IASDR 2019
International Association of Societies of Design Research Conference 2019
**DESIGN REVOLUTIONS**
Manchester School of Art
Manchester Metropolitan University
02-05 September 2019

# MUGEN: Teaching code to design students through game-making

James, Brian[a]

[a] St. John's University, New York City, United States
* hellobrianjames@gmail.com

Teaching computer coding to students of design presents a unique context, with its own set of challenges, from affective factors like motivation and stress, to the cognitive load of coding itself. But design students also bring unique strengths to the table, ripe to be magnified through code. This in-progress study introduces the Mini UnGame ENgine (MUGEN), a custom made software tool with associated pedagogical materials, in an attempt to bridge the gap between computer science research involving the pedagogical applications of game-making, with the the instructional needs of contemporary design classrooms.

***Keywords: pedagogy, game, code, computation, digital, interaction, web***

## 1 Introduction

Teaching computer coding to students of design presents a unique context, with its own set of challenges, from affective factors like motivation and stress, to the cognitive load of coding itself. Such challenges can make even small projects daunting to learners and instructors. But design students also bring unique strengths to the table. They are often highly motivated to learn tools that expand their creative powers, and they have invaluable productive skills such as illustration, photography, and even project management, ripe to be magnified through code.

This case study describes the in-progress development of one pedagogical response to these challenges and opportunities, informed by a strand of game-based education research in the constructionist tradition. The heart of the response is MUGEN: The Mini UnGame ENgine, a simple JavaScript library that allows novice coders to create small game-like experiences. MUGEN offers a flexible learning tool that can support an instructional approach focused on visual design, or an approach focused on coding, or on an approach that balances the two. The present study aims to use this tool and the pedagogical materials developed in conjunction with it to bridge the gap between computer science research involving the pedagogical applications of game-making, with the instructional needs of contemporary design classrooms.

## 2 Review of the literature

The educational potential of video games has been investigated, touted, and commodified for decades. The Minnesota Educational Computing Consortium, for example, began in the

1970s to produce iconic computer games like *Oregon Trail* and *Number Munchers*, which would become cultural touchstones for countless elementary school students in Generation X and the Millennial Generation (Jancer 2016). These types of educational video games follow what can be called an “instructionist” pedagogical model, wherein students learn by *playing* video games whose content imparts information or skills (Kafai 2006). While this has been a popular approach, there is another model of game-centric pedagogy: a “constructionist” approach of learning by *making* games (Kafai 2006). This approach has been explored by researchers of Human–Computer Interaction, Computer Science pedagogy, and related domains. Following a similar path, the present study in design education also adopts a constructionist approach in relation to teaching code.

In their 2016 study, Batista, Connolly, and Angotti conducted a broad review of studies on teaching code through game-making. The dozen papers they selected for discussion reflected a variety of programming languages, and a variety of learning objectives, all falling under the umbrella of text-based programming languages. Moreover, the studies varied by scale: while many were designed around a semester-long intervention, at least one (Sung et al. 2011) focused on much shorter instructional modules. Despite the diversity of these studies’ particulars, Batista, Connolly, and Agnotti noted that the overall body of research indicates that game-making code pedagogy is “a promising strategy to arouse the interest of young students” (2016).

Two of the papers included in that review stand out as particularly relevant to the present study. The first, by Lewis and Massingill (2006), addressed game-making pedagogy in a university-level, second-semester computer science course using the Java language. They created a custom software tool to simplify game programming to a course-appropriate level, and deployed it in a semester-long program of study, designed by the researchers to meet the course’s existing learning outcomes and broader departmental curriculum through a game-making approach. Lewis and Massingill (2006) note that their software tool was created in such a way as to force students to learn how to study software documentation—an intentional design decision to teach this vital skill, and one which the present study was informed by.

The results of their study indicated that the game-making approach tested by the authors was as educationally effective as the standard non-game-making approach, and furthermore that students found the game projects to be more engaging and rewarding than traditional assignments. Moreover, the researchers noted that the programming tool promoted acquisition of the aforementioned documentation-related skills more effectively than other teaching methods they have tried.

The second study, by Sung et al. (2011), described a related but distinct approach to integrating game-making into programming courses. Like the above study, this one also involved a custom software tool, this time in the C# language. As before, the tool was aimed at simplifying programming for students. But in this case, the authors note that the tool was also intended to assist faculty who are interested in game-making pedagogy, but who lack specific game programming expertise. Such explicit consideration of faculty needs sets their study apart from many others.

Another crucial faculty-related consideration was the timescale of the pedagogical interventions: Instead of a semester-long program, this paper tested a series oif seven short, independent assignment modules, which could be integrated piecemeal into traditional curriculum. This was intended to make the benefits of game-making pedagogy more accessible to faculty who, for institutional or other reasons, could not reorient their entire curriculum around games. Additionally, the researchers developed a series of workshops and materials to teach instructors how to use the software tool, as well as how to deploy the game-making modules alongside traditional classroom instruction.

Note that, due to certain pedagogical factors—and perhaps also the short timeframe of the modules—the games created in this study were quite simple. They were also not particularly entertaining, as the authors themselves note. For these reasons, Sung et al. describe the products of these assignments as "game-themed," and "real-time interactive graphics programs," rather than as proper "games" (2011). For the purposes of the present study, that distinction is important. We may observe that Sung et al. did not propose a course in game design, but rather they used video games as a point of departure from which to frame their assignments. That gesture will return in the present study's method as well.

Sung et al. (2011) deployed the tool and modules into standard computer science courses, mixing them into the established curricula. Although post-test surveys revealed some student dissatisfaction with the limited nature of the games, compared to students in control courses, students in the game-making groups exhibited higher rates of course completion, and higher assignment scores, even while spending less time on some assignments. Returning to the faculty considerations mentioned above, the post-test confirmed that the modules did not require more effort to implement than did traditional assignments, despite the test instructor's lack of experience with game-making or graphics programming.

## 3 Research gap and question

The aforementioned pedagogical theory of constructionism, which touts the importance of making tangible artifacts during the learning process, seems like a natural fit for design education. Previous studies on constructionist pedagogical approaches of learning code through making games also seems highly relevant to design educators.

A major limitation in the research, however, is the overwhelming focus on computer science as the disciplinary context of learning to code. While programming may once have been rarely taught outside of computer science courses, in recent years coding has become a mainstream tool in other disciplines, including design. Thus, the review of the literature presents exciting pedagogical possibilities for design educators who engage with code, however it also reveals a dearth of related research performed in the context of the design classroom. That gap offers a potentially fruitful and transformative area to build upon the findings of previous research, but in the novel context of design education.

This paper presents an in-progress report on an exploratory case study that attempts to fill the current research gap by developing a constructionist, game-making pedagogical package that can be used to teach coding in the context of a design curriculum. The current phase of the study is addressing the questions: What shall be the requirements and components of such a package? How can its effectiveness be tested?

## 4 Methods

### 4.1 Development context

The design of the pedagogical package is informed not only by the research precedents described above, but also by the unique context posed by the design classroom as a site of learning code. Inasmuch as this project is in its initial stages, development has begun in response to conditions in one particular design program, with the intent to abstract and adjust various elements of the package as additional field tests are completed in other locations.

This project is being developed alongside a web design course with an enrolment of eight students, offered in the Graphic Design program at a large private university in the Eastern United States. It is the second in a two-semester sequence of web design classes, and enrolled students had achieved intermediate levels proficiency in HTML and CSS, and very cursory familiarity with JavaScript.

### 4.2 Design

The design of the pedagogical package will be described in terms of (a) its requirements and (b) its components. These requirements and components were chosen as important design considerations based upon the precedents covered in the review of the literature, as well as the novel design context of the present study.

The following requirements were identified based upon the review of the literature:

- produces a functional, publishable design artifact
- supports short-term, low-stakes projects that are suitable for testing in a variety of classes
- usable by instructors with varying levels of coding expertise
- produces simple game-like works, not oriented around game design per se

Several additional requirements were formulated in response to the design context:

- offers meaningful engagement for coders of various skill levels
- allows learners to apply visually creative skills
- uses web languages (JavaScript, CSS), which are easier and commonly used by designers

The following package components were proposed, based on the requirements identified above, as well as the research precedents:

- software tool to simplify coding (MUGEN)
- robust documentation for learners to gain experience using
- game demonstration with functioning code and visual assets
- lesson planning materials for instructors

A final component was resolved upon based on the unique design context: an Adobe Illustrator file to act as a template for designing visual assets like characters and a background scene.

### 4.3 Early development

Inasmuch as the software tool would enable the rest of the package to function, development began from there. The author created a small JavaScript library called MUGEN (Mini UnGame ENgine), which enables users to quickly produce a single-screen, interactive, game-like experience. Note: The terms “UnGame” and “game-like” acknowledge the simple nature of the pieces produced, in the spirit of Sung et al. Hereafter, however, these pieces will simply be referred to as games for the sake of convenience. Each game consists of three visual elements: a player-controlled character, a computer-controlled character, and a background scene. Structurally, the game is in platform format, where characters are viewed from the side. Players can move their character left or right, jump, or perform a key action.

In functional terms, MUGEN handles advanced coding tasks like handling player input such as clicks and button presses, and interactively applies visual styles and animations—designed by the learners through their visual assets and with relatively simple CSS code. Optional features allow advanced learners to attempt more complex tasks, such as programming custom functions triggered by certain in-game events.

A fully functioning demonstration game is distributed with the library. This demo not only shows how to use MUGEN, and how the visual assets are to be organized, but also provides a hackable starting point for novice coders. Beginners might try tasks as simple as replacing the demo’s visual assets for characters and backgrounds with ones of their own design, and/or tweaking settings like character speed, or jump height, in the existing code.

A GitHub repository was then created from which to publish the package. GitHub is a web platform that supports collaboration on software development and distribution, as well as publication of documentation. Along with the tool and demonstration, software documentation was posted for learners to reference as they learned how to use the tool. The platform also allows users to report bugs, and to request new features.

The package was developed to the point described above, and was deployed in the researcher’s class as an early pilot test, the results of which will guide further development. The test was conducted as a two-week project to create a simple game to the specifications described above. Once again, it is important to note that the assignment was not an exercise in game design, but rather an opportunity for students to practice code by simplifying complex tasks and allowing them to integrate their visual skills into an interactive project, while building peripheral skills like using documentation.

## 5 Tentative findings

While formal pre- and post-test impact assessments are forthcoming in a subsequent phase of the study, initial student response to this pilot phase seemed positive. Students filed multiple feature requests for the software tool, indicating a degree of motivation and interest in the project. One student verbally reported satisfaction at the opportunity to practice a new kind of illustration for the first time while making visual assets for the game. From the instructor’s perspective, students seemed to spend more time on visual making for this assignment compared to other assignments in the same class.

Significant limitations, however, also exist. Some students expressed dissatisfaction with the small scale of the game, and with the lack of ability to create different kinds of games. Also, the interaction between the player character and the computer character was framed in terms of player–hero versus computer–monster in the assignment brief, which although a convenient shorthand, tended to limit designs to this simplistic and combat-oriented metaphor. While not necessarily a fatal flaw, it was realized in retrospect that this framing is both limiting to the students' design thinking as well as potentially off-putting to some students. This was truly a missed opportunity, as the MUGEN tool itself could likely support many other kinds of interactions. This realization underscores the importance of the next phase of the package's development: lesson planning and support materials.

## 6 Conclusions

The above findings revealed two key points that serve as a conclusion to this initial phase, and as feedback to inform future development of this study. First, student responses to and perceived engagement with the MUGEN tool indicate the value of continuing to develop this approach to teaching computer coding in the context of design. These responses also suggest several areas of focus for testing impact assessment in the next phase, where the researcher plans to compare student experience of "typical" (non-game-related web design projects) to the game-based project of this study. Namely, tests will assess: degree of motivation to complete the assignment, degree of satisfaction with the completed assignment, acquisition of intended code learning outcomes, and—perhaps unique to the present study's context of design pedagogy instead of computer science—proportion of student work time devoted to design versus coding, and acquisition of intended *design* learning outcomes. If, as initial observations suggest, students were able to spend more time on design thinking for this project than on their typical projects, the approach explored by this study may allow students to place more emphasis on practicing their visual design skills while still learning the coding objectives.

A second key finding is the limiting nature of the game scenarios supported by the MUGEN tool as presently constructed and administered. If possible, future rounds of development will aim to increase both the flexibility of game scenarios, and the scale or complexity of those scenarios. Regarding flexibility, as noted above, the project brief could probably be modified with minimal or no technical change to the MUGEN tool itself in order to promote other concepts of interaction aside from combat. Increasing the scale and/or complexity of scenarios may also increase student motivation, and it could support additional learning objectives around constructing visual narratives, interaction design, and so on. However, this would require great consideration so as not to undercut the simplicity and ease of use which the students and instructor experienced in the initial test phase, and which Sung et al. foregrounded in their aforementioned study (2011). As such, the scale/complexity aspect of development may best be left for later stages.

## 7 Next steps

The next step of MUGEN's development will focus on completion of the final component specified in the package design: lesson planning materials for instructors who want to use MUGEN in their classes. Toward that end, the following materials are currently in development or contemplation:

- Clear, brief written documentation of all the components of the package
- Brief tutorials/workshop plans for the instructor to learn MUGEN, especially for instructors with little or even no coding experience
- Lesson plans for a variety of coding levels, and ideas for emphasizing different aspects of design through MUGEN, ranging from character design and animations to more advanced JavaScript programming
- Suggestions for instructors on maximizing the flexibility of MUGEN, given its simple parameters and functionality

Further pilot testing will continue in design classes taught by the author in subsequent semesters. Additionally, design professors at multiple institutions have expressed interest in participating in further field testing once those lesson planning materials are complete. These future pilot tests will measure student response with pre- and post-test surveys of the students as well as the faculty, modeled after those of the studies discussed in the review, but informed by the insights discussed in the Conclusions section of this study. Quantitative measures of hard coding skill and design outcomes is under consideration for those tests, however this may be more difficult to meaningfully assess in the relatively subjective environment of the design classroom compared to that of computer science.

## 8 References

Batista, A., Angotti, J.A.P., & Connolly, T. (2016, October 6, 7). *A Framework for Games-Based Construction Learning: A Text-Based Programming Languages Approach*. Paper presented at 10th European Conference on Games Based Learning. Paisley, Scotland: Academic Conferences and Publishing International Ltd.

Jancer, M. (2016, July 22). How You Wound Up Playing The Oregon Trail in Computer Class. *Smithsonianmag.com*. Retrieved from smithsonianmag.com/innovation/how-you-wound-playing-em-oregon-trailem-computer-class-180959851

Kafai, Y. B. (2006). Playing and Making Games for Learning. *Games and Culture, 1*(1), 36-40. doi:10.1177/1555412005281767

Lewis, M. C., & Massingill, B. (2006). Graphical game development in CS2. *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education - SIGCSE 06*. doi:10.1145/1121341.1121499

Sung, K., Hillyard, C., Angotti, R. L., Panitz, M. W., Goldstein, D. S., & Nordlinger, J. (2011). Game-Themed Programming Assignment Modules: A Pathway for Gradual Integration of Gaming Context into Existing Introductory Programming Courses. *IEEE Transactions on Education, 54*(3), 416-427. doi:10.1109/te.2010.2064315

**About the Authors:**

**Brian James** is a design educator with industry experience ranging from local institutions to global brands. His academic work aims to enrich the language of graphic design by building bridges of dialogue to adjacent domains through technology.