Studio K: A Game Design Curriculum for Computational Thinking

Luke Kane, Wade Berger, Gabriella Anton, University of Wisconsin-Madison, 330 N. Orchard, Madison, WI 53705, lkane2@wisc.edu, wjberger@wisc.edu, ganton@wisc.edu
R Benjamin Shapiro, Kurt Squire, Morgridge Institute for Research, 330 N. Orchard, Madison, WI 53705, rbs@morgridgeinstitute.org, ksquire@morgridgeinstitute.org

Abstract: The Studio K curriculum is designed to engage students in habits of mind germane to game design, as well as computational thinking. Utilizing Microsoft *Kodu*, students are encouraged to reflect on their own gaming experiences to decompose and analyze the reasons why games are fun, and then transfer those patterns to their own games. Given the increasing demand by companies, governments, and society for people who know how to think computationally (i.e. think critically, logically, and solve problems in innovative ways using computational tools), in order to be competitive in the knowledge economy (Wing, 2006; National Academy of Sciences, 2010), the Studio K curriculum uses the potential of game design to prepare youth with skills germane to computational thinking, and the so-called STEM disciplines whose practices heavily rest on computation (Games, 2010; Hayes and Games, 2008). This potential has been recognized by the White House's efforts (White House, 2009) to support educational video game design, including national game design contests and supporting programs that teach computational thinking.

General Notes
This workshop will give participants hands-on experience with *Kodu* and a framework for incorporating the curriculum into their classrooms or other learning spaces. Computers and game controllers (optional) will be provided.

Introduction
Thirty years ago, learning to "program" was bracketed off as something specialized for uncool nerds or computer scientists. But as digital devices become integrated into every aspect of our lives, it is imperative that people become literate with digital technologies. This requires more than knowing how to use an operating system or applications—or even how to program—but rather how to abstract from situations and think algorithmically (Wing, 2006). Indeed, computational thinking is essential for success in the modern world as the digital revolution transforms professional practices—even the structure of entire industries—and citizens must understand some computation to participate in modern life. People need to understand what computation can do easily and what it cannot do, develop good intuitions about how computation operates within domains of practice (from social software to personalized medicine applications), and use computation to reach their own goals.

In that sense, computational thinking is defined as the ability to use computation to understand new content, synthesize it, and use it. Classic examples of computational thinking use computation to improve students' abilities in math or science class. A broader view suggests that computational thinking skills can help students read the paper, do their taxes, or participate in political life. The real power of computational thinking, however, lies in that "protean" power of the computer: creation. If understanding computation is valuable, using computation to build something personally meaningful is quite possibly the best way to get there. Computational thinking describes an ability to answer "What can I build to understand this problem?".

There are several paths to acquiring those abilities, most of which involve learning to program and learning to communicate with the logic of programming. While not everyone needs to understand the finer points of red-black trees, almost everyone would benefit from understanding how to create and communicate with computation. However, there is very little instruction in the US that teaches students how to cross-apply computational thinking. Computer science is rarely taught in high schools, and, when it is, it is taught in an disconnected way that misses the point.

Tools like *Kodu* can revolutionize education by building on the successes of precursors like *Logo*, the programming language commonly used in education, but also by employing social gaming structures (badges, achievements, collaborative problem solving, etc.) to deepen participation and encourage players to become game *designers* (Games, 2010). Becoming a game designer means going beyond technical creation to craft aesthetics, interactions, and stories that motivate users to play, and embedding Kodu in a larger social context would allow it to help children to learn to program socially and authentically.

In this workshop, we outline Studio K, an experimental club for teaching students *Kodu*, and *Studio K,* a social network designed to optimize and support learning complex computational content with *Kodu*. The Studio K curriculum is built on a complementary foundation of game design and computational thinking frameworks.

Computational Thinking Framework

Computational thinking is a way of thinking and solving problems effectively using the logical, mathematical, and representational tools that computers make available work our way through data, and transform it into actionable information. Over the years, multiple frameworks of computational thinking have been proposed by scholars, that emphasize different aspects of the construct from the more abstract and logical operations necessary to construct a software algorithm, to the more social aspects involved in solving a complex computational problem collaboratively (National Academy of Science, 2010).

In order to operationalize the construct for this study, we rely on a framework recently proposed by Google (2010), which synthesizes most of these perspectives according to the actual practices of software professionals today. The framework characterizes CT according to five distinct dimensions that encompass habits of mind and practice germane to solving problems using computational tools. These are:

- **Decomposition** is the breaking down, or ability to break down, a problem into its core components. Identifying the core components of a problem often leads to Pattern Recognition and Generalization, which aids in Algorithm Design.
- **Pattern Recognition** is the ability to see or identify recurring themes. With regards to CT, this specifically means that users/students/programmers/etc. can identify these recurring themes outside of the problems that they encounter, which will aid them in applying these patterns to their current problem (see: Pattern Generalization and Abstraction).
- **Pattern Generalization and Abstraction** is the ability to recall previously encountered patterns and use them to aid in the solving of problems of a similar pattern. Although all of the dimensions that Google defines here are important, *abstraction* is a critically important concept in CT due to its cross-field presence and its role in the foundations of some of the most important concepts that we as humans use. For example, not only is abstraction the basis for algebra, but it is also the foundation for language and the way that we interpret and organize the world around us.
- **Algorithm Design** is the construction of a step-by-step process that will allow the user to solve the problem. For example, if someone were trying to navigate a maze, there may be only one solution, in which case an algorithm can be written for the player so that when followed, subsequent players will find the end of the maze, or the goal. It may look something like this:
    - Turn north
    - Move 1 step
    - Turn south
    - Move 2 steps
    - Turn north
    - Move 2 steps
    - Turn west
    - Move 4 steps

  Additionally, using this algorithm, one can reliably recreate this maze, albeit the maze can take on a huge or infinite number of forms, so long as *one* of the solutions follows this algorithm. And while there may exist multiple paths to a correct solution for any given problem, some paths are also *more efficient* than others, which allows for this dimension to be evaluated at increasing levels of sophistication.
- **Data Analysis, Modeling, and Visualization** is the process by which we consume information and transform it into a meaningful form (input to output). A prime example of this is the reporting of statistical trends. Many people struggle with making meaning out of long sentences of numbers and data, so the presentations of those data are often visualized as

charts and graphs, and more abstractly, symbols and icons. These visualizations often carry much more meaning to viewers/listeners/consumers, and are thus more effective ways of communicating messages and information.

We chose these five dimensions given that they present two advantages over those presented in other frameworks: 1) They are measurable, meaning that there are concrete and perceptible processes and products that can be captured by systematic research either through observation or other means. 2) They are applicable to a broad range of professions and activities involving the use of modern computing tools, which in our view is a necessary prerequisite to differentiate computational thinking from other forms of logical and mathematical abstract thought.

Game Design Framework

The Game Design Framework that we use for the Studio K curriculum actually consists of two frameworks, one involves breaking down games, as a medium, into their core components, and the other is the process by which students are allowed to bring in their own gaming experiences.

Dimensions of Games:

- **Goals:** Goals are the objectives of the game. They tell the player what to do, where to go, and how to win the game. By changing the goals in the game, a game designer can change how the players navigate and interact with the game world.
- **Rules:** Rules define how players may behave within a system. In games there are consequences for breaking rules, such as lower score or the death of the player character.
- **Assets:** Assets are all of the physical things that make up the game world itself. The landscape, the buildings, the power-ups, and everything else that the player can see and interact with in the game are the assets. The design of the assets can impact the “atmosphere” and “feel” of the game.
- **Spaces:** Spaces are where the game takes place. By using different types of spaces, like tight and cramped or open and spacious, game designers can affect the kinds of experiences players can have.
- **Play Mechanics:** Play Mechanics are the things that players do in games. They are action words like “run” or “jump”. Giving access to or restricting the abilities or mechanics of players can change the extent to which players are able to interact with the game. This has an impact on how they are able to achieve the goals of the game, or win the game.
- **Scoring Systems:** Even though scores are not present in every game, they can have a large impact on player behavior when they are present. Scores can also be goals in games.
- **Narrative:** The narrative of a game provides the context for the players’ actions. It can also provide motivation for players to continue playing or quit.

Play, Fix, Create

- **Play:** In the first step, students play a game that highlights one of the seven above-mentioned dimensions. The goal is for them to think about how that goal influenced their play experience.
- **Fix:** The students are then presented with a similar, but broken game. Their task is to identify what is broken in the game and then fix the problem.
- **Create:** The students are then allowed to create their own games, keeping in mind the themes of the lesson (Goals, Rules, etc.). However, because starting a game from scratch can be a daunting task, the students are given design scenarios or constraints.

By thinking of games through this framework, students are able to bring in their own gaming experiences and translate those experiences into *Kodu*. In turn, students learn how to think of games as systems, break down those systems into simpler patterns, and then reapply those patterns to new languages and representations.

Studio K

Although only a pilot program, Studio K has thus far been successful in helping students develop ways of thinking about game design that will allow them to transfer their own gaming experiences into

Kodu. This workshop will give participants a chance to walk through *Kodu* and the Studio K curriculum and determine how it may fit into their own learning spaces.

References

Games, I.A. (2010). Gamestar Mechanic: Learning a designer mindset through communicational competence with the language of games. *Learning, Media and Technology, 35(1)*, 31-52.

Google. (2010). What is Computational Thinking? Retrieved from http://www.google.com/edu/computational-thinking/what-is-ct.html

Hayes, E.R., & Games, I.A. (2008). Making computer games and design thinking. *Games and Culture, 3(3-4)*, 309-332.

National Academy of Sciences (2010). Report of a workshop on the scope and nature of computational thinking, Washington, D.C.: National Academies Press.

Papert, S., & Harel, I. (1991). Situating constructionism. In I. Harel and S. Papert (Eds.), *Constructionism* (pp. 1-12). Norwood, NJ: Ablex.

White House. (2009). President Obama launches "Educate to Innovate" campaign for excellence in science, technology, engineering & math (STEM) education. Retrieved from the White House website: http://www.whitehouse.gov/the-press-office/president-obama-launches-educate-innovate-campaign-excellence-science-technology-en

Wing, J.M. (2006). Computational thinking. *Communications of the ACM, 49(3),* 33-35.

Acknowledgments

The authors would like to humbly thank Alex Games and Joe Booth for their support and wisdom. We would also like to thank Oregon Middle School for allowing us to pilot this program in their school, as well as Seann Dikkers, who helped us get set up at Oregon.