Code Hunt: Searching for Secret Code for Fun

Nikolai Tillmann
Judith Bishop
Microsoft Research
Redmond, WA 98052, USA
nikolait,jbishop@microsoft.com

R. Nigel Horspool
University of Victoria
Dept. of Computer Science
Victoria BC V8W 2Y2, Canada
nigelh@cs.uvic.ca

Daniel Perelman
Univ. Washington
Dept. of Computer Science
Seattle WA 98195, USA
perelman@cs.washington.edu

Tao Xie
University of Illinois at Urbana-Champaign
Dept. Computer Science
Urbana, IL, USA
taoxie@illinois.edu

ABSTRACT

Learning to code can be made more effective and sustainable if it is perceived as fun by the learner. Code Hunt uses puzzles that players have to explore by means of clues presented as test cases. Players iteratively modify their code to match the functional behaviour of secret solutions. This way of learning to code is very different to learning from a specification. It is essentially re-engineering from test cases. Code Hunt is based on the test/clue generation of Pex, a white-box test generation tool that uses dynamic symbolic execution. Pex performs a guided search to determine feasible execution paths. Conceptually, solving a puzzle is the manual process of conducting search-based test generation: the "test data" to be generated by the player is the player's code, and the "fitness values" that reflect the closeness of the player's code to the secret code are the clues (i.e., Pex-generated test cases). This paper is the first one to describe Code Hunt and its extensions over its precursor Pex4Fun. Code Hunt represents a high-impact educational gaming platform that not only internally leverages fitness values to guide test/clue generation but also externally offers fun user experiences where search-based test generation is manually emulated. Because the amount of data is growing all the time, the entire system runs in the cloud on Windows Azure.

Categories and Subject Descriptors D.2.5 Testing and Debugging, D.3.4 Parsing, K.8.0 Games

General Terms Algorithms, Experimentation, Human Factors, Languages, Verification.

Keywords Games for learning, white box testing, symbolic execution, data-mining, hint mechanisms, source translation, Pex.

1. INTRODUCTION

1.1 Background – Pex4Fun

Code evaluator systems are very popular, with the growth in student numbers and the popularity of MOOCs. These systems work on the basis of a problem specification and a set of test cases to establish if the student has achieved an acceptable program.

SBST'14, June 2 – June 3, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2852-4/14/06...$15.00
http://dx.doi.org/10.1145/2593833.2593838

Several years ago, we released Pex4Fun www.pex4fun.com which did the opposite: presenting an empty slate to the user and a set of constantly changing test cases [4]. To solve a puzzle in Pex4Fun, the player iteratively modifies code to match the functional behavior of a secret solution. The player's code modification is guided by a set of test cases. These are automatically generated by a white-box testing tool called Pex [3] to show under what sample inputs the player's code and secret code have the same outputs and have different outputs, respectively. To compare two programs, a new meta-program is generated that invokes both programs. The meta-program checks if given the same inputs, both programs produce the same result.

As a state-of-the-art implementation of dynamic symbolic execution [1], Pex conducts a search through the universe of feasible execution paths of the meta-program. This search is guided by fairness heuristics involving different code coverage criteria, and by fitness functions [5] to prefer branches that are most promising to eventually lead to previously uncovered code. Pex uses a constraint solver to determine if any potential path is feasible, and to compute test inputs that satisfy the path condition

Checking a puzzle can be viewed as the manual process of conducting search-based test generation: the "test data" to be generated by the player is the player's code, and the "fitness values" that reflect the closeness of the player's code to the secret code are the clues (i.e., Pex-generated test cases). When solving a puzzle, the player attempts to modify the code to improve two "fitness values": reducing the number of failing test cases while increasing the number of passing test cases. The fun and learning effects are especially augmented when the "fitness values" are only partially displayed to the player: only sample (not all) test data generated by Pex are displayed as clues to the player. Thus, a player cannot solve a puzzle by simply attempting to over-fit the partially displayed "fitness values" by hardcoding for input-output pairs.

Although Pex4Fun was, and is, very popular, we wanted to extend its capabilities as a game and investigate how far we could retrofit the data that is mined to provide hints to the player. We also wanted to bring the game to a larger audience with more languages. Thus Code Hunt was born (Figure 1).

Code Hunt differs from Pex4Fun in several ways: It uses a version of the Pex4Fun backend that we adapted to run in Windows Azure where it automatically scales to support an arbitrary number of users. The Code Hunt website provides an experience that is exclusively focused around the search-based game play idea, while the core experience of Pex4Fun is to showcase the capabilities of the Pex engine. Code Hunt supports Java in addition to C#. We are also in the process of deploying into Code Hunt a system that will provide hints when the system detects that a player is stuck.

1.2 Testing and Learning

Learning to code by solving a puzzle is not the same as learning to code by writing to a specification. There are many competitions that exist where students pit their wits against each other – and against the clock – to create a solution to a defined problem. This kind of coding is similar to what they encounter in courses or later in their careers. Code Hunt is different in that learning to code is a by-product of solving a problem which is presented as pattern matching inputs and outputs. The fun is in finding the pattern.

Fun is seen as a vital ingredient in accelerating learning and retaining interest in what might be a long and sometimes boring journey towards obtaining a necessary skill. In the context of coding, there have been attempts to introduce fun by means of storytelling [2], animation (www.scratch.mit.edu) and robots (e.g. www.play-i.com). Code Hunt adds another dimension – that of puzzles.

Figure 1. The opening screen of Code Hunt

Code Hunt represents a high-impact educational gaming platform that not only internally leverages fitness values to guide test/clue generation but also externally offers fun user experiences where search-based test generation is manually emulated. In this paper we describe how Code Hunt works and the steps we have taken to maximize the fun aspects of the system. We go into some detail on the use of different programming languages (C# and Java) and our experience in designing puzzles. We explain how Code Hunt is really one of a kind, and present some results based on an early version of the system. Code Hunt is now freely available to test in its new skin at www.codehunt.com.

2. OVERVIEW OF CODE HUNT

Code Hunt is a serious game where player write code to advance. Code Hunt runs in any modern browser at codehunt.com; see Figure 1 for the splash screen. The built-in tutorial reveals the following story:

> Greetings, program! You are an experimental application known as a CODE HUNTER. You, along with other code hunters, have been sent into a top-secret computer system to find, restore, and capture as many code fragments as possible. Your progress, along with your fellow code hunters, will be tracked. Good luck.

The game is structured into a series of sectors, which in turn contain a series of levels. In each level, the players write code that implements a particular formula or algorithm. In what follows, a level equates to a puzzle.

As the code develops, the game engine gives custom progress feedback to the player. It's part of the gameplay that the player learns more about the nature of the goal algorithm from the progress feedback. Figure 2 shows the feedback loop between the player's code in the browser and cloud-based game engine.

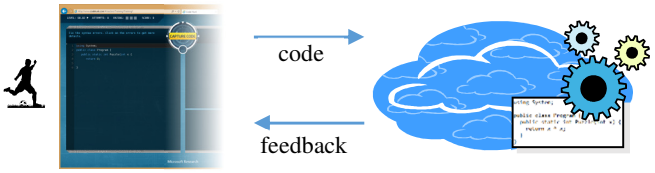

Figure 2. Gameplay

2.1 Sectors and Levels

As in any game, there are sectors and levels. The player can write code in an editor window, using either C# or Java as the programming language. This code must implement a particular formula or algorithm – represented by a top-level function called `Puzzle`. The function has some input parameters, and it returns a result. The player has only one way to test if the current code implements the goal algorithm: by pressing on a big "CAPTURE CODE" button. Pressing this button causes a chain of events:

1. The code is sent to a server in the cloud.
2. The server compiles the code (including an optional Java-to-C# conversion, as explained in Section 3)
3. The server starts an in-depth analysis of the code, comparing it to the goal algorithm.
4. The results are returned and shown to the player.

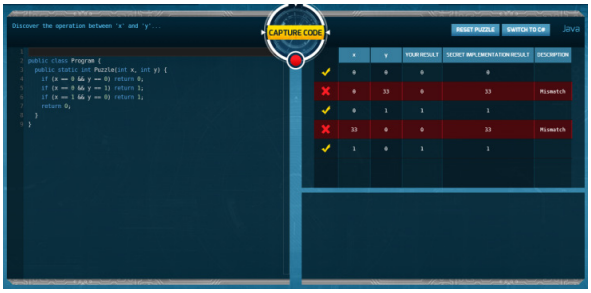

Figure 3 The Code Hunt main page, showing test results

The result will be either a compilation error, with information in the bottom pane, or some mismatches or agreements with the goal algorithm. Figure 3 shows the code on the left, and the mismatches (red crosses) and agreements (yellow checkmarks) are shown on the right. If the code compiles and there are no mismatches and only agreements with the goal algorithm, the player wins this level – or as the game puts it, the player "CAPTURED!" the code, as shown in Figure 4.

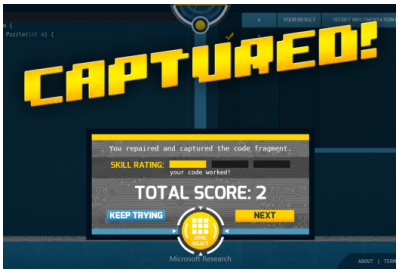

Figure 4. After completing a puzzle, the player gets a score

The in-depth analysis returns each mismatch and agreement with the goal algorithm in the form of a tuple (input, actual result, expected result).While the actual and expected result are the same when the player's code is in agreement with the goal algorithm,

they are different when there is a mismatch. The player then inspects the mismatches and determines how to change the code to make it more like the goal algorithm.

Some levels come with an English description of the goal, e.g. "Can you sum the factorials between x and y?" In other levels, it's a guessing game: the player has to infer the goal algorithm from the mismatches and agreements shown after pressing the "CAPTURE CODE" button; in other words, the player's task is to reverse-engineer some algorithm, and write semantically equivalent code.

In Code Hunt's basic world, the sectors are ordered by topics such as "arithmetic", "loops", "conditionals", "strings", "cyphers", and so on. The sectors and the levels within them are ordered by the difficulty of the topic area. Besides reflecting a natural progression in difficulty of the required programming constructs to solve the sectors and levels, we also used data from the Pex4Fun platform [4] to guide the ordering. There are some 1.4 million user submissions that give us an objective assessment of the difficulty of each problem.

After each level, the player is directed to immediately attempt a slightly more difficult level, thereby maintaining flow. Furthermore, the sectors except for the first one are initially locked. Players have to complete enough levels in a given sector in order to unlock the next sector. Figure 5 shows the list of sectors, most of which are still locked.

Figure 5. The game's sectors, unlocked as the user progresses

2.2 Skill Ratings and Score

When the player successfully completes a level, the Code Hunt game engine assigns a "skill rating" to the player's code. The rating is an integer 1, 2, or 3, and reflects the elegance of the solution, measured by its succinctness (a count of instructions in the compiled .NET intermediate language). 1 indicates that the solution is much longer than all other submitted solutions, 2 means about average, and 3 means significantly shorter.

The intention behind the skill rating is that it may motivate players to keep tinkering in a particular level in order to improve their code, thus greatly extending the gameplay time. This rating is multiplied by a level-specific value that reflects the difficulty of the level, resulting in the "score" for this level. Figure 4 shows the rating 1, and a score of 2 (implying a multiplier of 2 for this level), after the player completed a level.

Players can track their progress via an accumulated total score and the top 15 total players are displayed on a constantly changing leaderboard.

3. ARCHITECTURE

Code Hunt is a true cloud-based system hosted in Windows Azure. As shown in Figure 6, the player requests the page www.codehunt.com which is served from a front-end cloud app. If the player chooses to log in, Windows Azure Active Directory Access Control delegates authorization to one of the identity providers (Microsoft, Facebook, Google, Yahoo). Once the player engages in any particular level, a back-end cloud app is invoked at api.codehunt.com. The back-end exposes a publicly accessible REST-based service API which performs the actual program analysis tasks, and also persists user-specific data in Windows Azure Store. Guarded by an OAuth v2 authorization scheme, the back-end is available for use by other clients. We welcome other researchers who are interested in using this platform for other research topics.

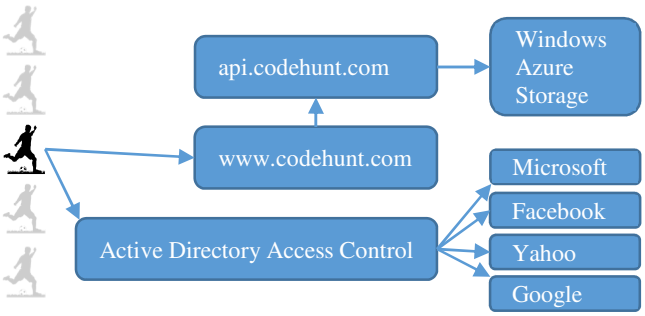

Figure 6. Architecture

Both the front-end and the back-end have been designed for maximum scalability, dynamically increasing the number of cores available to serve an arbitrary number of users.

To illustrate the need for scalability, consider that each concurrent user of Code Hunt who presses the "CAPTURE CODE" button as part of the gameplay potentially causes a single core of the back-end to be busy analyzing the submitted code for up to 30 seconds. Many cores are necessary to support the users at peak times (entire classrooms), while very few cores may be needed at other times.

4. SUPPORT FOR JAVA CODE

The Code Hunt website supports Java code by translating it into C# source code. If sufficient programming resources had been available, the ideal solution might have been to use a Java compiler that has been retargeted to output .NET intermediate code instead of Java bytecode. However that would still leave us with a major challenge – references to classes in the Java API.

Conversion from Java to C# at the source code level is the easier direction because C# is close to being a superset of Java.The j2cs translator performs the usual work of a Java compiler's front-end: it lexically analyzes the input, parses it, builds an abstract syntax tree, and traverses the tree to build a symbol table. It performs as little semantic checking as possible on the assumption that the generated C# code will be checked by the C# compiler. The final step is to traverse the tree and generate that C# code.

A few features of the Java language are very difficult to handle. Generic methods and classes are one, another one would be the `enum` type when used in its full generality. These relatively advanced features are not currently supported by j2cs on the assumptions that puzzles posed by Code Hunt will not require use of such features, and that most users of Code Hunt will not be advanced programmers.

The biggest challenge for providing a working translator was the handling of the Java API. The `java.lang` package is fundamental to the language and much of it needs to be supported by j2cs. For Standard Edition 7 of Java, the package contains 37 classes, 9 interfaces, 3 `enum`'s, 27 exception classes, and 23 error classes. If we look at just the `Math` class in that package, it provides 54 static

methods and two static fields. Instead of undertaking the massive effort needed to provide special-case translations for calls to each one of the thousands of methods provided in the `java.lang` package, we have provided an extensible translation mechanism and populated it with translations for the most common method calls plus those needed for the Code Hunt puzzles.

The mechanism is to provide a collection of translation templates. There is one file for each class in the Java API which has some translation support. That file is formatted as source code for the Java class which implements the desired methods *except that* the body of each method is C# code. Additional information needed by the translator is provided in the form of Java annotations. Extracts from *our* template file named `String.java` appears below.

```
@CSRewrite(JavaPackage = "java.lang",
          ClassName = "string", PartialClass = true)
public class String{
    …
@CSRewrite(Inline = true)
  public static String toString(int i){
    return i.ToString();
  }
  …
}
```

This template tells the translator that a Java expression like `String.toString(27)` is to be replaced with the C# code `27.ToString()`.

Most calls to methods in the `java.lang` package have trivial translations, just replacing a method name with a similar name. For example, `Math.sin(x)` simply becomes `Math.Sin(x)` in C#.

There is a major caveat with this approach. The replacement code is C# code, yet it has to be parsed as though it were Java code. We are yet to hit a case where that restriction causes a problem, although it could certainly happen.

5. EXPERIENCE IN DESIGNING PUZZLES

It is desirable to design puzzles that provide both fun and learning experiences. We have designed over 300 puzzles as learning materials for introductory programming, and for an engaging game at a contest at ICSE 2011 [4]. We also had experience of designing puzzles for a software engineering course to help students master the concept of design patterns. We discovered that much care and thought are needed when both fun and learning experiences are intended.

While the basic game provides continuous feedback, sometimes users get stuck and don't know how to proceed in the search for a correct program. In order to avoid frustrated users who give up, we have developed a hint mechanism to give additional information. For any given incorrect program, using code synthesis techniques, we can find a small code change that takes the program closer to a correct program. The synthesizer's search space is automatically directed by examining other users' attempts and solutions, so that no manual specialization to each puzzle is necessary. We then suggest the derived code change to the user.

Our hope is that this new hint mechanism will even out the experience of Code Hunt for players from different backgrounds.

6. RELATED WORK

In the learning environment, there are many systems that evaluate code. These all work by having a specification such as that on www.TuringCraft.com "Given that an array of int named A has been declared, assign 3 to its first element." and running test cases on the program. Students are familiar with this form of question; it is regarded as *work*, which is boring at worst, useful practice at best. Code Hunt might have exactly the same code hidden in one of its puzzles, but the aura of mystery intrigues users and keeps their interest.

In Code Hunt, the user has to perform the search-based game play to derive a matching program. An obvious research question is whether this search-based approach can be automated. Lakhotia [6] applied a generic programming system to automate solving coding puzzles, using the previously existing Pex4Fun platform as a backend. In one example, this system could fully automatically win in 76.57 steps.

On a practical level some of the code evaluator systems advertise that they run in the browser or in the cloud, but they actually require deployment on a local machine (e.g. CloudCoder www.cloudcoder.org needs two Linux servers). In contrast, Code Hunt runs directly in any modern browser and is backed by cloud execution in Azure.

Although the following comments from the community were for the precursor of Code Hunt, Pex4Fun, we fully expect them to apply to Code Hunt:

> HedonicPh0enix: **The geekiest fun you can have**
> Gide0nSkye: **Really cool app to help flex my programmer muscles**
> str8flushAKQJT, rated 10/10: **First game of this type. Very impressive.**
> JoshuaJEarl, rated 10/10: **Really cool concept and good execution. Like the code snippets and Intellisense.**
> Jace4Dana, rated 10/10: **Probably the most fully featured app on any mobile platform. Awesome idea and really intriguing implementation. Can't wait for later versions!!**

7. FUTURE WORK AND CONCLUSIONS

Code Hunt represents a high-impact educational gaming platform that not only internally leverages fitness values to guide test/clue generation but also externally offers fun user experiences where search-based test generation is manually emulated.

Code Hunt is a novel approach to build on serious search-based testing for a very large community, that of coders, and especially learning coders. Because the test cases are always changing, and are built on the mined data, players get a fresh experience every time. To code with the growing amount of data, the entire system runs in the cloud on Azure.

Our first task is to get user feedback on the game aspect of Code Hunt, and to improve that as required. Features we would like to add include being able to set time limits for puzzles, for use in competitions, tests, or lab sessions.

We would like to find out the current boundaries of what can be checked in a puzzle in terms of language structures. We then want to extend what can be checked by improving the underlying analysis engine.

REFERENCES

[1] Godefroid, P., Klarlund, N., and Sen, K. DART: directed automated random testing. In *Proc. PLDI* (2005), 213–223.

[2] Kelleher, C., and Pausch, R. F.: Using storytelling to motivate programming. *Comm. ACM*, July 2007/Vol. 50, No. 7, 58-64.

[3] Tillmann, N., and de Halleux, J. Pex – white box test generation for .NET. In *Proc. TAP* (2008), 134–153.

[4] Tillmann, N., de Halleux, J., Xie, T., Gulwani, S., and Bishop, J., Teaching and Learning Programming and Software Engineering via Interactive Gaming. In *Proc. ICSE* (2013), 1117–1126.

[5] Xie, T., Tillmann, N., de Halleux, J. and Schulte, J., Fitness-Guided Path Exploration in Dynamic Symbolic Execution, in *Proc. the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009)*, IEEE, June 2009

[6] Lakhotia, K. En Garde: Winning Coding Duels Through Genetic Programming. In *Proc. ICSTW SBST (2013), 421 - 424*