# On the Nature of Fires and How to Spark Them When You're Not There

Sarah Esper
University of California, San Diego
La Jolla, CA
sesper@eng.ucsd.edu

Stephen R. Foster
University of California, San Diego
La Jolla, CA
srfoster@eng.ucsd.edu

William G. Griswold
University of California, San Diego
La Jolla, CA
wgg@cs.ucsd.edu

## ABSTRACT
Traditionally, computer science education research contributes new tools, techniques, and theories to improve *institutionalized* learning spaces – e.g. classrooms. However, we take the position that the study and improvement of computer science learning spaces *outside* the classroom are just as important.

We take a step toward illuminating the critical qualities of non-institutional computer science learning spaces by engaging in a grounded-theoretical examination of first-hand accounts of non-institutional learning. To further study the topic, we attempted to recreate (in the lab) a learning environment with many qualities that characterize non-institutional learning. To make this possible, we employed a modified version of *CodeSpells* – a video game designed to teach Java programming in a way that engenders the sense of sustained, playful, creative exploration driven entirely by the learner. This study introduced 40 girls, ages 10 to 12, to programming for the first time. We use the results of both studies to develop a theoretical framework which we use to examine existing tools such as Scratch, Alice, and educational games in a new light.

## Categories and Subject Descriptors
K.3.2 [**Computer Science Education**]: Introductory Programming

## General Terms
Human Factors

## Keywords
Grounded Theory, Authentic Practice, Origin Stories, CS0, CS1, Gamification, Active Learning, Informal Learning Spaces

## 1. INTRODUCTION
That minds are "fires to be kindled rather than vessels to be filled" was first expressed by Plutarch and has become one of the most oft employed quotations within pedagogical discourse. Still, there would appear to be some magic and mystery to the sparking of flames: it is much easier to develop curricula and textbooks that seek to fill up minds with material rather than to spark a love of learning. Not surprisingly, it is a rare student indeed that catches fire in a classroom. Furthermore, our "factory" model of education has come under such heavy criticism in recent years, [1, 2, 3] that one might begin to suspect that some forms of institutional instruction may have a dampening effect on the sparking of flames. This suspicion (further supported anecdotally by several decades of the authors' personal experiences in academia), prompted us to study the nature of flame-sparking by investigating how such flames manifest themselves "in the wild" – i.e. outside of classrooms.

Indeed, the field of computer science is rife with stories of children (many strikingly young) who discovered computers and taught themselves to program without the benefit of formal instruction. We collected various such accounts from a variety of sources. The following section gives a grounded-theoretical analysis of these accounts and begins to set forth a list of five qualities that appear to correlate with the eventual sparking of a lifelong love of programming.

We then sought to recreate these five qualities in a laboratory study with 40 girls, ages 10 to 12, exposing them to programming for the first time using a video game called *CodeSpells*, which supports many of the aforementioned qualities by default. Our goal was to study these qualities "up close", in order to refine our understanding of them. We present the results of this second study in Section 4.

Ultimately, using the insights gained from our grounded-theoretical analysis and our laboratory study, we are able to further illuminate how a passion for programming spark "in the wild". Although our studies are both of non-institutional learning environments, we believe (as in [4]) that the study of these environments can help shed new light on the process of learning in general. Thus, in Section 5, we discuss various contemporary software tools for teaching programming – looking at them in the new light offered by our analysis of non-institutional learning.

## 2. *FORMAL* VS. *INFORMAL* LEARNING
In related literature, a distinction is generally made between "formal" and "informal" learning. However, we prefer the terms "institutional" and "non-institutional" because, as pointed out by Sefton-Green [4], the terms "formal" and "informal" are somewhat overloaded. "Formal" can refer either to "organized" (as opposed to "disorganized") knowledge acquisition or to "institutional" (as opposed to "non-institutional") settings. This can lead to confusions: for example, one could imagine a disorganized (or student-driven) acquisition of material within the confines of an academic classroom – which is difficult to classify as a "formal" or "informal" experience. Thus, we use "institutional" (and "non-institutional") to make clear that we are discussing settings that fall within (or out side of) traditional academic institutions.

There has been wide research on informal learning spaces, but relatively little that pertains directly to non-institutional *computer science education*. The only two studies that we know of both involve ethnographic research on how Scratch (a novice programming environment) has been employed voluntarily by minority teenagers in a Los Angeles-based Computer Clubhouse [5, 6]. Although, the clubhouse environment is technically an institution, we classify the setting as non-institutional because it is not a traditional academic classroom environment. Students have free rein over their activities and can even play video games if they wish.

Thus, this paper adds to the small amount of prior knowledge on non-institutionalized computer science education by performing two new studies, using two new methodologies, and offering novel analysis of non-institutional learning to better illuminate the nature of learning in general.

## 3. ORIGIN STORIES STUDY

Since our research interest lies in the notion of sparking flames that lead to *lifelong* passions, we elected not to begin by studying existing non-institutional learning spaces – e.g. the clubhouse-like settings mentioned above, or after-school programs. After all, the children partaking in these spaces are (at best) only just *beginning* a lifelong computer science career. Instead, we collected first-hand accounts from contemporary computer science professionals, whose lifelong passions began in non-institutional settings. In this way, we hoped to be better equipped to draw conclusions about the non-institutional learning qualities that correlate with long-term excitement toward programming.

Ultimately, we accumulated 30 of these "autobiographical origin stories" (as we call them) from 3 different sources. We first drew 12 such stories from: 1) the Computing Educators Oral History Project [7], and 2) a qualitative study performed by Hewner and Guzdial [8]. To obtain more data, we also obtained 18 origin stories from 3) a survey we conducted ourselves, wherein we instructed participants to respond to the simple prompt: "Describe your first experiences with programming." We sent out this survey to the faculty and graduate student population at our university as well as to industry professionals via email and LinkedIn. In all cases, we selected only first-hand accounts where it was clear that the participants were describing their first experience as a programmer within a non-institutional setting.

The demographics of the autobiographers were intentionally quite diverse: ranging in age from graduating college seniors to industry professionals to retired. There were 11 males and 14 females. The genders of the 5 authors acquired from the study performed by Hewner and Guzdial is not directly reported. Some accounts involved programming on ancient Teletype machines, whereas others used modern hardware and environments like Alice. Since our intention was to begin constructing a very general theory of effective non-institutional learning environments, we felt that diversity was key.

### 3.1 Methodology for Origin Stories Study

Following the well-accepted procedures of grounded theory as set forth by Strauss and Corbin [9], we first engaged in open coding [9, ch5], where we coded each sentence and often time partial sentences. In the open coding, we found 13 subcategories: self-drive, enjoyable/emotional connection, "flow state" while programming, confidence/belief that you can succeed, investment in the results of the code, empowerment, creation of "meaningful" artifacts, lots of hours/re-visitation, access to immediate feedback, wrongness is on a spectrum and is therefore not binary, access to support, feeling addicted, losing track of time.

We followed open coding with axial coding [9, ch7], which allowed us to derive 5 distinct categories based on the open coding. Table 1 shows how the open coding results were connected to the 5 categories found in the axial coding and in the next section we define and give examples to our 5 categories.

**Table 1. Results of Grounded Theory coding of Origin Stories**

| Axial Coding Results | Open Coding Results |
|---|---|
| Learner-Structured Activities | Self-driven, Access to immediate feedback, Access to support |
| Exploration/ Creativity/ Play | Creation of "meaningful" artifacts, Investment in the results of the code |
| Programming as Empowerment | Enjoyable/emotional connection, confidence/belief that you can succeed, wrongness is on a spectrum; not binary |
| Difficulty Stopping | "Flow-state" while programming, Feeling addicted |
| "Countless" Hours | Lots of hours/re-visitation, Losing track of time |

### 3.2 Results of Origin Stories Study

All 30 origin stories exhibited at least 1 of the 5 qualities. 20 of the origin stories exhibited 3 or more.

**Learner-Structured Activities**: An activity is learner-structured when the learner engages in activities that are not at all (or not entirely) structured by some outside force:

- "...when the Commodore PET came along and **I took it home one summer and taught myself** BASIC",
- "I was **by myself** with no assignments - **just me and the computer** to play with."
- "I... **bought books and wrote programs instead of taking courses** in high school."

**Exploration / Creativity / Play:** The act of creation through experimentation and play was common – a mentality of "What would happen if...?"

- "I'd also **try to add things**, so if it was print a 'hello world' string, I would make it ask for your name, and print 'hello <name>', and if your name was a friend's name, **I'd add an inside joke or something**."
- "I would **purposely try changing some of the parameters to see what would happen**"
- "And we had gotten a Radio Shack computer and I had **played around** with that."

**Programming as Empowerment:** Programming was commonly viewed as a means of increasing one's personal efficacy, self-esteem, sense of purpose, or social standing.

- "…so I was able to do it, and that **gave me confidence** that '**I knew computers**'"
- "To me, **computers are freedom**, they are entertainment and above all they are a **symbol of power and adulthood**."
- "Soon enough the computer was asking me what my name was and then asking me how I was and addressing me by name. **I'd made a new friend**! I think my most impressive program was when I got the computer to make different sounds."

**Difficulty Stopping:** We expected to discover that the autobiographers found programming enjoyable, but the origin stories included many accounts of how frustrating the process was. The common thread, however, is that the act of programming was perceived as an engaging and often addictive one – with descriptions liked being "hooked".

- "That was a moment where **I got hooked** because just the sense that you could program a loop that could do an array of any number of any size, any number of numbers, just seemed to be such a fantastic thing."
- "When that actually worked, I thought it was the coolest thing ever, and after that **I was hooked** =P"
- "[I was] becoming **so closely involved** with technology... [that] I was a little afraid of **being sucked into** the CS major stereotype..."
- "I went for Mechanical Engineering. But **my liking for computers never died...** and I took some private computer courses on the side"

**"Countless" Hours:** This is closely related, but distinct from, difficulty stopping. (Some autobiographers did not exhibit signs of addiction but still invested large amounts of time for other reasons.) Many origin stories describe spending so much time that the autobiographers could not readily quantify the amount, tending to use abstract generalizations like "countless" or "dozens" or metaphors like "journey" (which imply a large time-investment).

- "I wrote **countless Pascal programs** on my old PC. (I still have some of the old code, printed out, somewhere!)"
- "**My journey** has been full of excitements. **Since the beginning** I have enjoyed computers."
- "So back then it was really **very labor intensive**, was **not instantaneous by any stretch of the imagination**.."
- "I read the guide to my TI-89 calculator a **dozen times**, learning to make simple games or programs in it."

We stress that this study was intended to be a preliminary theory-building endeavor, not a means by which a definitive, overarching theory was to be achieved. Indeed, the main intent was to inform and structure the laboratory study that followed it.

## 4. LAB STUDY

Our laboratory study was intended to "enhance theoretical sensitivity" [9, ch6] through the use of questioning – moving a step beyond the brief, straightforward nature of the origin stories.

We wanted to be able to determine if 1) the above five qualities could be recreated under laboratory conditions, and if so 2) to study them "up close" in order to enrich our theoretical understanding of them. Furthermore, we contend that performing laboratory studies on non-institutional learning is extremely important if one's ultimate goal is to learn from non-institutional learning in order to help inform institutional learning environments. After all, a laboratory lies somewhere between a fully institutional setting and a fully non-institutional setting. This means that lessons learned in a laboratory may be more easily transferable to a classroom than lessons learned in the wild.

## 4.1 Methodology for Lab Study

### *4.1.1 Software and Experimental Setup*

*CodeSpells* is an educational video game – more specifically, an immersive fantasy role playing game designed to teach Java programming by immersing the player inside a 3D virtual world and a first-person storyline wherein she plays the part of an apprentice wizard [10]. *CodeSpells* teaches Java by giving players access to a novice-friendly API for crafting novel magic spells. We chose to use *CodeSpells* because its magic metaphor has been shown to be compelling and exciting to novice learners [ibid]. We felt that the game would be an engaging environment for girls in the 10- to 12-year-old age group.

Normally, learning in *CodeSpells* is encouraged by way of a series of quests that must be completed with the use of Java-based spell crafting. However, we particularly wished to study **learner-structured activities** and **creative exploration**. So we employed a version of *CodeSpells* that did not have explicit quests to complete. (The *CodeSpells* platform is quite extensible). Rather, players could walk up to in-game gnome-like characters who would give various spells to the player, along with simple explanations. Our hope was that these spells would serve as starting points for code exploration.

We recruited forty girls (ages 10 to 12) who had no prior programming experience in any language or programming environment. We gave them a short overview of the *CodeSpells* game mechanics – including how to write and edit code with the in-game IDE. We divided them into 12 groups of three and 2 groups of two, and encouraged them to explore the 3D world and to see if they could "do interesting things". We were purposefully vague, as we hoped to encourage a largely unstructured learning environment.

We then allowed the subjects to play *CodeSpells* for one hour while we observed. Data collection techniques during this time involved video and audio recordings of 6 groups; video was of the computer monitor so that we could record actual gameplay. We also assigned one undergraduate computer science major to each group; each undergraduate took notes on what the girls struggled with. Our main research team also took observational notes as they walked around the lab.

After one hour, we split the girls into groups of 12 to 15 members and engaged each group in a semi-structured group interview. The interview involved questions such as: "Describe what you have been doing for the past hour.", "Can anyone share something interesting they did in *CodeSpells*?", "Can anyone share something they were trying to do, what did you do to try to make it happen?", "Did anyone get stuck while playing, what did you do when you got stuck?", "If you had more time, what would you want to do next?", and "Can anyone describe to me what spells are and how you use them?". Data collection during the group interview was videotaped and the interviewer took brief notes.

## 4.2 Results of Lab Study

Our subjects played *CodeSpells* for the entire hour before we had to ask them to stop. Students expressed disappointment that it was "over so soon". 25 of the subjects showed interest in playing *CodeSpells* at home and wanted to know when it would be available for them to play. We consider this to be evidence that students experienced some **difficulty stopping**. And the lingering excitement is a rough indicator that novices might indeed play such a game for **"countless" hours** – though we would need to allow our users to play *CodeSpells* at home to measure this further.

Of the 6 groups of girls we recorded, roughly 90% of their time was spent exploring the 3D world and/or editing code (as opposed to, say, chatting amongst themselves). Also interesting is that the quest-less version of *CodeSpells* provides considerably less structure than most video games. In spite of this, subjects did not ask "What am I supposed to be doing?", nor did they seem at a loss for activities to engage in. The simple directive to "do interesting things" was sufficient for inspiring subjects to give structure and shape to their own activities. This strongly suggests that a **learner-structured mentality** arose within the lab study.

Some subjects tried changing method calls like `thing.levitate(3)` to `thing.hop()` or to `thing.blowup()` (which are not available in the API but are surprisingly correct syntactically). Even though these attempts failed to evoke the desired effect, students did not appear to become discouraged. Subjects also discovered quite valid changes in this way too – e.g. `thing.levitate(300000)`. This suggests a drive to **explore, play, and create.** And furthermore, this drive appeared to be *fueled* (rather than dampened) by syntax errors. Of the 6 groups video taped, 4 of them encountered some syntax error that they resolved either by undoing the error they introduced (55% of the time) or asking the undergraduate student that was near them (45% of the time). In all cases, acts of **self-structured activities** followed these interludes.

A particularly interesting phenomenon occurred with regard to what one might call "logic errors". One group of girls made the mistake of levitating an object so high into the air that it could not be reached. They were able to retrieve the object, however, by jumping onto *another* object and levitating it (and thus themselves) sufficiently high enough to reach the original object. This emergent use of code to surmount challenges of one's own making is an act that fits our definition of **exploratory play**. Logic errors were seen in all 6 groups recorded, but we note again that the exploration seemed to be *fueled* (rather than dampened) by things going awry.

During the lab study and more so in the group interviews we were encouraged to hear that the girls felt **empowered**. When changes to the code didn't accomplish what they wanted, they kept working towards their goal, trying different spells or different code changes until they eventually reached it. When asked about how they reached their goals, they conveyed that they "knew it could be done" and that they "just needed to figure out how". They described code as a "way to accomplish anything" within the 3D environment. At no point did they describe programming as a barrier to the **self-structured activities** and **creative exploration** in which they chose to engage.

## 5. DISCUSSION

### 5.1 Refining the Theory

With the benefit of the laboratory study, we were able to study 4 of the aforementioned qualities (with the exception of **"countless" hours**) up close. Our observations allow us to add a quality that seems to go hand in hand with **exploratory play**. Namely,

**A positive attitude toward "failure"**: An institutional setting often provides an objective and authoritarian definition of what "success" and "failure" are. It was striking to observe that, when left to their own devices in a *non*-institutional setting, subjects approached what would traditionally be labeled as "failures" (syntax and logic errors) without the slightest apparent awareness that they had done something "wrong". No one had ever told them that syntax errors were bad or that code ought to do what one expects it to.

Thus far, we have presented two studies, 1) a study of origin stories in computer science and 2) a laboratory study of 40 novice 10- to 12-year-old pre-programmers. In the first study, we were able to tentatively identify 5 qualities that correlate with non-institutional learning in such a way that leads to the sparking of a lifelong passion for computer science. In the second study, we showed that at least 4 of these qualities (all but **"countless" hours**) could be recreated in a laboratory setting. We further observed a striking relationship between a **positive attitude toward "failure"** and the tendency to engage in **exploratory play** – allowing us to further enrich our theoretical framework.

We believe that further study of non-institutional learning is critical if we, as educators, wish to better understand how the sparking of young minds happens in the wild. We feel that a clearer understanding of the conditions under which such a seemingly mysterious phenomenon naturally occurs can serve to shed light on how to recreate this phenomenon more readily within institutional settings.

Moreover, though, the theory we have developed so far can help enrich modern pedagogical discourses – for example, the ongoing discourse about tools for teaching novice programmers.

### 5.2 Applying the Theory

Even though a comprehensive theory of non-institutional learning is perhaps a long way off, the aforementioned theoretical framework is already sufficient to allow for existing tools to be discussed with a new vocabulary, leading to new insights. Consider, for example, some common tools for teaching introductory programming.

#### *5.2.1 Novice IDEs*

*Scratch* is a visual programming environment designed to allow novice programmers, particularly young programmers, to create media-rich results by dragging programming blocks into place to create programs [11]. The 3D *Alice* environment is a friendly IDE serving as a "stepping stone to computer science careers" [12]. *Alice* provides an experience that allows users to make their own movies or video games.

Such environments allow the production of visually stimulating effects – which certainly gives young pre-programmers more avenues for **exploration, play, and creativity**.

On the other hand, it is not clear whether such environments encourage the other four qualities. For example, these environments do not *overtly* seek to motivate **learner-structured activities**. Likewise, there is no overt effort to induce **"countless" hours** of practice. On the other hand, nothing in Alice or Scratch overtly *prevents* learners from spending **"countless" hours** on **learner-structured activities**. Then again, nothing overtly *prevents* a student from opening up Eclipse and playing around with Java for hours on end. It is no doubt a rare occurrence though.

When the above are embedded within a classroom environment, **"countless" hours** can easily be required of students – but at the risk of sacrificing **learner-structured activities** and **exploratory play**. A natural place to look for solutions to this problem is in educational video games.

#### *5.2.2 Educational Video Games*

There exists a long history of educational video games, some of which are intended to teach programming [13, 14, 15, 16]. Although programming-related educational games to date have not gained widespread popularity, one can still examine video games (in general) in light of the framework.

**Learner-Structured Activities**: Video gaming tends to be a self-driven and self-structured activity – even for individuals who are too young to apply the same kind of self-motivation to academic subjects.

**Exploration / Play / Creativity:** Games can be used to create rich 3D worlds that facilitate exploration. In *CodeSpells*, for example, the interplay between objects, physical laws, and magic makes for emergent properties that drive play and exploration.

**Programming as Empowerment:** In *CodeSpells*, for example, the code a player can write directly correlates with the efficacy of that player within the 3D world. What one can *code* and what one is empowered to *do* go hand-in-hand.

**Positive attitude toward "failure"**: Games are not always enjoyable. Players can lose. Characters can die. Tetris blocks don't always fit. Frustration is as common as euphoria. Yet it doesn't seem to phase gamers and is indeed an integral part of the experience.

**Difficulty Stopping:** Indeed, while not always strictly an enjoyable experience, video games inspire an unprecedented level of active engagement that (for some) borders on addiction.

**"Countless" Hours:** The gaming industry is a multi-billion dollar industry. At least one study shows that children spend 10,000 hours playing video games throughout childhood [17]. Clearly non-educational video games *do* spark lifelong passions… for playing video games.

One might then ask, if video games have all the requisite qualities, why aren't educational video games routinely sparking lifelong passions for all sorts of academic disciplines? Our response would be to point out that many explicitly *educational* video games actually lack some of the above qualities.

It is difficult for many educational games to support **creative exploration.** Consider, for example, the classic *Math Blaster*. In order to provide positive and negative feedback to the player, the game must be able check the player's answers to mathematical problems. This means that the game can only serve-up a series of checkable problems, each with an objective right or wrong answer. There is no room for creativity. The game cannot say to the player: "Do something mathematically creative. You'll get more points the more creative it is."

Similarly, many educational games have been criticized and called "chocolate-covered broccoli" [18] because the educational material interrupts what would otherwise be smooth gameplay. In other words, the education interrupts the fun stuff. Or, to put it the other way around, the fun stuff (chocolate) is intended to thinly disguise the education (broccoli). No doubt this has a detrimental effect on the **difficulty stopping** and the tendency to engage in **"countless" hours.**

In conclusion, it would appear that both novice-friendly IDEs and educational games alike leave something to be desired from the standpoint of sparking lifelong passions for programming. The gamification of programing environments like Alice or Scratch might be one solution. *CodeSpells*, for example, seeks to combine the best aspects of educational games with the best aspects of novice-friendly IDEs.

## 6. FUTURE WORK

Our main goal is to use lessons learned from highly successful non-institutional learning to shape and inform institutional learning. To this end, we are currently conducting a 6-week-long class in which unstructured play time in *CodeSpells* is supplemented with in-class instruction. Also, we have developed a multi-player, competitive version of *CodeSpells*, which we are utilizing to examine the non-institutional learning environment that has sprung up around a competitive *CodeSpells* team that practices three times a week while we observe.

Our grounded theory study on origin stories was somewhat limited because we reviewed static, written text and were not engaging in interviews. We would like to further define our 6 categories that classify non-institutionalized learning spaces through a series of interviews where we could employ the grounded theory technique more rigorously. This study has introduced us to the theory, but we would like to define this theory more detailed so that other education researchers may apply it when examining and creating tools and environments for novice programmers.

## 7. CONCLUSION

We take the position that it is highly relevant to study non-institutional learning in which young people teach themselves to program. We analyzed the origin stories of 30 individuals who eventually became successful in computer science. We identified five qualities that tended to occur across multiple origin stories and that can be tentatively posited to correlate with the sparking of lifelong passions for computer science.

To more deeply study these qualities, we performed a laboratory study with 40 girls (ages 10 to 12) and analyzed their experiences according to the aforementioned five qualities. This allowed us to refine our understanding of these qualities and even to observe a sixth quality – further deepening the theoretical framework.

Our contributions are three-fold:

- We give the first theoretical framework for understanding the conditions under which the sparks of lifelong learning are sparked in non-institutional computer science learning environments.
- We demonstrate two novel methodologies for further examining non-institutional computer science learning. (The small amount of prior work that exists uses only *in situ* ethnographic methodologies.)
- We contribute a new vocabulary for discussing and designing tools for novices, using our theoretical framework to point out opportunities for improvement in both novice-friendly IDEs and educational games.

Ultimately, we take the position put forth in [11] – namely, that studying non-institutional learning is a means by which to understand learning in general. After all, non-institutional learning has a unique character. It is unstructured. It is creative. It is play. Also, as we saw in our laboratory study, the absence of values implicitly instilled in institutional settings (e.g. that syntax errors are bad) can lead to differences in behavior. As such, the study of non-institutional spaces may serve as an untapped resource for computer science educators – a resource that can help inform how we structure our institutional spaces. We believe the study of these space can yield surprising new insights about what we're doing right as well as what we may be doing wrong. Ultimately, it is a line of research that can give us new ways to light fires and to keep them lit.

## 8. ACKNOWLEDGMENTS

This research is funded, in part, by two NSF Graduate Fellowships. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF. Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

## 9. REFERENCES

[1] Eric Mazur. 2009. Farewell, Lecture? Science, 323, 50-51 (2009).

[2] Christopher D. Hundhausen, N Hari Narayanan, and Martha E. Crosby. 2008. Exploring studio-based instructional models for computing education. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education* (SIGCSE '08). ACM, New York, NY, USA, 392-396. DOI=10.1145/1352135.1352271

[3] Andrew Luxton-Reilly and Paul Denny. 2009. A simple framework for interactive games in CS1. *SIGCSE Bull.* 41, 1 (March 2009), 216-220. DOI=10.1145/1539024.1508947

[4] Julian Sefton-Green. 2004. Literature Review in Informal Learning with Technology Outside School, 2004. A NESTA Futurelab Series -report 7.

[5] Kylie A. Peppler and Yasmin B. Kafai. 2007. From supergoo to scratch: exploring creative digital media production in informal learning. Learning, Media and Technology 32, 2 (2007), 149–166.

[6] John H. Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. Programming by choice: urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education* (SIGCSE '08). ACM, New York, NY, USA, 367-371. DOI=10.1145/1352135.1352260.

[7] CEOHP. 2008. Computing Educators Oral History Project. (April 2012). www.ceohp.org/.

[8] Michael Hewner and Mark Guzdial. 2008. Attitudes about computing in postsecondary graduates. In *Proceedings of the Fourth international Workshop on Computing Education Research* (ICER '08). ACM, New York, NY, USA, 71-78. DOI=10.1145/1404520.1404528.

[9] Anselm Strauss and Juliet Corbin. 1990. Basics of qualitative research: grounded theory procedures and techniques. Sage Publications, Newbury Park, Calif.

[10] http://sarahesper.ucsd.edu/CodeSpells, August 2012.

[11] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *Trans. Comput. Educ.* 10, 4, Article 16 (November 2010), 15 pages. DOI=10.1145/1868358.1868363.

[12] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. *J. Comput. Small Coll.* 15, 5 (April 2000), 107-116.

[13] Matthew Dickerson. 2011. Multi-agent simulation and netlogo in the introductory computer science curriculum. J. Comput. Sci. Coll. 27, 1 (October 2011), 102–104.

[14] Michael K¨olling. The greenfoot programming environment. 2010, Trans. Comput. Educ. 10, 4 (November 2010), 14:1–14:21.

[15] William H. Bares, Luke S. Zettlemoyer, and James C. Lester. 1998. Habitable 3d learning environments for situated learning. In Proceedings of the 4th International Conference on Intelligent Tutoring Systems, ITS '98, Springer-Verlag (London, UK, UK, 1998), 76–85.

[16] http://robocode.sourceforge.net/, August 2012.

[17] Clare Richards. 2003. Teach the world to twitch: An interview with Marc Prensky, CEO and founder Games2train.com. Futurelab. www.futurelab.org.uk/resources/publications_reports_articles/web_articles/Web_Article578.

[18] Amy Bruckman. 1999. "Can Educational Be Fun?" Game Developer's Conference, San Jose, California. (March 17th, 1999).