david **reitter**

Teaching Statement

The best students and the best teachers have much in common. They are open-minded, critical, constructive, eager to learn from each other and inquisitive. That is who I aspire to be as a teacher. As an instructor at Carnegie Mellon and Penn State, I have developed a number of core ideas to guide my teaching. These principles encourage students to *experiment*, to *dig deeper*, and to *connect* with the materials and the field of research. The only mechanism for an active researcher to teach effectively is to teach efficiently with well-organized materials and well-thought-out projects. This applies to both areas I have taught: cognitive science and computer science. I have taught at the undergraduate and graduate levels; occasionally, I have been invited to teach cognitive science at international summer schools (e.g., at King Abdullah University of Science and Technology, Saudi Arabia, and Sunkyunkwan University, Seoul, South Korea).

Especially in my programming classes at Penn State I found that my role shifted from that of a lecturer to that of an instigator and motivator; learning in these classes happened as students practiced creating algorithms and translating them into code. In the best case, they learn to research their own questions rather than being fed memorizable knowledge. In the following, I will describe some tools that I developed to facilitate this. This can apply to teaching science classes as well; instead of developing algorithms, students become familiar with existing theories and then develop experiments whose outcomes may modify the theoretical models.

Learning must be experiential. To take an example from my cognitive science class: many classical experiments can be conducted easily in class. For example, original video and audio material from experiments or datasets may illustrate how lesions in certain language centers in the brain affect a person's speech. I let students diagnose Broca's and Wernicke's aphasias based on video recordings; I demonstrate the reliability of judgements of crowds, where the audience is to guess the weight of a bull shown in a picture (a classical experiment). In my *Cognitive Psychology* class at CMU, I also used an online experiment system, *CogLab*, which allowed students take a range of classical behavioral experiments. The students' data were then later discussed in class. In my programming classes, *problem-based learning* has come to be the standard approach; I rarely lecture using prepared slides now. Students either observe a demonstration of how to solve a programming problem (from scratch), they engage with me during the demonstration, they are guided in addressing the problem themselves (with checkpoints), or they tackle it in their own time. In such cases, I promote group-work, but discourage adopting specific roles in lieu of gaining individual technical expertise. (Each student submits their own assignments.)

Classes have to put individual results in context. Models and theories are essential in understanding why humans behave in certain ways, and they are instrumental in telling us what questions to ask next. While we have to justify theories and models with empirical results, it is important to paint "the big picture". As we know, memory performance is improved where expectations are created and contextual cues ground facts in systematic knowledge. I refer to theoretical considerations, but also realistic experiences and practical applications. My computer science teaching in Edinburgh and at Penn State gave me the opportunity to discuss algorithms and artificial-intelligence processing strategies not just from a theoretical standpoint, but also in the context of actual applications. Cognitive science gives me the opportunity to point out connections between the study of the mind and philosophy, or computer science.

We need to be up-to-date. How we present research depends on the class. An introductory class may cover ongoing, much-publicized discussions—can starlings process some form of grammar, and what does that mean, if anything? A graduate class will involve reading current papers. However, what is critical for both is the need to go beyond the historical account of how our field came about and engage students in debates of our time in order to stimulate interest and enthusiasm for cognitive science and computational methods. Interaction with researchers that continue to make ongoing contributions to their fields is what makes the difference between attending a research university and reading textbooks. I have repeatedly engaged even undergraduates with research topics, simply by inviting them to attend overview talks and participate in the academic life in the department.

We must teach the scientific method. We need to teach critical thinking, questioning results, and modeling. Scientific thinking provides valuable life lessons—they are not just training for future scientists. Concretely, we need to explain experimental design and careful statistical analysis. Even though many details may be left out in classes that do not explicitly deal with methods, a critical, careful approach applies to all of the work we present. A large, introductory overview class in science has to explain experimentation and the notions of controls and confounds. An advanced class will critically evaluate research papers. Engaging with data allows students to develop hypotheses, test them, and modify their models. In my class at Carnegie Mellon, I invited two teaching assistants to present their senior thesis works as a demonstration of the scientific contributions undergraduates and their advisors can make. In my undergraduate and graduate classes in cognitive science at Penn State and in Korea, students have the opportunity to design their own experiments to test a theoretical prediction that they first define themselves. Informally, the experiment is carried out in class, and students then analyze the results, often confirming or disconfirming classic theories.

We should motivate students to work independently. I believe in fostering independent thinking. I encourage more advanced students to take responsibility for their learning and to engage in the topic critically, rather than absorbing pre-digested material in small portions. I am aware and concerned that we are often unable to give individual attention to students in large classes, and that we tend to design exams according to the constraints posed by undergraduate graders who compare answers against a key. Thus, fostering creativity requires that I point my students to relevant literature and suggest areas where they can find motivation to design their own challenges. It also means that I expect them to come up with project ideas rather than, figuratively speaking, to fill-in-the-blanks on a

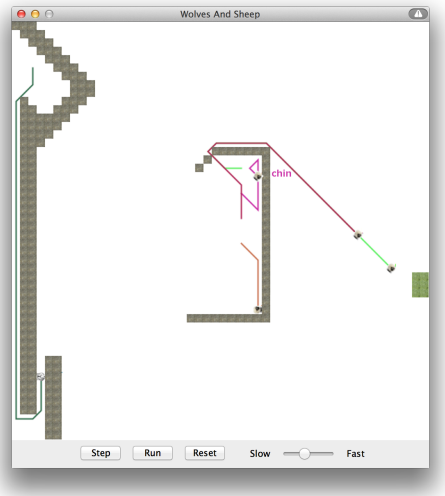

Figure 1: One of the more complex scenarios. The wolf is on the left and four sheep are visible on the right. Not all of them possess enough spatial intelligence to leave the box.

prepared worksheet. I am aware that we need to meet students where they are and not where we would like them to be. It is our job to guide them there.

These principles are embodied in the game of Wolves and Sheep. Programming can be a tough skill to learn. Those who excel at it have often started early. Most have spent countless nights doing detective work to figure out why their program does not work right. Programming takes a range of abilities beyond robust general computing skills: the syntax of an artificial language, the creative use of algebra, and intuition when to plan ahead and when to try out ideas in practice. Programmers read and write technical documentation, research tricky questions, and relate their work to business requirements. Learning these skills is a frustrating mountain ascent, with rocks in the way and many minor falls after which a climber has to get up and try again. All of this takes a great deal of motivation. A recently popular incentive in areas from crowdsourcing to classroom instruction is gamification. At Penn State, I have developed a game for my programming class that balances teamwork and individual work and addresses important lessons to be learned by novice programmers.

The basic idea is as follows. Students write small computer programs that act as players in a bigger bio-simulation. The simulated world has players move about a two-dimensional barren landscape with some obstacles and a few meadows. The players are of two kinds: sheep and wolves. The sheep aim to feed by moving to a meadow. Unfortunately, the wolves need to eat, too. The idea of computer programs as independent entities in a *toy world* is inspired by learning pioneers like Seymour Papert ("Logo"), who proposed "body-syntonic reasoning" as a way to learn algorithmic creativity (S. Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, New York, NY, 1980.)

The small agent programs written by the students play against each other in each round of the game. During the game, each player makes one step at a time in any direction on the game board, avoiding obstacles. A sheep's objective is to reach a goal location, while a wolf's objective is to eat a sheep.

The two-dimensional worlds these animals live in come in different scenarios. Sometimes, obstacles are arranged in a way that requires some navigation (see Fig. 1). Green pastures and start locations of the players are not always in the same place. Students are

given about 10 practice scenarios, but the tournament is run in earnest with additional, new ones. Such caveats force students to think about flexible strategies and implementations, which is an important real-world requirement. If a student's program assumes that obstacles or pastures are always in the same place, the bot will lose.

Each scenario starts with four sheep and one wolf at predefined or random locations (see Fig. 1). All players then take turns moving about; sheep naturally try to get to a green pasture as quickly as possible. The wolf attempts to get to and eat as many sheep as possible before they reach the pasture. The game ends when each sheep is dead or grazing happily, or when a timeout is reached.

I typically spend a number of sessions with the students to become familiar with the tournament interface. They work in small teams to help each other, although each student will submit his or her own, competitive program. Students can submit code at any time through a website. Their code runs in a tournament that starts out with some tried-and-tested players. The results are published immediately so that students can make incremental progress, but also feel motivated by the progress of their classmates.

Students sometimes do their own research to find algorithms for their bot, e.g., for path-planning. I have seen programming novices implement their own version of an A-Star search algorithm for this project. Many students tell me that they worked harder for this project than on any other one. I solicit regular comments on the class throughout each semester. As an example, here is one piece of feedback I received:

I thought that the Wolves and Sheep project was a lot of fun. I believe that it was a very unique opportunity to learn programming that I wouldn't have been offered if I hadn't taken this class. While I thought it was challenging (perhaps too challenging for someone that has no interest in learning programming) I think I got a lot out of the project. I never would have learned what an A algorithm was if not for the project, much less attempted to write a program that used one. Prior to this class, I'd never written a line of Java before. Coming out of the class, I feel very confident in my abilities to learn and understand programming. Bottom line, I had a great time writing my Sheep program and I really enjoyed the project.*

(See also: D. Reitter. Hungry wolves, creepy sheepies: The gamification of the programmer's classroom. In J. M. Carroll (Ed.), *Innovative Practices in Teaching Information Sciences and Technology.* Springer, New York, NY, 2014.)

Classes I Have Taught

I have taught classes at various levels at several universities. At Penn State, I taught cognitive science at the undergraduate information science majors (25 students), and also at the graduate level (12 students). I also taught programming (undergraduate beginner and intermediate-level, about 40 students per section). At Carnegie Mellon, I taught *Cognitive Psychology*, a large undergraduate overview class (140 students). I taught this class in short form to a graduate audience in Saudi Arabia and in South Korea.

I have also developed a new *Data Science* class that teaches R, visualization, and that introduces statistical modeling including standard machine-learning approaches. Empirical methodology is important to future computer engineers. I am also interested in teaching human-computer interaction, either as it relates to design (software development; desktop and mobile), or more from a human performance perspective (perception, higher-level cognition, language, modeling). I think that computational cognitive science has much to offer to students of computer science, as it captures important lessons for human behavior, such as those concerning learning, and decision-making under uncertainty (e.g., economic models of behavior).