Design Games: Rethinking Architectural Design Software

AARON WESTRE
University of Minnesota

INTRODUCTION

How is the architectural design process like a game? How could video gaming influence architectural design? In Design Games, a course taught by the author, students explore the world of digital game play and propose new modes of designing based on their findings. Using a simple programming toolkit, they design and build software that presents new ways to see the design process. The course relies primarily on process-based instruction as a mode of inquiry, utilizing computer programming as a primary medium. The majority of learning happens in programming workshops in which students and the instructor collaboratively translate ideas into computer code. In addition to the process-based instruction, students engage in research into digital game play and human-computer interfaces.

The final product of the course is a set of software prototypes that present playful ways for designers to create using computers. These applications take the form of design games, a speculative type of software that combines the engaging qualities of video games with the productive qualities of available digital design tools. What follows is a description of the structure, tools, and outcomes of the course.

COURSE STRUCTURE

The course is taught as a seven week (half- semester) workshop as part of the school's Bachelor of Design in Architecture (BDA) program. The core of the BDA program consists of a number of these workshops, taken in the final two years, on topics as varied as critical writing, furniture design, and photography. Workshops are intended to offer a broad education in design thinking through the lens of architecture.

Design Games starts with a research assignment. Each student is asked to select a video game and describe three aspects: premise, play, and interface. The first aspect consists of the narrative of the game and its visual environment. Play refers to the rules of the game and how the player negotiates the environment to reach the goal. Interface includes physical devices that the player manipulates, as well as feedback cues that the game provides in the form of graphics and audio. After a thorough description, students are asked to identify an element of the game and speculate on its potential uses in the architectural design process.

The primary project of the course is the team- based development of the design game itself. The term "design game" is left intentionally vague to allow for a broad exploration, except for the stipulation that it be simple, engaging, and instrumental. This is to say that a design game should be engaging in same way a video game is, but have as its outcome some concrete design product. Teams first sketch out their concept using storyboards, diagrams, and writing. In this phase, teams are asked to define four characteristics of their design game: environment, objects, interaction, and product. Environmental issues include the definition of the space of play along with other factors such as scenery. Objects include all elements of the game that the player can influence or that have their own capacity to act. Interaction defines how the player causes

change in the game, specifically with respect to the active objects. Product refers to the final design outcome that can be taken away from the design game, whether this is a 2D drawing, 3D model, or some other outcome such as a strategy or educational lesson. The product is the primary aspect that differentiates a design game from a traditional video game. While the product can take on a variety of forms, its essential purpose is to function as a seed for further design exploration.

After two initial programming lessons, teams proceed to write the code for their design games. Code writing happens through team- based exploration and through feedback from the instructor in which resources and techniques are suggested. Each class session leads to a refinement of the design game concept and a gradual increase in the functionality of the software. Collaboration between teams, in the form of code and resource sharing, is encouraged throughout the course. In this way, the class amasses resources and techniques through the instructor, online resources, and books. This collaborative model is similar to a traditional design studio, with the exception that it is dependent upon a different body of technical knowledge. Similar to a traditional design studio, failure with demonstrable lessons learned is valued as much as unequivocal success.

SOFTWARE TOOLS

Previous incarnations of the course have used Processing, a software development tool geared toward novice programmers in the art and design fields. In the most recent class, Unity was adopted as the primary development tool. Unity is a software development tool built by Unity Technologies specifically for the creation of 3D, interactive video games. In relation to Design Games, it exhibits many advantages over traditional software development tools.

The primary advantage is that Unity is asset- focused rather that code-focused. Traditional software development happens primarily in tools that look very much like text editors. The developer writes code and then executes it to see the result. This method of programming allows for flexibility, but requires the acquisition of some programming skill before any results can be seen. In previous iterations of the course this learning curve has been the largest obstacle to progress, since the students have no prior training in computer programming.

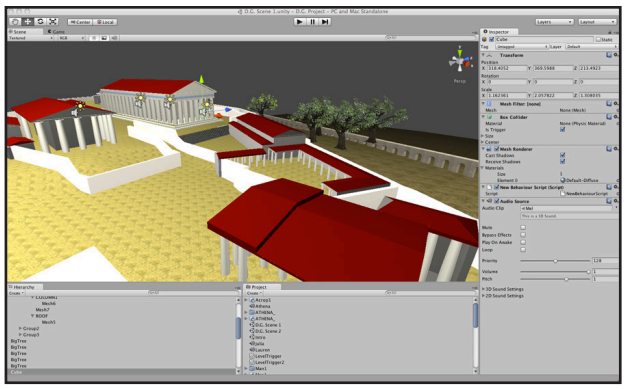

Figure 1: Unity interface

The Unity development environment, by contrast, presents an interface similar to many 3D modeling tools with which students are already familiar.

Games are built up by importing 3D models, images and sounds, then positioning those assets in the 3D space of the game world. These operations happen in the usual drag- and-drop, visually-oriented manner to which the students are accustomed. Many complex features such as physics-based motion, lighting and sound can be accomplished without writing any code in Unity.

When coding is necessary, it occurs at a local level. For instance, if the player should be able to pick up an object, a script can be written that defines this behavior and then attached to the object. This script can then be attached to any object in the game world that requires this behavior. This reuse of code and other assets allows students to develop sophisticated interactivity in a short amount of time; and by learning only the coding skills necessary to achieve it.

Unity, like other popular development tools, has a large community surrounding it which provides tutorials and answers to technical questions. These resources were a critical component in the success of the course.

OUTCOMES

Work carried out by students in the course far surpassed the initial expectations of the author. The success of the students, despite their lack of experience, was most likely the result of three factors. First, architecture students are encouraged to de-

velop robust creative problem-solving skills through their coursework. Additionally, the students enrolled in Design Games, due to the broad scope of the BDA program, may be especially willing to engage in an activity such as software development without initially seeing its direct implications for design. Second, the use of video games as a framework for thinking about design software had an immediate engaging effect upon the students. Of the students enrolled, approximately half did not regularly play video games. Despite this fact, the concept of combining video gaming with architectural design was overwhelmingly cited as the primary reason for enrolling in the course. Third, the informal, iterative software development process used in the course encouraged experimentation and research. Through feedback sessions between the instructor and students, the complex task of writing a software application was broken down into manageable components. As students solved each problem and their software became more and more functional, confidence increased fueling further exploration.

The design games produced in the course were diverse in terms of graphic style, game play, and design product. Some explored rather conventional design issues such as space planning or knowledge-testing, while others implemented more abstract, experimental kinds of form-making. However all of the design games incorporated concepts of play from the video game world to arrive at new processes that either re-envisioned traditional design tasks or enabled a new type of design. Below are summaries of four design games prototypes produced in the course that demonstrate the variety of the projects undertaken.

Figure 2: History's Mysteries by Alison Hagberg, Kristin Lundquist and Ashley Wright

History's Mysteries is an educational design game in which the player's goal is to collect the lost pieces of the Parthenon and reassemble the famous structure. Guides offer the player advice about the actions they need to take to complete their task and learn about the construction and history of the Parthenon.

Figure 3: Egress by Zachary Keilholz, Elizabeth Poppe and Jeffrey Trechter

Egress is a design game aimed at helping architects evaluate the accessibility of their designs. The player uses a wheelchair interface to navigate the architectural design and their progress is tracked. At the completion of the game, visual feedback is given to give the designer cues about problem locations that might hamper egress in an emergency situation.

Figure 4: Junkitecture by Adam Lapacz, Ryan Radzak and Jacob Sevelius

Junkitecture is a design game in which the player finds themselves in a junk yard filled with a variety

of discarded objects. The player can pick up objects and attach them to other objects. Objects bind to each other in an automatic way when in range of each other, allowing the player to construct a variety of structures in a short time.

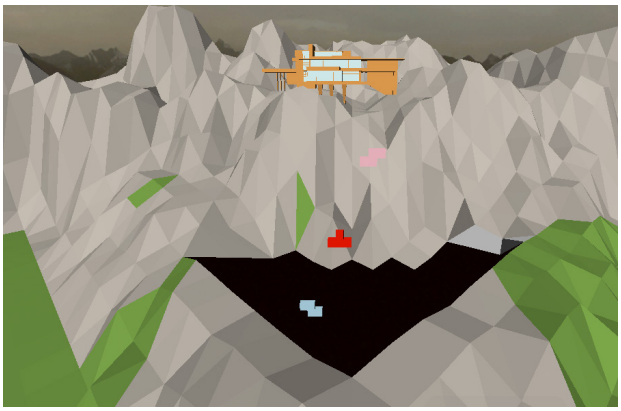

Figure 5: Architekris by Kyle Johnstone, Jared Latterell and Michael Wood

Architekris is a casual design game intended to give architects a tool that acts both as a stress reliever and an inspiration machine. The player is presented with a cascade of shapes which they can fish for and grab with an accelerometer-enabled controller. With the controller, the player can throw these shapes into each other to create unique sculptural forms.

CONCLUSIONS

Critical to the success of Design Games was the use of video games as a platform from which to explore software development and as a critical lens through which to consider the state of digital design tools. Traditional instruction in computer programming involves a formal sequence from basic syntactical rules to complex techniques. Harvesting ideas from video games gave the students an immediate sense of the types of visual effects and interactivity that were possible. This library of ideas, combined with informal feedback and iterative development, yielded a process that bypassed formal instruction. Software developed quickly as students collected code and techniques from the instructor and the game development community. In-depth understanding of specific code syntax developed as it was necessary to accomplish particular tasks, rather than in a particular formal sequence. Since the primary goal of the course was to build software as a way of speculating about new kinds of design tools, it was not crucial that the students had a complete education in programming. The loose structure of the course allowed each team of students to construct their design games to a much greater level of functionality and visual sophistication than would have been possible in the seven weeks available if more formal instruction had monopolized the time.

In addition to the technical success of the software projects, there were two underlying lessons that students learned. The first relies on the fact the programming languages require a particularly precise level of execution. Code either works or it doesn't – there is no partial success. By learning a bit about programming, design students learn a way of thinking that emphasizes a very explicit form of problem- stating and decision-making. Coupled with an informal, iterative process, this way of thinking encourages continual restatement of problems and refinement of solutions until satisfactory outcomes are achieved. In this way, programming can expand the modes of thought available to the designer by offering a method to clarify fuzzy ideas and simulate their implications.

The second lesson involves the relation of designers to their digital tools. Design Games reforms this relationship from one of consumption to one of production. By developing basic software development skills, students understand that they need not rely on software companies entirely to provide them with the digital tools they use. Once this realization sets in, not only can students build their own software, but they can start to think critically about the tools currently available. Critical thought about digital tools can be developed just as criticism of design methods is taught in architecture programs.

In the end, the success of the course was evinced by a distinct change in students' attitudes toward software. In seven short weeks, students who had no previous programming experience made the empowering discovery that they could not only use software, but actually produce it themselves. By using the enticing framework of video gaming and informal, pragmatic instruction, the course presented the challenge of making software as a simple, non- threatening task. This structure, coupled with the students' enthusiasm and creativity, led to a great deal of research, learning, and creativity. The projects that the students carried out far surpassed the expectations of the author

in terms of functionality, variety, and imagination. The recombinant nature of the course, importing concepts from the video gaming world into design discourse, offers a model for engaging and productive curriculum. This model has long been used to migrate ideas from art, science, and culture into architectural education. In Design Games it serves to empower students with a new set of skills, enabling them to construct their own digital tools.