MOOT: AN EXPLORATION OF DIGITAL DESIGN AND CULTURE THROUGH MOBILE APPLICATION DEVELOPMENT

Erin Bleiweiss

TC 660H
Plan II Honors Program
The University of Texas at Austin

May 11, 2016

______________________________
Robert J. Quigley
School of Journalism
Supervising Professor

______________________________
Ramona Broussard
School of Information
Second Reader
ABSTRACT

Author: Erin Bleiweiss

Title: Moot: An Exploration of Digital Design and Culture Through Mobile Application Development

Supervising Professor: Robert J. Quigley

This thesis explores the design and development of a mobile application for iOS. The app — called Moot — is a puzzle game that requires users to execute specific interactions with a digital space. Certain levels of the game require users to scan product barcodes in order to solve puzzles. Other levels require a certain amount of disengagement with the mobile device, such as stepping away from the screen for a specified duration of time. The common theme is that no puzzle can be solved using the app alone.

There is a commonly perceived disconnect between the virtual realm and what is colloquially referred to as “real life.” We tend to create divisions by constructing digital identities, and in general considering our online relationships and interactions to be non-genuine. However, the Internet plays a significant and inseparable role in our culture, making these divisions problematic.

Typical case studies in design explore the relationship between form and function, examining how individual design principles aid the user towards achieving a goal. Contrastingly, this app’s design is not rooted in efficiency, but is rather a satirical exercise in closer examining our society’s relationship with technology and digital media.
Acknowledgments

This thesis is the culmination not only of my undergraduate studies, but also of a personal journey. I am forever grateful to the following individuals for their knowledge, encouragement, and support.

To my supervising professor, Robert Quigley, whose humor, creativity, and enthusiasm has inspired me for the entire duration of this project. I thank him for mentoring and motivating me throughout the development process, and for always pushing me to exceed my goals.

To my second reader, Ramona Broussard, for helping me to navigate through big ideas, to organize my opinions, and to contextualize this thesis in a meaningful way.

To my mom, Judy Bleiweiss, for reading through countless drafts, making sense of my thoughts, and always being available for a phone call.

To my dad, Freddie Bleiweiss, for being the source and inspiration of my nerdiness, for being my own personal server administrator, and for instilling in me a sense of creative problem solving.

To my sister, Rikki Bleiweiss, for her incredible intelligence, quick wit, and uncanny ability to always know what I’m trying to say before I can manage to say it.

To my roommates, Allyson Rosenthal and Renae Jackson, for listening to me ramble and rant, play testing my app, giving me feedback, and supporting me through all of the ups and downs of this process.
# Table of Contents

“Real Life” ................................................................................................................................. 5

Moot: The Game .......................................................................................................................... 8

What is a Game? .......................................................................................................................... 10

Gamification ............................................................................................................................... 12

The Internet is Not A Game ........................................................................................................ 21

Behind the Scenes: A Reflection ............................................................................................... 24

Appendix: Random Recursive Maze Generation .................................................................... 33

Download and Source Code ...................................................................................................... 39

Bibliography ............................................................................................................................. 40

Technological Attribution ........................................................................................................ 41
“This is all a moo point. It's like a cow's opinion, you know, it just doesn't matter. It's 'moo.’”

– Joey Tribbiani, Friends

“Real Life”

My relationship with technology is complex, and I know that I am not unique in this experience. I rely on my phone to receive my email when I’m away from my computer. It is my alarm clock to wake me up in the morning and my personal assistant to collect my messages when I’m asleep. It lets me text my friends to make plans, and update my calendar with new events. I might forget about an assignment due Friday or a meeting Monday morning, but my phone's seemingly infinite wealth of storage never fails to remind me. My memories of high school may have faded, but my friendships are preserved on Facebook, even if artificially, through archives of photos and messages. My phone has become an integral part of my life. When I say I "can't live without it," the statement probably holds more true than I ever intended.

As part of a conversation about digital security and identity, internet creator and educator CGP Grey offers the following insight about mobile phones:

“Thinking of your phone as an extension of yourself isn’t crazy. To say that your phone knows more about you than you know about you isn’t an exaggeration. It's a statement of fact. Do you remember your location every minute of every day? Do you remember what you said to your friend last leap day at 10:47 word-for-word? Yeah, of course not... Since you bought it, how many hours has your phone been more than an arm's reach away? Possibly zero. There's no other object like that in your life. Given the choice to have someone read your mind or read your phone, if you seriously think about it, you'd probably pick the former. Compared to what's in your phone, your brain holds a tiny amount of information — much of it wrong, all of it lossy. It’s easy to forget what kind of embarrassment your phone contains, because it has so much you can't even remember.”

1 CGP Grey, 2016.
Despite this deep connection, I am keenly aware of how frequently and assertively we try to separate ourselves from the digital realm. There is an often-stated question that I myself am guilty of asking, yet it is one of my biggest personal annoyances. Far too frequently in everyday conversation, I notice the phrase “Did that happen online or in real life?” We place so much importance on the means by which a conversation takes place, the nature of a friendship, or the origin of an argument that this phrase is usually unremarkable. But in a culture where technology is so interwoven in our lives, I find it problematic that we make this distinction. That a friendship is sustained largely or even exclusively on the internet should have no bearing on its perceived strength or legitimacy. That a dialogue takes place via a text message rather than by word of mouth shouldn’t make that conversation less valid or credible. To suggest that interactions had online are inauthentic or inferior to those offline is to discredit one of the greatest societal innovations of our lifetime, and to ignore an enormous facet of our culture. Distinguishing any non-digital experiences as “real life” implies that the internet is somehow separate from reality. This division is not only personally frustrating, but also the origin for potentially dangerous discourse.

Because of my simultaneous love for technology and skepticism of its impact on our culture, I sought to develop an app that would explore our relationship with technology, as well as how we interact with digital spaces. The app — called Moot — is a puzzle game that requires users to execute specific physical interactions within a digital platform. I developed Moot for iOS (Apple’s mobile operating system), and it is available for both the iPhone and iPad. In each level of my game, I focused heavily on the user’s experience. When designers and developers evaluate user experience, they are typically concerned with the relative success of a product in terms of the needs of their target audiences. I did not develop Moot
with the intention of it ever being marketable, and I also did not expect or intend to profit off of it. Rather, it is an exploratory venture. The app is designed to be a satire, criticizing how our culture treats technology. We tend to take our interactions with digital devices for granted. Despite how thoroughly we have integrated them into our lives, we treat digital spaces as completely disconnected and inconsequential. Moot unites the experiences of both the digital and physical realms, and encourages a deliberate and critical evaluation of how we use technology.

Figure 1: Moot’s login screen, presented when a user first launches the app.

Figure 2: Moot contains four levels, which can be accessed from the main menu.
Moot: The Game

Moot consists of four levels, each of which presents a slightly different interaction between digital and physical space. Level 1 is a word game, similar to hangman. In order to advance, players must guess the correct set of letters in order to spell a series of randomly generated four and five letter words. Before a user can input a guess, they must first scan the barcode of a product whose name begins with the desired letter. For example, on the first screen, users are presented with a set of tiles spelling “SC_N” (Figure 3). In order to progress past this stage, a user might find a bottle of Aquafina water. After scanning the barcode on the water bottle, the game would register the letter “A” as an input, thus filling in the blank and completing the stage (Figure 4).

Level two functions similarly, with barcode scanning as the main mechanic, but it begins to focus on space and color. Instead of a word game, this level presents a series of simple mazes. A user must navigate a token from the upper left corner of the maze to the bottom right by tapping directions on an on-screen compass. There are eight possible directions the token can move, each corresponding to a color on a color wheel. At the start of the level, all eight directions on the compass are locked. When a user scans a barcode, the game will determine the color of the product and unlock the appropriate segment on the compass. For example, if a user scans a bottle of French’s Yellow Mustard, the yellow segment on the right-hand side of the compass will be illuminated. By pressing on this segment, the user can now move the token to the right (Figure 5).
Figure 3: In Level one of Moot, players must guess letters to complete the word.

Figure 4: A player scans a package of Aquafina water in order to guess the letter “A.”

Figure 5: In Level two of Moot, a player uses the color of products to navigate through the maze.
Because the game is somewhat experimental, I wanted the user’s experience to be exploratory as well. For this reason, the game contains no explicitly written instructions. A user might gain hints from on-screen cues and iconography, such as the camera button, or by exploring the game’s achievements or “about” page, but unless they seek out this extra information, they will not be presented with any suggestions about how to play. I anticipate that most users of Moot at least own a smartphone, understand basic game mechanics, or have played puzzle games before. Most levels feature familiar game mechanics, which should not be new to them. I was not focused on making Moot intuitive, pleasing, or even fun. In fact, I would expect complaints that gameplay appears tedious and arbitrary. Many mobile games are intended to be somewhat mindless. They offer a desirable experience by providing the player with a distraction, or a sense of escape. With Moot, a player must always be engaged.

The game is not sedentary in nature, as the player must engage in a sort of scavenger hunt in order to obtain the diverse array of barcodes necessary to play. Instead of providing the user with an experience that is fun and rewarding, I hoped to provide the user with a thought experiment. At the very least, I want the player to be fully aware of the actions required to progress through the level. If the player gets frustrated, and begins to question exactly why they are spending their time this way, then from a creator’s standpoint, I would consider the experience a success.

What is a Game?

Moot doesn’t exemplify the properties we typically associate with gameplay. It fits more closely within the category of a game than it would within other categories of mobile
apps, such as productivity, business, or music, but it wasn’t designed with the intention of being fun. Whether something is fun is difficult to objectively measure, so it is advantageous to establish some other criteria that identify and distinguish games. In 1938, Dutch cultural theorist Johan Huizinga suggested five characteristics that define gameplay\(^2\).

First, games must be “free.” A game is a voluntary experiences that provides a sense of freedom and autonomy by giving its players some degree of control over the outcome. A game isn’t a constructive task constituting work, but is rather something superfluous done for leisure. The barcode-scanning element in Moot provides the sort of autonomous freedom Huizinga refers to, while the total fruitlessness of its objectives ensures that its gameplay is done for more of an enjoyment than a need. Even if the app seems tedious, it is still clearly closer to play than work.

Second, a game must be unordinary. Games are separate and distinguished from “real life” experiences, as they put us into a temporary unique and imaginative space. Activity within this space should be inconsequential beyond the confines of the game. Thus, digital games occupy a unique place in the digital sphere. Although technology is unavoidably part of our real lives, digital games allow us an opportunity to “ste[p] out of ‘real’ life into a temporary sphere of activity with a disposition all of its own.”\(^3\)

When Huizinga talks about this temporary space, he uses the term “magic circle,” which encompasses the third and fourth characteristics of gameplay. Games are both limited and ordered. Games have boundaries, both spatially and temporally. They often have clear definitions of a physical limited space (such as the boundaries of a football field), or rules that impose limits on the game’s timespan (such as a clock keeping track of four 15-

\(^2\) Huizinga, 1949.
\(^3\) Ibid.
minute quarters). Games are ordered in that they operate with a specific set of rules. If at any time you break those rules, the game is over and can no longer function. Figuring out the best strategies to operate within these limits is usually what adds to a game’s challenge and appeal.

Finally, the fifth defining characteristic of games is the sense of exclusiveness among those who participate. Players might be on a team or part of a club, and are all united by certain customs, laws, or rules that are valid only within the magic circle. Huizinga refers to a “feeling of being apart together” as players “mutually withdraw from the rest of the world.” Of all of these characteristics, Moot challenges the idea of exclusivity the most. When playing Moot, users must cross the boundaries of the magic circle in order to advance. The first two levels require the user to directly interact with their phone, and integrate it into their life by scanning product barcodes. The goal is to make playing the game a little bit more difficult than doing something like sitting on the couch playing Tetris. Because of the extra effort necessary for each interaction, a user will be more aware of what they do while they’re doing it.

**Gamification**

Moot is not the first game to challenge the boundary of Huizinga’s definition. If analyzed according to Huizinga’s characteristics of play as defined above, then Moot certainly qualifies as a game. Yet, the game feels tedious, and doesn’t seem to have the same kind of entertainment value as most other games we’re used to. By examining the popular health and fitness app SuperBetter, as well as the satirical Facebook game Cow Clicker, we

---

4 Ibid.
can begin to see an emerging trend of games that push the limits of their more traditional predecessors.

SuperBetter was created by Jane McGonigal, who writes and speaks in the field of gamification. She believes that we should focus on spending more time playing games, as games can be a force for great change.\(^5\) The idea is that by “gamifying” our lives and turning ordinary tasks into games, we can more effectively achieve our goals and ultimately be more successful.\(^6\) SuperBetter, along with most lifestyle or productivity based games, is very mundane in its gameplay elements. In these games, you receive points and achievements for things like exercising, or eating healthy, or walking your dog.\(^7\) These types of objectives tend to dilute any elements of strategy or challenge that are typical of gameplay.

Gamification has expanded rapidly in recent years, even beyond SuperBetter’s “gaming for good” model. Companies are starting to take advantage of game-based marketing in an effort to better engage their customers. Promotions like frequent flyer programs and loyalty clubs are just a few of the ways that businesses hope to excite their clients and keep their attention. A 2011 report from the Gartner Enterprise Architecture Summit suggested that more than 70% of companies would begin to gamify divisions such as marketing, innovation, training, employee performance, and more over the next three years.\(^8\) Furthermore, it is estimated that employee-focused gamification is even more prevalent than gamification for customer engagement. Some companies, such as Walmart, gamified their safety training procedures in 2013. They claimed that the addictive and competitive nature of this new training process helped produce a 54% decrease in workplace

---

\(^{5}\) McGonigal, 2010.  
\(^{6}\) McGonigal, 2012.  
\(^{7}\) SuperBetter.  
\(^{8}\) Gartner, 2011.
incidents among eight different distribution centers.\textsuperscript{9} A study by M2 Research Group predicted in 2011 that gamification would become a $2.8 billion market by 2016.\textsuperscript{10} The motivations behind this trend are varied. Many cite gamification as an effort to reach the younger generation, who respond well to engaging interactive experiences, but are not as responsive to older marketing techniques. Others point to dramatic increases in productivity and innovation when gamification is used as a tool among the company’s employees. It is almost certain that gamification will continue to rise in popularity and find its way into more sectors in the future.

By contrast, Ian Bogost takes a different approach with the game Cow Clicker. Cow Clicker was a Facebook game that became a viral phenomenon with over 50,000 users in 2010.\textsuperscript{11} The gameplay is comprised almost solely of logging onto Facebook every six hours to click on a virtual cow. Unlike SuperBetter, which was created out of a real belief in the power of “gaming for good,” Bogost created Cow Clicker as a satire of the emerging genre of social games. These kinds of games are designed to take advantage of the user, and for the user’s gameplay to inherently promote the success of the game.\textsuperscript{12} If you want to succeed, you have to take advantage of your friends by sharing your cow-related achievements and getting them involved in the game as well. The game also creates extremely compulsive tendencies, as it is requires a task to be done every day at regular intervals, just like checking our email or refreshing our Twitter feed. It is designed specifically to squander our time.

\textsuperscript{9} Meister, 2015.  
\textsuperscript{10} Meloni, 2011.  
\textsuperscript{11} Bogost, “A Facebook Game”.  
Figure 6: SuperBetter presents the player with a list of “powerups” they can earn by completing lifestyle goals.\textsuperscript{13}

Figure 7: Cow Clicker’s interface on Facebook, circa 2010, in which users can click on their own cow or view their friends’ cows.\textsuperscript{14}

\textsuperscript{13} SuperBetter, 2015.

\textsuperscript{14} Bogost, “A Facebook Game”.

15
Although Cow Clicker differs widely from SuperBetter in its motivation, design, and objectives, they both share a similar challenge-free model when it comes to game tactics. If taken to an extreme, SuperBetter fails as both a model for productivity, and as a game. While SuperBetter may be effective in motivating its users to accomplish a few small tasks, it cannot serve as a universal motivational tool by which we make our life decisions. By gamifying our personal resolutions, we change the objectives from something that is personally fulfilling to something that is very impersonal. Gamification takes away both the burden and the consequences of our actions, as we trade self-gratification for points and badges. Ultimately, both completing and establishing these objectives are completely up to the user. The repercussions of failure become a lot less meaningful, as do the celebrations of success. Simultaneously, SuperBetter also fails as a game. To attempt to “play” SuperBetter, players must consider their phones as digital magic circles, with implications that are exclusive and separated from the “real world.” This kind of action suggests that digital spaces ought to be treated as utopian, idealistic places where even the most mundane actions can become fun and engaging. However, we know that technology cannot be this isolated from us, as it exists synchronously with every facet of our lives. There must exist a similar limit where companies will start to discover that an influx of gamification may become damaging to their business model, as focus shifts too far away from their products and services. At the point where customers begin to care more about the game than the product it represents, then the campaign will have failed.

For similar reasons, Cow Clicker succeeds as a satire, by pointing out the foolishness in how we spend our time. It rewards the user achievements, although the user has never really achieved anything, as there is no real skill involved, and no action required beyond
clicking a few buttons. There can be no satisfaction associated with these in-game awards, because this is not a form of success that can be measured.

Like Cow Clicker, Moot also incorporates several in-game achievements. You can earn points and badges as you play, although they do not do much to reflect a user’s progress through the game. The badges are somewhat arbitrary, as the actions required to unlock them are tangential to successful completion of the level. It is not necessary, for example, to scan eight different colored products and unlock the “Taste the Rainbow” badge when most mazes can be completed by unlocking just two or three colors (Figure 8). Moreover, the badges are not consistent in the amount of effort required to earn them. While unlocking “Taste the Rainbow” would likely involve a thorough inventory of products in order to obtain an appropriate assortment of colors, users can earn the “New Moot on the Block” badge the moment they scan their first barcode. Earning “Moot Points” in the game is even more arbitrary, as points are awarded according to a mathematical algorithm that has no bearing on the product scanned. A bottle of ketchup might be worth 100 points, while mustard is only worth 2.
Figure 8: Achievements are earned in Moot by completing specific objectives separate from those necessary to advance.

Figure 9: Players can view a list of their locked and unlocked achievements.
In levels three and four of Moot, the type of action required is inverted. To some degree, you must separate yourself from the phone in order to succeed. While the first two levels allow a sense of exploration through action, the second two levels explore the player’s relationship to technology through inaction. Level three requires the completion of a QR code jigsaw puzzle. There is no way to advance past the level unless you can manage to scan the puzzle that you have just completed (Figure 10). Using just your phone, it is impossible to do so as you cannot use your phone’s camera to take a picture of the phone’s screen. Almost every method conceivable for completing the level requires you to remove that image from your phone, such as using a second phone or taking a screenshot and printing it out. Either action requires some sense of removal from your phone. Level four is the most obvious exploration of this idea. In this level, the player is presented with a 5 minute timer (Figure 11). If you touch the screen, pick up the phone, or even bump the table that it’s sitting on, the timer will start over. The only way to win is to let the timer hit zero. This kind of interaction is in direct opposition to most games, which aim to be addicting and to keep the user playing for as long as possible. Levels 3 and 4 offer the player an opportunity to reflect upon their gameplay experience, as they must take a break before proceeding.
Figure 10: In level three of Moot, a player must complete a QR code, and then figure out how to scan it.

Figure 11: In level four, the player must do absolutely nothing until the timer runs out.
The point of Moot, if there is one, isn’t to make your life seem more exciting and more like a video game. Rather, it is to make a video game interaction seem more mundane and more like a real world interaction. In that sense, Moot meets all of the traditional characteristics of the game — it is voluntary, has rules, and is limited — but I wouldn’t call it a game. Cow Clicker and SuperBetter both pass as games if evaluated using Huizinga’s standard game characteristics from 1938. However, advancements in technology over the past 80 years have caused an explosion in the tools that creators have and the platforms through which they can share their work. There is now a spectrum of possibilities for new experiences that would have been hard to imagine even just a few decades ago. With this expanded realm of possibilities, the classical definition of gameplay deserves a re-evaluation. In this new era of game creation, it is no longer sufficient that something simply looks and functions like a game. If an experience is to be classified as gameplay, it should maintain the spirit of a game as well. This is, of course, a very subjective standard to evaluate. However it seems reasonable to construct a more narrow definition as a means to deconstruct games in the digital sphere.

The Internet is Not A Game

Everything online today is game-like. The internet is full of things that are interactive, social, shareable, and creative. We have reached a point where we can take any phenomenon and give it the characteristics of a game or disguise it as a game, just by moving it into a digital space. Digital spaces naturally possess most of Huizinga’s characteristics. They are limited and ordered, and they provide a means for recreation and imagination. Moreover, they feel game-like because they are interactive and engaging. Screens evoke images of televisions and arcade machines, naturally suggesting an
entertainment value. Even actions like amassing likes on a comment or photo or tracking
your progress on Twitter is a direct mirror of how many points or powerups you achieve in a
game. The internet has become a space where you’re constantly trying to earn more, and
compare yourself to others. We carry our social networks in our pockets, waiting for
indications of success in the form of push notifications — one new message, five retweets,
two new followers. It’s very easy and very tempting to treat the internet like a game.
When we play video games, the boundaries of play are very clear. When we go offline or
turn off the console, we know that play has been suspended, and we make a cognitive
transition away from the world of the game. However, the boundary surrounding our lives
on the internet is much less clear. Our lives on the internet proceed uninterrupted, even
when we’re asleep. Our actions online have real implications that persist even when we
close our browsers. In a way, the internet has created a perpetual magic circle that we
operate in whenever we’re online. It makes us blind to the fact that our engagement is not
necessarily isolated within this magic circle, although we may consider ourselves in an
isolated realm. Even though, from our perspective, we are one person interacting with one
device, every action we have online has real implications on whomever is on the other end of
that device. When we “escape” into the digital realm, we experience a reality that looks and
feels a lot like a magic circle because we simply can’t fathom what’s beyond it.

Because we treat the internet as a game, we often treat it as if it has fewer or no
consequences as compared to “real life.” We tend to see the internet as a space that everyone
is participating in at the same time, as if it is a magic circle where everyone is playing by the
same rules. However, we know that this is not the case. The internet has become a massive,
diverse space where there are many different individuals and groups with different
intentions striving toward different goals. The internet is not as limited as we typically treat it, and thus our online actions are not as limited as they are when we play games.

Moot plays with these limitations, as it requires game elements that are very unusual and don’t make a lot of sense. Most games try to keep our attention as long as possible, while Moot directly encourages you to separate yourself from the game. Games provide us with tangible objectives, while Moot has no instructions, little feedback, and arbitrary indicators of success. Games provide a sense of recreation and escape, while Moot makes activities out of rather mundane tasks. Moot deconstructs most of the traditional game mechanics, yet it is still game-like. It’s categorized under “Puzzle Games” in the app store, and takes its inspiration from classic games we are familiar with such as hangman and jigsaw puzzles. Moot suggests that the boundary between work and play is more blurry than it has ever been. Perhaps there is no true definition of a game, and there is no “right way” to gamify our lives. Around the same time that Ian Bogost concluded Cow Clicker with a game-ending “Cowpocalypse” event, he wrote

“Technology never saves nor condemns us. It influences us, of course, changing how we perceive, conceive of, and interact with our world... The Internet extends us in both remarkable and unremarkable ways. From keeping a journal to paying a bill to reminiscing about an old television advertisement, the Web offers just as many mundane uses as it does remarkable ones. Probably more.”

If we can learn anything from Moot, it is that technology has complicated our lives in so many ways — some good, some bad. But to try and create divisions between self and technology is to ignore a fundamental component of our culture. We should never be asking ourselves whether something happened “online or in real life,” because the Internet is just as real as every other part of our lives. The only inauthentic experience we can have surrounding technology would be to discredit it or ignore it.

---

15 Bogost, 2011.
Behind the Scenes: A Reflection

Creating Moot was the largest project that I have completed from start to finish. It allowed me to bring together most of the skills I have developed throughout my college experience, and ended up being the perfect culmination of my four years of study. When most people open my app, they see a user-facing finished product. However, no user of my game will ever be able to experience the development process, or to understand the immense amount of work and decision-making that went into even the smallest details of Moot. The following is just an abbreviated chronicle of how Moot came into being.

Aside from the technical and academic challenges I faced throughout the development of this project, one of my proudest personal achievements is taking this project from an initial idea to completion. Working on a project of this scale individually, there were many times when I faced what seemed like insurmountable hurdles. Maintaining my motivation and continuing to find inspiration in my own learning process was both extremely challenging and incredibly rewarding.

Moot was written in the Swift programming language for iOS. Swift is a fairly new language, first released by Apple in June 2014 as an alternative to its predecessor, Objective-C. Compared to Objective-C, Swift is syntactically much simpler, less prone to code errors, and generally easier to use. Moot’s backend application was written in Python, and interfaces with a PostgreSQL database. The backend runs on a Linux server using the CentOS operating system.

Before I could start programming the gameplay mechanics, I had to determine which of the levels I had designed would actually be feasible to produce. Part of my original intention behind the app was for it to be an exploratory experience for those who played it. However, it also provided me with an extraordinary exploratory experience while developing
For every new idea I had for a new feature or mechanic, I found myself poring over online tutorials, reading forum posts from other developers implementing similar projects, and figuring out how to adapt what I had learned in order to fit my own app. Although I borrowed a lot of design patterns, code structure, and inspiration from resources others had shared online, there was also an immense amount of customized adaptation that had to occur to make these features work for Moot.

Barcode scanning was one of my original ideas, but as was the case for most of my original designs, I had no prior experience implementing these mechanics. The first challenge was being able to utilize the phone’s camera as a scanner. In this, I learned a lot about iOS permissions, as the app must prompt the user before it is granted permission to access the camera. Once I was able to achieve a reliable procedure for accessing the camera, I had to get the camera to recognize barcode patterns, and to translate that image into a 12-digit UPC string. iOS has built-in support for recognizing barcode data in a library called AVFoundation, and I relied heavily on Apple’s documentation to get this working. When all of this was completed, I was able to successfully scan a barcode and retrieve the product UPC. However, this was just the first step of a series of features I needed to develop to achieve a complete game.

Retrieving product information would be a key component of the first two levels of my game. This means that simply reading the barcode and returning a 12-digit string was not enough. I had to then take that barcode string and retrieve information about the product so that I could display it on the screen and use it in the game. This led me to begin searching for an appropriate UPC database that would return data including a product name and picture when provided with a 12-digit string. These kinds of data requests between a database and an application are usually handled through an API, or Application
Programming Interface. In short, a request is sent to the API (in this case, containing the UPC string), and a plain text response is returned, containing relevant data (the product name and image URL). In my research, I found multiple APIs online that would provide me with product information for UPCs. I would eventually use multiple APIs for different purposes, selected based upon their price model (ideally free to use) and limits on requests-per-minute.

Once able to retrieve information for product scanning, I had to build in the game logic for the first level’s hangman game. There are many factors that influence the program architecture for web and mobile games. Rather than retain most of the data processing within the app itself, I chose to write my own backend API on a cloud server that would handle all of the user interactions and game data. The most important consequence of this choice was that it allowed me to collect and compare data between users and store that data in my own database. I originally opted not to create a database in an effort to simplify the development process. However, I quickly realized that I was making an unnecessary effort to avoid storing data and that adding in database functionality would allow me much more flexibility to incorporate even more features into the game. This became the basis for how I tracked progress for achievements and high scores for users.

Finally, having my own backend also made it easier to collect and manipulate data before presenting it back to the user. Moreover, it allowed me to move the computation-heavy logic away from Swift, and into another language. I chose Python for my backend language because of my familiarity with it, its ease of use in handling network requests, and its wide availability of libraries that would allow me to do things like handle user authentication, connect to my database, manage level progress, and perform color analysis on images.
Authentication is a facet of Moot that was revisited many times until I arrived at the final iteration. Since my original prototype for Moot did not store user data in a backend database, I had no need to identify individual users. Any user data was stored on individual devices and not on my server. As soon as I added the database, it became necessary to associate stored data with unique users. I chose a very standard approach at first, requiring new users to set up an account with a username, password, and recovery email address. Database entries were then associated with unique usernames, and each request to my backend required username and password authentication. Although this approach definitely accomplished what it needed to do, having a complex registration page for a simple app felt very cumbersome, especially when I was not implementing features such as interacting between users in the game. My final approach was to identify users by the phone’s unique device identifier (UUID). This meant that I could reduce the registration and login screens into a single button. New accounts would be automatically generated, and the need for passwords were eliminated, since the login was now associated with a phone rather than with an account. The drawback to this process is that the UUID is associated not just with an individual device, but also with a particular installation of the game. If at any point the game is uninstalled, all data and progress for the app would be lost. Since the game does not include something like in-app purchases, which would be problematic if not recoverable, I still found this approach to be the most effective solution.

Level two is by far the most programmatically complex level in the game. For this level, I had to do a lot of independent research on two complex algorithmic processes. First, I had to figure out how to randomly generate square mazes of a given size. Secondly, I had to figure out how to take the image of the product and find its most dominant color.
A maze, by definition, must have at least one valid passage from start to finish. In addition, each tile must be accessible from any given starting point. That is, no passages of the maze can be completely “walled off.” I implemented a recursive backtracking algorithm in order to generate mazes. The next challenge of maze generation was to be able to efficiently represent that data in the game, as well as how to represent that data graphically.

It is a trivial task to draw a maze out on paper. However, the maze had to be represented as a grid of data in order to be able to continually calculate a user’s position, as well as which directions are valid moves. Finally, the data had to be simple enough that it was easy to send back and forth between the database and the app itself. I wound up creating my own structure to represent a maze of any size as a simple string of numbers, with each number representing one tile. I was then able to use that list of numbers to evaluate position, as well as to display the maze on the screen. For a more detailed explanation of the recursive backtracking algorithm, as well as my custom data structure, please refer to the appendix on random recursive maze generation.

The need to take this data and present it on screen required me to learn about Core Graphics in the Swift library. While some graphical elements in Moot’s layout are static images, such as the achievement badges, the maze tiles needed to be created dynamically. It wouldn’t be possible (or it would be very difficult) to create an image for every possible maze. Not accounting for invalid configurations of maze tiles, there are \(16^{25}\) possible combinations of tiles for a 5x5 tile maze. Core Graphics allows you to create vector drawings composed of basic lines and shapes. I was then able to take the integers from my maze string, and translate them into a vector image of a square tile with the appropriate walls drawn in. I was also able to utilize Core Graphics when creating the color wheel compass that is used for navigation in level two. While it would have been easier to simply insert an image of the
compass onto the screen, I needed to be able to detect which portion of the compass was
touched in order to prompt the movement of the token through the maze. There is not a
simple way to distinguish different portions of a plain image. However, with Core Graphics,
each colored shape comprising the compass is its own object stored in data. Thus, it became
trivial to tell when the blue sector was touched, rather than trying to calculate the position
and boundaries of that sector within a static image.

Finally, the last challenge of level two was to determine the most dominant image
color of scanned products. The API I used provided me with a photo of each product I
retrieved from the database. I then had to perform an analysis on the pixels of that image in
order to determine which color was most prevalent. To perform a color analysis on each
individual pixel in the image is a task that requires an enormous amount of computation. It
would not have been feasible to do this in a game, as each picture scanned would probably
require at least half a minute of processing time before the user would receive any feedback.
Instead, a common technique used for this is k-means clustering. The goal is to randomly
choose just a few clusters of pixels in an image, and then to group pixels based on their
proximity to those clusters. In digital graphics, colors are broken down into their red, green,
and blue (RGB) components. With these three values, any given color can be represented as
a point in three-dimensional space. Using this representation, a simple three-dimensional
distance formula can be used to measure a single color’s “proximity” to a given cluster. This
process would successfully determine which color in the spectrum was most dominant, and
return the RGB value for that color. However, that triplet of integers was not immediately
useful to me, as I would need to determine whether that numerical value corresponded to a
particular named color (for example, “orange”). For this, I had to select which RGB value I
wanted to define as my prototypical “orange,” and so forth. I would then use the distance
calculation once again to determine which vernacular color name the dominant color most resembled. In the end, I was successful in implementing this complicated process. However, I was still slightly unsatisfied with the time it took to process a given image. I could achieve a more accurate color analysis by using more clusters, but this resulted in a longer calculation time (around 7 to 10 seconds). I ultimately discarded my work in favor of a third-party library that managed to implement the same process more efficiently. This dropped the scanning time to about 3 to 5 seconds, which was a much more satisfactory waiting duration for a game.

Beyond my rejected implementation for color analysis, there were many more bits and pieces I worked on that did not make it into the final version of Moot. One of the features I had planned to add from the very beginning of this project was location-based interaction. My goal was to add physical movement to the game’s repertoire of real-world interactions. I developed a prototype for a GPS-based level. A user would be presented with an indicated direction, and they would then have to travel a certain distance in that direction before moving on to a new instruction. The level would have functioned as a parallel to the maze in level two, but instead of moving a token on screen, the user would have to physically move themselves. I was able to successfully determine a player’s bearing using the phone’s built-in compass. However, I had a very difficult time calculating distance traveled. Current GPS technology is very effective at calculating large distances traveled, such as those used for turn-by-turn navigation. However, it is often too imprecise to calculate smaller distances, especially those less than 10 feet. My intention was to have the users travel just a few steps before receiving a new instruction, so all of my efforts to calculate distances walked using GPS data were wildly inaccurate. I had a new idea to calculate distances traveled using the phone’s accelerometer. However, a cursory amount of research revealed that trying to
calculate linear distance using acceleration data would first require a double-integration on the acceleration data. Even if this were done, the resulting margin of error renders this calculation too imprecise for practical measures. Finally, I did some experimentation with the new step-counting tools that recently became available with the release of Apple’s HealthKit. This was not a desirable solution either, most notably because this feature is not available on iPhones earlier than the iPhone 5s, or on any iPads. Moreover, the step-counting tools present in this library are intended to track a large range of motion over a period of time, rather than giving real-time feedback. While I would have loved to implement a location-based level in Moot, I ultimately could not find a solution that I was happy with.

Additionally, there was a very early design for level one in which I wanted to use handwriting analysis to input letters. This was another attempt to introduce more technology into the game, as well as to provide more unique mechanisms for game interaction. I wound up abandoning this feature rather early, as I discovered that effective handwriting recognition is not easily achieved, and is a still-emerging field in computer science. While there are plenty of OCR (optical character recognition) programs to recognize printed text, recognition of hand drawn characters is much harder to achieve.

Finally, my last big challenge was packaging and submitting Moot in preparation to secure approval to the App Store. There was an enormous amount of detail that went into the submission process, including adding a privacy policy to the app, classifying an age rating, and taking screenshots for six different device sizes. I also had to thoroughly evaluate Moot according to the App Store Review Guidelines, in order to make sure that Moot was using data appropriately, upheld an acceptable level of user privacy, functioned reliably without bugs or crashes, and maintained a design consistent with iOS interface standards.
After a five day waiting and review period, Moot was approved on the first attempt, and is now available for download in the iTunes store.

Completing this game was by far the biggest feat of design I have ever accomplished. For every small decision I made along the way, I had to consider what consequences that choice might have on my app in the future. When I was choosing a method of authentication, I had to weigh the implications of each possibility in terms of security as well as aesthetic appearance. Devising a method to extract colors from images accurately yet efficiently required a careful balance between data efficiency and quality of gameplay. When I have worked on smaller processes in the past, it was very easy to be shortsighted, only focusing on the visual design, algorithmic efficiency, or ease of implementation. By creating most of the pieces of Moot from scratch, I gained a true appreciation for how all of the various aspects of mobile development fit together to deliver an experience that both looks appealing and functions efficiently. Finally, working independently meant that I had to be both self-sufficient and practical when it came to my design decisions. I learned so much through independent research and experimentation and I ultimately was able to execute so many different features that I had never attempted before. On the other hand, I struggled to accept my own limitations, and to concede features that were unreasonable to develop due to either time or difficulty. As this project comes to an end, I have also gained a great appreciation for just how much work goes into any one feature of an application, no matter how small. Fittingly, the internet was my greatest ally in completing this project. That I was able to learn so much about the field of mobile development from the comfort of my own desk chair is truly remarkable. My respect for and understanding of technology has changed in so many ways due to this experience, and I am eager to continue to bring technology a little bit closer into my “real life.”
Appendix: Random Recursive Maze Generation

As mentioned previously, I used a recursive backtracking algorithm in order to randomly generate mazes of a given size. In my research, I found at least a dozen different algorithms that can be used for maze generation. I ultimately chose this algorithm for its ease of implementation, computation speed, and aesthetically pleasing path generation. Some other algorithms tend to generate mazes that contain lots of short dead-ends, long straight corridors, or directional bias (either horizontal or vertical). The recursive backtracker tends to produce mazes with a lot of variation and interesting passageways.

This method starts with a square grid, and then carves passages out of the grid according to a specified procedure. First, a starting tile is randomly chosen from the grid, and then marked as “visited.” All adjacent tiles are analyzed, and one tile is randomly chosen among the non-visited tiles. The wall between these two tiles is then carved out, and the adjacent tile becomes the new current tile. This process continues until there are no unvisited tiles adjacent to the current tile. At this point, the algorithm keeps backing up to the previous tile until a tile with uncarved walls is reached. Then, it continues as previously indicated. When it has backed up all the way to the starting tile, the process is complete. Figure 12 shows a visualization of this algorithm in progress.
Figure 12: The recursive backtracking algorithm randomly carves out one wall at a time until every tile in the maze has been visited. In the above sequence, unvisited tiles are shown in gray, visited tiles are shown in red, and finalized tiles are shown in white. This diagram has been reproduced from a simulation originally created by Jamis Buck.
After I learned how to implement this algorithm, I had to proceed with creating a mechanism by which I could easily save, transport, and manipulate a maze as a simple data structure. If I were working solely in one programming language, it would be easy to create my own class object representing a maze tile, which would then contain Boolean values for the state of the four walls. However, I needed to send this maze data as plain text over a network call to my app. This meant that the kind of data structure I could use was limited to a simple text string.

Taking the information I had thus far, I devised a method to represent any given tile as an integer between 0 and 15. A tile contains data about its four walls. Each wall is essentially a boolean value — either the wall exists, or it has been carved away. Thus, each tile can be represented with only four bits of data. I took the four cardinal directions in clockwise order (north, east, south, and west), and used this as the order in which I would define my tiles. I assigned a value of either a 1 or 0 (representing “true” — the wall exists, or “false” — the wall does not exist, respectively), to each position. A tile with no walls would be represented as “0000,” whereas a tile with only the north wall would be “1000,” and so on. All 16 combinations are reflected in figure 13. At this point, I had created a system wherein any tile could be represented with a four-digit binary number. By converting these binary numbers to their base-10 equivalents, each tile can be represented by a single integer in the range (0, 15).
Figure 13: All 16 possible maze tiles can be represented using Boolean values to represent the state of each of the four walls. In this diagram, a wall is represented with a thick blue line.

Figure 14: The Boolean values have been converted to integer values for easier readability and data manipulation.
Figure 14 shows the updated representation of tiles using base-10 integers. With this simplified notation, an entire maze can be represented as an underscore-delineated string of integers, each integer representing one individual tile. For example, the maze generated previously in figure 12 can be represented as the string “12_11_10_9_5_12_3_12_3_4_3_14_3_13_6_9_12_10_1_14_2_3_14_3” (see figure 15 below).

![Maze Diagram](image)

**Figure 15:** The generated maze is now easily represented with integer values.

As long as there is a square number of integers in the string, a square maze can be reconstructed. After this data is passed back from the API, it can easily be separated into an array of integers. From there, each integer can be translated back to binary, and new tile class objects can be constructed for more flexible data manipulation.
Download and Source Code

At the time of this printing, Moot can be downloaded freely from the iTunes App Store, where it is published under the title “Moot: The Game.”

Moot is an open-source project. The full source code is currently available for view on GitHub at http://github.com/erinbleiweiss/Thesis.
Bibliography


**Technological Attribution**

While the development of Moot was an independent undertaking, I obtained an enormous amount of guidance and inspiration from blogs and tutorials from other developers. In particular, I owe the technological success of Moot to the information provided by the following sources.


In addition, Moot uses the following libraries and APIs.

**Swift**

Fu, R., & Tang, P. SwiftyJSON (Version 2.3.1). Retrieved from https://github.com/SwiftyJSON/SwiftyJSON

High Bay. PageMenu (Version 1.2.9). Retrieved from https://github.com/HighBay/PageMenu


Noon, C., & Fuller, K. Alamofire (Version 3.3.1). Retrieved from https://github.com/Alamofire/Alamofire

Olcay, C. EmoticonParticles (Version 0.0.1). Retrieved from https://github.com/yemeksepeti/EmoticonParticles

Ramotion, Inc. RAMAnimatedTabBarController (Version 1.3.1). Retrieved from https://github.com/Ramotion/animated-tab-bar

To, M. Spring (Version 1.0.3). Retrieved from https://github.com/Mengto/Spring

**Python**
Feng, S. Color Thief (Version 0.0.2). Retrieved from https://github.com/fengsp/color-thief-py


Ronacher, A. Flask (Version 0.10.1). Retrieved from http://flask.pocoo.org/

**APIs**


Erin Bleiweiss was born and raised in Houston, Texas. She followed in her family’s tradition of attending the University of Texas at Austin, and enrolled in the Plan II Honors Program in August 2012. During her time at the University of Texas, she pursued her passions for technology by completing the Elements of Computing certificate from the Department of Computer Science, as well as the Bridging Disciplines certificate in Digital Arts and Media. In her senior year, she served as the Associate Director for New Media for the University’s Student Government. She graduated in May 2016 and accepted a position to work full-time as a software developer in Austin, Texas.