Building Games to Learn from Their Players: Generating Hints in a Serious Game

Andrew Hicks, Barry Peddycord III, and Tiffany Barnes

North Carolina State University,
Raleigh, NC
{aghicks3,bwpeddyc,tmbarnes}@ncsu.edu

Abstract. This paper presents a method for generating hints based on observed world states in a serious game. BOTS is an educational puzzle game designed to teach programming fundamentals. To incorporate intelligent feedback in the form of personalized hints, we apply data-driven hint-generation methods. This is especially challenging for games like BOTS because of the open-ended nature of the problems. By using a modified representation of player data focused on outputs rather than actions, we are able to generate hints for players who are in similar (rather than identical) states, creating hints for multiple cases without requiring expert knowledge. Our contributions in this work are twofold. Firstly, we generalize techniques from the ITS community in hint generation to an educational game. Secondly, we introduce a novel approach to modeling student states for open-ended problems, like programming in BOTS. These techniques are potentially generalizable to programming tutors for mainstream languages.

Keywords: Serious Games, Hint Generation, Data-Driven Methods.

1 Introduction

BOTS is a serious game designed to teach basic programming concepts to novice computer users and programmers [6]. BOTS, in its current state, contains no mechanisms for personalized feedback or problem ordering. One method of providing such feedback is to have experts create it for each problem. However, BOTS features open-ended problems with many possible solutions, as well as user-generated problems, making such expert annotation difficult. In this paper, we describe an effort to incorporate ITS-like personalization through data-driven hint generation.

Our contributions in this work are twofold. We generalize ITS hint-generation techniques to an educational game, and introduce a novel approach to modeling student states for open-ended programming problems. It is our hope that these techniques can be further generalized to programming tutors for mainstream languages in future work.

S. Trausan-Matu et al. (Eds.): ITS 2014, LNCS 8474, pp. 312–317, 2014.

2 Prior Work

Intelligent Tutoring Systems (ITS) have been shown to be effective at improving student performance [1,8]. ITS originally relied heavily on subject matter experts to anticipate common mistakes and misconceptions, but in spite of subject matter knowledge, experts are not always able to detect difficulties or misconceptions (the "expert blind spot") [10]. Additionally, such content is very costly, with Murray [9] estimating a cost of around 300 expert hours to create one hour of content in an ITS. Data-driven methods are proposed as a way of combating these effects, and can provide students individualized help based on previous observations.

The developers of Deep Thought (a propositional logic tutor) employed a method called Hint Factory [12]. As users work on problems, their actions are used to build a Markov Decision Process (MDP). This was later generalized by Eagle, et al, defining an Interaction Network as a complex network containing data about student-tutor interactions. [4] Hints can be generated from this data by searching the Interaction Network for users with the same solution path. Based on the previous users' actions, a potential next step can be suggested. If no user has succeeded on that path before, wec an suggest the current user try a different approach.

Systems such as the Lisp Tutor [1] and ACT Programming Tutor[3] were developed using knowledge engineering. Recent attempts to automate programming tutors have started with hint generation; however, when compared with domains like Propositional Logic, representing programming using state-action pairs poses many more challenges. For example, equivalent solutions to a problem can be expressed in many different ways. Directly applying Stamper's Hint Factory could result in a sparse state space, and we would need many more records in order to provide hints to most students. Some approaches have attempted to condense these similar solutions. One approach converts solutions into a canonical form by strictly ordering the dependencies of statements in a program [11]. Another approach compares *linkage graphs* modelling how a program creates and modifies variables, with nested states created when a loop or branch appears in the code [7].

3 Context

In BOTS [6], the goal of each puzzle is to program a robot to move blocks into specific 'goal' positions on the map. The player controls a robot by writing a program in a graphical, drag-and-drop programming language, as shown in Figure 1. The language supports basic robot operations (move, turn, pick up block) and flow control constructs (variables, loops, and functions). Once the puzzle is completed and the solution terminates without an error, the player is given a score based on the number of instructions they have used. After completing the puzzle, the player is encouraged to make modifications to their program and complete the level again using fewer instructions.

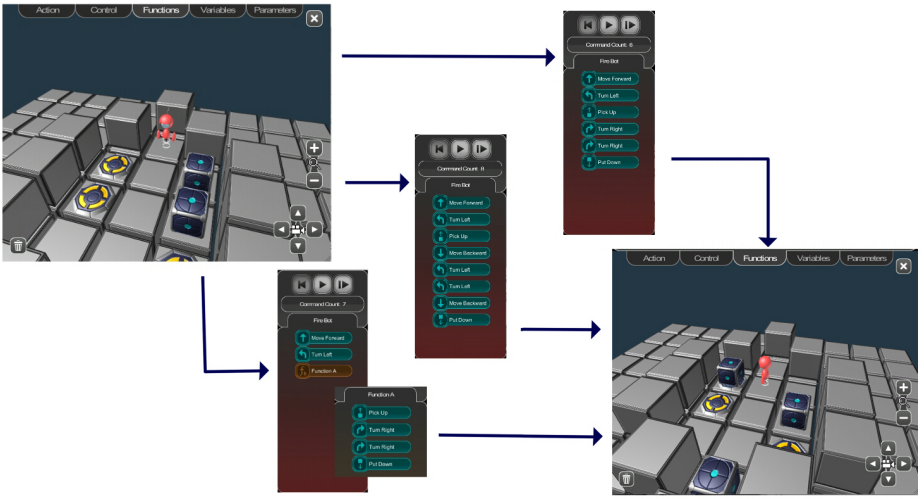

Fig. 1. In the game, players direct a robot to solve puzzles using a simple drag-and-drop programming language. Here we see three different programs which result in the same final state: The robot has moved a block from one side of the room to the other.

4 Methods

The intelligent tutoring system literature agrees on the definition of *interactions* as the low-level, click-by-click behavior of a student in a tutor [13]. This low-level representation is not ideal for our context, as the state space in BOTS is large and sparsely populated when compared to other tutors which have used Hint Factory. Instead, we will use the output of programs that players have written. For example, in Figure 1, the screenshot depicts the initial configuration on the left, and three distinct programs that each result in the output on the right. In our representation, one "World State" would encompass all three programs. We will show that this representation substantially reduces the state space and also facilitates the generation of meaningful hints.

To develop our alternative model, we first looked to other tutors that use data-driven hint generation, such as Deep Thought, a tutor for propositional logic used in introductory discrete math courses [2,4], and iList, a tutor that teaches the concepts of linked lists [5]. Both of these tutors use Hint Factory [12] to generate hints, but do so with different underlying models of the student states.

An interaction in Deep Thought is a single user input such as selecting a rule to apply. These states are represented as vertices of a graph, with edges between vertices being labelled with the logical rule (modus ponens, modus tollens) that was used to derive the most recently added state. The developers of iList also use Hint Factory, but their underlying model is based on snapshots of the tutor's internal state rather than the sequences of user interactions [5]. The authors look at the *results* of the student actions rather than at the actions themselves,

automatically resolving the situation in which multiple unique sequences result in an identical state. In order to find similar states, the authors compute which internal states are isomorphic to each other. For this work, we represent the output of a student's program as a grid representing the size of the stage, with unique markers for boxes, switches, and robots, as well as a height map of the stage. An example can be seen in Figure 2. This way, regardless of the contents of their programs, students who are performing the same actions (such as putting a particular block on a particular switch) will be grouped into the same state.

5 Analysis

To test the practical applicability of this state representation for analysing student solutions and providing hints, we used a corpus of past data collected from middle school aged players in classes and STEM-related afterschool programs.

Table 1. Results of our method for 24 puzzles. Rows indicate the Puzzle ID, number of students who attempted the puzzle, number of individual attempts, number of unique programs, number of "hintable" output states, and number of unique output states.

Puzzle	Students	Attempts	Unique Programs	Hint-Generating States	Unique States
1	60	95	9	3	5
2	57	284	234	41	65
3	50	189	121	15	21
4	43	77	39	9	9
5	42	181	193	22	24
6	42	84	26	5	7
7	40	127	182	31	41
8	35	50	16	8	10
9	35	89	81	25	29
10	**33**	**227**	**325**	**79**	**130**
11	31	53	77	20	25
12	28	79	41	3	4
13	27	145	187	50	75
14	22	40	57	19	23
15	21	76	119	16	18
16	19	40	96	33	39
17	18	76	103	26	32
18	15	44	59	4	35
19	15	34	64	16	38
20	14	56	43	5	25
21	13	33	34	15	20
22	10	67	71	18	23
23	8	30	25	13	16
24	8	13	32	0	22

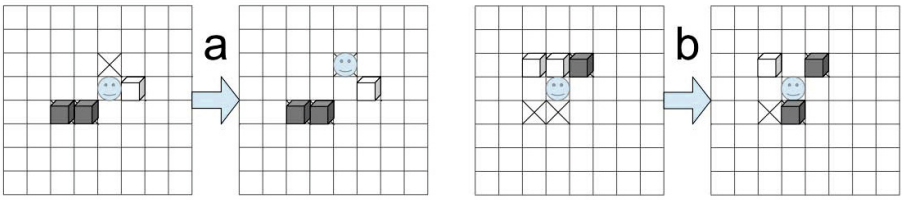

Fig. 2. Two of the generated hints for a simple puzzle. The blue icon represents the robot. The 'X' icon represents a goal. Shaded boxes are boxes placed on goals, while unshaded boxes are not on goals.

5.1 Hint Generation and State Space Coverage

To evalute how much our method was able to improve hint coverage, we compared the number of unique programs written to the number of unique output states. We then considered the number of those states for which a hint was available as shown in Figure 1. For the problems analyzed, our approach was consistently able to reduce the state space. For puzzle 10, a puzzle with a rich data set of solutions, we were able to reduce the state space from 325 unique programs to 130 unique output states. However, this reduction is meaningless unless we are able to provide useful hints from the created states. Out of 130 unique observed states, 79 states had potential to generate hints (that is, a student was in that state and then correctly solved the puzzle). 33 of these hints led to Error nodes, in cases where the Error was the only observed next-step. Of the remaining 45 hints, we found 42 to be meaningful. It is important to note that while this problem contained more records and students than other problems in our data set, the number of records was still quite small. Despite the lack of data we were able to provide hints more than half of the time, and able to provide hints for every state reached by multiple users.

6 Conclusions and Future Work

We have developed an approach to modeling student interaction with a serious game. This approach can be used to automatically generate hints with the Hint Factory algorithm. Rather than attempting to encode the programs or step-by-step interactions of the user, we instead use the resulting configuration of the world after each compilation of the student's code. Doing so, we are able to cover all of the unique code submissions with only a fraction of the states in the graph. While we use a naive implementation of Hint Factory, the hints that are generated are still useful and interesting, particularly those that lead out of error states. This work demonstrates that even with a small number of records, useful hints can be generated by grouping user actions according to their results. A similar system could be used in real-time games, generating hints based on important results or milestones rather than from low-level interaction data.

Acknowledgements. Thanks to the additional developers who have worked on this project or helped with our outreach activities so far, including Aaron Quidley, Veronica Catete, Trevor Brennan, Irena Rindos, Vincent Bugica, Victoria Cooper, Dustin Culler, Shaun Pickford, Antoine Campbell, and Javier Olaya. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 0900860 and Grant No. 1252376.

References

1. Anderson, J.R., Reiser, B.J.: The lisp tutor. Byte 10(4), 159–175 (1985)
2. Barnes, T., Stamper, J.C.: Automatic hint generation for logic proof tutoring using historical data. Educational Technology & Society 13(1), 3–12 (2010)
3. Corbett, A.T., Anderson, J.R.: Student modeling and mastery learning in a computer-based programming tutor. In: Frasson, C., McCalla, G.I., Gauthier, G. (eds.) ITS 1992. LNCS, vol. 608, pp. 413–420. Springer, Heidelberg (1992)
4. Eagle, M., Johnson, M., Barnes, T.: Interaction networks: Generating high level hints based on network community clusterings. In: EDM, pp. 164–167 (2012)
5. Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, C.W., Chen, L., Cosejo, D.G.: I learn from you, you learn from me: How to make iList learn from students. In: AIED, pp. 491–498 (2009)
6. Hicks, A.: Creation, evaluation, and presentation of user-generated content in community game-based tutors. In: Proceedings of the International Conference on the Foundations of Digital Games, FDG 2012, pp. 276–278. ACM, New York (2012)
7. Jin, W., Barnes, T., Stamper, J., Eagle, M.J., Johnson, M.W., Lehmann, L.: Program representation for automatic hint generation for a data-driven novice programming tutor. In: Cerri, S.A., Clancey, W.J., Papadourakis, G., Panourgia, K. (eds.) ITS 2012. LNCS, vol. 7315, pp. 304–309. Springer, Heidelberg (2012)
8. Koedinger, K.R., Anderson, J.R., Hadley, W.H., Mark, M.A., et al.: Intelligent tutoring goes to school in the big city. International Journal of Artificial Intelligence in Education (IJAIED) 8, 30–43 (1997)
9. Murray, T.: An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. In: Authoring Tools for Advanced Technology Learning Environments, pp. 491–544. Springer (2003)
10. Nathan, M.J., Koedinger, K.R., Alibali, M.W.: Expert blind spot: When content knowledge eclipses pedagogical content knowledge. In: Proceedings of the Third International Conference on Cognitive Science, pp. 644–648 (2001)
11. Rivers, K., Koedinger, K.R.: Automatic generation of programming feedback: A data-driven approach. In: The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013), p. 50 (2013)
12. Stamper, J., Barnes, T., Lehmann, L., Croy, M.: The hint factory: Automatic generation of contextualized help for existing computer aided instruction. In: Proceedings of the 9th International Conference on Intelligent Tutoring Systems Young Researchers Track, pp. 71–78 (2008)
13. Stamper, J., Koedinger, K., Baker, R.S.J.d., Skogsholm, A., Leber, B., Rankin, J., Demi, S.: PSLC datashop: A data analysis service for the learning science community. In: Aleven, V., Kay, J., Mostow, J. (eds.) ITS 2010, Part II. LNCS, vol. 6095, pp. 455–455. Springer, Heidelberg (2010)