Re-conceiving Introductory Computer Science Curricula through Agent-Based Modeling

Forrest Stonedahl, Michelle Wilkerson-Jerde, Uri Wilensky
Center for Connected Learning and Computer-Based Modeling
Northwestern University, Evanston, Illinois, U.S.A.
{forrest, m-wilkerson, uri}@northwestern.edu

ABSTRACT

We present a preliminary version of the MAICS (Multi-Agent Introduction to Computer Science) framework, which is a new approach for revitalizing introductory undergraduate or high school computer science curricula through the deep integration of agent-based modeling (ABM) and multi-agent systems (MAS) perspectives. We have developed a suite of educational agent-based models highlighting several key ideas of computer science. We discuss the merits of using multi-agent systems as a lens for conceptual understanding across disciplines, and how this approach can be beneficial for exploring topics that computer science educators might not normally consider to fall under the heading of ABM or MAS. We show that this perspective offers insights for many sub-fields, including searching, sorting, optimization, graphics, machine learning, networks/security, and operating systems. In particular, we highlight several areas where parallel, distributed, stochastic, and emergent methods can be incorporated fruitfully into early computer science curricula that too often focus solely on serial, deterministic, and centralized algorithms. It is our belief that an ABM/MAS paradigm can also improve accessibility of content for students, by providing motivating example models, and a 'glass-box' approach that encourages both understanding and experimentation. Furthermore, bringing disparate topics in computer science together through the common focus on emergent systems can promote a broader, more accurate, view of the field as a whole.

Categories and Subject Descriptors: K.3.2 [Computing Milieux]: Computer and Information Science Education – *curriculum.*

General Terms: Human Factors

Keywords: Computer Science Education, Agent-Based Modeling, Multi-Agent Systems, Curricular Models, Computational Thinking

1. MOTIVATION

Last year, Rick Rashid, a senior vice president for research at Microsoft, asked the rhetorical question of whether computer science is a dying profession [29]. Indeed, shrinking undergraduate computer science enrollment and concern about the underrepresentation of both women and minorities in computer science has been the subject of much speculation, concern, and debate, particularly in North America [18, 22, 20]. Diversifying the introductory curriculum is one approach for reaching a broader audience (see, e.g., [18, 17, 19]), which has met with some success. In this paper, we present the MAICS (Multi-Agent Introduction to Computer Science) framework as a new and powerful method for diversifying the introductory computer science curriculum. Through the MAICS framework, we demonstrate the potential to address many conventional topics of computer science (such as searching, sorting, optimization, graphics, machine learning, networks/security) in a unconventional way, through an agent-based modeling (ABM) and multi-agent systems (MAS) perspective. The framework focuses on two central goals. First, it seeks to enrich early ("low-level") computer science courses by engaging students with conceptually rich "high-level" topics. Second, it emphasizes the role of distributed, decentralized systems; a concept that has strong implications both within and far beyond the domain of computer science.

More specifically, the first strand of our research aims to address the enrichment of early ("lower level") computer science courses with a series of dynamic agent-based models coupled with compelling and interactive visualization. While an ABM/MAS approach can easily reach out to interdisciplinary examples from fields such as biology, economics, physics, sociology, biomedicine, and others, for the purposes of this paper we chose to stay primarily within the bounds of computer science. This approach provides the additional benefit of offering a survey of several conventionally "higher-level" topics, and gives introductory students a broader intellectual taste of what computer science has to offer, beyond "hello world", sorting algorithms and syntax errors. Concepts of elementary programming can be covered concurrently through experimentation with the provided source code, and through extending the model or writing new agent-based models from scratch.

The second strand is based more on the technical content of computer science education and our vision of the future of computing. In recent years, computing and computer science have been undergoing an important shift toward parallelism. This includes the current prevalence of multiple and multi-core processors, the ubiquity of high performance computing clusters in academia and industry alike, cloud computing, massive peer-to-peer networks, social networking and Web 2.0 applications, increased deployment of massively parallel supercomputers for research, consumer-grade GPUs (graphical processing units) that deliver a teraflop of parallel computing power, as well as parallel languages and language features to accompany these developments. We

are not advocating that CS101 students need to be learning GPGPU programming techniques or dealing with mutual exclusion semaphores for accessing shared memory. The point is a broader one: that it is time to re-examine whether the prevalent focus in contemporary introductory computer science courses on centralized, deterministic, serial algorithms is best preparing our students in the long run, when they will eventually need to face a world of computation that is ubiquitously distributed, potentially stochastic, and increasingly parallel. Our work is further motivated by the larger goal of providing universal instruction in "computational thinking" (as described by Wing [37]), wherein students across all disciplines become fluent in computational methods and models. The MAICS framework is a step in this direction, both in supporting a broader view of computer science, and working to increase the accessibility of computational concepts for a larger audience.

Toward addressing these two concerns, we have developed a suite of educational agent-based models highlighting several key ideas of computer science, which illustrate the ideas of the MAICS framework. Since the word "agent" connotes different meanings to different audiences, we should mention that we interpret "agent" somewhat broadly to include very simple elements acting with a small fixed set of rules, and we do not limit its use only to agents that use highly sophisticated decision-making processes. Furthermore, the agents used in the suite of models we present here are all fairly simple agents, which we believe is appropriate for an introductory course on computer science, and also beneficial for developing skills in decentralized thinking.

The paper is structured as follows. We first discuss related research in computer science education, and argue for the merits of using agent-based modeling (ABM) as a "lens for conceptual understanding" when exploring topics that computer science educators might not traditionally consider to be in the domain of ABM or MAS. Next, we explore three example curricular models in some detail, and discuss how they may be used to promote understanding of multi-agent systems while learning about computer science topics. We also offer a brief overview of each of the other models in the suite. We conclude with some remarks about potential implementation considerations and discussion of our future work.

2. RELATED WORK

There have been many suggested approaches for revitalizing computer science education. In this brief review we list only a few, for comparison to our own ideas and approach. Specifically, we believe that introducing agent-based modeling to introductory computer science curriculum addresses many calls in the computer science education literature to engage students in motivating consequential tasks and to highlight the interdisciplinary nature of computer science and its applications. Additionally, we believe that the ABM/MAS paradigm is particularly amenable to introducing computer science topics at not only the collegiate level, but also to students at the secondary level and earlier.

Some proposed changes to help promote understanding, motivation, and retention in the introductory computer science sequence include the integration of design-first programming [25], pair programming[16], and robotics [11]. Others argue that attention to content delivery techniques should complement efforts to promote students' own relationship with computer science material: Naps et al. [26] discuss the role of visualization in CS courses, and in particular argue that "[visualization] technology, no matter how well it is designed, is of little educational value unless it engages learners in an active learning activity." We argue that agent-based modeling offers an excellent approach for addressing these issues. Specifically, a curriculum designed around the ideas of ABM/MAS can provide an effective coupling of advances in computer science education methods (e.g., visualization technology) with the more general goals of active engagement and intellectual inquiry on the part of students.

In addition to local changes to the core computer science curriculum, many have argued for computer science to be better integrated into a broad curriculum centered around science and technology. In response to declining CS enrollment, Denning and McGettrick [18] call for a "recentering" of computer science, lamenting that in the public mind "computer science" has become narrowly associated with the job of "programmer", and suggest an introductory CS sequence with a theme of technological "innovation." Our suggestion to use ABM/MAS as an introductory theme is a less radical change, particularly since learning the art of computer programming remains a central piece of our educational framework (students will simply be programming multi-agent simulations, rather than, for example, writing programs for counting prime numbers). However, we do agree that early computer science courses often provide too narrow a view of what it means to be a computer scientist, and suggest that our approach will offer broader exposure to "upper level" topics.

Cushing et al. [17] suggested broadening introductory CS by offering interdisciplinary courses with math and science (such as ecology) as a means to improve retention and increase interest in the field. We believe that the ABM/MAS paradigm is especially conducive to such interdisciplinary integration, and one particularly powerful interdisciplinary idea is that of emergence – how the interactions between agents each exhibiting simple behavior at the individual level can result in surprising and complex aggregate-level phenomena that appears to be "more than the sum of its parts." [21, 36] Emergence is a key conceptual bridge to a multitude of disciplines (including chemistry [23], materials science [12], electromagnetism [32], and biology [35]), and we believe it is equally important for it to be an ingredient in computer science education. In Section 3 we illustrate how emergence is a key component of many of the models in the MAICS suite. With the MAICS framework we have chosen to focus on topics that are considered within the discipline of computer science in order to provide a broad survey of higher level computer science topics, but at the same time the ABM/MAS paradigm allows us to make connections to a wide array of interdisciplinary endeavors.

This notion of core concepts within and between disciplines is also why we refer to ABM/MAS as a "lens for conceptual understanding" in computer science. Often, subjects such as neural networks, particle systems, genetic algorithms, sorting and searching, are organized topically within the computer science curriculum, and are thus taught separately, without making conceptual connections between them. In contrast, an ABM paradigm uses concepts such as agents and micro/macro level phenomena in order to highlight the similarities and differences between the mechanisms at work, rather than focusing primarily on the subject area.

Model Name	Topic
PageRank	Searching
Painted Desert Challenge	Sorting
Virus on a Network	Network Security
Simple Genetic Algorithm	Optimization
Particle Swarm Optimization	Optimization
Artificial Neural Net	Machine Learning
Particle Systems Flame	Computer Graphics
Flocking 3D	Computer Graphics
Dining Philosophers	Operating Systems

Table 1: MAICS suite models and related computer science topics, listed by order of appearance in this paper.

Finally, while most introductory sequences in computer science are offered at the college or university level and we present this framework primarily in that context, we also acknowledge the important role of computer science education at the pre-collegiate level [28]. Just as in college, enrollment in high school computer science courses is low, and there have been calls for a more diverse, integrated computer science curriculum [20]. We suggest that an ABM/MAS paradigm, and the MAICS framework specifically, is also a powerful way to introduce computer science to a younger audience. In support of this assertion, we note that the NetLogo modeling environment [34], and even several of the agent-based models discussed specifically in this paper, have been successfully used in workshops and educational interventions as early as primary school (as well as with industry professionals and academic researchers both familiar and unfamiliar with computer programming).

3. THE MAICS FRAMEWORK

The MAICS framework is situated to address several goals, including the enrichment of early CS courses with a broader range of content, improving students' understanding of parallel, non-deterministic, and distributed systems, and offering a more exciting and dynamic introduction to the field. We have developed a wide collection of agent-based models (available for download from the NetLogo Models Library `http://ccl.northwestern.edu/netlogo/models/`), and for the purposes of this paper and the MAICS framework we selected a cross-section of those models that relate to important or motivating topics in computer science. The suite of models we describe herein consists of nine models spanning seven topics, as shown in Table 1. Some of these models have been used with great success in short workshops, or introductory courses on multi-agent modeling, but we have yet to implement an entire course using this approach. We present here a cohesive framework for such a design that we hope will be refined through trial, as well as feedback from others working in this area.

This is certainly not intended to be a comprehensive list of topics in computer science that could benefit from re-examination from an agent-based perspective. Instead, we seek to highlight several examples where parallel, distributed, stochastic, and emergent methods can be incorporated fruitfully into early computer science curricula that too often focus solely on serial, deterministic, and centralized algorithms. Furthermore, this list contains only fully implemented and documented models that are presently ready for educational use. More models could certainly be added to this list, highlighting other important ideas. Some of these topics (such as searching and sorting) are similar to those traditionally covered in an introductory CS sequence while others (such as particle swarm optimization) are more typically found in upper-level undergraduate or even graduate-level courses. Due to space constraints, we will discuss only the first three example models in detail, and then briefly explain the scope and purpose of each of the others.

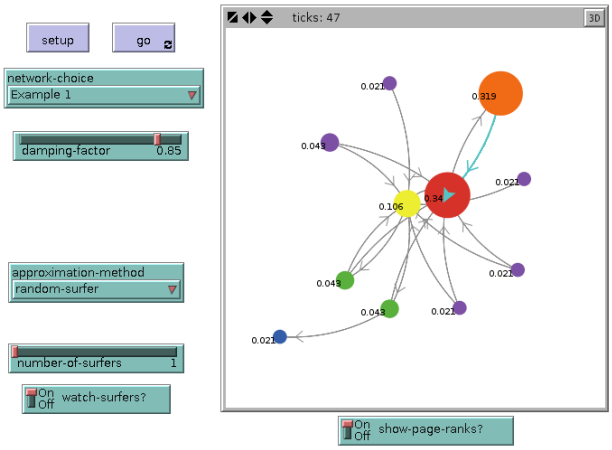

Figure 1: A screenshot from the PageRank model. Larger nodes represent higher PageRanks.

These models were all implemented using the NetLogo [34] agent-based language and integrated modeling environment, which permits interactive modification of the model's parameters and the code itself. The NetLogo language, following the Logo tradition [27], has also been designed to be easy to read and easy to learn, and the integrated modeling environment contributes to a low barrier for entry [33]. Equally important, NetLogo is no "toy language"; it is a real language currently being used by researchers across the globe, offering a wide range of control structures and data types, and it is extensible via the Java programming language if access to additional libraries is required. In addition, NetLogo's built-in facilities for model visualization provide students with a convenient graphics library. In short, even if our curriculum was not based around a multi-agent perspective (which is NetLogo's predominant feature), NetLogo would still be a suitable choice for a first course in computer programming. We also wish to emphasize the "glass box" nature of the suite of models: besides the visual interfaces (shown in figures below), each model comes complete with educational documentation and full source code that students can easily edit and run within the NetLogo modeling environment.

3.1 Searching: PageRank model

Traditional computer science curricula invariably include discussions of searching, often starting with students learning to do a sequential search in an array of numbers of strings. Later on, they are often taught how to perform a binary search of sorted data, and to search other data structures such as trees or graphs, perhaps using depth-

first search, breadth-first search (or perhaps Dijkstra's algorithm). While we have no desire to debate the merit of these venerable and classic algorithms, we note that they are all designed to run deterministically on a single processor accessing an unchanging data set. It seems prudent to balance this with a decentralized algorithm designed for searching massive quantities of constantly changing data, i.e. the World Wide Web. Furthermore, we suspect students may be more motivated to learn about how Google "magically" returns relevant search results about their favorite curling team, as opposed to discovering how to find the position of "milk" in an alphabetized grocery list. The PageRank model [6] (see Figure 1) is based on the now famous PageRank algorithm developed by the founders of the Google search engine in the late 1990s [14]. PageRank is not technically a search algorithm, but rather a ranking algorithm, which provides a basis for ranking the information on one page as being more useful/important/relevant than the information on another page. The algorithm assigns a PageRank score to each web page, based on its relationship to other pages determined by the hyperlink structure of the web. Our PageRank model actually demonstrates two distinct agent-based methods for calculating the PageRank of a directed network (such as the web), though the two methods result in the same limiting behavior, and ultimately would assign the same PageRank scores to each page.

Method 1: Random Web Surfers. In this case, we assume there are "page" agents which are connected to each other in a directed network of hyperlinks, and there are also "web surfer" agents, which operate using these simple rules. They start at a random web page, and begin wandering the web. To wander, they either click on a link from the current page and travel to a new page, or they may (through some unspecified means – perhaps a TV commercial, typing in a web address, an email from a friend, etc) jump directly to a random page somewhere on the web. If they run into a dead end page, they also jump to a random page. The probability with which they follow a link versus jump to a random page is controlled by a parameter called "damping factor" (typically set at 85% chance of link-following). As these agents, move, the model records the number of times a web surfer has visited each page. One definition for the PageRank metric is given by the probability of a single random web surfer being at that page at a given instant. Using the random web surfers model, this can be easily calculated by dividing the number of visits for each page by the total number of visits. In more formal mathematical terminology, this can be viewed as finding the stationary distribution for a certain Markov Chain, where each page is a state, and there are transitional probabilities specified between each pair of states. However, introductory CS students do not need to have acquired this level of mathematical formalism to appreciate the emergent behavior of the agent-based model.

Method 2: Diffusion of PageRank Scores. In this case, the primary agents in the model are the web pages themselves. Each page starts off with an equal amount of PageRank score. At each time step, pages divide their PageRank up equally, and send it off as a gift to each the web page that they link to. (Pages with no out-bound hyperlinks are treated as if they linked to every single other page in the web.) Each page then receives PageRank gifts from each of the pages that link to it. Also, each page receives a certain amount of PageRank, just for existing (determined by the "damping-factor" parameter). This redistribution of PageRank via diffusion is carried out repeatedly, and over time the PageRanks converge toward the correct PageRank value. Mathematically, this method is related to the "power method" for finding the dominant eigenvector of a modified adjacency matrix for the directed graph formed by the hyperlinks.

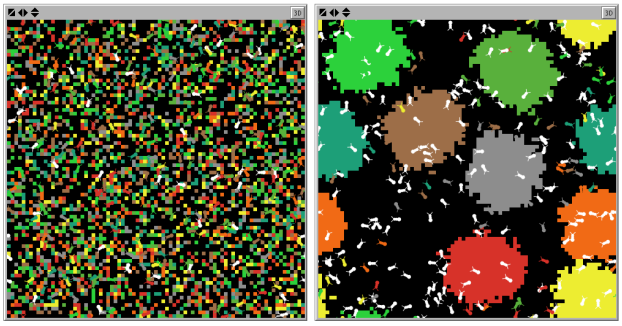

Figure 2: "Before" and "after" from the Painted Desert Challenge model, demonstrating the reduction in entropy caused by the agents' behavior.

Beyond the clear benefits of exploring and understanding this classic algorithm that is so instrumental in making information accessible on the web, our PageRank model also provides an excellent launching point for students to experiment by creating their own distributed link analysis and/or ranking algorithms. For example, students could endow the "random surfer" agents with more sophisticated behavior (use of the "back" button, bookmarks) and see how the rankings would be affected. A broader discussion about emergent search techniques could also encompass ant foraging mechanisms, or the search of fitness landscapes performed by genetic algorithms (making a connection to the Simple Genetic Algorithm model also included in our suite).

3.2 Sorting: Painted Desert Challenge model

Sorting algorithms are another staple of early computer science education, inevitably including at least several of the following collection: bubble sort, selection sort, insertion sort, merge sort, quick sort, heap sort, bucket sort, shell sort, and radix sort. Again, a common theme is the deterministic single-threaded and serial aspects of sorting (although many of these algorithms can be at least partially parallelized). As a counterpoint, we wish to present a messier, distributed, and stochastic view of sorting, in the Painted Desert Challenge model [7]. While it may strike some as an incredibly inefficient approach to sorting, one should note that it is intrinsically parallel, reasonably robust, and could be applied in situations where the data is shifting during the sorting process, as a result of noise. However, it is important to keep in mind that we are not interested here in arguing for the merits of this particular sorting algorithm, but instead we are arguing for the merits of the ideas that students will be exposed to by exploring this model. The Painted Desert Challenge model offers insight into emergent systems, and in particular ant colony and other problem solving techniques inspired by nature.

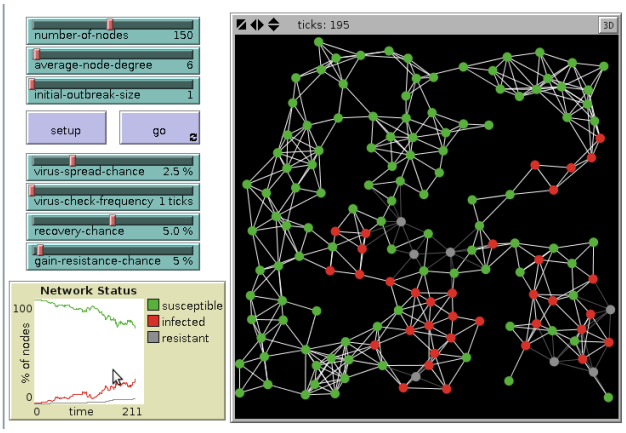

Figure 3: A screenshot from the Virus on a Network model.

The inspiration of this model goes back to a problem posed to participants in a study on decentralized thinking [30], where it was prefaced by a short whimsical vignette about insects that live in a painted desert and want to sort out each of the colors of sand after a windstorm mixed all the sand together. In this model, each termite follows the same set of very simple rules. It wanders in a 2D grid, wherein each grain of sand occupies one grid square. If it runs into a grain of sand, and it isn't already holding one, it picks it up. It continues to wander. If it runs into a grain of sand that is the same color as the one its carrying, it drops its grain in an adjacent location. The emergent result of this random wandering and picking up and dropping is shown in Figure 2. This is, however, just one possible set of rules. We do not expect students to learn computer science by passive observation, any more than we expect people to learn to bicycle by watching the Tour de France on television. It is imperative that they get their hands dirty in the code, take the model apart and put it together again. For instance, a simple extension would be to have the sand shifting while the termites are working, and measure the rate of entropy-reduction the termites are capable of. A more complicated extension would be to give the termites greater vision and more intelligence, and test if more complicated rules yield more efficient sorting. On the more theoretical side, we might ask students to try to prove that the algorithm will eventually yield a complete separation of each of the different colors. It is worth noting that there are other emergent sorting algorithms (e.g., Brueckner's sorting networks [15]) that could also be discussed in class and/or implemented as student projects.

3.3 Security: Virus on a Network model

Discussions about computer networks and security are not particularly common in introductory CS classes, which often focus more on programming and data structures. However, many computer science graduates go on to pursue careers in information technology where security is of paramount concern, which provides motivation for bringing this type of material into earlier coursework.

Rather than focusing on lower-level details of security, such as open ports or overrun buffer exploits, the Virus on a Network model [3] (see Figure 3) is concerned with security on a grander scale. In particular, worms and viruses that self-propagate from computer to computer through the Internet form a grave risk for today's society due in part to the creation of large "botnets" capable of acting in unison to carry out destructive distributed denial-of-service attacks, or other illicit activities. Virus on a Network is an abstract model, based on the SIR (Susceptible, Immune, Resistant) models found in epidemiology. The setup consists of nodes (i.e. computers) on a network, and links between them, which could represent a variety of different connections depending on the attack vector of the virus (e.g., email contacts, shared network drives, shared USB keys, external hard drives, or floppy disks, etc). Nodes start as susceptible, except for some specified number that are infected with the virus. With some probability (which is controlled by an adjustable model parameter), a node that is infected by the virus can spread that virus to each of its neighboring nodes. Infected nodes also have a chance of recovering (e.g., an antivirus program removed the virus but didn't close up the vulnerability), and they have a chance of recovering and becoming resistant to future attacks (e.g., an antivirus program inoculated the computer against this virus).

Through exploration of the model, students can learn about how the parameters affect the rapidity with which the virus moves through the network, as well as the lifetime of the virus, and the extent to which vaccination of a few nodes can or cannot prevent a widespread epidemic. The Virus on a Network model also has potential connections to other disciplines (such as medicine, marketing, or sociology), and promotes high-level discussions about computer security practices, the structure of social and computer networks, and the Internet. This also leads naturally to student projects and extensions of the model. For instance, the default network structure found in this model is based on spatial proximity of the nodes, with nodes that are closer together in the 2D plane having a high probability of being linked, whereas there are no long-distance links. Students can discuss whether such a configuration is plausible for virus contagion[1] and write code to generate other types of network. A few other possible extensions include allowing the virus to mutate and evolve, and thus be able to reinfect computers which had become immune to a previous version of the virus, or to allow for coordinated (botnet) attacks by groups of infected nodes, or two have multiple different viruses present in the network. There are always opportunities for ambitious students to take this type of work further, and spin it off into summer research projects.

3.4 Remaining Model Suite Overview

The remaining six models in our model suite (shown in Figure 4) cover topics from an additional four areas of computer science. The Simple Genetic Algorithm model [5] and the Particle Swarm Optimization model [4] both offer an introduction to the area of stochastic optimization algorithms. While these topics are not usually covered un-

[1]Generally speaking, it is not. Real-world networks usually display a power-law degree distribution and "small-world" structure where long-distance links are present. We purposefully chose this spatially-restricted structure to benefit visualization of the processes at work.

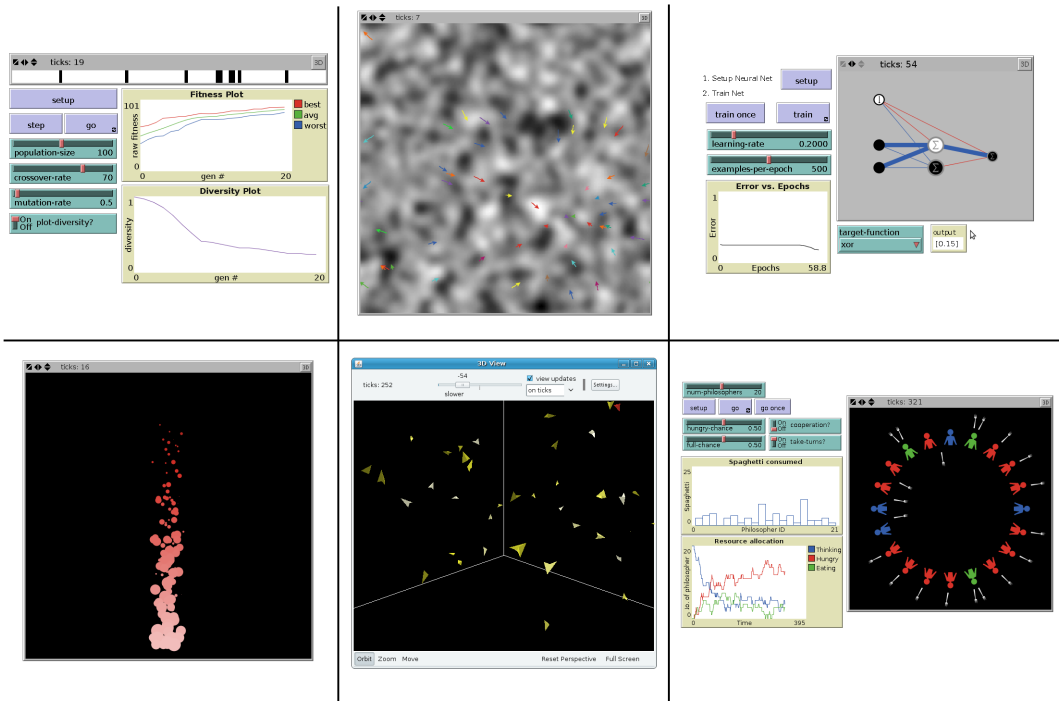

Figure 4: Model screenshots. Top row: Simple Genetic Algorithm, Particle Swarm Optimization, Artificial Neural Network. Bottom row: Particle System Flame, Flocking 3D, Dining Philosophers

til much later in a traditional computer science curriculum (probably an upper-level elective course), we believe they are thoroughly accessible to introductory-level students through the ABM/MAS perspective that is being cultivated through this framework. Additionally, the brevity of the code and the visualizations included in the models enhance the accessibility of content for these topics. Both of these examples show how very simple agents, acting with limited intelligence and information, can result in a population of agents moving toward a goal. In the Simple Genetic Algorithm model, this progress is measured by plotting the fitness and diversity levels in the population over time, as well as showing a visual representation of the best individual agent solution found in each generation. In the Particle Swarm Optimization, the progress towards a goal can be viewed as agents traverse a 2D fitness landscape, searching for a global optimum. These two models also offer interdisciplinary connections to evolutionary biology and particle physics. Artificial neural networks are also amenable to elucidation through an agent-based perspective, as we hope to demonstrate to students through exploration of the Artificial Neural Net model [2]. Each perceptron can be conceived as an agent, which follows certain rules during the training phase, and then another set of rules when it is being tested. This is another fairly advanced topic, which admittedly may take some effort for students to understand and appreciate. However, we should remind the reader that it is not necessary for students to understand every detail of the back-propagation training algorithm or what the nice mathematical properties of a sigmoid function are – such things can wait. The important thing is for students to gain a qualitative understanding of how the agents are activating each other, and that by automatically modifying the weights of connections between agents, it is possible for the system as a whole to "learn" pattern recognition skills. A class side-discussion comparing and contrasting this agent-based model with the neural networks found in humans should also prove provocative and educational. Agent-based modeling is useful in computer graphics as well, and is being increasingly explored as a means of automatically creating realistic procedural animations of systems with many interacting creatures or objects. Through the Particle System models[2] [1] students can get a taste of the classic "particle systems" approach sometimes used in cinematic animation to create the illusion of water, fire, or smoke. While each particle is fairly passive, being pushed or pulled by an externally determined force field, it is still useful to think of each particle as one agent of a distributed multi-agent system, and it is not difficult to modify the code to make the agents take a more active and/or intelligent role in their movement patterns. For instance, more sophisticated agent behavior is exhibited in the "Boids" algorithm [31] for creating realistic-looking flocks of animated creatures, which is the inspiration for our Flocking model [8] . As the Flocking screenshot in Figure 4 shows, the NetLogo modeling environment also has a 3D version that allows development and visualization of models in three dimensions, which provides students with an early glimpse into programming in 3D environments as they work to extend, modify, or create their own multi-agent computer

[2]Particle Systems is actually composed of four distinct models – Basic, Flame, Fountain, and Waterfall – but they all express the same fundamental idea.

animations. We mentioned above that part of the motivation for the MAICS framework was the increasingly parallel nature of computing, such as the shift to multi-core and multi-processor machines. In reaction, multi-threaded and multi-process programming will become more pervasive, and it seems quite appropriate to include in the curriculum an agent-based model that addresses issues of resource sharing. The Dining Philosophers model [9] introduces a classic case study in the synchronization of concurrent processes, posed as a puzzle about philosophers sitting around a table eating spaghetti that requires two forks to eat with, but having to share forks between them. Through this metaphor, concepts such as deadlock and resource starvation are explained.

4. DISCUSSION AND FUTURE WORK

Our intention here is to offer a window into an alternative introduction to computer science. In practice, we would not expect this approach to be used to the complete exclusion of other curricula. We emphasize that in many cases it would be most beneficial to compare and contrast centralized and decentralized approaches to the same topic. Furthermore, the introductory course can still have a strong emphasis on learning to write computer programs. However, starting with existing programs (in this case, agent-based models) provides an opportunity for students to explore, to modify, and to learn to read the language at the same time as they learn to write it.

One important thing to note is that the MAICS framework, and an ABM/MAS approach in general, still allows for the integration of a number of other techniques shown to be beneficial for computer science education. There is no reason, for example, that the pair programming approach [16] or the integration of robotics [11] cannot be implemented successfully within the context of MAICS. Indeed, the NetLogo programming environment includes interfaces to various physical devices and a variety of "bifocal modeling" [13] activities, which allow users in a variety of contexts to compare computational agent-based models with real-world data collected using robotic sensors and actuators. The VBot curriculum [10], designed primarily for middle school students, engages users in programming independent robot agents, which can then interact with one another in a shared context (such as a robot soccer arena).

Several avenues present themselves for future work. One interesting consequence of integrating ABM and MAS into introductory level computer science courses is that students are encouraged to think about multiple programming language paradigms. Specifically they are encouraged to consider how centralized and decentralized systems can be used to solve different problems, or even to solve the same problems differently. This is exciting in light of recent work that shows that even students with no formal computer science training are able to reason about concurrency, and some even to develop both decentralized and centralized solutions for problems involving concurrent systems [24]. However, the questions of when and how students should be introduced to multiple paradigms (e.g., agent-based programming, object-oriented programming, functional programming) in the computer science sequence deserves further attention. While teaching a wide range of paradigms in early courses offers students with a variety of choices to better solve different problems, there is the danger of leaving students afloat in a sea of shallowly understood paradigms. Additionally, implementing the MAICS framework would provide a context for additional cognitively motivated research about the affordances and constraints of decentralized thinking for students with regard to the principle ideas of computer science.

Another key piece of future work is to perform a concrete implementation of this framework in the form of an introductory level computer science course, or two-course sequence. We suggest that an implementation will provide empirical support for the theoretical underpinnings of the MAICS framework, as well as offer new insights about the potential for introducing students to computer science through an ABM/MAS paradigm. NetLogo and NetLogo models have been successfully introduced to and used by novice programmers in a variety of domains, and we expect that doing so in the context of computer science education using the MAICS framework will produce similar results. Specifically, we would evaluate how this implementation affects three major goals: (a) student engagement and retention rates, (b) student ability to exhibit distributed thinking when presented with problems that can benefit from it, (c) student knowledge of programming fundamentals (such as basic control structures, recursion, loops, variables, etc.). Previously, we have evaluated both student engagement and distributed thinking skills with positive results, but this was in the context of undergraduate and pre-collegiate computer science courses and workshops that focussed on modeling and simulation, rather than as a first course on computer science. In preparation for this work, we encourage discussion regarding implementation concerns and potential pitfalls for integrating agent-based modeling into computer science curriculum.

Through the MAICS framework we are offering a first attempt at producing a coherent introductory computer science curriculum centered around a series of agent-based models spanning a variety of topics. We believe that this framework addresses recent calls by computer science educators to introduce widely applicable, engaging curricula early in the computer science sequence that focus on the notion of "computational thinking", rather than specific algorithms and techniques. There unquestionably remains much room for improvement in this framework, and we hope that feedback on the selection of appropriate models or multi-agent systems that highlight important areas of computer science, as well as a broader discussion of the ideas we have proposed, will enable us to further refine these ideas.

Acknowledgments: We are particularly grateful to our colleagues – past and present – at the Center for Connected Learning and Computer-Based Modeling for their contributions to the models and ideas presented in this paper.

5. REFERENCES

Model References:

[1] Kornhauser, D., and Wilensky, U. NetLogo Particle System Flame model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 2007. http://ccl.northwestern.edu/netlogo/models/ParticleSystemFlame.

[2] Rand, W., and Wilensky, U. NetLogo Artificial Neural Net model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 2006. http://ccl.northwestern.edu/netlogo/models/ArtificialNeuralNet.

[3] Stonedahl, F., and Wilensky, U. NetLogo Virus on a Network model. Center for Connected Learning and Computer-Based Modeling, Northwestern University,

Evanston, IL., 2008. http://ccl.northwestern.edu/netlogo/models/VirusonaNetwork.

[4] Stonedahl, F., and Wilensky, U. Particle Swarm Optimization model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 2008. http://ccl.northwestern.edu/netlogo/models/ParticleSwarmOptimization.

[5] Stonedahl, F., and Wilensky, U. Simple Genetic Algorithm model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 2008. http://ccl.northwestern.edu/netlogo/models/SimpleGeneticAlgorithm.

[6] Stonedahl, F., and Wilensky, U. NetLogo PageRank model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 2009. Forthcoming.

[7] Wilensky, U. NetLogo Painted Desert Challenge model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 1997. http://ccl.northwestern.edu/netlogo/models/PaintedDesertChallenge.

[8] Wilensky, U. NetLogo Flocking 3D model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 1998.

[9] Wilensky, U. NetLogo Dining Philosophers model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 2003. http://ccl.northwestern.edu/netlogo/models/DiningPhilosophers.

General References:

[10] Berland, M., and Wilensky, U. Complex play systems: Results from a classroom implementation of VBot. Paper presented at the annual meeting of the American Educational Research Association, Montreal, Canada, 2005.

[11] Blank, D. Robots make computer science personal. *Communications of the ACM 49*, 12 (2006), 25–27.

[12] Blikstein, P., and Wilensky, U. An atom is known by the company it keeps: A constructionist learning environment for materials science using multi-agent simulation. Paper presented at the Annual Meeting of the American Educational Research Association, San Francisco, CA, April 7-11 2006.

[13] Blikstein, P., and Wilensky, U. Bifocal modeling: a framework for combining computer modeling, robotics and real-world sensing. Paper presented at the annual meeting of the American Educational Research Association, Chicago, IL, April 9-13 2007.

[14] Brin, S., and Page, L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems 30*, 1-7 (1998), 107 – 117. Proceedings of the Seventh International World Wide Web Conference.

[15] Brueckner, S. Emergent Sorting. Software Demonstration at the Fourth International Conference on Autonomous Agents (Agents 2000), Barcelona, Spain., 2000.

[16] Carver, J., Henderson, L., He, L., Hodges, J., and Reese, D. Increased Retention of Early Computer Science and Software Engineering Students using Pair Programming. In *Proceedings of the 20th Conference on Software Engineering Education & Training* (2007), IEEE Computer Society Washington, DC, USA, pp. 115–122.

[17] Cushing, J. B., Weiss, R., and Moritani, Y. CS0++ broadening computer science at the entry level: interdisciplinary science and computer science. *J. Comput. Small Coll. 23*, 2 (2007), 51–57.

[18] Denning, P. J., and McGettrick, A. Recentering computer science. *Communications of the ACM 48*, 11 (2005), 15–19.

[19] Downey, A., and Stein, L. Designing a small-footprint curriculum in computer science. In *Proceedings of the 36th ASEE/IEEE Frontiers in Education Conference* (2006).

[20] Goode, J. If You Build Teachers, Will Students Come? The Role of Teachers in Broadening Computer Science Learning for Urban Youth. *Journal of Educational Computing Research 36*, 1 (2007), 65–88.

[21] Johnson, S. *Emergence: The Connected Lives of Ants, Brains, Cities and Software*. Allen Lane, 2001.

[22] Katz, S., Allbritton, D., Aronis, J., Wilson, C., and Soffa, M. L. Gender, achievement, and persistence in an undergraduate computer science program. *SIGMIS Database 37*, 4 (2006), 42–57.

[23] Levy, S. T., Novak, M., and Wilensky, U. Students' foraging through the complexities of the particulate world: Scaffolding for independent inquiry in the connected chemistry (MAC) curriculum. Paper presented at the annual meeting of the American Educational Research Association, San Francisco, CA, 2006.

[24] Lewandowski, G., Bouvier, D. J., McCartney, R., Sanders, K., and Simon, B. Commonsense computing (episode 3): Concurrency and concert tickets. In *Proceedings of ICER '07* (Atlanta, Georgia, USA, September 15-16 2007), pp. 133–144.

[25] Moritz, S. H., Wei, F., Parvez, S. M., and Blank, G. D. From objects-first to design-first with multimedia and intelligent tutoring. In *Proceedings of the 10th annual conference on Innovation and techology in computer science education* (Caparica, Portugal, June 27-29 2005), pp. 99–103.

[26] Naps, T. L., Rössling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., and Ángel Velázquez-Iturbide, J. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bull. 35*, 2 (2003), 131–152.

[27] Papert, S. *Mindstorms: children, computers, and powerful ideas*. Basic Books, Inc. New York, NY, USA, 1980.

[28] Patterson, D. A. Restoring the popularity of computer science. *Communications of the ACM 48*, 9 (September 2005), 25–28.

[29] Rashid, R. Image crisis: Inspiring a new generation of computer scientists. *Communications of the ACM 51*, 7 (2008), 33–34.

[30] Resnick, M., and Wilensky, U. Diving Into Complexity: Developing Probabilistic Decentralized Thinking Through Role-Playing Activities. *The Journal of the Learning Sciences 7*, 2 (1998), 153–172.

[31] Reynolds, C. W. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM, pp. 25–34.

[32] Sengupta, P., and Wilensky, U. On Learning Electricity with Multi-agent Based Computational Models (NIELS). In *Proceedings of the International Conference of the Learning Sciences (ICLS)* (Utrecht, The Netherlands, 2008), vol. 3, pp. 123–125.

[33] Tisue, S., and Wilensky, U. NetLogo: A Simple Environment for Modeling Complexity. In *Proceedings of the International Conference on Complex Systems* (2004).

[34] Wilensky, U. NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 1999. http://ccl.northwestern.edu/netlogo.

[35] Wilensky, U., and Reisman, K. Thinking Like a Wolf, a Sheep, or a Firefly: Learning Biology Through Constructing and Testing Computational Theories-An Embodied Modeling Approach. *Cognition and Instruction 24*, 2 (2006), 171–209.

[36] Wilensky, U., and Resnick, M. Thinking in Levels: A Dynamic Systems Approach to Making Sense of the World. *Journal of Science Education and Technology 8*, 1 (1999), 3–19.

[37] Wing, J. M. Computational thinking. *Commun. ACM 49*, 3 (2006), 33–35.