Ghosts of the Horseshoe:
A Mobilization of a Critical Interactive

by

Richard Lee Walker

Bachelor of Science
University of South Carolina Aiken 2007

Master of Science
Medical College of Georgia 2009

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2014

Accepted by:

Duncan A. Buell, Co-Major Professor

Heidi Rae Cooley, Co-Major Professor

Marco G. Valtorta, Committee Member

Jenay Beer, Committee Member

Laura Kissel, Committee Member

Lacy Ford, Vice Provost and Dean of Graduate Studies

Dedication

I dedicate this work to my wife Shuiqing Qiu, my family and the family dogs Wiley, Lucy, Honey and Xiao Hei. I also thank everyone who made this project possible, including Dr. Heidi Rae Cooley and Dr. Duncan Buell.

Thanks for all the Fish.

Acknowledgments

I first thank my advisors Dr. Duncan Buell and Dr. Heidi Rae Cooley for helping guide this project to its completion. Without their help, it would not have become as succesful.

I also to thank my committee, Dr. Jenay Beer, Laura Kissel and Dr. Marco Valtorta. I appriciate all their help and patience with me as I worked to complete the dissertation.

I would also like thank my colleagues Jeremy Greenberger, John Hodgson, Jess Tompkins, J. J. Shepherd, and Renaldo Doe who have helped with conceiving ideas and as individuals I could talk to when defining *Ghosts of the Horseshoe*.

I also thank the graduate students who built the "Slavery at South Carolina College" website. Without them, there would have been no project. I give thanks to the design teams that helped build the application during the Fall of 2012 and the Spring of 2014.

I also thank my dog. Who eats all my money and is great for relieving stress.

Finally, I thank my parents, my family and my wife.

Abstract

Critical Interactives (CIs) are designed to harness the voluntary, reality-bending excitement of discovery as afforded by play, but to do so in the context of rules that mobilize procedural rhetoric to instantiate critical awareness. Critical interactives are not just about improving lives through code or education; rather, they establish a methodology for generating more aesthetic and reflective interactive experiences. To grasp more fully the logic underpinning CIs, we need to understand the powerful nature of interactivity and outline how such interactivity involves a notion of ethics, i.e., a way of living, in and through media practice.

Ghosts of the Horseshoe is a critical interactive, in this case a mobile interactive application for iPad, that presents the largely unknown role of South Carolina College, the predecessor of the University of South Carolina, in slavery during the years prior to the Civil War. The USC Horseshoe was built by enslaved persons, and the bricks of the Wall and buildings made by enslaved persons, and yet this history is for the most part not known by the USC community and not acknowledged by the institution.

We discuss the role of critical interactives as instruments of procedural rhetoric—software artifacts that interact with their participants to carry a message, in this case a message about a sensitive topic in the history of the institution. *Ghosts* as a CI uses ludic methods as a rhetorical technique. We place CIs, and *Ghosts* in particular, in the general context of games, computer video games, and serious games, commenting on the use of ludic methods in presenting topics like slavery about which one cannot legitimately produce a "game". We discuss further the iterative development and testing process that produced the final version that is available today.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Critical Interactives (CIs) are designed to harness the voluntary, reality-bending excitement of discovery as afforded by play, but to do so in the context of rules that mobilize procedural rhetoric to instantiate critical awareness. Critical interactives are not just about improving lives through code or education; rather, they establish a methodology for generating more aesthetic and reflective interactive experiences. To grasp more fully the logic underpinning CIs, we need to understand the powerful nature of interactivity and outline how such interactivity involves a notion of ethics, i.e., a way of living, in and through media practice.

1.2 Play and Games

At the most basic level, the CI requires that we contend with the notion of play. Leyden J. Huizinga [Huizinga 1971] defines play in his work *Homo Ludens* as an activity with no clear material or profitable gain, meaning that no real world materials like money, land, or toys are "won" by participating in the experience. He also describes play as existing within its own set of boundaries (i.e., the time and location that define the activity of a play session). There is also a fixed set of rules that govern the play session, with the sole purpose of weighting the play session by determining degree of fairness and balance (e.g., handicap). Although in most sessions of play some sense of a "set of rules" is established before the session begins, rules are not

required to be in place at the beginning. Huizinga argues that play helps promote the development of productive social interactions and is designed to help individuals envelop themselves in a special and secret place that allows them to establish their difference from the common world by disguise or other means. He notes that those in play are able to learn social norms by pretending to be different from whom they really are. Huizinga further defines play as an act performed by the living with no exact correlation to species per se. In other words, it serves a more vital function than simply providing for the basic needs of any particular kind of being. Rather, the work of play is more socio-cultural (broadly construed) than biologically-specific. One just has to watch animals at play to recognize a kinship between, for example, a dog chewing a stuffed plush armadillo or an elephant playing in a waterfall, and the play of children. As Huizinga underscores, play is not species specific [Huizinga 1971].

However, as Roger Caillois [Caillois 1961] asserts, Huizinga's definition of play is frequently too vague. Caillois explains that while play is voluntary, distinct from ordinary life, and riddled with rules to maintain the enjoyment of the act, there is more to it. Caillois qualifies play. He outlines a specific taxonomy of attributes that characterize play: play is free, separate, uncertain, unproductive, governed by rules, and make-believe. That play is free refers to the fact that it "is not obligatory" and "would at once lose its attractive and joyous quality as diversion" [Caillois 1961, p. 9] were it to be so. Play is about being free to play as one wishes—or as one is drawn to—without having to put emphasis on arbitrary conditions that restrict it. If it is not attractive or joyous, interesting or engaging, a person will become bored and specifically avoid that type of play. One cannot force a person to play against his free will and get the same level of commitment as a person who freely engages in some exercise of play. Like Huizinga, Caillois contends that play is separate from the outside world and requires a special spatial and temporal delimitation. Those in play

construct their own space in which to play, whether preconceived or *in medias res*; and in some instances, players set some form of arbitrary time constraints. Often, outside stimuli conclude the play session. For example, the toy being played with is destroyed beyond use or a parent forces a child to stop and do his homework. But sometimes the activity simply reaches an ending, such as when the light outside fades to night or the other party in play decides to go home for dinner. In addition to being free and separate spatially and temporally, play is malleable, insofar as most rules are considerably flexible and subject to evolution. The rules of play change and flow based on the whim of the participants and according to the process of play itself. Any object might very well become incorporated into an act of play with a set of rules attached (e.g., a game based on not touching peppers on a tree). The outcome of play is unknown, as is the amount of time spent in play. So, too, is the location, be it in someone's backyard, the middle of the street, or even in line at a local market.

Play is also arguably unproductive. By "unproductive" Caillois means that there is no clear, pre-established goal of creating goods, wealth, or new elements of any kind beyond the scope of the play-time and play-space. However, we must also acknowledge that play is hardly unproductive, especially to the extent that it provides a context for learning. Animals commonly use play as a means of acquiring new skills. Cats pounce on smaller animals; dogs bury stuffed armadillos in the garden; and humans interact with each other. An individual always steps away from a play session a different person from when he first entered into play; meaning that while play is unproductive in a fiscal (or capitalistic) sense, it still produces meaningful and/or beneficial outcomes. While the rules may be unspecified at the beginning of play, all forms of play are governed by some set of rules, for example, establishing some form of conditions of play that keep players safe while playing or setting limits to what is considered a success within the context of play. In other words, rules always arise during play, but they also morph and their terms shift. If a child is playing on

his own and decides that the best form of play is getting dizzy, he has established a rule. The rule would be to spin as much as possible until he topples to the ground. If someone joins the child, then the rules would have changed. The session would establish a competition at this point: who can spin the longest. Establishing a set of rules helps define the play session for all those engaged.

Finally, play is inherently—and by extension, necessarily—make-believe. All preceding characteristics of play converge in a make-believe reality. The realm of make-believe fosters the conditions whereby the individual can experiment safely and "play" without the pressures of the real world pushing back. The make-believe of play affords conditions for experimentation; it allows for an individual to become something they are not for an instant or go to places that are far out of their normal reach. A person has to get lost in play for it truly to be "play", and that always takes some form of make-believe [Caillois 1961]. This is ever present when one watches a group of children play "the ground is lava" or when an adult role-plays in a game of *Dungeons and Dragons*.

Given that the definition of play established by Caillois is similar to the concept most commonly associated with games broadly construed, one has to wonder what, in turn, is a game. According to Katie Salen and Eric Zimmerman, a game is a defined system with any number of players, artificial boundaries that maintain the specified playspace, conflict to keep players interested, rules to define the system, and some quantifiable outcome (i.e., a "win" state of some sort). One might find this definition very similar to Caillois' definition of play. The system is a functionally related group of elements that provide a structure for the game. It contains an organized set of interrelated ideas and principles, objects defined according to the established playspace, and a hierarchy of how people are to act and react with/in the system. A game also contains players, specifically the individuals who interact with the system and participate in the experience of play. The game has to be artificial,

setting up a boundary between the real world and the playspace so as to establish an environment suitable for the game to take place. Games also are rife with conflict, power struggles of some sort or other—be it to get the players to compete with one-another or work together against a common enemy. The rules of a game commonly work in tandem with the system to provide structure to facilitate play. Finally, games have a quantifiable outcome, a set goal that needs to be completed and that subsequently concludes the game [Salen and Zimmerman 2004].

Given how the expanded definition of play provided by Caillois shares many similarities with the definition of a game provided by Salen and Zimmerman, one might assume that play is synonymous with game. While a game might appropriately describe play (because many think of play in such a way), "play" and "game" are not inherently analogous. Play is determined by all six criteria defined by Caillois, but only when one is playing. Play itself is abstract and encompasses a large range of activities: ranging from free form experiences, such as people watching or doing somersaults; to ludus, which is structured by rules, like tag; to paidia, which is unstructured and spontaneous, as in the case of a sandbox [Caillois 1961]. When one participates in the act of playing, the possibility of options compress, that is, become more limited, as the parameters of play are established. Brown and Vaughan identify seven types or modes of play: Attunement, Body, Objective, Social, Imaginative, Narrative and Transformative. Attunement is about establishing connections, such as a baby getting used to his new family. Body is about an individual discovering how his body works and functions, such as twisting one's arm enough to see how far he can reach. Object is about playing with toys. Social play is about building connections with others; anyone trying to make new friends will engage in some form of social play. The last three—Imaginative, Narrative and Transformative—are very similar to make-believe as Caillois posits it: pure fantasy, storytelling to help build language skills, and being integrative, or pretending to be something one is not. If

one engages in any of these types of play, he is not necessarily in a game. By contrast, anyone participating in a game is definitely engaging in play [Brown and Vaughan 2010]. Those that are engaging in play are considered to be playful, but this does not mean that play and being playful are the same.

Play is a category or activity in which an individual invests time and effort. Play describes the space of activity. Play defines the tone and location, the type of situation in which one will be engaged, and the type of activity that the play session will feature. Being playful or "playing" is specifically individual, that is, someone performs the act of playing. Being playful is about the person actually engaging in play. Moreover, it refers to the quality or nature of that engagement. It is about the person going into the play session and getting lost, i.e., immersed, in the activity. In light of these distinctions, we can say that play is an environmental context into which a person enters, whereas being playful describes those participating in the play session. When one is being playful, one invokes the concept of play and adds rules and systems to make the time more entertaining. By gamifying play, one is limiting all possible play potentialities in order to contain, or define, the conditions of play according to certain rules.

Many individuals harness the energies of play to complete tasks, even if they are not playfully engaged in the activity. This is commonly done when one makes a "game" out of a particular situation, similar to how an office worker creates a game to motivate the completion of an obligatory task or how graduate student teaching assistants turn grading papers into a competition. Games come into effect to facilitate and nurture play, but again, games and play are not the same thing. While people do say they are "making a game out of it", this expression is not asserting that all forms of games elicit playfulness. Rather, the idiom suggests that someone has taken an act he was performing and applied a set of rules to it in order to turn it into a game-like structure. Such structures, ones that invite play or play-like behaviors, are

stimulating, and because they are, they keep us participating.

Salen and Zimmerman postulate that game and play are different entities, even as they share the same plane of existence. They argue that play is a subset of games, at the same time that games are a subset of play. They contend that because not all forms of play are games, games can be only a subset of play. At the same time, they explain that because games contain rules, indicative of culture (and its forms of play), play is a subset of games. This bi-directional definition is odd given that many different facets of life also contain rules and play (culturally specific, in both cases), but would not be considered "games" in the traditional sense. An example is filing taxes. There are some who could take the terrible task and turn it into a "game". The rules would be to finish the paperwork as quickly as possible, the play would be to get the task done as interestingly as possible, and the cultural context is what dictates the fact that the individual had to do taxes in the first place. Even then, filing one's taxes is not a game. Given this, one could argue for a Venn diagram structure wherein play and game overlap heavily, but not completely. Similar to the notion of $P \neq ?NP$[1], it is never clear whether or not the one (play) and the other (game) are truly fully equivalent [Salen and Zimmerman 2004]. Given this distinction between play and game, one can examine the foundation of what constitutes a critical interactive. One engages in play voluntarily mostly for enjoyment; but at times, one would engage in play to learn something new about his environment. Games are centered in a cultural context, wherein play transpires according to rules that create a space conducive of facilitating further play. A CI, however, does not strive to be a traditional game that facilitates play for enjoyment. Rather, a CI endeavors to facilitate play-like

[1] $P \neq ?NP$ is one of the major unsolved computer science problems. It asks whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time. Certain algorithms, such as the Traveling Salesman Problem (i.e., a salesman is planning on traveling to all Augustas, as seen in Nixons film, and needs a route to allow him to visit all locations without doubling back and in the fastest time possible) takes non-polynomial time (i.e., $n^{number\_of\_augustas} - 1$ time to solve the problem).

engagement for learning. Learning, here, is not explicitly quantifiable, even though it may be observable. This is because the rules that define interaction invite one to consider a social, political, and/or cultural context anew.

The *Ghosts of the Horseshoe* CI embodies this condition. It is designed to introduce participants to complex historical content that draws attention to the politics of race at South Carolina College. Its established rules facilitate exploration of (i.e., critical play with) textual, audio, and visual materials in the context of the historic Horseshoe. In doing so, *Ghosts* attempts to promote in its participants a more nuanced cultural understanding of USC's historic Horseshoe. The rules come from the gamic nature of the experience and the modes in which content is loaded. Increased cultural awareness is derived from the rich history present on the Horseshoe. To be sure, *Ghosts of the Horseshoe* is not a traditional game, but it bears a relation to games and video games.

1.3 Games and Video Games

Caillois commonly uses "play" and "game" synonymously. Given this, when he is describing the complexity of play by referring to games, he is for the most part describing the different types of games that exist. Aside from ludus, which is rule-governed, or paidia, which is unstructured and spontaneous, Caillois specifies four sub-categories: agon, alea, mimicry, and ilinx. Salen and Zimmerman refer to these categories as "ludic activities", which form a special subset of games. Agon games are "competitive, that is to say, like a combat in which equality of chances is artificially created in order that the adversaries should confront each other under ideal conditions, susceptible of giving precise and incontestable value to the winner's triumph". In contrast to agon, alea is a ludic activity "based on a decision independent of the player, an outcome over which he has no control, and in which winning is the result of fate rather than triumphing over an adversary". Mimicry involves an individual who

"forgets, disguises, or temporarily sheds his personality in order to feign another". An individual who is playing in this fashion pretends to be someone or something they are not to build an illusion for engaging in play. The final type of ludic activity, ilinx, is based on the notion of shock. This type of ludic activity primarily focuses on the "pursuit of vertigo" and involves "an attempt to momentarily destroy the stability of perception and inflict a kind of voluptuous panic upon an otherwise ludic mind".

Games incorporate at least one of Caillois' four types of ludic activity. A board game such as Monopoly^{TM}, for example, combines agon and alea. Players participate in agon when buying up property, selling land, trading properties with other players, and determining the best course of action for bankrupting their opponents. Monopoly takes capitalistic strategy and skill to crush the other players. Alea determines how players traverse the board. The throw of the dice decides how the playing piece advances and, therefore, what property a player can buy and whether the player suffers a monetary fine or goes to jail (without collecting two hundred dollars).

When a game such as Monopoly^{TM}is reworked in digital form, one has a video game. A video game uses a digital system to create an environment conducive to play. It commonly uses electronic media to create "a mental contest, played with a computer according to certain rules for amusement, recreation, or winning a stake" [Zyda 2005]. The rules of play for video games differ, however, from those of other games, especially those of board games. Video games use the electronic media to create the rules of play and provide the system governing play. The rules of board games are used to limit the play space and determine what are and are not valid actions. In Monopoly^{TM}, for example, the game exists in the real world and the rules of play limit what is a valid move for each players turn. The nature of video games is different—in a video game the rules (expand the play space) because they define what the computer will permit as game play.

Here, it is worth noting that there are two tiers of rules at work: 1) the rules that

determine play; and 2) the rules—code—that produce the software environment. A person plays by interacting with the system, either playing within the constructed rules to enjoy whatever the designer was intending or by trying to break the rules to discover the boundaries of the system. The first set of rules strictly relate to objectives and concepts the designer chose to limit the play space. If the designer decided that players were to take turns, as in *Super Mario Bros.*, then the rules are used to limit the play space.

Video games often attempt to place players in situations that are representative of the real world, and thus the designers create rules to represent that world. Board games tend to be more abstract and use rules to delineate the game's "magic circle" from the "real world" that we live in. With video games, the world the player occupies in the play space is the "real world" for that player and the rules help create what is possible. The rules that govern the world allow for all possible actions, giving video games a more free form nature.

The rules of play that govern the play session to create the experience and the rules in place to govern the environment and world are seperate, but in communication. Video games commonly use multiple ludic activities to develop the entire complex system of rules. These rules can be very simple, from the *Flappy Bird* mechanic of pressing the screen to more complex transactions such as those seen in *The Elder Scrolls: Skyrim* whose many mechanics create an elaborate playspace.

In the world of video games, gameplay mechanics (rules of play) determine the type or "genre". Unlike movies or text, for which plot structure, character, and "framing" serve to define type (i.e., a suspense film, a noir film, a documentary, a religious text, non-fiction, etc.), video games are categorized by the gameplay mechanics (i.e., first-person, platformer, puzzler, action-adventure, etc.). These mechanics not only establish the play space, but also determine the messages the game conveys and how. Typical video games—those adhering to game industry standards—do not usually

intend to impart messages explicitly. Rather, they aim to provide entertainment; after all entertainment ensures profit. For example, we might think of two high yielding games, such as *Super Mario Bros.* or *Grand Theft Auto 3*. These games do not attempt to be more than just "fun". But even though their primary goal is fun, each game achieves this in entirely different ways.

Super Mario Bros. is a classic video game from the 1980s and was one of the most popular video games released on the Nintendo Entertainment System. It is a one player game (although two players can alternate) and pits the player against the environment. The game asks players to dodge, jump, and avoid obstacles to reach an end goal which then allows the player to proceed to the next stage or level. The rules of play established by the video game are very simple: run right, jump to avoid obstacles, jump over enemies, acquire extra lives and power-ups to help complete the level, and capture the flag or an axe at the end of the stage in order to advance to the next stage. All these rules are established and create the unique play space (see the discussion of Caillois in "Play and Games").

Grand Theft Auto 3 creates an equally valid, yet entirely different, play space and experience. Released in 2001 for Sony's Playstation 2 gaming device, *Grand Theft Auto 3* changed the way video games were made. *Grand Theft Auto 3* introduced the notion of "sandbox" games, games whose systems permit more open worlds that offer a richer set of play possibilities. Sandbox games emphasize interacting with the rules that create the play space more than reaching an arbitrary end goal. A player can create his own tasks and objectives. The player may choose to do nothing but drive in circles all day, or he might follow proper track law within the game world; he might run over people with his car; or he might opt to play the game for its story. This is because the software allows for a variety of possible play transactions—a player can steal cars, drive the cars in any direction he wishes, drive boats, fly planes, shoot at cops, race other drivers, collect hidden packages, etc. The play space is limited only

by the software program and the imagination of the player. The experience created by *Grand Theft Auto 3* is very different from *Super Mario Bros.*, but both invite play by creating a space for the player to have fun. In *Super Mario Bros.*, world building rules were limited to creating the linear experience and the goal of saving the princess was the focus. In *Grand Theft Auto 3*, the rules that create the play space are many and varied, allowing for any possible action. Even though both games, *Super Mario Bros.* and *Grand Theft Auto 3*, have world building rules, this does not mean *Super Mario Bros.* is a sandbox game. The rules created for *Super Mario Bros.* only exist to push the main objective of the game while rules in *Grand Theft Auto 3* exist to expand possible actions and outcomes.

A pure sandbox video game, one with no win state and which only has rules that govern the world and the possible actions in which a player can make is *Minecraft*. *Minecraft* has no objective, no rules designed to push the player to an end result. The game only has rules designed to govern the world and rules that establish what the player can and cannot achieve. The main rules involve hitting environment blocks with ones fists or items to produce a resource. Many gathered resources can then be combined into a new object that can then be used to further dig up newer materials. The rules of the system allow the player to dig up rocks, sand, and emeralds and then build items such as axes, shovels, and pick-axes to then dig up more materials to then build more items. *Minecraft* best shows how rules are applied to a system and how rules can establish an open world environment to let the players just have fun.

While many video games emphasize fun or focus on just the play experience, others attempt to be more than just "fun". A game *Ico* provides a good example. A player is tasked to protect a non-player character (NPC) from harm. Its game play is designed to invite a player to empathize with the NPC. The rules in place make the hero weak. It is very different from *Super Mario Bros.*, where the player is powerful.

The character is very slow in running, cannot defeat all the enemies that appear, and can only guide the NPC's movements; the player character cannot give commands or bark orders. *Ico* generates a play space shrouded in anticipation and suspense that is derived from fun. All the rules created are designed specifically to create a challenge and to be fun. If the game was not fun, the games overarching themes and message would never be heard. Video games like *Silent Hill 2* and *Papers, Please* are meant to be more engaging than fun, but the sense of fun is still ever present in the design, feel and goal of the project. *Silent Hill 2* uses very rough controls and a thick atmosphere to create a sense of tension. This directly taps into the ilinx concept of games. The idea is to be scary and unnerving and thus to create an engaging experience. *Papers, Please* does something similar with the use of "moral choice" systems that require the player to make uncomfortable decisions. The player in *Papers, Please* assumes the role of a security guard at the border of a country and determines who gains entry and who does not. His choices affect the lives of the people he encounters; each decision has potentially devastating consequences. Both experiences use play rules to create unique experiences that are more engaging than fun. These games differ starkly from *Super Mario Bros.* and *Grand Theft Auto 3*. These three games take their subject matter more seriously, assigning rules to actions to emphasize more than just being powerful and free. The rules are used to restrict the player, to get them to feel something more than just fun. *Super Mario Bros.* and *Grand Theft Auto 3* create a power-like fantasy while the other games focus more on narrative and generating a unique experience. These types of games tend to focus more on engagement over just "fun". By making something to be engaging, more meaningful experiences can be constructed without having to make sure the rules are "fun". It is difficult to make a game about sending a person to their death at a security gate "fun" given that the action itself is not fun and the impact it has on the game world. This is more engaging, giving the player something to experience and think about.

This is important: engagement is more important than just "fun" because this is one of the major concepts behind creating meaningful experiences and for constructing serious games.

1.4 Serious Games

Studies have shown that video games can improve an individual's performance in executing complicated tasks such as surgery [Rosser et al. 2007]. Such games are usually referred to as "serious games". They work from the assumption that an individual's experiences with an interactive medium can directly affect his life. Common video games such as *Grand Theft Auto 3* and *Super Mario Bros.* have no such goal.

Clark Abt coined the term "serious games" in his book *Serious Games* [Abt 2002], in which he outlines what he considers to be a serious game. Abt asserts that serious games require "an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement". While his book discusses non-digital media, many of his core concepts can still be applied to the video/electronic context [Abt 2002]. In 2005, Mike Zyda offered a revised definition for serious games: "a mental contest, played with a computer in accordance with specific rules that uses entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives". Zyda's definition focuses on training and/or instruction [Zyda 2005]. Currently, serious games use ludic methods to enforce a political, social, marketing, economic, environmental or humanitarian objective [Arvers 2009].

The field of serious games can be divided into several loose categories. The most relevant for a comparison with critical interactions are advergames, edutainment, newsgames, simulations and art games. Common examples of such serious games include:

- *Sneak King* (`http://en.wikipedia.org/wiki/Sneak_King`)

- *Food Force* (no longer avaliable)
- *Darfur is Dying* (`http://www.darfurisdying.com`)
- *McDonald's the Video Game* (`http://www.mcvideogame.com`)
- *JFK Reloaded* (`http://en.wikipedia.org/wiki/JFK:_Reloaded`)
- *Loneliness*
  (`http://www.necessarygames.com/my-games/loneliness/flash`)
- *Freedom Bridge*
  (`http://www.necessarygames.com/my-games/freedom-bridge/flash`)

Advergames are probably the loosest branch of serious games. Advergames rule sets emphasize fun as do regular video games. But because the overall message of the advergame experience is merchandising, such games are considered serious games. An advergame's only goal is to market products and entice people to purchase a product. Frequently, advergames feature in-game advertising, where real world advertisements appear during the video game experience.[2] The purpose, of course, is to seed in the player a desire for and subsequent purchase of the advertised commodity. An example of such a serious game is *Sneak King*, created by Blitz Games in 2006 for the Xbox and Xbox 360 gaming system. *Sneak King* promotes hamburger consumption. It uses gamic actions such as stealth (sneaking around the stage unnoticed) and platforming (jumping from location to location) to have the player character give hamburgers to non-playable characters (NPCs). Success involves sneaking up on an NPC undetected and giving him a hamburger. Failure is getting caught in the act.

[2]While advergames may include in-game advertisements, in-game advertising is a separate practice. Advergames are explicitly marketing tools, they have no function outside mobilizing fun to sell a product. In contrast, in-game advertising is simply the practice of placing ads in a video game. *Mario Kart 8*, for example, allows a player to select a Mercedes Benz as a kart. But the fact that a Mercedes is a kart option does not make *Mario Kart 8* an advergame because the point of the game as a whole is not the sale of a car. Of course, this is not suggest to that Mercedes might not benefit from this visibility.

While receiving poor reviews from critics, the game's entire goal was advertising hamburgers through fun.[3] Edutainment games are games whose main purpose is to educate. Of course, many games might educate people through tangential learning. One might very well learn about the Greek gods by playing *God of War*, even though the game is not necessarily intended for such purpose. In other words, not all games that teach are considered edutainment (educational) games. Educational games are designed specifically to provide some form of educational value. They are designed primarily to teach, commonly sacrificing rules of play to get the educational message across. An example of an educational game is *Food Force* [United Nations World Food Programme 2005]. Created by the United Nations World Food Programme in 2005, the game attempts to educate players about famines and the processes required to help stabilize famine stricken countries. The player is tasked with entering a fictional country to help feed citizens, balancing diets for everyone to be healthy, locating food to send to the country, and developing the means for stabilizing the country. The mechanics, subject matter, and experience are focused more on educating the player and less on fun. This proves problematic at times. After all, if fun is secondary to instructional value, players might simply perform requisite actions in order to complete the game. In this context, play becomes pro forma and learning may or may not be accomplished.

Games with a journalistic intent are considered newsgames. Newsgames can be used to educate, similar to edutainment games, and can be used to push political or social opinion or perform in other journalistic ways. Ian Bogost offers a basic definition of newsgames: "a broad body of work produced at the intersection of video games and journalism". The generality of this definition makes it a very inclusive category, allowing for very different games, such as *Food Force* and *JFK Reloaded*, to

[3]And the strategy proved successful. 3.2 million games were sold, which resulted in 80% brand recall that translated into a 40% increase in sales for Burger King.

be considered newsgames. One of the better examples that fits the category is *Darfur is Dying* [Rulz 2006]. *Darfur is Dying* is a browser based Flash game depicting the impact war has on displaced families. The game tasks the player with searching for water while avoiding local militias and using the water resources to keep their small village alive for seven days. The mechanics and narrative of the experience are based on real world events, with everything in the game designed to help establish the game's political position.

Unlike the previous types of serious games, simulation games fall into a gray area, occupying a place between video games (see "Games and Video Games") and serious games. Games like *SimCity* boast rules that model real world systems, specifically that of running a city. While *SimCity* mimics various city management operations, it is not a "serious simulation game". *Flight Simulator*, on the other hand, does simulate very closely the real-world procedures of flying an actual aircraft. Still other simulation "games" are nothing more than pure simulations and use very few gamic methods. For example, a "player" might be able to adjust a limited number of parameters: a simulation of a comet striking a planet might allow one to tweak the size of a comet and its velocity, thereby affecting trajectory and, likely, degree of impact.

Other simulations work to emphasize political positions or challenge widely held assumptions. *McDonald's the Video Game* [Molleindustria 2006] and *JFK Reloaded* are two examples. *McDonald's the Video Game* was created specifically to show corruption within the fast food industry. The player is tasked with running McDonald's and must do anything within his power to keep the enterprise operating. To "win" the game requires breaking laws, destroying historical locations, massacring cows, using unhealthy chemicals, and bribing politicians. The game's simulation hyperbolizes tactics that corporations are often accused of deploying to make its point. *JFK Reloaded* attempts to simulate the assassination of President Kennedy to raise

questions about the position that Lee Harvey Oswald was a lone gunman. It places the player in the position of Oswald, tasks him with mimicking the assassination, and then rates how closely the player's shot matches the deadly shot. In both of these instances, simulation opens onto a site of contestation where fun is relative. This does not mean that "being fun" and being a "serious game" are mutually exclusive; it simply demonstrates how play might serve other purposes.

The last category in the serious games taxonomy is that of "art games". Art games, as defined by Scott Steinberg, are games "designed to emphasize art or whose structure is intended to produce some kind of reaction in its audience" [Steinberg 2010]. The art game category is expansive. Art games range from games that focus on style (i.e. *Grim Fandango* or *Okami*), to ones that privilege narrative (i.e. *BioShock* or *Portal*), to those that feature gameplay (i.e. *Braid* or *Flower*) or a message (i.e. *Loneliness* or *Freedom Bridge*). Oftentimes, this category is used to describe examples of games that a community considers exemplary of the serious games medium. This definition is too broad and inclusive, since it even allows for categorizing a game such as a *Super Mario Bros.* as an art game.[4] Perhaps a more precise definition of an art game might be: a game that focuses on its artistic intent and message more than on fun. It is a game that foregrounds artistic style and gamemaker expression; the game itself is a piece of art rather than the rules that govern the play space.

Art games as seen through the lens of serious games offer an experience far different from that of "normal" games. Two good examples include Magnuson's *Loneliness* and *Freedom Bridge*. *Loneliness* is a simple game in which the player navigates a single block, moving it up, down, left and right towards groups of other blocks. As the player block moves towards these groups, the other blocks disperse and fade away. The game never asks any questions about what to do and never prompts the player

[4]In this context, *Super Mario Bros.* illustrates how artistic trends have evolved. Worth noting, as well, is the fact that in recent years art games have begun adopting a retro pixel art style with simplistic graphics to match the graphical style and power of early video games.

with the current task; it just asks to be played. *Loneliness* is about the loneliness faced by Korean children; it aims to engage the player with these feelings and to create the sense of the children's loneliness and isolation [Magnuson 2011b]. *Freedom Bridge*'s game mechanic resembles that of *Loneliness*, only that a player is limited to moving left and right. The player moves his cube through three sets of barbed wire before finding a bridge. Each set of wire slows the player character down a bit and adds a trail of red. At the bridge, the player cube is shot and explodes from the screen. Play, here, evokes the experience of North Koreans attempting to escape to South Korea [Magnuson 2011a]. Again, no context is given until after the experience. Neither of these games can aptly be considered "fun", but the underlying mechanic in each case functions to open onto experiences that might elicit empathy. Mechanics are metaphorical in both instances, serving to emphasize the starkness of each reality. The minimal artistic style encourages the player to consider more fully the thematic—of loneliness and futility, respectively.

This taxonomy and the previous discussion regarding genres demonstrate the messiness of categorization. Not many games can be definitively assigned to a type or genre, even as this is common practice in the game industry. Figure 1.4 presents the games discussed above, arranging them on a continuum: from most gamic—or fun—to most serious. In so doing, it calls into question the divide between serious games and video games. There is a division that can be made, but it is not very clear. The next figure, Figure 1.4, shows how the classification of the electronic game can be broken down. This diagram uses examples seen earlier to show the relative categories that exist.

1.5 Critical Interactives

Games, as typically understood, establish a set of rules that govern play. When these rules are applied to the electronic medium and a system is set in place to create a

Figure 1.1 The Spectrum of Games

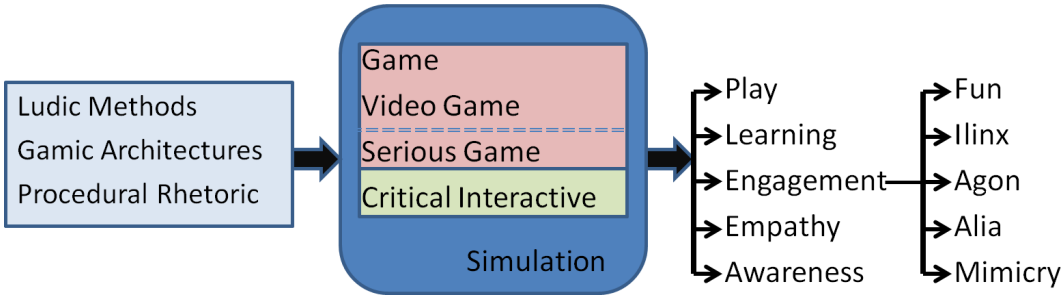

Figure 1.2 Genres of Games

digital game space, then one has a video game. Games like *Grand Theft Auto* and *Shadow of the Colossus* use this framework to entertain and give the player something fun to do. In the case of serious games, the game-space design pursues some other goal: sell products, create simulations, or educate individuals. Here, the notion of persuasive games, a sub-category of serious games, offers a useful, because narrower, perspective of what a game might accomplish. As Bogost explains in *Persuasive Games: The Expressive Power of Videogames* [Bogost 2007], the mechanics of a game can be designed to make arguments; they can function persuasively. That is, software has rhetorical potential. Critical interactives mobilize this potential to elicit empathic awareness, i.e., an intellectual sensitivity.

Heidi Rae Cooley and Duncan Buell coined the term "critical interactive" in 2011 in an article titled "Critical Interactives: Improving Public Understanding of Institutional Policy" [Buell and Cooley 2012]. Mary Flanagan's notion of "critical play" [Flanagan 2009] and Bogost's "procedural rhetoric" [Bogost 2007] inform their con-

ceptualization of CIs. As Cooley and Buell explain, CIs are interactive systems that use ludic methods to engage individuals, impart knowledge, build awareness, question past observations, and challenge preconceived notions. Those who interact with a CI are invited to become active participants in an ongoing conversation about the particular subject matter and content presented through the interactive. Using the medium's strengths—computer software and code—mobile devices such as phones and tablets (although computers might also serve as CI interfaces), one might present socially, politically, and/or philosophically charged ideas and elicit questions in a ludic fashion through interactivity [Buell and Cooley 2012].

1.6 A Foreshadowing of Things to Come

The critical interactive brings together three fields of thinking: digital humanities, human computer interaction, and computing. The next chapter, Chapter Two, will focus heavily on the methodologies behind digital humanities and how it applies to critical interactives. Other examples from the digital humanities will also be discussed to help further define CIs. Chapter Three will be dedicated to human computer interaction and its methodologies. It will go into studies showing how the way an application is built can greatly affect its persuasiveness. The fourth chapter will discuss how the computer factors into a CI. The computer, and more specifically programming, will be addressed. The fifth chapter will take up *Ghosts of the Horseshoe* as an example of a CI. It will focus on the code. The last chapters will discuss how usability and effectiveness of the *Ghosts* application were assessed, present final conclusions, and offer suggestions for further development of *Ghosts of the Horseshoe* and the design of critical interactives more generally.

Chapter 2

Digital Humanities and the Work of Framing Content

At the time of this writing, the digital humanities (DH) are a new and expansive field that continues to evolve. Others have defined DH to be "The scholarly study and use of computers and computer culture to illuminate the human record" [Priego 2012], "a critical investigation and practice of humanities research in the digital medium" [Flanders 2012], and "the use of digital tools and methods in humanities study and dissemination" [Rockwell 2012]. UCLA offers a more robust definition: "Digital Humanities interprets the cultural and social impact of new media and information technologies—the fundamental components of the new information age—as well as creates and applies these technologies to answer cultural, social, historical, and philological questions, both those traditionally conceived and those only enabled by new technologies" [UCLA Center for Digital Humanities 2014]. The term is "an umbrella term that covers a wide variety of digital work in the humanities: development of multimedia pedagogies and scholarship, designing and building tools, human computer interaction, designing and building archives and so on". [Gossett 2012]. DH is interdisciplinary; its tenets shape what critical interactives are and how they are designed. When one examines the methodologies behind a CI with a DH focus, one better understands one of the major goals of a CI: to frame and present well-researched content in a way that can facilitate critical thinking—through interaction with a rules-based system—about a subject of concern, such as how racial politics

give shape to an historical site.

2.1 Framing of Past Mediums - Text

While not the first form of communication, text is one of the most basic. In writing, signs and symbols convey language. Not all forms of writing are the same; most western cultures base their signs and symbols on ancient Greek; others use images or ideographs for language representation. The way text frames its content is unique because the reader has to fill in most of the gaps. When a picture is described in text, most of the details are simplified or exaggerated to get the reader to comprehend the image.

Text is both personal and impersonal for a reader. On the one hand, the reader has a stronger connection to the events about which he is reading. He has to imagine everything in his mind; this allows him to fill in the blanks. A person has to reference the object in his own way, relate it to something that is similar and then represent it as his own image [Peirce 1991]. This makes the final construction more personal. The impersonal aspect comes from the fact that text abstracts. This is because text has to offer a representation that many readers can access. So while an individual has constructed his own representation of the content, he does so from a position detached from the actuality of the situation. Take for example the Abu Ghraib scandal. In reading the story about the hooded man standing on the box with his hands outstretched in the form of a cross, a different image is created in each individual's mind. While there will be consistency in the basic details (e.g., hood, arms outstretched), how these details take shape mentally for person will differ. Anyone reading about the hooded man would not directly be connected with him in that situation, although they might very well feel a connection to the story. Word choice is also important to consider. An article about the Abu Ghraib scandal can be written in a multiple ways, each account giving an accurate representation of the events that

transpired, but ultimately with a different affect on people's opinions of the event. Language is important; word choice is even more so. Words connotative of urgency or threat might evoke panic or rage. A more moderate use of language might frame the situation more casually. In other words, one could go away from an account with the impression that the United States military were unjustified in their approach to interrogation or that they used necessary techniques.

Textual accounts are biased, even those that intend to be neutral. But they can also misrepresent or diminish the gravity of a situation. Text about an uncomfortable subject can easily dismiss what it describes. While CIs involve their participants in an interaction with digital systems, the content they present includes text. It is important to make sure text is appropriate and accurate. Special care in word choice is likewise important so as to represent the content faithfully. Word choice is made even more important by the small number of words permitted (35 to 50 words) in a public history setting.

Many of the weaknesses of text can be mitigated by the use of images—especially photographic images. While text can misrepresent the reality it depicts, such misrepresentation is much harder to achieve with images (assuming, of course, that the images themselves are faithful and accurate). Presenting an image itself provides more information in a smaller space, and in the case of photography, an indexical representation of an event. While images seem to offer a more reliable depiction of information, there is still the matter of how they frame that information.

2.2 Framing of Past Mediums - Images, Still and Moving

In the context of photographic representation, framing has three functions: it selects and puts into view the content that is featured in the image; it serves as a boundary for that selection; and it excludes the context that exists beyond that boundary (e.g., the outside world). Because of this, as Judith Butler argues, images have the

potential to normalize how people think about things [Butler 2009]. Through the lens of the Abu Ghraib1 scandal[1], it is possible to see how images can be manipulated to shape an understanding of the "war on terror". During the George W. Bush presidency (2001-2009), when Abu Ghraib occurred, the Bush administration worked diligently to keep "unpleasant" images of the wars in Afghanistan and Iraq from the public eye. (Of course, governments have always attempted to regulate the circulation of images.) This included removing images from print that were deemed damaging to the cause, controlling what images could be taken, and the "frame" in which the photos were shot. This allowed the government to control what the public consumed. The controlling of images "suggests that the frame can conduct certain kinds of interpretations" [Butler 2009]. By controlling the images and the framing of images, the Bush administration could control its identity and prevent the terrorists from defining us as monsters. Like Butler, Richard Grusin discusses how management of images impacts the management of populations [Grusin 2010]. Standard media practices allowed for creating a fog about the war. All images were framed, produced, faked or forged. This control of the visual evidence allowed the media to sway public opinion. If the government wanted to strike terrorists with drones, then the media could frame a drone strike as necessary to complete an objective. This power influenced the morality of those participating in the media. The Abu Ghraib photos were different because they were unfiltered, unprocessed images that showed people the "truth" about parts of the war. It produced shock at the level of affect in those who saw them. This unfiltered shock, which for Grusin is not conscious in nature, was difficult for the administration to control and ended up revealing the biopolitics of the situation surrounding the war. It illuminated the darker sides of

[1] The frame of the Abu Ghraib was from the perspective of the soldiers in the prison. Using personal cameras, they took photos of the Iraqi prisoners in many uncomfortable and uncompromising positions. These photos were not from the carefully constructed government framing [Butler 2009].

habits many people share (i.e., taking photos first—automatically; reflecting upon the situation after).

Butler and Grusin are useful for thinking about how CIs frame their information. For example, *Ghosts of the Horseshoe* works with the history of the University of South Carolina's historic Horseshoe. This landscape—the Horseshoe—was built by enslaved labor. The framing of all the images present in *Ghosts* can be used to call into question not only the history of the present, but also how the history has been presented in the past. Commonly, controversial information is archived and a more pristine, because sanitized, image is offered for public consumption. Butler would argue that this framing defaces the overall history, and that only by revealing the frame for what it is can one bring new insight. Grusin would urge us to understand that our own everyday practices with technologies can participate in the same framing that Butler contends should be revealed. Both argue for careful consideration about how information is represented and how content is framed.[2]

Ghosts of the Horseshoe attempts to present the information about the slavery at the South Carolina College differently. When approaching a sensitive subject, it is important to consider other media produced on the subject and how that media frames its content so as to communicate a message. It is easy to replicate other work. But *Ghosts* strives to use its images to raise awareness. One example of giving awareness through images is the fingerprint image used as buttons throughout the experience. Slaves built the structures that populate the Horseshoe campus, maintained the grounds, and worked for faculty, but no artifacts of theirs remain. The built structures remain, but those who built them are gone. All has disappeared except for fingerprints found in the bricks molded by the hands of the enslaved persons. Because these impressions are the only (to our knowledge) indexical traces of that

[2]While this discussion only addresses still images, a similar argument can be applied to moving images, for example, film, video, and animation. In this case, one would have to consider how editing the way the narrative moves through time shapes meaning.

labor, the fingerprint was used to promote awareness and get participants thinking about their relationship to a history of slavery specific to the physical grounds of the Horseshoe.

2.3 Framing of Past Mediums - Artificial Intelligence

Critical interactives are the result of interdisciplinary collaboration across the humanities and computer science. Because of this, it is useful to take another look at framing. Derrida, Grusin, and Butler [Butler 2009; Derrida 1987; Grusin 2010] understand framing from the humanities perspective and their conceptualization differs from that presented by Minsky and by Levesque [Minsky 1974; Levesque 2012] who approach the concept from the perspective of computer science. Even so, Minsky's version of framing can be used to formalize various Derridean framing constructs.

Derrida's frame at its most basic alludes to the frame placed around a painting. A metaphor, it refers to the fact that when one looks at a framed image, the contents of the image are not the only things the frame draws attention to. The frame also will draw attention to itself as well as separate the painting from the outside world. This "frame" draws attention to all parts of the image, the content, the frame itself and the outside world. While the divide being discussed is in terms of a painting, this happens with all forms of media; the way in which the content is presented will frame the content and how it is divided from the world. But sometimes it is difficult to differentiate between where the frame starts and ends.

Framing as discussed by Minsky concerns the knowledge known about a system. Minsky describes a frame as "a data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party" [Minsky 1974]. He further explains that a frame is "a network of nodes and relations" and argues that "different frames of a system describe the scene from different viewpoints, and the transformations between one frame and another represent the effects

of moving from place to place" [Minsky 1974]. To Minsky, a frame is the information of a given state that is used to represent a given situation presented in the system. This construction of a frame is more algorithmic in nature and is useful for developing explanations of or structures for solving a problem. Different frames in the system are not mutually exclusive; they share terminal information that can "correspond to the same physical features as seen in different views" [Minsky 1974]. A frame can be used to describe a room from different angles and perspectives, to describe the different stages of a plot from different points of view, or to decompose a sentence into key parts. Frame-like structures can be used to separate sentences into key parts and to parse the meanings of the sentence.

Minsky's version of framing is different from Derrida's, but there is overlap. Minsky's is a logic-based system that is used to decompose scenes and scenarios into small parts for logic-based reasoning. This framework can be used to outline and model some of Derrida's ideas. Derrida's concept of framing is concerned with how an object (broadly construed) is separated from other objects and how this separation draws attention to the objects inside the frame, the frame itself and objects outside the frame. A Derridean frame can be used to explain how a certain media object expresses ideas and conveys meaning. The parts of the frame call attention to more than just the content. This is in some regards similar to Minsky's definition of a frame, in which a scene can be decomposed into smaller parts for logical reasoning.

Given the different interpretations of framing, for example framing a photograph, Derridean framing would look at the way photo was taken, the content inside the photo, the means by which it was taken, the world outside that resulted in the image being created and the impact that comes from the image that is produced. Mitsky would decompose all this information into a frame for modeling, taking the frame, the producer, the angle, the information appearing inside, and creating a dataset. A good example of the two types of frames appears in Errol Morris's documentary *Standard*

Operating Procedure [Morris 2008], which investigats the events that culminated in the Abu Ghraib political debacle. In one scene, a forensic media expert describes aligning photographs taken with three different cameras. The way in which all the photos from the source camera were lined up can be mapped to the way Minsky derives a frame. US military officials found three photos that depicted the same event (the stacking of prisoners into a pyramid) and broke down the internal details to construct a timeline of the events. Those three photos would be separate Minsky frames. The overall impact of the photos and how they reflected on the army, the war in Afghanistan and Iraq, the way the public received and processed the information, and the way it "cloned terror" are all framing information similar to what is discussed by Derrida.

2.4 Framing of Past Mediums - Video Games

The first video game on record was a missile simulator called *Cathode Ray Tube Amusement Device* [*Cathode-Ray Tube Amusement Device - The First Electronic Game*]. This game used analog circuitry to control CRT lights to position a dot on the screen to attack screen overlay targets. When one applies the DH notion of framing to the video game experience, visual recognizers and rules of the system both have to be examined. Similar to text, images and film, what is seen inside the game is important. The computer screen frames what is viewed in a manner similar to film, given that a video game experience involves motion. Moreover, the frame is entirely dependent on the technology that makes possible the game's creation and the amount of resources at the creator's disposal. If a video game has a large budget and a large staff of programmers, the game can produce a more intrinsic, less explicit, frame. Given the technology of the time, *Cathode Ray Tube Amusement Device* used dots to simulate missiles and overlays of any image chosen to be a target. The visual framing is military and required the player to imagine that the dotted lines were the missiles

in "action". More examples of framing can be seen in games like *Missile Command* and *Call of Duty: Modern Warfare*.

As the technology improved, so did the ability to improve visual framing. Moving from CRT dots to digital pixels on a television screen allowed for more detailed visual cues. In *Missile Command,* the player controls three mounds each of which bears little line pixels as cannons. Beneath the mounds stand six cities, divided and surrounded by hills. The player sees a pixelated landscape against a black background. Missiles fly into view from the top of the screen towards one of the six cities. The framing of the game is simple, but the visuals show a small set of cities, maybe a small state, that is being bombarded by missiles.

In a more modern example, *Call of Duty: Modern Warfare*, high polygon models and effects are used to put the player in the middle of a Middle East war zone. The player sees lush environments, desert storms, realistic terrorists, and semi-believable weaponry. For the graphical power at the time, the framing of the experience was as real as possible to make the player believe he was in a real war zone. Instead of the flat third person perspective seen in *Missile Command*, *Call of Duty: Modern Warfare* uses a first person view. This view makes all the framing appear more personal, as if everything being seen is controlled by or directed at the player himself. Many different video games can provide different framings of content similar to text, images and film, but video games also have an extra framing device that needs to be recognized. While content provides a specific frame of reference, the rules of play created by the system provide a different kind of framing only capable with an interactive system.

2.5 Framing of Past Mediums - Procedural Rhetoric

While most video games in the standard market focus on generating "fun" experiences, there are some that demand more from their audience. Games like *Silent Hill 2* or *Papers, Please* are about engaging the player in an experience that goes beyond

just something fun to do. As mentioned above, a serious game or a persuasive game is a mental contest, played with a computer in accordance with specific rules that uses entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives [Zyda 2005]. "Serious game" is currently an overall moniker of the "serious" movement, while "persuasive game" is used more for interactive experiences that persuade, that is, influence behavior. Examining serious games, one sees the strength of an interactive experience. Interaction with the rules and systems remove the seeming passivity found with text, images and video and require the participant to have a physically active role with the material. This interaction can shape how an individual perceives the content presented in ways that other forms of media can not. Game design that uses procedural rhetoric mobilizes the very rules of interaction to persuade a participant to make a change in his worldview.

Procedural rhetoric is one of the defining features of a CI. It stems from the concept of a procedure, i.e., the notion of performing a routine. We emphasize, in speaking of a procedure, the process being used, regardless of the task. A procedure could be as simple as a recipe for cooking, as brutal as the standard operating procedure for brutalizing the Iraqi prisoners at Abu Ghraib in 2004, or more computational like a computer algorithm to count the votes for the next major election. Procedural rhetoric is the application to rhetoric of procedure, which is a form of persuasion. Rhetoric is the practice of using arguments and ideas to persuade an individual to adopt a specific point of view. This can be done through prose, through pictures, and through motion pictures. When the two concepts of procedure and rhetoric combine, the notion of procedural rhetoric arises: persuasion by means of procedure. Video games, serious games, and importantly, CIs all employ a version of procedural rhetoric [Bogost 2007].

Even though some may disagree, many video games emphasize a point of view

through play and the procedures throughout the game, even if unintentionally. If you look at games such as *Call of Duty* or *Candy Crush Saga*, the procedures in place can express the notion that "violence solves all your problems" or "throwing money at a problem will help you solve it". Even if the rhetoric is never noticed, it still resides in the mechanical choices made possible by the game's code. An example of a serious game that uses procedural rhetoric is *the McDonald's game* [Molleindustria 2006]„ which uses its gameplay mechanics to push its environmental agenda.

The framing effect is present in interactive experiences in many facets of design. Similar to text, a problem's contextualization can affect the participant's perception. The way in which a question is framed has an effect on the perspectives and perceptions [Tversky and Kahneman 1981]. Framing not only affects the way in which text is presented, given that the phrasing of an idea can affect how a person interacts with the content, but also with the decisions that are presented to the individuals, the outcomes and choices provided by the interactive experience. A good example is from the video game series *Mass Effect*. The *Mass Effect* series uses a moral choice system to allow a player to select paragon (good) or renegade (bad). The framing of many of the moral quandaries is intended to get players thinking about decisions such as whether to reprogram or to exterminate the Geth. The way the decision is framed puts the player in control of the fate of the Geth, but the way it is coded destroys the carefully framed event. The coding of the event is purely based on the moral choice dichotomy; this event suggests that committing global brainwashing is acceptable while committing genocide is terrible. Neither action in this decision can be considered humane, but the code treats one choice as pure and correct while the other as terrible and wrong.

Ghosts of the Horseshoe uses procedural rhetoric to establish the interactive experience. GPS was used to evoke spatial awareness in the participant in order to emphasize the particularity of the site. When a participant encounters a content

point location, he is prompted to engage with content associated with a specific building. The content includes pictures, text, and sound, which sometimes appear as Augmented Reality (AR) overlays. The GPS and the content encourage participants to think location and history together. The site-specific content loads immediately when the participant arrives at a location. The point is to "shock" or surprise participants into awareness. Augmented reality, the use of the real time camera view with information and/or graphical overlays, and the ability to translate documents give more visibility to the past for items that no longer exist, items that have changed over the years and for documents written in a different era.

2.6 Framing of Critical Interactives

Given the goal of previous sections, it is clear that the presentation of knowledge has as much impact on the content (produced by research, etc.) as does the content itself. The way in which any form of content is presented affects the reception of the final message. Given that a CI is an interactive medium, it is important to examine how previous media frame[3] content before finally examining how a CI might do so differently and to what effect. As described before, the concept of CIs is "Informed by Mary Flanagan's scholarship on 'critical play' and Ian Bogost's work on 'procedural rhetoric' ". CIs strive "to impart knowledge, build awareness, and provoke thinking and raise questions". While serious games appear to have some overlap with CI, a CI frames its content differently. Here it is important to understand that terminology is central to framing what a CI is. A CI is not a "game". The term "game" tends to suggest fun. In this light, even "serious games" proves troubling. "Critical

[3] By "frame", we are commonly referring to the frame placed around a painting. It is in reference to the notion that when one looks at a framed image, the contents of the image are not the only things the frame draws attention to. The frame also draws attention to itself as well as separates the painting from the outside world. This "frame" draws attention to all parts of the image, the content, the frame itself, and the outside world. While the divide being discussed is in terms of a painting, this happens with all forms of media; the way in which the media is presented will frame the content and how it is divided from the world [Derrida 1987].

interactives", as a term, intends to underscore a different kind of engagement. It means engaging participants in ludic interaction with socially and politically sensitive, indeed controversial, subject matter. Some topics are difficult to discuss, such as slavery and creating a "game" based on real world enslaved people is not acceptable. This however does not mean that gamic methods, including procedural rhetoric, cannot be used; it just means that it is important not to create a "game".

Attention to language also applies to labeling those who interact with a CI. When one describes an individual as a "user" of a product, the item in question has no further value than being a tool. Users are people who "use" the product; nothing more than functionality is expected. When someone participates with an application, one expects that person to gain more from the experience. They are not just "using" the application like a tool or a product, they are participating in some form of ritual and will hopefully gain more from the experience. Participating requires both the "user" and the application to work systematically to generate a meaningful experience. The term "participant" suggests active engagement.[4] The framing of a CI is similar to that of serious games. The notion of creating a meaningful product to be interacted with is the same, but CI differs when it comes to acknowledgement that not all experiences are "games" and that using such a nomenclature does not work with certain subject matter. CI is not a replacement of serious games nor is it a subcategory. It is a methodology for framing content in order to impart knowledge, build awareness, provoke thinking, and raise questions using gamic methods while respecting the material and not "making a game out of it".

[4] While "participant" offers a reasonable counter to "user", we still find the term limiting. Critical interactives, as we think about them, address their audience as "interactants". If a person is an interactant, he works with the application. This distinction is significant, even if seemingly small. That said, throughout this dissertation, "participant" is used because it is a more recognized term.

2.7 Projects Similar to Critical Interactives

The notion of a CI is a relatively new idea, and there are no projects other than *Ghosts of the Horseshoe* that capture the distinctive nature of a CI. Even with this in mind, there are four projects worth mentioning that come relatively close and embody some of the concepts and ideas of a CI. These projects include Augusta App, *Desperate Fishwives*, the *Resurrection Man* prototype, and *[Threshold]*.

Augusta App is a mobile application for iPhone. It was coded by Jeremy Greenberger as a digital supplement to Dr. Heidi Rae Cooley's book *Finding Augusta: Habits of Mobility and Governance in the Digital Era* [Cooley 2014]. The app features QR codes that are embedded throughout the text. Scanning the QR codes delivers additional content to the touchscreen, including information regarding others who have also joined the Augusta App community. Scanning also allows the application to track how far a person has read. The interface features an old magnifying loupe that uses a rotary wheel to allow selection of menu items. A person can take and upload photos, see photos taken by others, read content not available in the book proper, and provide feedback to help improve the application. While at first one might think Augusta App is simply a gimmicky tool for the book, it was designed with the CI philosophy in mind. Augusta App in combination with *Finding Augusta* raise the questions, What is Augusta? and Where might one find Augusta? More theoretically, it asks its participants to recognize that "forms of media [can] alter individuals' experience[s] of their bodies and shape the social collective, problematizing the most salient fact of contemporary mobile media technologies, namely, that they have become, like highways and plumbing, an infrastructure that regulates habit" [Cooley 2014, back cover]. The app's gamic qualities, which include basic world map exploration, notification function, and a Twitter-like feed, keep participants aware of their relation to Augustas of all sorts. Augusta App is a more practical application than the others that will be discussed, but could be considered a CI none the less.

Desperate Fishwives, another project managed by Drs. Cooley and Buell, was one of the precursors to the notion of a CI. The concept of *Desperate Fishwives* came to fruition at the Humanities Gaming Institute hosted at the University of South Carolina during the summer of 2010. The project was proposed by Dr. Ruth McClelland-Nugent of Augusta State University, with the purpose of mobilizing procedural rhetoric to help students learn about social interactions of a 17th-century village in Britain. Using sprite-based avatars and a 3D environment constructed to imitate 17th-century artistic stylings, the project attempted to immerse the participants in the 17th-century world. Play revolved around talking with other players' avatars as well as non-playable characters (NPCs) to gather resources. "Discussions" or social rituals were represented by minigames. There are several play styles; one might play as Cuthbert Blacksmith, Margery Midwife, Andrew Apprentice, etc. Each style has an initial set of statistics that determine social interactions. Participants, as a collective, are expected to accomplish four social rituals before the end of a time-limited play session. If they fail, a final mini-game is played to determine a lesser win state.[5]

The *Resurrection Man* prototype, developed by Jess Tompkins, is an historiographic game.[6] She contends that games based on history can be used to develop alternate histories and that gameplay is also a mechanism for creating a historiographic record. As hypothesized by Tompkins on the subject of docugames, that the "documentary" label should "not be readily applied to video games because it is an insufficient one", and "[i]nteractivity affords more opportunity than documenting or preserving; it invites new perspectives on and interpretations of history". Tompkins

[5] While the game's developer John Hodgson categorizes *Desperate Fishwives* (DF) as a serious game, it does boast characteristics that make it possible to be understood as a critical interactive. In particular, DF uses top-down minigame-based gameplay to simulate the complexities of social encounters in order to invite student-players to consider 17th-century social norms and build their understanding of societal structures and systems [Hodgson 2012].

[6] As in the case of *Desperate Fishwives*, *Resurrection Man* readily meets the requirements of a critical interactive.

argues further that "[h]istoriographic game design acknowledges the multiplicities of history and embraces multiple interpretations based on player-based decisions and outcomes". She maintains that games do not merely transcribe a single history, but provide the possibility of producing many simultaneous (valid and invalid) histories. Moreover, she defines historiographic games to be "a critical mode of interaction that draws attention to certain behaviors or actions specific to the historic moment to encourage player awareness".

Resurrection Man puts a participant in the shoes of the 19th-century grave robber and enslaved person Grandison Harris. It tasks the participant-as-Harris with stealing dead bodies from the cemetery to provide cadavers for dissection to the Medical College of Georgia. The participant takes control of Harris and leads him through the game's environment in an effort to locate, excavate, and steal bodies. All the while, he must avoid being caught by watchmen and dogs that guard the cemetery. There is also an "integrity" meter, which functions as a pseudo health/insanity meter. This tells the participants that Harris is stressed, about to be caught, or generally losing health (e.g., exhaustion). Currently, there are two versions of the prototype. The original version used a "third-person perspective" to follow around the playable character. This framing allowed the participant to see Harris and notice any changes in his health, walk, or mood. The newer version of the prototype uses a "first-person perspective", where the participant "is" Grandison Harris.[7] In both cases, gameplay produces an historical account of the 19th-century practice of cadaver acquisition by means of grave robbing [Tompkins 2014].

[Threshold] is an experimental video game created by Cecil Decker. It asks its participant to examine what "noise" is, in both the audio and visual senses of the term. Typically, noise is considered a disruption. A blip on a music track, a scratch

[7] While "more personal", this perspective runs the risk of allowing the player to forget the matter of enslaved labor, which is fundamental to its goal. Tompkins will have to address this as she continues to develop the project.

on a record, a dead pixel on a computer screen, video buffering, or lag in a video game are all commonly associated with "noise". Many of these instances hinder the task, and commonly breaks the moment. Decker questions if noise is much more than that. He asks whether or not noise might be more than just a disruption; he posits that it might also be a creative or generative tool.

[Threshold] has very simple rules: point-and-click. A participant is first presented with the title screen of the experience. Then, the title slowly becomes pixelated. Each pixel shows a small part of a larger film. If the participant clicks on a pixel, the video changes to another. Given there are no set goals set for with the participant, the only course of action is to click and unify the disjointed image. As the image is unified, random pixels will change to another film, thus making it incredibly difficult to unify the frame. If a pixel is pressed too many times, it "breaks" and can no longer be changed. While this is happening, experimental audio tracks play based on the current dominant video. It is usually cluttered and sounds like noise.

Everything in this experience emphasizes the concept of noise. The visuals make the entire experience disjointed, the experience of which is exacerbated by the fact that the audio does not have melody or use rhythm. Making sense of either audio or visual is usurped by the randomness of the system [Decker 2010]. Because of the unique play style and lack of overall goal, *[Threshold]* does not fit into the defined video game category. Insofar as it plays with a person's perception of what is noise and tries to challenge what people call noise, it can be argued that it is an artistic game from the serious games grouping, but a more valid category would be CI.

As three of the five examples discussed here suggest, CIs prove very useful for addressing historical themes. But in the broader context, CIs aim to build awareness about a topic. In this regard, a CI can be something created for a practical purpose (Augusta App), something that is more gamic in nature (*Desperate Fishwives* and the *Resurrection Man Prototype*), or be purely experimental *([Threshold])*. *Ghosts of the*

Horseshoe attempts to occupy a middle ground, being practical for use by anyone on the historic Horseshoe, gamic with GPS and display of information and experimental with the way content is provided and viewed.

2.8 The Historic Horseshoe of the University of South Carolina

The University of South Carolina in Columbia is home to the historic Horseshoe, arguably one of the most intact "landscapes of slavery" in the nation. Slaves built the bricks used to build the South Carolina campus, now known as the Horseshoe, and the wall surrounding the grounds. Slaves also provided labor to perform daily tasks around the College (i.e., cooking, cleaning, chopping wood). While the College did not own many slaves, it participated in a "hiring out" system, where slaves could be temporarily leased to the College to perform daily tasks. Much of this history has been overlooked, in part because of the fact that what remains in the archives was produced by those who benefited from enslaved labor—that is, the faculty. But a ripe history lurks on site, in the material remains of the structures. As a CI, *Ghosts of the Horseshoe* attempts to impart knowledge, build awareness, and question past observations [Buell and Cooley 2012]. It does so by means of careful framing of historical information. In this case, it is not only interested in how its system works to facilitate engagement, but also in the rich history that defines how all that information might be most effectively presented.

Ghosts of the Horseshoe derives its content from the rich history that is the Horseshoe. The relevant history of the Horseshoe for *Ghosts* spans the years 1801 to 1880, from the founding of the predecessor South Carolina College through the years of the Civil War and Reconstruction. Of the 14 buildings built during the antebellum period, only 11 still stand.[8] And all but one of the "lesser" outbuildings—structures

[8] The eleven buildings that still remain on the Horseshoe at the University of South Carolina in Columbia are Rutledge College (1805), DeSaussure College (1809), First Professors House, now the President's residence (1810), Second Professors House, now McCutcheon House (1813), Third

that functioned as kitchens and slave quarters—have been demolished. The buildings of the Horseshoe were built mostly by hired-out slaves. Enslaved persons molded the bricks for the buildings, erected the buildings, and maintained the premises. All this labor is essentially unremarked today given how history was recorded in the past. An example is the surrounding wall that encloses the Horseshoe. The wall was built by enslaved persons, with each brick being hand molded, carried to location, and assembled into the wall. Yet today, the historic structure is little more than a canvas for posting banners for the sport or other collegiate event of the day. Much of what is documented of the enslaved persons stems from receipts. The College participated in the hiring-out system, which was common in urban slave environments. This allowed for some form of recordkeeping, but also it allowed the institution to cover up its past dealings with slave ownership [Weyeneth et al. 2011]. *Ghosts of the Horseshoe* pulls from this history to bring a voice to this absent history and provide an avenue for future conversations about slavery at the University of South Carolina.

2.9 The Evolution of *Ghosts of the Horseshoe*

The *Ghosts of the Horseshoe* application has been in development for roughly three years as of this writing. The initial inception of the project was during the Fall 2011 "Gaming the Humanities" course. Dr. Robert Weyeneth proposed that the class create some form of digital artifact to mobilize the history of the Horseshoe. A previous course led by Weyeneth gathered many historic documents about the Horseshoe and the slaves that built and maintained the campus [Weyeneth et al. 2011]. Two projects were proposed during the Fall 2011 course based on the Weyeneth, et. al., website. One was an interactive fiction, the other an iPhone application.

Professors House, now Lieber College (1837), Elliott College (1837), Pinckney College (1837), the South Caroliniana Library (1840), Harper College (1848), Legare College (1848), and the Fourth Professors House, now Flinn Hall (1860). The buildings that do not remain are First Stewards Hall (1806), First Presidents House (1807), and all but one of the slave quarters. Many other utility out buildings also do not remain [Weyeneth et al. 2011].

The interactive fiction was built by colleagues Renaldo J. Doe and John Hodgson. They used Unity™to build a digital Horseshoe for an interactant to explore. As the participant explores the Horseshoe, he can click on objects to get pieces of history and dialogue. For example, a hammer tells the story of a builder who created the wall, while a protest sign divulges the history of student revolts at the campus. The second project, and the one that grew into the current application, was an iPhone application. This version of the application was spearheaded by Grace Hagood. The initial concept was to allow an interactant to walk the Horseshoe. As he explored the Horseshoe, GPS would trigger augmented reality with video or still image overlays of slaves or students to present some form of content. The content ranged from historical dramatizations to poems and short stories about the slaves at the historic campus. It was planned for the iPhone because many of those working on the project had mobile devices and insisted on it being portable. This version of the application used history as a springboard but did not do more with the source material. While the application did receive some praise for novelty, the lack of historical accuracy and the liberties with such sensitive source material did not give the application academic credibility. The original versions proposed were more gamic and were designed around the concept of digital fiction.

The idea of such a mobile application was well received. The original idea was expanded to be an iPad application in the "Critical Interactives" class offered by Cooley and Buell in the fall of 2012. More historical content was added to the application and a stronger emphasis on historical accuracy enforced. Once the concept was more fully fleshed out, it became apparent that fabricating content to produce a fiction was not going to be productive to the message being told. In conjunction with historians, artists, and videographers, the a second version of the application was developed. This version of the application was very menu driven and structured content differently from the current *Ghosts* version 2.0. This second version of the

application relied on a spinning wheel to "dial" in a specified time period. Once the time period was selected, the participant was allowed to view the map screen which had a background map that accurately represented the Horseshoe and content that was related to the given time period. This iteration relied heavily on built-in menu-driven systems, with many required button presses to access content. A participant had to get through the main menu screen, the content time period selection screen, and the GPS activated content point to access one piece of interactive content. These types of content consisted of Augmented Reality (AR) images, a picture, loading a camera overlay and a list of text based content points. The types of content were more important than the content and a minor update to that version had multiple versions of these types of content at each content point (albeit by adding another menu). The data structure underlying the application was also very complicated, with much of the data buried deep in tree structures (site location - type of content - time period - location of subcontent - subcontent information). The menu system was also buried in the participant icon button, which also returned the participant to the time period selection screen. At this menu, the participant could change the time period and return to the map, see everywhere he had traversed, submit feedback, change options and view a tutorial. Overall, this version was very clunky and inefficient.

Over the following year between the"Critical Interactives" course in the fall of 2012 and the second "Critical Interactives" course held in the spring of 2014, the second version of the application was streamlined. The menu-driven nature of the application was removed, with more focus on the content being presented in the application. The time period wheel was removed, all forms of content were given the same priority, the introduction screen removed a screen press, and the options menu was relegated to the participant icon and only accessible from there. It was decided that all content should be available all the time, thus removing the need for a time wheel. A participant would be able to walk to a GPS site and see all content

without having to exit the viewed content to find a menu, change the time period and then re-walk the path to re-trigger the site location. All content was given the same form of treatment and the original major categories (AR point, pan point, etc.) were pushed into content points (thus reversing the methodology of how the content was represented). Instead of treating AR or panoramic views as more important than text and images, AR and panoramic views were converted into assets with the same content status point of access as text and images. The introduction screen was also revamped to show the name of the project and quickly transition to the map to allow participants to engage with content faster. Finally, the options menu was relegated to the participant icon and was only accessible by pressing that icon. This contained all older versions of the options menu, but the review screen of where the participants had walked was given precedence over changing options or submitting feedback. Much of the menu systems were removed to make the content more of the focus and many features that were deemed necessary at first were scrapped due to their being distracting (such as the ability to select a time period at any time to restrict content). This revision was the alpha build for the final application that has now been constructed. This version of the application was tested on multiple occasions, and it was found to be very difficult to use. There were too many button types, and it was not easy to enter the options menu. While the final version of the application has been built with the previous builds in mind, it has also included additional types of content, connectivity to a server for new content to be added easily, and has fixed many of the design problems of the previous versions. Full detail of the final version of the application can be found in later chapters.

Given much of the discussion of framing, it is important to understand the theory behind each part of *Ghosts of the Horseshoe*. When it comes to content, there are segments of text for participants to read, images for participants to manipulate (e.g., historic photographs that fade in and out according to "pinch" or "pull", zoom-in on

to analyze details, and 19th-century script translation for old documents, eventual video and the interactive component). The text provides information gathered from the Weyeneth, et al., website [Weyeneth et al. 2011]. All information was historically verified through historical documents. Some content was rendered by an interpreter and cited properly to help those using the application understand which is interpreted and which wis genuine. When it came to images, only historical images were chosen and only images that had more historically relevant information were desired. One of the major themes of the CI is that there is an absence of evidence about the events that transpired and persons who lived and worked on the historic Horseshoe. The theme of absence was to provide more concrete evidence to those interacting and give a more tangible idea of specific historical proof. Recipes and hand written records are used to show what little information still we have about enslaved persons.

When it comes to the interactive experience, the idea was to set the rules to allow exploration and discovery. The thumb print icons fade in and out based on proximity to content points: nearing a content point results in an increasingly opaque icon, while moving away results in greater transparency. This means that the further you are from a specific set of content, the more invisible it becomes. Once content is triggered at a site location, the application loads up all content related to the location and presents it in a fact-sheet-like fashion. Given the public history slant, this allows for more control over the information being presented and the small snippets of information allowed for participants to not get overwhelmed. GPS is used to track the location of the participant to make traversing the historic horseshoe easier. The thumb print is representative of the content at each site. Given that no images of slaves from the South Carolina College exist, the one artifact that the slaves left behind was used as the content location identifier.

2.10 Conclusion

The digital humanities play a key role in constructing the background of a CI. Notions of awareness and framing as well as use of language are all derived from DH thought and help build the concept of a CI. Given that *Ghosts of the Horseshoe* is a collaborative project that used not only computing and human computer interaction to build the project, but also the philosophy of DH, this project is truly multidisciplinary. It is possible to argue that there are three components required to create a CI: the procedural rhetoric, the usability, and substance. DH helps provide the substance, the framing of the substance, and the context of the substance.

Chapter 3

Human Computer Interaction (HCI)

As technology advances, so does the need to understand it. Computers were originally large and bulky; but they became smaller over time because of advances in technology. From giant mainframe machines used for mathematical calculations to personal computers on everyone's desk to pocket computers that cannot be differentiated from rocks or clothes, computers are getting smaller and ever more present in daily life, and people are less aware of their presence. The third wave of computing, ubiquitous computing, is a generation of computers that have shrunk to be so small that they can be and are incorporated everywhere [Dourish and Bell 2011]–clothing, objects, material landscapes. Because computers are getting smaller and more discrete, thus more ubiquitous, understanding and learning how humans interact with them is becoming more important. On mainframe machines and early personal computers, the target audience was commonly more tech-savvy individuals who were comfortable entering text commands to get the machines to perform tasks. As PC's became more global and everyone began owning one, how information was presented and how individuals were able to interact with the computer became more important. The study of how humans interact with computers and methods designed to help measure and improve usability is known as Human Computer Interaction (HCI).

HCI commonly focuses on the "design, evaluation, and implementation of interface computing systems for human use and the study of major phenomena surrounding them" [Association for Computing Machinery 1992]. This focus strictly looks at computer systems and how one-or-more humans interact with one-or-more machines.

While many consider the "GUI" system and its ease of use to be the specific goal of HCI, the fact that humans, machines and interaction are vaguely defined concepts expands the notion of HCI to more than just interface systems. As defined by ACM, HCI is "concerned with the joint performance of tasks by humans and machines; the structure of communication between humans and machines; human capabilities to use machines (including the learnability of interfaces); algorithms and programming of the interface; engineering concerns that arise in designing and building interfaces; the process of specification, design, and implementation of interfaces; and design trade-offs" [Association for Computing Machinery 1992].

The discipline of HCI emerged in the context of a set of technological and industry developments: emerging computer graphics technologies, advancements in operating systems, the need for non-technical workers to use computers, and the introduction of cognitive psychology for the purpose of more user-friendly design. Because HCI's roots span so many fields, the discipline has a complex history and varied sites of examination. As computer graphics improved from dots and lights on early machines to the advanced LED screens with high definition images, how humans interact with the display has changed. Operating systems have improved with time, going from wires to text based systems like windows DOS, to point-and-click GUI systems found on most modern PC's to mobile phone button presses. Each new update to an OS brings new features and functionality, which can improve a person's computer experience. With attention to advancements in computing and the lessons learned from previous product failings, HCI shapes how technology design might better address how non-experts and people who have no previous experience with a product interact with that product. The more people who can use a product, the more successful that product becomes. Cognitive psychology, which studies human information processing and performance, adds to HCI by focusing on how humans relate to and interact with computers. Ultimately, the work of HCI aims to build better, that is, more

user-friendly machines.

While HCI is an expansive discipline, it is but a small part of Interaction Design (ID). ID not only looks at the design and evaluation of a human interaction, but also examines the wider theory, research, and practice of designing user experiences with technology. Not only does ID cover issues similar to HCI, such as designing a user experience and testing the experience to see if it is effective, but also more general theoretical concepts. It also examines what constitutes an interaction and what social interactions are, as well as ways to gather and analyze data based on how people interact.

3.1 Usability and Design

It is important in the modern era for computer applications to be user-friendly. This means that an application must be intuitive and consistent. An application is intuitive if a participant can pick it up and use it without needing too many prompts or cues. If a person can complete tasks quickly with the application (i.e., efficiency), without having to spend too much time familiarizing himself with it (i.e., learnability), and is able to repeat said tasks without having to reread a tutorial (i.e., memorability), then the application can be considered intuitive. One common way to make sure an application is easy to learn and use is consistency. Consistency of buttons, layout, and other design features can help prevent confusion and can help users remember how to use an application.

One of the major goals of *Ghosts of the Horseshoe* is to design an experience that uses ludic methods and interactivity to encourage participants to engage critically with the material presented. Engagement can be measured by observing attention, pace, and flow. Measuring attention and pace can help to determine level or degree of engagement. Play, interactivity, and style of narrative also play some part in a participant's engagement. Flow, first introduced by Csikszentmihalyi, refers to the

intense emotional involvement that participants feel when they are fully immersed in an activity. The more immersed an individual is, the more likely he will respond to and/or retain the experience [Csikszentmihalyi 2008]. Immersion is difficult to gauge. Consider, for example, video game controls. As video games evolved, the controls became more streamlined. When the Nintendo Wii released motion controls, some games got harder to play because the controls did not adhere to standard control methods. What could have been a design flaw ended up being accepted by players, however, because the new experience proved more engaging than previous control schemes. Since engagement and flow are important for the development of an application, so is examining the target demographic. While one would like to design an application that targets every possible person, it is not reasonable or feasible. Personal differences among people, as well as differences among social groups can vary enough to make a unifying experience difficult to achieve. Aside from the target demographic, the larger group of stakeholders also plays a role in determining usability and design. A stakeholder is a group of "people or organizations who will be affected by the system and who have a direct or indirect influence on the system requirements" [Kotonya and Sommerville 1998]. The stakeholders include developers, managers, direct users, as well as those who can potentially lose work because of the application, etc. [Kotonya and Sommerville 1998].

3.2 Effectiveness of Design

Once an application is built, it is important to determine if the combination of design elements results in a good or productive experience. If all parts do not work collectively, then the experience can become disjointed. For example, the Nintendo video game *Dr. Jekyll and Mr. Hyde* asks players to control Dr. Jekyll as he walks through a level. As the player walks, some enemies hurt Dr. Jekyll while others do not. Attacking does little to no damage to any enemy even though it is clear that

enemies need to be attacked. When the player dies, he turns into Mr. Hyde and the video game plays a constant moving segment where the player shoots projectiles towards enemies. The goal is to defeat foes before the player character arrives at the location of Dr. Jekyll's death (i.e., in the preceding level). The system's rules make no sense and the overall experience is disjointed. Distinguishing friendly NPCs from enemies is impossible and the win condition is never explained to the player. While some experiences attempt to have the player engage and learn about the system as part of the engagement, Dr. Jekyll and Mr. Hyde gives to much contradictory information to make discovery of the system possible. When designing an iOS application, similar issues need to be considered to prevent it from being confusing. Too many button choices, hiding buttons in the background, placing relevant information on scroll views and then not telling people that the information exists, moving the location of menu items on each screen can all create an awful, or at least frustrating, experience.

Design Assessment takes each component of the overall design and examines if each part is performing its task functionally and is user friendly. If a user at any time gets lost, frustrated or confused, then the product has poor design. Keeping the application consistent, such as mirroring common design tropes from other applications, using one style of button, keeping the layout similar at all times, helps users navigate the application.

3.3 User Testing / Developer Testing

It is important to begin evaluating and testing a product as early as possible, since discovering design issues in the middle or at the end of the development cycle can prove damaging. Of course, one can ignore design flaws. More frequently, developers attempt to fix flaws. In some instances, derivative flaws persist resulting in a dysfunctional product going to market. And sometimes, fixing a flaw means a delayed

release. The further in development a project is when a design flaw is discovered, the more time and money it takes to fix. While it is important to evaluate at the beginning of the design and development processes, performing formative evaluations and summative evaluations throughout improve test design effectiveness. Formative evaluations are performed during the design and implementation of the product. These include prototyping and tweaking the design based on feedback. Summative evaluations are performed on the finished product in order to determine the possibility of future editions [Kotonya and Sommerville 1998].

There are three types of testing that can be performed to determine if a product pleases stakeholders. These three types can be seperated into two major categories, being User Testing and Developer Testing. The three types of evaluations are Controlled Settings, Natural Settings, and Developer Accounts. Controlled Settings evaluations involve users commonly performing activities in a controlled environment to target a specific hypothesis and measure individual components. Controlled settings tests are great for targeting a specific piece of the product, but can, at times, require large sample sizes. Natural Settings tests evaluate the product in a natural setting and attempt to determine the product's usability in the real world. This is commonly done through field studies. While the last two types of evaluations focus on users, the last evaluation test primarily focuses on developers. Developer testing consists of developers, consultants and researchers critiquing, predicting, and modeling aspects of the product to improve usability. The goal is to determine obvious usability problems through inspections, heuristics, walkthroughs, models, and analytics [Kotonya and Sommerville 1998].

3.4 Critical Interactives and Empathic Awareness

Interaction and how a participant reacts to a design logic are very important to HCI. Commonly what is asked when designing, building, and assessing an application is

"What will be successful?" In which case, a developer must have a sense of what constitutes success. For a CI such as *Ghosts*, success is measured by increase in empathic awareness. *Ghosts* does not ask participants to sympathize with enslaved labor; it asks its interactants to express a greater sensibility for the fact of enslaved labor. Sympathy emerges from direct experience; empathy is about a shared understanding, one that is based, not on direct experience, but an imagined relation. *Ghosts* uses simulation to inspire in its participants a sense of the history of enslaved labor. The goal is to make them empathize with the history through experiencing a concrete, tangible–quite literally by means of the touchscreen interface–relation to the past. It is not politic to simulate slave life on the Horseshoe. But inviting participants to imagine how the Horseshoe came to be and under what conditions–and to question what has changed or remained consistent–is possible. *Ghosts* attempts to bring to visibility a history that many would like to forget. It uses rules to invite an empathic understanding of past transgressions.

Chapter 4

Computer Technologies

> "In our view, computing is fundamentally a modeling activity. Any modeler must establish a correspondence between one domain and another. For the computational modeler, one domain is typically a phenomenon in the world or in our imagination while the other is typically a computing machine, whether abstract or physical. The computing machine or artifact is typically manipulated through some language that provides a combination of symbolic representation of features, objects, and states of interest as well as a visualization of transformations and interactions that can be directly compared and aligned with those in the world. The centrality of the machine makes computing models inherently executable or automatically manipulable and, in part, distinguishes computing from mathematics. Therefore, the computational lists acts as an intermediary between models, machines, and languages and prescribes objects, states, and processes." [Isbell, Charles, et al. 2009]

The capability of computer technology has developed exponentially since the early days of room size vacuum tube heat boxes. As technology advanced, computers became more efficient and streamlined, moving from large rooms to desktops and eventually to mobile devices in the hands of individuals. With this technological shrinkage, other tangential components of computing also changed. Not only did the ability to interact with the computer (i.e., the input mechanisms of the computer), the cooperative components (i.e., networking, printing, etc.), and the way a computer

is "coded" change as technology advanced, but so did the purpose of computers. The mobilization of CI with *Ghosts of the Horseshoe* is informed by this rich history of computing to help build a meaningful mobilization. It is worthwhile to review the history of people's interactions with computers in the run-up to the mobile boom in order to understand design decisions for mobile devices, to examine how advancements in networking and coding help shape content, and to recognize that "coding" has become more than just rules and data.

4.1 The Building Blocks of Mobile Technologies

As mentioned before, computers have gone through three waves of development. The first wave of computers used vacuum tubes and magnetic drums to process and save data. As technology improved, vacuum tubes were replaced with circuit boards holding transistors, and then integrated circuits. The first wave also introduced the keyboard, mouse, and computer monitor. This wave of technology introduced many common design choices seen in later generation devices, such as mouse hover-over and keyboard shortcuts. With the invention of the microprocessor, computers became more affordable, and the second wave of computing, the personal computer, soon followed.

The second wave of computers saw the advent of many different devices, including video game consoles and personal computers. These were small, but not usually handheld, computers that used inputs similar to those of the first wave. Personal computers were built with the "everyman" in mind. In other words, PCs (back in the day when "PC" meant "personal computer" and could even include a machine bought from Apple) had to be simple enough for everyone to use. As technology improved, the computers became smaller. This led to the third wave of computing, ubiquitous computing.

The third wave has introduced many complications that did not confront the two

previous waves of technology. A principal difficulty involves user-device interaction. Computers are now small enough to fit in cell phones (or be cell phones), car tires, nano-robots and/or medical devices (e.g., pacemakers). The traditional method of using a keyboard and mouse has changed. A smart phone, such as the iPhone, does not come with an external keyboard or mouse. Instead, the screen of the device is used for input. While there is a digital keyboard on the device, its function and use are more cumbersome given the small size. Absent the tactile feeling of the keys (even if the audio keystroke cues can be turned "on"), the interaction with the iPhone ends up being very different from the interaction with what is now referred to as a desktop computer.

Another example of this shift can be seen in software and coding. Ada Lovelace, creating programs for Charles Babbage's Difference Engine, is usually credited as being the first programmer. While she used mechanical means to produce code, the ideas she created continue today. Eventually programming would transpire via punch-cards, circuit boards, and paper tape. Decks of cards would be fed into a machine that read in and executed the program. As technology moved digital, so did the coding. The von Neumann architecture, in which code and data reside together in the same memory, has become the norm, and this major decision has led to many programming paradigms today. From binary, to assembly, to higher level languages like C++ and Objective-C, the idea of communicating with the computer has remained the same [Dale and Lewis 2011]. Most modern applications are built with high-level languages. This allows for easier creation and maintenance because the program is written in plain text and not in binary.

Even the execution of programs has evolved with technology. For PCs an application required to be downloaded and a prompt menu would appear to help calibrate settings and install the program. This could be as simple as a couple of button clicks or as complex as building the backend of the software. As technology grew more

ubiquitous, program installation became simpler. For smartphone devices and tablet computers, all that is required is a single press download and the operating system handles everything else. This is good because it is easy to use, but it also limits functionality. A person is not allowed to adjust settings, speed, or other application variables as was possible with personal computers. Greater constraints result in a trade-off that has to be considered while building a mobile application.

4.2 Networking, Database Design and Security

Computers have advanced to the point where connecting physical wires to create networks has been replaced with WiFi, Bluetooth, etc., signals, and thus new definitions of a "connected" system have arisen. The ability to connect any device to a network, no matter where the device is located, has revolutionized the physical placement of "computers". Apple iBeacons, for example, can be buried in the ground and yet still connect with other devices. With the unlimited locations where one can now find computers, it is important to think about what constitutes establishing a network, designing data structures and storing data for use, and implementing the security governing the storage and use of the data.

Networking describes how multiple computers interconnect. A standard network usually consists of a few machines, but given the advent of smart phones, networks have grown expansive. The Internet connects many different devices together on a global scale. Any device around the world can download an application, say *Ghosts*, and experience its content. Given this, special considerations, such as what to do when individuals are not on the Horseshoe, need to be considered. Given that there is connectivity within *Ghosts*, it is important to understand how *Ghosts* connects to the Python-based server. The network and bandwidth available have to be considered when passing data. If the network transfers data slowly, this can affect the application. *Ghosts* connects to a Python-based server to pull content information.

This allows the application to change and update content without having to change or update the application itself. This relies on solid database design and security to help ensure the integrity of the experience. Database design consists of taking a known set of data and breaking it down into tables to help prevent duplications. This improves the internal validity of the data and provides consistency among data variables. When the data is passed over the network, it needs to be secure. Since transferring information over a network consists of breaking the file into packets and transferring said packets, it is possible for these packets to get intercepted. If a hacker intercepts the packets, it is possible to steal information or change what is present. This is not a major concern given the scope of this application, but it is important to consider proper security when dealing with networking. Since WiFi is wireless, it is important to understand how vulnerable data can be. A hacker can simply walk by a phone, and with the correct code, read data through the signal. If the device is transmitting, security can be compromised and data can be stolen. It was important that *Ghosts* was written with consideration for the network, for database design, and for security.

4.3 Experience = Data Plus Code Plus Users

We suggest by the title of this section that data, code, and users together work to create a meaningful interactive experience. When the title of this section is broken down into each of its component parts, each part provides a unique facet which reflects the main chapters presented in this dissertation. "data" embodies the digital humanities and "user" mirrors human computer interactions. In this chapter we will discuss "code".

The word "data" means different things in different fields of study, but it is commonly associated with some set of facts associated with a given topic. In computer science, "data" usually references raw facts, facts that have not yet been processed,

that exist in a pure state, and that can appear to mean absolutely nothing. This can be seen in small data, such as simple grocery lists, to big data, like Facebook profiles and posts. Raw data makes no real sense until it is processed into "information". Individuals take raw data and apply analytical and statistical tests to discover important "meta" facts about the data. Processing a simple list of groceries could tell us more about the person who purchased the items, or give clues regarding the buyer's shopping habits. By processing Facebook profiles and posts, one could create a model to predict common things that people say or how often a specific idea is referenced, or to parse out hidden underlying messages buried in the Facebook posts. How this information is used is "knowledge". This might include using the processed data to influence the buying habits of shoppers. By looking at a simple grocery receipt and processing items purchased, one could create a system to push the shopper into making more purchases of a particular sort. If a shopper is seen to buy a six pack of beer on a Tuesday, an offer of a discount the following Monday might spur more sales.

Knowledge allows powerful entities to "mine" out special target groups that may be hostile, determine the most discussed topics, and find a way either to repeat such events or even to disrupt political movements by defining the movement's values and goals and undermining them in some fashion. The use of knowledge can help disrupt, dismiss, or sell, but it can also educate, illuminate and teach. As discussed before, how a CI is framed given a digital humanities lens helps to promote ways of thinking and understanding. The process of looking at the human condition, examining the parts to generate concepts and ideas, and then using digital technology to express these ideas is very similar to the data/information/knowledge model. It can be argued that for *Ghosts of the Horseshoe* the framing and the historical content is the data of the application.

In *Algorithms + Data Structures = Programs*, Niklaus Wirth [Wirth 1976] explains

that "code" is traditionally thought to be datasets together with algorithms defined to work on the datasets. This is commonly seen in software packages designed purely for computational processes. An individual uses Microsoft Word, a basic accounting package, or a statistical software package like SAS to input some data, process the data, and output results. While the data and the computation of the data changes for each individual and for each use, the overall experience remains the same. The notion of programs being just code and data is contested by two differing arguments, one by copyright and one by digital cultural artifacts.

With a traditional software artifact, like an accounting package, the software can largely be described as code acting on data. When one is designing a digital cultural artifact, however, such as a video game or a critical interactive, there is more to the software experience than just the code and data. The software package requires some form of input, — from another computer or from a user. A user of a program normally is just that, a user. The person uses the package as a tool and treats the package as an asset for completing a job. The application could produce a more meaningful experience, with a person being invited to engage. If the experience requires more, but does not ask much out of the user, then the user is a participant. The individual is asked by the set of rules to partake in an interaction that is more than just routine, but the application is the primary controller of the experience. If the rules established require more than just input and the application requires that the interaction with the system is directed by the participant, then the participant becomes an interactant. While at first glance, the three terms may seem synonymous, all terms describe very different experiences. A user just uses an application, while a participant is lead, and an interactant shares in the process of production.

Does the type of interaction have any legal bearing on the software (i.e. does the program provide something more than just being code)? As described by Calvin Mooers the meer act of "running a program is an infringement of copyright" [Mooers

1975]. This is based on the argument that executing the code on a machine is always different, similar to how a play is performed on a stage. As in a play, the actors will stand in different places, with different postures, and will recite their lines slightly differently with every iteration. The lighting will change given the equipment and time of day, and even the audience will be different with each rendition. Software shares similar traits, with data being loaded in different registers, the data being processed being different, and the operator changing. Mooers' paper paves the way for licensing methodologies and subscriptions because it argues that there is more to programs than just data and code. The act of executing the software and the variables that accompany it are also considered.

While the argument of copyright is correct in discussing the variations in a computer system when executing a program, there is a more human aspect to software use that contends with Wirth's traditional view of programs. An example of software being more than just code comes from interactive video games. The definition of a game as stated by Salen and Zimmerman is "A system in which players engage in an artificial conflict defined by rules that results in a quantifiable outcome". In a digital medium, these rules are created and enforced using computer programs, or code and data. The software creates a "magic circle", which is "the space in within which a game takes place", and sets the game's rules to create "a special set of meanings for players of a game" [Salen and Zimmerman 2004]. Players are then allowed to explore the magic circle, with the point of the code and data being to establish this system of play instead of to produce a specific output. The purpose of the program is to establish the system of rules for players to experience, not just the set of rules to complete a task. This allows for the same software to create very different and personal experiences for different players.

We use as an example the game *Loneliness* [Magnuson 2011b]. This game is about a single pixel character who is tasked with walking upwards. As the player

walks, he will encounter many other pixels, some standing, some moving in circles, some following other pixels, and others pursuing (or proceeding according to) random paths. As the player approaches these entities, the pixels disperse and run away. This simple mechanic can create a variety of emotions in the player and invoke a multitude of play styles. Should one continuously try to meet the groups of pixels? Should one just give up and continue the quest upward? Or should one change his strategy (perhaps by running faster towards groups or by inching closer bit by bit) to see if there is any form of acceptance? All these different play styles are supported by the same set of code and data. Each person that interacts with the game will play it differently and take something different away from the experience.

Another example is cited by Anne Balsamo in *Designing Culture* [Balsamo 2011]. She discusses different types of interactive experiences, such as the Reading Wall, a giant wall comprised of three separate sixteen-foot long walls, each standing ten feet high. These screens display ever changing texts from a multitude of languages. This platform allows for different unique experiences for many. Each person experiences the interactive differently, given the text that is being display and the emotional investment of the viewers. From this we can observe that there is more to a program than just code and data.

Loneliness and the Reading Wall are both examples of programs doing more than just processing data. They show the human component that exists when a individual interacts with computer software. Given this, Wirth's notion of programs being code plus data needs revision to include along with the code and data, the human interaction. In the following chapter, the code of *Ghosts* is examined to help emphasize and discuss many of the points raised in this chapter.

Chapter 5

Ghosts of the Horseshoe iOS Application

Ghosts of the Horseshoe is built for the Apple iPad device. An Apple family device was chosen due to Apple's standardization, including that of screen sizes. Both the iPad and the iPhone have two established sizes: (1) the original iPad (9.5 by 7.31 by .37 inches) and the generation 1 iPhone and subsequent models 3 and 4 (4.5 by 2.31 by .37 inches); (2) the iPad mini (7.87 inches by 5.3 by .29 inches) and the iPhone 5 series (4.87 by 2.31 by .3 inches)[1]. This allows for simplifying application frame sizes, unlike Android devices which vary in size according to company. Another reason for choosing an Apple family device was the portability and adaptability of the code for the device families. A project created with one device in mind can be easily ported to another. The iPad was specifically chosen because of screen real estate as compared to the iPhone device and because iPads were readily available for testing. While this decision allows for larger screen real estate, it does limit possible dependable functionality. For example, we do not have solid GPS tracking due to the fact that our test iPads have no network carrier and, therefore, must rely on a third-party device–the Garmin GPS peripheral–as well as a reliable WiFi signal for internet connectivity.

Unlike development for an Android platform, apps for Apple devices must be developed on Apple computers, using Xcode, the Apple-supported development environment that allows one to write code that can be tested using the built-in simulator.

[1] It is anticipated that Apple will be releasing the iPhone 6 in fall 2014 and that it will be available in two sizes–a 4.7 inch version followed by a 5.5 inch version. ("iPhone 6." Mac Rumors. 2000-2014. Accessed 22 May 2014: `http://www.macrumors.com/roundup/iphone-6/`)

We have tried to stay up to date with the most recent version of Xcode. At the time of this writing, Xcode version 5.0.0 was used on a Mac OS X 10.8.4 system. Applications that run as native programs for iPads must be written in Objective C, as compared to applications being coded in Java for Android devices. Objective-C is a superset (or descendant) of the C programming language, similar to C++ or C#. As a superset of C, Objective-C inherits much of the primitive types, simple syntax, and control flow structures that originated with C. C does not have many modern programming paradigms, so object oriented methodologies, language-level support for graphics management, the ability to define class structures, and dynamic typing have been added to make Objective-C more dynamic and object oriented.

While the base language of Objective-C is complete, much of its power comes from supplemental libraries and Application Program Interfaces (APIs). APIs are used to add new functionality to a programming language while affording selective choice of functionality. A common API, the inherited base class `NSObject`, is a standard library that was introduced to Objective-C with the software API OpenStep. This library, known as NEXTSTEP, was released by Apple in September 18, 1989 and allowed for more advanced Graphical User Interface (GUI) systems. Other known APIs provide ready access to display, color, touch sensing, Augmented Reality, Voice Recognition (e.g., Siri), etc. While providing for new functionality, APIs tend to force an application to fit into an overall set structure [Singh 2003].

There are many different software libraries/packages that are provided by Apple, with many more being added with each update to standard template libraries. There are also user/professionally distributed libraries, often supplied on GitHub (an online repository for code sharing), which provide further functionality options. The ability to add just controls of augmented reality without adding in unneeded functionality of photo image processing helps with keeping overall code structures simple and efficient. Apple supports many common functionalities, such as the ability to play audio and

video, and to modify images. These are done with common libraries that are used so often that Apple has pre-built them into Objective-C's standard libraries. Other not so common functions usually have a respective library created and supported by a third party. In the case of *Ghosts of the Horseshoe*, Qualcomm's Vuforia™was included to allow for augmented reality support, `AFNetworking` was added to simplify connections with the backend server, and Cocos3D was added to allow for 3D model viewing. These different libraries are being used because Apple does not currently support them; and although Apple continues to develop and add new libraries to the Apple suite that comes with updates of Xcode, the company has yet—that is, as of Xcode version 5.0.0—to include libraries that provide the kind of functionality that *Ghosts* requires.

Extra libraries for an Xcode project have to be connected using Xcode's project settings tab. All imported libraries, the type of device being used, a connection to any "Storyboard", the app delegate, and many other project settings are present in this section. Given that 3D party libraries are not built in, linker references must be established so the project has direct access to the libraries; otherwise the project cannot find the referenced libraries. When everything is connected, all major libraries must be included in the `prefix.h`, so the entire project has access. The `prefix.h` file is a global header file that the iOS application uses to make sure all classes inherit specific libraries and APIs.

5.1 The Application Flow

A newly-created "single view" app in Xcode is provided with an "app delegate" class and a "view controller". The displays on the screen that a user sees, referred to as "views", are controlled by the view controller, with a background "navigation controller" containing the many view controllers. The navigation controller is a standard class for the display of multiple views and performs this task by implementing a stack

of views. When the app begins execution, control is transferred from the "main" to the app delegate, which acts as the central hub. The app delegate sits at the top of the application and functions as a kind of puppeteer. The app delegate controls many of the interactions with the device's operating system (OS). If the application is sent to the background on the device, brought back into the foreground, or even terminated, the app delegate has to negotiate with the OS. The app delegate tells when one puppeteer is required to exit the stage and when the next one is allowed to put on a show. Within the app delegate the initial action is to push a "root view" (e.g., a splash screen) to the navigation stack. Once this base is established, it is possible to push and pop different views on and off the stack. When a new view is required, it is pushed onto the stack, keeping the entire previous view intact but in a paused state. When a view has finished its usefulness, it is removed (see Figure 5.1). One peculiar property of the navigation stack is its ability to break stack rules. Commonly, only pushing and popping are allowed when interacting with a stack structure, but in Objective-C it is possible to treat the navigation stack as a linear array and to remove a view from the middle (e.g., steps 6 and 7 in Figure 5.1). This is helpful for modifying control flow of the application, because there are instances in which a pure stack can be cumbersome.

In Xcode, there are two methodologies for both (1) creating and maintaining the elements displayed on the screen and (2) the flow of the screen display in the navigation stack: using the Storyboard or doing it programmatically. The Storyboard method appears to be Apple's preferred method. The Storyboard is a drag-and-drop interface that simulates an iOS screen and the flow of an application. It looks like a state diagram with each view appearing with arrows representing segues to subsequent views. The interface calls variables by means of arrows and the setting of properties by selecting options from a menu (e.g., setting font sizes, font colors, background colors, etc.). While it can look very sleek and clean, using the Storyboard method

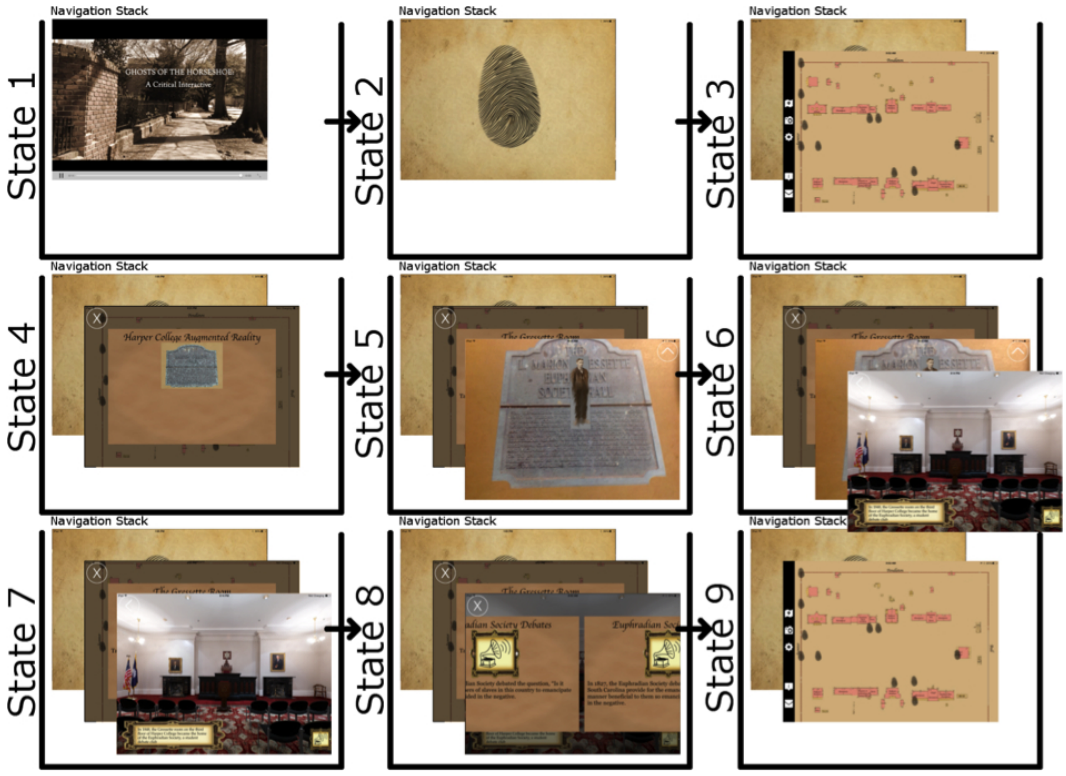

Figure 5.1 Stacks of Views

separates UI elements from their instantiation. To edit the properties of an object (e.g., size, font style, color), a programmer has to look in both the Storyboard editor to change the properties and within the code to change the functionality. We found it overly complicated that things are hidden in the background and that we must use Apple's requisite commands to connect simple elements to code fragments. We prefer to implement both screen display of elements and flow of control by using the Objective-C program language directly. We create all objects in code and manage the navigation stack manually. While this is initially more difficult to set up and maintain, we find it easier overall because all related code is in the same location.

5.2 Objective-C Classes

We have made use of only a small subset of the many classes made available in Xcode, Objective-C, and the iOS libraries. *Ghosts of the Horseshoe* makes extensive use of

UIViewcontroller, UIView, UIButton, UIScrollView, and UIGestureRecognizer. In conjunction with the navigation controller, a UIViewcontroller helps to control the flow of the application. It is an intermediary between one or more application views whose function it is to interpret user interactions with a view's content features and to request actions from the operating system so the device knows what actions to perform. The UIView is where all visible elements of the application are presented, including images, buttons, text fields, etc. A UIView can also be trained to recognize gestures and relay them to the view controller for action. The UIViewcontroller and UIView are part of a Model-View-Controller design pattern that specifically separates user interface (UI) display and background control to simplify coding.

Similar to the app delegate's control of the schedule for the puppeteers and the navigation stack's control of the foreground, the UIViewController and the UIView control the setting of the engagement. The navigation stack sets the location, but it does not set neither the style of the stage nor the background props. UIViewControllers and UIViews both inherit from the NSResponder NSObject in the UIKit framework, but both have different functions. UIViewControllers are built for navigation and have many functions designed for interacting with the stack. UIViews are built for drawing to the screen and commonly have functions designed for modifying the visual display of the view. Only UIViewControllers can interact with the navigation stack. This means that neither buttons nor gesture recognizers created in a UIView can directly transition to a new view; such transitions must be done by invoking a delegate or passing through the Notification Center. The assets in the UIView are implemented in the UIViewcontroller. The UIView assets are free-standing and can be transferred from one view to another, but a UIView cannot be displayed to a screen without a view controller. Conversely, the UIViewController is not as flexible in its display capability as a UIView. While a UIViewController has a primitive UIView present within it, creating an independent UIView object helps

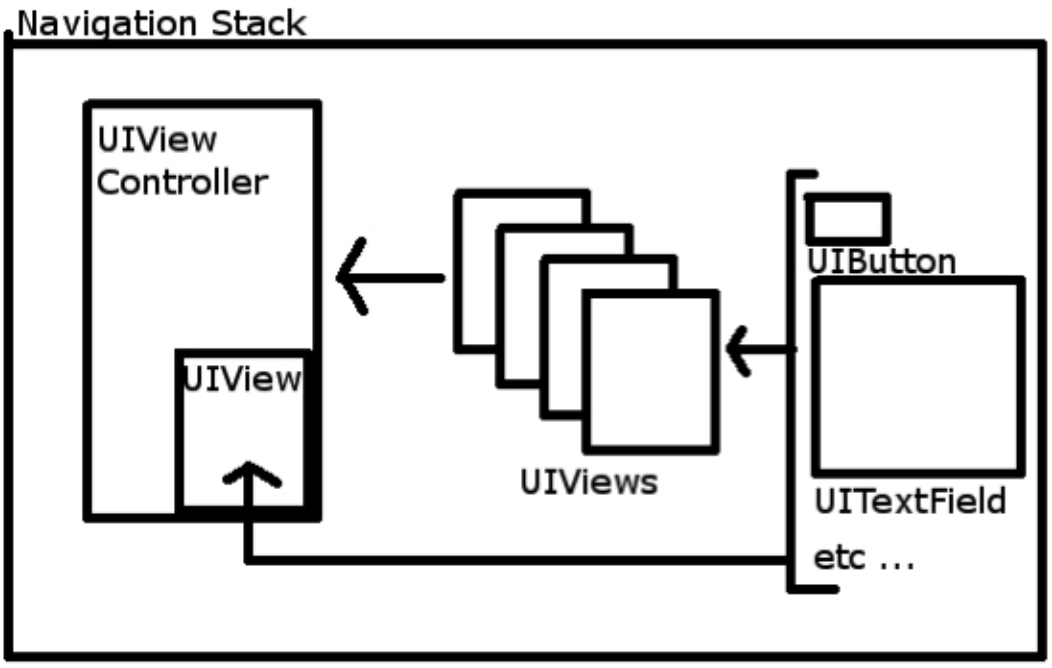

Figure 5.2 Views and Subviews

to modularize large parts of code and simplifies structure overall. Each `UIView` can contain a single idea and loads all related displayable content. Stacking each `UIView` layer on top of each other allows for a multi-part dynamic display that separates all complicated parts into modules but creates the illusion of single image. A good example of this dynamic in *Ghosts* is the participant icon trail that is drawn onto the map in real time. The map and participant icon are all stored in separate `UIViews` that are connected to the `UIViewController`. The trail that follows the participant icon around the map can only be done in the `UIView` with the `drawrec` method. (See Figure 5.2.)

Other class objects such as `UIButton`, `UIScrollView` and `UILabel` add special functionalities common to interactive applications. `UIButton` is derived from the base class `UIControl`, which is used to convey user intent to an application. While `UIControl` cannot be used directly, its subclasses are used by many objects to establish a common behavioral structure. `UIButton` allows for "button" events, such

as pressing inside or outside a button, holding one's finger on a button, etc. *Ghosts* uses buttons for loading content, switching to and returning from views, and for directing the application to play video and audio assets. `UIScrollView` is a special view with a built-in gesture recognizer that implements scrolling. Gesture properties like multi-touch swiping, tapping, pinching, rotating, multi-finger actions, etc., are implemented in iOS applications using the `UIGestureRecognizer`. Unlike the `UIbutton` or `UIScrollView`, which are both implemented with a visual cue on the screen, the `UIGestureRecognizer`'s presence is invisible. The `UIGestureRecognizer` records and waits for actions. We have used both buttons and gesture recognizers in *Ghosts*; they provide transitions of content views and cause actions via the navigation controller.

Within Class objects, there are two types of functions: "class functions" and "instance functions". Class functions are usually signified by a "+" in Xcode and are global functions of a class that can be called at any time. Instance functions, signified with a "-" in Xcode, can only be accessed by creating an instance of the class and calling the function from the object. This is important because class functions can be called globally without having to create an object, while the instance functions are perfect for maintaining specific individual objects. One issue that we have had to consider while coding *Ghosts* is that class functions and instance functions cannot mix (e.g., a class function cannot call an instance function and vice versa).

All constructed classes have an associated header file (`filename.h`) and an associated implementation file (`filename.m`). The header file contains forward declarations of all methods that need to be accessed by other class objects and defined properties of the class. All required imports for the class are also contained in the header file. The implementation file contains an interface similar to what is seen in the header file, but only contains global variables. The implementation section in the implementation file contains any synthesized variables at the top, followed by the standard initialization

functions, defined initialization functions, and functions used to maintain the class.

5.3 *Ghosts of the Horseshoe* Class Structure

The flow of *Ghosts* can be broken down into several major parts: Global Control classes, Main View Controller classes, Content View Controller classes, general utilities, licensed libraries, and the Content Management classes. Licensed libraries include `AFNetworking`, Qualcomm's Vuforia™AR, and Cocos3D. The Global Control classes contain the classes `GlobalState` and `NoteLogger`. `GlobalState` contains all the variables and functions required by the entire application. These variables and functions include the size of the device screen, the size of the font used throughout the application, the color and style of the font used in buttons and labels, important images that are used repeatedly, and the calculations that determine the location of the Horseshoe.

`NoteLogger` records every action that is performed within the application. All other classes and objects pass information to the logger to be recorded in a timeline. This timeline registers how long a participant stays on a specific screen, which buttons are pressed, how long the participant spends reading content, and where the participant has walked on the Horseshoe. In other words, the `NoteLogger` performs as a scribe, recording everything that transpires during an event. Once a participant has finished interacting with the application, `NoteLogger` saves the record of the experience as a file that uploads from the device to the server at the next WiFi connection.

Other important classes include the main view controllers and the content view controller classes. These view controllers and the navigation controller (the "NavController") control the flow of the application. `S01_SplashViewController` and the `UIview` `S01_SplashScreen` control the application's splash screen.

`S01_SplashViewController` loads the introductory video and then presents the par-

ticipant with a fingerprint. Upon pressing the fingerprint button, the splash screen is pushed to the NavController and implements a transition from the splash screen to the main map screen.

The next view consists of `S02_MapViewController` and `S02_1_MapScreenView`. `S02_MapViewController` is the main hub of the application. This class contains functions that record a participant's GPS location, activate the options menu system, and make possible the transitions to different views. This view uses `CoreLocation` to record the device's GPS location, which stores in a separate array. This location is then passed to the `S02_1_MapScreenView`, which subsequently repositions the participant icon as well as redraws the trail that follows the participant around the Horseshoe. This class also constructs the options menu found on the left side of the screen. It creates the `UIViews` that load the options selection menus and uses notification centers to pass changes to associated subviews. The menu controls font size of all text presented in the content loader, the color of the trail that follows the participant, the kind of map (original Sanborn map, a screenshot of the Google view, or some alpha-channel constructed variation of the two) displayed, whether or not the paths and legend are displayed on screen, the ability to load the camera for taking pictures, and the ability to email comments and suggestions to the development staff regarding future development of the application. If pictures are taken, the application will send the participant to a review screen that will allow him to add text to the image and submit it to a database for review by the application administrators. These screens are part of a separate viewcontroller, but do nothing more than allow for the addition and passing of text to the server.

The final major part of this class is the navigation stack controls. Given that some content points require transitioning to a new screen, there is a notification center in place to catch the command to transition and do so accordingly. `S02_1_MapScreenView` contains the content location icons, the map displayed on the device, and the trail

that follows the participant as he traverses the Horseshoe. These are placed inside a `UIView` class because of special functions that only exist inside `UIView`. The Drawrec function that handles the participant's walking trail is drawn and maintained inside the `UIView`. This function only exists as part of the `UIView` class. The other prominent component belonging to this class is the fingerprint icon buttons. The class loads up two subviews that contain both black and white fingerprints, as well as all associated buttons. The different subviews of buttons exist because of the different colors present on the Sanborn and Google maps. All fingerprint buttons are attached to a delegate method that examines if the scrollview parent object in `S02_MapViewController` is zoomed in or zoomed out. If the scrollview is zoomed, the delegate method resizes the buttons to keep them the same size at all zoom levels. The notification center inside this class listens to the options menu contents, changing the color of the participant's content trail and switching between the black and white fingerprints when a participant toggles from one map view to a different map view.

The next view controllers all have the same basic functionality, but are designed with different displays in mind. `CV_ARViewController` loads up Qualcomm's Vuforia™AR package and instantiates "EagleView" to help the iOS camera recognize QR codes and load associated content. In some instances, images that the AR loads are pressable. If a pressable image loaded by the AR is pressed, the participant is presented with content. For example, `CV_PanoramicViewController` uses a scrollview to place a large image on the background. This image cannot be zoomed in any way but can be panned by swiping left and right. There are also content points that are present on the screen that have the same function as the sites on the map. `CV_PictureViewController` also uses a scrollview, but the main function is to allow the participant to see details present in the image by being able to "expand" (zoom in) and "pinch" (zoom out). Each picture loaded can be zoomed two times its screen size. Some images, such as receipts, display nineteenth-century

handwritten script that many twenty-first century readers find illegible. To address this obstacle, the scrollview loads invisible buttons; that is, button objects, whose presence is not defined by visual cues, are placed on top of each word. When pressed, any associated text appears as reader-friendly typeface. `CV_VideoViewController` loads up any video content. It instantiates an `MPMoviePlayerController` to play the video (as well as animation) and the standard back button to return to the previous screen. Finally, `CV_3DModelViewController` loads Cocos3D to display any 3D models incorporated into the application.

While there are various types of content, the main content class referenced by most view controllers is the `ContentScreenViewLoad` class. This class is called by any site button to search the associated data structure and pull all related content. This uses the site unique ID and the time period to pull all views. The background of the screen is darkened and all content associated with the specific site and time period is presented in a `UIScrollView`. If there are multiple time periods, associated buttons (up and down arrows) are added to the screen to indicate that one can change the time periods. This class is a subclass of `UIView` and uses the alpha channel to place this object on top of everything else inside the view which instantiated the object. All content that is shown inside this class is called from the data structures assemble and port class `ContentStarter`.

Three third-party libraries were included in the application to afford additional functionality that is not supported by base Apple libraries. `AFNetworking` was included to simplify and streamline pulling data from the server. It works by creating a get request from the URL of the location where the data is stored. The URL is provided by the server and is known by the application in advance. The application then waits for a response from the server. Either the data is returned to the application to be parsed into the data structure or the application receives an error. An error means no data is accessible. Factors underpinning an error instance include,

for example, loss of connection, absence of data on the other end, wrong port access, etc.. The data that is returned from a successful query is bundled in JSON. The application takes the JSON file, decodes it into a `NSMutableDictionary` and is used to populate the main data structure tables. These tables are then stored as a plist on the device. The backend server that passes the JSON objects is coded in Python, and has its own built-in JSON modules for compressing and sending data over the internet when requested.

Qualcomm's Vuforia™API was also included to enable QR code functionality. At the time of development, reading QR codes was still difficult. Many different software packages exist to read QR codes, but few allow for the creation of codes from real world objects. Qualcomm's Vuforia™allows the iOS device to recognize real world landmarks as QR codes and subsequently load content. While it is impressive to use any object as a QR code, it does have drawbacks. The uniqueness of the QR reference image, as well as factors of sunlight, weather, and shading, affect how the camera reads the QR object. If the reference image used for the QR code does not have distinguishing features, the software cannot recognize the real world object. Likewise, because we are checking against an image database, an image taken in perfect conditions (optimal sunlight, cloudless sky, absence of shadows, etc.) is only viable if the conditions are recreated at the time of participant interaction. For example, a perfect image serving as a reference for a QR code will not be recognized by the device on a rainy day because what the camera "sees" will not compute as matching the landmark object.

The last third party API used was Cocos's 3D. This is a 3D model-loading software built for Objective-C. This software allows a participant to manipulate any 3D model loaded to the view. An object can be rotated around an axis so that different angles can be explored. There is also the ability to zoom in on details.

Because many objects, such as audio buttons and instances of particular `UILabels`,

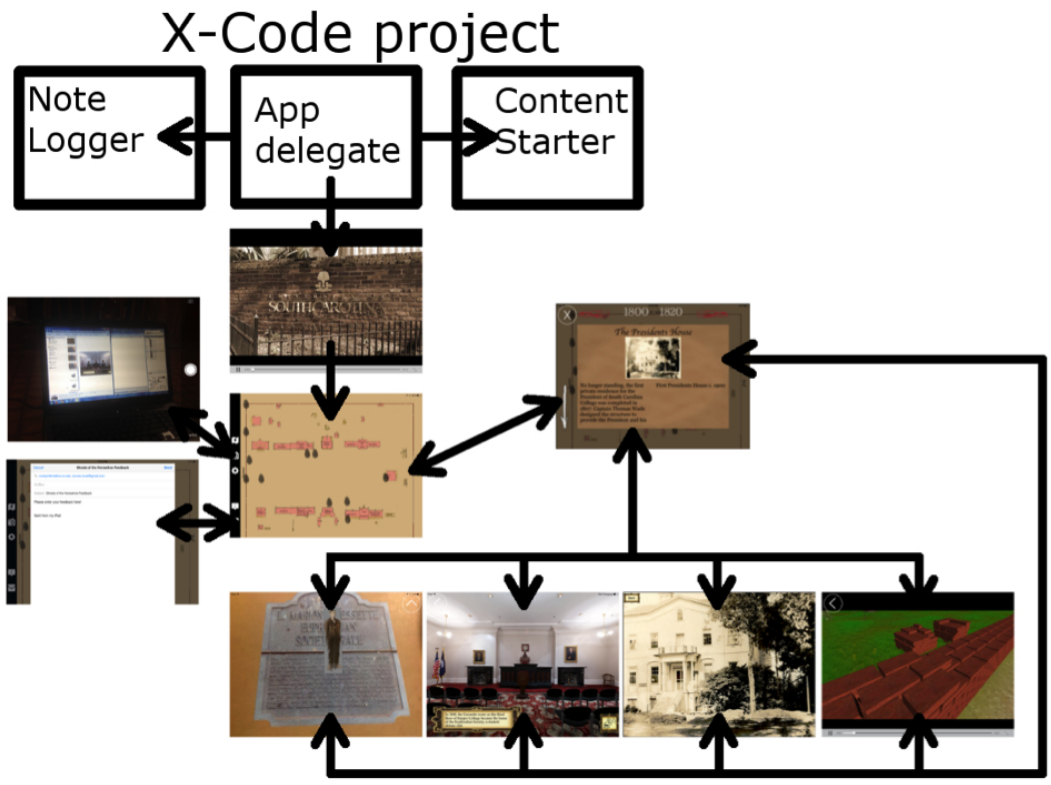

Figure 5.3 Flow Chart of the Application

are commonly reused throughout *Ghosts*, special general utility objects were created. These objects include `GHButton`, `GHTextView` and `GHLabel`. `GHButton`, `GHTextView` and `GHLabel` are all expansions of `UIButton`, `UITextView` and `UILabel`, respectively. These utility objects determine instantiating specific properties (font sizes, font style, color, etc.) and whether or not an audio element accompanies the pressing of a button. This makes the objects inside the rest of the code simpler and allows for easy modification of all similar objects. An example of program flow using the defined classes discussed can be seen in 5.3.

5.4 *Ghosts of the Horseshoe* Data Structure

In addition to class structuring, it is important to understand the data structures in place for holding the content presented in the application. The device's main data

structure mimics that on the calliope server, which hosts seven data tables, each of which contains separate data objects. `T03_Sites` controls all site content. A site is a location on the screen that has associated "content point" information. Each site contains a unique ID, a title, x- and y-coordinates, a site-SubSite ID, latitude and longitude GPS points, and a data table ID. The unique ID identifies specific data objects inside the data structure that belong to a specified site.[2] The siteID is used across related tables (e.g., `T04_ContentPoints` and `T01_SiteDetails`) for querying content that is relevant to any one site. The associated title (or name) is for people editing the database or debugging the data structures. It is easier for a data manager to debug and test a data structure's validity by reading an English word instead of trying to remember an arbitrarily assigned number. Each site has a pair of associated x- and y- coordinates that is tied to the size of the map image. The x- and y- coordinates are associated with GPS coordinates for all map-related sites and are user selected for all other views. The `siteSubsiteID` separates map points from other view controller content points. For example, pressing a thumbprint on the map invokes the same function as pressing the sound button on the Gressette Room panoramic image. If the site is a subsite, then a related `dataID` field is used to determine which view controller and transition background this site appears on.

`T01_SiteDetails` contains all related information needed to display a content point at a specific site. Because not every site has secondary or tertiary backgrounds or audio, the `T01_SiteDetails` table is checked when a site is loaded to the screen. If there are any special backgrounds, or audio or video elements required for display, that is, aside from the content itself, they are stored inside this table. This data structure

[2]It is important to note that any data object may be called more than once. This allows for a one-to-many relationship inside the database (i.e. one site can have multiple content points). Given that there are many pieces of information associated with one site, this not only has to be mimicked inside the data structure (with the use of one-to-many relationships), but also in code (with for-loops to iterate through the many items and dictionary structures holding many items for a specific key). All code was written with the one-to-many concept, making the code more generic and adaptable.

has a unique ID called `SiteDetailsID` and a site reference ID called `SiteID`. `SiteID` is the same as the unique ID in the `T03_Sites` table. Other fields managed through `SiteID` are `SiteDetailsPicture`, `SiteDetailsPictureType`, `SiteDetailsAudio`, `SiteDetialsAudioType`, `SiteDetailsVideo`, and `SiteDetailsVideoType`. These six fields contain picture, audio, video, and the associated file type extensions (e.g., .png, .mp3, .mov, etc.).

The next table in the hierarchy is `T04_ContentPoints`. This table connects the sites (e.g. each individual location) with groups of content to be displayed. `ContentPointID` is the unique ID for the table. `ContentPointTitle` is the title for each content point and it appears at the top of each content point page. This title does not have to be unique for each piece of content: only the `ContentPointID` must be unique. Since the application covers many different time periods, `ContentPointYearID` is used to determine which content point loads according to year range (i.e., 1801-1820, 1821-1840, 1841-1860, 1861-1880). It is an ID because there is a separate table that contains the naming of time periods. The final cell in `T04_ContentPoints` is the `SiteID` field, which connects back to the `T03_Sites` table, linking content points to specific sites. Given this structure, there is a many-to-one relationship with many content points being attached to one site.

Content points contain all presentable information. Because one content point may have many different pieces of content (i.e. text, images, etc.), a separate table was created for gathering all *Ghosts* information. Aside from the unique identifier `DataID`, there is also the connector field `ContentPointID`. Other fields control the type and placement of each piece of content. `xCoord` and `yCoord` control the placement of the piece of content, while the dimensions of the content view are controlled by the width and height fields. The final three fields control the type of content. `DataTypeID` determines if the type of content at the specified location is a video, image, text, or a screen transition. `DataInfo` and `DataInfoFileType` are used to store the content.

DataInfo privileges text over other file types. DataInfoFileType is only referenced when DataTypeID indicates that the DataInfo is not text, but rather another file type with an associated extension (e.g., .png, .mov, .mp3).

All remaining tables are associated with the DataID table. These tables expand on the data objects present in content points by expanding functionality. These extra tables exist mainly because not all data objects need the extra fields. T02_Source deals with sources for each piece of data; T06_PictureTransistionDetails deals with translations of documents; and T05_DataTransitionDetails deals with screen transition data points. The fields in T02_Source consist of the associated unique ID (SourceID), the data link (DataID), and associated fields that identify the pertinent source. The source fields include the SourceTitle, SourceWeb, SourceAuthor, SourceYear, SourcePublisher, SourcePlace, SourcePages, and SourceOther.

T05_DataTransitionDetails contains the same fields as the T01_SiteDetails table except for the SiteID is replaced with the DataID. This table assists with screen transitions; but it does not provide information for the content loader. When a piece of content is supposed to transition to a new screen, such as loading a photo or video, this table contains the associated backdrop. For example, loading the Gressette Room requires the background image of the Gressette Room panorama (which a participant can swipe to pan right and left across the image). The T05_DataTranisitionDetails table holds that information.

The T06_PictureTransitionDetails table is the last table. Some images are digital scans of old documents like receipts, letters, ledgers, and disciplinary records. All these documents are handwritten and the script can be difficult to read. When a participant transitions to an image that has handwritten text, this table is queried, and if there are words to be translated, they are placed on the screen as pressable buttons. This table consists of a unique word ID, PTDID, and the associated DataID. Because these buttons are word element-specific, they also have x- and y-coordinates,

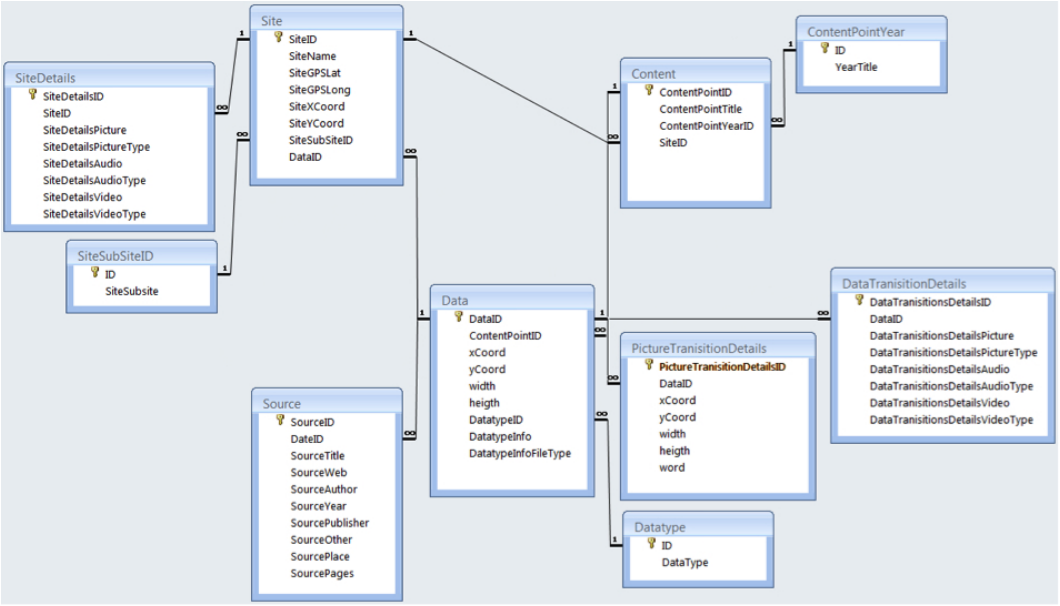

Figure 5.4 Content Relations

as well as information regarding the button's dimensions. The final field displayed is the word itself "translated" into legible typeface. The relationships for all the tables and how they interact are shown in Figure 5.4.

The class that connects all the data tables is the `ContentStarter` class. `ContentStarter` controls the network connections and pings the server if content is available and/or needs to be downloaded. It handles the JSON input from the server and parses all data points into the respective tables. If there is no data, `ContentStarter` polls the web server, downloads data, and then stores it on the device. If there is content on the device, `ContentStarter` loads that after checking if it's up-to-date. It contains all data structures, commonly as a collection of `NSMutableDictionary` objects, and also handles any requests for data. `ContentStarter` controls all data requests. If a screen requires content, `ContentStarter` will search the requested data table, and if it exists, construct an array of views. This array of views is then passed back to the caller. The content starter takes a site location ID and searches the associated content points (`T03_ContentPoints`) for all relevant data

elements. It then takes the new array of content points and finds all associated data objects (`T07_Data`). `ContentStarter` then builds an array of views, with each view being a separate content point with each data object placed on the view. This array is then passed back to the caller. No view controllers have direct access to stored content.

5.5 *Ghosts of the Horseshoe* Data Types

Content comes in different formats. For *Ghosts*, content includes text, still images, moving images, and 3D models. Text can be modified according font size, color and type. All of these properties are stored in application (in app). The font used for all titles is Apple Chancery while the general font throughout the application is Georgia. Images come in different file types. In app, PNG is commonly used due to the inclusion of an alpha channel, and is preferred over JPG for this reason. (JPG is used, but sparingly.) The alpha channel allows for transparency in an image and the ability to overlap many different components of a complex image. The ability to separate parts of an image into separate "layers" mimics what is happening inside *Ghosts* with the many different views. It makes modifying images easier. For video, MP4 is mainly used given its ubiquity and portability. While other video file types exist, MP4 is the most popular due to its compression algorithm. Music is stored as an MP3 for similar reason as video is stored as MP4. It is one of the most popular file types and is widely supported. All content and its differing formats are stored in two locations. Currently all images, videos, and audio are stored in app due to the large file sizes. All text related content is stored on the server. Eventually, all data will be stored server side and passed into the application.

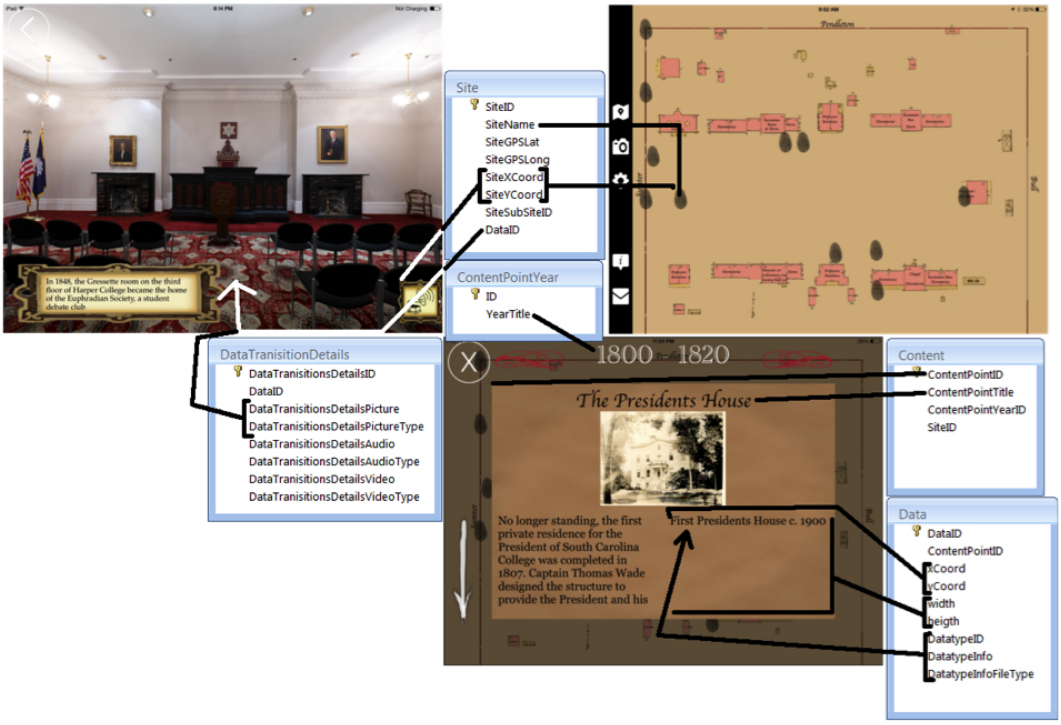

Figure 5.5 Content Field Screen Location

Chapter 6

Usability Assessments

To analyze *Ghosts of the Horseshoe* and determine if it meets the goals of a CI, concerns specific to digital humanities, human computer interaction (HCI), and computing must be addressed. The application must provide adequate framing for content to meet the DH requirement. The application must meet certain design components discussed in Chapter 3 to show it meets usability requirements. Finally, the application must be coded properly using techniques to help establish rules that illude to both the framing and heuristics.

For *Ghosts of the Horseshoe* to provide adequate framing for the content, the way in which all text, audio, video, and gamic rules must help establish the historic Horseshoe narrative. There is little recorded history that survives to this day about the slaves that built and worked on the University of South Carolina campus. Receipts, discipline records, and notes are all the textual information that remains. There are no images or videos of the slaves. It is important that *Ghosts* frames this apparent lack of knowledge while also expressing what knowledge we do know.

For *Ghosts of the Horseshoe* to meet the HCI requirement, it must pass the five specific heuristics discussed in Chapter 3. *Ghosts* will operationalize the concepts of "usability" and "effectiveness" by following suggestions proposed by Hornbaek [Hornbaek 2006]. Usability is defined as the "[e]xtent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction" while the concept of effectiveness is defined as "[a]ccuracy and completeness with which users achieve specified goals" [Hornbaek 2006]. To conclude that software

is usable, it has to meet its intended goal in a manner that shows it was effective in performing the task it was created for, that it is efficient in how it completes the task and that users are satisfied enough with the software package to consider continuing to use it. The most important measure of usability to consider is effectiveness, because efficiency can be studied by analyzing the algorithms, and because an exit questionnaire can be used to determine satisfaction [Hornbaek 2006].

Of the different methodologies to measure effectiveness, we will be using binary task completion and recall. Measuring binary task completion is done by recording the percentage of tasks that users successfully complete. With the *Ghosts* application, the use of the interface and features without any guidance from an external aid or guide was analyzed. Tasks one would consider would include (but not be limited to): navigating to the menu, returning to the map screen from the menu, finding content on the horseshoe, scrolling the screen to new content points, figuring out how to scale images, discovering AR and other sub screen features, getting AR to read the QR codes, navigating back to the content screen, and returning back to the map. If a participant can perform these tasks without aid, we would score a positive mark. If the participant fails to perform tasks during the duration of the test or requires help, we would score a negative mark. If more than 95the binary task completion portion of the study.

Use of recall would also be important given that participants are required to repeat tasks. If a participant loads content, realizes how to scroll, but then forgets how to scroll the next time around, we would conclude that the application needs to be changed. Bayles measured the recall abilities of users who viewed banner ads in a 2002 study [Bayles 2002]. Individuals were tasked with differentiating between banner ads and distracter ads and recalling webpage layouts given the ads. For *Ghosts*, using a questionnaire that targets certain aspects of the application, such as asking how easy it was to navigate or how often did they get confused on their current position in the

application can help determine if individuals remembered how to use the features and is valuable in determining the applications effectiveness. This method was chosen due to time constraints. Given how hard it is to assess questionnaires, a more quantitative assessment of recall would have been to give participants the task several weeks later and assess their recall performance or to ask them to view a screenshot of the CI and ask them to recall what each button does.

Another measurement to test effectiveness is the time needed to learn and complete tasks. Participants who use the application will have all their information recorded, from every content point read, every step taken and every button pressed. The time between tasks can be used to assess how fast individuals are picking up the layout, the position of information, and the interface choices.

Usability and effectiveness will be measured using binary task completion, recall, and quality of outcome. Once the application meets defined standards, which will be chosen based on future literature review, the application will be deemed usable and effective. If the design choices fail to meet the standards described, modifications to the program will be performed to improve each measurement.

Another measure that can be tested has to do with improvement in design. *Ghosts* was built with the final user in mind. All design choices were based on improving the experience and presenting the information in the simplest possible manner. This included several design assessments from fellow colleagues over the course of development. The look and feel of the application and the way the application flowed were examined by fellow developers and in consultation with Dr. Jenay Beer. Semi-structured usability assessment were administered throughout the application development cycle. Approximately N=25 participated in the usability assessments during each of the three major application presemtations. On smaller usibility tests, usually with smaller classes or family demonstartions, approximately N = 5 participated in each demonstation. The participants were recruited from /em Ghosts demon-

startions at the end of Gaming the Humanities, Critical Interactives, and Critical Interactives: The Wall. University 101 courses, MART 701 students and family also participated in usibility testing. The methods used to evaluate the Ghosts of the Horseshoe include questionnaires, user tests, and heuristic evaluations. Questionnaires were administered after each demonstration via Survey Monkey. Example questions include <list>. User tests were observational by nature. Participants were asked to use the CI in a group setting, and developers observed their interactions. Observational metrics included recall and ease of use. Finally, heuristic evaluations were used at all stages of the design process. The application was given to developers at the beginning of each course, being Critical Interactives and Critical Interactives: The Wall, and their usibility results were used to determine critical flaws in design to be fixed as the semester progressed. The observational results would then be the foundation to improving the design for the next prototype. When participants were able to navigate the CI without getting confused, we concluded that we had a good design for the CI. Of the design elements examined, it was the menu based system that best allowed participants to adjust settings in the application, the way the material was represented and the accessibility of content and if participants were able to read the content easily under differing weather conditions. The usability data was not collected in a controlled setting, and thus not formally statistically analyzed. Instead a rapid prototyping approach was used were usability data was quickly assessed by the developer team and immediately incorporated into the prototype design in an iterative fashion.

For *Ghosts of the Horseshoe* to be computationally successful, all code sould be as computationally efficient as possible, easy to modify and read, and sould follow common object-oriented programming (OOP) paradigms. Code is considered efficient if the algorithm used does not break on execution, handles special cases that could potentially break the application logic, and executes in a reasonable amount

of time (i.e., cycles through data as quickly as possible). If the code is documented correctly: a good choice of function and variable names, and accurate data types for variables, then *Ghosts* will be easy to read and modify. When it comes to OOP, each constructed class should include the following: proper measures for hiding data from other classes, correct getters and setters for all variables that each class requires, and proper inheritance (i.e., each class inherits what it needs and nothing extra). Other considerations relate to connectivity with outside systems. Bandwidth and memory constraints for a mobile device while communicating with the backend data server were also considered. If *Ghosts* is able to pull content from a server without experiencing issues, then pulling the content was considered a success.

Chapter 7

Discussion

7.1 Conclusions on Digital Humanities

Ghosts of the Horseshoe frames its content not only using the framing constructs of text, image and film, but also with its code and the system the code creates. As discussed in the proceeding chapters, programs are more than just code plus data [Balsamo 2011]. When an individual interacts with a program such as *Ghosts*, there is more to the experience than just the end product produced by the code. *Ghosts of the Horseshoe* frames content with the code by the choices made and the algorithms that are used. Algorithms are used to produce the images, load content, track the participants' location on a map and record their progress through the application. The speed of which these are performed, the amount of space in which the algorithms store information and the options available are all framing devices to be considered with the code. The aspect ratio of the images, the size of the text to be read, the quality of the film, and the layout created by the code can affect an individual's interpretation of the information being presented. By framing an image small, with large text, emphasis on the text would be more relevant. By making it difficult to find and watch film placed within the code, there will be a diminished impact to any content within that film and any framing of that film. The framing by the code not only affects what can be seen, but what cannot. The coding process requires many decisions to produce a functional program; these decisions will emphasize more important features and regulate minor features and thus could cause greater emphasis

on certain framing devices and, in the extreme, refute others entirely. If chosen, certain types of content can be ignored entirely by just not coding in the ability to use such content. Because of this, much care was taken when developing *Ghosts of the Horseshoe* to make sure the framing of the content is faithful to the history and that the content can be expressed in a fashion that produces the desired message.

Even the design of the data structures has an effect on how content is framed and experienced. During one of the early builds of *Ghosts of the Horseshoe*, data was stored in a hierarchical design. This meant that only certain pieces of content could be accessed through a specific portal. This allowed for easy retrieving of data, but it was a very rigid structure that only allowed a specific type of interaction with content. When the design specifications changed to a more data centric model, having nested data clashed with the new methodologies in play. *Ghosts of the Horseshoe* now frames its content by means of multiple tables. Content is loaded from the server into one of eight `NSMutableDictionary` structures. These dictionaries contain the tables pulled from the server with keys being associated with other reference tables. The site dictionary uses the `SiteID` as well as the `ContentPoint` table. This is done so data retrieval is fast. The goal of content points is to coalesce data information and be accessed from specific sites, making it intuitive to use `SiteID` as the key of the content point table. The same logic applies for the data table and the `contentPointID`. The `NSDictionary` structures provide the basis for how the content is displayed. The data table contains placement x- and y- values for each piece of content. This gives the content administrator the ability to create any look and feel for the content.

Ghosts of the Horseshoe is about the history of the historic Horseshoe at the University of South Carolina in Columbia, primarily the history of slavery and of the enslaved persons that built the Horseshoe. But this is not the only history being presented to a participant. The development process of an application also produces a history about the design choices being made, the design decisions lost, the way in

which content and its storage changes and the way in which the application develops into a finished product. This history, in essence the rough drafts of code and the application, is often overlooked because many do not consider the amount of work and the number of iterations required to produce a solid application. Features that get implemented in an early version, for example the ability to select different time periods before interacting with the map, get removed in later versions due to complications in framing the content's message. Similarly, features get emphasized more or even added in later versions, such as the ability to translate text of old documents, to improve the quality of the application. All this affects the final framing of the application and the balance between what is most important and what is possible.

The design choices for *Ghosts* attempted to emphasize the lack of knowledge about the enslaved persons who built the Horseshoe. This is present in not only the content itself, but also in its framing. The fingerprint has been chosen as the button icon because it is one of the few remaining physical artifacts of the enslaved persons that is not a receipt or discipline record. The Sanborn map has been chosen because it is most likely the best map of the antebellum campus and buildings. The parchment used as the background for content points has been chosen to suggest the historical nature of the material. The compass rose icon to represent the participant has been chosen because it is a standard image found on maps and it thus seemed an appropriate icon for a participant engaged in an exploration of the Horseshoe.

Content is presented as text, images, moving images, animations, 3D models, and audio. In some instances, these contents overlay the real tiem view on screen (augmented reality). Each has its strengths and weaknesses. Text is used to express many concepts and to get the participant to think about the Horseshoe. Images have the power of immediacy and were used when available. These include display receipts, portraits, photographs of old structures, and other documents. These media artifacts help show some surviving documents of the enslaved persons who built and worked on

the historic campus. Moving images have even greater power and are used to convey abstract concepts, such as how the wall was built or the time it took to complete a given task. AR and pannable images allow for more interaction with content, going further in depth about a specific location. AR and the 3D model viewers allow for viewing 3D models of bricks and buildings in the context of the present day site. Finally, audio was used to build atmosphere and give impact to events. Audio clips included environmental sounds, decisions made from the debate society and slave interpretations. All design decisions were made to frame the content in a way to create a compelling experience for a participant.

7.2 Conclusions on Human Computer Interactions

The target demographic for the *Ghosts of the Horseshoe* Critical Iteractive (CI) is University 101 (UNIV101) students and the rest of the university community, visitors to the campus (especially prospective students and their familes), and historians interested in the Horseshoe. With this in mind, presenting content was at the forefront of development. How the content looked, how it was presented, and how it was navigated was examined. In the first version of *Ghosts*, content was presented separately on individual screens, and a participant had to delve deep into a menu-driven system to reach content. This prototype was built for the iPhone. When it was tested by the developers, it was decided that the screen real estate was too small for the goal of the application.

The next iteration of the application used the larger screen of the iPad and became more menu-driven. The developers decided that the ability to select the different types of content was important. Given the change and growth of the Horseshoe over time, content was broken into 20 year intervals. This version also treated major content concepts such as augmented reality as the focal point. User testing of this version found the menu systems simple to use, but the content difficult to get to.

After another test with participants, it was decided that the content was too difficult to find; traversing up and down a hierarchy of screens to see new content was too cumbersome.

The third iteration focused on fixing content presentation, with content being brought to the forefront. The menu system was relegated to the participant icon and the content was given precedence, with AR and others sharing the same scope as text or small images. In user testing, this version performed well, with participants being able to navigate to all types of content with ease. Participants were also able to change time periods without difficulty. What was learned from this testing phase was that while content was present, it was difficult to tell if there was more content present at a site. Some site locations had time periods with one piece of content, while others had multiple pieces of content. Participants commonly failed to recognize that there was more content. Another design flaw discovered early in user testing was the menu screen. Participants would press the screen, intentionally or by accident, and usually end up at the menu screen. They would then become confused as to why content failed to load and why the participant icon stopped moving. Given that this would usually be the first time they entered the screen, many did not know how to navigate back to the map. Once they figured out the menu system, some were able to navigate to the options screen and back to the map, but this was seen as an inconvenience.

The final developers' test took much of what was learned from the user tests and implemented fixes. The fact that "fingerprints" were buttons confused many participants. To address this, a participant was presented with a fingerprint immediately after the splash screen, which upon pressing, tranisitions to the map screen. This worked to teach a participant that fingerprints are buttons. Once the participant enters the map screen, he usually tries pressing the fingerprint button again. This does not work given that site content is GPS specific and requires the participant to be within a distance to expereince the content. If the participant gets close to a

fingerprint on the map, then content appears. The participant also has the option to press the fignerprint button that he is standing near to load content. The menu inside the particiapnt compass rose icon was removed, making the participant icon function only as a GPS locator. The menu has been streamlined and placed on the left-hand side of the map screen. It contains option pop-ups that offers participants the opportunity to change aspects of their interaction. A participant can change the color of his trail or increase the text size to make content easier to read. Participants had difficulty reading the map in previous versions; they constantly turned the iPad around in an attempt to get their bearings. A Google map, legend, and the paths on the Horseshoe were added as toggles to allow a participant to adjust the map to their specifications. This made it easier for a participant to discover his location and further explore the Horseshoe. Within content site locations, content points were placed closer together. This made it easier to determine if there was additional content to experience. Icons for switching the time period and closing buttons were made larger and more noticeable so a participant could navigate without difficulty. From observations of user test groups, all changes helped improve participant response time. All these changes appeared to improve usability.

7.3 Conclusions on Computer Technologies

Ghosts was created with advanced computer technologies in mind, and all algorithms were carefully considered so as to improve speed. There was extensive use of the `NSMutableDictionary` data structure to achieve constant time look-up. Originally, content was stored in two arrays, one based on site locations (GPS coordinates) and the other based on content panels. This structure allowed content to be accessed anywhere within the program with a simple table lookup function but took $O(n)$ time to find content. In the first data structure used, content was hierarchical. That meant that to get to content, one had to go through three layers of nested meta-data

(location, time period and panel number), ending with more than $O(n)$ time if a piece of content needed to be searched. This did not allow for complicated viewing of data, such as showing related topics on a building from multiple time periods. Other considerations for speed were to store all changeable settings as constants inside a `GlobalState` file. This allowed for constant lookup and changing of application settings, but makes most of the important variables global and exposed.

The framing of the content as it is shown to the participant is another stage of framing. The participant is greeted with a splash screen followed by a fingerprint icon. Following the fingerprint is a map. The participant is tasked with walking the Horseshoe from site to site to trigger content with which he might interact. Content is site specific, meaning that an individual is not overwhelmed with all content at once. Content pages are also designed to express as much information as possible on the screen. All content images have a feature which allows a participant to press said image and tranisition to a new screen so it is possible to view special types of content or zoomed in images. Each screen transition is handled programmatically and uses the notification center to determine when a transition needs to occur. This is not the best way to do it, but given that all content comes from a content class, it was one of the only ways to perform the task. Apple has special listeners called "delegates". These allow for two classes to openly discuss information before making a decision on the correct course of action. Delegates also allow for more specific control over memory, because a delegate focuses on the information it needs instead of parsing through all the information available. We used delegates because they function more efficiently than the notification center and because not all classes with a listener need to be pinged when the screen has to be transitioned. Space is allocated and deallocated for each view as needed, with nothing kept in memory unless required.

For network connectivity, *Ghosts* is using the `AFNetworking` framework to pull content and store it on the device. AFHTTP controls all networking protocols, mak-

ing it simple to connect, pull information, and store it. All data is stored in plists for easy access. If a packet of data has already been downloaded, then the application pulls the content from the device. Any images taken through the citizen archaeology features will eventaully be passed back to the server. The data transfer only happens when a WiFi connection is established. Camera images when taken are stored on the device and sent to the on site server at the first established connection. Feedback regarding the application and its content is sent through email to the administrator's email address.

The CI was created programmatically to make future updates easier to read and many algorithms were chosen for speed efficiency. Network connectivity was kept to a minimum given that the application is meant for an iPad device and a network connection cannot be assumed. All code was commented thoroughly with the belief that others would not only comment and edit the code, but study it in a non-programming manner.

7.4 Conclusions on the Mobilization of a CI

Ghosts has a complex development history. It began as a small iPhone application to tell short stories and present poetry about the historic Horseshoe and it grew to an iPad CI with an emphasis on framing content and using gamic qualities to illuminate the history. With critical interactives, we seek to use ludic methods to engage individuals, impart knowledge, build awareness, question past observations, and challenge preconceived notions. The rules of the system create the space for exploration. A participant, an interactant, is asked to load the *Ghosts* application and then to engage with the content provided.

Ghosts also serves as an example of code as speech. All programs have three states of existence: as code, as speech, and as executables [Cox and McLean 2013]. In this regard, execution was successful because the project works. As code; *Ghosts*

was written to be self documenting as well as to have comments spread throughout to help non-programmers read. It resembles a users manuel, complex procedure and a unique decision making process. As speech; the code is procedural rhetoric that tells at least some of the story of the Horseshoe. Variables are named specifically to represent the content and options available. If a participant is standing still on the Horseshoe, the code is specifically within the `S02_Map` class and shows the potential content options available. Displayable content is set to nil, similar to the knowledge the participant currently knows about the location. The code for the GPS waits for a signal from the device, but once a specific GPS location is found, content is queried and the participant is asked to engage with the content. The code expresses this idea not only in the way it was written and operates, but in the way that extends beyond the Objective-C compiler[1].

7.5 Future Work

This instantiation of *Ghosts* may be finished, but there is a need to add to and update the CI. Discovering new ways to add additional content and would provide greater variety of content information; given that much of the current content ends up as scrolling text. Adding in more AR content points and even newer types of content (such as a camera-image fade system to show old vertop present) would help further expand the CI's immersive experience. Another addition would be to expand the physical area dealt with by the app. *Ghosts* focuses currently only on the Horseshoe, but the University of South Carolina has a larger history that expands, for example, into the neighborhoods of what was Ward One. This would require changes and additions to accommodate newer locations beyond the Horseshoe, but it is something to consider for future updates. Other technical considerations are to move the images

[1] A compiler takes high-level language code, such as Objective-C and turns english nominclature into code which the computer can read.

permanently onto the server and to pass the images to the device with all database content.

A further issue comes from GPS and AR. We have not found the GPS reliable under all environmental constraints. The GPS has failed to work on several occasions and on other occasions has repeatedly given inconsistent results. This seems inherent in the technology, however. Instead of relying on GPS, the use of iBeacons to transmit a small signal that is listened for by the application would probably help improve loading content. For AR, the package used in the current version, Qualcomm's Vuforia™, has become outdated. The newer version of Vuforia™needs to be incorporated to help improve functionality. This does not fix issues with reading QR targets. More images of each target under different weather conditions needs to be incorporated into the AR database to help recognize said targets.

Other future work includes building out a non-iPad version of the application. Most of the code written for the iPad version can be tweaked to fit on the iPhone. Some design elements may need to change given that the iPhone has smaller screen real estate than the iPad. An Android version would also help expand the target audience. On an iPhone version is finalized, making a comparable product on Android would be straight forward.

More tests on usability and effectiveness need to be performed. While preliminary tests have been performed and many developer tests have been performed over the four-version, three-year development span, more tests with users and updates based on user feedback are important. Most design kinks have been worked out, but it may prove useful to mimick how websites and social media use images, such as clicking a second time to shrink the image. Finally, adding in more cues to tell participants how to use some of the lesser known features (such as being able to press AR objects to load more content) would improve functionality.

7.6 Conclusions

Ghosts of the Horseshoe is one of the first attempts to mobilize a critical interactive. Using the digital humanities as a springboard to frame and present content, human computer interaction to construct the CI in a way to make it easy to use to help present content and programming to create the CI, *Ghosts of the Horseshoe* is truly a multidisciplinary project. With the three-year development cycle and the many teams that helped produce this project, it was possible to create the first true version of a CI. User testing and Developer testing helped improve the entire experience, addressing many design flaws that persisted through the various versions of the application. *Ghosts* ended up being a succesful mobilization of a Critical Interactive.

Bibliography

Abt, Clark C. (2002). *Serious Games*. Lanham, MD: University Press of America. ISBN: 978-0819161482.

Arvers, Isabelle (Nov. 2009). "Serious Games". In: *digitalart1*, pp. 24 –25. URL: `www.digitalarti.com`.

Association for Computing Machinery (1992). *ACM SIGCHI Curricula for Human-Computer Interaction*. Baltimore, MD, USA. ISBN: 0-89791-474-0.

Balsamo, Anne (2011). *Designing Culture*. Durham, NC: Duke University Press. ISBN: 978-0-8223-4433-9.

Bayles, M. (2002). "Designing Online Banner Advertisements: Should We Animate?" In: *Proceedings, SIGCHI 2002 ACM Conference on Human Factors in Computer Systems*. Association for Computing Machinery. New York: ACM Press, pp. 363 –366.

Bogost, Ian (2007). *Procedural Rhetoric*. Cambridge, MA: The MIT Press, pp. 1 –64. ISBN: 13:978-0-262-02614-7.

Brown, Stuart and Christopher Vaughan (2010). *Play: How it Shapes the Brain, Opens the Imagination, and Invigorates the Soul*. New York: Avery Trade. ISBN: 978-1583333785.

Buell, Duncan A. and Heidi Rae Cooley (Dec. 2012). "Critical Interactives: Improving Public Understanding of Institutional Policy". In: *Bulletin of Science, Technology and Society* 32 (6), pp. 489 –496. URL: `http://bst.sagepub.com/content/32/6/489`.

Butler, Judith (2009). "Torture and the Ethics of Photography: Thinking with Sontag". In: *Frames of War: When is Life Grievable?* Brooklyn, NY: Verso, pp. 63 –100.

Caillois, Roger (1961). *Man, Play and Games*. Translator: Meyer Barash. Chicago, IL: University of Illinois Press. ISBN: 978-0-252-07033-4.

Cohen, D.S. *Cathode-Ray Tube Amusement Device - The First Electronic Game*. Accessed 05/22/2014. URL: http://classicgames.about.com/od/classicvideogames101/p/CathodeDevice.htm.

Cooley, Heidi Rae (2014). *Finding Augusta: Habits of Mobility and Governance in the Digital Era*. Sudbury, MA: Dartmouth College Press. ISBN: 978-1611685220.

Cox, Geoff and Alex McLean (2013). *Speaking Code*. Cambridge, MA: The MIT Press.

Csikszentmihalyi, Mihaly (July 2008). *Flow: The Psychology of Optimal Experience*. New York: Harper Perennial Modern Classics. ISBN: 978-0061339202.

Dale, Nell and John Lewis (2011). 4th. Sudbury, MA: Jones and Bartlett Publishers. ISBN: 978-0-7637-7646-6.

Decker, Cecil (2010). *[THRESHOLD]: Understanding Noise through Play*. MA thesis. University of South Carolina.

Derrida, Jacques (1987). *The Truth in Painting*. Translators: Geoff Bennington and Ian McLeod. Chicago, IL: The University of Chicago Press.

Dourish, Pail and Genevieve Bell (2011). *Diving A Digital Future*. Cambridge, MA: The MIT Press. ISBN: 978-0-262-01555-4.

Flanagan, Mary (2009). *Critical Play: Radical Game Design*. Cambridge, MA: The MIT Press, pp. 1–15.

Flanders, Julia (2012). "Days of DH: Defining the Digital Humanities". In: *Debates in the Digital Humanities*. Ed. by Matthew K. Gold. Minneapolis, MN: University of Minnesota Press, p. 69.

Gossett, Kathie (2012). "Days of DH: Defining the Digital Humanities". In: *Debates in the Digital Humanities*. Ed. by Matthew K. Gold. Minneapolis, MN: University of Minnesota Press, p. 67.

Grusin, Richard (2010). "Affect, Mediality, and Abu Ghraib". In: *Premediation: Affect and Mediality after 9/11*. London, UK: Palgrave Macmillan, pp. 62–89.

Hodgson, John (2012). *Desperate Fishwives: A Study in Applied Game Design*. MS thesis. University of South Carolina.

Hornbaek, Kasper (2006). "Current Practice in Measuring Usability: Challenges to Usability Studies and Research". In: *Int. J. Human-Computer Studies* 64, pp. 79–102.

Huizinga, Johan (1971). *Homo Ludens*. Boston: Beacon Press. ISBN: 0-8070-4681-7.

Isbell, Charles, et al. (2009). "(Re)Defining computing Curricula by (Re)Defining Computing". In: *SIGCSE Bulletin* 41.4, p. 195.

Kotonya, Gerald and Ian Sommerville (Sept. 1998). *Requirements Engineering: Processes and Techniques*. Hoboken, NJ, USA: Wiley. ISBN: 978-0471972082.

Levesque, Hector (2012). *Thinking as Computation*. Cambridge, MA: MIT Press. ISBN: 978-0-262016995.

Magnuson, Jordan (2011a). *Freedom Bridge*. URL: `http://www.necessarygames.com/my-games/freedom-bridge`.

— (May 2011b). *Loneliness*. URL: `http://www.necessarygames.com/my-games/loneliness/flash`.

Minsky, Marvin (June 1974). "A Framework for Representing Knowledge". In: *MIT AI Laboratory Memo 306*. URL: `http://web.media.mit.edu/~minsky/papers/Frames/frames.html`.

Molleindustria (2006). *McDonald's the Videogame*. Accessed 11/03/2013. URL: `http://www.mcvideogame.com/game-eng.html`.

Mooers, Calvin N. (1975). "Computer Software Copyright". In: *ACM Computing Surveys*, pp. 45–72.

Morris, Errol (2008). *Standard Operating Procedure (film)*. Accessed 11/03/2013. United States.

Peirce, Charles Sanders (1991). *Peirce on Signs: Writings on Semiotic by Charles Sanders Peirce*. Chapel Hill, NC: The University of North Carolina Press.

Priego, Ernesto (2012). "Days of DH: Defining the Digital Humanities". In: *Debates in the Digital Humanities*. Ed. by Matthew K. Gold. Minneapolis, MN: University of Minnesota Press, p. 69.

Rockwell, Geoffrey (2012). "Days of DH: Defining the Digital Humanities". In: *Debates in the Digital Humanities*. Ed. by Matthew K. Gold. Minneapolis, MN: University of Minnesota Press, p. 69.

Rosser, J. C. et al. (Feb. 2007). "The Impact of Video Games on Training Surgeons in the 21st Century". In: *JAMA* 142 (2), pp. 181–186.

Rulz, Susana (2006). *Darfur is Dying*. Accessed 11/04/2013. URL: http://www.darfurisdying.com/.

Salen, Katie and Eric Zimmerman (2004). "Unit 4: Culture". In: *Rules of Play: Game Design Fundamentals*. Cambridge, MA: The MIT Press, pp. 502–588. ISBN: 13:978-0-262-24045-1.

Singh, Amit (Dec. 2003). *Mac OS X Internals*. Accessed 05/12/2014. URL: http://osxbook.com/book/bonus/ancient/whatismacosx/history.html.

Steinberg, Scott (Aug. 2010). *Who says video games aren't art?* Accessed 05/22/2014. URL: http://www.cnn.com/2010/TECH/gaming.gadgets/08/31/video.games.art.steinberg/.

Tompkins, Jessica E. (2014). *Playing at History: Resurrection Man and Historiographic Game Design*. MA thesis. University of South Carolina.

Tversky, Amos and Daniel Kahneman (Jan. 1981). "The framing of Decidiond and the Psychology of Choice". In: *Science* 211 (4481), pp. 453–458. URL: http://links.jstor.org/sici?sici=0036-8075%2819810130%293%3A211%3A4481%3C453%3ATFODAT%3E2.0.CO%3B2-3.

UCLA Center for Digital Humanities (Jan. 2014). *What is DH?* URL: http://www.cdh.ucla.edu/about/what-is.html.

United Nations World Food Programme (2005). *Food Force*. No longer available.

Weyeneth, R. et al. (2011). *Slavery at South Carolina College, 1801-1865: The Foundations of the University of South Carolina*. Accessed 05/22/2014. URL: http://library.sc.edu/digital/slaveryscc/index.html.

Wirth, Niklaus (1976). *Algorithms + Data Structures = Programs*. Prentice Hall.

Zyda, Michael (Sept. 2005). "From Visual Simulation to Virtual Reality to Games". In: *Computer Archive* 38 (9), pp. 25–32.

Appendix A

List of Major Classes, Functions, and Objects in the Application

S01_SplashViewController - (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil - (void) didReceiveMemoryWarning - (void) handleFingerButton - (void) loadButton - (void) loadingView - (void) moviePlayBackDidFinish:(NSNotification *)aNotification - (void) removeBlank - (void) removeSpinner - (void) transisitionView - (void) viewDidLoad
S01_SplashScreen - (id)initWithFrame:(CGRect)frame - (void) iPadCode - (void) iPhoneCode

```
S02_MapViewController
 - (id)initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil
 - (void) changeColor:(UIButton *)button
 - (void) dealloc
 - (void) didReceiveMemoryWarning
 - (void) flip:(UISwitch *)switchFlipped
 - (void) handleCloseButton:(id)sender
 - (void) handleContentButton:(id)sender
 - (void) handleExit
 - (void) handleReturnButton:(id)sender
 - (void) imagePickerController:
                (UIImagePickerController*) reader
                didFinishPickingMediaWithInfo:
                        (NSDictionary*) info
 - (void) loadContent:(id)item
 - (void) LoadGPS
 - (void) loadMenuItem:(UIButton *) butt
 - (void) loadingView
 - (void) locationManager:(CLLocationManager *)manager
                didUpdateLocations:(NSArray *)locations
 - (void) locationManagerDidPauseLocationUpdates:
                (CLLocationManager *)manager
 - (void) locationManagerDidResumeLocationUpdates:
               (CLLocationManager *)manager
```

S02_MapViewController cont.

```
- (void) locationManager:(CLLocationManager *)manager
                 didFailWithError:(NSError *)error
- (void) mailComposeController:
                 (MFMailComposeViewController*)controller
                 didFinishWithResult:(MFMailComposeResult)result
                 error:(NSError*)error
- (void) moveOptions:(id)sender
- (void) notificationCenterLoader
- (void) receiveNotification:(NSNotification *) notification
- (void) removeSpinner
- (void) scrollViewDidEndZooming:(UIScrollView *)scrollView
                 withView:(UIView *)view
                 atScale:(float)scale
- (void) scrollViewDidZoom:(UIScrollView *)scrollView
- (void) sliderControl:(UISlider *)slider
- (void) tranCamView
- (void) viewDidLoad
- (UIView *) viewForZoomingInScrollView:
                 (UIScrollView *)scrollView
```

S02_1_MapScreenView

```
- (id) initWithFrame:(CGRect)frame
- (void) alphaValueName:(id)contentName
                 distanceValue:(float)distance
- (void) dealloc
```

S02_1_MapScreenView cont.

```
- (void) drawCompass
- (void) drawPoints
- (void) drawRect:(CGRect)rect
- (void) handleCloseButton:(id)sender
- (void) handleContentButton:(id)sender
- (NSArray * ) moveParticipantIcon:(CLLocation *)location
- (void) notificationCenterLoad
- (void) receiveNotification:(NSNotification *) notification
- (void) touchesBegan:(NSSet *)touches
              withEvent:(UIEvent *)event
- (void) zoomIconChange:(float)newScale
```

ContentScreenViewLoad

```
- (id) initWithFrame:  (CGRect)frame
- (id) initWithSiteContent:  (id)content
- (id) initWithContentPoint:(id)content
              forButton:(id)button
- (void) buildView:(NSMutableArray *)contentTemp
- (void) buttons
- (void) handleViewsSwipe
- (void) handleViewsSwipe2
- (void) LoadContent:(id)content
- (void) LoadContentPointContent:(id)content
              forButton:(id)button
```

```
CV_ARViewController
 - (id) initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil
 - (id) initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil
                fromContent:(NSString *)fromContent
 - (void) dealloc
 - (void) handleDoubleTap
 - (void) notificationCenterLoader
 - (void) receiveNotification:(NSNotification *) notification
 - (void) returnToPrev
 - (void) viewDidLoad
```

```
CV_PanoramicViewController
 - (id) initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil
 - (id) initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil
                fromContent:(NSString *)fromContent
 - (void) dealloc
 - (void) didReceiveMemoryWarning
 - (void) handleExit
 - (void) loadContent:(id)item
 - (void) notificationCenterLoader
 - (void) receiveNotification:(NSNotification *) notification
```

```
CV_PanoramicViewController Cont.
 - (void) returnToPrev
 - (void) viewDidLoad
```

```
CV_PictureViewController
 - (id) initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil
 - (id) initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil
                imageToLoad:(NSDictionary *) Anote
 - (void) didReceiveMemoryWarning
 - (void) picLoad
 - (void) returnToPrev
 - (void) scrollViewDidEndZooming:(UIScrollView *)scrollView
                withView:(UIView *)view atScale:(float)scale
 - (void) scrollViewDidZoom:(UIScrollView *)scrollView
 - (void) viewDidLoad
 - (UIView *) viewForZoomingInScrollView:
              (UIScrollView *)scrollView
```

```
CV_VideoViewController
 - (id)initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil
 - (id)initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil
                fromContent:(NSString *)fromContent
```

CV_VideoViewController Cont. - (void)didReceiveMemoryWarning - (void)returnToPrev - (void)viewDidLoad
CA_ModifyImage //Not implemented yet CV_3DModelViewController - (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil - (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil fromContent:(NSString *)fromContent - (void)didReceiveMemoryWarning - (void)returnToPrev - (void)viewDidLoad
CA_ReviewAndSubmit //Not implemented yet
GHButton - (id)initWithFrame:(CGRect)frame - (void) createButton - (void) PlayClick
GHTextView - (id)initWithFrame:(CGRect)frame - (void) createTextField

GHLabel - (id) initWithFrame:(CGRect)frame withText:(NSString *)text - (void) establish:(NSString *)text
AFNetworking Read AFNetworking for more information.
Qualcomm's Vuforia™ Read Qualcomm's Vuforia™for more information.
Cocos3D Read Cocos3D for more information.
T01_SiteDetials - (NSDictionary *) applToDictionary - (T01_SiteDetails *) init - (T01_SiteDetails *) init:(NSInteger) which
T02_Sources - (NSDictionary *) applToDictionary - (T02_Source *) init - (T02_Source *) init:(NSInteger) which
T03_Sites - (NSDictionary *) applToDictionary - (T03_Sites *) init - (T03_Sites *) init:(NSInteger) which
T04_ContentPoints - (NSDictionary *) applToDictionary - (T04_ContentPoints *) init - (T04_ContentPoints *) init:(NSInteger) which

T05_DataTranisitionDetails - (NSDictionary *) applToDictionary - (T05_DataTransitionDetails *) init - (T05_DataTransitionDetails *) init:(NSInteger) which
T06_PictureTranisitionDetails - (NSDictionary *) applToDictionary - (T06_PictureTranslateDetails *) init - (T06_PictureTranslateDetails *) init:(NSInteger) which
T07_Data - (NSDictionary *) applToDictionary - (T07_Data *) init - (T07_Data *) init:(NSInteger) which