AGENTCUBES: ENABLING 3D CREATIVITY BY ADDRESSING COGNITIVE AND AFFECTIVE PROGRAMMING CHALLENGES

Alexander Repenning
University of Colorado, AgentSheets, Inc., USA
ralex@cs.colorado.edu

Corrina Smith
AgentSheets, Inc., USA
corrina@agentsheets.com

Robert B. Owen
AgentSheets, Inc., USA,
bobowen@agentsheets.com

Nadia Repenning
AgentSheets, Inc., USA,
nadia@agentsheets.com

Abstract: Computational thinking has been identified as an essential 21st century skill. Unfortunately, many computer users, particularly children, consider programming to be hard and boring. There are two fundamental barriers to making programming more appealing: cognitive challenges to overcoming the hard part, and affective challenges to overcoming the boring part. AgentCubes is a new 3D creativity tool that addresses both barriers. AgentCubes addresses cognitive challenges by advancing beyond current drag and drop programming approaches that are limited to syntactic support. Instead, AgentCubes provides semantic support that helps users in fully comprehend and debug their programs. AgentCubes addressed affective challenges by including innovative 3D authoring mechanisms that let users create their own exciting 3D models and worlds. AgentCubes lets users share their creations as HTML5 web applications without needing Java or Flash.

Introduction

There is a recognized, urgent need to create an information technology (IT) enabled workforce with *computational thinking* skills ((PITAC), 2005). While there are diverging interpretations of computational thinking (Lin et al., 2010), a definition is emerging. A common thread is that computational thinking (CT) is not just programming, but also the ability to conceptualize computation with notions of abstraction and formal representations. This is not about trying to make people think like computers, nor trying to explain how computers work. Instead, computational thinking can be considered a synergistic combination of human skills with computer affordances to solve problems.

Unfortunately, CT has had little impact up to this point (A. Repenning, 2012). Most CT-related activities include some form of programming. Programming, in turn, suffers from a severe negative public perception problem (Prensky, 2008). Perhaps the most compact summary of this perception has been provided by a middle school student stating, *"Programming is hard and boring."* Notice the lack of a trade off. Students do not say, "programming is hard but very fulfilling." This suggests that there are two fundamental challenges to effectively teaching computational thinking:

1) ***Cognitive Challenge***: Programming is difficult for a large number of reasons (Pea, 1983). In Figure 1, the main cognitive challenges are represented by the *syntactic* challenge and the *semantic* challenge. The syntactic challenge refers to the need for students to understand how to write a correct program. For instance, in a traditional textual programming language such as Java, a single misplaced semicolon can quickly create a serious problem. Drag and drop, tile-based programming languages can often completely overcome the syntactic problem. With AgentSheets (Alexander Repenning, 1993), we pioneered drag and drop programming to create games and scientific models. However, visual languages are no panacea (Lister, 2011) as they do not address the much harder semantic problem. For example, understanding that a well-formed sentence must have a noun and a verb does not mean a student can write a best-selling book. The semantic challenge is all about the meaning of a program. This is the level at which program bugs manifest. The process of creating a useful and working program requires debugging, which Pea (Pea, 1983) describes as removing the discrepancies between the program desired and the program the programmer has actually written."
2) ***Affective Challenge***: The affective challenge is concerned with motivation. Why should students want to program? Is programming an end in itself or a means to an end? There is no simple answer. When trying to broaden the participation of students in computer science (Webb et al., 2012), it becomes clear that the activity must be relevant. Sorting numbers, while interesting to some, is likely to be considered an extremely boring programming project to most. The idea of game design is quickly gaining ground as an exciting activity that engages most students. The challenge then becomes one of scaffolding game design activities (A Ioannidou et

al., 2011). That is, game design activities are appealing to many students, but designing and implementing games is far from trivial (A. Repenning et al., 2010). Our experience with the Scalable Game Design project (Webb et al., 2012) suggests that it is possible to expose students to game design on a large scale with extremely high degrees of motivation, but the resultant efficacy depends on conducive pedagogies—especially when trying to foster the participation of women.

The "holy grail" of computational thinking is to create a CT tool that would change programming from "hard and boring" to "simple and exciting." Unfortunately, like the holy grail of myth, this goal remains elusive for a number of reasons. Cognitively, there is intrinsic complexity in programming that no approach can completely overcome. Affectively, interest and motivation are varied. No single activity will appeal to every student.

Over the last 15 years, we have gained rich experience by creating authoring tools and using them to teach an increasingly wide range of students worldwide. The Scalable Game Design initiative is running large-scale studies investigating student motivation and skills while learning CT skills through game and science simulation design. School test beds included affluent technology hubs and inner city, rural, and Native American sites. This experience helped us create what we believe to be the next generation of CT tools to make programming less hard and boring.

The focus of this paper is to illustrate how AgentCubes (Andri Ioannidou et al., 2008; A. Ioannidou et al., 2009; A. Repenning et al., 2006), a new CT tool for creating 3D games and 3D simulations, addresses the cognitive and affective challenges outlined above. We will explore AgentCubes, and then report on our experiences with a diverse set of users. This paper investigates the overhead of 3D programming by contrasting AgentCubes game design activities with related approaches to 2D and 3D.

Related Work

The space of programming environments facing the "programming is hard and boring" challenge is vast. To narrow our discussion of related work, we will focus on programming environments connected to computer science education to broadening participation through activities related to game design. Common to all these systems is the aim to make programming more accessible, typically through drag and drop programming, and to enable users to create projects significantly more interesting than sorting numbers. Figure 1 illustrates a number of activities and tools that could further this aim. The figure axes indicate some subjective measures that should be considered rough estimates. We have found it possible to gage affective challenges statistically. With AgentSheets, we collected data from over 8000 non self-selected students who expressed their level of motivation by indicating how interested they were in participating in similar game design courses in the future.

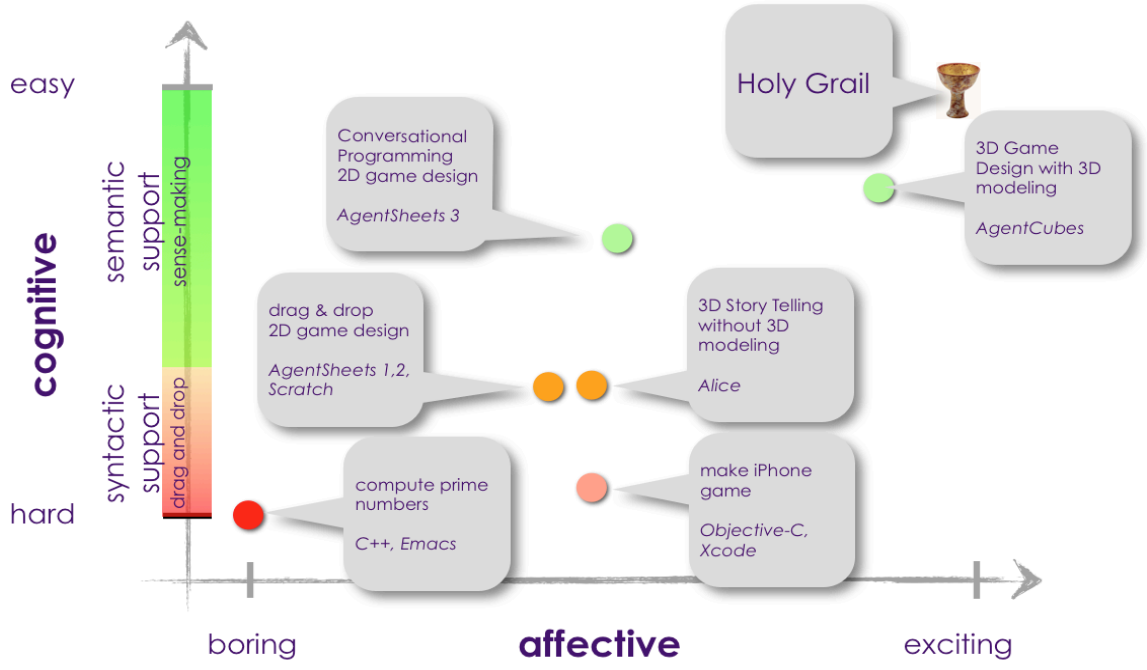


Figure 1: Cognitive versus Affective Programming Challenges: sample activities and sample tools.

The following sections describe a number of educational authoring systems:

AgentSheets. We have used the AgentSheets authoring environment to explore new programming approaches over the past two decades. AgentSheets93 took an agent-based graphical rewrite system approach, and combined rule-based programming with programming by examples (Cypher, 1993; Lieberman, 2001). While this approach let users quickly build a basic game, it did not easily allow them to move beyond the basics. AgentSheets95 dropped programming by example, and pioneered drag and drop programming for educational programming applications. Users programmed agents by dragging conditions and actions from palettes into rules. AgentSheets introduced Conversational Programming in 2011 (A. Repenning, 2011), and moved beyond the syntactic programming challenge to helping users with the semantics of programming.

Alice: story telling. Alice uses a drag and drop environment to create computer animations that enable story telling. The current version uses 3D models, and a later version is expected to use models from The Sims 2. There are no 3D modeling tools built in to Alice allow to make new 3D models from scratch.

Scratch: animations. Scratch enables users to quickly create interactive animations. Users can make their own 2D sprites with a built in image editor. A limited repertoire of math functions, low performance and the lack of tools such as plotters make Scratch less suited for STEM simulation building.

GameMaker: game design: GameMaker has a strong game design focus and is not intended for STEM simulation building. The authoring tools are primarily for 2D. 3D shapes can be imported but not created in GameMaker.

Kodu: game design: Kodu runs on the Microsoft Xbox 360 or in Windows, and is designed specifically for simple 3D game development. Kodu includes a number of sophisticated 3D shapes but does not include authoring tools to create new 3D shapes or edit existing ones.

Gentle Slope 3D Authoring in AgentCubes

The vision for AgentCubes is to provide an extremely accessible 3D authoring tool that enables users with no background in 3D modeling or programming to create interactive 3D worlds. Educationally, this suggests a kind of computational thinking that includes 3D fluency. A fundamental challenge to the notion of fluency is the need to define skills, explore motivational means of promoting skills, and devise ways to assess these skills. Some talk about programming as the new literacy. The focus of our research is to promote the notion of 3D fluency. People live in a 3D world, and thanks to computer gaming, today's computers are highly capable of processing 3D information. Unfortunately, creating computational 3D artifacts and games can be a truly daunting task. Even end users comfortable with 2D game construction are likely to find the transition to 3D to be difficult. Typically, they need to master a completely new set of tools to create, animate, and program 3D models. For instance, there is very little skill transfer from 2D paint programs like Photoshop to a 3D modeling editor such as Maya 3D. This raises the question: is this discontinuity a conceptual consequence of 2D versus 3D with potential roots in human cognition, or is it simply a consequence of computational tools that have emerged disjointedly for 2D and 3D applications?

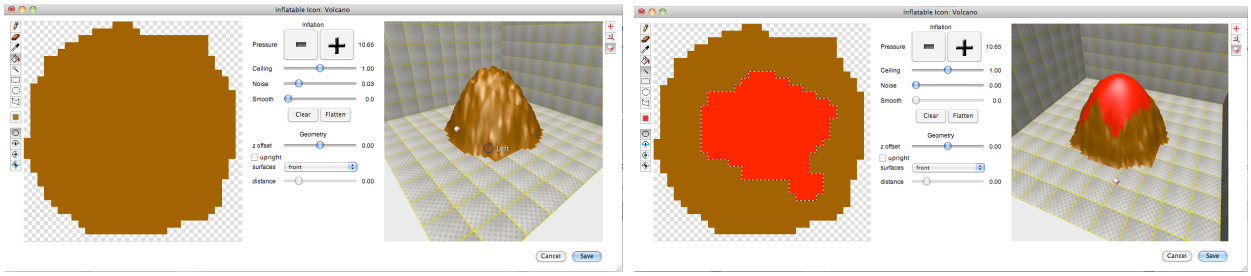
We promote 3D fluency through a gradual approach that we call incremental, or *gentle-slope 3D* (A. Ioannidou et al., 2009). We reconceptualize the universe of 2D and 3D tools and skills as a continuum rather than a dichotomy. AgentCubes supports 3D authoring through incremental approaches for all components of the 3D authoring process—modeling, animation, and programming. A gentle slope (Dertouzos, 1992) approach lets end users develop 3D games by first creating a 2D version of that game and then gradually moving along well-defined stepping-stones towards a 3D version. Ideally, this incremental process transfers existing skills to the point where end users work in 3D as easily as 2D.

Gentle Slope 3D Modeling

Gentle slope 3D modeling is enabled through the Inflatable Icons technology (Repenning, 2005). Instead of limiting end users to using only stock 3D art like The Sims in Alice, or professional 3D modeling tools with very steep learning curves such as Maya 3D, we enable them to gradually acquire 3D modeling fluency by creating their own 3D models. With Inflatable Icons, users draw 2D images and gradually turn them into organically shaped 3D models with thousands of polygons using diffusion-based inflation techniques. User testing has confirmed that students are able to make basic inflatable icons with minimal training. Our rather ambitious authoring benchmark is that we want to enable users to create their own inflatable icons after about one minute of training. We do not attempt to compete with high functionality 3D modeling tools by trying to build high-quality models. Instead, we want to enable almost anyone to create reasonably recognizable 3D sketches of everyday objects such as people, animals and cars. The

value of being able to create a 3D model, even one of somewhat low quality, it not simply artistic. We have found over many years of working with children worldwide making games in AgentSheets that the ability for a child to construct their own creations is highly relevant to the affective challenges. Even students playing top production quality 3D games at home deeply enjoy their own simple creations through increased ownership.

There are other end-user 3D modeling approaches, but Inflatable Icons are unique in the sense that they are based on the *paint-then-shape* principle—not the *shape-then-paint* approach employed in 3D end-user modeling tools such as Teddy (Igarashi et al., 2006), Google Sketchup and Sculptris. Figure 2 shows the AgentCubes Inflatable Icon editor. On the left hand side, with the 2D editor, the user draws a quick sketch using Photoshop-like tools—including pens for drawing and a bucket tool for flood fills. This is can be aided by employing the symmetry tool. More advanced shapes like mountains can be created by using noise functions to add 3D textures and ceiling functions to clip height values to fixed values. A somewhat irregular circle is drawn and filled with brown color. The shape is not only inflated but a lot of noise has been added to make the mountain look a bit rugged. A volcano requires lava, which is added as an irregular red shape. That shape is selected and its noise is removed. With the lava still selected a ceiling is pushed down to drop the level of lava below the level of the crater. Drawing can continue. The user added a little lava spill to the side visible in the 2D editor and 3D editor.



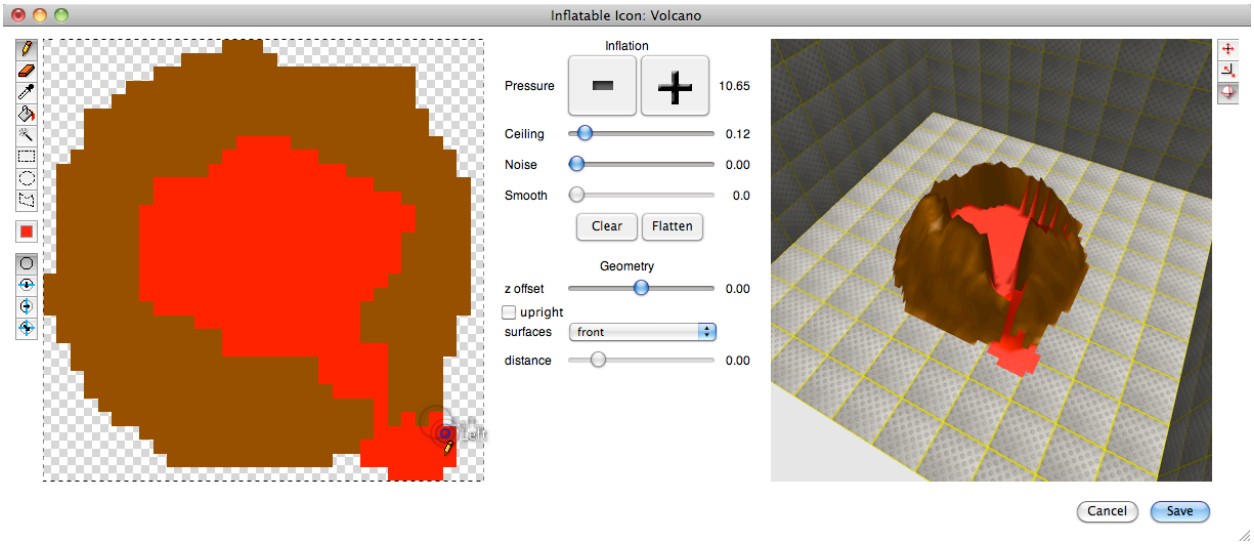
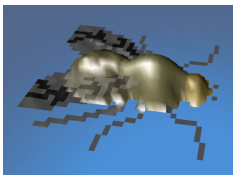
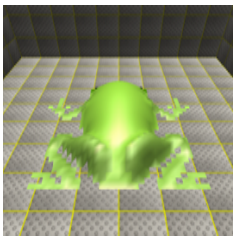


Figure 2: Using Inflatable Icons to make a Volcano: drawing outline, inflating with noise, filling in lava and pushing it down.



The user gets additional choices by being able to have front and back shapes, by connecting the front and back shapes, or by turning them into a cube. Even more options emerge from the ability to combine individual inflatable icons into composite shapes such as the fly, which combines a body with semi transparent wings. The frog represents the creation of another sophisticated 3D shape.



Our experiences with students (elementary school, middle school, undergraduate and graduate college level) creating their own 3D inflatable icons are overwhelmingly positive. The biggest concern we have received so far from teachers have been concerns about student becoming almost too excited about drawing inflatable icons, which sometimes resulted in less time left for programming tasks. To avoid the need to draw every 3D shape from scratch, AgentCubes includes a shape browser for selecting cubes, cylinders, spheres, tiles and, of course, inflatable icons.

World Design

In AgentCubes a world is a 4-dimensional structure containing rows, columns, layers and stacks. Layers are typically used in advanced 3D simulations. An example would be a SimCity-like simulation employing layers to represent different service layers of the City such as roads, the water system and the electricity system. Layers are considered an advanced concept, which is typically introduced at a later stage. Stacks are frequently used to construct complex worlds such as this CityTraffic simulation (Figure 3). Cars are stacked on top of roads, and building blocks, represented as cubes, are stacked up into skyscrapers. Players can select any agent in the world to become a first person agent. When the agent is a movable agent such as a car in the CityTraffic world, then the camera—which is attached to the agent—will follow the agent and turn when the agent turns.

Figure 3: A world contains stacked up roads, cars and building blocks (left). A car is selected and put into first person view (right).

Worlds can be connected and even hierarchical. In the student created Wizard Quest, the player navigates a set of intricate worlds such as the village (Figure 4) with the help of maps and advice given by the villagers. The player can step into a building. Every scene, including the village or the inside of a house, is a connected world that agents can teleport to. Users have additional control over worlds to customize their look and feel. For instance, they can control background colors, background textures, sky domes, light-source locations and parameters.

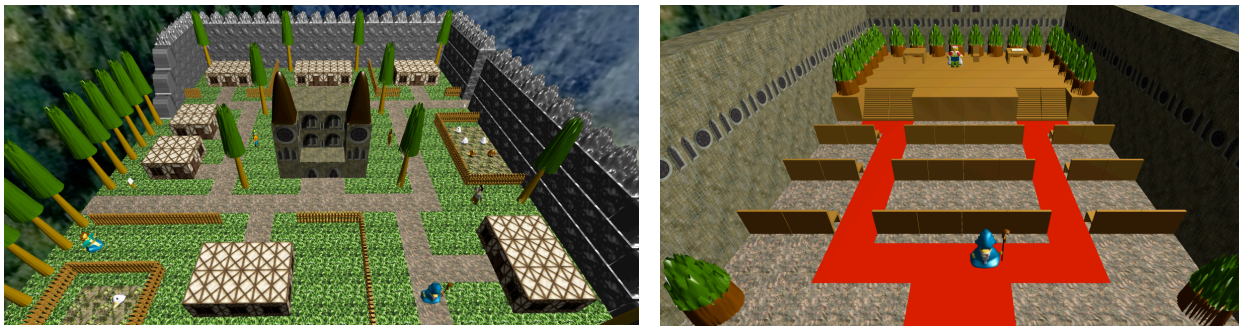


Figure 4: Worlds can be connected and hierarchical. The village (left) contains a number of buildings such as a church that the player can enter (right).

End-User Programming

AgentCubes is an authoring tool that integrates 3D modeling, world design and end-user programming (Lieberman et al., 2006) in a single project window (Figure 5). Visual AgenTalk 3D is the drag and drop visual programming language of AgentCubes, which is inspired by the original drag and drop programming language of AgentSheets (Alexander Repenning et al., 1996). This interface features a number of panels:

- ***Tool Bar*** (below window title bar). The tool bar is used to start and stop simulations, provide tools to edit worlds, and provide camera controls.
- ***Agent List*** (left). The Agent List is the collection of all the agents that are part of a simulation. Each agent can have a list of shapes to represent the state of an agent, e.g., a healthy versus a squished, dead looking frog in a Frogger-like game.
- ***World*** (top). The user interacts with one world at a time. The user can edit the world and save it by using tools from the tool bar. There are tools to control the camera and switch between first person and bird's eye view.
- ***Conditions Palette*** (bottom, left). The Conditions Palette contains the list of conditions which can be dragged into the IF part of rules.
- ***Actions Palette*** (bottom, right). The Actions Palette contains the list of actions which can be dragged into the THEN part of rules.
- ***Behavior Editor*** (bottom, center). The Behavior Editor, shown zoomed in, contains methods and rules. There is a behavior editor for each agent class.

The interface makes program composition quite simple by supporting the syntactic aspect of the programming challenges. At the same time, the panel structure also supports the coordination of panel content based on selection. For instance, when a user selects the bug (lower left) in the world, then its behavior is automatically shown. This not only handy allows quick access to the program of the selected agent, but also supports the semantic part of the answer to cognitive programming challenges.

The semantic aspect of the cognitive programming challenge consists of the discrepancy between the program a user wants and the one she/he has (Pea, 1983). Assume that at an earlier point in time the programmer introduced at least one bug into the program. The basic problem is with the meaning of the program, not its syntax. For instance, to test for a "game over" condition when the hero represented by the bug is next to a zombie, the programmer needs to use some kind of condition dealing with spatial adjacency. The nextTo condition can do the trick. However, it is possible to use the wrong mathematical comparator—for instance, one might use "<" instead of ">". Syntactically, this is not a bug. Both versions are well formed and could be executed by the computer. Therefore, static program analysis will not provide any useful feedback. Furthermore, the computer does not know which of the two versions of the program is the one desired by the programmer. Theoretically speaking, this problem can be reduced to the halting problem in computer science, which can only be uncovered by actually executing the program.

Conversational Programming (A. Repenning, 2011) is a brand new technique integrated into AgentCubes to provide semantic annotations to the programmer by constantly running the program created by the user one step into the future. In other words, the program is not just a visual representation, but could be considered to be alive in a sense similar to McDirmid's notion of live programming (McDirmid., 2007). For instance, in Figure 5 the programmer has selected the ladybug, which is the cursor key controlled main character of a game. Conversational Programming displays the program of the lady bug, runs the program in the context of the current game situation, and annotates the program to show the programmer possible futures. Notice that the programmer is not actually running the game. The game is stopped. However, the red annotated first rule indicates that that rule would not fire, and even indicates why. The nextTo > 0 target condition is not satisfied. That is, the game is not won. The green annotated second rule suggests that a certain sound would be played and the "You Lost!" message would be shown to the user because the lady bug is indeed nextTo > 0 zombies. The precise details of how Conversational Programing works are beyond the scope of this paper. However, the important insight here is that the program is executed in real time in the context of the current game situation, and feedback is provided as to what the program would or would not do if the user was actually running it. One way to conceptualize the benefit of this type of semantic support is to think about the time it can take between making a programming mistake and experiencing its negative manifestation. Images of punch card programming come to mind, but even in modern programming the delay between cause and effect can be highly detrimental—to the point where a novice programmer might get so frustrated and confused that they simply give up. This makes semantic support particularly important to educational programming environments.

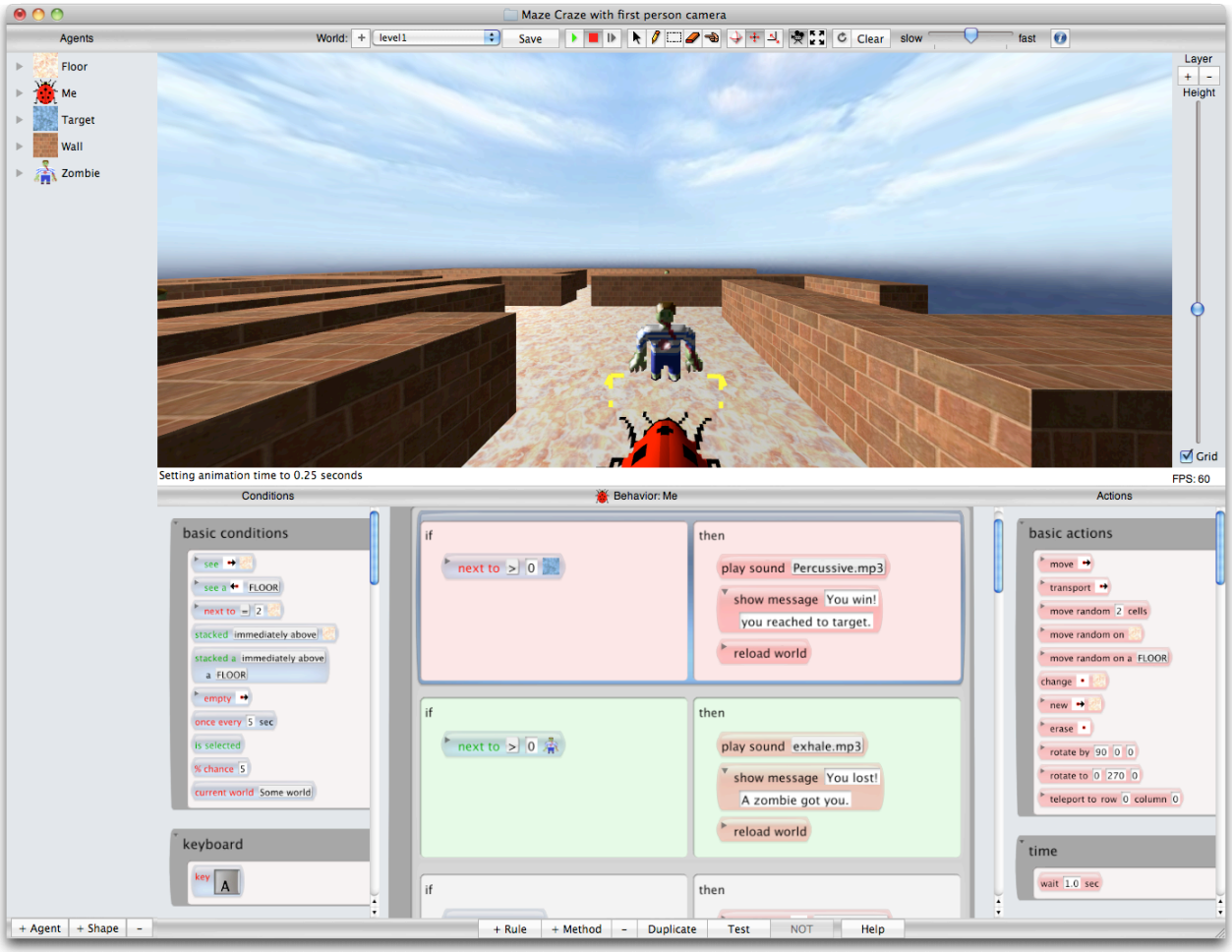


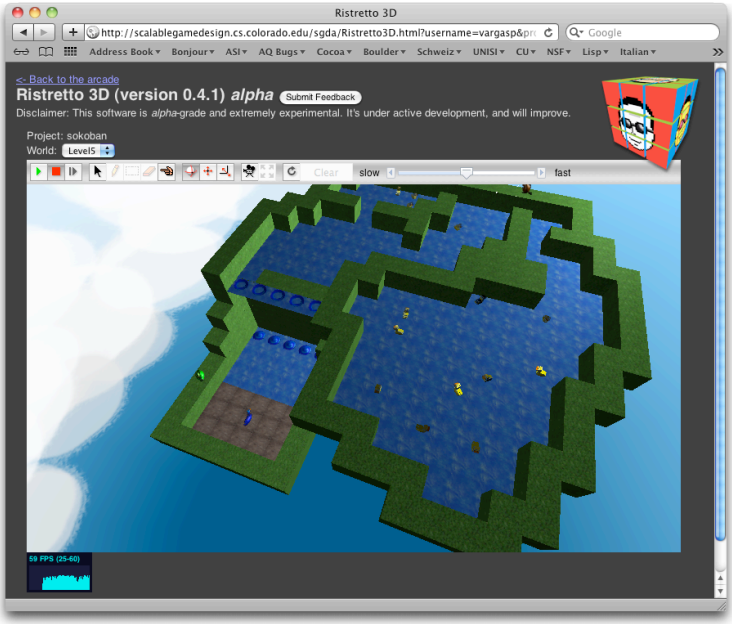
Figure 5: The AgentCubes project window. Users add agents (left) to the world (top) and program them by dragging conditions (bottom left) and actions (bottom right) into the behavior editor (bottom center).

None of the programming environments discussed in the related work section include this level of semantic support. Some, however, do have mechanisms that support program testing. In Alice and GameMaker, game testing is strictly moded. In programming mode the user creates a program, and in running mode the user tests the program. The significant overhead of switching between two modes can make it difficult to connect cause and effect. Scratch is a bit better, in the sense that it does allow a program to be changed while the program is running. However, there is no semantic feedback when the program is not running. Moreover, the lack of a class instance model means that each sprite has its own individual program, making it difficult to debug and change behavior in a set of objects. For instance, in our Maze Craze game (above) there are multiple zombies. Each one needs to have its own copy of the script. A problem found in the behavior of a particular zombie would require that the scripts of each of all the other zombies would have to be individually examined and fixed accordingly.

The feedback of Conversational Programming is not only immediate, but in some sense is even predictive. The moment a programmer drops a new condition into a rule, changes a rule parameter, or changes the situation in the world the program reacts by updating its annotations. For instance, if the user erases the zombie in Figure 5 then the second rule would immediately change to red. The predictive part of the Conversational Programming is less obvious but is due to the fact that even the conditions in the palette, which strictly speaking are not—yet—part of the program, are annotated semantically. Each condition in the palette is executed all the time in the context of the situation. This can help a programmer find a handy condition to define a complex situation, and suggests that Conversational Programming may help to avoid creating bugs in programs that have not even been written yet. Some call this general idea prebugging (Telles et al., 2001), in contrast to debugging.

Social Support

The affective challenge of programming goes beyond just making interesting games and simulations, and includes social aspects. With AgentSheets, we started to explore mechanisms of sharing early on (Alexander Repenning et



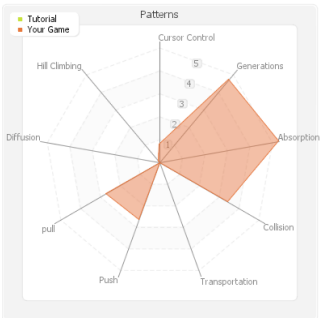
al., 1997). AgentSheets includes a technology called Ristretto, which compiles a game into a web page. Ristretto was one of the first Java bytecode compilers. It is quite important to many users, especially kids, to be able to share their creations with their friends. Unfortunately, new mobile devices such as the Apple iPhone or the iPad do not support Java technology or Flash. With Ristretto 3D for AgentCubes we have switched to HTML5 based technologies such as JavaScript and WebGL. This lets AgentCubes games run on fully HTML5 enabled browsers, including mobile ones, without Java, Flash or other plug ins.

Ristretto 3D is a project compiler generating HTML5 output. This allows users to play games in a browser and includes interaction and full camera control. For instance, a user can select any agent in a world and make it the first camera view agent. Our experience with Ristretto 3D is in performance it almost matches desktop 3D applications.

Evaluation

Can students build and debug 3D video games? Our main pedagogical framework is called *Zones of Proximal Flow* (A. Repenning, 2012), and combines Csíkszentmihályi's notion of Flow (Csikszentmihalyi, 1990) with Vygotsky's notion of the Zone of Proximal Development (Vygotskiĭ et al., 1978). The essence of Scalable Game Design is that programming challenges and skills should be balanced and that there are different paths, some better suited than others for broadening participation, along which students can advance their skills and tackle more advanced challenges. The idea of Gentle Slope 3D subscribes to the Zones of Proximal flow framework by exploring how to scaffold students in making basic 3D worlds and gradually creating more sophisticated ones. An early pilot study (A. Ioannidou et al., 2009), was done in two middle schools in Boulder (technology hub) and Aurora (urban) Colorado. This study demonstrated that all participating students were able to create and debug a working 3D game in less than 5 contact hours, all students were able to create sophisticated 3D models using Inflatable Icons technology, and that all students were able to add animations to their games and customize their animation parameters. Additionally, *most* students were able to program their character in both 1st person and bird's eye view. This challenging 3D programming task includes understanding and application of modulo arithmetic—a concept unfamiliar to most middle school students. Since 2009, additional scaffolding has been added to the curriculum for this first-person programming skill, which supports syntonicity in a video game. This early study also demonstrated that the 3D video game programming activity was motivational across the entire student population (girls, all ethnicities), and that student proficiency in the evaluative activities was not statistically significant over the student population.

Can we measure Computational Thinking Skills? A continuing problem in K-12 IT education has been the evaluation of anything but very simple, canned student programs for evidence of learning or transfer. When teachers themselves are not programmers, how can they evaluate the programming skill, or indeed the learning demonstrated by a given student through programming a game or simulation? AgentCubes attempts to solve this problem with the addition of Computational Thinking Pattern Analysis, or CPTA (Ashok Basawapatna, 2011; A Ioannidou et al., 2011), which originally was introduced through AgentSheets. CTPA employs a Latent Semantic Analysis inspired approach to locate code patterns that suggest the presence of phenomenalistic expressions such as collision, or diffusion. CTPA can find similarities between games suggested in tutorials and the ones actually



implemented by students. Moreover, CTPA can provide early indication of skill transfer between game design and STEM simulation building activities.

How Gentle is Gentle Slope 3D? One fundamental question concerns the price one pays for 3D creativity. We explored this question by employing CTPA to compare 180 games produced with AgentCubes (3D) in the 2012 Educational Game Design University of Colorado computer science course with 180 games produced with AgentSheets (2D) in the 2011 class. This course was structured to have each student build one complete game each week. In the first 4 weeks, each student had the goal of building four classical games (Frogger, Sokoban, Centipede, and The Sims). In the next 4 weeks, each student built four more games based on his or her own idea. Then, they built a final game based on the skills that they had learned. Using CTPA, we compared games and tracked individual as well as group learning trajectories by estimating their mastery of various computational thinking skills over time. The results showed slightly higher, but not statistically significant, gains in computational thinking skills for the 3D group over the 2D group. We interpret this result as positive, since while both groups were instructed to spend the same fixed amount of time to make their games, the games produced by the AgentCubes group were 3D and looked substantially more sophisticated. Given the roughly equivalent programming complexity, it would seem that Gentle Slope 3D did not add the level of overhead one would typically expect for a 3D authoring environment.

Conclusions

Cognitive and affective challenges of programming currently prevent a large adoption of computational thinking education in schools. AgentCubes is a new, creative 3D authoring tool that makes programming and 3D modeling more accessible to students and teachers. AgentCubes addresses cognitive challenges by providing end-user programming support that goes beyond the syntactic support of drag and drop programming languages. It also provides semantic support by indicating what programs would be doing as they are created, enabling the end user programmer to perceive the difference between the program they have and the program they want. AgentCubes addresses the affective challenge by letting end users create sophisticated 3D worlds, produce their own 3D shapes and share their creations as HTML5 Web applications. In educational settings, teachers find the ability to automatically analyze student-built games and simulations with respect to computational thinking patterns useful in gaging learning progress.

References

(PITAC), (2005). Report to the President: Computational Science: Ensuring America's Competitiveness.

Ashok Basawapatna, K. H. K., Alexander Repenning, David C. Webb, Krista Sekeres Marshall. (2011). *Recognizing Computational Thinking Patterns*. Paper presented at the The 42nd ACM Technical Symposium on Computer Science Education (SIGCSE).

Csikszentmihalyi, M. (1990). *Flow: The Psychology of Optimal Experience*. New York: Harper Collins Publishers.

Cypher, A. (1993). *Watch What I Do: Programming by Demonstration*. Cambridge, MA: MIT Press.

Dertouzos, M. e. a. (1992). ISAT Summer Study: Gentle Slope Systems: Making Computers Easier to Use. Presented at Woods Hole, MA, August 16.

Igarashi, T., Matsuoka, S., & Tanaka, H. (2006). *Teddy: a sketching interface for 3D freeform design.* Paper presented at the ACM SIGGRAPH 2006 Courses (SIGGRAPH '06).

Ioannidou, A., Bennett, V., Repenning, A., Koh, K., & Basawapatna, A. (2011, April 8-12). *Computational Thinking Patterns.* Paper presented at the Annual Meeting of the American Educational Research Association (AERA), symposium on "Merging Human Creativity and the Power of Technology: Computational Thinking in the K-12 Classroom", New Orleans.

Ioannidou, A., Repenning, A., & Webb, D. (2008). *Using Scalable Game Design to Promote 3D Fluency: Assessing the AgentCubes Incremental 3D End-User Development Framework.* Paper presented at the submitted to the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '08), Herrsching am Ammersee, Germany.

Ioannidou, A., Repenning, A., & Webb, D. (2009). AgentCubes: Incremental 3D End-User Development. *Journal of Visual Language and Computing, 20*(4), 236-251.

Lieberman, H. (2001). *Your Wish Is My Command: Programming by Example*. San Francisco, CA: Morgan Kaufmann Publishers.

Lieberman, H., Paternò, F., & Wulf, V. (Eds.). (2006). *End User Development* (Vol. 9): Springer.

Lin, H. S., Williams, E., & Bradley, S. (2010). *Computational Thinking for Everyone: Report of a Workshop on the Scope and Nature of Computational Thinking*: The National Academies.

Lister, R. (2011). Computing Education Research: Programming, syntax and cognitive load. *ACM Inroads, 2*(2).

McDirmid., S. (2007). *Living it up with a live programming language.* Paper presented at the Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications (OOPSLA '07).

Pea, R. D. (1983). *Chameleon in the Classroom: Developing Roles for Computers, Logo Programming and Problem Solving*. Paper presented at the American Educational Research Association Symposium (Montreal, Canada, April 1983).

Prensky, M. (2008). Programming: The New Literacy. *Edutopia (available at* **http://www.edutopia.org/programming-the-new-literacy)**, (January 13, 2008). Retrieved from **http://www.edutopia.org/programming-the-new-literacy**

Repenning, A. (1993). *Agentsheets: A Tool for Building Domain-Oriented Visual Programming Environments.* Paper presented at the INTERCHI '93, Conference on Human Factors in Computing Systems, Amsterdam, NL.

Repenning, A. (2005). Inflatable Icons: Diffusion-based Interactive Extrusion of 2D Images into 3D Models. *The Journal of Graphical Tools, 10*(1), 1-15.

Repenning, A. (2011, Sept. 18–22, 2011). *Making Programming more Conversational.* Paper presented at the in Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC '11, Pittsburgh, PA, USA.

Repenning, A. (2012). Programming goes back to schools: Broadening participation by integrating game design into middle school curricula. *To appear in the Journal of the Association for Computing Machinery*.

Repenning, A., & Ambach, J. (1996). *Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing.* Paper presented at the Proceedings of the 1996 IEEE Symposium of Visual Languages, Boulder, CO.

Repenning, A., & Ambach, J. (1997). *The Agentsheets Behavior Exchange: Supporting Social Behavior Processing.* Paper presented at the CHI 97, Conference on Human Factors in Computing Systems, Extended Abstracts, Atlanta, Georgia.

Repenning, A., & Ioannidou, A. (2006). *AgentCubes: Raising the Ceiling of End-User Development in Education through Incremental 3D.* Paper presented at the IEEE Symposium on Visual Languages and Human-Centric Computing 2006, Brighton, United Kingdom.

Repenning, A., Webb, D., & Ioannidou, A. (2010). *Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools.* Paper presented at the SIGCSE 2010, The 41st ACM Technical Symposium on Computer Science Education, Milwaukee, WI.

Telles, M., & Hsieh, Y. (2001). *The Science of Debugging*. Scottsdale: Coriolis Group Books, Scottsdale AZ, USA.

Vygotskiĭ, L. S., & Cole, M. (1978). *Mind in society: The development of higher psychological processes.* Cambridge: Harvard University Press.

Webb, D., Repenning, A., & Koh, K. (2012). *Toward an Emergent Theory of Broadening Participation in Computer Science Education.* Paper presented at the ACM Special Interest Group on Computer Science Education Conference, (SIGCSE 2012), February 29 - March 3, Raleigh, North Carolina, USA.

Acknowledgement

This work is supported by the National Science Foundation under grant numbers 0848962, 0833612, and 1138526. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.