**Сопроводительный лист на издание в авторской редакции**

Название работы \_\_\_ Создание Single Page Application (SPA) – на примере простой игры. \_ Методические указания к выполнению лабораторной работы № 3 для студентов очной формы обучения по направлениям подготовки 09.03.04 – «Программная инженерия», 09.03.01 – «Информатика и вычислительная техника» 02.03.03 – «Математическое обеспечение и администрирование информационных систем»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Актуальность и соответствующий   
научно-методический уровень подтверждаю \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись научного редактора)

Рукопись сверена и проверена автором \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись автора)

Рекомендуется к изданию \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись заведующего кафедрой)



**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Брянский государственный технический университет

Утверждаю

Ректор университета

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Н. Федонин

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017

**Специальные разделы web программирования**

**Создание Single Page Application (SPA) – на примере простой игры.**

**Методические указания**

**к выполнению лабораторной работы №3**

**для студентов очной формы обучения**

**по направлениям подготовки**

**09.03.04 – «Программная инженерия»**

**09.03.01 – «Информатика и вычислительная техника»**

**02.03.03 – «Математическое обеспечение и администрирование информационных систем»**

**Брянск 2017**

**УКД 004.**

Создание Single Page Application (SPA) – на примере простой игры. [Текст] + [Электронный ресурс]: Методические указания к лабораторной работе №3 для студентов очной формы обучения по направлениям подготовки 09.03.04 – «Программная инженерия», 09.03.01 – «Информатика и вычислительная техника» 02.03.03 – «Математическое обеспечение и администрирование информационных систем». – Брянск: БГТУ, 2017. – 24 с.

Разработали:

Д.Г.Лагерев

канд. техн. наук, доц.,

Д.А.Касницкий

ассистент

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ

**Методические издания публикуются в авторской редакции**

# Цель работы

Целью работы является ознакомление с принципами разработки не сложных Single Page Application с применением распространённых архитектурных принципов (MVC, MVVM, MVP).

Продолжительность работы – 4 часа.

# Системы сборки frontend-проекта: gulp, grunt и альтернативы

Таск-раннеры и системы сборки сильно ускоряют работу, автоматизируя компиляцию, тестирование и другие рутинные задачи. Как и в любой другой области, на этом рынке существует сильная конкуренция. До 2014 года среди них главенствовал таск-раннер grunt, но позже из состава проекта выделилась небольшая команда, которая решила делать альтернативный инструмент, gulp, ориентированный на сборку проекта.

Чтобы помочь вам определиться с выбором, в рамках статьи рассмотрим основные таск-менеджеры:

* gulp
* grunt

Забегая немного вперед, скажем, что мы в WaveAccess пользуемся именно gulp. Внедрить инструмент оказалось очень легко: у семейства продуктов JetBrains (IDEA, WebStorm, ReSharper), которые мы используем уже много лет, есть отличные плагины для работы с gulp/grunt и npm/nodejs.

## Таск-менеджер vs. система сборки проекта: в чем разница?

Таск-менеджер — инструмент для автоматизации задач. В конфигурации раннеров можно записать имена этих задач; функцию, которая их выполняет; плагины для ускорения стандартных действий, но сами задачи могут быть произвольными. Например:

* Задачи для деплоя (zip проекта, загрузка проекта на удаленный сервер и тп)
* Задачи по сборке проекта (минификация, оптимизация, проверка кода на валидность и тп)
* Задачи для миграции данных и т.д.

Примеры таких инструментов — grunt и gulp.

Система сборки — это инструмент, который решает только одну типовую задачу сборки проекта на java script, в которую входят:

* конкатенация,
* проверка кода на валидность,
* минификация кода, и тд.

К подобным инструментам относятся Webpack, Broccoli, Brunch, Browserify и другие.

Все подобные frontend-задачи можно автоматически выполнять при помощи других средств: к примеру, с помощью npm run.

# Системы контроля версий

Что такое "система контроля версий", и почему это важно? Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Для контроля версий файлов в этой книге в качестве примера будет использоваться исходный код программного обеспечения, хотя на самом деле вы можете использовать контроль версий практически для любых типов файлов.

Если вы графический или web-дизайнер и хотите сохранить каждую версию изображения или макета (скорее всего, захотите), система контроля версий (далее СКВ) — как раз то, что нужно. Она позволяет вернуть файлы к состоянию, в котором они были до изменений, вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда, и многое другое. Использование СКВ также значит в целом, что, если вы сломали что-то или потеряли файлы, вы спокойно можете всё исправить. В дополнение ко всему вы получите всё это без каких-либо дополнительных усилий.

## Локальные системы контроля версий

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже, директорию с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.

Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами. Даже популярная операционная система Mac OS X предоставляет команду rcs, после установки Developer Tools. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

## Централизованные системы контроля версий

Следующая серьёзная проблема, с которой сталкиваются люди, — это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как: CVS, Subversion и Perforce, имеют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.

Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать ЦСКВ, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус — это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми он работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё — всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Локальные СКВ страдают от той же самой проблемы — когда вся история проекта хранится в одном месте, вы рискуете потерять всё.

## Децентрализованные системы контроля версий

Децентрализованные системы контроля версий (ДСКВ). В ДСКВ (таких как Git, Mercurial, Bazaar или Darcs), клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени): они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.

Более того, многие ДСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно, в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

# GIT

Что же такое Git, если говорить коротко? Очень важно понять эту часть материала, потому что, если вы поймёте, что такое Git и основы того, как он работает, тогда, возможно, вам будет гораздо проще его использовать. Пока вы изучаете Git, попробуйте забыть всё, что вы знаете о других СКВ, таких как Subversion и Perforce; это позволит вам избежать определённых проблем при использовании утилиты. Git хранит и использует информацию совсем иначе по сравнению с другими системами, даже несмотря на то, что интерфейс пользователя достаточно похож, и понимание этих различий поможет вам избежать путаницы во время использования.

GIT - распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

Примерами проектов, использующих Git, являются Ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt и некоторые дистрибутивы LINUX.

Система спроектирована как набор программ, специально разработанных с учётом их использования в скриптах. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Например, Cogito является именно таким примером оболочки к репозиториям Git, а StGit использует Git для управления коллекцией исправлений (патчей).

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удалённый доступ к репозиториям Git обеспечивается git-daemon, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

Преимущества и недостатки git по сравнению с централизованными системами управления версиями (такими как, например, Subversion) типичны для любой распределённой системы и описаны в статье «Система управления версиями». Если же сравнивать git с «родственными» ей распределёнными системами, можно отметить, что git изначально идеологически ориентирован на работу с изменениями, а не с файлами, «единицей обработки» для него является набор изменений, или патч. Эта особенность прослеживается как в структуре самой системы (в частности — в структуре репозитория), так и в принципах построения команд; она отражается на производительности системы в различных вариантах её использования и на достоинствах и недостатках git по сравнению с другими DVCS.

Часто называемые преимущества git перед другими DVCS:

* Высокая производительность.
* Развитые средства интеграции с другими VCS, в частности, с CVS, SVN и Mercurial. Помимо разнонаправленных конвертеров репозиториев, имеющиеся в комплекте программные средства позволяют разработчикам использовать git при размещении центрального репозитория в SVN или CVS, кроме того, git может имитировать cvs-сервер, обеспечивая работу через клиентские приложения и поддержку в средах разработки, специально не поддерживающих git.
* Продуманная система команд, позволяющая удобно встраивать git в скрипты.
* Репозитории git могут распространяться и обновляться общесистемными файловыми утилитами архивации и обновления, такими как rsync, благодаря тому, что фиксации изменений и синхронизации не меняют существующие файлы с данными, а только добавляют новые (за исключением некоторых служебных файлов, которые могут быть автоматически обновлены с помощью имеющихся в составе системы утилит). Для раздачи репозитория по сети достаточно любого веб-сервера.

В числе недостатков git обычно называют:

* Отсутствие сквозной нумерации коммитов монотонно непрерывно возрастающими целыми числами. Во многих проектах используется автоматическое получение номера этой версии (например, командой svnversion), построение .H файла на основе этого числа, и далее его использование при создании штампа версии исполняемого файла, некоторых вшитых в него строк и так далее.
* Отсутствие переносимой на другие операционные системы поддержки путей в кодировке Unicode в Microsoft Windows (для версий msysgit до 1.8.1). Если путь содержит символы, отличные от ANSI, то их поддержка из командной строки требует специфических настроек, которые не гарантируют правильного отображения файловых имён при пользовании тем же репозиторием из других ОС. Одним из способов решения проблемы для git 1.7 является использование специально пропатченного консольного клиента. Другой вариант — использование графических утилит, работающих напрямую через API, таких как TortoiseGit.
* Некоторое неудобство для пользователей, переходящих с других VCS. Команды git, ориентированные на наборы изменений, а не на файлы, могут вызвать недоумение у пользователей, привыкших к файл-ориентированным VCS, таким как SVN. Например, команда «add», которая в большинстве систем управления версиями производит добавление файла к проекту, в git подготавливает к фиксации сделанные в файлах изменения. При этом сохраняется не патч, описывающий изменения, а новая версия целевого файла.
* Использование для идентификации ревизий хешей SHA1, что приводит к необходимости оперировать длинными строками вместо коротких номеров версий, как во многих других системах (хотя в командах допускается использование неполных хеш-строк).
* Бо́льшие накладные расходы при работе с проектами, в которых делаются многочисленные несвязанные между собой изменения файлов. При работе в таком режиме размеры наборов изменений становятся достаточно велики и происходит быстрый рост объёма репозиториев.
* Бо́льшие затраты времени, по сравнению с файл-ориентированными системами, на формирование истории конкретного файла, истории правок конкретного пользователя, поиска изменений, относящихся к заданному месту определённого файла.
* Отсутствие отдельной команды переименования/перемещения файла, которая отображалась бы в истории как соответствующее единое действие. Существующий скрипт git mv фактически выполняет переименование, копирование файла и удаление его на старом месте, что требует специального анализа для определения, что в действительности файл был просто перенесён (этот анализ выполняется автоматически командами просмотра истории). Однако, учитывая тот факт, что наличие специальной команды для переименования/перемещения файлов технически не вынуждает пользователя использовать именно её (и, как следствие, в этом случае возможны разрывы в истории), поведение git может считаться преимуществом.
* Система работает только с файлами и их содержимым, и не отслеживает пустые каталоги.

В ряде публикаций, относящихся преимущественно к 2005—2008 годам, можно встретить также нарекания в отношении документации git, отсутствия удобной windows-версии и удобных графических клиентов. В настоящее время эта критика неактуальна: существует версия git на основе MinGW («родная» сборка под Windows), и несколько высококачественных графических клиентов для различных операционных систем, в частности, под Windows имеется клиент TortoiseGit, идеологически очень близкий к широко распространённому TortoiseSVN — клиенту SVN, встраиваемому в оболочку Windows.

# Демонстрация MVC

Создадим структуру проекта:

* assets/css/style.css - Здесь будет лежать файл со стилями
* assets/img - Здесь будут лежать наши спрайты
* assets/sounds - Папка для звуков
* js/marioController.js
* js/marioModel.js
* js/marioView.js
* .gitignore
* game.html
* README.txt

Задаём title. Сделаем простую интерпретацию марио. Сверстаем простую главную сцену, на которой будет располагаться земля, персонаж и гриб.

|  |
| --- |
| <div class="mainScene">  <div class="character"></div>  <div class="goomba"></div>  </div> |

Подключим файл стилей. Верстать, сразу видеть, как смотрятся стили можно сразу в любом браузере. Самый быстрый и дружелюбный по интерфейсу – Google Chrome.

Для того, чтобы были видны границы блока выставим border. Затем задаём нужную длину и ширину и автоматически выравниваем по центру по горизонтали. Стилизуем небо и землю линейным градиентом.

|  |
| --- |
| .mainScene {  position: relative;  border: 1px solid black;  width: 800px;  height: 600px;  margin: 0 auto;  background: linear-gradient(to bottom, rgba(0, 0, 255, 0.44) 75%, rgba(0, 128, 0, 0.65) 75%);  } |

Добавим заранее подготовленный спрайт марио.

|  |
| --- |
| .character {  position: absolute;  width: 30px;  height: 40px;  background-image: url(../img/mario.png);  background-repeat:no-repeat;  background-size: 100% 100%;  } |

Подключим в index.html в <head> три скрипта.

|  |
| --- |
| <script src="js/marioModel.js"></script>  <script src="js/marioView.js"></script>  <script src="js/marioController.js"></script> |

Итак, начнем с модели. Пока что у нас на экране будет находиться только один марио. Необходимо знать его дефолтные координаты, где он появляется со стартом игры и его текущие координаты.

Имена классов, как правило пишутся с большой буквы, чтобы не путать с обычными функциями. В данном проекте не будет использоваться ES6, поэтому классы на самом деле будут объявляться с помощью ключевого слова function.

Начальные координаты нашего Марио 0, 0. Сейчас разрабатывается только модель. Как это всё будет отображать view - это здесь не важно. Будем действовать поэтапно. Для начала сделаем так, чтобы он бегал.

|  |
| --- |
| const INITIAL\_MARIO\_X = 0;  const INITIAL\_MARIO\_Y = 0;  const INITIAL\_GOOMBA\_X = -200;  const INITIAL\_GOOMBA\_Y = 0;  const LAND = 0;  const LEFT\_BORDER = -400;  const RIGHT\_BORDER = 370;  var Model = function () {  this.objs = {  'mario': {  x: INITIAL\_MARIO\_X,  y: INITIAL\_MARIO\_Y  }  };  };  Model.prototype.init = function(renderFunction){  this.needRendering = renderFunction;  };  Model.prototype.setCoords = function (obj, x, y) {  x = x == (undefined || null) ? obj.x : x;  y = y == (undefined || null) ? obj.y : y;  checkScreenBorders.call(this, obj, x);  this.needRendering();  };  Model.prototype.getCoords = function (obj) {  return {  x: obj.x,  y: obj.y  }  };  Model.prototype.marioMove = function(e){  var keyCode = e.keyCode;  var x = marioModel.getCoords(marioModel.objs.mario).x;  switch (keyCode) {  case KEY\_CODE\_RIGHT: {  marioModel.setCoords(marioModel.objs.mario, x + MARIO\_STEP);  break;  }  case KEY\_CODE\_LEFT: {  marioModel.setCoords(marioModel.objs.mario, x - MARIO\_STEP);  break;  }  }  };  function checkScreenBorders(obj, x) {  if (!(x <= LEFT\_BORDER || x >= RIGHT\_BORDER)) {  obj.x = x;  }  else {  if (obj.hasOwnProperty('direction')) {  obj.direction = obj.direction === 'right' ? 'left' : 'right';  }  }  }  var marioModel = new Model(); |

Затем view.js.

|  |
| --- |
| var View = function() {  this.mario = document.querySelector('.character');  this.goomba = document.querySelector('.goomba');  this.onKeyDownEvent = null;  };  View.prototype.render = function (objs) {  this.mario.style.left = 'calc(50% + ' + objs.mario.x + 'px)';  this.mario.style.top = 'calc(68.5% + ' + objs.mario.y + 'px)';  };  View.prototype.init = function (){  document.addEventListener('keydown', this.onKeyDownEvent);  };  var marioView = new View(); |

И, наконец, controller.js.

|  |
| --- |
| var Controller = function (View, Model) {  this.marioView = View;  this.marioModel = Model;  };  Controller.prototype.init = function() {  this.marioView.onKeyDownEvent = this.moving.bind(this);  this.marioView.init();  this.marioModel.init(this.needRendering.bind(this));  this.needRendering();  };  Controller.prototype.moving = function(e) {  this.marioModel.marioMove(e);  };  Controller.prototype.needRendering = function(){  this.marioView.render(marioModel.objs);  };  var marioController = new Controller(marioView, marioModel);  marioController.init(); |

Теперь марио может бегать. Добавляем музыку с помощью специального тега. В папку sounds скачиваем нужный mp3 файл. Затем в html добавляем.

|  |
| --- |
| <audio loop autoplay>  <source src="assets/sounds/main.mp3" type="audio/mp3" />  </audio> |

И теперь при загрузке страницы сразу же будет грузиться и воспроизводиться наш файл. Добавим прыжки и тоже со звуковым сопровождением. В html добавим ещё один аудио элемент, который позволит нам воспроизводить прыжок.

|  |
| --- |
| <audio class="jump">  <source src="assets/sounds/jump.mp3" type="audio/mp3" />  </audio> |

Добавим обработку прыжка по клику во View и наш файл примет следующий вид.

|  |
| --- |
| var View = function() {  this.mario = document.querySelector('.character');  this.goomba = document.querySelector('.goomba');  this.jumpSound = document.querySelector('.jump');  this.onKeyDownEvent = null;  this.onClickJumpEvent = null;  };  View.prototype.init = function (){  document.addEventListener('keydown', this.onKeyDownEvent);  document.addEventListener('click', this.onClickJumpEvent);  };  View.prototype.render = function (objs) {  this.mario.style.left = 'calc(50% + ' + objs.mario.x + 'px)';  this.mario.style.top = 'calc(68.5% + ' + objs.mario.y + 'px)';  this.goomba.style.left = 'calc(50% + ' + objs.goomba.x + 'px)';  this.goomba.style.top = 'calc(70% + ' + objs.goomba.y + 'px)';  };  var marioView = new View(); |

Теперь идем в контроллер. Нам необходимо сообщить моделе, что мы прыгаем. В init:

|  |
| --- |
| this.marioView.onClickJumpEvent = this.jumping.bind(this); |

И просто функцию, которая будет сообщать модели, что надо начать просчитывать прыжок:

|  |
| --- |
| Controller.prototype.jumping = function() {  this.marioModel.initMarioJump(this.marioView.jumpSound);  }; |

Добавим анимацию прыжка. Для анимации прыжка будет использован RequestAnimationFrame.

Принцип её действия прост. Она вызывает саму себя, до тех пор, пока не выполнится определенное условие.

Для этого нам надо ещё несколько вспомогательных функций. Можно это реализовать по-разному, в т.ч. просто с постоянно действующей гравитацией, но не будем усложнять. Добавляем в модель вспомогательные функции.

|  |
| --- |
| Model.prototype.isBusy = function () {  return this.busy;  };  Model.prototype.changeBusy = function () {  this.busy = !this.busy;  };  Model.prototype.isUp = function () {  return this.up;  };  Model.prototype.changeUp = function () {  this.up = !this.up;  };  Model.prototype.isJumpEnd = function () {  return this.endJump;  };  Model.prototype.changeJumpEnd = function () {  this.endJump = !this.endJump;  }; |

Инициализируем эти переменные в конструкторе:

|  |
| --- |
| this.busy = false;  this.up = true;  this.endJump = false; |

Добавим в setCoords проверку на прыжок и инициализируем сам прыжок отдельной функцией.

|  |
| --- |
| if (y !== obj.y) {  checkEndJump.call(this, obj, y);  }  Model.prototype.initMarioJump = function (sound) {  if (!marioModel.isBusy()) {  marioModel.changeBusy();  requestAnimationFrame(marioModel.marioJump);  sound.play();  }  };  Model.prototype.marioJump = function(){  var y = marioModel.getCoords(marioModel.objs.mario).y;  if (marioModel.isUp()) {  marioModel.setCoords(marioModel.objs.mario, null, y - MARIO\_STEP);  }  else {  marioModel.setCoords(marioModel.objs.mario, null, y + MARIO\_STEP);  }  if (marioModel.isJumpEnd()) {  marioModel.changeBusy();  marioModel.changeJumpEnd();  marioModel.changeUp();  cancelAnimationFrame(MarioID);  }  else {  requestAnimationFrame(marioModel.marioJump);  }  }; |

Реализуем движение Goomba. В модели добавляем. В init:

|  |
| --- |
| GoombaID = requestAnimationFrame(this.walkingGoomba); |

И добавляем анимацию перемещения

|  |
| --- |
| Model.prototype.checkMarioGoombaCollision = function (mario, goomba) {  var marioLeft = mario.getBoundingClientRect().left;  var marioRight = mario.getBoundingClientRect().right;  var marioY = this.objs.mario.y;  var goombaLeft = goomba.getBoundingClientRect().left;  var goombaRight = goomba.getBoundingClientRect().right;  var goombaY = this.objs.goomba.y;  if (goombaLeft < marioLeft && marioLeft < goombaRight ||  goombaLeft < marioRight && marioRight < goombaRight) {  if (goombaY - marioY >= KILLER\_HEIGHT && goombaY - marioY < GOOMBA\_HEIGHT) {  return 'kill';  }  else if (goombaY - marioY < KILLER\_HEIGHT) {  return 'die'  }  }  return false;  };  Model.prototype.walkingGoomba = function () {  var x = marioModel.getCoords(marioModel.objs.goomba).x;  if (marioModel.objs.goomba.direction === 'right') {  marioModel.setCoords(marioModel.objs.goomba, x + GOOMBA\_STEP);  }  else {  marioModel.setCoords(marioModel.objs.goomba, x - GOOMBA\_STEP);  }  var collision = marioModel.checkMarioGoombaCollision(marioView.mario, marioView.goomba);  if (collision === 'kill') {  // cancelAnimationFrame(GoombaID);  marioController.dieGoomba();  }  else if (collision === 'die'){  marioController.dieMario();  requestAnimationFrame(marioModel.walkingGoomba);  }  else if (!collision) {  requestAnimationFrame(marioModel.walkingGoomba);  }  };  function checkScreenBorders(obj, x) {  if (!(x <= LEFT\_BORDER || x >= RIGHT\_BORDER)) {  obj.x = x;  }  else {  if (obj.hasOwnProperty('direction')) {  obj.direction = obj.direction === 'right' ? 'left' : 'right';  }  }  } |

Добавим во view две функции, которые скрывают марио или гриб в случае их соответствующих столкновений.

|  |
| --- |
| Controller.prototype.dieMario = function() {  this.marioView.mario.setAttribute('hidden', 'true');  this.marioView.render(marioModel.objs);  };  Controller.prototype.dieGoomba = function() {  this.marioView.goomba.setAttribute('hidden', 'true');  this.marioView.render(marioModel.objs);  }; |

Полный исходный код и работающее web-application можно скачать из кафедрального GIT репозитория <http://git.iipo.tu-bryansk.ru/kasnitskiy-dmitriy-alekseevich/For_14PrI.git>.

# Индивидуальные задания

Разработать Single Page Application, используя обычную разметку DOM-дерева.

## Оцениваемые критерии:

1. Умение работать с событиями.
2. Умение работать с DOM-деревом.
3. Качество кода.
4. Оригинальность решения (сценарии, пасхалки и т.д.).

## Примерный порядок выполнения работы:

1. Продумать концепцию приложения, интерфейса
2. Проработать структуру DOM-дерева
3. Проработать модель приложения (см. https://habrahabr.ru/post/215605/)
4. Сверстать основные элементы приложения, используя css (**не** использовать Canvas или SVG)
5. Разработать следующие пункты:
   1. Инициализация приложения (создание/заполнение данными игрового поля, создание модели на основе случайных данных/настроек пользователя)
   2. События, вызываемые действиями пользователя
   3. События, срабатывающее по таймерам
   4. Для игрового блока: начисление очков, условия окончания игры/перехода на следующий уровень
6. Протестировать его в браузере на выбор: Google Chrome, Mozilla Firefox, Edge, Safari (проверка будет так же производится в выбранном вами браузере)

## Обязательные условия для сдачи:

1. Наличие работы в GIT репозитории.
2. Следование одному из паттернов (MVC, MVVM, MVP)
3. Отсутствие JQuery.
4. Работоспособность приложения.
5. Контрольные вопросы.

## При защите работы студент должен:

1. Продемонстрировать результат работы.
2. Продемонстрировать исходный код. Быть готовым дать комментарий касательно его структуры и содержания.
3. Сделать выводы о проделанной работе.

**Таблица 1**

*Варианты заданий*

|  |  |
| --- | --- |
| №  варианта | Задание, которое необходимо реализовать |
|  | Арканоид |
|  | Танки из «Тетриса»  Реализовать в упрощенном варианте игру в танки  Картинки по запросу Танчики тетрис |
|  | Движение с препятствиями  Игра, где присутствует управляемый пользователем объект (например, транспортное средство) и поле игры, изменяющееся с течением времени, на котором располагаются различные препятствия, столкновение с которыми приводит к потере очков/жизней.  Пример одной реализации на html5:  image2017-10-13 21-49-39 |
|  | Пятнашки  При клике на соответствующую цифру - она перемещается в свободное поле рядом, если таковое есть. Или же управление стрелочками. Или Drag'n'Drop. |
|  | Саймон  Simon game.jpg |
|  | Сапёр  Поля достаточно будет 10х10. Можно использовать как левую/правую кнопку мыши для обозначения типа поля, так и одинарный/двойной щелкой левой кнопкой мыши. |
|  | Игра на запоминание цифр  На экране перед пользователем появляется несколько цифр на блоках на короткий промежуток времени, после их исчезновения, пользователь должен выбрать блоки по порядку по возрастанию чисел, ранее на них изображенных. |
|  | Ping Pong  Реализовать игру Ping-Pong. На экране обязательно должны присутствовать две планки, которые возможно контролировать с клавиатуры, а также мяч и счет.  image2017-6-9 12-24-35 |
|  | Гонки из «Тетриса»  Реализовать модифицированный вариант гонок из тетриса. 2 дорожки, управление своей "машиной" с клавиатуры. Предусмотреть так, чтобы всегда было "окно" для проезда и не было безнадежных ситуаций. В качестве разной сложности можно сделать изменение скорости. Счет – опционально.  Картинки по запросу тетрис гонки |
|  | Игра «Жизнь»  Реализовать игру "Жизнь". Реализовать правила игры "Жизнь" в простейшем варианте (поле - плоскость с границами, а не Тор). Пользователь задаёт начальные клетки кликом, нажимает кнопку и наблюдает за эволюцией. |
|  | Упрощенная вариация “Pocket Tanks”  Реализовать простейший случай - танки каждый раз появляются в случайном месте (не пересекаясь друг с другом). Стрелочками у одного из танков изменяется угол дула. Пробел - выстрел.  Сила выстрела и угол выставляются заранее. Необходимо попасть в другой танк снарядом. Реализация передвижений опциональна. |
|  | Упрощенная вариация “Space Invaders”.  «Пришельцы» перемещаются вправо-влево и в случайном порядке стреляют в игрока.  Игрок контролирует свой корабль с помощью стрелочек. Пробел – выстрел. |
|  | Шашки  Шашки на двоих, без искусственного интеллекта, должна учитываться очередность хода и подсвечиваться возможные ходы. |
|  | Карта торгового центра  Карта торгового центра из двух этажей, одновременно показывается один этаж. Помещения отделены друг от друга (произвольной формы), при наведении помещение подсвечивается другим цветом, появляется pop-up с краткой информацией: название, тип и т.п. |
|  | График  Реализовать интерактивный график двух функций:  y = kx + b,  y = kx^2+b  Отображаются функции или нет должно задаваться чекбоксом (не радио button), которые выбирает пользователь. Предусмотреть input для k и b (снаружи canvas).  image2017-6-9 12-8-36 |
|  | Солнечная система  Необходимо реализовать сцену, на которой изображено следующее: в центре Солнце, а на некотором расстоянии от Солнца находятся остальные планеты (реалистично реализовывать нет необходимости). Текстуры для объектов могут быть произвольными. Реализовать вращение 8 планет вокруг Солнца.  image2017-6-9 12-11-19 |
|  | Круги  Реализовать появление закрашенных случайным цветом кругов в произвольных местах экрана каждые 0.5 секунды. В начале круг должен быть с радиусом (20-80)px и разрастаться до нач. R\*2. После чего круг должен быть на экране ещё секунду, а затем исчезнуть.  image2017-6-9 12-14-5 |
|  | Печатная машинка  Реализовать "печатную машинку" (Виртуальную клавиатуру) с интерактивными клавишами (не обязательно реализовывать всю раскладку). Должна присутствовать возможность набирать слова кликая мышкой по клавишам, а также просто с клавиатуры  Картинки по запросу canvas печатная машинка |
|  | Часы  Реализовать часы. У часов в обязательном порядке должны присутствовать часовая, минутная и секундные стрелки.  image2017-6-9 12-11-28Предусмотреть вывод времени (ЧЧ:ММ:СС) в input, которые находятся вне canvas и возможность настраивать время (а не получать системное время). |
|  | Цветной «Ковёр»  Реализовать произвольную сетку MxN, состоящую из произвольно закрашенных квадратов. При наведении курсором на квадрат, последний изменяет свой цвет на отличный от текущего.  image2017-6-9 12-12-15 |
|  | Pipes  Реализовать скринсейвер из windows XP (в 2D). Несколько разноцветных линий в случайно появляющихся местах и после определенного времени меняющие своё направление.  Картинки по запросу Pipes 2D screensaver |
|  | Эквалайзер.  Реализовать эквалайзер средствами Canvas. Т.к. медиа ещё не было пройдено, то в качестве входных параметров для столбцов можно использовать случайно сгенерированные числа.  Картинки по запросу Эквалайзер |
|  | Разноцветные шары  Реализовать движение разноцветных шаров. При столкновении со стеной шар отскакивает на 45 градусов. Реализация столкновений между шарами - опциональна.  image2017-6-9 12-23-49 |

# Контрольные вопросы

1. Что такое Front-end разработка?
2. Чем концептуально веб-сайт отличается от веб-приложения?
3. Что такое SPA?
4. Что такое RIA?
5. Какие функции выполняет система сборки Front-end приложений?
6. Какие варианты решения поддержки дизайна для мобильных устройств вы знаете?

# Список рекомендуемой литературы

1. Браун Э. Изучаем JavaScript. Руководство по созданию современных веб-сайтов. - Мск, Альфа-книга, 2017 - 363с.
2. Фрэйн, Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Б. Фрэйн; [перевод с английского В. Черник]. - Санкт-Петербург: Питер Пресс, 2017. - 272 с.
3. Pro GIT. 2nd Edition –, 2014.- Режим доступа: https://git-scm.com/book/ru/v2
4. Современный учебник Javascript. –, 2017. – Режим доступа: https://learn.javascript.ru

Создание Single Page Application (SPA) – на примере простой игры. [Текст] + [Электронный ресурс]: Методические указания к лабораторной работе №3 для студентов очной формы обучения по направлениям подготовки 09.03.04 – «Программная инженерия», 09.03.01 – «Информатика и вычислительная техника» 02.03.03 – «Математическое обеспечение и администрирование информационных систем». – Брянск: БГТУ, 2017. – 24 с.

ЛАГЕРЕВ ДМИТРИЙ ГРИГОРЬЕВИЧ

КАСНИЦКИЙ ДМИТРИЙ АЛЕКСЕЕВИЧ

Научный редактор Д.И. Булатицкий

Компьютерный набор Д.А. Касницкий

Иллюстрации Д.А. Касницкий

Подписано в печать \_\_\_\_\_\_\_\_\_ г. Формат 60х84 1/16 Бумага офсетная.

Офсетная печать. Усл.печ.л. \_\_\_ Уч.-изд.л. \_\_\_ Тираж 1 экз. Бесплатно

Брянский государственный технический университет

Кафедра «Информатика и программное обеспечение», тел. 56-09-84

241035, Брянск, бульвар 50 лет Октября, 7 БГТУ