CodeSpells: Embodying the Metaphor of Wizardry for Programming

Sarah Esper
University of California, San Diego
Computer Science and Engineering
sesper@cs.ucsd.edu

Stephen R. Foster
University of California, San Diego
Computer Science and Engineering
srfoster@cs.ucsd.edu

William G. Griswold
University of California, San Diego
Computer Science and Engineering
wgg@cs.ucsd.edu

ABSTRACT
This paper addresses how CodeSpells uses the metaphor of wizardry, along with an embodied API to engage students in learning to program in Java. Giving novice programmers a concrete representation of code has been encouraged and shown to help students understand the concepts with more ease. There have been many attempts to improve the novice learning experience by providing: a visual programming language, a hardware component or an application that is more approachable. The benefit of this research is that students are better able to understand how abstract code effects the environment.

We build on this work through CodeSpells by immersing novices in the abstraction of code through embodiment to allow them to understand complex and abstract programming problems as if *they* were being affected by what they wrote. In this paper we present a new approach to novice programming environments, one that embodies the user and encourages a quick grasp of introductory concepts followed by a deep understanding through exploration.

Categories and Subject Descriptors
K.3.2 [**Computer Science Education**]

General Terms
Human Factors

Keywords
Computer Science Education, CS1, Embodiment, Metaphor

1. INTRODUCTION
Novices have often struggled with learning to program, often times due to their inability to connect abstract programs with their effects on the environment. Novices can also struggle with low self-efficacy due to their dis-immersion with their programming environment. In this paper we present features of CodeSpells [21] that make this environment particularly immersive and therefore an environment where novice programmers are encouraged to try hard and not to become discouraged.

Though there has been much improvement in novice programming environments, with visual programming such as Alice or Scratch [24,6], gamified environments such as Kahn Academy and Codecademy, and applied environments such as Media Computation [12], we believe that none has immersed the novice into an environment that not only encourages a deep understanding of the concepts, but also to instills a sense of determination.

In the next section we discuss how other languages and environments compare with CodeSpells. We then go deep into the API of CodeSpells, discussing the immersive and applied properties that we believe create a unique novice experience. We then present findings from an exploratory study and discuss the implications and future work.

ITiCSE'13, July 1–3, 2013, Canterbury, England, UK.

2. RELATED WORK

2.1 Programming Languages
Languages such as Pascal and BASIC were early efforts to make programming concepts easier for novices [1] by sacrificing some degree of authenticity. Today, tools such as Alice [24], Scratch [6], and Lego Mindstorms [9] take this trend to its logical conclusion, providing a syntax-free programming interface that involves dragging and dropping blocks of a program into their appropriate places. Similarly, the Toque [23] and Tern [17] systems allow programs to be constructed by acting out cooking actions (in front of a Wii) and by assembling physical blocks.

Yet, there has been a counter-trend toward teaching industry-standard languages in academia, at least to computer science majors. This has resulted in an Optimal Novice Language Debate over which industry-standard language strikes the best balance between palatability and authenticity [16,7,4,10,19,5].

CodeSpells avoids these either/or tradeoffs. It relies significantly on rich gameplay to inspire players to engage with challenging programming concepts and syntax in ways they might never voluntarily do otherwise. Also, the implementation of CodeSpells is factored into both game engine and programming engine, enabling the use of whatever programming language is preferred.

One of the design goals for CodeSpells was to immerse novice programmers in the effects of programs, specifically by focusing on video game features. Guzdial has penetrated the CS1 curriculum with his Media Programming pedagogy [12], which adds novice-centered libraries to Java and Python in order to assist programmatic manipulations of photographs, audio clips, and other rich media. Although students are not necessarily *immersed* with Media Computation, they are introduced to the complex concepts through an approachable (known) application: photo manipulation. Even if students do not have experience with programs such as Photoshop, they understand the basic effects of photography (e.g. greyscale, green-screen effect).

On the other hand, environments such as Scratch [6], Alice [24], the Logo family [3], and Lego Mindstorms [9] engage the user by providing a scene that can be watched and explored. This allows

students to fully control the environment they are effecting and therefore better understand implications of changes they make in the code. Though these environments offer a scene where students can watch coding effects in real time, they tend to sacrifice authenticity for this "immersion". CodeSpells, on the other hand, increases engagement and therefore authenticity.

2.2 Context of Programming

By giving the novice programmer a new domain in which to work, programs can take on a new significance, enhancing the learner's motivation and persistence. Recent enhancements to the "How to Design Programs" methodology [15] provide a scheme environment for programming 2D animations. As programming environments, both require external learning support and lack the motivational hooks provided by a first-person video game.

Logo programs allow the user to mobilize an on-screen "turtle"; Alice and Scratch programs allow the user to create 3D and 2D animations (respectively); and Lego Mindstorms programs allow users to affect microcontrollers embedded in a Lego apparatus.

The completeness of CodeSpells' contextualizing metaphor means that the game requires neither supplemental lectures nor textbooks, like the systems above.

All learning resources are seamlessly interwoven into gameplay, making the game into a self-contained learning environment that can be used even where formal instruction is unavailable. In addition, the storyline and 3D environment provide added engagement to sustain motivation, say, in the absence of an external motivator. This significantly increases the sense of immersion amongst students, not ever being taken away from the programming environment to "learn a new concept" and instead finding them throughout gameplay.

The quests provide novices with pedagogically relevant tasks. Coding constructs are introduced via the spellbook. Reasons to persist come from the first-person storyline, badges, and visually stimulating 3D environment.

Likewise, the CodeSpells approach is also distinct from the many popular game *building* environments, many of which are used in classrooms: e.g. Greenfoot [18], NetLogo [13], Unreal Scripting [11], the RPG creator [20], the Starcraft map editor, Kodu [14], the Spring RTS Engine [22], and ToonTalk [8]. CodeSpells is an RPG (Role Playing Game) that more fully immerses the students in the gameplay, but also in the learning. Most games used in the classroom require the students to be the "builders" or "engineers", but not actually be affected by the code. We wanted to focus on the students "feeling" the effects of their modifications and programs, not just "witnessing" them.

3. A NEW PROGRAMMING CONTEXT

3.1 An Immersive Video Game

3.1.1 The Metaphor

Magic. Magic is a metaphor that crosses both programming and fantasy RPGs. In programming, what an expert programmer can do is often referred to as "magical". As Brooks states "The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard, and a display screen comes to life, showing things that never were nor could be" [2]. It can then be considered that expert programmers have a suite of "magical spells" that they can use at any time to solve various problems. These spells might be small programs they have written, methods, or simply concepts they now understand (e.g. the "Iterate through an array spell").

Similarly, in most fantasy RPGs, players often have magic spells that they can use at any time to solve various problems. Comparable to the "spells" of expert programmers, RPG players may start with simple spells, and have to earn and learn how to use more complex ones. CodeSpells embodies the idea of "spells" as both an artifact that players can see, read and write and also an executable object that players can use to interact with the world and solve problems.

Figure 1 is an example of the spellbook provided to players as an in-game resource for game-play and learning. To fully immerse the players in CodeSpells we designed a similar resource that they might find in any other fantasy RPG, a spellbook. This spellbook provides them with information about how spells work and when they might be useful. Players can use the spellbook, along with the Non-Player Character's advice, to navigate through quests.

Most importantly though is the spellbooks dual purpose as a learning resource. Players are encouraged to read through the spellbook, but also to execute spells to test them out. Being in a non-competitive environment allows them to take their time in fully understanding the effects and limitations of each spell. Modifications can easily be made in the in-game IDE shown in figure 2. This particular resource is the gate to programming within CodeSpells and is essential to immersion and embodiment.

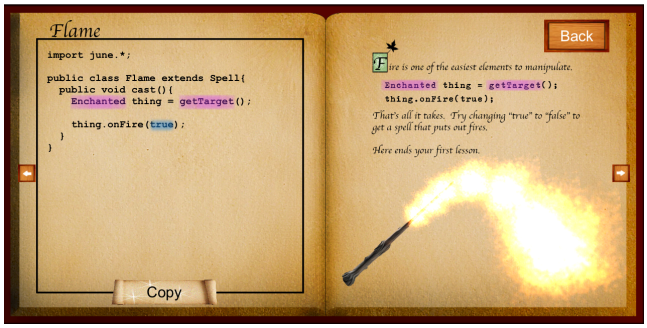

Figure 1. Spellbook for in-game resource (solving challenges and quests) and learning (understanding Java concepts).

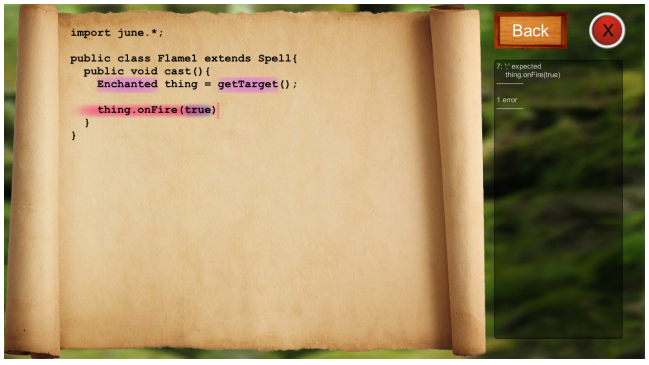

Figure 2. In-Game IDE that allows players to make modifications to the code very easily to test effects and limitations of spells.

3.2 DESIGNING FOR EMBODIMENT

In this section, will explore the API of CodeSpells, paying particular attention to how we designed classes and methods to encourage players to understand the effects of spells (programs) from a first person perspective.

3.2.1 An Innovative API

We designed our API to be similar to that of Java web applets where `Applet` is subclassed and the `run()` method is implemented, by having a superclass `Spell` and subclasses that implement the `cast()` method.

The first spell that is provided to the player in their spellbook is the Flame spell. The player casts this spell onto an item that is flammable, and then the item is set on fire.

```
public class Flame extends Spell{
   public void cast(){
       Enchanted thing = getTarget();
       thing.onFire(true);
   }
}
```

Our first challenge was making the spells powerful enough such that players could interact with the environment relatively easy, but authentic enough such that players could engage with the typical concepts presented in a CS1 course. We studied the balance of power and authenticity in Guzdial's Media Computation and within Alice and Scratch and decided on a threshold close to reality, allowing players to easily accomplish tasks that are easily done in the real world (setting something on fire, moving forward), but having difficult tasks be represented as complex spells that combine the simple API calls and typical CS1 level concepts such as loops, conditionals and parameters.

In addition to making complex magic difficult to attain, we wanted other, intuitively mundane tasks, to be easy. Our rule of thumb was that if it a task is typically performed without much conscious thought by someone in the real world, then it should require a minimal amount of code in the fantasy world. One such intuitive task is the ability to specify and talk about things in the world around us. We do this with various combinations of pointing, gesturing, and naming -- all of which can be done with little conscious thought. We felt that if such intuitive tasks required a non-trivial amount of code to be written then players would quickly become frustrated at the API's lack of expressiveness.

We tested the limitations of our API with 3 intermediate programmers (undergraduate students in the computer science major) making as many complex spells as they could in 3 months. We observed them during this process, improving upon the API when they struggled due to a minimal API. Though further refinement might be done once a larger scale study of CodeSpells and it's learning effects is complete, we are confident that the level of specificity in our spells will promote the understanding of typical CS1 learning goals.

Two key outcomes of this design process are intuitive mechanisms for (a) expressing entity reference and (b) specifying spatial manipulations. As will be seen, the basic design rule that emerged was to make the API work in ways that are analogous to the way humans think and work when performing everyday tasks. That is, not only should the interface make mundane tasks easy, but the interface should be easy to understand by analogy to everyday experience. As a result, there is a degree of embodiment designed into many of the operations, via reference to the avatar. In the next section we will explore how embodiment played a role in the effects of CodeSpells on novice programmers.

4. EMBODIMENT

4.1 Referencing Objects

4.1.1 Entity Reference

Obtaining a Java reference to a particular entity in the 3D world is something we wanted to make as easy as possible. We handled the trivial case using the aforementioned `getTarget()` function to reference any individual entity that the apprentice wizard is able to see and point to -- i.e. entities in close proximity to the player's embodied location in the virtual 3D world.

However, during the course of working with our intermediate programmers, it quickly became necessary to refer to more than one game entity, for example to move one entity to the location of another entity. Thus, we added the ability to reference certain special entities by name. For example, the player may obtain a reference to their own in-game avatar by writing `Enchanted e = getByName("Me")`. To make this practical, we gave permanent names to certain fixtures in our environment -- e.g., "Me" and "Rain", which the wizard can learn by reading her spellbook. The following spell can be used to make the local rain cloud named "Rain" follow the player around:

```
public class SummonRain extends Spell{
   public void cast(){
       Enchanted rain = getByName("Rain");
       Enchanted myself = getByName("Me");
       while(true){
          Direction toward_me =
             Direction.between(rain,myself);
          rain.move(toward_me, 5);
       }
   }
}
```

Not all entities are so special, of course. If a player wants to refer to a particular rock in a spell, she can give the rock a name and then later use `getByName()` to obtain a reference to the rock.

4.1.2 Referencing Groups of Entities

When building a bridge out of 20 rocks, however, our interns found it tedious to give a name to every rock and to declare 20 variables of type `Enchanted`. We handled this by adding API support for defining variably-sized lists of entities.

Our first attempt was to provide support for querying a radius around an entity (identified by `getTarget()`) using a `getWithinRadius(float)` method. However, it was too easy to accidentally obtain too many rocks, too few, etc. The root of the problem is that a radius of size X is difficult to visualize before casting one's spell.

The solution to this problem was to bring the senses into play. We designed a magic staff that can be used to designate a static area of effect. Putting the staff down on the ground *displays* a radius within which all entities are selected, and a textual name for the area is displayed above it. The area's radius can be resized to suit the player's needs. Supposing a staff has been used to create an

area, and it has been renamed to "Fire Trap Area", the following spell lays a permanent trap for one's enemies:

```
public class FireTrap extends Spell{
   public void cast(){
      Enchanted area
          = getByName("Fire Trap Area");
      while(true) {
         EnchantedList list
             = area.getWithin();
         for(Enchanted e : list)
             e.onFire(true);
      }
   }
}
```

4.1.3 Vector Math

Another challenge was to allow novices to manipulate entities in 3D space without expressing their ideas in the language of linear algebra, trigonometry, or geometry. That is, our API had to provide intuitive ways of performing spatial manipulations without using mathematical syntax. After all, human beings are able to perform a wide variety of spatial tasks (driving, building with blocks, throwing balls, etc.) without formal math skills. Because CodeSpells is a first-person game, we were able to simplify the API by leveraging the player's existing embodied intuition. That is: the ability to easily move and rotate one's avatar, and to write code that is dependent on the avatar's location and rotation, gives the player a surprising amount of expressivity without the need for higher-order mathematics.

In CodeSpells there are 4 avatar-referenced directions -- forward, backward, left, and right and 6 world-referenced directions -- up, down, north, south, east, and west. The following spell uses the "forward" direction to allow the player to fly when standing on top of the spell's target entity (e.g., a magic carpet). Because "forward" is dependent on the direction the player is currently looking, the behavior of the spell is affected at runtime by the player's movements and head rotations, flying in whichever direction the player's embodied avatar is currently looking.

```
public class Flight extends Spell{
   public void cast(){
       Enchanted target = getTarget();
       while(true){
          target.move
              (Direction.forward(), 0.2);
       }
   }
}
```

The flight spell is a good illustration of our general approach: Just as human beings are trivially able to push physical objects along their "forward vector" in the real world -- we wanted players to be able to easily employ magic to move objects forward in the fantasy world. Also, just as human beings can trivially comprehend how their own bodily rotations change what "forward" means, we wanted spells to be able to make dynamic adjustments when the player's avatar rotates. Aside from this use of the API's powerful and intuitive spatial affordances, the only "magical" API call used above is the basic `move()` operation. But it has been wrapped within a non-terminating Java control structure, so that it will run until the player manually stops the enchantment. Until then, the player can fly freely throughout her virtual 3D environment without knowledge of the underlying vectors and quaternions that make this possible.

4.1.4 Emergent Pedagogical Properties of Embodiment

The experience of embodiment instilled by immersive RPGs gives a CodeSpells program (spell) the potential to take on more personal significance than, say, a program written in Alice or Scratch. Rather than making movies or games (as in Alice or Scratch), a CodeSpells player is able to live out fantasies that are essentially embodied. By jumping on top of an inanimate object and casting the flight spell, a player can live out the fantasy of flight. Many of our users have expressed that this is enjoyable.

Thus, the emotional payoff for learning how to program in CodeSpells can be high. Furthermore, because we have designed the APIs for simplicity, a spell of just a few lines stands in the way of the novice and these moments of accomplishment and joy. The learner only needs to meet a handful of learning goals before receiving positive feedback.

5. EVALUATION

After validating that the CodeSpells API and environment would make it possible for programmers to design and write spells (programs) that were using typical CS1 learning goals, we decided to conduct and exploratory study to understand how novices responded to the API. Specifically we were interested in the ability of the novice programmer to:

- Use the spells correctly without modification,
- Modify the spells correctly to accomplish a desired goal and
- Create their own spells using the API that currently exists.

5.1 Methodology

We recruited 17 subjects to play CodeSpells (see figure 3). None of the subjects had prior programming experience in any industry standard language but all knew what "programming" meant.

Each subject completed a background questionnaire before attending the study. Before the study began each subject was walked through the game mechanics by one of our research team. We then encouraged them to attempt to solve the quests in the game, but did not provide any other guidance or assistance unless they requested it. We recorded each session with standard-video equipment. Finally, we conducted a 15-minute semi-structured exit interview where we queried their immersive experience.

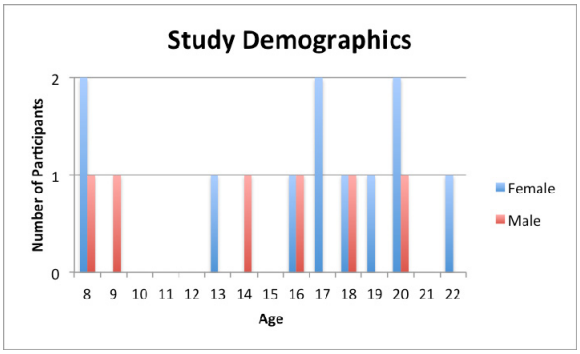

Figure 3. Study Demographics

6. RESULTS

Due to the explorative nature of this introductory study we were able to receive very intriguing results on the game-play behavior of our subjects. We will first discuss their ability to interact with the programs then discuss their sense of immersion in the game.

6.1 Using Spells

It is very encouraging that each one of our subjects was able to successfully cast at least 6 of the 8 spells that were in their spellbook (meaning they were able to successfully execute 6 of the 8 small Java programs). The documentation provided in the spellbook was not as detailed as a textbook or a tutorial, therefore to read and test spells which led to a correct understanding of those spells within only a 45 minute period with no prior experience with the CodeSpells API or any Java was astounding.

6.2 Modifying Spells

We also witnessed an average of 20 code edits in each 45-minute session. A code edit is defined as a successful change in a method argument, number of loop iterations, an entire API call or moving/adding/deleting entire lines of code. This level of interaction with code was surprising to us, especially since each code edit was a positive change towards a desired goal.

6.3 Creating Spells

Finally, 2 of our subjects were able to successfully write their own spells. Only 2 of our subjects wrote their own spells because in the short session only 2 were able to successfully complete all of the 8 challenges presented to them. Once they had completed the challenges, they were able to come up with their own challenge and design their own solution for it. One of these subjects designed a particularly interesting spell: a portal spell that used the staff (introduced in section 4.1.2 that was meant for referencing many objects) to create a portal on one side of the river and send anything that stepped inside the portal to the other side. This innovation in such a short period was very positive.

6.4 Physical Immersion and its Value System

As is the case for perhaps all game designers, our intention was to architect a new world -- to fully recontextualize computer programming. In our case, we wanted a world where magic exists and where wizards can break the laws of physics. But more than a world with new *mechanics*, we wanted a world with new *values* -- one where the protagonist feels entwined with the fate of the world, one where one's education is meaningful and valued, one where executable code is an exciting arcane mystery rather than a "thing for nerds", and one where anyone can safely learn those mysteries. We focused on the exit interviews to gather information about the effects of the immersive properties.

Physical Immersion. In each of the exit interviews, subjects talked about how their programs affected *their* world. When asked about times when they might have struggled a bit to accomplish a task, they typically described the situation as "I was trying to levitate the rock that I was standing on, but then I accidentally clicked on it and I fell off". It was also interesting to hear how these subjects would talk about this environment with others. When asked if they would play CodeSpells with peers, one subject responded "Yeah, that would be cool because we could work together to do harder things. Like I could levitate a rock while they caught it on fire." In our informal experience with students using other interactive environments, such as Alice, these speech acts differ in that our users seem to have an embodied experience with CodeSpells, whereas Alice users typically talk about the objects in the scene in the third-person.

Values Immersion. Along the same lines, using the environment of a video game was also very interesting as it applied to immersion. Our players seemed to not give up easily when attempting a difficult quest or challenge. This was evident by their exit interviews, when asked about their experience with particularly difficult challenges, one subject described her approach by saying "[I] never gave up -- because it's a video game!" This is not a typical sentiment held by novice programmers, especially with girls we often see a feeling a failure or discouragement when the program doesn't work properly. In this case, the immersive features seemed to have made the difficult challenges *fun*.

It seems that through the CodeSpells API, environment and metaphor, our subjects became immersed in their programming, something we hope all novices can do in an introductory setting.

7. DISCUSSION AND FUTURE WORK

This study is particularly interesting to the computer science education community because it has begun to address two major concerns: 1. Students' inability to understand how abstract programs effect the environment and 2. Students' determination to succeed during difficult programming challenges. We are excited by these initial results and are encouraged to continue this research with more subjects over longer periods of time.

In the next three months we will be running a large CodeSpells study with 35 9-10 year old students in a near-by elementary school. During this study we plan to fully examine and explore the immersion properties of CodeSpells and how each aspect (the API, the video game features, the 3D environment) effect student motivation, learning and efficacy. We are also interested to see how CodeSpells compares with programming environments such as Media Computation and Alice. We believe that CodeSpells will offer more motivation for students due to it's immersive qualities.

8. CONCLUSION

Our goal was to get novice programmers immersed in a programming environment that would lead to their ability to read, execute, understand, modify and create spells (programs) with determination and a positive view of their ability. We believe we have begun to show how CodeSpells is a viable solution for encouraging novice programmers to invest in their programming as they do with video games.

We plan to run a larger study where we can compare how novice programmers engage with CodeSpells and Media Computation (which is used in our institution's CS1 course), as well as other environments such as Alice and Scratch that also offer immersive qualities. We plan to improve on our API as we measure learning gains in our future studies, but posit that the level of authenticity versus power, along with the metaphor of magic, engages novice programmers such that they can feel positive about their struggles when learning their first industry-standard language.

Our contributions are two-fold: We have presented here an API and environment that has:

- Allowed novice programmers to read, execute, understand, modify and write CS1-level Java programs within a 45-minute period.

- Immersed novice programmers into their programming environment so much so that they have developed a determination to solve problems and a positive outlook on programming challenges.

9. ACKNOWLEDGMENTS

This work is supported in part by two NSF Graduate Fellowships and a Google Faculty Research Award.

10. REFERENCES

[1] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.* 37, 2 (June 2005), 83-137.

[2] Frederick P. Brooks, Jr.. 1995. *The Mythical Man-Month (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[3] George Lukas. 1972. Uses of the LOGO programming language in undergraduate instruction. In*Proceedings of the ACM annual conference - Volume 2* (ACM '72), Rosemary Shields (Ed.), Vol. 2. ACM, New York, NY, USA, 1130-1136.

[4] Janusz Jabłonowski. 2007. A case study in introductory programming. (CompSysTech '07), Boris Rachev, Angel Smrikarov, and Dimo Dimov (Eds.). ACM, New York, NY, USA, Article 82 , 7 pages.

[5] Jason Hong. 1998. The use of Java as an introductory programming language. *Crossroads* 4, 4 (May 1998), 8-13.

[6] 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.

[7] Jonathan D. Blake. 2011. Language considerations in the first year CS curriculum. *J. Comput. Sci. Coll.* 26, 6 (June 2011), 124-129.

[8] Ken Kahn. 1996. Drawings on napkins, video-game animation, and other ways to program computers. *Commun. ACM* 39, 8 (August 1996), 49-59.

[9] Lego Mindstorms. 2013. http://mindstorms.lego.com.

[10] Linda Mannila and Michael de Raadt. 2006. An objective comparison of languages for teaching introductory programming. (Baltic Sea '06). ACM, New York, NY, USA, 32-37.

[11] Magy Seif El-Nasr and Brian K. Smith. 2006. Learning through game *modding*. *Comput. Entertain.*4, 1, Article 7 (January 2006).

[12] Mark Guzdial. 2003. A media computation course for non-majors. *SIGCSE Bull.* 35, 3 (June 2003), 104-108.

[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] Matthew B. MacLaurin. 2011. The design of kodu: a tiny visual programming language for children on the Xbox 360. *SIGPLAN Not.* 46, 1 (January 2011), 241-246.

[15] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt and Shriram Krishnamurthi. How to design programs: an introduction to programming and computing. 2001. MIT Press. Cambridge, MA, USA.

[16] Michael de Raadt, Richard Watson, and Mark Toleman. 2003. Language tug-of-war: industry demand and academic choice. (ACE '03), Tony Greening and Raymond Lister (Eds.), Vol. 20. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 137-142.

[17] Michael S. Horn, Erin Treacy Solovey, R. Jordan Crouser, and Robert J.K. Jacob. 2009. Comparing the use of tangible and graphical programming languages for informal science education. (CHI '09). ACM, New York, NY, USA, 975-984.

[18] Michael Kölling. 2010. The Greenfoot Programming Environment. *Trans. Comput. Educ.* 10, 4, Article 14 (November 2010), 21 pages.

[19] N. Solntseff. 1978. Programming languages for introductory computing courses: a position paper. In *Papers of the SIGCSE/CSA technical symposium on Computer science education* (SIGCSE '78). ACM, New York, NY, USA, 119-124.

[20] RPG Creator. 2013. http://rpgcreator.net.

[21] Sarah Esper, Stephen R. Foster, and William G. Griswold. 2013. On the nature of fires and how to spark them when you're not there. (SIGCSE '13). ACM, New York, NY, USA, 305-310.

[22] Spring Real-Time Strategy Engine. 2012. http://pringrts.com.

[23] Sureyya Tarkan, Vibha Sazawal, Allison Druin, Evan Golub, Elizabeth M. Bonsignore, Greg Walsh, and Zeina Atrash. 2010. Toque: designing a cooking-based programming language for and with children. (CHI '10). ACM, New York, NY, USA, 2417-2426.

[24] Wanda P. Dann, Stephen Cooper, Randy Pausch. 2006. Learning to Program with Alice, Brief Edition. Prentice Hall, Inc. Upper Saddle River, NJ, USA.