



Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

AgentCubes: Incremental 3D end-user development

Andri Ioannidou^{a,*}, Alexander Repenning^{a,b}, David C. Webb^b^a AgentSheets, Inc., USA^b University of Colorado at Boulder, USA

ARTICLE INFO

Keywords:

Incremental 3D
Game design
Visual programming
End-user development
IT fluency
Computational thinking

ABSTRACT

3D game development can be an enticing way to attract K-12 students to computer science, but designing and programming 3D games is far from trivial. Students need to achieve a certain level of 3D fluency in modeling, animation, and programming to be able to create compelling 3D content. The combination of innovative end-user development tools and standards-based curriculum that promotes IT fluency by shifting the pedagogical focus from programming to design, can address motivational aspects without sacrificing principled educational goals. The AgentCubes 3D game-authoring environment raises the ceiling of end-user development without raising the threshold. Our formal user study shows that with Incremental 3D, the gradual approach to transition from 2D to 3D authoring, middle school students can build sophisticated 3D games including 3D models, animations, and programming.

© 2009 Elsevier Ltd. All rights reserved.

1. Introduction: why Incremental 3D?

Kindergarten to 12th grade (K-12) Information Technology (IT) education fails to attract the necessary number of students to Computer Science (CS) especially at the middle school level, when students make critical career decisions by judging their own aptitudes for math and science. Fueled by bad experiences with programming, middle school IT curricula have disintegrated into key-boarding, web browsing, word processing and PowerPoint workshops with little authentic enticement foreshadowing CS careers. This is a very serious problem because, despite the growing need for IT workers, the enrollment in undergraduate degree-granting CS programs in the US dropped by 70% between 2000 and 2005 [1].

The notion of IT fluency is slowly gaining momentum in education as a means to train and evaluate IT skills beyond just using applications. For instance, the National

Academy of Sciences' Fluency with Information Technology (FIT) framework [2] postulates a set of skills including meta-skills such as problem solving, creativity, working in groups, algorithmic thinking, and computational thinking [3]. Game design [4] and computational science [5] are gradually establishing themselves as application domains capable of balancing the educational and motivational concerns of IT fluency and attracting not only boys, but students underrepresented in CS such as girls and minorities. In fact, an independent study conducted by the Stanford School of Education using AgentSheets [6–9], our 2D authoring environment, suggested that girls and boys alike are interested in game design [10]. With the right combination of tools, curriculum and teacher training, game design can be employed effectively to teach IT to middle school students in a motivating way.

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 [11]. The focus of our research is to promote the notion of 3D fluency. People live in a 3D world; meanwhile, because of computer gaming, today's computers are highly capable of processing 3D information. Unfortunately, creating

* Corresponding author.

E-mail addresses: andri@agentsheets.com (A. Ioannidou),
alexander@agentsheets.com (A. Repenning), dcwebb@colorado.edu
(D.C. Webb).

computational 3D artifacts and games can be a truly daunting task. Even end users familiar with making 2D games are likely to find the transition to 3D to be difficult. A completely new set of tools is usually necessary to create 3D models that can be animated and programmed. For instance, there is very little skill transfer from 2D paint programs such as Photoshop to a 3D modeling editor such as Maya 3D. This raises the question: Is this discontinuity a conceptual consequence of 2D vs. 3D with potential roots in human cognition, or is it more of an accidental consequence of computational tools that have emerged disjointedly for 2D and 3D applications?

Our goal is to promote 3D fluency through a gradual approach that we call Incremental 3D. We reconceptualize the universe of 2D and 3D tools and skills as a continuum rather than a dichotomy. Most tools support either 2D or 3D authoring. For example, NetLogo [12] and Scratch [13] are 2D authoring environments aimed at K-12; BlueJ [14,15] and GreenFoot [16] are targeted for more advanced students, typically at the undergraduate level, and Macromedia Flash at professional designers. Alice [17], NetLogo 3D, StarLogo TNG [18], DarkBASIC [19], and Macromedia Director are 3D authoring environments with varying degrees of usability for different audiences. Some 2D tools are starting to integrate 3D authoring. However, some of them have a limited degree of integration with the 2D product (e.g. Swift3D is a separate component for Flash) or force the user to drop from a visual language level to a textual language with a 3D application programming interface (API) (e.g. GameMaker

[20]). AgentCubes, on the other hand, is a tool that supports 3D authoring through incremental approaches for all components of the 3D authoring process, namely modeling, animation, and programming. A gentle slope [21–23] approach allows end users to 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. Our hope is that this incremental process ultimately allows end users to make 3D applications just as easily as 2D applications by transferring existing skills.

This article assesses the idea of Incremental 3D as an approach for end users to create 3D games and acquire IT fluency in the process. The focus of the paper is not the technical implementation but to describe and evaluate the notion of Incremental 3D. A more detailed description of the AgentCubes architecture can be found elsewhere [30]. We first describe the components of Incremental 3D, namely incremental modeling, animation, and programming, in the context of AgentCubes, then outline the steps to transform a 2D into a 3D application, and report the findings from assessing 3D fluency in two schools.

2. AgentCubes: an Incremental 3D authoring environment

AgentCubes is a 3D rapid game-prototyping environment that enables even 10-year-old children to make simulations and games in just a few hours. While simple

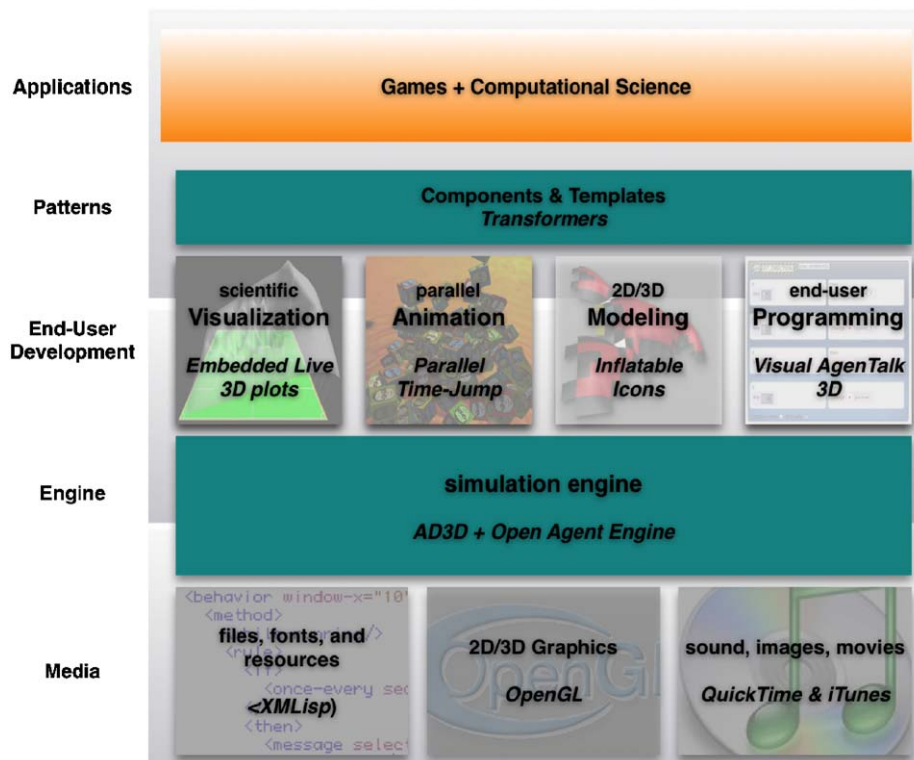


Fig. 1. The AgentCubes technical architecture.

compared with commercial games, these are complete, playable games. Versatility is an essential characteristic for systems to be used for Scalable Game Design [9]. They should enable students to easily create simple content, but also allow the creation of more sophisticated content. AgentSheets [7,9], our 2D simulation and game-authoring tool, has a low threshold and a relatively high ceiling, but AgentCubes raises the ceiling considerably while keeping the threshold low. Rich media such as audio, 2D images, and 3D models, a 3D environment with layers, and camera controls to switch perspectives (first-person vs. bird's eye view), and sophisticated user-controlled animations enable the creation of 3D games.

The AgentCubes architecture (Fig. 1) provides the following layers of functionality:

- **Application layer:** At the highest level, AgentCubes supports the creation of games and computational science applications that can be embedded in curriculum material to implement Scalable Game Design.
- **Pattern layer:** AgentCubes enables the customization of templates to instantiate model shapes (e.g. templates for inflatable icons) and patterns of behavior (e.g. templates for perspective-dependent programming).
- **End-user development layer:** End-user development [24] in AgentCubes is supported by Incremental 3D modeling, animation, programming, and visualization (for details, see Section 3).
- **Engine layer:** AD3D, the underlying simulation/game engine in AgentCubes, is built on top of our open-source Open Agent Engine. It provides the necessary APIs to the low-level functionality.
- **Media layer:** AgentCubes provides libraries and interfaces to low-level functionality such as OpenGL for 3D

graphics and QuickTime for media capabilities (sounds, 2D/3D images, models, movies). It also provides a unique interface to XML used for specifying resources and fonts.

AgentCubes is an agent-based framework. Agents are computational objects that can have autonomous behaviors [25] defined by end users. In AgentCubes, an agent has a visual manifestation on the screen called a shape. With this shape, the agent can represent real-world objects such as cars, people, and animals or more abstract entities such as ideas and numbers. In AgentCubes, agents are organized spatially in a three-dimensional space called the agentcube. An agentcube consists of layers. Each layer has a row-column grid similar to a spreadsheet. Each cell identified by a row, column, and layer can contain a stack of agents. In Fig. 2(2), stacks of agents organized in a layer of a cube are used to represent a city with agents such as cars, road pieces, and building components.

AgentCubes components (Fig. 2) include the following:

- (1) **Gallery:** The gallery (Fig. 2(1)) is the project inventory where end users create and manage agents. Through the gallery, users select agents and access their shapes and behaviors.
- (2) **World:** The world (Fig. 2(2)) contains all the agents organized in the agentcube. In the world, users add, select, delete, and copy agents. Using drag and drop, users move agents from one location to another in the same world or even into different worlds. The world toolbar includes tools for camera control (e.g. zoom, pan, and rotate), animation control (from running as fast as possible without any animation all the way to

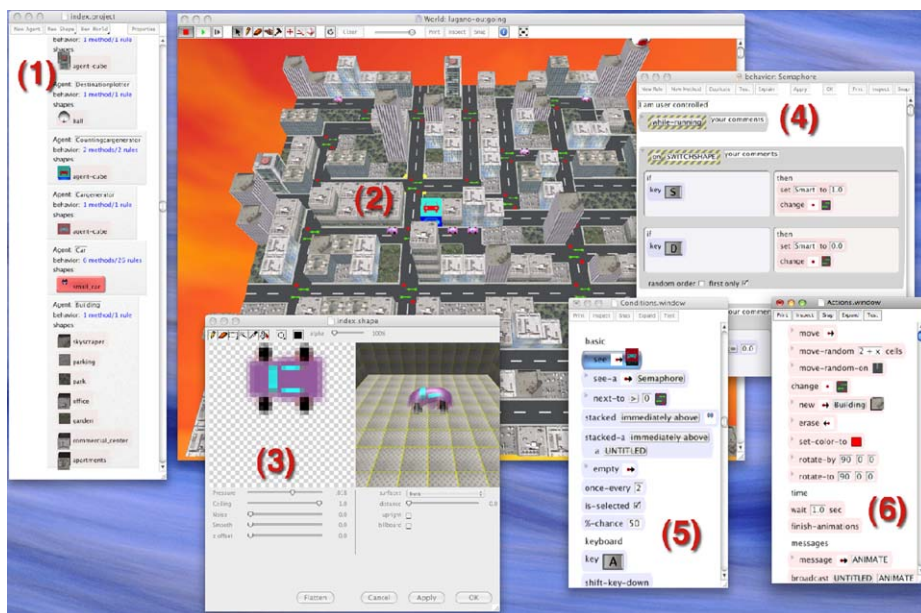


Fig. 2. A Traffic Simulation in AgentCubes: (1) the gallery where all the agents and their shapes are defined; (2) the world where the simulation or game unfolds; (3) an Inflatable Icons Editor for creating 3D objects from 2D images; (4) rule-based agent behavior defined in Visual AgentTalk 3D, using conditions (5), and actions (6).

animating all transitions that result from moving and rotating agents – details in Section 3.2), process control (run, stop, and step simulations) and screen control (window mode or full screen mode). Agents can be piled up in the world as stacks or can be positioned in suspended layers.

- (3) *Inflatable Icons Editor*: The Inflatable Icons Editor (Fig. 2(3)) allows end users to quickly draft 3D shapes by drawing 2D icons, which they render into an organic 3D shape through an inflation process [26]. The ability to quickly draft 3D shapes is an important part of the design process (Section 3.1).
- (4) *Behavior Editor*: Behavior Editors (Fig. 2(4)) are used to define, modify and test agent behaviors. Visual AgentTalk 3D (VAT3D) is a conceptual extension of Visual AgentTalk (VAT) [6,8] which is part of AgentSheets. VAT3D is a rule-based language based on conditions and actions which end users assemble through a drag and drop mechanism into complete behaviors. Groups of rules can be turned into methods with a name. These methods can then be invoked through actions sending messages spatially, e.g. send an “impact” message to the agent to the right of you, or via more general mechanisms such as broadcasting, e.g. send the “melt” message to all agents anywhere in the world of type “candle”. Like VAT in AgentSheets, VAT3D includes a number of helpful testing and debugging tools [8] including the ability to test if conditions are true with the currently selected agent or to run actions on the currently selected agent to see what they do (Section 3.3).
- (5) *Condition Palette*: Conditions (Fig. 2(5)) are language primitives used to test the environment and receive input from users. Basic conditions can check for agents next to the agent that executes them, deal with probability or with timers. Attribute conditions can check and compare the values of agent attributes and simulation properties. User input includes keyboard, mouse and game pads. Camera control conditions can determine if the world is in bird’s eye or first-person view and if there is a camera attached to a specific agent.
- (6) *Action Palette*: Actions (Fig. 2(6)) make agents do things. Basic actions include the ability to move and rotate. Message actions allow agents to send messages to other agents through space or by class association. Sound and speech actions allow agents to play sounds and speak synthesized text. Attribute actions enable agents to set attributes and simulation properties and to plot in 3D visualizations overlaid on the world.

Some of these components of AgentCubes are also discussed in subsequent sections in the context of incrementally building interactive 3D worlds (Section 4), and problem-solving situations (Section 5.1). While 3D authoring is far from a simple task, AgentCubes’ Incremental 3D approach is a scaffolding mechanism [27–29] that provides considerable support for modeling, animation, and programming.

3. Incremental 3D

Incremental 3D [30] is a design approach featured in AgentCubes for media-rich end-user development with low threshold, i.e. a low barrier of entry to create simple projects, and high ceiling, i.e. the ability to create highly sophisticated projects. The fundamental idea of Incremental 3D is that a user should be able to suspend important design decisions to the point in the design and development process when the decision really needs to be made. Many game and simulation applications can start as simple 2D applications that may be turned into 3D applications. Initially, the user should not have to worry about the precise look, size, orientation, and location of objects in 3D space or how objects need to be animated when they move. For instance, by utilizing grids we transition from dealing with Euclidian information (e.g. move my object 1.5m to the right), to topological information (e.g. move my object right to the next space).

The main aim of Incremental 3D is not just to address usability concerns, but also to support a gradual design and problem-solving process aligned with computational thinking [3]. Specifically this means that Incremental 3D must support a gradual formalization process. Problem descriptions may initially exist in textual form. Users can recognize objects through nouns and relationships between objects through verbs, in ways consistent with object-oriented design. To facilitate computational thinking, users should be able to gradually capture objects and their relationships. Initially, they may represent objects as highly abstract 2D blobs. As their understanding of the problem gradually increases, they should be able refine or change existing representations. This way, a game or a science simulation will gradually transition from an informal set of 2D blobs with no behavior, to a formal, fully working 3D application.

Our Incremental 3D approach no longer limits the scope of authoring to programming, but includes all aspects of development necessary to create 3D applications, namely modeling, animation, programming, and visualization.

3.1. Incremental modeling

Incremental 3D modeling is enabled through the Inflatable Icons technology [26]. Instead of limiting end users to using only stock 3D art, including licensed characters such as 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 fluency in modeling by creating their own 3D models. With Inflatable Icons, users draw 2D images and gradually turn them into 3D models using diffusion-based inflation techniques.

Early user-testing in local schools confirmed that students were able to make basic inflatable icons quickly, but needed additional means for producing more sophisticated 3D models, including benchmark shapes such as bugs and cars. Selection-based inflation is one such feature. We therefore created an Adobe

Photoshop-inspired set of tools that allows users to make and extend pixel selections. For instance, we included a magic wand tool to make selections based on pixel color values.

Say we want to create a frog. First we use the 2D editor with the symmetry mode enabled to sketch a frog (Fig. 3a). In the 3D view, the frog looks completely flat (Fig. 3b). Inflating the entire frog is a good start (Fig. 3c), but fails to highlight the strong legs of the frog. Using the magic wand, the frog legs get selected and inflated more (Fig. 3d).

3.2. Incremental animation

End users who program 3D worlds appear to have higher expectations for run-time behavior. For instance, if agents move or rotate, users would like to have at least the option to have the world change in an animated way. With

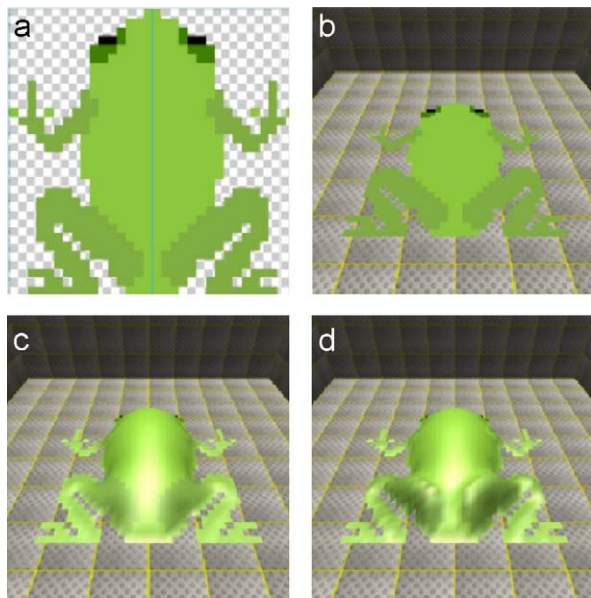


Fig. 3. A frog as an Incremental 3D shape.

no animation, the agents in Fig. 4a, which are programmed to simply move and rotate randomly, would instantly arrive at the next frame (Fig. 4c) without seeing any in-between frames. However, with animation, the agents move and rotate smoothly in a series of frames such as the one shown in Fig. 4b.

AgentCubes supports incremental animations. That is, initially users may not need or want to deal with animations. As they are getting ready, they can access animation parameters that are optional to language pieces such as move and rotate actions. Moreover, built-in scene awareness assisted by the notions of grids, stacks, and layers (e.g. built-in gravity) significantly scaffolds 3D animation authoring for users. Finally, the Parallel Time-Jump animation approach [30] allows any number of agents to animate in parallel without the need to track object locations and the overhead of sequential animation.

An important role of animation is to communicate complex relationships among objects. We have devised a novel animation approach that can be employed incrementally.

Facilitating the perception of causality through animation: With his work on the perception of causality, Michotte [31] showed that humans perceive causality between objects depending on the exact timing of movements. To be able to achieve the desired effect in the Michottian sense, AgentCubes includes a number of mechanisms to enable and control animations. Users can adjust the time, the trajectory, and the acceleration of an animation.

Separation of logic and animation: An important aspect of Incremental 3D is that logic and animation are kept

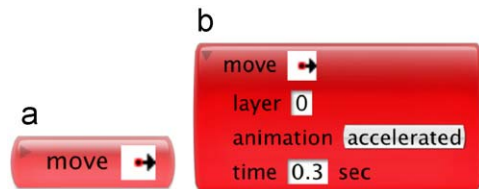


Fig. 5. Separate logic from animation: (a) move right action; (b) disclosed version showing additional parameters relevant to animation.



Fig. 4. (a) Cube agents programmed to move up to 4 cells and rotate randomly. (b) A snapshot of a frame in the middle of animation, showing the agents moving and rotating. (c) Agents arrive at their final positions at the end of the cycle. Without animation, the viewer would only see the first and last frame without the intermediate animation frames.

separate. The logic part describes what the agent will do. For instance, in a Frogger game, the cursor-controlled frog will move one grid space to the right. The user will simply use the Move <right> action to achieve this (Fig. 5a). Later in the development process, the user may want to add animation information by using an accelerated animation in which the agent continuously accelerates and at the mid-point starts to decelerate until it comes to a complete stop (Fig. 5b).

Scene awareness: Animations quickly become complex for users if they have no physical awareness of a scene. The Move action hides enormous complexity, as it includes automatic interpretations of the world. In 2D environments, a move will simply remove an agent from one location in the grid and add it to a new one. It should not be any harder for a user to do this in 3D, but the system has to interpret a move in 3D space. An agent moving from one stack to another will automatically move on top of the new stack, using a trajectory of automatically generated x , y , z animation components to avoid object intersections. If an agent moves out of the middle of a stack, then the stack will be compacted. Consistent with a world with gravity, all the agents above that agent will drop.

Parallel Time-Jump: AgentCubes uses the novel Parallel Time-Jump animation approach to allow any number of agents to animate in parallel. Even in a simple simulation in which agents are moving around randomly, agents moving to the same stack in the same layer will have to pile up (Fig. 6). This would not be a problem if animation was handled sequentially, with the first agent moving to the stack and then the second agent moving on top of it. The total time it takes to transition an agent world from one step to the next will be the product of the animation time and the number of agents. While this would work with a small number of agents, animating, for example, 1000 agents with 0.3 s per animation would total in a seemingly never-ending 5-min animation. In such a case, animation should be done in parallel. But if animations need to be done in parallel, we can know where the agents are moving only once all the agents got dispatched and moved to their final destinations. Parallel Time-Jump [30]

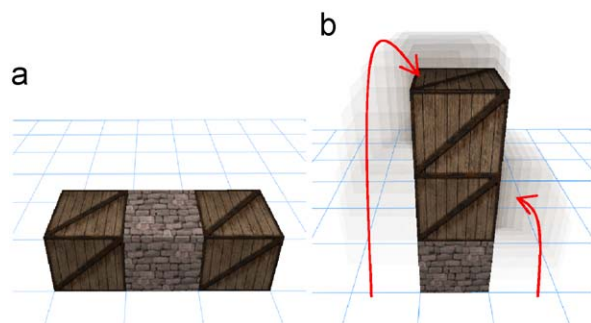


Fig. 6. (a) Two crate agents (left, right) both want to move on top of the brick agent. (b) Right crate gets dispatched first, but both crates know where they need to move to. Both crates move in parallel to their respective destinations. The animation makes the left crate overshoot vertically to avoid intersection.

deals with this by moving forward and backward in time. Conceptually speaking, the Parallel Time-Jump will first dispatch, move and rotate all agents without animation and without displaying the changes on the screen. Then it leaps back in time and generates all the transitional animations from where the agents currently are to where they should end up. This way, 1000 agents will only take 0.3 s to be animated in parallel.

3.3. Incremental programming

To support 3D fluency, we needed a programming language that would allow students to create behavior in 3D. Our conceptual starting point was our previous work with the Visual AgentTalk programming language in AgentSheets [6]. VAT had established the usability of the rule-based approach for authoring 2D games and simulations and computational science applications in school settings [32]. For AgentCubes, we enhanced the language to include the notion of Incremental 3D, leading to Visual AgentTalk 3D, which includes the ability to author and run 2D projects and gradually add control over 3D aspects. VAT 3D has the following characteristics:

- **3D grid:** Worlds in AgentCubes consist of layers with stacks of agents. VAT 3D features conditions and actions that orient and move agents in 3D, providing incremental support through optional parameters.
- **Camera control:** Attaching cameras to agents (first-person view) makes the agent the location of the camera. If the agent moves, the camera will move too. If this agent turns, the camera will turn too. This seemingly simple extension resulted in a number of cognitively interesting challenges, including the need to have conditions to test if the simulation is currently running in bird's eye or first-person view.
- **Lighting control:** End-user support for the use of light sources in sophisticated scene rendering.
- **Formula language:** The formula language allows users to express equations as functions of agent attributes using special notation to access agents via their grid locations in relative and absolute terms similar to spreadsheets. For instance, the expression “weight+weight[left]” adds the value of the agent's attribute called “weight” with the value of the “weight” attribute of the agent to the left. Unlike AgentSheets, which features a 2D spatial structure and operators to express computation in 2D, AgentCubes allows users to express computation in 3D.
- **Animation support:** Optional animation parameters in movement and orientation language constructs (i.e. conditions and actions) enable the separation of logic and animation in agent behavior, thus ensuring that the logic part works without obliging the user to first define animation.

Examples of programming in AgentCubes are given in subsequent sections.

3.4. Incremental visualization

AgentCubes includes some sophisticated 3D visualization capability. Embedded live 3D plots are rendered as an overlay on game and simulation worlds. For example, Figs. 7 and 8 show visualizations of the collaborative diffusion [33,34] algorithm used for AI path-finding in different games. The visualization is a logarithmic 3D surface plot of the collaborative diffusion values. Collaborative diffusion spreads target values through space in a way allowing agents to find target efficiently. The targets manifest themselves as peaks in the surface plot. In Pacman (Fig. 7), the user-controlled Pacman character is the target, creating the peak in the upper left corner, attacked by ghosts. Walls in the Pacman world stop diffusion values and create a complex diffusion landscape indicating where ghosts need to go to track down the Pacman. In soccer (Fig. 8), the ball is the target of the soccer players. The presence of other players modulates the collaborative diffusion values in a way to allow collaborative interactions between players from the same team.

Our early experience with these kinds of visualizations is that they can be essential in explaining complex

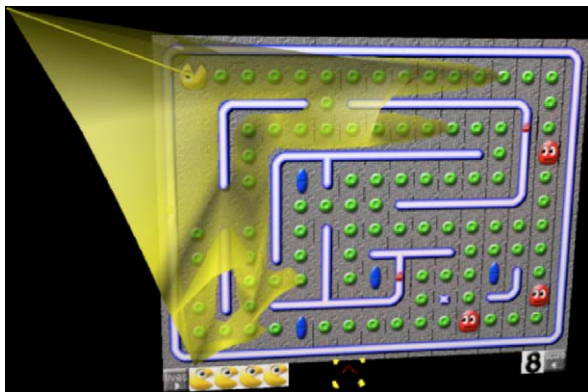


Fig. 7. Collaborative Pacman Game. The user-controlled Pacman is in the upper left corner. The Pacman “scent” is diffused over the entire worksheet. Diffusion values are plotted logarithmically. The plot is intersected by the worksheet. Walls are obstacle agents with a zero diffusion value.

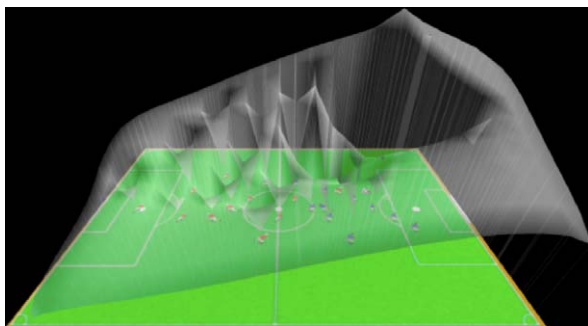


Fig. 8. Ball diffusion in a soccer game simulation. Peak indicates the location of the ball.

mathematical relationships relevant to numerous Science, Technology, Engineering and Mathematics (STEM) topics. One of the unique strengths of our AgentSheets 2D authoring tool is that it can be used for game design as well as for computational science [5] applications, which can provide students with IT fluency leading to scientific careers outside CS.

4. Incremental 3D process in game design

Student progression to 3D fluency is established by having a process that is gradual enough to keep students in the optimal flow of learning [35]. The process of creating a 3D game starting with a 2D game involves four successive steps: (1) creating a 2D game; (2) creating a first-person 2D game; (3) creating a first-person 3D Game; and (4) constructing a 3D world. These steps are described below:

- (1) *Creating a 2D game:* Students are guided through a game design process we call Gamelet Design to create an initial 2D version of a game. We typically use the classic arcade game of Frogger (<http://en.wikipedia.org/wiki/Frogger>) because even young children are aware of it and it seems to be gender neutral. The result is a simple, but completely playable version of the first level of the Frogger game. In this version, a cursor-controlled frog tries to cross a highway with cars driving across. Cars get automatically generated and absorbed at the beginning and end of the highway, respectively. Finally, the game deals with the car–frog collision that results in the frog perishing and being generated again, if there are any lives left. The 2D version of the game (Fig. 9a) does not include custom animations or 3D models at this point.
- (2) *Creating a first-person 2D game:* Using incremental modeling, animation, and programming, the look and basic behavior of the 2D Frogger game gets transformed to 3D. We motivate the transition from 2D to 3D by attaching the camera to the user-controlled character, namely the frog, and therefore changing perspectives from a world-view where the user looks at the game world from a bird’s eye view to a first-person view where the user sees the game world through the “eyes” of the frog (Fig. 9b). After the initial “the world is flat” shock, students typically want to create 3D looking objects. Inflatable Icons are used for incremental modeling to create 3D game objects from the 2D images that the students had created during the previous step (Fig. 9c). Seeing the game run and the jerky movement of the cars prompts students to change the animation parameters for the movement. To make the games seem more realistic, AgentCubes supports different animation modes (constant vs. accelerated). For cars, for instance, it makes sense to have constant animation speed, whereas for the frog it is better to have accelerated animation to simulate jumping. Moreover, simple behavior changes are incrementally implemented. With the camera attached to the frog, the students see the need to

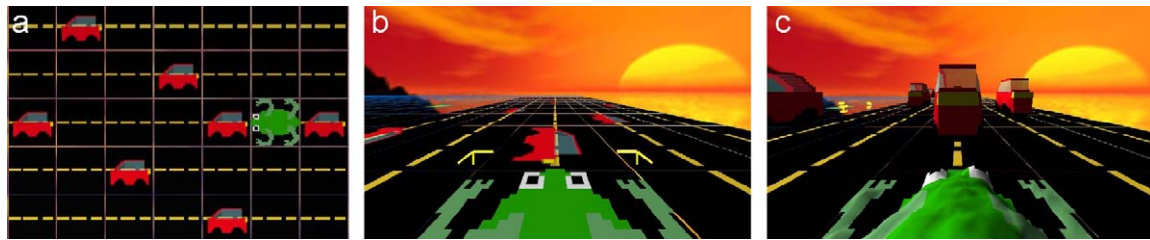


Fig. 9. (a) Bird's eye view of Frogger; (b) flat frog in first person looking at flat cars; (c) 3D frog looking at 3D cars.

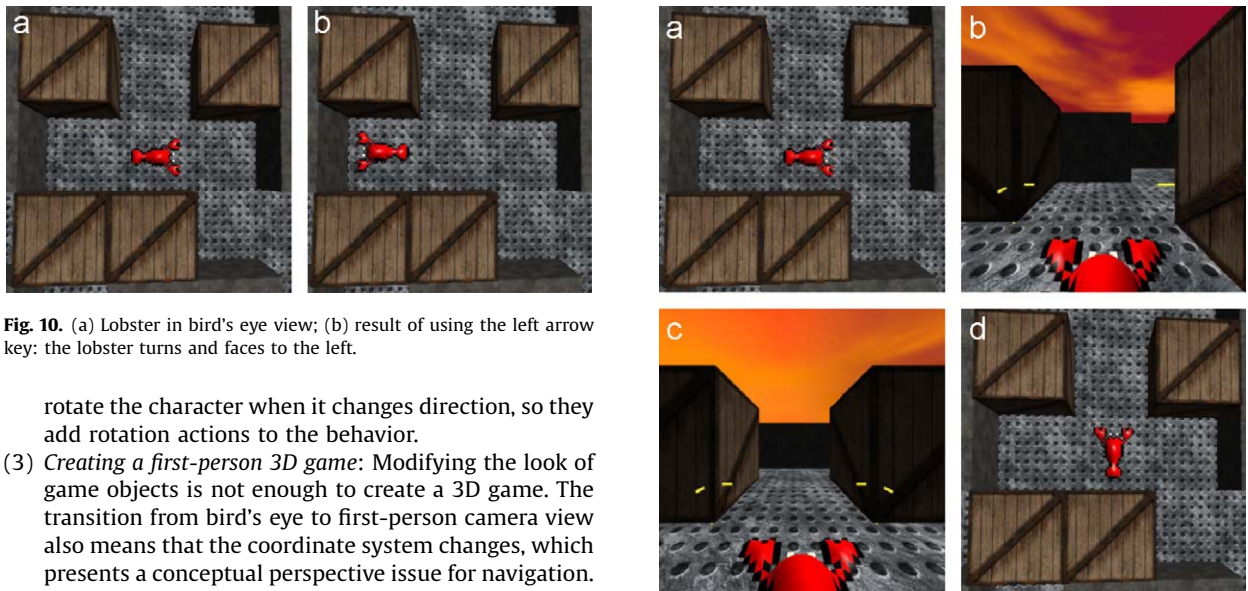


Fig. 10. (a) Lobster in bird's eye view; (b) result of using the left arrow key: the lobster turns and faces to the left.

rotate the character when it changes direction, so they add rotation actions to the behavior.





- (3) *Creating a first-person 3D game:* Modifying the look of game objects is not enough to create a 3D game. The transition from bird's eye to first-person camera view also means that the coordinate system changes, which presents a conceptual perspective issue for navigation. The “absolute” right, left, up, down directions that make sense when looking at the world from a bird's eye view no longer make sense in first-person mode (Figs. 10 and 11). Students expect the user-controlled character to transition seamlessly from absolute to relative coordinates (Table 1). Instead, they need to implement additional navigation behavior to deal with the relative coordinate system.

With an incremental behavior approach, students are taught how to implement world-view vs. first-person navigation, extending existing code with language able to deal with different versions of character navigation based on the camera position. This is a fairly difficult concept that requires more than trivial programming, but at the same time presents great opportunities for learning about coordinate systems and modulo arithmetic – a concept not covered in the middle school math curriculum. Game design provides many such opportunities for learning complex concepts on demand, rendering it an experience that synthesizes many different STEM skills, not just programming. Indicative of this was a quote from the only student who indicated he knew about modulo arithmetic in our experiment: “I knew about modulo arithmetic, I understood it, but now I know how to apply it.”

- (4) *Constructing a 3D world:* At this point, students have a simple but complete 3D game. As a final step, we

Fig. 11. (a) Lobster in bird's eye view; (b) lobster in first-person view; (c) result of using the left arrow key: the lobster turns to its left; (d) result viewed from birds' eye view: lobster is actually facing up in the absolute coordinate system.

Table 1
Perspective-dependent interpretation of user input.

Bird's eye perspective	Cursor control	1st person perspective
Move left		Turn left 90°
Move right		Turn right 90°
Move up		Move forward in direction
Move down		Move backward in direction

introduce students to a truly 3D world. Not only are the objects of the game 3D, but there is movement in all three dimensions using layers in the 3D grid. This 3D environment enables students to first navigate a ready-made 3D maze and then construct their own mazes by directing the movement and rotation of a spaceship drilling holes in a solid cube. Indicators of

3D fluency in this activity are specific design aspects of the mazes students create (e.g. toggling between bird's eye and first-person views, toggling between visible and invisible walls to evaluate the maze structure, rotating the world to view the possible routes in the maze) and the use of orientation and visualization tools to verify that the maze satisfies the given design criteria.

5. Impact of AgentCubes on IT fluency

We formally evaluated the effectiveness of the Incremental 3D approach as a way to achieve 3D and IT fluency at the middle school level. In this section, the design of the study, the requirements of the troubleshooting scenarios used as a culminating activity, the context of the study and the findings are reported.

5.1. Study design

The evaluation study was designed in collaboration with educational researchers from the University of Colorado's School of Education, who have experience in working with students in technology-intensive instructional environments, as well as expertise in conducting classroom-based research in K-12 settings. The study was designed to document the impact of student use of AgentCubes on identifiable learning objectives with respect to the development of student IT and 3D fluencies, mainly following the Fluency with Information Technology framework. Given the scope of the feasibility study, we focused on a subset of FIT and 3D fluency elements that included *IT Skills* such as using a graphics package to create illustrations, *IT Concepts* such as algorithmic thinking and programming, and *Intellectual Capabilities*

such as managing complexity, engaging in sustained reasoning, and managing problems in faulty situations [2].

Instruction followed the Incremental 3D steps mentioned above. In addition to formative evaluations during instruction, as an activity to measure fluency during the final session, we designed problem-solving situations in which students were asked to troubleshoot programming scenarios. Instead of traditional pre- and post-tests, we opted to perform an authentic assessment [36,37] that would require students to draw upon what they had learned about game design and programming agent behavior to identify and solve problems in troubleshooting scenarios that involved an intentionally defective version of a 3D Frogger game. Within a 45-min period in the fifth (and last) session, students had to figure out at least five things that were wrong with the game and re-program the agents' behaviors to fix those problems. These included issues with movement in world and first-person views, missing behaviors, and defective generation rates.

Specifically, the students had to solve the following:

- (1) *Car movement bug*: One type of car was not moving on the highway from left to right, as it should. Its behavior was missing the rule that specified that the car should move to the right if there is highway; as a result, the car remained stationary. Students had to identify the correct method (Fig. 12) where the movement rule (Fig. 13) was to be inserted and add it.
- (2) *Car generation bug*: On one side of the highway, the cars generated to move from left to right were stacking up (Fig. 14a and b). The behavior of the car generator was set up so that it was creating cars too often and without checking whether there was an empty piece of highway there first (Fig. 16). Detecting these kinds of issues in 3D is much easier than in the equivalent 2D environment. Because of the third dimension, the piling cars are discernible immediately (Fig. 14a) without even tilting the 3D world (Fig. 14b). The 2D equivalent of the situation cannot be discerned just by looking at the world. Multiple car agents can be stacked on top of each other, but one cannot tell just by looking at it (Fig. 14c). Situations like this can lead to performance degradation of the system, since one can end up with thousands of agents piled up using up system resources without the user knowing why, making debugging of such issues extremely difficult, especially for novices. To fix the faulty behavior (Fig. 15), students had to slow down the generation rate and put a check to see if there is an empty piece of highway before creating a new car (Fig. 16).

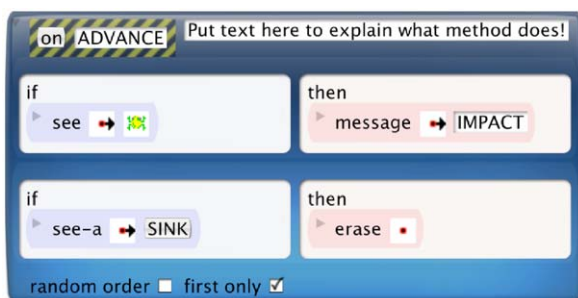


Fig. 12. Faulty behavior for the car that causes the car to be stationary. To fix the problem, the missing rule for actually moving the car (shown in Fig. 13) should be inserted in the "Advance" method.

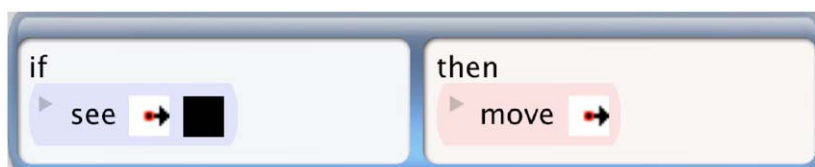


Fig. 13. Missing rule for the car to be added to the behavior in Fig. 12 to make it move right, but only if there is road ahead of it.

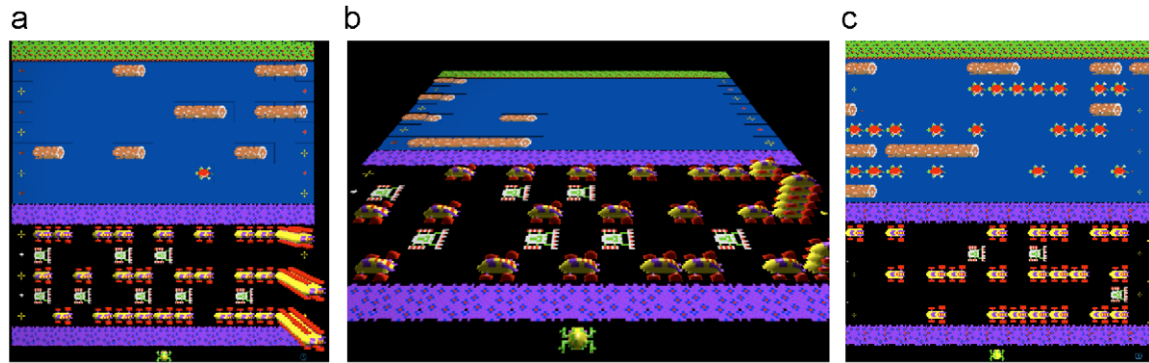


Fig. 14. Identifying the car generation issue with cars piling up: (a) looking at the 3D world top-down you can still see the piling cars; (b) looking at the 3D world from a different perspective; and (c) in the 2D equivalent, the problem is not apparent at all.

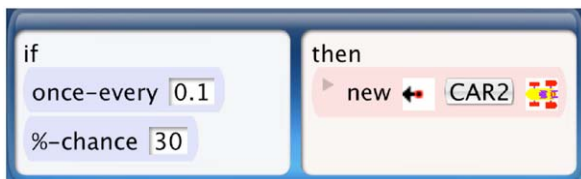


Fig. 15. Faulty behavior for the Car Generator agent. New car creation gets called too often and with high probability. There is no check for an empty spot to create the car either.

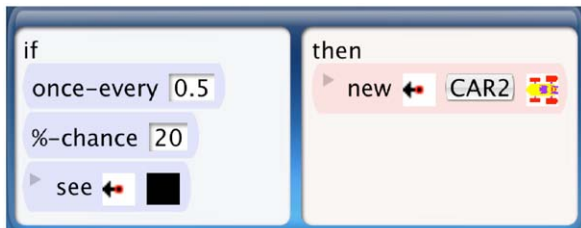


Fig. 16. Fixed behavior for the piling cars issue.



Fig. 17. Faulty behavior of 2D navigation bug. When the right and left arrow keys are pressed the frog does not move in correct direction. Notice in 2nd and 4th rule the opposite arrows checked in the key condition and the arrows of movement in the move action.

- (3) *2D navigation bug*: The movement for the Frog in bird's eye view perspective was incorrect. The rule for when the left and right arrow key was pressed did not match the direction in which the frog should move (second and fourth rule in Fig. 17). Students had to locate the relevant method in the behavior and change the movement direction to match the key pressed.
- (4) *3D navigation bug*: The movement for the frog in first-person perspective was incorrect. In the move-in-direction method (Fig. 19) called from the navigate-first-person method (Fig. 18), there was a duplicate condition for dealing with direction = 0. That essentially means that the frog could never turn left in first-person mode. The condition of the second rule in the move-in-direction method needed to be changed from testing for direction = 0 to direction = 1. Students had to locate the relevant method in the behavior and fix the error (Fig. 19).

- (5) *Turtle generation bug*: There were not enough turtles being generated for the frog to make a successful crossing of the river (shown in Fig. 14a and b). This turned out to be an elusive problem for the students to identify, as it was a usability issue, not a programming issue, per se. To fix it, the turtle generation should have been increased by lowering the frequency with which the generation rule was checked (once-every condition) and possibly increasing the percentage of generation (Fig. 20).

These troubleshooting tasks were unfamiliar situations to students and were not discussed in previous sessions. Students were required to complete the activity on their own and could only ask the instructors questions of

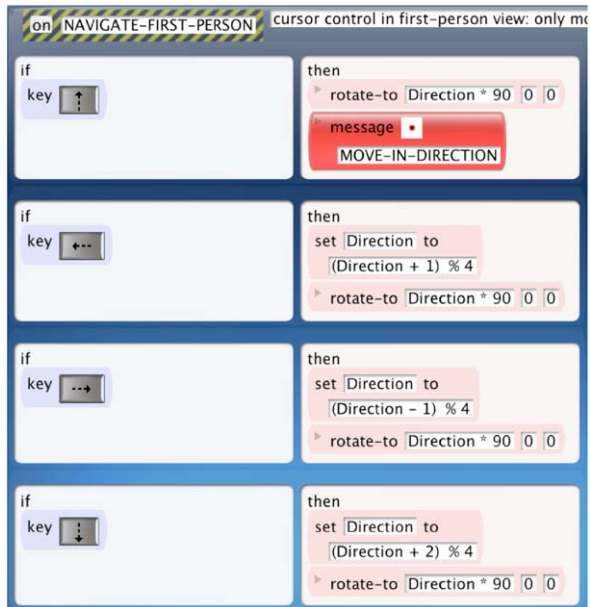


Fig. 18. Method implementing first-person navigation. This part of the behavior is correct, but calls the “Move-in-Direction” method (Fig. 19) that contains the faulty behavior.

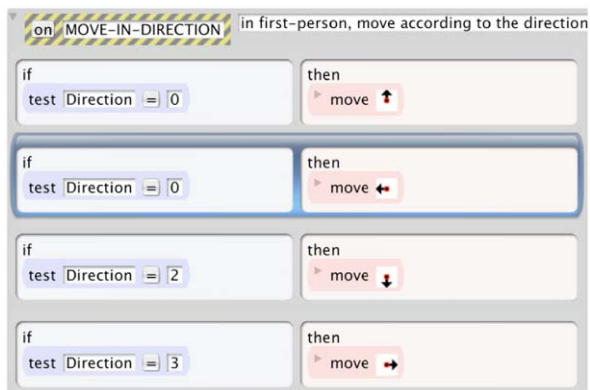


Fig. 19. Faulty behavior for first-person navigation. Notice that there is condition to correctly handle movement to the left (direction = 1), but instead a duplicate rule for checking for direction = 0. Therefore, the frog can never turn to the left in first-person perspective.

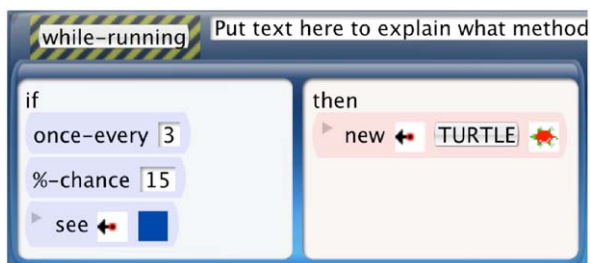


Fig. 20. Faulty turtle generation behavior. Generation frequency and chance are too low for the game to be winnable, as there are not enough turtles for the frog to jump onto to cross the river.

clarification. The debugging scenarios were challenging since students were neither told what the problems were nor how to locate the problematic procedures within the AgentCubes environment. They needed to identify the problem, locate the problematic agent and its behavior, locate the exact problematic procedure in the code, and correct the program for the agent.

We recognized that offering students an opportunity to engage in troubleshooting was an authentic experience familiar to any computer programmer. It required managing problems in faulty situations in addition to sustained engagement in reasoning and application of programming skills. Our eagerness with presenting such tasks to students was tempered by uncertainty regarding students' ability to identify the problems, students' insight in locating the problematic procedures for a given agent, and knowing how to resolve the problems. However, using the troubleshooting assessment to gather evidence of student FITness was rewarded by the intensity of student engagement throughout the assessment and what students were able to accomplish, which is discussed in the findings section.

5.2. Contexts

The evaluation study was administered in collaboration with Science Discovery, the University of Colorado's science outreach program, and was conducted in the context of four after-school classes in two middle schools, one in Boulder and one in Aurora, Colorado. Forty students attended the initial session. The race and ethnic background of students recruited for the AgentCubes course was a close approximation to the background of students found at the participating schools, with the majority of participants at the Boulder school reporting a Caucasian background and the majority of participants at the Aurora school reporting a Hispanic background (Table 2). Participation was voluntary. A large number of students were recruited by researchers and teachers. School administration and teachers reduced the recruitment group down to the 40 students we could accommodate in the experiment. The requirements included having two groups of all-female students and a participant sample that represented the school population. It is also interesting to note

Table 2

Study participants from Aurora (top) and Boulder (bottom) schools.

	Male	Female	Total	AgentCubes (%)	School (%)
African-Am	4	2	6	30	17
Asian-Am	0	1	1	5	3
Caucasian	1	0	1	5	11
Hispanic	4	6	10	50	68
Multi-Eth	0	1	1	5	nr
Native-Am	1	0	1	5	1
	10	10	20		
African-Am	1	0	1	5	1
Asian-Am	0	0	0	0	4
Caucasian	7	8	15	75	84
Hispanic	0	1	1	5	11
Multi-Eth	1	2	3	15	nr
Native-Am	0	0	0	0	<1
	9	11	20		

that the Boulder school is located in a technology hub region whereas the Aurora school is located in a less affluent, blue-collar area.

5.3. Findings

The findings resulting from the overall evaluation study are grouped in three categories: technology; curriculum; and broadening participation.

5.3.1. Technology

For the technology category, the criterion to measure success was whether students could build a simple game from scratch, including 3D models and behavior programming in a short period of time (less than 5 h). The technology findings (TF) were as follows:

TF1 – All students were able to create a working 3D game in less than 5 h: All students made at least one game. Several students went beyond what was expected in class and created extra games. It is interesting to note that it was mostly boys from the Aurora school who created the extra games.

TF2 – All students were able to create sophisticated 3D models from scratch using Inflatable Icons: The Inflatable Icons technology turned out to be highly accessible to all students. Inflatable Icons were able to cover the spectrum from rough and ready abstract looking 3D model drafts all the way to sophisticated 3D models. It is interesting to note that, on average, girls spent more time and paid more attention to detail in creating their 3D models than the boys.

TF3 – All students were able to add animations to their games incrementally and customize animation parameters: Students managed to enable and disable animations as well as customize them. Customization allowed students to control the animation timing and acceleration parameters. The incremental nature of the animation approach built into AgentCubes allowed students first to build a game and then, when necessary, add the animations after they had developed the main game mechanics.

TF4 – Most students (85%) were able to program their own character control in 1st person and bird's eye view successfully: This was a very challenging task: it included understanding and application of modulo arithmetic, a concept unfamiliar to most middle school students. Even so, students were able to follow instruction and 85% of them were able to complete the implementation of the first-person navigation. Also, 75% of them were able to fix the intentionally defective version of first-person navigation in the unassisted troubleshooting session.

5.3.2. Curriculum

The criterion to evaluate curriculum was based on achievements towards FITness goals. During the sessions, an AgentCubes FITness Observation Protocol (AFOP) was used along with a pre-assessment observation checklist to document students' opportunities to engage in activities that had the potential to promote Fluency in Information Technology. During the final session, an additional observation checklist was used to document students' problem solving and design of 3D mazes in AgentCubes, and students' ability to solve various troubleshooting scenarios using AgentCubes.

The curriculum findings (CF) were as follows:

CF1: Most students (75%) were able to solve most issues (60% or more) in the troubleshooting activity. Almost all students demonstrated sustained engagement and persistence in resolving these problems. All students were able to identify at least three of the problems and attempted to resolve the problem by reprogramming agent behavior. As a matter of fact, 75% of students solved the majority of the issues (3 or more).

Table 3 summarizes the percentages of students able to troubleshoot each scenario. Out of the 40 original participants, 24 students participated on the day the troubleshooting activity took place. In addition to overall results, data are disaggregated by school, gender, and ethnicity. It is worth noting that female students and students at the Boulder School were more successful in resolving car movement and generation issues. Male

Table 3

Percentage of students identifying and completing the five troubleshooting tasks discussed in Section 5.1: (1) cars not moving; (2) cars piling up; (3) frog 2D movement; (4) frog 3D movement; and (5) turtle generation.

Groups	Troubleshooting tasks						
	N	Cars not moving (%)	Cars piling up (%)	Frog movement (2D) (%)	Frog movement (3D) (%)	Turtle generation (%)	Average (%)
All students	24	67	88	79	75	42	70
Schools							
Boulder	14	71	93	64	64	50	69
Aurora	10	60	80	100	90	30	72
Gender							
Male	16	63	81	88	88	50	74
Female	8	75	100	63	50	25	63
Ethnicity							
Caucasian	13	69	92	69	62	46	68
Hispanic	5	60	80	100	100	20	72
Afr-Am	3	67	100	100	100	33	80
Other	3	67	67	67	67	67	67

Table 4

Intensity level of opportunities for student development of FITness.

Fluency with Information Technology Goals	Session 1	Session 2	Session 3	Session 4	Session 5	Average
Using a graphics package to create illustrations	4	5	4	2	2	3.4
Algorithmic thinking and programming	2	3	5	5	5	4
Managing complexity	1	2	3	3	4	2.6
Engaging in sustained reasoning	5	4	3	4	5	4.2
Managing problems in faulty situations	0	2	2	2	5	2.75

Intensity Level (Key for Table 4)

5	primary focus with opportunity for student unassisted use
4	secondary focus with opportunity for student unassisted use
3	primary focus but student actions heavily guided/instructed
2	opportunity for only a few students
1	potential opportunity, but not observed in students
0	No opportunity

students and students at the Aurora school were more successful in resolving the scenarios related to frog movement.

Furthermore, 25% of the students went beyond the scope of the activity and improved the program in other ways, such as using the graphics tools to change or inflate game components such as cars and turtles so they would be easier to see in first-person view.

CF2 – Scalable Game Design is a feasible strategy to create a FIT-oriented curriculum using AgentCubes: Data from the AFOP were analyzed to reveal opportunities to address the five elements of the FIT framework [2] that were prioritized for observation and assessment:

- Using a graphics package to create illustrations.
- Algorithmic thinking and programming.
- Managing complexity.
- Engaging in sustained reasoning.
- Managing problems in faulty situations.

A hierarchical rating scheme was developed to distinguish potential opportunities from observed opportunities with and without guidance. As summarized in Table 4, every session included opportunities to address multiple goals, but what distinguished the latter sessions from the earlier ones were the opportunities for students to demonstrate their achievement of FIT goals apart from instruction. Since the last session included several assessment-like activities, there were several opportunities for students to demonstrate their ability to engage

in sustained reasoning, troubleshoot errant programming, and manage faulty situations (which, in fact, the majority of students demonstrated, as illustrated in the results for the troubleshooting assessment).

As an example of how these FIT goals were addressed within a particular session, in Session 1 students were asked to complete a brief survey and complete a set of 10–16 visualization tasks involving blocks. Students were then introduced to AgentCubes through a bridge design simulation and the Sokoban game (<http://en.wikipedia.org/wiki/Sokoban>). These computer-based assessment activities and problem-solving challenges provided students an opportunity to engage in sustained reasoning through most of Session 1 (Level 5 intensity). During the same session, students also had the opportunity to use a graphics package as they selected agents in the bridge design simulation and modified the design to construct a bridge with the fewest number of bridge elements. The creation of illustrations, therefore, was a secondary FIT goal that students had the opportunity to explore in Session 1 (Level 4 intensity). A few students were able to devise an algorithm for placing blocks and solving levels in Sokoban (Level 2 intensity). Although there were some opportunities in Session 1 for students to manage complexity, in terms of programming agent behavior, students were directed to a different activity before they had the chance to explore this feature in AgentCubes (Level 1 intensity).

CF3 – Students have capacity for visualization and representing 3D objects as illustrated by their ability to navigate 3D mazes and create their own: All students were

able to navigate existing mazes and create their own 3D mazes with varying degrees of complexity. Students could create a 3D maze with AgentCubes, by designing pathways through a large solid cube, following specific design criteria and with the expressed goal of constructing a maze that would offer sufficient challenge to the maze user.

5.3.3. Broadening participation

The criterion we used to evaluate this category was whether the technology and curriculum could be used across ethnicity and gender, both in technology hub areas and inner city school cultures. The broadening participation findings (BPF) are as follows:

BPF1 – The idea of Game Design is compelling to middle school girls. We were able to easily recruit more than 50% girls: The percentage of female students involved at both schools was greater than 50%. Organizing the weekly sessions by gender may have had some influence on the ability to recruit a higher percentage of female students to agree to participate in these sessions. This was influenced by earlier experiences in recruiting female students in after-school STEM courses offered by Science Discovery. Student attendance over the five sessions experienced some attrition, with the most significant attrition occurring among the Aurora school female group. Based on follow-up discussions with teachers and students, it appears that there were various reasons for this attrition such as overlapping family commitments or other after-school commitments.

BPF2 – Students from the Aurora school did better than the tech hub school in authentic assessment (but the difference was not statistically significant): The troubleshooting performance of students at both schools was essentially the same. The Aurora students outperformed the Boulder students on the challenging frog movement tasks.

BPF3 – There was no major difference between the ethnicity groups in troubleshooting performance: From Table 3, we see that African American students on average completed 80% of the troubleshooting tasks during the authentic assessment activities. Hispanic students on average completed 72% of the tasks. Caucasian students on average completed 68% of the tasks. Other Ethnicity students on average completed 67% of the tasks. Note that both the African American and the Other Ethnicity groups were small ($n = 3$).

6. Conclusions and future work

Our preliminary experiences and findings with Scalable Game Design, our low-threshold/high-ceiling framework supporting skills beyond programming, ranging from theoretical design skills to concrete development skills, lead us to believe that we can establish IT fluency and broaden participation in computer science with game design activities. The results from the study described herein indicate that it is educationally effective to use AgentCubes as a low-threshold game design environment featuring Incremental 3D for teaching IT skills to middle

school students. The AgentCubes instructional sequence did result in opportunities to promote student fluency and the troubleshooting scenarios designed to be used with AgentCubes can be used to document student IT fluency.

Promoting student IT fluency: The results suggest that 10h of instruction using AgentCubes did result in the development of student IT fluency across several elements, in particular algorithmic thinking, programming, and managing faulty situations. Even though students had some prior experience with computer software, no student had previous experience with AgentCubes and yet by the end of five sessions they were able to demonstrate that they could identify and remediate problematic agent behavior. Data from the observation protocol outline how particular activities and instructional emphases contributed to the development of student IT fluency and the results from the troubleshooting activity confirm that students understood some key features of AgentCubes, game design, and programming.

Promoting computational thinking: The discussion on what computational thinking [3] is and how to promote it is an ongoing discussion. We believe that the combination of incremental approaches with the low-threshold end-user programming of AgentCubes is an essential combination for building computational thinking tools. Perhaps the most important aspect of Incremental 3D with respect to computational thinking is the support of incremental formalization. The ability to draw simple, abstract 2D or 3D shapes that can be manipulated without the need for any programming can facilitate design, or more generally, the thinking process. Similar to classic LEGO blocks, these agents can be employed to represent just about anything. Then, the process of adding behaviors and evolving the simple 2D objects into more sophisticated 3D objects, becomes an essential part of a computational thinking process.

Using troubleshooting scenarios as authentic assessment: Designing assessments that reveal what students have learned through use of computer software necessarily relies more on how students can apply what they have learned rather than showing learning gains beyond a pre-assessment. Given students' lack of familiarity with the AgentCubes interface and the relatively brief contact time with students, it was important to design an activity that could be motivating to students, have some instructional value (i.e. the assessment was a learning opportunity), and serve as an assessment of student fluency.

We would argue that the troubleshooting activity is an authentic assessment [36,37], since it emulates the type of work expected of game designers and computer programmers, requires the application and synthesis of knowledge and skills to find a solution to a problem worth solving, involves some degree of self-assessment on the students' part to determine when the goals are satisfied, and leaves room for student creativity. The results demonstrated that students were not only quite successful in completing most of the tasks, but they were also fully engaged in the activity for the entire time. It is worth emphasizing that the participants in this study were sixth and seventh grade students who had no previous experience with AgentCubes, and yet very few students demonstrated any

outward signs of frustration or reluctance to improve agent behavior. The major finding from this study rests, in part, with the results from this authentic assessment. That is, over the five sessions, students developed sufficient fluency with programming agent behavior to be able to apply their knowledge of AgentCubes procedures in new problem scenarios. Furthermore, we would argue that the troubleshooting activity is an appropriate authentic assessment to use with middle school students and is likely the ideal approach to assessing students' understanding of how to use new software. Although students' facility with these activities may be the result of the user-friendliness of the AgentCubes' programming interface, we feel this type of activity is worth pursuing to assess student fluency with other design software.

Differential commitments for female and male students: While there may be some differential gender effects regarding sustained attendance of female students in AgentCubes sessions, there is no indication from survey results or school personnel who assisted with recruitment that the course was less attractive to female students. Rather, anecdotal evidence suggests that other after-school commitments (e.g. band, clubs, sports, etc.) seemed to have had a greater impact on attendance, in general, at the Aurora school and may have had a greater impact on attendance of female students. The after-school sessions can be a productive time for many students; school administrators are also supportive of using after-school time in this way. However, for some students other after-school activities, transportation arrangements, and family commitments challenge sustained attendance in an after-school CS program. In addition, the greater the duration of the instructional sequence, the greater the chance students will be absent from sessions which will hinder their opportunity to stay with the rest of the group in terms of learning new programming and design techniques.

Although the reasons for these differential attendance patterns are conjectures, we feel that a promising strategy to improve the attrition rate would be to either offer the AgentCubes sessions during the school day (i.e. as part of an applied technology or computer applications course) or provide a more compact session over the course of one week (five half-day sessions) rather than organized as 2-h sessions each week over a period of 5 weeks.

Recommendations for future studies: The five sessions of AgentCubes provided a sufficient balance of instruction with the user interface, essential aspects of agent behavior, and programming needs to transition from 2D to 3D behavior. The sufficiency of instruction was demonstrated by what students were able to accomplish within the context of instructional activities and assessment tasks. There is, however, a sense that students were eager to learn quite a bit more about game design and would have continued to attend additional sessions, if offered. To enhance middle school students' IT fluency, conceptions of design, or programming of agent behavior, additional instructional time is required. We plan to explore these learning potentials in subsequent studies.

In our future work, we intend to study the systemic needs and impact of the implementation of this approach to increase IT fluency among middle and high school

students. To accomplish this we will scale up research and development along different dimensions:

- *Technology:* provide more scaffolding techniques [27–29] especially for incremental programming.
- *Content and curriculum:* develop longer modules offered as part of the curriculum for comprehensive coverage of IT standards.
- *Teacher training:* a systematic approach to teacher training is essential for technology adoption in schools.
- *Social factors:* explore the factors leading to the somewhat disappointing attrition rates for girls, given their interest in game design and ability to achieve the level of fluency required to create their own games.

Acknowledgements

This work was supported by the National Science Foundation (NSF) under Grant no. IIP 0712571. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

- [1] Computer Research Association, CRA Bulletin: Enrollments and Degree Production at US CS Departments Drop Further in 2006/2007, 2008.
- [2] Committee on Information Technology Literacy, Computer Science and Telecommunications Board, Commission on Physical Sciences, Mathematics, and Applications, National Research Council, Being Fluent with Information Technology, National Academy Press, Washington, DC, 1999. Available from: <<http://newton.nap.edu/html/beingfluent/>>.
- [3] J.M. Wing, Computational thinking, Communications of the ACM 49 (3) (2006) 33–35.
- [4] K. Salen, E. Zimmerman, Rules of Play: Game Design Fundamentals, MIT Press, Cambridge, MA, 2003.
- [5] President's Information Technology Advisory Committee (PITAC), Report to the President: Computational Science: Ensuring America's Competitiveness, June 2005.
- [6] A. Repenning, A. Ioannidou, Behavior processors: layers between end-users and Java virtual machines, in: Proceedings of the 1997 IEEE Symposium of Visual Languages, Capri, Italy, 1997, pp. 402–409.
- [7] A. Repenning, A. Ioannidou, Agent-based end-user development, Communications of the ACM 47 (9) (2004) 43–46.
- [8] A. Repenning, A. Ioannidou, What makes end-user development tick? 13 design guidelines, in: H. Lieberman, F. Paternò, V. Wulf (Eds.), End User Development, Springer, New York, NY, 2006, pp. 51–86.
- [9] A. Repenning, A. Ioannidou, Broadening participation through scalable game design, in: Proceedings of the ACM Special Interest Group on Computer Science Education Conference (SIGCSE 2008), Portland, OR, USA, 2008, pp. 305–309.
- [10] S. Walter, B. Barron, K. Forsell, C. Martin, Continuing motivation for game design, in: CHI 2007, San Jose, CA, USA, 2007, pp. 2735–2740.
- [11] M. Prensky, Programming: The New Literacy, in Edutopia. Available from: <<http://www.edutopia.org/programming-the-new-literacy>>, 2008.
- [12] U. Wilensky, W. Stroup, Learning through participatory simulations: network systems learning in classrooms, in: Computer Supported Collaborative Learning (CSCL '99), Stanford University, CA, USA, December 12–15, 1999.
- [13] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, M. Resnick, Scratch: a sneak preview, in: Second International Conference on Creating, Connecting, and Collaborating through Computing, Kyoto, Japan, 2004, pp. 104–109.
- [14] D.J. Barnes, M. Kölling, Objects First with Java: A Practical Introduction using BlueJ, third ed., Pearson Education/Prentice-Hall, Englewood Cliffs, New Jersey, 2006.

- [15] M. Klling, B. Quig, A. Patterson, J. Rosenberg, The BlueJ system and its pedagogy, *Computer Science Education* 13 (4) (2003) 249–268.
- [16] H. Poul, K. Michael, Greenfoot: combining object visualisation with interaction, in: *Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, ACM, Vancouver, BC, Canada, 2004, pp. 73–82.
- [17] M. Barbara, L. Deborah, C. Stephen, Evaluating the effectiveness of a new instructional approach, in: *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* Norfolk, ACM, VA, USA, 2004.
- [18] E. Klopfer, S. Yoon, Developing games and simulations for today and tomorrow's tech savvy youth, *TechTrends* 49 (3) (2005) 33–41.
- [19] J.S. Harbour, J.R. Smith, *Beginner's Guide to DarkBASIC Game Programming*, Course Technology PTR, Florence, KY, 2003.
- [20] M. Overmars, Teaching computer science through game design, *Computer* 37 (4) (2004) 81–83.
- [21] M. Dertouzos, *Creating the People's Computer*, vol. 100(3), MIT Technology Review, Cambridge, MA, 1997, pp. 20–28.
- [22] A. Henderson, M. Kyng, There's no place like home. Continuing design in use, in: *Design at Work*, Lawrence Erlbaum Association Publishers, London, 1991, pp. 219–240.
- [23] A.I. Mørch, Three levels of end-user tailoring: customization, integration, and extension, in: M. Kyng, L. Mathiassen (Eds.), *Computers and Design in Context*, MIT Press, Cambridge, MA, 1997, pp. 51–76.
- [24] H. Lieberman, F. Paternò, V. Wulf, *End User Development*, vol. 9, Springer, Berlin, 2006, p. 492.
- [25] P. Maes, *Designing Autonomous Agents*, MIT Press, Cambridge, MA, 1990.
- [26] A. Repenning, Inflatable icons: diffusion-based interactive extrusion of 2D images into 3D models, *Journal of Graphical Tools* 10 (1) (2005) 1–15.
- [27] M. Guzdial, Software-realized scaffolding to facilitate programming for science learning, *Interactive Learning Environments* 1 (1994).
- [28] B.J. Reiser, Scaffolding complex learning: the mechanisms of structuring and problematizing student work, *Journal of the Learning Sciences* 13 (3) (2004) 273–304.
- [29] L.A. Shepard, Linking formative assessment to scaffolding, *Educational Leadership* 63 (3) (2005) 66–70.
- [30] A. Repenning, A. Ioannidou, AgentCubes: raising the ceiling of end-user development in education through incremental 3D, in: *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'06)*, Brighton, United Kingdom, 2006, pp. 27–34.
- [31] A. Michotte, *The Perception of Causality*, Methuen, Andover, MA, 1962.
- [32] A. Ioannidou, A. Repenning, C. Lewis, G. Cherry, C. Rader, Making constructionism work in the classroom, *International Journal of Computers for Mathematical Learning* 8 (1) (2003) 63–108.
- [33] A. Repenning, Collaborative diffusion: programming antiobjects, in: *OOPSLA 2006, ACM SIGPLAN International Conference on Object-oriented Programming Systems, Languages, and Applications*, Portland, Oregon, 2006, pp. 574–585.
- [34] A. Repenning, Excuse me, I need better AI!: employing collaborative diffusion to make game AI child's play, in: *ACM SIGGRAPH Symposium on Videogames*, Boston, MA, USA, 2006, pp. 169–178.
- [35] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*, Harper Collins, New York, 1990.
- [36] J. Gulikers, T. Bastiaens, P. Kirschner, A five-dimensional framework for authentic assessment, *Educational Technology Research and Development* 52 (3) (2004) 67–86.
- [37] F. Newmann, W. Secada, G. Wehlage, *A Guide to Authentic Instruction and Assessment: Vision, Standards, and Scoring*, Wisconsin Center for Education Research (WCER), University of Wisconsin, Madison, WI, 1995.