Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра Информационно-аналитических систем

Езус Альбина Николаевна

Разработка рейтинговой системы для платформы CodeFights

Бакалаврская работа

Научный руководитель:
доцент, к.ф.-м.н. Михайлова Е. Г.

Рецензент:
главный исполнительный директор CodeFights
Слоян Тигран

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Department of Analytical Information Systems

Albina Ezus

Development of rating system for CodeFights platform

Bachelor's Thesis

Scientific supervisor:
associate professor, PhD Elena Mikhailova

Reviewer:
CEO CodeFights Sloyan Tigran

Saint-Petersburg
2016

Содержание

Введение

В настоящее время разработано множество платформ, позволяющих программистам изучить самые различные области. Большинство этих платформ делятся на две категории: обучающие платформы, такие как Codecademy и Coursera, и соревновательные, например CodeForces или CodeChef. Первые предлагают онлайн курсы по любым темам, которые могут быть полезны программисту, а вторые - алгоритмические задачи, на решение которых чаще всего отводится определённое время. В последнее время начали набирать популярность игровые платформы, которое предлагают пользователям соревноваться друг с другом в реальном времени через написание программ. Примером таких платформ могут послужить CodinGame и CodeFights.

Помимо программистов, заинтересованных в самообучении, у данных платформ есть и другие крупные пользователи: IT-компании, ищущие специалистов на инженерные должности. Это неудивительно, ведь целью получения образования и изучения новых навыков чаще всего является получение рабочего места, так что компании, сотрудничающие с обучающими платформами, находят не только хороших специалистов, но и людей, заинтересованных в трудоустройстве.

Конечно, у работодателей остаётся заинтересованность в специалистах с высшим образованием и имеющих дипломы и сертификаты, подтверждающие определённый уровень знаний, однако в последние годы в связи с увеличением количества и качества информации, доступной онлайн, становится гораздо труднее распознать хорошего программиста. Ранее компании оценивали кандидатов без опыта работы по диплому и другим сертификатам, и хотя этого, разумеется, было недостаточно, наличие документа показывало, что человек знает хотя бы что-то. Сейчас же большинство программистов являются самоучками, в связи с чем сложно отличить классного специалиста, поскольку на бумаге все кандидаты обычно равны.

Платформа CodeFights предложила новый рекрутинговый метод, в основе которых лежит ”сражение” с ботами, принадлежащими различным компаниям. По окончании сражения, в зависимости от его результатов, программисту может быть предложено отослать свои данные в соответствующую ком-

панию и пройти собеседование. Для того, чтобы обеспечить наилучший отбор кандидатов, необходимо правильно настроить ”уровень” бота. Поскольку боты настраиваются таким образом, чтобы наиболее точно симулировать результат программиста конкретной компании, необходимо правильно оценить уровень каждого программиста компании и в соответствии с этим обучить бота. Разработка рейтинговой системы, позволяющей оценить правильно оценить уровень программиста, а также обучение ботов и предсказание исходов боёв и является темой данной работы.

1. Постановка задачи

Целью данной работы является разработка рейтинговой системы для платформы CodeFights, позволяющей оценить владение навыками программирования, знание алгоритмов, средств разработки и языков программирования. Рейтинговая система должна позволить отвечать на следующие вопросы:

1. Какой уровень знаний у данного программиста?
2. Какой уровень сложности у данной задачи?
3. Решит ли данный программист данную задачу и, если да, с каким результатом?

Помимо этого, система должна быть наглядной для потенциальных работодателей, поэтому должна быть реализована возможность просмотра изменения рейтинга с течением времени, а сама его величина должна быть легкой для понимания.

На сайте CodeFights необходимо реализовать возможность просмотра текущего рейтинга пользователя, динамики его изменений и рейтинговой таблицы пользователей.

2. Обзор предметной области

Платформа CodeFights предлагает несколько режимов игры, в каждом из которых пользователю предлагается решить некоторое количество алгоритмических задач за отведённое время. Решением задачи является функция с заданным именем, реализованная на одном из доступных языков программирования. Параметры, которые эта функция применяет, заданы в различных тестах. Каждый тест представляет из себя набор входных данных (аргументов функции) и выходное значение, которое корректно написанная функция должна возвращать для данных входных значений. Задача считается решённой если проходит все тесты, в том числе скрытые.

Все задания можно поделить на три типа:

1. BugFix: дано условие задача и её решение на одном из доступных языков, в одной из строчек которого есть ошибка. Необходимо найти её и исправить.

2. Recofery: дано условие задачи её решение на одном из доступных языков, часть которого упущена. Необходимо восстановить пропущенный кусок кода.

3. CodeWriting: дано условие задачи. Необходимо написать с нуля её решение.

Возможный следующие режимы игры:

1. Бой один на один: двум пользователям предоставлен одинаковый набор из трёх заданий каждого типа. Каждое задание оценивается в зависимости от её типа, времени, за которое она была решена, и количества незачтённых решений. Побеждает пользователь с наибольшим количеством очков. Противники на такой бой выбираются случайно.

2. Дружеский бой: аналогично бою один на один, но пользователь сам решает, против кого сражаться (”вызывает” другого пользователя на бой).

3. Тренировка против бота: пользователь, начавший такой бой, сражается не против другого программиста, а против бота. Результат бота зависит от его уровня, а также от сложности задачи.

4. Бой против бота какой-либо компании: то же, что и тренировка против бота, однако типы заданий могут отличаться. Основной особенностью такого режима является то, что результат бота зависит от результатов программистов компании, которым было предложено решить тот же набор задач.

5. Турнир: турнир на выбывание, в котором для прохода на следующий круг необходимо победить в бою один на один.

6. Марафон: в течении часа необходимо решить набор из шести представленных задач. Этот режим игры не отличается от стандартных соревнований олимпиадного программирования.

7. Code Golf: особый режим, в котором предлагается решить задачу с использованием как можно меньшего количества символов в решении. Задача находится в открытом доступе и её может пытаться решить любой пользователь платформы.

2.1. Изначальная рейтинговая система

Существующая изначально рейтинговая система CodeFights основана на системе Glicko[4], использующейся для оценки уровня игроков шахмат или го. Эта рейтинговая система присваивает две величины - рейтинг и отклонение - задачам, пользователям и ботам.

Рейтинги пересчитываются по следующим формулам:

$$q = \frac{\ln 10}{400}$$

$$g = \frac{1}{\sqrt{1 + 3 * (\frac{q * d_2}{\pi})^2}}$$

$$E = \frac{1}{1 + 10^{\frac{g*(r_2 - r_1)}{400}}}$$

$$d^2 = \frac{1}{q^2 * g^2 * E * (1 - E)}$$

$$scaling = \begin{cases} 1 - E & \text{если } scoreDiff > 0 \\ E & \text{иначе} \end{cases}$$

$$newRD = max\left(150, \frac{1}{\sqrt{\frac{1}{d_1^2} + \frac{1}{d^2}}}\right)$$

$$newRating = r_1 + q * newRD^2 * g * scaling * scoreDiff$$

, где d_1 и r_1 - отклонение и рейтинг первого пользователя, и d_2 и r_2 - отклонение и рейтинг или второго пользователя, или задачи, у которой необходимо пересчитать рейтинг. Величина $scoreDiff$ является дополнительным коэффициентом, описание которого прилагается ниже.

2.1.1. Рейтинг пользователей

Рейтинг пользователей меняется после каждого боя один на один с помощью чуть модифицированной системы Glicko, в которой также учитывается разница очков в качестве дополнительного коэффициента. Этот дополнительный коэффициент зависит от разницы очков и количества раундов в матче.

2.1.2. Рейтинг задач

Каждой задаче присваивается некоторый изначальный рейтинг, зависящий от её сложности, установленной авторами. После каждого боя, в котором эта задача встречается, её рейтинг пересчитывается в зависимости от того, была ли решена задача, и количества времени, затраченного на его решение.

2.1.3. Рейтинг ботов

Рейтинг бота является предустановленным числом и не меняется (кроме случаев, когда это делается специально в настройках). Этот рейтинг используется для того, чтобы как можно точно моделировать поведение пользователей с похожим уровнем. Ответы боя зависят от его рейтинга и рейтинга задач, на основе которых определяется решит ли бот задачу и, если да, то за какое время.

2.1.4. Минусы изначальной системы

Хотя изначальная система и позволяет примерно определить уровень программиста и ранжирует задачи по сложности, существующие минусы вынудили от неё отказаться.

Во-первых, рейтинг пользователя меняется в зависимости от рейтинга противников в боях. Очевидно, что такой подход в корне неверен, поскольку уровень знаний программиста нельзя оценить по уровню другого программиста, особенно если речь идёт о разных областях.

Во-вторых, довольно странным решением является изменение сложности задачи. Задача может стать проще или сложнее лишь для конкретного пользователя, но менять её сложность в связи с этим является довольно странным решением.

В-третьих, данная система не учитывает, что задачи могут повторяться. Если одному пользователю уже случалось решать поставленную задачу, а другому нет, неразумно ставить их в равное положение и оценивать их результаты одинаково.

Обозначенные выше проблемы сделали дальнейшее использование системы нецелесообразной, в связи с чем было принято решение о разработке новой системы, не имеющей подобных недостатков.

2.2. Существующие решения

2.2.1. Сайты олимпиадного программирования

Большинство сайтов, посвящённых олимпиадному программированию, ранжируют пользователей по их уровню. Платформы TopCoder и CodeForces используют Glick-подобные системы рейтингов, которые обладают теми же проблемами, что и описанная выше изначальная система рейтингов CodeFigts. Их проблема заключаются в том, что они не способны адекватно отражать уровень знаний пользователя, поскольку зависят от выступлений других пользователей, результаты которых меняются с течением времени.

2.2.2. Кредитный скоринг

Наиболее интересным примером рейтинга для поставленной задачи является кредитный скоринг, использующийся для оценки кредитоспособности людей в США. Его суть заключается в следующем:

1. Скоринг является приватным, и доступ к нему может быть предоставлен или самим человеком для заинтересованной стороны (например для получения ипотеки), или подтверждённым кредитором для верификации данных;

2. Кредитный скоринг не является конкретным числом, а абстрактной величиной, периодически вычисляемой соответствующими агенствами. Для того, чтобы его увидеть, необходимо запросить формальное разрешение.

3. *Мягкий запрос* это запрос человека на просмотр своего кредитного скоринга. Стоимость подобного отчёта немаленькая, однако некоторые службы обязаны составлять бесплатные отчёты раз в неделю.

4. *Жёсткий запрос* подаётся кредитором (например банком) для проверки кредитной истории.

На основе этой оценки кредиторы решают о платёжеспособности заёмщика и выносят решения о предоставлении кредита. Эта система интересна

для рассмотрения, поскольку рейтинговая система, которую необходимо разработать, должна предоставлять работодателям информацию о кандидате в сотрудники, то есть показывать, насколько программист хорош как потенциальный работник.

Рис. 1: Уровни оценки FICO.

3. Разработанная система

3.1. Общая модель

Разработанная система представляет из себя одно число, которое обозначает некоторый общий уровень, и дополнительные значения, показывающие, насколько программист разбирается в конкретной области. Этими областями являются:

1. **Бэк энд:** знания алгоритмов, на данный момент общая оценка основывается только на ней;
2. **Фронт энд:** знания HTML, CSS и JavaScript. Эти задачи разрабатываются в данный момент и должны в ближайшем времени выйти в продакшен;
3. **Обеспечение качества и тестирование:** оценка знаний методов тестирования систем;
4. **Базы данных:** теоретические знания реляционных и No-SQL систем, а также языков запросов к базам данных;
5. **Мобильная разработка:** умение разрабатывать приложения для мобильных платформ;
   (a) Android;
   (b) iOS;
6. **Наука о данных:** знание статистических методов и методов машинного обучения;
7. **Языки программирования:** уровень владения конкретным языком программирования. Предполагается использовать методы статического анализа кода, чтобы определить уровень владения стандартными библиотеками и функциями.

3.2. Новая рейтинговая модель

Как и кредитный скоринг, новая рейтинговая система присваивает каждому пользователю число в промежутке от 300 до 850. Однако это сделано лишь для удобства понимания: на деле рейтингом является число от 0 до 1, которое потом масштабируется на большие числа. Этот рейтинг присваивается как пользователям, так и задачам, однако в отличие от пользовательского рейтинга, рейтинг задач не меняется с течением времени. Тем не менее, его можно изменить вручную в случае отклонений (если рейтинг пользователей, встретивших эту задачу, слишком сильно падает / возрастает).

Пересчитывание рейтингов не происходит постоянно, а раз в неделю, когда система наименее нагружена. Изначально происходит пересчитывание уровня задач и масштабирование получившихся результатов, а затем - пересчитывание рейтинга пользователей и их масштабирование.

Рейтинг пользователя пересчитывается, если он сталкивается с задачей, которую до этого ещё не решал. Эта задача может принадлежать любому типу, Если у этой задачи ещё нет рейтинга, пересчет рейтинга откладывается до того момента, как рейтинг задачи станет известным.

3.2.1. Рейтинг задачи

Для того, чтобы понять уровень пользователя, необходимо понять уровень задач, которые он решает. Уровень каждой задачи зависит от времени, которое обычно уходит на её решение, и процента верных решений, причём все интересующие величины должны быть найдены для случаев, в которых задача встречалась впервые. Таким образом, рейтинг задачи, которую в лучшем случае решили за $bestTime$ секунд и которая встречалась пользователям $taskSeen$ раз и была решена $taskSolved$ раз, рассчитывается по следующей формуле:

$$taskRating = bestTime * \frac{taskSolved}{taskSeen}$$

.

3.2.2. Рейтинг пользователя

Пересчитывание рейтинга пользователя осуществляется по чуть более сложность системе. Пусть каждое из событий E_1, E_2, ..., E_n является встречей пользователя с задачей, которую он видит впервые. Для каждого события E_i рассчитаем его оценку следующим образом:

- если задача не была решена, оценка соответствующего события будет равна 0;
- в противном случае пусть $bestTime$ - среднее лучшее время решения задачи события E_i из десяти лучших времён, $solvedTime$ - время, потребовавшееся данной пользователю, а $taskRating$ - рейтинг задачи из события E_i. Тогда:

$$timeRatio = min\left(1, \frac{bestTime}{solvedTime}\right)$$
$$E_i = taskRating * \left(\frac{1}{2} + \frac{1}{2} * \sqrt{timeRatio}\right)$$

После чего все события сортируются в порядке уменьшения их оценки, и рейтинг пользователя рассчитывается по следующей формуле:

$$\sum_{i=n}^{1}\left(\frac{1}{2^i} * E_i\right)$$

Таким образом, рейтинг пользователя зависит от того, насколько хорошо он справляется с задачами, которые видит в первый раз. Наибольший вес присваивается задачам с наибольшей оценкой, поэтому есть вероятность, что пользователю просто пару раз повезло со сложными задачами. Для того, чтобы точнее определять рейтинг, было принято решение воспользоваться количеством оценок n.

Пусть рейтинг $rating$ был посчитан на n оценках. Тогда нижняя и верхняя границы рейтинга рассчитываются по следующим формулам:

$$minRating = \frac{rating * n}{n + 1}$$
$$maxRating = \frac{rating * n + 1}{n + 1}$$

Понятно, что при достаточно больших n эти оценки совпадут:

$$\lim_{n \to \infty} minRating = \lim_{n \to \infty} maxRating = rating$$

Полученные таким образом оценки приводятся к более привычному виду по следующей формуле:

$$\lfloor 300 + 1100 * \frac{rating^{0.6}}{rating + 1} + 0.5 \rfloor$$

Округление вниз и слагаемое 0.5 здесь нужно для того, чтобы применить обычное округление.

3.2.3. Поведение ботов

Изменения в рассчитывании рейтинга ботов компаний не произошло, однако изменился пересчет ответа ботов, которые используются в режиме ”тренировка”.

Пусть рейтинг бота равен $botRating$, рейтинг задачи равен $taskRating$, а лучшее время её решения - $bestTime$. Будем рассматривать три случая:

1. $botRating \geq taskRating$: задача решена с оптимально возможным временем;

2. $botRating * 2 \leq taskRating$: бот не в состоянии решить задачу и пропускает её;

3. $taskRating * 2 > botRating * 2 > taskRating$: бот решает задачу. Для того, чтобы рассчитать время, требующееся для её решения, воспользуемся формулами, обратными тем, которые использовались в подсчете

рейтинга для пользователей.

$$timeRatio = min\left(1, \left(\frac{2 * botRating}{taskRating} - 1\right)^2\right)$$

$$res = \frac{bestTime}{timeRatio}$$

Таким образом, ответ бота напрямую зависит от его рейтинга и позволяет достаточно точно моделировать результат пользователя с определённым рейтингом, который впервые столкнулся с данной задачей.

4. Интегрирование системы

Интегрирование новой системы происходит постепенно, поскольку рейтинг пользователей основывается на рейтинге задач, который в свою очередь зависит от собранной статистики. В целом весь процесс делится на два этапа, описание которых представлено ниже.

4.1. Пересчёт статистики задач

Первым этапом является сбор статистики по пользователям. Для каждого пользователя находятся все задачи и челленджи, которые он решал, а также его результаты в момент первой встречи с задачей. Поскольку в базе данных отдельно хранятся бои, результаты решения задач в турнирах и результаты посылок решений в челленджи, сбор статистики в каждой из этих групп являлся отдельной задачей.

1. **Бои:** Для задач, встречающихся в боях, находится количество верных и неверных решений, а так же время, за которое эта задача решалась с первого раза;

2. **Турниры:** Поскольку в турнирах невозможно определить время, затраченное на решение задачи (предоставленный пользователю набор задач он может решать в любом порядке), отсюда можно собрать лишь информацию о том, была ли решена задача или нет;

3. **Челленджи:** Для челленджей хранится лишь информация о том, была ли решена задача или нет. Также учитывается время решения: если оно произошло после завершения задачи, решение игнорируется, поскольку к этому времени оно могло быть найдено в интернете.

Данные всех задач собираются таким образом, что учитывается лишь те случаи, в которых пользователь увидел задачу впервые. Поскольку перебор всей базы данных занимает достаточно много времени, все запросы выполняются асинхронно.

4.2. Пересчёт рейтингов

После того, как вся статистика собрана, выполняется пересчёт рейтинга задач, как это указано в 3.2.

Поскольку все данные, собранные на первом этапе, остаются в базе, нет необходимости производить их пересчёт в дальнейшем. Теперь каждый раз, когда пользователь начинает решать некоторую задачу, происходит её поиск в данных на этого пользователя, и если задача решается впервые, результат этого решения заносится в базу. Пересчёт рейтингов, тем не менее, необходимо производить заново, что и происходит раз в неделю.

5. Используемые технологии

CodeFights развёрнут на платформе Meteor, которая позволяет разрабатывать веб и мобильные приложения используя JavaScript и Node.js. Базой данных, поставляемой вместе с платформой, является MongoDB, нативная поддержка которого позволяет использовать его реактивность по максимуму.

Для того, чтобы не отвлекаться на проектирование базы данных, используется платформа compose.io. Для анализа результатов был загружен бэкап системы, который был проанализирован с помощью iPython notebook и Python 3.

6. Результаты

В рамках выполнения работы были достигнуты следующие результаты:

1. Разработана рейтинговая система, позволяющая оценивать уровень пользователей и задач, а также и моделировать поведение ботов определенного уровня в боях один на один.

2. Новая рейтинговая система была интегрирована в платформу CodeFights, и теперь является ее основной системой. Реализован алгоритм регулярного пересчета рейтинга по событиям, произошедшим за последнюю неделю.

3. Для 70% задач подсчитан новый рейтинг, и реализована возможность просмотра рейтинга и статистики по задачам[1].

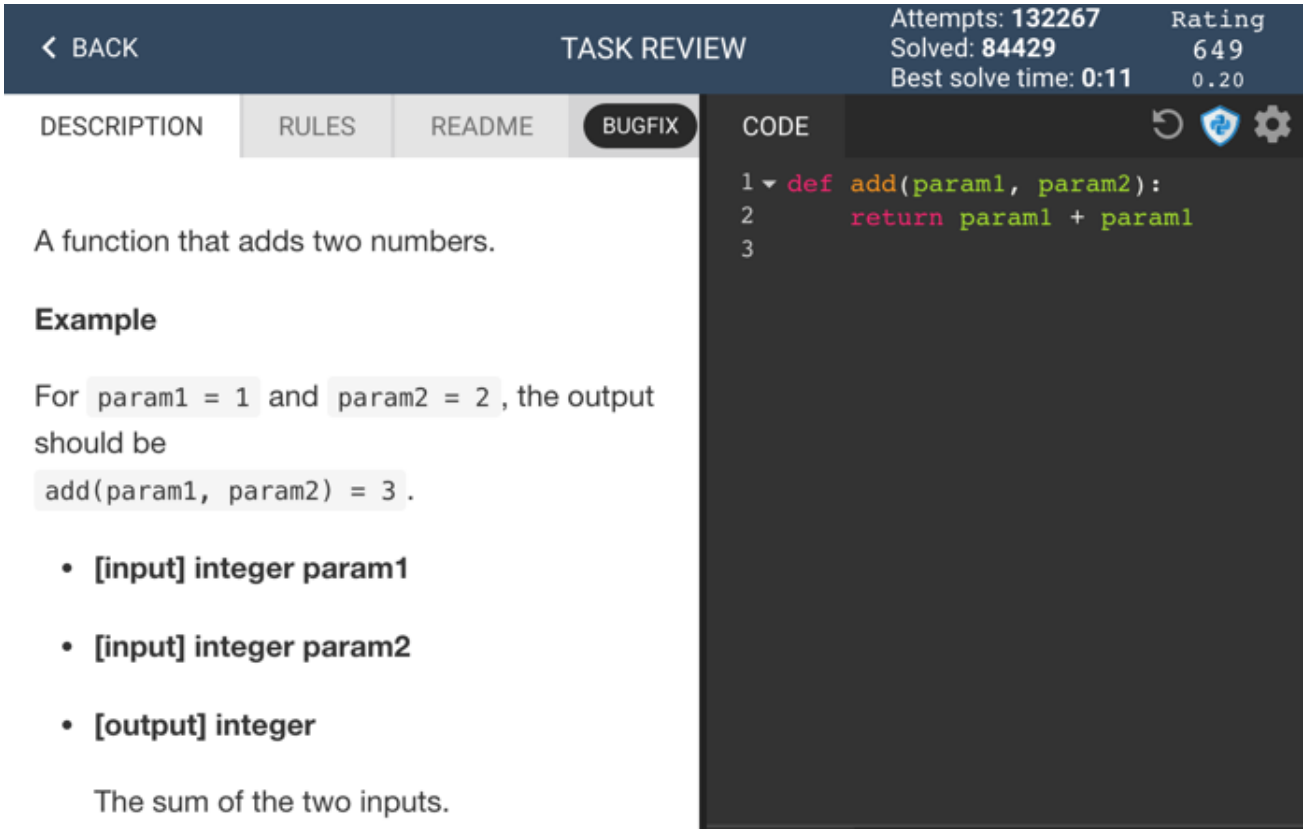

Рис. 2: Страница задачи с приведением статистики.

[1]Просмотр личный рейтингов и рейтингов задач доступен только пользователям платформы CodeFights с правами администратора

4. У 30% пользователей рейтинг рассчитан как минимум по 10 событиям. Реализована возможность просмотра рейтинга пользователя, а также динамики его изменений и просмотра решений задач, по которым рейтинг пересчитывался.

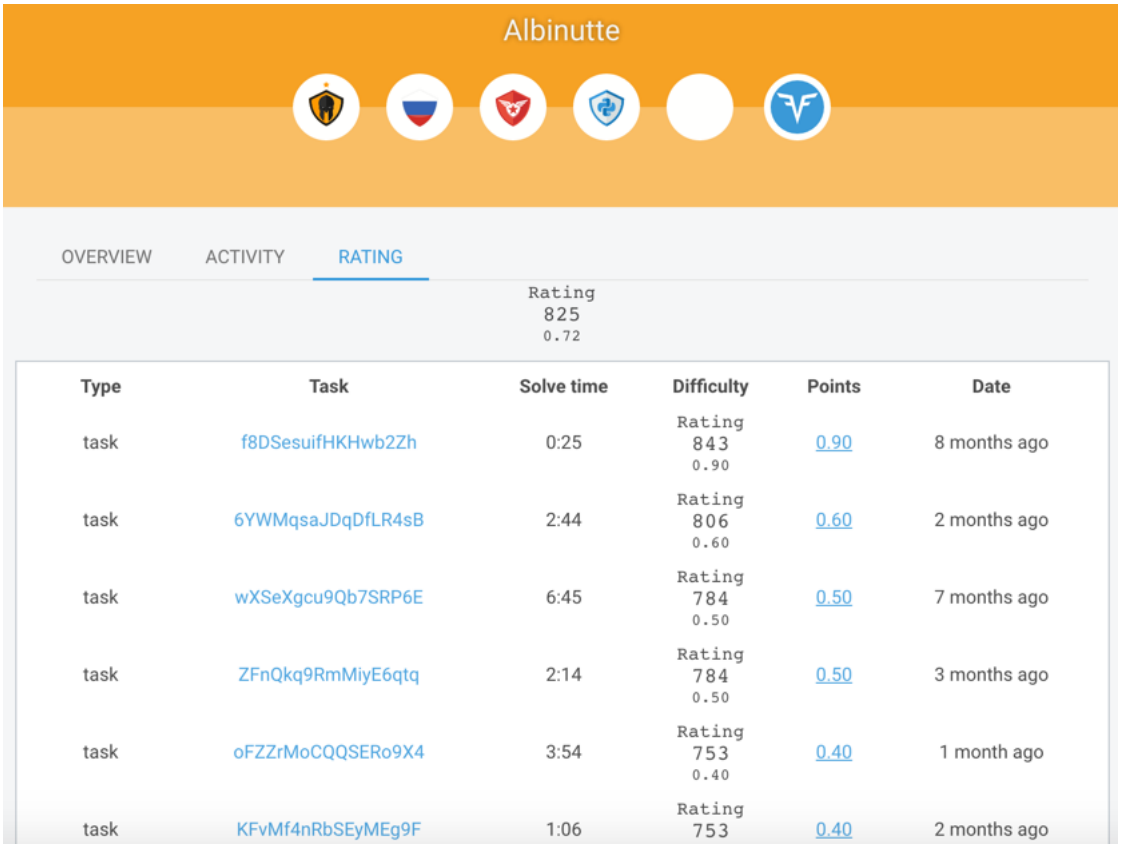

Рис. 3: Рейтинг пользователя.

5. Осуществлено внедрение нового алгоритма поведения ботов, позволяющее моделировать результаты пользователя определенного уровня, а также реализована возможность устанавливать уровень сложности для бота.

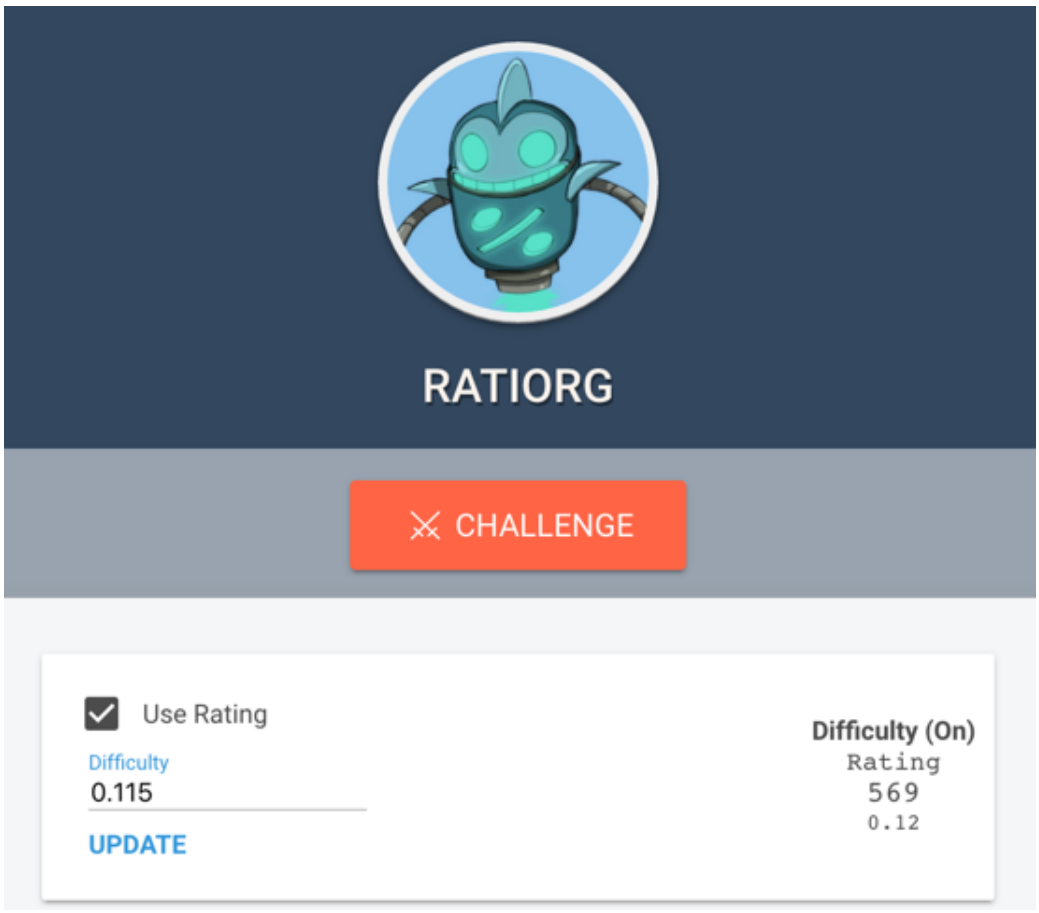

Рис. 4: Страница бота (для пользователя с правами администратора).

6. Реализована возможность просмотра кандидатов с наивысшим рейтингом в соответствии с различными фильтрами. Данная страница доступна работодателям и позволяет отслеживать этап собеседования, на котором находится кандидат.

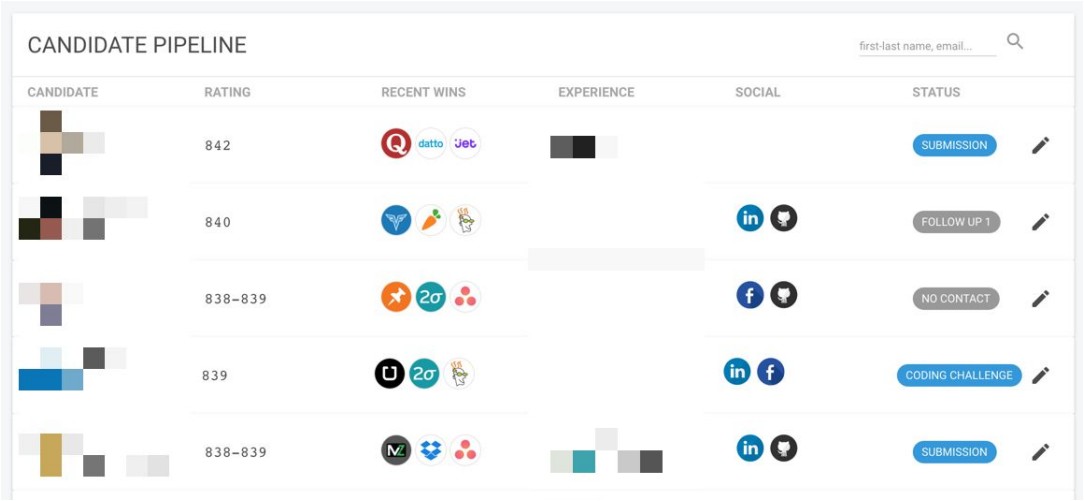

Рис. 5: Список кандидатов, отсортированных по рейтингу.

Нововведённая рейтинговая система используется для сортировки пользователей, победивших ботов. Лучшим из них было предложено пройти собеседование, и многие из них получили должности в крупных компаниях. Процент кандидатов, которым поступило предложение о трудоустройстве после собеседования, увеличился в 10 раз [3], что говорит о том, что кандидаты с наивысшим рейтингом являются востребованными специалистами.

Сбор данных о пользователе, включающий в себя запись боев, промежуточные решения и пройденные тесты, позволит в будущем улучшить систему, используя сохраненную информацию.

7. Заключение

В дипломной работе была поставлена задача разработки рейтинговой системы для платформы CodeFights, позволяющей оценивать уровень программиста и сложность предоставляемых задач, а также моделировать результаты пользователя с определенным рейтингом.

Разработанная система доказала свою эффективность и уже является лучше предыдущей. Представленная система введена в эксплуатацию и позволяет оценивать алгоритмические знания программиста. Для того, чтобы ее возможности можно было использовать полноценно, требуется создание достаточного количества заданий в более широких областях и, вероятно, модификации их оценки.

Рейтинг задачи оценивается по результатам пользователей, которые её решают. Разработанная система хранит как промежуточные решения каждой задачи, так и версии, проходящие все тесты, что дает возможность анализировать их исходные коды, и, соответственно, иначе оценивать сложность задачи.

Каждой задаче присвоена определённая категория, порой не одна. Чаще всего по категории можно получить примерную оценку сложности задачи, что следует учесть при дальнейшем развитии предложенной рейтинговой системы.

Таким образом, проделанная работа является основой на пути к созданию глобальной рейтинговой системы, которая позволит оценивать уровень программистов и упростить жизнь как работодателям, так и программистам, находящимся в поиске рабочего места.

Список литературы

[1] Andrew Alan Armstrong, C. P. Automated analysis of code developer's profile. Patent, 2012.

[2] Chrysler, E. Some basic determinants of computer programming productivity. *Communications of the ACM 21* (June 1978), 472–483.

[3] Dishman, L. You might apply for your next job by playing a mobile game, 2016.

[4] Glickman, P. M. E. The glicko system. Tech. rep., Boston University, 1999.

[5] Michael Lydon, J. M. H. Method and system for communicating programmer information to potential employers. Patent, 2006.

[6] Raymond M. Berger, R. C. W. The development of programmer evaluation measures. *SIGCPR '65 Proceedings of the third annual computer personnel research conference* (1965), 6 – 17.

[7] Stolba, L. Professional rating system and method. Patent, 2006.

[8] Tennant, D. How competing against a 'bot' could land you a developer job, 2016.