Preliminary Analysis of Contestant Performance for a Code Hunt Contest

Adrian Clark
Jonathan Wells
Angello Astorga
University of Illinois at Urbana-Champaign
{ajclark3,jjwells2,aastorg2}@illinois.edu

Andrew Xie
Independent
xie.yandi@gmail.com

Jalen Coleman-Lands
Tao Xie
University of Illinois at Urbana-Champaign
{clmnlnd2,taoxie}@illinois.edu

Abstract

Platforms of programming contests are increasingly adopted to incentivize students' interests in programming and train their programming and problem-solving skills. Code Hunt (https://www.codehunt.com/) is one such popular platform from Microsoft Research, being adopted in various contests worldwide. For a contest, Code Hunt hosts a sequence of programming puzzles provided by the contest organizers and provides interactive feedback to the contestants to assist them in solving the puzzles. Analyzing platform-collected data for a Code Hunt contest can provide valuable insights on understanding both the contestants and the puzzles in the contest, in order to improve the design of future contests and training of the contestants. In this paper, we present preliminary analysis of contestant performance among all contestants along with comparing contestant performance between contestants using mostly Java and contestants using mostly C#. Such analysis is conducted on a Code Hunt data set (released to the public) that contains the programs written by students worldwide during a contest over 48 hours. The contest was attended by 259 contestants to attempt to solve 24 puzzles. The contest finally included about 13,000 programs submitted by these contestants. The analysis results expose a number of interesting and useful observations for future research.

CCS Concepts • **Social and professional topics → Software engineering education**;

Keywords Code Hunt, Educational Software Engineering

1 Introduction

Platforms of programming contests are increasingly adopted to incentivize students' interests in programming and train their programming and problem-solving skills. Among various such platforms, Code Hunt [1] (https://www.codehunt.com/), released by Microsoft Research, is a web-based platform for online programming education through gaming. Solving a puzzle in Code Hunt can teach a student algorithms and give the student a coding mindset in a mystery format: the functional requirements for the coding task, in the form of the secret solution, are not known to the student. Instead, the student gets clues from a table (produced by Code Hunt) containing sample test inputs (generated by the underlying automatic test generation engine named Pex [3, 5]) and their outputs for the secret solution and the student's working solution (named as the student solution), respectively. Based on the clues, the student iteratively tries to improve the student solution to match the functional requirements (i.e., the functional behaviors of the secret solution). There is no limit on the number of iterations/attempts that the student can make to solve a puzzle. The student successfully solves the puzzle with the student solution if it matches the functional behaviors of the secret solution. A student can solve puzzles in either Java or C#, as their programming language of choice. For a contest, Code Hunt hosts a sequence of programming puzzles provided by the contest organizers. Code Hunt can be also leveraged for teaching or training purposes [4, 6–9].

In 2015, Microsoft Research has released to the public a data set collected from a student-only worldwide contest that took place over 48 hours, called the Imagine Cup 2014. There were 259 contestants, who attempted to solve 24 puzzles (i.e., levels) organized in 4 sectors (6 puzzles per sector) based on a learning topic and increasing level of difficulty. Approximately 13,000 programs (i.e., student solutions) were submitted by these contestants during the contest. After a contestant successfully solves a puzzle, the Code Hunt game engine assigns a score (1, 2, or 3) to the contestant's solution code. The score reflects how elegant the student solution is, measured by how succinct the solution is in terms of the number of instructions in the compiled .NET intermediate language. In particular, score 1 indicates that the solution is much longer than all other so far submitted student solutions, while 2 is about average, and 3 is significantly shorter. Each puzzle requires a different algorithm of a certain type, and puzzles with the same type are organized into the same sector. Sectors and puzzles are organized by difficulty. Code Hunt imposes a rule that a contestant can access a puzzle in a sector only if the contestant has won at least all but one

PLATEAU'17 Workshop on Evaluation and Usability of Programming Languages and Tools, October 23, 2017, Vancouver, CA

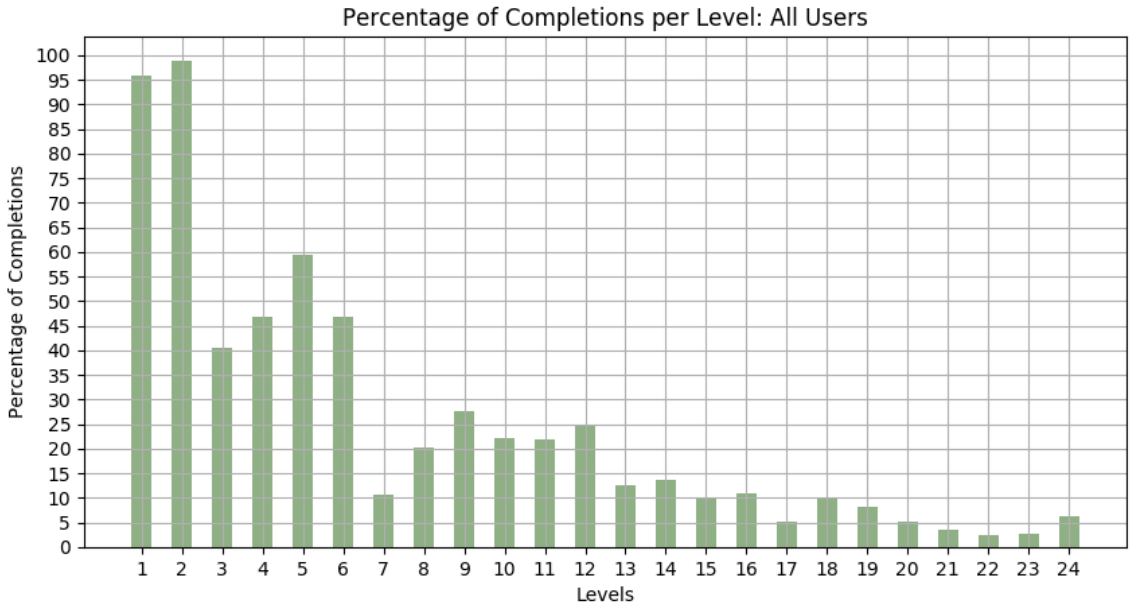

Figure 1. Percentage of contestants who complete a puzzle (i.e., level) (among all contestants).

puzzle of the previous sector, and therefore contest/puzzle creators tend to arrange puzzles in order of increasing difficulty.

Analyzing the data set released for the Code Hunt contest can provide valuable insights on understanding both the contestants and the puzzles in the contest, in order to improve the design of future contests and training of the contestants. In this paper, we present preliminary analysis of contestant performance among all contestants along with comparing contestant performance between contestants using mostly Java (named as Java contestants) and contestants using mostly C# (named as C# contestants). The analysis results expose a number of interesting and useful observations for future research.

The analysis in this paper complements an analysis conducted in previous work [2] on the same data set in two main ways. First, the analysis in this paper investigates the scores (1, 2, or 3) obtained by contestants besides puzzle completion as focused by the previous work. Second, besides studying the performance of all contestants as focused by the previous work, the analysis in this paper additionally compares the performance between Java contestants and C# contestants.

The rest of the paper is organized as follows. Section 2 presents the contest performance for all the 259 contestants. Then Section 3 presents the performance comparison of Java contestants and C# contestants. Finally, Section 4 concludes the paper.

2 Performance Analysis of All Contestants

The difficulty levels of puzzle solving should be incrementally increased such that later puzzles in the sequence are more challenging to solve than earlier ones in the contest. For our analysis, we measure the difficulty encountered by the contestants for a puzzle, i.e., level, (along with the contestants' performance), based on (1) the percentage of contestants who complete (i.e., successfully solve) the puzzle among all the contestants, and (2) the percentage of contestants who obtain a specific score (1, 2, or 3) for the puzzle among all the contestants (if a contestant could not complete a puzzle, the contestant obtains score 0). Note that there can be cases where some contestants may give up attempting a puzzle because the puzzle's being too easy causes these contestants to lose interest, or they get bored and move on to other activities. However, such cases might not be very common among the contestants within such serious worldwide contest over 48 hours.

To help understand the general trend of difficulty in the contest, Figure 1 shows the percentage of contestants who complete a particular puzzle among all the contestants. Figure 2 shows the percentage of contestants who receive scores of 1, 2, and 3 for a particular puzzle, denoted by three bars, respectively. For both figures, the x axis shows the 24 puzzles (i.e., levels).

Figure 1 shows that a trend of increasing difficulty can be observed across sectors and puzzles, indeed with some fluctuations (e.g., at Puzzles 3, 4, 7, 8, 17, 24). The percentage of contestants who complete puzzles from increasing sectors

(1-4) gradually decreases. In particular, for Sector 1 (i.e., Puzzles 1-6), the percentage of contestants who complete puzzles range from 98% to 41%. For Sector 2 (i.e., Puzzles 7-12), the percentage of contestants who complete puzzles ranges from 27% to 11%. For Sector 3 (Puzzles 13-18) and Sector 4 (Puzzles 19-24), the percentage of contestants who complete puzzles ranges from 14% to 6%, and 8% to 3%, respectively.

However, compared to their nearby puzzles, Puzzles 3, 7, and 17 are relatively more difficult, as also discussed in previous work [2].

Puzzle 3's secret solution is shown below:

Puzzle 3's secret solution

```
public static bool Puzzle(bool x, bool y, bool z) {
    return x | y & z;
}
```

Puzzle 3 is more difficult to solve than nearby puzzles primarily because its solution is based on using bit-wise operations, whose functional behaviors tend to be a bit challenging for contestants to infer based on the table of test inputs and outputs.

Puzzle 7's simplified secret solution (with the same functional behaviors as the original secret solution) is shown below:

Puzzle 7's simplified secret solution

```
public static int Puzzle(int[] a) {
 int sum = 0;
 foreach (var n in a) {
    sum += n;
  }
 int len = a.Length;
 return (sum + len/2) / len;
 }
```

Only about 10% of the contestants could complete this puzzle. The main reason is that computing the proper rounded average from the list tends to be challenging for contestants to infer based on table of test inputs and outputs.

The secret solution in Puzzle 17 is to emulate the movement of a knight piece in a chess game, as shown below:

Puzzle 17's secret solution

```
public static class Program {
 static int[] deltaX = {-2, -2, -1, -1,  1,  1,  2,  2};
 static int[] deltaY = {-1,  1, -2,  2, -2,  2, -1,  1};

 public static int[][] Puzzle(int x, int y) {
      PexAssume.IsTrue(1<=x & x<=8 & 1<=y & y<=8);
      if (x==5&y==1 | x==3&y==8);  // Hint to user
      int[] moves = new int[8];
      int moveIndex = 0;
      for( int i=0; i<8; i++ ) {
          int xx = x+deltaX[i];
          if (xx < 1 | xx > 8) continue;
          int yy = y+deltaY[i];
          if (yy < 1 | yy > 8) continue;
          moves[moveIndex++] = (xx << 8) | yy;
      }
      int[][] result = new int[moveIndex][];
      for(int i=0; i<moveIndex; i++) {
          int val = moves[i];
          result[i] = new int[]  val>>8, val&0xFF ;
      }
      return result;
  }
}
```

Only about 5% of the contestants could complete this puzzle. The difficulty primarily comes from the relatively complex logic in the secret solution.

Figure 2 shows a similar trend of increasing difficulty. In general, contestants who complete a puzzle tend to obtain score 3. Note that after a contestant completes a puzzle, the contestant has a chance to optimize his/her student solution by reducing its length while preserving the functional behaviors (e.g., removing those extra conditional statements removing which would not change the functional behaviors of the student solution). However, we have two additional interesting observations. For Puzzles 19-24 (the most difficult group of puzzles) along with Puzzle 13, the contestants who complete these puzzles overwhelmingly obtain score 3. For Puzzles 1-3 and 5 (among the easiest puzzles), similarly, the contestants who complete these puzzles overwhelmingly obtain score 3.

Among the 24 puzzles, for Puzzles 6, 8, and 17, the percentage of contestants obtaining score 1 is similar to the percentage of contestants obtaining score 3. Such situation is in big contrast to the other puzzles, for which the percentage of contestants obtaining score 3 is dominating.

The secret solution in Puzzle 6 is about counting words in the given string s with the following two lines as its key statements:

Puzzle 6's partial secret solution

```
string[] list = s.Split((char[])null,
                        StringSplitOptions.RemoveEmptyEntries);
return list.Length;
```

Therefore, about half of the contestants who complete the puzzle implement such functionality in many lines of code without realizing that they could leverage the string API method `Split`.

The secret solution in Puzzle 8 is to count the depth of nesting parentheses in the given string s. Interestingly the percentage of contestants obtaining score 2 is higher than the percentage of contestants obtaining score 1 or 3. This puzzle is the only one with such situation. The secret solution includes the following lines as its key statements:

Puzzle 8's partial secret solution

```
int openClose = 0;
int maxDepth = 0;
foreach (char c in s) {
  if ( c == '(' ) {
      openClose++;
      if ( openClose > maxDepth )
         maxDepth = openClose;
  }
  else
  if ( c == ')' ) {
      openClose--;
      if ( openClose < 0 ) return 0;
  }
}
return (openClose == 0) ? maxDepth : 0;
```

A non-trivial percentage of contestants who complete the puzzle fail to write the last return statement in a succinct way as adopted in the secret solution, leading to the observation for Puzzle 8.

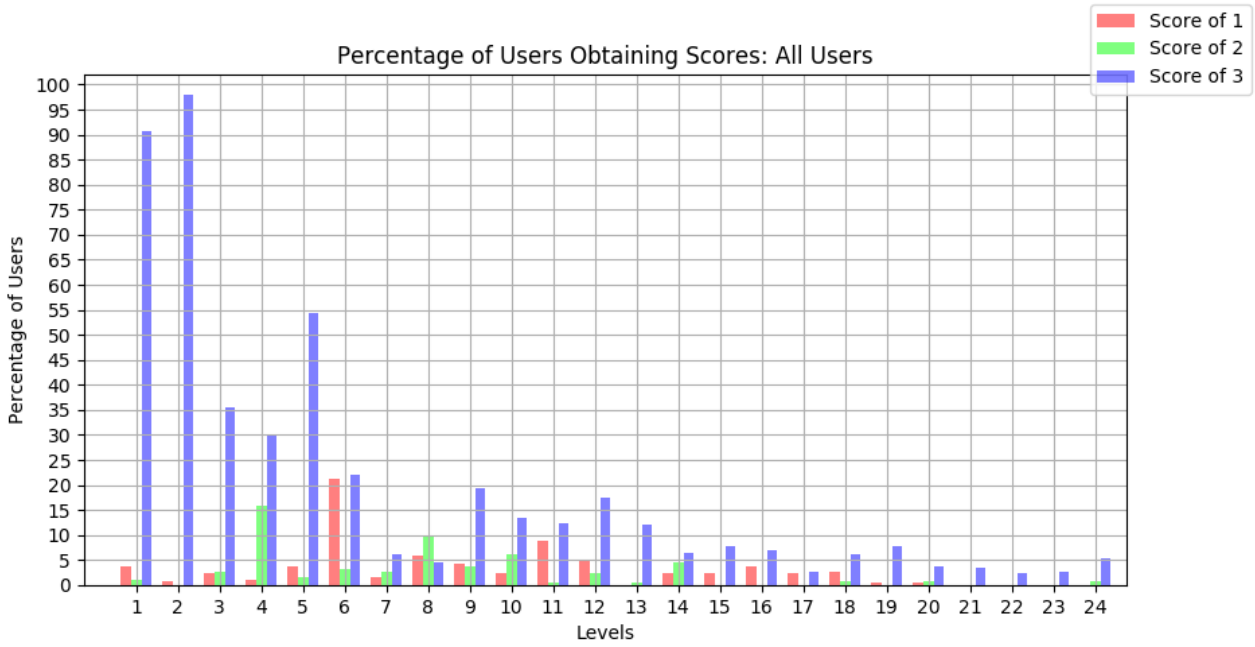

Figure 2. Percentage of scores 1, 2, and 3 achieved by all contestants

About half of the contestants who complete Puzzle 17 implement such functionality of emulating the movement of a knight piece in many lines of code, without being able to leverage the use of bit-wise operations to solve the puzzle, as used in the secret solution shown earlier.

3 Performance Analysis of Java Contestants vs. C# Contestants

In this section, we compare the performance of those contestants who attempt more than half of the attempted puzzles using Java (named as Java contestants) and those contestants who attempt more than half of the puzzles using C# (named as C# contestants).

By analyzing the data set, we find out that among the 259 contestants, there are **143 C# contestants**, **113 Java contestants**, and 3 contestants who use both Java and C# to attempt the same number of puzzles, respectively. In fact, the 3 contestants are not the only ones to switch languages while attempting the puzzles; 21 contestants switch the used language at some point during the contest including 11 Java contestants and 7 C# contestants. In the analysis presented in the rest of this section, we focus on comparing the performance of the 143 C# contestants and 113 Java contestants. In other words, the numbers of Java contestants and C# contestants are at a similar scale but Java contestants are about 20% more than C# contestants. Such result is not surprising given that in general Java is more popularly adopted than C#[1].

[1] http://statisticstimes.com/tech/top-computer-languages.php

Figure 3 shows the percentage of C# contestants who complete a particular puzzle among all the C# contestants (indicated by the left purple bar) and the percentage of Java contestants who complete a particular puzzle among all the Java contestants (indicated by the right yellow bar). In the figure, the x axis shows the 24 puzzles (i.e., levels). As is shown in Figure 3, generally the Java contestants perform better than the C# contestants across puzzles. The only two exceptions are Puzzles 3 and 17, for which C# contestants perform better than Java contestants; the difference is especially obvious for Puzzle 3. Recall that Puzzles 3 and 17 fall into the observed fluctuations: those puzzles that are more difficult to solve than their nearby puzzles. In future work, we plan to compare the details of the student solutions for these two puzzles by C# contestants and Java contestants, in order to dig out primary reasons for such outlier cases.

Figures 4 and 5 show the percentage of C# contestants and Java contestants, respectively, who receive scores of 1, 2, and 3 for a particular puzzle, denoted by three bars, respectively. For both figures, the x axis shows the 24 puzzles (i.e., levels). Generally, the percentage of C# contestants obtaining score 3 is lower than the percentage of Java contestants obtaining score 3, indicating that C# contestants perform worse than Java contestants with respect to the scores. Such result is consistent with that from Figure 3.

In addition, there are some interesting differences across Figures 4 and 5. For Puzzle 8, the percentage of C# contestants obtaining score 2 is highly dominating whereas the percentage of Java contestants obtaining score 1 is higher than both that of Java contestants obtaining score 2 or 3.

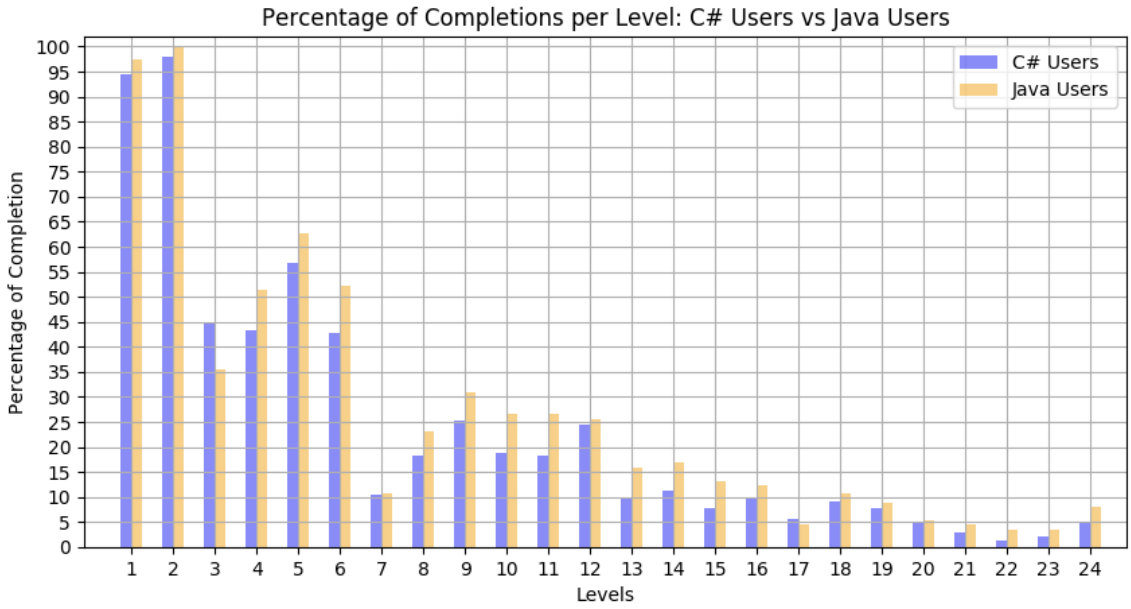

Figure 3. Percentage of successful attempts (completions) per puzzle (i.e., level) by C# contestants and Java contestants.

For Puzzle 11, the percentage of Java contestants obtaining score 1 is higher than that of Java contestants obtaining score 3, and both are much higher than that of Java contestants obtaining score 2.

One possible reason to explain that Java contestants perform better than C# contestants could be that Java, being a very popular language, is typically taught very early on in students' careers, and thus Java contestants may be more experienced programmers. In future work, we plan to investigate the correlation between the self-declared experience levels of contestants and their language of choice in conducting the contest.

4 Conclusion

In this paper, we have presented preliminary analysis of contestant performance among all contestants along with comparing contestant performance between Java contestants and C# contestants. As expected, the puzzles in the contest are generally having increasing difficulty, with a few exceptional cases. Generally, the contestants obtain score 3, especially for the easiest puzzles (the first group in the sequence) and the most difficult puzzles (the last group in the sequence). Java contestants tend to outperform C# contestants. These analysis results expose a number of interesting and useful observations for future research.

Acknowledgments

This work is supported in part by National Science Foundation under grants no. CCF-1409423, CNS-1434582, CNS-1513939, CNS-1564274.

References

[1] Judith Bishop, R. Nigel Horspool, Tao Xie, Nikolai Tillmann, and Jonathan de Halleux. 2015. Code Hunt: Experience with Coding Contests at Scale. In *Proc. International Conference on Software Engineering (ICSE 2015), JSEET*. 398–407.

[2] Pierre McCauley, Brandon Nsiah-Ababio, Joshua Reed, Faramola Isiaka, and Tao Xie. 2016. Preliminary Analysis of Code Hunt Data Set from a Contest. In *Proc. International Code Hunt Workshop on Educational Software Engineering (CHESE 2016)*. 7–8.

[3] Nikolai Tillmann and Jonathan de Halleux. 2008. Pex-White Box Test Generation for .NET. In *Proc. International Conference on Tests and Proofs (TAP 2008)*. 134–153.

[4] Nikolai Tillmann, Jonathan de Halleux, Judith Bishop, Tao Xie, Nigel Horspool, and Daniel Perelman. 2014. Code Hunt: Context-Driven Interactive Gaming for Learning Programming and Software Engineering. In *Proc. International Workshop on Context in Software Development (CSD 2014)*.

[5] Nikolai Tillmann, Jonathan de Halleux, and Tao Xie. 2014. Transferring an Automated Test Generation Tool to Practice: From Pex to Fakes and Code Digger. In *Proc. ACM/IEEE International Conference on Automated Software Engineering (ASE 2014)*. 385–396.

[6] Nikolai Tillmann, Jonathan de Halleux, Tao Xie, and Judith Bishop. 2014. Code Hunt: Gamifying Teaching and Learning of Computer Science at Scale. In *Proc. ACM Conference on Learning @ Scale Conference (L@S 2014)*. 221–222.

[7] Nikolai Tillmann, Jonathan De Halleux, Tao Xie, Sumit Gulwani, and Judith Bishop. 2013. Teaching and Learning Programming and Software Engineering via Interactive Gaming. In *Proc. International Conference on Software Engineering (ICSE 2013), SEE*. 1117–1126.

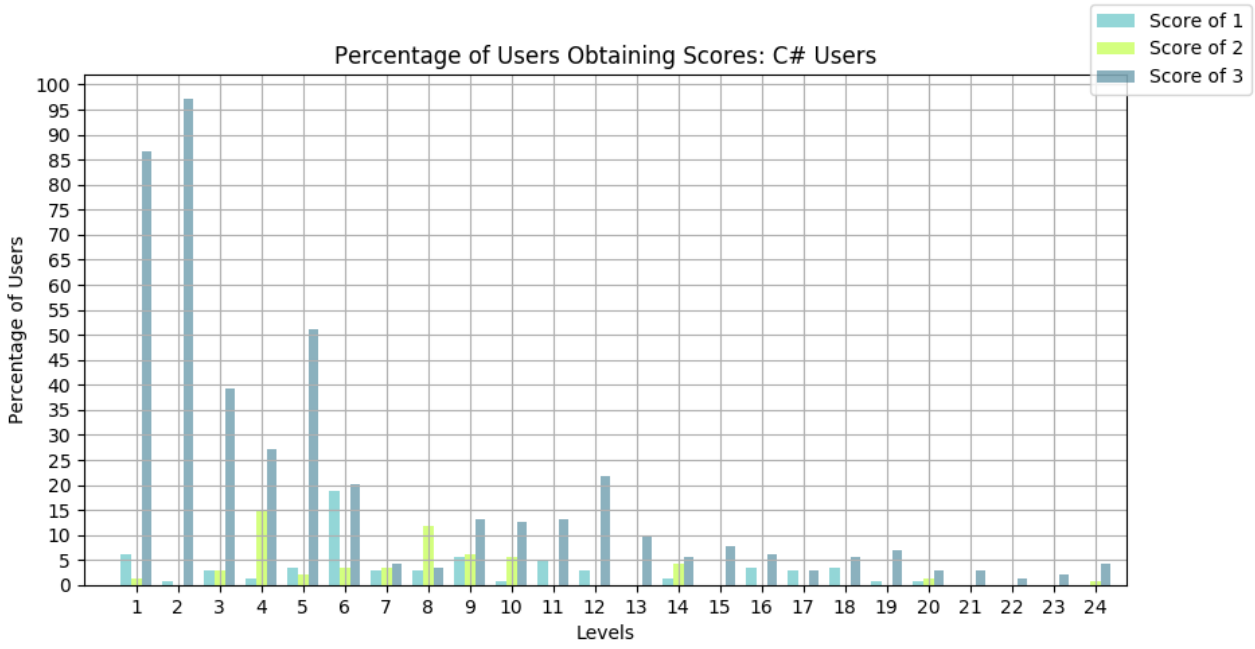

Figure 4. Percentage of scores achieved by C# contestants

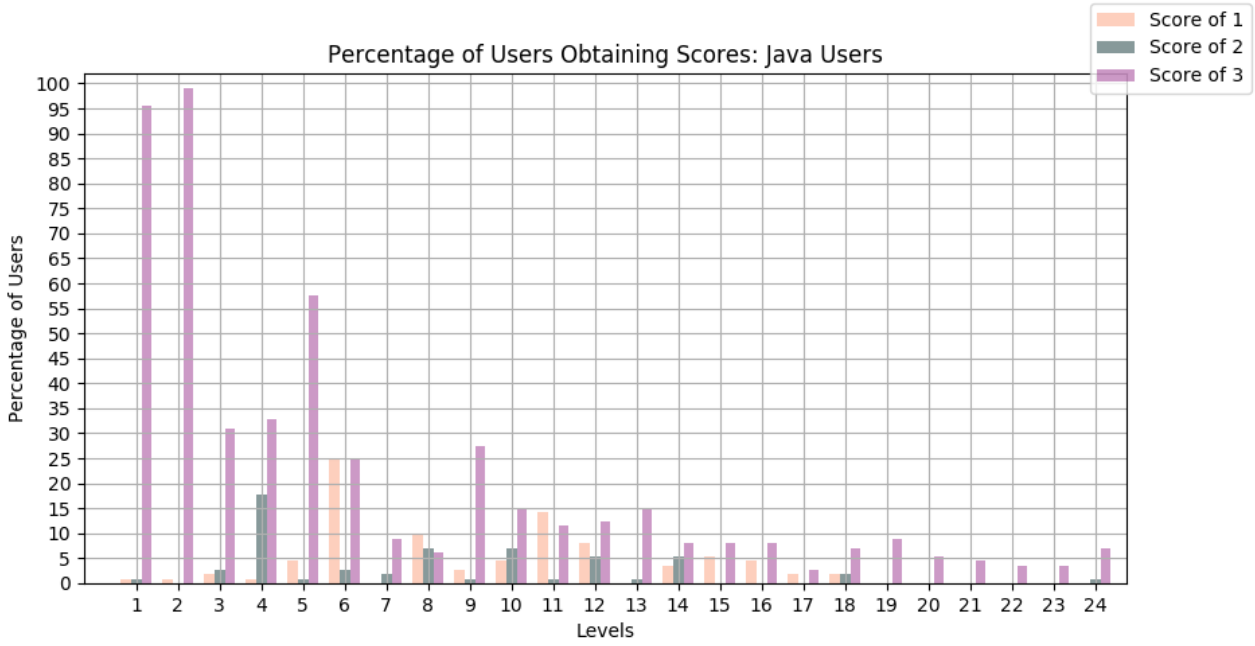

Figure 5. Percentage of scores achieved by Java contestants

[8] Tao Xie, Judith Bishop, R. Nigel Horspool, Nikolai Tillmann, and Jonathan de Halleux. 2015. Crowdsourcing Code and Process via Code Hunt. In *Proc. IEEE/ACM International Workshop on CrowdSourcing in Software Engineering (CSI-SE 2015)*. 15–16.

[9] Tao Xie, Judith Bishop, Nikolai Tillmann, and Jonathan de Halleux. 2015. Gamifying Software Security Education and Training via Secure Coding Duels in Code Hunt. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security (HotSoS 2015)*. 26:1–26:2.