Retention of Flow: Evaluating a Computer Science Education Week Activity

Alexander Repenning^{*}
School of Education
University of Applied Sciences
and Arts Northwestern
Switzerland (FHNW)
Windisch, Switzerland
alexander.repenning@fhnw.ch

Ashok Basawapatna[†]
SUNY Old Westbury
Dept. Mathematics, Computer
& Information Science
Old Westbury, NY, USA
basawapatnaa@oldwestbury.edu

Dorit Assaf
School of Education
University of Applied Sciences
and Arts Northwestern
Switzerland (FHNW)
Windisch, Switzerland
dorit.assaf@fhnw.ch

Carmine Maiello
School of Education
University of Applied Sciences
and Arts Northwestern
Switzerland (FHNW)
Windisch, Switzerland
carmine.maiello@fhnw.ch

Nora Escherle
School of Education
University of Applied Sciences
and Arts Northwestern
Switzerland (FHNW)
Windisch, Switzerland
nora.escherle@fhnw.ch

ABSTRACT

High profile computer science education events such as the Hour of Code can reach millions of students but without proper evaluation it is not clear what motivational and educational consequences the participation has. If, for instance, participants' levels of motivation towards the end of an hour long activity are significantly fading, then their perception of programming to be "hard and boring" may actually get reinforced. By simply measuring how far participants progressed with their projects we have been able to collect retention data from thousands of participants in a way that allows us to interpret these data in terms of not only cognitive but also technical and practical activity challenges. Inflection points overlaying a negative exponential retention distribution serve as indicators of these challenges with potential impact on Flow. Retention of Flow is an evaluation approach to analyze computer science education activities, including interactive tutorials and online programming environments, with respect to cognitive as well as affective challenges.

^{*}Department of Computer Science, University of Colorado, Boulder CO, USA

[†]School of Education, University of Applied Sciences and Arts Northwestern Switzerland (FHNW), Windisch, Switzerland

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGCSE '16, March 02-05, 2016, Memphis, TN, USA

© 2016 ACM. ISBN 978-1-4503-3685-7/16/03...\$15.00

DOI: <http://dx.doi.org/10.1145/2839509.2844597>

1. INTRODUCTION

Recently, there have been encouraging developments in getting computer science and computational thinking into K-12 classrooms [25][5][24]. For example, Computer Science Education Week (CS EdWeek) that took place on December 7-13 2014, reached more than 100 million students through its Hour of Code activities [24]. Such broadened participation and excitement around computer science education affords opportunities to motivate, enrich, and empower great numbers of students with the 21st century skills necessary to succeed.

CS EdWeek activities expose massive numbers of students to programming mostly through visual languages in a cyberlearning context [3]. They are typically aimed at students with no prior programming experience, consequently the activities are highly scaffolded with interactive tutorials embedded in the exercise environment itself [15]. These interactive instructions allow students to proceed self-paced through the activity. Examples include games wherein programming is the main task, such as "Lightbot" [11] and "Code Combat", as well as more open-ended visual language environments in which students create artifacts such as "Scratch", "App Inventor" [15][21], and AgentCubes online. In "Lightbot", for instance, students program a robot that navigates around obstacles. Similarly, the official 2013 Hour of Code activity included a game wherein students used block programming to control angry birds characters towards a goal [15]. Activities like "Lightbot" and "Code Combat" usually consist of predetermined, puzzle-like tasks, that increase in complexity with each level by introducing more sophisticated programming concepts throughout the exercise. The 2013 Hour of Code "Scratch" exercise guides students through the creation of a Christmas card. The idea is to initially get students started with a highly scaffolded tutorial in order to introduce the basic functionality of the programming environment. The environment then provides an open enough programming palette to let the students eventually diverge from the tutorial and to proceed to cre-

ate their own custom artifacts, such as an Android App with App Inventor, a custom Christmas card with Scratch, or a 3D game using AgentCubes online.

Massive exposure of students to computer science and programming activities is a first step towards motivating students and broadening participation. However, exposure itself is just the beginning of the story—the student activity-experience is also important. The hope is that students react to the activity as a “cliffhanger” wherein they are engaged at every step and are motivated to continue on after it being finished, opposed to having an “are we there yet?” disposition to the activity by being disengaged and simply waiting for the tutorial to be completed.

One might argue that regardless of the outcome, exposing as many students as possible to computer science is worthwhile. Research shows, however, that the first impression of computer science, especially at the middle school level, is crucial for students in deciding whether or not to continue subsequent pursuits of these activities [12][6]. These CS EdWeek activities can be a pivotal moment for hundreds of millions of students. If it is an unsatisfactory first experience, it could, actually, have negative implications for computer science education [7].

Even though answering these questions is vital for the discipline of computer science education, and the emergence of cloud-based programming environments enabling large data collection, there is not much state-of-the-art research yet that addresses these questions. One reason for the lack of research on massive exposure of students to computer science and programming activities might be the difficulty to collect data. Usually, there is little if any information on the participating students (e.g. age, gender, background) or the specific classroom contexts during these CS EdWeek Hour of Code events. We believe that despite these challenges we can still collect meaningful data, especially because we can exploit the large amount of data that participants of these events provide us. In this paper we will show that by looking at retention data, we can draw conclusions on the quality of the tutorial as well as learn about factors that lead to a student losing interest in the activity. A high student retention might be explained by several reasons: It can be high motivation leading to a cliffhanger effect but unfortunately it could also be a sign of high endurance to finish the tutorial despite boredom leading to the unpleasant “are we there yet?” experience. A low retention, however, might show a rather negative learning experience, be it due to overwhelming of the student which causes stress and anxiety, or technical challenges causing frustration, or under challenging causing boredom. These are examples of many other situations that might occur during these kind of education activities, which can cause a discontinuation of a student’s Flow and consequently, loss of retention.

We designed and introduced an Hour of Code activity based on an existing Scalable Game Design (SGD) curriculum which has been developed for more than twenty years at the University of Colorado [20]. This curriculum is usually taught in a classroom context by teachers who have completed a thorough training in SGD of at least 25 hours. Through that curriculum students are exposed to computational thinking tools throughout their school year and the learning through game design activities are well embedded in an educational context. Conversely, Hour of Code activities are designed in a way as to enable teachers to introduce it in

their classrooms without any previous training and to make it accessible for any student from the age of nine and older. Furthermore, we aimed at designing an activity that it is well scaffolded in the beginning to enable a low-barrier introduction to the programming environment but leaves room for open-endedness as the students reach the end of the tutorial, to hopefully give them a cliffhanger experience. Ideally, it will motivate students to continue working on their game projects at the same time it might inspire teachers to look more detailed into SGD or possible other computer science education curricula. Considering that we only get one hour to motivate students and teachers to continue being engaged in computer science education activities, it is of the essence that we evaluate the quality of the tutorial. It is the tutorial, after all, that guides students (and often teachers) through their first exposure to computer science and plays a crucial role in their first experiences with programming. We took the SGD curriculum and its programming environment AgentCubes online as resources to create our Hour of Code 3D Frogger activity. In this paper we do not evaluate the SGD curriculum and the AgentCubes online platform, as we refer to existing relevant evaluation and research publications [19][20]. We rather focus on our specific Hour of Code tutorial we created from these resources and evaluate it by looking into student retention. While we do not expect to provide final answers with this first study, we hope that both our approach and our findings will prove relevant and useful for further research in massive exposure of students to programming activities.

2. THE 3D FROGGER ACTIVITY

Based on the Scalable Game Design curriculum, we provided in 2014 a CS EdWeek Hour of Code 3D Frogger activity¹ that enabled students to create a game inspired by the 80’s arcade game Frogger using the browser-based 3D environment AgentCubes online. The positive motivational aspects of this activity have already been established on the basis of experiences reported by thousands of students [20]. Notably, this included the first ever CS EdWeek activity launched in Switzerland and a pilot study in Mexico [8][9]. To support U.S., Mexico and all the participating Swiss cantons, the activity included an embedded tutorial video in English, Spanish, German, Italian, and French. This video provides step-by-step instructions to build a complete Frogger game, including custom characters that can be made from scratch, and employing programming pieces, such as conditions, actions and rules, to create in-game interactions between these characters. The agent gallery lists in-game characters called “agents” created by the user (in the this case a frog). Notably, students can create their own 3D characters by inflating 2D images (see Figure 1 Checkpoint 0: Frog is designed). The possibility to create custom characters and game worlds was found to be surprisingly important from a motivational point of view, due to an increased sense of ownership [19]. The embedded video tutorial walks students through the creation of a complete Frogger game. It is supplemented by an interactive table of its contents consisting of indexed links that allow students to (re-)access instructions for each single programming step. All in-game interactions are being implemented by “if-else” rules assigning actions to conditions (see Figure 1 Checkpoint 2: key

¹<http://www.csedweek.us> (2015 version)

Figure 1: Retention graph for the Hour of Code 3D Frogger activity. The graph displays data of student projects with at least one line of code (LOC=1). The negative exponential curve is overlaid in blue. Program length corresponds to lines of code, or percentage of progress within the tutorial video. Color coded parts of the graph refer to subtasks including checkpoints within the embedded video tutorial. The identified inflection points on the graph refer to example challenges we found causing a drop of retention.

controlled frog). The conditions and actions palettes contain visual programming blocks that can be dragged into the behavior window, which is a work area that contains all rules. A more detailed description about the AgentCubes online game programming environment can be found under this reference [13]. The Hour of Code 3D Frogger activity is designed in an open-ended way, that means after the completion of the embedded video tutorial the students are able to make their game more complex by adding more agents and interactions on their own. AgentCubes online is cloud-based, which gives students the possibility to return working on their game (or create a new one) anytime later, for example from home.

3. RELATED WORK

As pointed out in the previous section there is little research that investigates the effects of large-scale exposure of students to computer science and programming activities in the context of events like the CS EdWeek. Among the few, the ones most relevant to this paper are those by Piech et al. [18] and Lee et al. [16]. Both studies employ a data-driven method to evaluate online programming activities and focus on student engagement which they correlate with retention. Specifically, Piech et al. claim “that data of how previous students navigated their way to the final answer can be leveraged to autonomously understand the landscape of such assessments and enable hints for future students,” thereby potentially increasing future student retention. Lee et al. find that inserting in-game assessments after each level of their activity increases student retention. An important factor

we need to consider in our analysis refers to those aspects that can potentially work against student retention, killing their motivation, making them quit the activity and thereby impeding their learning. Ko [14] has defined several learning barriers, which all refer to cognitive challenges. In online programming activities, which are typically characterized by a complex context where many practical and also technical factors play a vital role, we claim that there are many challenges and learning barriers which are not cognitive.

Massive Open Online Courses (MOOCs) have analyzed retention, including Survival Analysis techniques, to better understand at what point in time and why students might drop a course [23]. Though often on a larger time-scale, MOOC retention analysis shares many of the same challenges with the CS EdWeek activities stemming from the unknown populations of students who enroll. In recent studies, MOOCs have shown to be characterized by a negative exponential retention graph and, by some estimates, retain under 6.5%. In the future, the method presented here can be extended to MOOCs, for example, a programming class that assigns partially-scaffolded online assignments.

We strongly agree with Piech and Lee that student engagement is an important aspect of informal learning in computer science and programming, and that it can be correlated to retention, but we further claim that the scope of analysis needs to be broadened beyond issues of cognition. In particular, we claim that is important to include affective concerns [17] such as the framework of Flow [4]. Specifically, the framework we have utilized to keep students engaged in end-user programming is called the Zones

of Proximal Flow [2]. The Zones of Proximal Flow starts with Csikszentmihalyi's diagram of Flow and inserts Vygotsky's Zone of Proximal Development [22] between the regions of Flow and Anxiety. The strategy employs this in an end-user programming context as follows. The teacher challenges students with a game creation activity matched to their skill level. However, throughout the course of the activity some students might drift towards Anxiety; at this point, scaffolding targeted at these students' Zone of Proximal Development hopefully guides them back into Flow. In order to make the theoretical Zones of Proximal Flow framework actionable, we have researched breaking down games in terms of constituent computational thinking skills and creating a path from games to simulations through a framework entitled Computational Thinking Patterns. We have also developed methods to assess computational thinking skills by validated automatic evaluation of student artifacts called Computational Thinking Pattern Analysis [1].

We employ three broad categories of challenges that could impact Flow. Cognitive challenges are challenges of understanding that may be caused by overly complex or simply confusing tutorial instructions. Technical challenges such as losing a network connection may seem trivial in nature but could be responsible for significant impact on Flow when for example the goal is to create a project and that project gets lost. Practical challenges, such as the end of class time, again, may result in significant impact on Flow. We do not claim for these categories to be exhaustive but from our experience with Scalable Game Design classroom observations [20], in schools around the USA, we find them to cover most of the cases that have measurable impact on students state of Flow.

4. METHODOLOGY

We can begin activity evaluation by employing the large numbers of participants to investigate retention of Flow. By analyzing student retention through the activity we can begin to identify potential problem areas in the activity. We calculated student retention based on the program length or lines of code (LOC) a student has created in AgentCubes online. Specifically, every time a user modifies their program for a given agent in AgentCubes online, by adding methods, rules, conditions, and actions, an XML file corresponding to that agent's programmed behaviors is modified on the server. Analyzing all the XML files corresponding to each agent allows for an operationalization of program length akin to a lines of code metric. Using this measure of program length, we can begin calculate student retention through the activity. At each program length (LOC), we can measure the number of students still retained in the activity by summing the number of games with lines of code greater or equal to that program length. For example, if a student programmed a game of length 80, then that student is counted from a program length of 0 to a program length of 80. To put another way, that student was *retained* until a program length of 80.

Following this measurement we looked at the relationship between this measure of retention and different levels of progression through the activity. Due to the highly scaffolded nature of the tutorial where every student initially implements the same code as well as the assumption that most of the students follow the tutorial, we can correlate lines of code to position in the tutorial video. We realize that

students that did not follow the instructions of the tutorial and decided to program something else on their own might distort the data. However, with the large number of participants in events such as the CS EdWeek, we are able to collect data that drowns out this effect.

5. RESULTS

Figure 1 shows the retention graph for the Hour of Code 3D Frogger activity collected from 5,512 student projects in the context of CS EdWeek that took place in the U.S., Switzerland, and Mexico during December 7 - 13 2014. We solely include data of games that consists of at least one line of code (LOC=1). For this activity it implies that the first Frog keyboard "up" movement control has been implemented. Those datasets of student that only created a Frog agent (LOC=0) but didn't implement any behaviors were sorted out prior to analysis. The colored subsections of the graph displayed in Figure 1 represent each subtask of the embedded video tutorial. The first blue section is the Frog keyboard "up" movement control, followed by the orange subsection representing the implementation of all four keyboard directions i.e. "up", "down", "left", "right", and so on.

The Frogger game eventually is built by implementing each of these video tutorial subtasks. The x-axis of the retention graph represents the progress within the video tutorial, that means it is directly correlated to program length, or lines of code. The y-axis represents the percentage of students retained based on the number of projects that were included in the data, which equals number of students, assuming each student created and submitted only one project during this hour-long activity. The trend of retention is a negative exponential fit overlaid in black on the plot (goodness of fit: $r - squared = 0.993$; $F(1, 212) = 31324.837$, $p < 0.01$).

The retention graph presents several inflection points where students tend to discontinue the activity. The analysis of the game artifacts of students who discontinued the activity at these inflection points revealed particularly interesting patterns that we categorize as cognitive, technical, and practical challenges. The inflection points give us feedback in which position of the video tutorial we lost a significant number of students and by identifying the nature of these challenges we can take measures to improve the quality of the activity. As follows we describe the three challenge categories through examples we found.

5.1 Cognitive Challenge

Cognitive challenges pertain to situations in which students' code included errors. For instance, by analyzing game artifacts of students who discontinued the tutorial at program length LOC=27 (Figure 1) we uncovered a programming error previously seen in classroom observations: It is the task to implement the Frog's keyboard movements in four directions "up", "down", "left", and "right". The correct implementation involves four separate "if-else" rules, each containing a keyboard event condition and a corresponding Frog moving action. That means, if the user hits one of the arrow keys on the keyboard, the respective "if-else" rule executes and lets the Frog to move into the desired direction. This Frog movement is a tutorial subtask which is represented in by the orange colored subsection of the retention graph in Figure 1. The correct implementation has a program length of LOC=45. The identified inflection point at LOC=27 within this subsection of the retention graph re-

vealed that numerous students tried to implement the Frog movement by adding all four keyboard arrow conditions into one single “if-else” rule. By doing that, however, the code results in a different logic: for this rule to evaluate to true, all four conditions must be met corresponding to all four keyboard arrow keys being pressed at the same time (i.e. “up” AND “down” AND “left” AND “right”). This error is a typical problem for students, the difference of the correct and incorrect implementation in AgentCube online’s visual user interface is only slight. Obviously, this is very frustrating for the students since they perceive to have everything implemented correctly according to the tutorial, but the Frog doesn’t move at all. This kind of challenge we define as cognitive, it can emerge due to a misunderstanding of the mechanics of the programming environment, or because of more general problems. In this case, improvements of the programming environment or the activity must be considered.

5.2 Technical Challenge

Technical challenges correspond to problems such as networking issues, memory issues, and software bugs. Analyzing the inflection point discovered around program length LOC=45 revealed a common technical problem encountered by the participants. Specifically, students using the Internet Explorer browser would get an error message near this point when they tried to run their game. The instructions for the CS Ed Week activity specify that it works best with Chrome, Safari, or Firefox having WebGL enabled. However, this information is only shown at the beginning of the activity and requires the user to scroll down within the initial activity screen. The information will not show up again at a later point during the activity. When this Internet Explorer compatibility error occurs, the programmed Frog behavior is unreadable and there is no possibility to recover the student project without back-end server intervention. This is the point where the activity unfortunately must be quit, and it is not visible that it might be a browser issue. We assume that happened to a large number of students, the inflection point in the retention graph might correspond to this student loss.

5.3 Practical Challenge

Practical challenges refer to external environmental factors that affect student retention. In Figure 1 a practical challenge was identified at program length LOC=59. Now the student has implemented the frog win-state, that is the goal where the frog has to navigate to during the game. The student has created a minimal and simple functioning game where a user can navigate a Frog towards a goal using keyboard arrows. At this point the first three subtasks (colored blue, orange, and yellow) of the retention graph in Figure 1 have been implemented. This is a moment, where usually an hour has been passed and the class might have ended. Some teachers reserved additional class time, so that students had time to continue the activity.

6. DISCUSSION

It is not clear how to normalize retention. A very large number of people may come to some web-based activity only to instantly abandon it again because they did not really understand what they would get into. To that end, it often makes sense to define a minimal commitment point to define

the 100% point of retention. Piech [18] normalizes his retention data by only counting the number of participants who complete the first of the 20 activities as 100%. Similarly, we only count the number of participants who have created a least one line of code and normalize them as 100%.

Assessing retention of Flow can be difficult because it is not clear what good or bad retention is. One measure could be the retention drop rate as defined by how much retention is dropping per time. In the case of hour of code activities, for instance, it could be argued that the hourly retention drop rate could be indicative of Flow. The 2013 Angry Birds activity analyzed by Piech [18] appears to have a retention drop of about 35% in one hour which is nearly identical to the retention drop of the 3D Frogger activity. However, the shape of the retention distribution appears to be qualitatively different. Angry Birds retention seems to drop linearly, whereas 3D Frogger drops negative exponential. Partially, this could be explained with the closed nature of the puzzle-based approach of Angry Birds, which only features 20 finite puzzles. 3D Frogger, in contrast, allows participants to continue due to its open ended nature. That is, when users do reach the end of the tutorial the activity does not necessarily have to end. Users keep a link to their project capturing not only the programming but also the 2D and 3D design work they have created. Additionally, they have full access to the programming environment and not just a limited subset. Data indicate that users do indeed continue along a trajectory suggested by the exponential fit curve presented in Figure 1 beyond the final checkpoint (LOC=215). This could be considered an indicator of Flow or perhaps even of the activity being a cliffhanger. Continuing beyond the end of the tutorial would not be possible with most puzzle-based approaches because users do not actually have the ownership of projects which they could continue.

A fundamental challenge in this research is to properly interpret inflection points and to correlate their manifestations with initial cause of a problem. Our retention data does exhibit a number of easily identifiable inflection points (Figure 1), but it is not always clear if these inflection points necessarily pinpoint the moment when the problem was caused. For instance, in the cognitive challenge example of Section 5.1, the temporal gap between making a programming mistake and its manifestation could be significant. This instance of a cognitive challenge is rooted in a confusion about how rules work. When participants create a faulty rule with all four key conditions and all four move actions in the same rule, they will not notice their mistake before they finish and test the frog cursor key control behavior. This mistake is a big Flow detractor manifesting itself significantly as an inflection point, because not only the move “right”, move “left” and move “down” keys do not work, the originally move “up” rule stopped working as well. The possibility is high for a student to give up the activity at this point. In cases like this, the inflection point is several steps behind the programming step starting the problem, which makes debugging much more difficult. A potential solution for this particular cognitive challenge is to extend the programming tool with a proactive critiquing system that will catch these kind of programming errors as early as possible [10].

Data-driven research methodologies that are based on large numbers of participants afford new analytical approaches for meaningful interpretation. During the Computer Science Education Week in 2013, we reached nearly a quarter mil-

lion participants. Each project consisted not only of code but also of individually created 3D agents that populated sometimes complex game worlds. For the purpose of employing this rich data collection as feedback to improve an activity, it is necessary to reduce this information significantly. With our methodology to investigate retention of Flow, we have gone to the extreme of reducing each project to a single “lines of code” indicator. To get meaningful retention data, is it necessary to get some kind of critical mass of participation as our retention of Flow approach may now scale down well.

7. CONCLUSION

We have shown in this paper that it is possible to evaluate open online programming activities that involve a large number of participants by looking at student retention. We collected data from thousands of students during 2014 CS EdWeek that took place in the U.S., Switzerland, and Mexico. The data showed the nature of student retention closely followed a negative exponential trend. Inflection points in the activity retention data indicated challenges which we could categorize as cognitive, technical, and practical. We demonstrated that these challenges might be responsible for a drop of Flow and consequently, loss of retention. As more and more learning tools are being cloud-based, our presented method can be used to improve similar activities that are highly scaffolded.

8. ACKNOWLEDGMENTS

This research has been funded by Swiss Hasler foundation and the National Science Foundation (G.No:1312129, 1138526). Any opinions, findings, and conclusions or recommendations expressed are those of the author(s) and do not necessarily reflect the views of these foundations.

9. REFERENCES

- [1] A. R. Basawapatna, A. Repenning, and K. H. Koh. Closing the cyberlearning loop: Enabling teachers to formatively assess student programming projects. In *Proceedings of SIGCSE '15*, pages 12–17. ACM, 2015.
- [2] A. R. Basawapatna, A. Repenning, K. H. Koh, et al. The zones of proximal flow: guiding students through a space of computational thinking skills and challenges. In *Proceedings of ICER '13*, pages 67–74. ACM, 2013.
- [3] C. L. Borgman, H. Abelson, L. Dirks, et al. Fostering learning in the networked world: The cyberlearning opportunity and challenge. a 21st century agenda for the national science foundation. *Report of the NSF task force on cyberlearning*, 2008.
- [4] M. Csikszentmihalyi. *Finding flow: The psychology of engagement with everyday life*. Basic Books, 1997.
- [5] A. D’Alba, K. Huett, A. Remshagen, et al. ucode@uwg: Building kids’ interest in stem-c. In *Proceedings of SITE '15*, volume 2015, pages 96–99, 2015.
- [6] J. Denner, S. Bean, and J. Martinez. the girl game company. *Afterschool Matters*, page 26, 2009.
- [7] C. Duncan, T. Bell, and S. Tanimoto. Should your 8-year-old learn coding? In *Proceedings of WiPSCE '14*, pages 60–69. ACM, 2014.
- [8] N. Escherle, D. Assaf, A. Basawapatna, et al. Launching swiss computer science education week. In *Proceedings of WiPSCE '15*, pages 11–16. ACM, 2015.
- [9] N. Escherle, S. Ramirez-Ramirez, A. Basawapatna, et al. Piloting computer science education week in mexico. In *Proceedings of SIGCSE '16*. ACM, 2016.
- [10] G. Fischer. Domain-oriented design environments. *Automated Software Engineering*, 1(2):177–203, 1994.
- [11] L. A. Gouws, K. Bradshaw, and P. Wentworth. Computational thinking in educational activities: an evaluation of the educational game light-bot. In *Proceedings of ITiCSE '13*, pages 10–15. ACM, 2013.
- [12] C. Hill, C. Corbett, and A. St Rose. *Why so few? Women in Science, Technology, Engineering, and Mathematics*. ERIC, 2010.
- [13] A. Ioannidou, A. Repenning, and D. C. Webb. Agentcubes: Incremental 3d end-user development. *Journal of Visual Languages & Computing*, 20(4):236–251, 2009.
- [14] A. J. Ko, B. Myers, H. H. Aung, et al. Six learning barriers in end-user programming systems. In *VL-HCC 2004 IEEE Symposium*, pages 199–206. IEEE, 2004.
- [15] D. Kumar. Digital playgrounds for early computing education. *ACM Inroads*, 5(1):20–21, 2014.
- [16] M. J. Lee, A. J. Ko, and I. Kwan. In-game assessments increase novice programmers’ engagement and level completion speed. In *Proceedings of ICER '13*, pages 153–160. ACM, 2013.
- [17] R. W. Picard, S. Papert, W. Bender, et al. Affective learning—a manifesto. *BT Technology Journal*, 22(4):253–269, 2004.
- [18] C. Piech, M. Sahami, J. Huang, and L. Guibas. Autonomously generating hints by inferring problem solving policies. In *Proceedings of 2nd Conference on Learning@ Scale*, pages 195–204. ACM, 2015.
- [19] A. Repenning, D. C. Webb, C. Brand, et al. Beyond minecraft: Facilitating computational thinking through modeling and programming in 3d. *IEEE Computer Graphics and Applications*, (3):68–71, 2014.
- [20] A. Repenning, D. C. Webb, K. H. Koh, et al. Scalable game design: A strategy to bring systemic computer science education to schools through game design and simulation creation. *ACM Transactions on Computing Education (TOCE)*, 15(2):11, 2015.
- [21] M. Resnick, J. Maloney, A. Monroy-Hernández, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.
- [22] L. Vygotsky, E. Hanfmann, and G. Vakar. *Thought and language*. MIT press, 2012.
- [23] J. Williams, D. Paunesku, B. Haley, et al. Measurably increasing motivation in moocs. In *AIED 2013 Workshops Proceedings*, page 55, 2013.
- [24] C. Wilson. Hour of code—a record year for computer science. *ACM Inroads*, 6(1):22–22, 2015.
- [25] M. Yildiz and K. Scharaldi. Introduction to engineering and computer science in teacher education: Hour of code project. In *Proceedings of SITE '15*, volume 2015, pages 8367–8375, 2015.