен инструмент, позволяющий расширять проект и управлять модулями заранее декларированным образом, что делает рост проекта более управляемым.

Заключение

Web-приложение «Тортуга» создано для обучения школьников основам программирования, и для достижения максимального результата этот процесс должен быть удобен как ученику, при выполнении заданий, так и учителю, при формировании урока. Длинные ссылки вида (2), которыми затруднительно делиться, использовать в презентациях, размещать в соц-сетях, отсутствие инструмента для редактирования готовых уроков, отсутствие функции очистки экрана являлись слабым местом приложения, требующие исправления.

Возможность автоматизации сборки веб-приложения обладает самым крупным потенциальным резервом для повышения эффективности разработки, снижения требуемых материальных и трудовых ресурсов, сокращения монотонной работы, повышения производительности разработчиков и качества выпускаемого продукта. Тем самым ускорив его разработку.

Достигнутые результаты:

- Реализовано сокращение ссылок
- Добавлена функция очистки экрана
- Изменение толщины рисования
- Написан модуль по редактированию уроков
- Добавлено изменение вида концов толстых линий
- Реализовано перестроение урока без перезагрузки
- Добавлена возможность изменения координат и угла поворота
- Написан модуль обработки событий мыши
- Поддержка перемещенных файлов
- Реализована автоматизация сборки
- Произведен рефакторинг архитектуры

Литература

1. Blockly. Visual programmingeditor. URL: https://code.google.com/p/blockly/ (датао бращения 06.06.2013)
2. Codecademy. URL: http://www.codecademy.com/ (дата обращения 06.06.2013)
3. MicrosoftKodu. URL: http://fuse.microsoft.com/projects/kodu (дата обращения 06.06.2013)
4. Обмен данными для документов с разных доменов. URL: http://javascript.ru/ajax/cross-domain-scripting (дата обращения 03.06.2013)
5. XMLHTTPRequest: описание, применение, частые проблемы. URL: http://xmlhttprequest.ru/ (датао бращения 29.05.2013)
6. Современный учебник по JavaScript. URL: http://learn.javascript.ru/ (дата обращения 13.06.2013)
7. Прототип объекта. URL: http://javascript.ru/Object/prototype (дата обращения 01.06.2013)
8. Дронов В. JavaScript народные советы. Санкт-Петербург: БХВ-Петербург, 2012 год.
9. URL Shortener API. URL: https://developers.google.com/url-shortener/v1/getting_started (дата обращения 06.06.2013)
10. googleapi Google APIs Client Library for JavaScript (Beta). URL: https://developers.google.com/api-client-library/javascript/samples/samples (дата обращения 07.06.2013)
11. Поиск элементов в DOM. URL: http://javascript.ru/tutorial/dom/ search (дата обращения 07.06.2013)
12. Работа со строками. URL: http://htmlweb.ru/java/string.php (дата обращения 14.06.2013)

13. Шапошникова С.В. изучение языка программирования Logo в среде KTurtle: Курс для детей и подростков по программированию: лаборатория юного линуксоида, 2011 год.

14. Welcome to MSWLogo. URL: http://www.softronix.com/logo.html (дата обращения 16.06.2013)

15. Scratch. URL: http://scratch.mit.edu/ (дата обращения 17.06.2013)

16. Стефанов С. JavaScript Шаблоны. Санкт-Петербург – Москва. O'REILLY, 2011 год.

17. MIT App Inventor. URL:http://appinventor.mit.edu/(дата обращения 17.06.2013)

18. Alice. URL: http://www.alice.org/index.php (дата обращения 17.06.2013)

19. Online Logo. URL: http://www.transum.org/software/Logo/ (дата обращения 17.06.2013)

20. Logo Interpreter. URL: http://www.calormen.com/logo/ (дата обращения 17.06.2013)

21. Logo Tortoise. URL: http://andyhd.github.io/Logo-Tortoise/ (дата обращения 17.06.2013)

22. Tortue Logo. URL: http://tortue-logo.fr/en/logo-turtle (дата обращения 17.06.2013)

23. Papert. URL: http://logo.twentygototen.org/EcsvFC4p (дата обращения 17.06.2013)

24. JsonLib. URL: http://call.jsonlib.com/ (дата обращения 17.06.2013)

Приложение А

А1. Сокращение ссылки

Отправляет запрос с передачей длинной ссылки:

```
var parseShortenedResponse = function(m, longUrl) {
    console.log(m, m.content);
    var url = null;
    try {
        url = JSON.parse(m.content).id;
        if (typeof url != 'string') url = longUrl;
    } catch (e) {
        url = longUrl;
    }
linkarea.innerHTML = "";
var textinput = document.createElement("INPUT");
textinput.type = "text";
textinput.disabled = true;
textinput.value = url;

var link = document.createElement("A");
link.href = url;
link.innerHTML = "Try lesson";

linkarea.appendChild(textinput);
linkarea.appendChild(link);
textinput.select();
}
```

Получает ответ в формате JSON и, распарсив, выводит на страницу constructor.html в текстовое окно.

```
var getShortenURL = function(url) {
    jsonlib.fetch({
            url: 'https://www.googleapis.com/urlshortener/v1/url',
            header: 'Content-Type: application/json',
            method: 'POST',
            data: JSON.stringify({longUrl: url})
        },
        function(m){parseShortenedResponse(m, url)});
    }
```

A2. Очистка экрана

Очищает прямоугольник, размером с Canvas

```
...
var clearCanvas;
Tortuga.initDrawing = function(canvas){
...
    clearCanvas = function() {
        ctx.clearRect(0, 0, canvas.width, canvas.height);
    }
}
```

А3. Изменение толщины рисования

В объект, представляющий черепаху, добавлено свойство width:

```
var Tortoise = function(xx, yy, color, width, tortoiseContainer) {
    ...
    this.width = width || 1;
    ...
}
```

В прототип (объект, представляющий свойства и методы, общие для всех черепах) была добавлена функция setWidth:

```
var proto = {
    go : function(t, length){
        ...
        if(t.isDrawing){
        ...
        oldWidth = setWidth(t.width);
        ...
        setWidth(oldWidth);
        }
    },
    ...
    setWidth : function(t, w){t.width = w || t.width}
}
```

Добавлена публичная функция setWidth для отражения на Canvas действий черепахи:

```
Tortuga.initDrawing = function(canvas){
    ...
    setWidth = function(width){
        ctx.lineWidth = width;
    }
}
```

А4. Редактирование уроков

Формирование необходимых элементов для ввода и вывода информации:

```
<input type="text" id="lessonInput">
<button id="changebutton">Изменить урок</button><br>
<textarea id="area"></textarea><br>
<button id="createbutton">Создать урок</button><br>
<div id="linkarea"></div>
```

Передача данных из текстового поля lessonInput в объект Черепаха:

```
<script>
var init = function(){
    ...
    Tortuga.givLessonArea(document.getElementById("area"),
        document.getElementById("changebutton"),
        document.getElementById("lessonInput"));
}
</script>
```

Вызов закрытой функции updateArea:

```
Tortuga.givLessonArea = function(area, button, input){
    button.onclick = function(e){updateArea(area.value, input.value)}
}
```

Обрабатывание полученных из lessonInput данных:

```
var updateArea = function (areaValue, inputValue){
    var t =  Tortuga.ParamsUtil.getLessonTextFromGetUriValue(inputValu
    var paramBegin = null;
    var paramAnd = null;
    var paramText = null;
    var resultText = "";
```

```
    paramBegin = t.indexOf(':"');
    paramAnd = t.indexOf('"', paramBegin + 2);


    while (paramBegin > 0){
        paramText = t.substr(paramBegin + 2, paramAnd - paramBegin - 2
        resultText = resultText + paramText + '\n\n';
        t = t.substr(paramAnd + 2);
        paramBegin = t.indexOf(':"');
        paramAnd = t.indexOf('"', paramBegin + 2);
    }

    document.getElementById('area').value = resultText;
}
```

Извлечение текста урока из ссылки:

```
var getUriValue = function(u){
    var str = u;
    var vhozhd = str.indexOf('?');
    var result = str.substr(vhozhd + 1);
    return result;
}
var getLessonTextFromGetUriValue = function(u){
    return prezip_to_utf8(RawDeflate.inflate(atob(getUriValue(u))));
}
```

A6. Изменение урока # без перезагрузки страницы

описание действия

```
if ("onhashchange" in window)
    {
        window.onhashchange = function ()
        {
            replacementDOMListOfTabs(bg, list, descrDiv, env, lesson);
        }
    }
```

```
    else
    {
        var storedHash = window.location.hash;
        window.setInterval(function ()
            {
                if (window.location.hash != storedHash)
                {
                    replacementDOMListOfTabs(bg, list, descrDiv, env,
                }
            }, 100);
    }

var buildListOfTabs = function(bg, list, descrDiv, env, lesson)
{
    list.appendChild(createList(lesson.items, bg, descrDiv,
    new LessonEnv(env, lesson.title)));
}

var replacementDOMListOfTabs = function(bg, list, descrDiv, env, lesso
{
    for(var i=1; i<=list.children.length; i++)
    {
        var child = list.children[i];
        list.removeChild(child);
    }

    buildListOfTabs(bg, list, descrDiv, env, lesson);
}
```

A8. Методы изменения координат угла и поворота черепашки

TORTOISE

```
var setCoords = function(jsConverter, jsTortoise, x, y)
    {
        if (typeof x == "number")
        {
```

```
            jsConverter.parseNode(jsConverter.nodes.setX, jsTortoise,
            jsConverter.parseNode(jsConverter.nodes.setY, jsTortoise,
        } else
        {
            jsConverter.parseNode(jsConverter.nodes.setX, jsTortoise,
            jsConverter.parseNode(jsConverter.nodes.setY, jsTortoise,
        }
    }

var getAngle = function(jsConverter, jsTortoise)
    {
    return jsConverter.parseNode(jsConverter.nodes.getAngle, jsTortois
    }
```

Добавлены в конструктор комманд

```
var constructProto = function(jsConverter)
    {
        return {
            .........
            getAngle: jsConverter.nodes.getAngle,
            setAngle: jsConverter.nodes.setAngle,
        }
    }
```

добавлены в свойства черепахи

```
Tortoise.prototype.getAngle = function()
        {
            return getAngle(jsConverter, this.jsTortoise)
        }
```

JS CONVERTER добавлены ноды

```
.........
    var NODE_GET_ANGLE  = new FirstParamIsVariableResultNode(TR.comman
    var NODE_SET_ANGLE  = new FirstParamIsVariableNode(TR.commands.set
    var NODE_SET_X      = new FirstParamIsVariableNode(TR.commands.set
    var NODE_SET_Y      = new FirstParamIsVariableNode(TR.commands.set

JsConverter.prototype.nodes = {
        .........
        setX     : NODE_SET_X,
        setY     : NODE_SET_Y,
```

```
        getAngle :  NODE_GET_ANGLE,
        setAngle :  NODE_SET_ANGLE,
    }
```

JS CONVERTER

```
var runGetAngle = function runGetAngle(runner, getTrTortoise, handler)
    {
        handler(getTrTortoise().deg)
    }

var runSetAngle = function runGetAngle(runner, getTrTortoise, deg)
    {
        getTrTortoise().deg = deg
    }
var runSetX = function runSetX(runner, getTrTortoise, x)
    {
        getTrTortoise().x = x
    }
var runSetY = function runSetX(runner, getTrTortoise, y)
    {
        getTrTortoise().y = y
    }
TortoiseRunner.commands = {
        .......
        setX     : runSetX,
        setY     : runSetY,
        getAngle : runGetAngle,
        setAngle : runSetAngle,
        .......
    }
```

A9. Обработка всех мышинных событий в рамках канвы

DRAWINGSISTEM

добавлены методы возвращающие координаты черепахи в координатах canvas, и наоборот

```
var convertCoordsTortugaToCanvas = function(ctx, tortugaX, tortugaY)
    {
```

```
        return {x : tortugaX, y : (ctx.canvas.height - tortugaY)}
    }

var convertCoordsCanvasToTortuga = function(ctx, canvasX, canvasY)
    {
        return {x : canvasX, y : (ctx.canvas.height - canvasY)}
    }
```

MOUSEJS

```
ns("Tortuga.Events");
(function()
{
    var tortugaEventsHolder = Tortuga.Events;

    Tortuga.initMouse = function(drawingSystem)
    {
        var handlerEvent = function(e)
        {
            var point = drawingSystem.convertCoordsCanvasToTortuga(e.l
            var event_canvas =
                {
                    tortugaX: point.x,
                    tortugaY: point.y,
                    originalEvent: e
                }

            var event_name = "on" + e.type
            if (typeof tortugaEventsHolder[event_name] == "function")
            {
                tortugaEventsHolder[event_name](event_canvas)
            }
        }

        var canvas = drawingSystem.getCanvas()
        var events = [
            "onclick", "onmousedown", "onmouseenter", "onmouseleave",
            "onmousemove", "onmouseout", "onmouseover", "onmouseup",
            "onmousewheel"
        ]
        events.forEach(function(event_name)
        {
```

```
            canvas[event_name] = handlerEvent
            tortugaEventsHolder[event_name] = null
        })
    }
})()
```

A10. D&D файлов

FILES.JS

```
var processScript =  function(script)
    {
        preAction();
        var scriptElement = document.createElement("script");
        scriptElement.innerHTML = script;

        var headElement = document.getElementsByTagName("head")[0];
        headElement.appendChild(scriptElement);
        postAction();
    }

var processFile = function(file)
    {
        var reader = new FileReader();
        reader.onload = function(e)
        {
            processScript(e.target.result)
        };

        reader.readAsText(file);
    }


var processText = function(script)
    {
        processScript(script)
```

```
    }

var preventDefault = function(e)
    {
        e.preventDefault ? e.preventDefault() : (e.returnValue = false
    }


var doDrop = function(e) {
    canvasObjekt.classList.remove("tortuga-canvasContainer-dragging")
        if (e.dataTransfer.files)
        {
            var file = e.dataTransfer.files;
            for (var i=0; i<file.length; i++)
            {
                processFile(file[i]);
            }
            preventDefault(e)
        }

        if (e.dataTransfer.getData('Text'))
        {
            var text = e.dataTransfer.getData('Text');
            processText(text);
            preventDefault(e)
        }
        return false;
    }

var handleDragOver = function(e)
    {
        preventDefault(e)
        canvasObjekt.classList.add("tortuga-canvasContainer-dragging")
        console.log("rrr");
    }
var handleDragLeave = function()
    {
        canvasObjekt.classList.remove("tortuga-canvasContainer-draggin
    }
```

```
...
canvasObjekt.addEventListener("dragleave", handleDragLeave, false);
canvasObjekt.addEventListener("dragover", handleDragOver, false);
canvasObjekt.addEventListener("drop", dodrop, false);

    }
```