Министерство образования и науки РФ

Государственное образовательное учреждение высшего профессионального образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт (факультет) – Кибернетический Центр

Направление (специальность) – Информатика и вычислительная техника

Выпускающая кафедра – Автоматики и Компьютерных Систем

«**СПРАЙТОВАЯ ИГРА НА HTML5 «АРКАНОИД»**»

Пояснительная записка к курсовой работе

по дисциплине «Технология разработки программного обеспечения»

|  |  |  |
| --- | --- | --- |
| Студент гр.8ВМ34 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (дата) | А.С. Лизин |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (дата) | И.А. Заикин |

Томск – 2013

# Содержание

Оглавление

[Содержание 2](#_Toc372018959)

[Введение 4](#_Toc372018960)

[Используемые средства разработки 4](#_Toc372018961)

[Интегрированная среда разработки 4](#_Toc372018962)

[Браузер 6](#_Toc372018963)

[Отладчик и профайлер 6](#_Toc372018964)

[Система контроля версий 7](#_Toc372018965)

[1. Требования к программе 9](#_Toc372018966)

[1.1 Структурированный перечень функциональных и нефункциональных требований к программе 9](#_Toc372018967)

[1.2 Варианты использования 9](#_Toc372018968)

[1.2.1 Вариант использования «Управлять ракеткой с помощью клавиатуры» 10](#_Toc372018969)

[1.2.2 Вариант использования «Управлять ракеткой с помощью курсора мыши» 10](#_Toc372018970)

[1.2.3 Вариант использования «Посмотреть текущую скорость» 11](#_Toc372018971)

[1.2.4 Вариант использования «Посмотреть количество попыток» 11](#_Toc372018972)

[2. Анализ 12](#_Toc372018973)

[2.1 Классы анализа 12](#_Toc372018974)

[2.2 Диаграмма активности 17](#_Toc372018975)

[2.3 Диаграмма компонентов 18](#_Toc372018976)

[2.4 Диаграмма развертывания 19](#_Toc372018977)

[2.5 Диаграммы деятельности 20](#_Toc372018978)

[3. Проектирование 21](#_Toc372018979)

[3.1 Проектные классы 21](#_Toc372018980)

[3.2 Компоненты и программные интерфейсы 21](#_Toc372018981)

[3.3 Конечные автоматы 21](#_Toc372018982)

[4. Реализация 22](#_Toc372018983)

[4.1 Тестирование 22](#_Toc372018984)

[4.1.1 Модульное тестирование 22](#_Toc372018985)

[4.1.2 Интеграционное тестирование 22](#_Toc372018986)

[4.1.3 Системное тестирование 22](#_Toc372018987)

[4.2 Непрерывная интеграция 22](#_Toc372018988)

[4.3 Автоматизация сборки. Конфигурационный файл системы автоматизации сборки с пояснениями 22](#_Toc372018989)

[4.4 Конфигурация системы непрерывной интеграции для автоматического тестирования и сборки проекта 22](#_Toc372018990)

[4.5 Развёртывание 22](#_Toc372018991)

[5. Документация 23](#_Toc372018992)

[5.1 Руководство оператора 23](#_Toc372018993)

[Заключение 24](#_Toc372018994)

[Список использованных источников 25](#_Toc372018995)

# Введение

Существует несколько подходов к разработке программного обеспечения: процедурно-ориентированный, структурно-ориентированный, объектно-ориентированный, функционально-ориентированный и логико-ориентированный.

Объектно-ориентированный подход основан на объектной декомпозиции задачи, абстракции объекта в виде класса, типизации и наследовании объектов.

При выполнении данного курсового проекта будет разработано клиентское веб-приложение выполняющееся в браузере с использованием объектно-ориентированного подхода. Процесс разработки сопровождается составлением документации в виде документа SRS, описания программы, а также различных диаграмм, сопровождающих процесс проектирования программного обеспечения.

## Используемые средства разработки

# Интегрированная среда разработки

Среда разработки является инструментом призванным упрощать разработку программного продукта, за счет интеграции таких инструментов как: текстовый редактор, компилятор, средства оптимизации сборки, система контроля версий и т.д, в одном продукте.

Выбор среды разработки, является критичным как с точки зрения времени выполнения проекта, так и с точки зрения качества проекта.

Грамотный выбор подходящей среды разработки может позволять: уменьшать время кодирования и количество ошибок, за счет:

• Подсветки синтаксиса используемого языка программирования;

• Наглядном представлении файлов проекта;

• Интеграции с системами контроля версий;

• Интеграция с системами тестирования;

• Использовании различных плагинов, например jsLint и т.д.

Для разработки была выбрана интегрированная среда разработки Netbeans. Выбор Netbeans обусловлен:

• Стоимостью, среда бесплатна;

• Большим количеством пользователей, в случае возникновения проблем проще найти решение;

• Поддержкой синтаксиса языка JavaScript;

• Поддержка распределенной системы контроля версий Git.

• Возможность использовать утилиту jsLint в виде плагина, что позволяет использовать данную утилиту совместно со средой разработки.

Вывод: Использование интегрированной среды разработки дало возможность сосредоточиться на процессе кодирования, что вероятно уменьшило количество возможных ошибок и улучшило качество кода.

# Браузер

В силу заложенных требований к кросбраузерности, каркас тестировался на трех основных браузерах:

* Chrome 26.0.1410.64;
* Opera 12.13;
* FireFox 18.0.1.

В первую очередь тестирование проводилось на браузере Chrome из-за наличия встроенного отладчика и профилировщика.

Вывод: выделение трех целевых браузеров позволило оптимизировать каркаса только к наиболее востребованным браузерам.

# Отладчик и профайлер

Отладчик и профайлер и это основные инструменты программиста. С помощью отладчика, иногда его называют дебагером, выполняется поиск семантических ошибок, пожалуй, создание почти любого приложения не происходит без отладчика.

Профайлер это инструмент для выявления количественных характеристик работы приложения, таких время работы отдельных функций или количество используемой памяти. Профайлер используется для поиска так называемых «узких мест» программы, где происходят максимальные потери времени или памяти.

В ходе разработки использовался встроенные отладчик и профайлер браузера Chrome. Это было вызвано тем, что браузер Chrome был одним из целевых браузеров и содержал субъективно наиболее удобные отладчик и профайлер среди встроенных инструментов других браузеров.

Использование отладчика браузера было более предпочтительным, чем использование отладчика среды Netbeans, так как процесс отладки можно было вести полностью в браузере.

Вывод: использование отладчика и профайлера позволило выявлять ошибки и оценивать производительность отдельных участков кода, что позволило повысить производительность каркаса

# Система контроля версий

В проекте была использована распределенная система контроля версий Git и бесплатный сервис для хостинга репозиториев под управлением Git – GitHub [15]. Это позволило обезопасить проект от возможной порчи диска и утраты файлов проекта, упростило возможность переключатся между версиями кода.

Разработка велась через создание ветки для разработки отдельного компонента системы. В случае если разработка компонента признавалась удачно законченной – ветка сливалась с базовой веткой репозиория, если неудачной – совершался окат либо к предыдущим версиям кода в ветке, либо создание новой ветки для разрабоки компонента с учетом предыдущих ошибок.

Выбор распределенной, а не централизованной системы контроля версий позволил не зависеть от наличия доступа в интернет.

Вывод: Использование системы контроля версий уменьшило риск потери исходного кода проекта и упростило процесс кодирования.

# Требования к программе

Предъявляемые требования к программному продукту отражены в документе SRS, документ доступен в приложении А.

## Структурированный перечень функциональных и нефункциональных требований к программе

## Варианты использования

Пользователь может управлять ракеткой, просматривать текущую скорость шарика и количество попыток. На рисунке 1 представлены варианты использования.

• Основное действующее лицо: пользователь;

• Уровень цели: цель пользователя;

• Область действия: приложение.

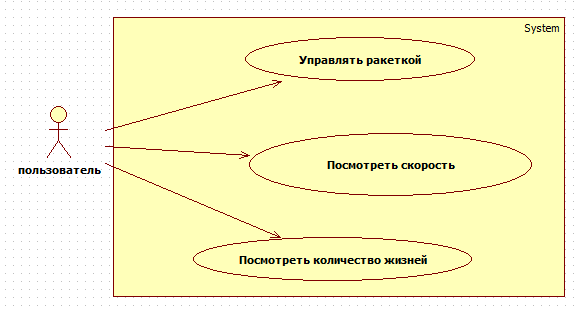


Рис. 1 – Диаграмма вариантов использования

### Вариант использования «Управлять ракеткой с помощью клавиатуры»

Краткое описание: Передвижение ракетки по экрану при нажатии на кнопки «влево» или «вправо».

Предусловие: Игра начата.

Основной поток:

1. Вариант использования начинается, когда пользователь нажимает на клавиатуре клавиши «влево» или «вправо»;
2. Система сдвигает ракетку по экрану влево или вправо;
3. Вариант использования завершается.

Постусловие: Ракетка сдвинута вправо либо влево.

### Вариант использования «Управлять ракеткой с помощью курсора мыши»

Краткое описание: Передвижение ракетки по экрану при перемещении курсора мыши. Ракетка устанавливается согласно X-координаты курсора.

Предусловие: Игра начата.

Основной поток:

1. Пользователь перемещает курсор мыши по экрану;
2. Система сдвигает ракетку по экрану влево или вправо;

Постусловие: Ракетка сдвинута вправо либо влево.

### Вариант использования «Посмотреть текущую скорость»

Краткое описание: Просмотр текущей скорости шарика в условных единицах.

Предусловие: Игра начата.

Основной поток:

1. Пользователь переводит взгляд на табло со скоростью;
2. Пользователь узнает текущую скорость.

Постусловие: Пользователь узнал скорость шарика.

### Вариант использования «Посмотреть количество попыток»

Краткое описание: Просмотр количества оставшихся попыток. Попытка забирается при вылете шарика за нижнюю часть экрана.

Предусловие: Игра начата.

Основной поток:

1. Пользователь перевел взгляд на табло с количеством попыток;
2. Пользователь узнал количество попыток.

Постусловие: Пользователь узнал количество попыток.

# Анализ

## Классы анализа

Целью этапа анализа является создание аналитической модели, которая состоит из классов анализа и прецендентов, которые описывают взаимодействие между экземплярами классов анализа для выполнения определенного варианта использования.

Классы анализа представляют собой абстракции предметной области, не относящиеся к области решения.

На рис. 1 представлены классы анализа для разрабатываемого приложения.

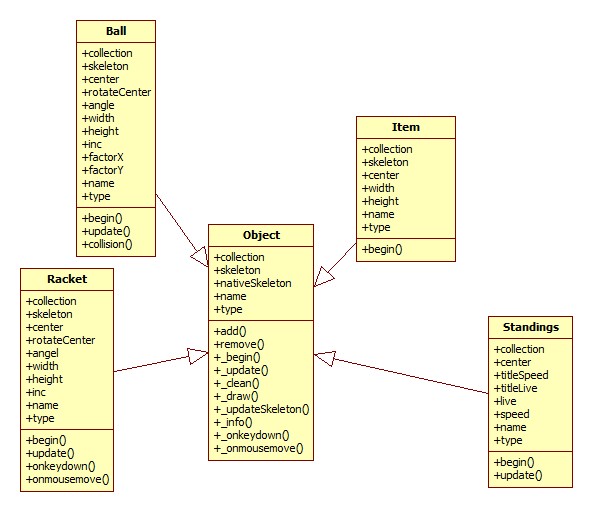


Рис. 2 ­­­– Классы анализа приложения

Абстрактный класс Object предназначен для наследования от него классов Ball, Racket, Item, Standings.

Поля класса Object:

• collection – список вложенных объектов данного класса;

• skeleton – полигон, описывающий очертания объекта;

• nativeSkeleton – полигон, описывающий очертания объекта до изменения положения объекта в начале кадра;

• name – имя объекта;

• type – строка описывающая тип объекта;

Методы класса Object:

• add() – метод для добавления дочернего объекта;

• remove() – метод для удаления дочернего объекта;

• \_begin() – метод для инициализации объекта;

• \_update() – метод для обновления состояния объекта;

• \_clean() – метод для удаления объекта с экрана;

• \_draw() – метод для прорисовки объекта;

• \_updateSkeleton() – метод для обновления полигона описывающего очертания объекта;

• \_info() – метод для вывода отладочной информации;

• \_onkeydown() – метод для обработки события нажатия на кнопку клавиатуры;

• \_onmousemove() – метод для обработки события перемещения курсора мыши;

• collision() – метод для поиска пересечения объектов;

Класс Ball описывает шарик, летящий по экрану

Поля класса Ball:

• collection – поле, содержащее объект, описывающий изображение шарика;

• skeleton – полигон, описывающий очертания шарика;

• center – центр шарика;

• rotateCenter – точка поворота шарика;

• angle – угол поворота шарика;

• width – ширина изображения;

• height – высота изображения;

• inc – скорость шарика;

• factorX – содержит число, указывающее на уменьшение или увеличение координаты X;

• factorY – содержит число, указывающее на уменьшение или увеличение координаты Y;

• name – имя объекта;

• type – строка описывающая тип объекта.

Методы класса Ball:

• begin() – метод для инициализации объекта;

• update() – метод для обновления состояния объекта;

• collision() – метод для обработки события пересечения шарика с другими объектами на экране.

Класс Item описывает фишку.

Поля класса Item:

• collection – поле содержащее объект описывающий изображение льдины;

• skeleton – полигон, описывающий очертания льдины;

• center – центр фишки;

• width – ширина изображения;

• height – высота изображения;

• name – имя объекта;

• type – строка описывающая тип объекта.

Методы класса Item:

• begin() – метод для инициализации объекта.

Класс Racket описывает ракетку для отбивания шарика.

Поля класса Racket:

• collection – поле содержащее объект описывающий изображение ракетки;

• skeleton – полигон, описывающий очертания ракетки;

• center – центр ракетки;

• width – ширина ракетки;

• height – высота ракетки;

• inc – скорость движения ракетки;

• name – имя объекта;

• type – строка описывающая тип объекта.

Методы класса Racket:

• begin() – метод для инициализации объекта;

• update() – метод для обновления состояния объекта;

• onkeydown() – метод для обработки события нажатия на кнопку клавиатуры;

• onmousemove() – метод для обработки события перемещения курсора мыши.

Класс Standings описывает два информационных табло: количество оставшихся жизней и текущая скорость шарика.

Поля класса Standings:

• collection – поле содержащее объекты, описывающие текст на информационных табло;

• center – центр шарика;

• live – количество оставшихся жизней;

• speed – текущая скорость шарика;

• titleSpeed – текст «Скорость шарика: »;

• titleLive – текст «Жизней осталось: »;

• name – имя объекта;

• type – строка описывающая тип объекта.

Методы класса Standings:

• begin() – метод для инициализации объекта;

• update() – метод для обновления состояния объекта;

## Диаграмма активности

На рис. 2 представлена диаграмма активности. Данная диаграмма необходима для детализации основных прецедентов в игре.

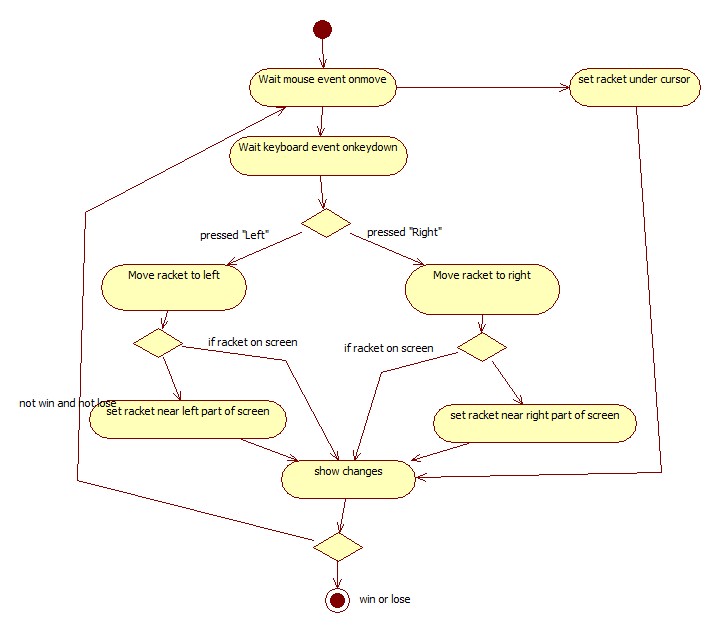


Рис. 3 ­­­– Диаграмма активности приложения

Как видно из диаграммы пользователь может управлять положением ракетки на экране с помощью мыши и с помощью стрелок клавиатуры. Данные действия возможны до наступления состояния проигрыша либо выигрыша.

## Диаграммы последовательности

С помощью диаграммы последовательности были показаны взаимодействия объектов, упорядоченные по времени их проявления (рисунок 4).

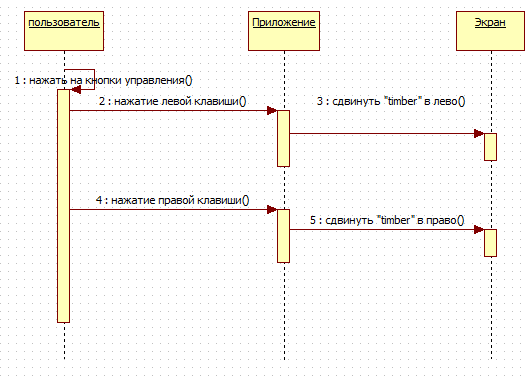


Рис. 4 – Диаграмма последовательности для процесса управления ракеткой

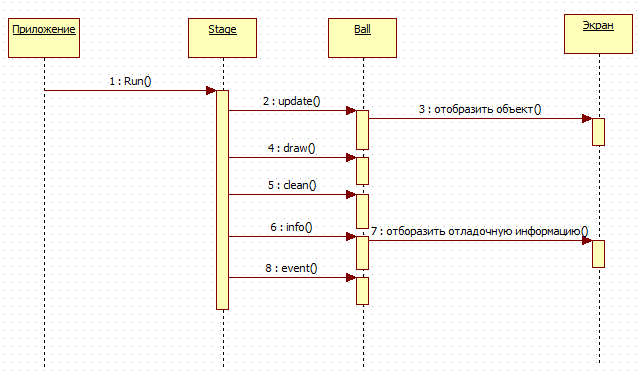
****

Рис. 5 – Диаграмма последовательности для обновления и отображения объекта класса Ball

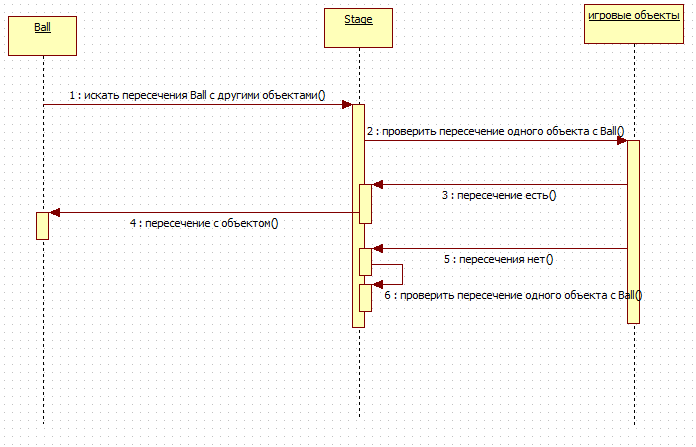


Рис. 6 – Диаграмма последовательности пересечения объекта класса Ball с другими игровыми объектами

# Проектирование

## Проектные классы

## Диаграмма компонентов

На диаграмме компонентов отображаются компоненты программного обеспечения и связи между ними. Диаграмма показана на рисунке 7.

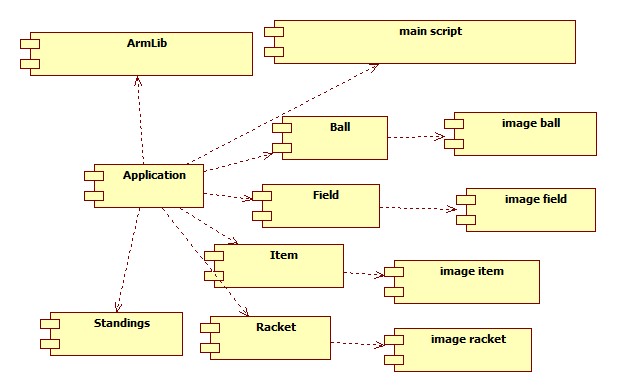


Рис. 7 – Диаграмма компонентов

Приложение разделено на компоненты игры и компонент движка. Класс Ball и описывает шарик, класс Racket описывает ракетку, класс Item описывает фишки, которые нужно сбивать шариком, класс Standings описывает табло попыток и текущей скорости шарика.

## Диаграмма развертывания

Диаграмма развертывания отображает аппаратные элементы компьютера, другие устройства и программные компоненты, а также процессы и объекты, которые им назначены. Диаграмма показана на рисунке 8.

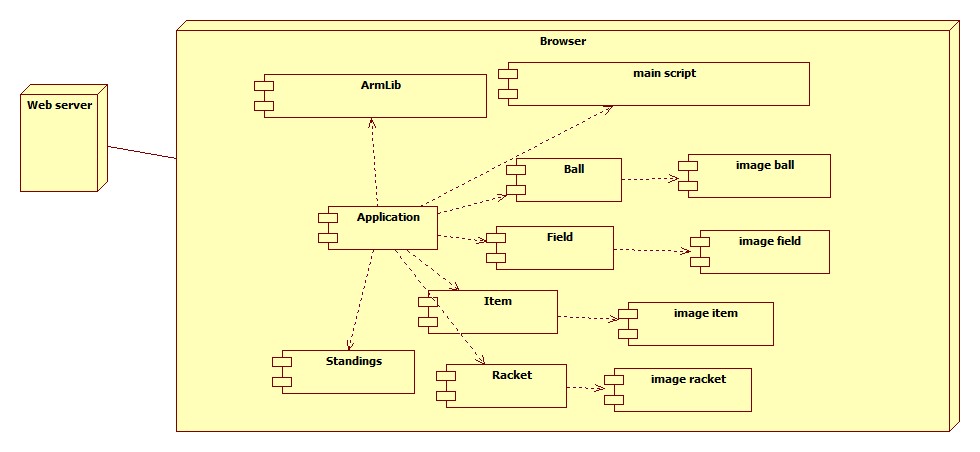


Рис. 8 – Диаграмма развертывания

# Реализация

## Тестирование

### Модульное тестирование

Для тестирования проекта был использован фреймворк QUnit. В процессе написания добовлялись необходимые тесты.

Пример теста:

module( "baseVaribleFunctions" );

test( "gizmo.type", function() {

// String

ok( gizmo.type( "1234ergt#%" ) == "String", " ok( gizmo.type( \"1234ergt#%\" ) == \"String\" Passed!" );

// Number

ok( gizmo.type( -2423.2424 ) == "Number", " ok( gizmo.type( -2423.2424 ) == \"Number\" Passed!" );

// Array

ok( gizmo.type( [3,4,[1,3],{}] ) == "Array", " ok( gizmo.type( [3,4,[1,3],{}] ) == \"Array\" Passed!" );

// Object

ok( gizmo.type( {fwe:22, fg: [46,3]} ) == "Object", " ok( gizmo.type( {fwe:22, fg: [46,3]}} ) == \"Object\" Passed!" );

// Function

ok( gizmo.type( function() {} ) == "Function", " ok( gizmo.type( function() {} ) == \"Function\" Passed!" );

// Date

ok( gizmo.type( new Date() ) == "Date", " ok( gizmo.type( new Date() ) == \"Date\" Passed!" );

// Boolean

ok( gizmo.type( true ) == "Boolean", " ok( gizmo.type( true ) == \"Boolean\" Passed!" );

});

## Непрерывная интеграция

В ходе написания проекта, была реализована возможность автоматической сборки и запуска тестов. Для этих целей использовался инструмент непрерывной интеграции Jenkins, утилита phantomjs, фреймворк QUnit и репозиторий в github.

Сборка минифицированой версии и запуск тестов выполнялись с помощью bash скрипта compile.sh

Содержимое скрипта:

../compress.sh

phantomjs tests/run-qunit.js tests/index.html junit-xml > tests/test.xml

Инструмент непрерывной интеграции Jenkins опрашивал репозиторий в github через каждые пять минут о наличии новых коммитов, в случае их наличия Jenkins выполнял команды git pull, получая новый код, и выполнял скрипт compile.sh в результате чего запускался phantomjs выполняя тесты написанные с помощью QUnit. Результаты тестирования помещались в файл test.xml, после чего обрабатывались Jenkins и выводились в результаты сборки.

## Развёртывание

Для развертывания данного веб-приложения необходим сервер под управлением любой операционной системы. Для запуска работы приложения необходимо поместить код приложения в некоторую директорию веб-сервера, открытую для публичного доступа.

# Документация

## Руководство оператора

# Заключение

В результате проделанной работы было реализовано программное обеспечение являющиеся веб-приложением, а именно игрой Арканоид. Данное приложение не зависит от операционной системы так как выполняется в браузере.

В ходе разработки были изучены материалы по объектно-ориентированному программированию и тестированию, а так же была изучена работа с системой контроля версий Git.

# Список использованных источников

1. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно–ориентированный анализ и проектирование, 2–е издание. – Пер. с англ. – СПб: Символ–Плюс, 2007. – 624 с.