5-2014

A Framework for Constructing Serious Games

Taylor Glen Yust
University of Arkansas, Fayetteville

A Framework for Constructing Serious Games

A Framework for Constructing Serious Games

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

by

Taylor Yust
University of Arkansas
Bachelor of Science in Computer Science, 2012

May 2014
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

Dr. Craig Thompson
Thesis Director

Dr. David Fredrick
Committee Member

Dr. John Gauch
Committee Member

Dr. Gordon Beavers
Committee Member

ABSTRACT

Communicating ideas and knowledge through serious games is a trend that is currently gaining in popularity. However, at present, there is a distinct lack of a game development methodology that takes a critical approach to transforming information into gameplay. This thesis presents a framework that can be followed to construct compelling serious games that are effective at transferring knowledge to the player. This is accomplished through an analysis of atomic knowledge items and their relationships to one another, followed by a meaningful and synergetic implementation of these ideas at a foundational mechanics level.

This thesis also describes the development of *Mythos Unbound*, a large-scale online serious game designed to teach students about classical Greek and Roman culture and literature. Offered as part of an online course at the University of Arkansas in the 2013 fall semester, deidentified user feedback suggests the game improved the learning process. The project was developed alongside the gamification framework and serves as the prime real-world application of its principles, and so demonstrates the potential of serious games as a means of meaningful communication.

ACKNOWLEDGEMENTS

I thank my mentor and advisor, Dr. Craig Thompson, for not only the incredible help and insight he provided during the writing of this thesis, but for his immense support and guidance through my years of university studies. I would not have written this thesis without his encouragement to pursue a graduate degree.

Without Dr. Fredrick, Keenan Cole, and all of my colleagues on the *Mythos Unbound* development team, I would not have been able to refine my programming and game development skills to their current level, nor would I have had the opportunity to work on such a meaningful project that I found so compelling. This has been the most significant endeavor in my career; I am so glad I got to be a part of it, and I thank you all for letting me contribute.

I also thank Dr. John Gauch and Dr. Beavers for being on my committee, reviewing my thesis, and being supportive of my somewhat untraditional interests.

Finally, I thank my family for always being there and supporting me, whether for a short phone call when I need advice or a trip home for the weekend. I don’t know where I’d be without their love and faith in my abilities. In particular, I do not think I would be writing this thesis if my mother was not so adamant that it would be beneficial for my career, and so I give her special thanks.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Problem

Video games have entertained for decades, but only recently are they beginning to be seen as transformative works with the potential to impact players in positive ways. Found in both academia and private industry alike, new supporters of the medium seek to create "serious games," or games that reach beyond entertainment to embrace fields like education and art. As such, there have been several attempts at converting knowledge and ideas into content that can be communicated through gameplay, often with varying degrees of success. So far, these works have not reported taking a formal approach to game development that is thoughtful in its design and execution. Thus, there has been little progress in developing critical game design methodologies or understanding the communication potential of game mechanics. As a consequence, these endeavors are not always effective; there is a need for a serious game framework that is reliable in its efficacy and can be applied to a wide variety of subject matter.

In October 2012, University of Arkansas professor Dr. Fredrick sought to develop a serious game with the purpose of teaching players about classical Greek and Roman culture and literature. The game, *Mythos Unbound*, [1] would be made available online as a semester course and offer participating students full class credit for successful play. With little in the way of academic literature to describe a process for creating such a game, it was up to the team to forge a new path and discover for themselves how to best accomplish such a feat. The author of this thesis was offered a lead programming position from the outset of the project, and it was from there that the team's trials and tribulations contributed to the development of a framework from which other serious games could be derived.

In the first half of 2013, University of Arkansas graduate student Chad Richards was also tackling the problem of obesity in children. The solution, a system called *Edufitment*, [2] was an online suite of serious games that were linked with an overarching website. The author of this thesis was involved in developing the *Edufitment* framework and also several of the plugin games, each covering a different knowledge item that the children were being taught. The experience of solving health issues through serious games was another large influence on the framework presented in this thesis.

1.2 Objective

The objective of this thesis is to develop a framework methodology that can be used to construct serious games that are effective in communicating their knowledge and ideas to players.

1.3 Approach

The formation of the gamification framework was very much a back-and-forth process as it was developed alongside *Mythos Unbound* itself. The team relied largely on its own wealth of gaming experience as well as the thoughts and writings of favored game developers. The team also looked towards the successes and failures of other serious games, adopting the ideas and methods that worked best. Team members were students with varied backgrounds including computer science, writing, art, music, architecture, and more. Each member contributed unique skills and talents to create a work that would have been impossible without such interdisciplinary collaboration.

After identifying the key knowledge items the game was to communicate to its players, our writers drafted an overarching narrative with an outline of specific levels. The team often broke into smaller sub-teams that each tackled individual levels. This afforded us greater mobility and kept levels more focused in vision and execution.

Much of the knowledge needed to create such a large-scale game was self-taught. While artists had a background in traditional forms of art, they needed to teach themselves computer art technologies like ZBrush and 3D-Coat to model realistic and compelling characters and environments. Programmers learned a variety of techniques in areas like artificial intelligence, web streaming, and workflow modeling. The team read books, watched online lectures, and even attended conferences like the Game Developers Conference and Unite to acquire this knowledge.

Even after *Mythos Unbound* was released in August 2013 and students were enrolled in the course, the team continued to develop and improve upon the game. By observing grades and setting up an online message board for player feedback, the team could identify problem areas and iterate so that the game was constantly improving.

The progress of the project directly influenced the framework. If a certain approach to a particular level was successful, it was analyzed and factored into the framework. Even mistakes provided lessons that were valuable in revising the framework to avoid similar pitfalls. The final framework presented in this thesis is the culmination of all of the experience learned from working on *Mythos Unbound*.

1.4 Organization of this Thesis

Chapter 1 introduces the problem, the objective, and how this thesis will approach them. Chapter 2 covers the background knowledge needed to understand the game design principles of the framework in addition to a look at related works. Chapter 3 outlines and describes the framework itself, detailing all of its tenets and core concepts. Chapter 4 provides an overview of *Mythos Unbound*, a serious game that was developed alongside the framework and implements its principle ideas. Chapter 5 gives an analysis of the framework by looking at student data and feedback from *Mythos Unbound*. Chapter 6 concludes with a final summary of what was accomplished and learned as well as how the framework could be expanded upon in the future.

2. BACKGROUND

2.1 Key Concepts

A basic grasp of game design is necessary to understand the framework. The field is vast and varied with many views and schools of thought being advocated by all manner of game design figures around the world. Entire books [3, 4, 5] have been written on the subject, and, because the field is still dominated by practitioners and not academics, still other voices have only been expressed through talks and interviews (such as *Super Mario* and *Donkey Kong* creator Shigeru Miyamoto's gameplay-focused "form follows function" philosophy, as seen in Section 3.2.6).

The point being emphasized is that there are many approaches to game design, and all are arguably subjective; there is no "answer" to quality game design. As such, this chapter will present a condensed explanation of game design based on the author's ideas, theories, and interpretations which have been founded on many years of experience, reading, and critical thought. In particular, it will focus on subject matter that pertains to the presented framework for gamification and the construction of serious games. A background in some ideas and technologies related to the framework and projects will also be provided.

2.1.1 Game Mechanics

A **game mechanic**, or simply **mechanic**, is an atomic unit of game design. Also known by the term "ludeme," [6] they are the "rules" of a game. At the lowest, most fundamental level, a game consists of a large number of mechanics, but they are often generalized for the sake of simplicity and communication.

For example, a “jump” mechanic is easily understood, but it is actually composed of many individual mechanics. A specific formula might describe a jump’s vertical motion, while another might dictate its lateral movement. Perhaps the apex of a jump is dependent upon how long a button is held. The jump might also have a minimum height before descending. And all of this is assuming a mechanical context involving ideas of “space” and “physics,” whereas in another context like Checkers a “jump” mechanic might take on a whole new meaning.

As illustrated, while it is perfectly acceptable to describe mechanics at a high level, it is critical to understand mechanics as atomic units that are often great in number but coalesce together into higher level ideas (see Section 2.1.2).

2.1.2 Gameplay and Game Design

A game could be described as being a collection of mechanics. We can describe these mechanics in terms of higher level ideas (like “jumping”), and large, related groups of mechanics can by labeled as a **system** (such as a “physics” system). The thoughtful construction and arrangement of these mechanics and systems is the art and science of **game design**.

A game’s many mechanics are like tight, interlocking gears in a complex, interwoven construct, but they have no meaning without input. When a user (or **player**) provides input into a game system, the complex interactions of many mechanics will crank out a series of outputs that can be collectively understood as having some sort of high-level meaning that will in turn influence future input. This cycle of input and output through the arrangement of mechanics is **gameplay**.

When a game designer is designing a game, it is with an eye towards the resulting gameplay. It is important to choose and arrange mechanics such that the desired gameplay is achieved (see Section 3.2.6).

It is also important to identify and understand the separation of gameplay and the aesthetic elements used to represent that gameplay. Consider the example of id Software's *Wolfenstein 3D* [7] and Wisdom Tree's *Super 3D Noah's Ark* [8]. In *Wolfenstein 3D*, the player takes the role of an American soldier that storms a Nazi complex in World War II, while *Super 3D Noah's Ark* involves the Biblical Noah trying to calm down the animals on his ark. The former depicts violent shootings and bloody corpses while the latter includes fruit-firing slingshots and "sleepy" goats, sheep, and the like.

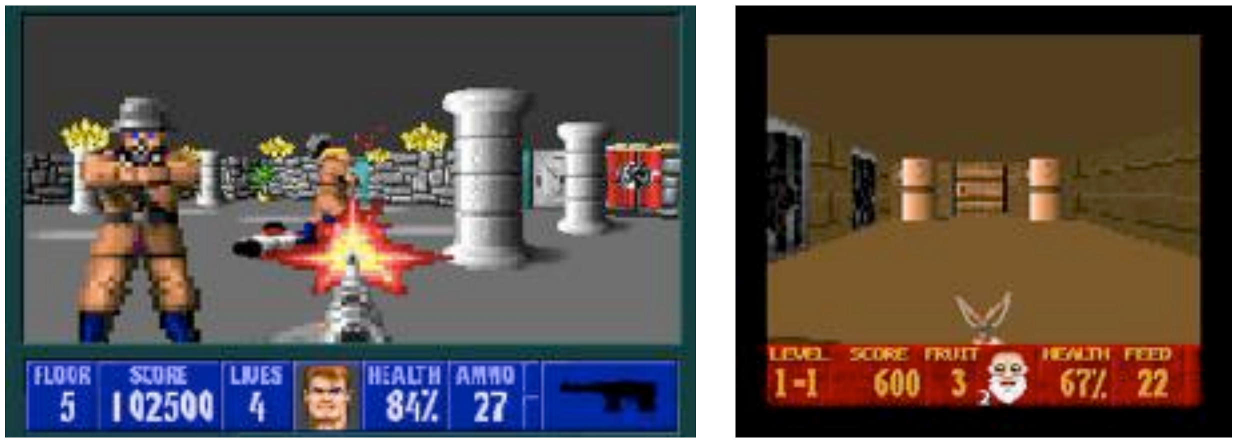


Figure 1: *Wolfenstein 3D* (left) [9] and *Super 3D Noah's Ark* (right) [10]

Despite their different aesthetics, the two games actually share the same source code [11] and have nearly identical mechanics and gameplay; they just represent this gameplay in different aesthetic contexts (also called domains). By acknowledging this separation, we can more critically analyze gameplay and the actual mechanical ideas and skills being taught through the gameplay. This is also important when considering the procedural narrative generated by the combination of gameplay and its aesthetic elements, as seen in Section 2.1.4.

2.1.3 Design Space

A game design provides a domain of mechanics that result in a range of gameplay possibilities known as a design space [12, 13, 14]. Each mechanic can be thought of as a sort of "dimension" to this design space. Different gameplay ideas can be explored within this space.

For example, consider the previous "jump" mechanic, defined in a spatial context as in the classic video game *Super Mario Bros* [15]. If there was no "jump" mechanic, the design space would be dramatically reduced. Without the ability to jump, many mechanics lose their contextual relevancy; ideas like "jumping" over obstacles or onto enemies no longer have meaning because they are not within the range afforded by the game mechanics.

Now let's reconsider a game like *Super Mario Bros.* within the context of a "jump" mechanic. Depending on its implementation, a wide range of gameplay ideas could be explored, but for the purpose of illustration let us consider the mechanics mentioned in Section 2.1.1.

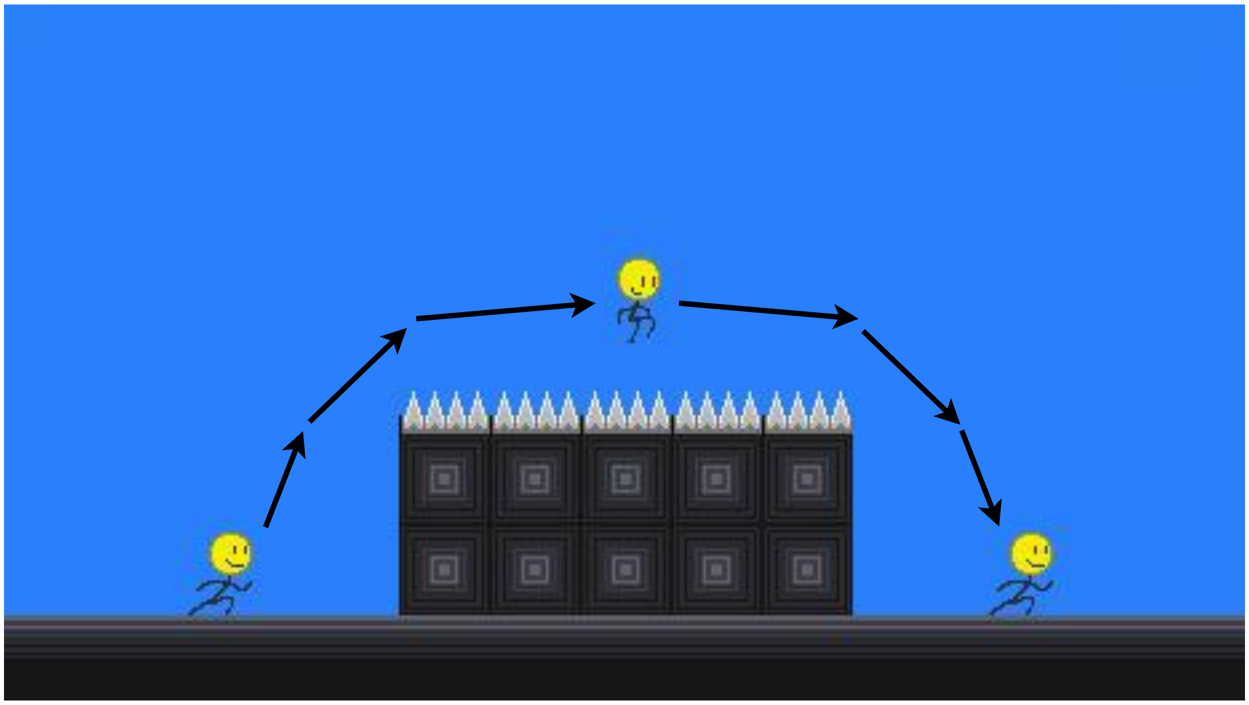


Figure 2: Long and tall obstacles are made meaningful by the design space [16]

The formula describing a player's motion through a jumping arc will affect how the player interprets and interacts with the environment. If the player's vertical velocity slows near the apex of the jump in a parabolic motion, the player will spend a proportionally larger amount of time higher in the air. Coupled with a fast horizontal acceleration, an obstacle like the one in Figure 2 could be surmounted by the player. Thus, the gameplay idea of long and tall obstacles can be explored through the *level design*, i.e. a specific arrangement of mechanics designed to explore or communicate ideas through gameplay. If these formulas were to change (say, the vertical component is more linear, or horizontal movement has slower acceleration), the types and shapes of obstacles that can be implemented must change.

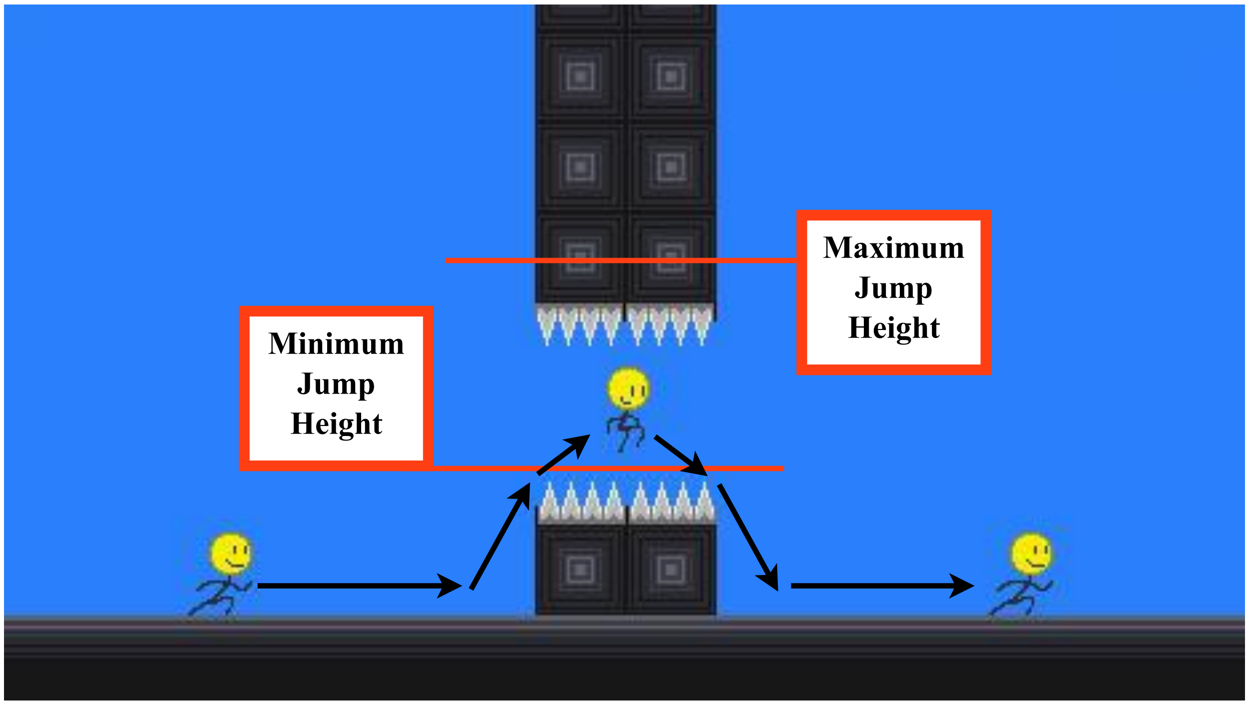


Figure 3: This aspect of the design space is made by variable jump height [17]

Let's say the height of the jump can be adjusted via player input. Without having to assume that the player will always jump to the same relative height, a designer can now construct hazards like the ones above. They can only be circumvented by jumping to the correct height. If all obstacles were of the same height, this mechanic would lose relevancy, and that dimension of the design space would be empty.

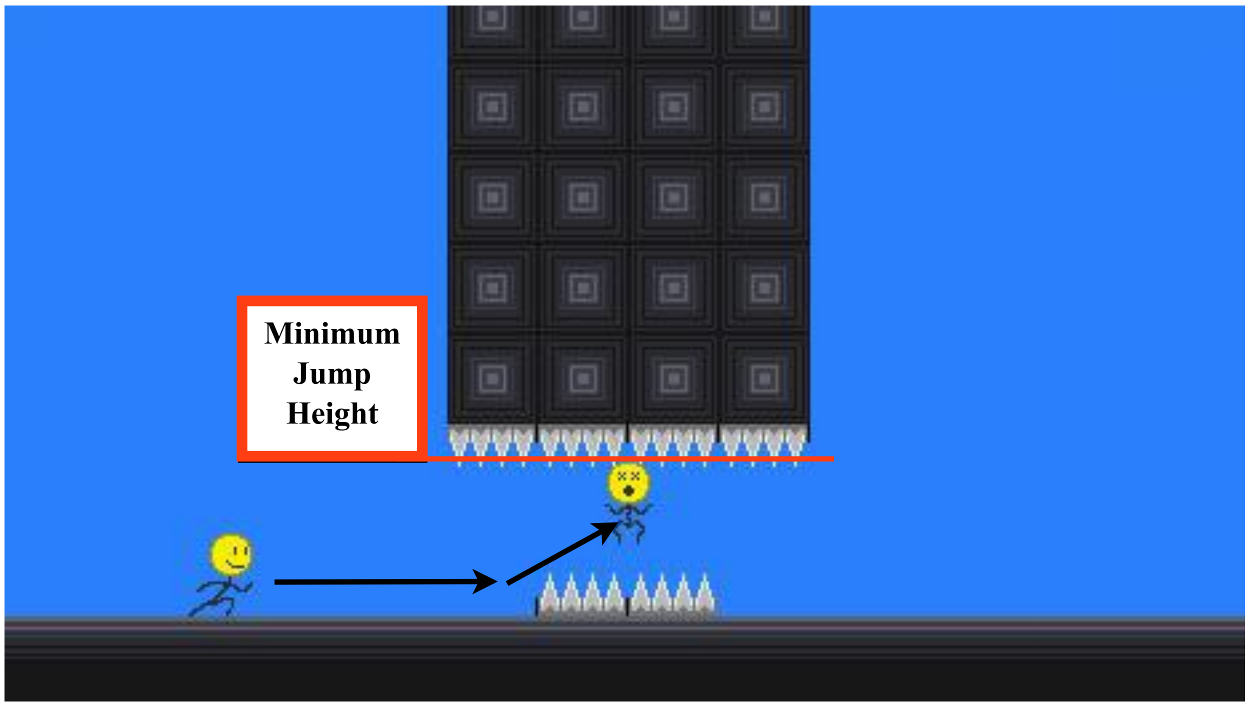


Figure 4: A minimum jump height gives gameplay context to this level obstacle [18]

Finally, a minimum jump height forces the player to consider the above scenario in a different light. Because the minimum height would force the player to hit the spikes on the low ceiling, an alternate route must be explored. Without this mechanic, the player could simply jump lightly over the lower obstacle.

In all three examples, these "jump" mechanics affect what types of arrangements of obstacles can be constructed. In other words, they affect the types of gameplay ideas that can be explored. In this context, the gameplay ideas are "physical" and "spatial" in nature, but the same principle applies to any mechanical domain; if the subject were algebra, a "division" mechanic could allow for the idea of "fractions" to be explored. It is important that a game's mechanics be designed and arranged such that the resulting gameplay and design space match the desired range of knowledge that is to be gamified (see Section 2.1.5).

In addition, it is ideal to have a game explore its design space in an efficient manner. That is, the desired design space should be reached with a minimal number of mechanics, and the entire breadth of the design space should be explored with minimal overlap. The former is to reduce unnecessary complexity which would otherwise act as a barrier to the player's learning without contributing significantly to the transfer of knowledge. The latter is both out of respect for the player's limited time and energy, and to effectively teach and evaluate all of the ideas afforded by the design space. By "overlap", this means if the design space is again thought of as some n-dimensional space with gameplay ideas occupying some space of intersecting mechanics, it is desirable to have these intersecting spaces be "spread out" across the entire design space and not overlapping or "clumped" in one or a few regions. This is analogous to a final exam in a traditional class setting, where the exam would be designed to test a student's knowledge in most or all of a course's subject matter and not just one or two items. This is also similar to the narrative principle of Chekhov's Gun [19] in that we want to utilize the full range of gameplay offered by the design space; if a mechanic or idea is going to be introduced into the gameplay, then it should be explored. Otherwise, it is not serving a purpose and is either contributing to unnecessary complexity (and should be removed) or the game is not effectively teaching this item of knowledge (in which case the design should be adjusted to incorporate it).

2.1.4 Gameplay Themes and Ludonarrative

When crafted and ordered in a particular way, a game's mechanics can communicate high-level ideas or arguments through gameplay. The rules of a game will lead to a natural conclusion, with the context giving it deeper meaning.

For example, consider a resource-driven game where the goal is to reach a certain level of acquired resources. Due to the game's mechanics, the optimal strategy might be to share the player's resources with other non-playable characters that will in turn grow and return the favor many times over. Here, the author might be arguing that *altruism* and *selflessness* are ultimately rewarding and profitable, thus suggesting the player should incorporate such qualities into their own life. Or, perhaps with a slightly different arrangement of mechanics, the optimal strategy of the game would be for the player to keep all of their resources and find ways to acquire the resources of the other non-playable characters by any means necessary. In contrast to before, this author might be preaching *selfishness* and *independence* as successful life qualities. In neither game is there an explicit narrative; without the use of language, the gameplay alone can communicate a high-level idea.

These mechanical ideas could be considered "themes through gameplay," but the concept has been more formally described by terms like Dr. Ian Bogost's "procedural rhetoric" [20] and game designer Jonathan Blow's "dynamical meaning" [21]. Although each individual might have slight variations on their definitions, the common idea is that a game's mechanics elicit certain themes or arguments. When combined with a game's ludonarrative (the narrative of a game as influenced by a player's input), it is important that the two are in harmony. For example, a game that is promoting peace and non-violence through its narrative should not include mechanics that encourage players to subvert or harm other players or factions. A disharmony between gameplay themes and ludonarrative is called "ludonarrative dissonance", a term coined by game developer Clint Hocking [22]. Ludonarrative dissonance is the result of conflicting or opposing ideas being communicated at the same time, which is not desirable when the goal is to transfer an idea or item of knowledge clearly to the player.

Understanding of ludonarrative and gameplay themes is essential for the gamification of subject matter in areas like literature, art, and history. However, these ideas are important for clear communication in any serious game that attempts to layer art or narrative on top of its gameplay (see Section 3.2.9).

2.1.5 Gamification

The term **gamification** is a recent term that varies in its definition and meaning depending on who is being asked. It generally refers to applying game mechanics and design techniques to products and services in order to better engage users. Often employing psychological tricks and incentives, gamified elements are usually layered on top of existing components and attempt to reframe information in a way that is more attractive to users. If a particular action is not desirable to a consumer, it might become "gamified" by attaching a badge or achievement to the action that instills a sense of "reward" or "progress" in the user, thus incentivizing the action [23]. If users do not feel an attachment to a service, a social ladder or leaderboards might be introduced to retain a user through social ties and obligations.

Many of these ideas are compelling and worthwhile when applied to certain games. However, this approach to gamification could be criticized because it implies that the underlying work was not designed with effective game design principles. Usually the product is designed without an application of critical game design methodology and instead relies on "gamified" elements to substitute for quality design.

Rather, this thesis will define **gamification** as a transformation of actionable knowledge or behavior to the user through the medium of a game. That is, after playing the game, the user should have acquired some degree of mastery over a designated body of knowledge or behavior.

2.1.6 Unity

Unity [24] is a video game engine developed by Unity Technologies. It provides tools and resources for rapidly developing both 2D and 3D gaming applications by integrating code, art, sound, and other assets into a streamlined environment. Using an intuitive "drag-and-drop" interface, developers can easily build GameObjects by pulling assets into the current scene. A "Play" feature allows developers to instantly start running and editing the game in real-time. This enables programmers to quickly engineer and iterate on scripts written in C# and JavaScript.

Unity also has extensive features and support. Third party tools packages and art assets can be purchased through the Unity Store. There is integrated animation support through Mecanim and a particle system and editor in Shuriken. Web communication protocols like SCORM (see Section 4.2.6) are supported. Many new and upcoming gaming technologies offer Unity support through third parties, including the Oculus Rift virtual reality headset and Microsoft's Kinect camera. Version control for all aspects of game development (textures, music, code, etc.) are offered through the Unity Asset Server with support for outside source control tools like Git or Perforce. Unity can export to a plethora of platforms ranging from desktop computers to mobile platforms and even gaming consoles, all using the same base project. Unity games can even be played and streamed through a web browser (see Section

4.2.6). And even if a developer cannot afford a professional Unity license, there is a free version of the engine that includes nearly every feature an independent game developer would need to build and sell a quality game.

It is for all of the above features that the *Mythos Unbound* team opted to choose Unity as its development platform. Considering the team was composed of students with no game industry experience, the ease of use was vital for productivity. The team knew the game needed to be deployed online to Blackboard services, so the web player and SCORM support was key. Since the team didn't have the resources to construct an entire game without tools from scratch, access to both first- and third-party technologies would speed our development and lower costs. And, most importantly, many of the team's members had a background in Unity and its workflow, so it was simply the best means to tackle such a large endeavor.

2.2 Related Work

The field of serious games and gamification is still relatively new, and there is a distinct lack of research being done in the area. However, there are a number of related projects and endeavors that both connect to the proposed framework and also demonstrate the potential of this emerging field. They also provide context for the serious game *Mythos Unbound* that is discussed in Chapter 4.

2.2.1 Learning

The serious game framework being proposed is backed by relevant ideas related to learning. In behavioral psychologist B.F. Skinner's *The Technology of Teaching* [25], he describes techniques that can be applied to the learning process to enhance the acquisition of skills. One of the key steps is to break down tasks into small goals that can be achieved one at a time, building up to larger, more complex skills. When a student attempts to complete a task, he must be provided with immediate feedback regardless of whether or not the attempt was successful. (In fact, immediate feedback is often cited as having a critical effect on learning [26, 27, 28].) This feedback should not be punitive; instead, correct actions should be rewarded with positive reinforcement. These steps should be repeated in small, tight cycles until the student has mastered the desired skill, at which point he graduates to the next bite-sized chunk of knowledge. Over time, sophisticated ideas and skills can be transferred to the student with these methods.

What is interesting to note is that most traditional university courses do *not* follow these principles. Due to educators' limited time and resources, it is common for only a handful of larger assignments to be given to students. Feedback is often not provided for days or even weeks after submitting homework, let alone immediately after a student answers each individual question. Even after receiving delayed feedback there is no opportunity for repetition, and the student is forced to move onto the next topic even if the previous idea has not been mastered.

In contrast, games are uniquely suited to following these principles in widespread applications. Since games can be widely distributed, developers can justify larger budgets with more time and resources than a single teacher can provide. As such, a game can afford to cover a wide variety of material that can be broken down into smaller gameplay units. A game's dynamic and interactive nature allows it to provide immediate feedback to a player's input, providing a completely customized experience that adapts to the player's successes and shortcomings. This enables players to continually repeat tasks through programmatically generated problems and tests until mastery has been achieved.

2.2.2 Khan Academy

Khan Academy [29] is an online educational resource that features a catalogue of lecture videos covering a wide range of topics across subjects like mathematics, science, and history. The site includes questions and problems to evaluate a user's understanding of the material. Tools and metrics are used to process a user's input and provide feedback by recommending which videos and topics the user should explore next. This new approach to education is seen as having much potential, with organizations like the Bill and Melinda Gates foundation and Google donating millions of dollars to the site [30].

Founder Salman Khan describes Khan Academy as "flipping" the classroom by emphasizing the use of well-designed content delivery means (formerly "lectures" and the like) *outside* the classroom, and instead using class time for problem-solving and personalized teaching [31]. The idea is that technology can be used to create more dynamic learning tools that can adapt to a user's needs and most effectively transfer a base set of knowledge to the user. Khan claims that they can measure when a user pauses or stops a video, indicating which segments of a lecture are confusing or ineffective. Combined with data received from user answers to questions and problems, Khan Academy has a means of computationally understanding a user's knowledge and can iterate and improve upon it.

Khan Academy is founded on three core learning principles [32]. The first is that learning should be mastery-based, that learning should consist of mastering given concepts and ideas before moving onto more advanced content. There is research to suggest that this is a highly effective learning method [33], which factors into Khan Academy's emphasis on the philosophy and its integration into their foundational knowledge maps (see Section 3.2.3). This also parallels the learning ideas presented in Section 2.2.1 in that smaller cycles of learning are necessary to acquire skills before advancing to higher-level content.

Khan Academy's second principle is personalized learning. The emphasis is on adaptive teaching based on immediate feedback. Again, immediate feedback is referenced in Section 2.2.1 as being key to effective learning and allows content to be adjusted to fit a user's particular needs.

The final principle is that learning should be interactive and exploratory. Users should be engaged in dynamic problem-solving that allows them to apply the material they are learning. Unlike traditionally static assignments and tests, games are defined by their interactive nature. They can be designed to be ever-changing and create new scenarios and situations to evaluate user skills. The potential for networked multiplayer even opens the door for peer collaboration, which Khan Academy cites as being ideal to the learning process.

After a recent update [34], Khan Academy's features now include levels, badges, and leaderboards. As the site evolves, it is becoming more and more game-like, emphasizing interactivity and adaptability to individual users. This increased interactivity is even evident in some of the video lectures. For example, in most of the programming lectures there are code scripts being run live alongside the audio. The speaker is actually editing the code in real-time alongside the lecture, and the user can "rewind" or "fast-forward" the lecture at any time and the code will change to match it. The user can even edit the code himself at any point during the lecture for the purposes of experimenting or testing for understanding. It would not be difficult to see how this sort of interaction could be extended to fields like physics or mathematics.

2.2.3 Edufitment

From fall 2012 to summer 2013, the author was involved with a team of developers led by University of Arkansas MS student Chad Richards. Working with health care specialists, education experts, and undergraduate computer science students, the team tackled the problem of child obesity through the use of gamification. The result was *Edufitment*, an online hub of mini-games aimed at teaching young elementary and middle school students nutrition and fitness through applied game design principles.

One of the lessons learned from that project was that it is not easy to take a number of mechanically unrelated ideas and attempt to wrap them all up in one large gameplay system. No one gameplay system could cover all the wide range of knowledge and changes in habit that can substantially change a person's behavior related to nutrition and exercise.

The solution was a more modular approach. By identifying key ideas that were to be taught, the team could construct separate smaller "mini-games" designed around each one. They were smaller in scope, easier to understand, and could explore wildly different design spaces without infracting on one another. It was also much easier to delegate tasks by assigning team members to individual mini-games, thus allowing the team to operate as separate, independent groups. The mini-games would all plug into a common hub that would tie together the varied ideas and provide a consistent environment in which a student's progress could be contextualized.

Some of these ideas would later influence framework presented in this thesis. Because both *Edufitment* and *Mythos Unbound* were developed concurrently with the author acting as the sole link between them, the framework was influenced by various aspects of the two. For example, both projects took a modular approach to design due to disparate ideas requiring separated but ordered gameplay systems; this revelation inspired a major tenet of the framework (see Section 3.2.3).

As examples of this modular approach, presented below are the three mini-games the author developed for *Edufitment*. Take particular note of how the knowledge items that were to be taught had to be explicitly defined and were at the core of each gameplay system.

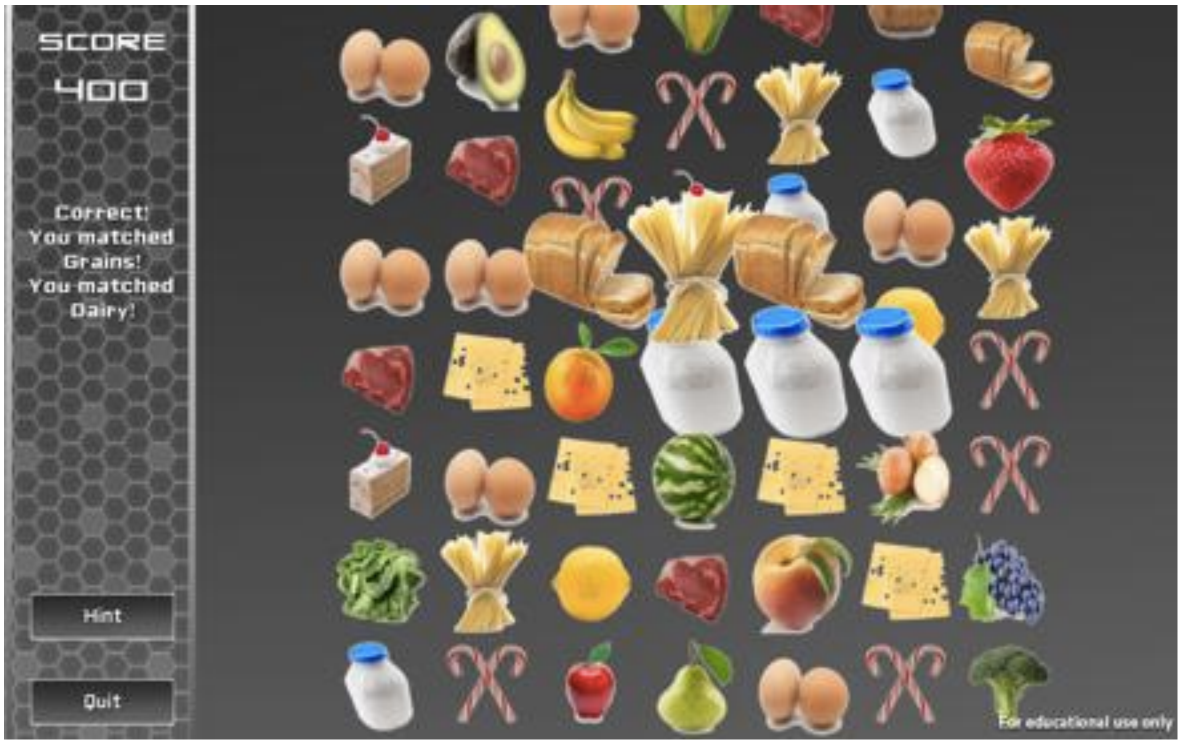


Figure 5: *Food Group Match Up*, an *Edufitment* game the author developed [35]

Inspired by the puzzle classic *Bejeweled*, the goal of *Food Group Match Up* is to swap adjacent cells of food items to create strings of matching food items that share the same food group. The goal behind this game was to test for and reinforce knowledge of food groups. By applying a twist to a familiar gameplay system, students have to think outside the box to discover matches. The nature of the game's mechanics also ensure that a student cannot progress without knowledge of food groups.

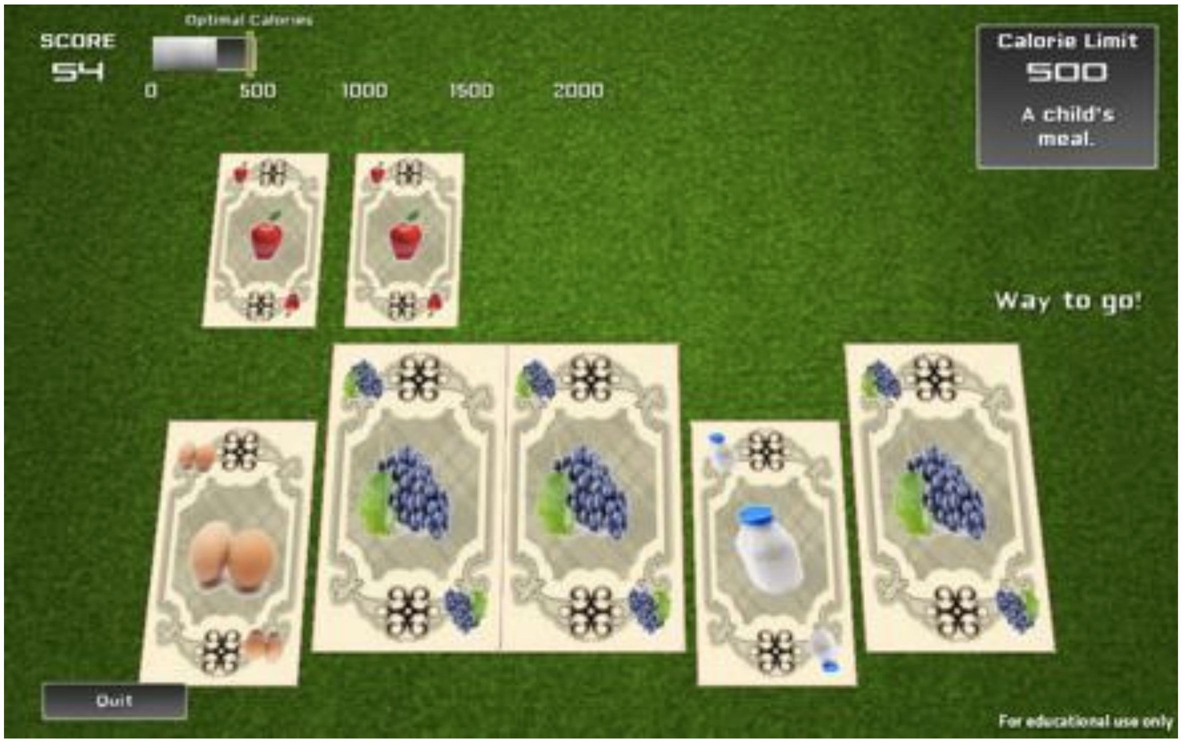


Figure 6: *Calorie Meter: The Card Game*, another *Edufitment* game [36]

The concept of "calories" was one of the ideas that students were to learn. "How many calories does the food item contain?" and "What is an appropriate sum of calories?" were questions students needed to answer. *Calorie Meter: The Card Game* embeds these questions as game mechanics in a blackjack-esque game that replaces the numbers on its playing cards with pictures of various foods. The food pictures symbolize a number that is equal to the number of calories that food item contains. From his dealt hand, a player must submit a combination of cards with a combined calorie sum that gets as close as possible to a given calorie limit without actually going over that limit and "scratching." These calorie limits represent real-world values, such as "the number of calories in a child's meal."

Clearly, a player can only succeed by knowing the values of each card, thus ensuring knowledge of calorie values. The meanings behind the calorie limits also helps students associate a combination of food items with a label that gives context to those number values.

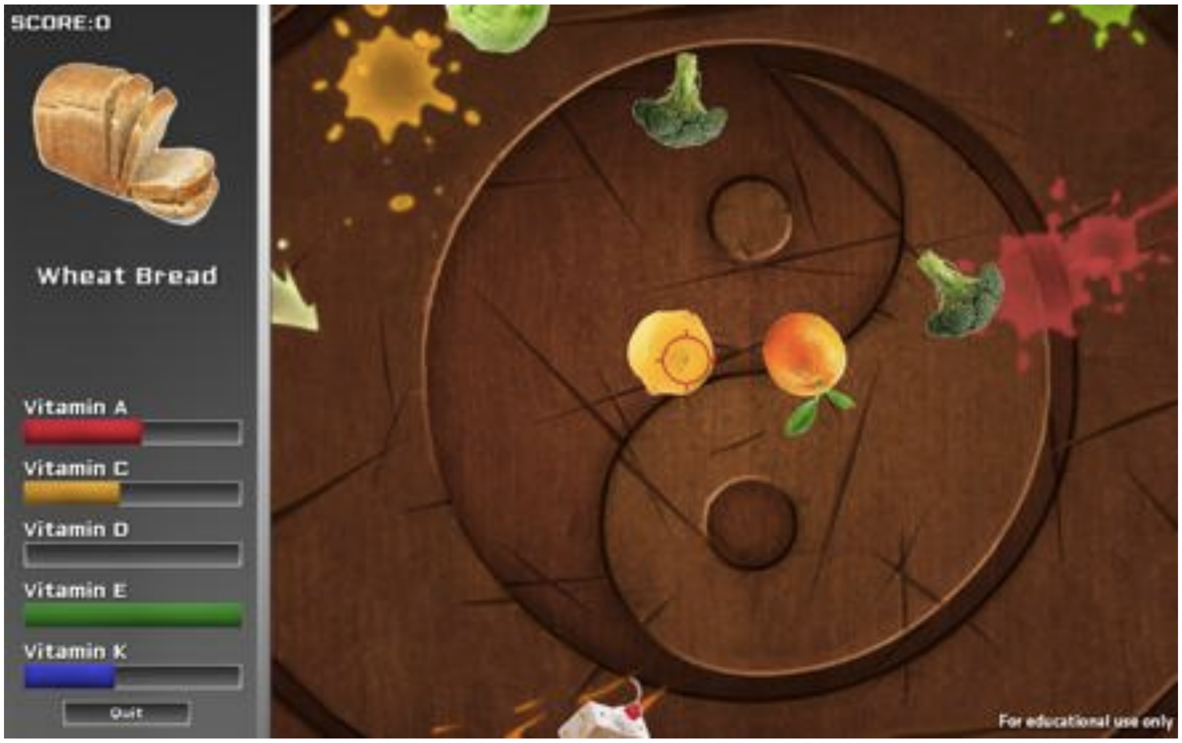


Figure 7: *NutriHunt*, the last of the author's *Edufitment* games [37]

The author developed *NutriHunt* with the help of undergraduate student Andrew Johnson. For *NutriHunt*, the team needed to have students associate food items with nutritional values ("orange" with "vitamin C," for example). This was accomplished by creating an action game where food items would fall from the sky and the player must shoot appropriate food items with a slingshot to obtain its nutrients. The game was separated into "levels" where success was found by reaching certain thresholds for specific nutritional values. However, certain mechanics (such as food group restrictions, limited ammo, and time limits) forced players to make judicious choices by striking only those food items that were most optimal. In order to do this players must know which food items contained which nutrients, which was the educational goal of the mini-game.

2.2.4 Other Forays into Serious Games

Although the realm of serious games is still in its developing stages, major strides are being made that are attracting both attention and money. At the forefront is the Game Innovation Lab housed at the University of Southern California under the leadership of professor Tracy Fullerton [38]. With grants ranging from The National Endowment for the Arts to The Department of Education, the Game Innovation Lab's projects cover serious topics such as the crisis in Darfur (*Darfur is Dying* [39]) and the life of Prime Minister of Iran Mohammad Mosaddegh (*The Cat and the Coup* [40]). The goal is to take themes, ideas, and knowledge and translate them into interactive works that leverage gameplay to communicate their meaning. For example, *Collegeology* [41] gamifies skills like writing college applications and acquiring financial aid in order to help younger students learn how to succeed in higher education. *Walden, A Game* [42] is another USC project that is designed to recreate Henry David Thoreau's years at Walden Pond, embodying his thoughts and philosophies in the game's mechanics and environments. Graduates of the Game Innovation Lab have gone on to create other more serious and artistic games in the private sector, such as Thatgamecompany's *Journey* [43] that went on to win multiple Game of the Year awards.

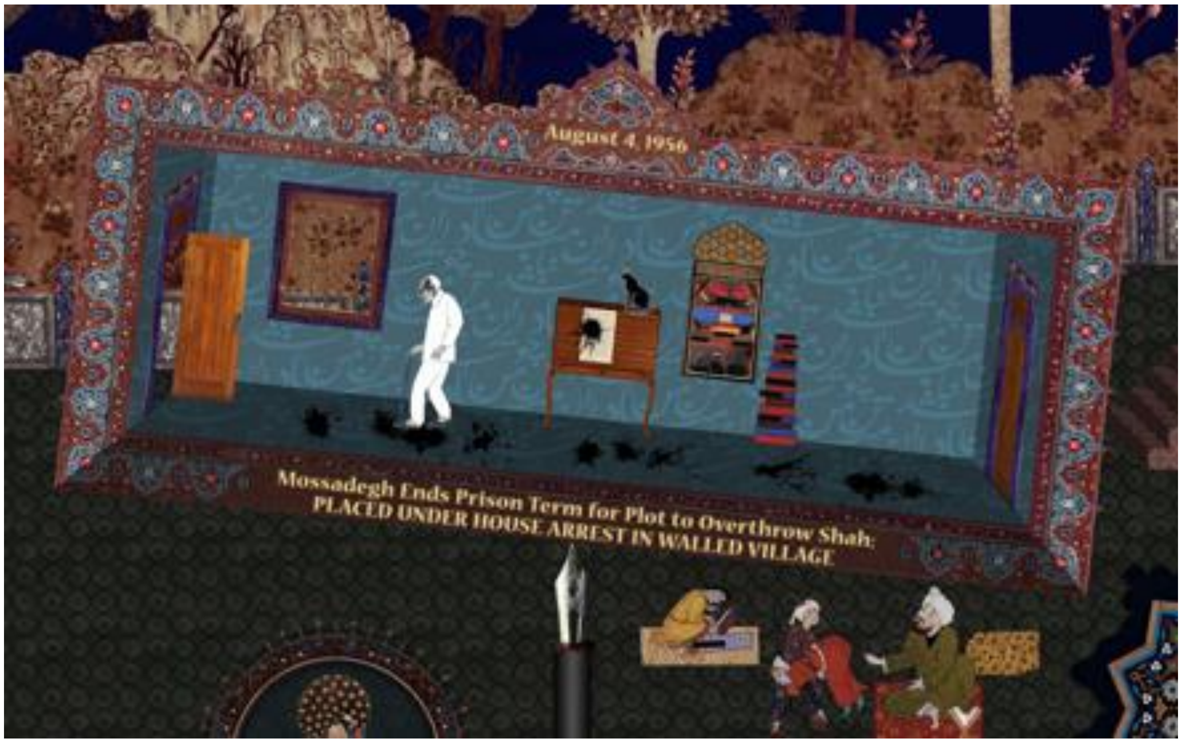

August 4, 1956
Mossadegh Ends Prison Term for Plot to Overthrow Shah:
PLACED UNDER HOUSE ARREST IN WALLED VILLAGE

Figure 8: *The Cat and the Coup* follows the life of Mohammed Mosaddegh [44]

Like the Game Innovation Lab, the Games Learning Society [45] at the University of Wisconsin-Madison is behind a number of serious games. For example, games like *Citizen Science* [46] and *Trails Forward* [47] teach STEM ideas about biology and ecology while also promoting community involvement to solve real-world issues. Even social issues like discrimination and prejudice are confronted in a game like *Fair Play* [48], which has players take on the role of an African American graduate student pursuing a career in academics. The Games Learning Society's efforts have earned them grants from the National Science Foundation and the MacArthur Foundation [49].

Extending beyond academia, serious games have been developed in private industry to communicate complex themes through game mechanics. Independent developer Jason Rohrer is well-known for leveraging gameplay to represent his ideas; *Diamond Trust of London* [50] exposes the sad realities and history of the diamond trade, while *Passage* [51] symbolizes the brevity of life. Ian Bogost (who coined and described the idea of "procedural rhetoric," see Section 2.1.4) and his company Persuasive Games developed *Take Back Illinois* [52] for the Illinois GOP, a game that promoted Republican policies through the use of game mechanics and procedural rhetoric. The recent release of Lucas Pope's *Papers, Please* [53] has players assume the role of an immigration officer that must determine who to admit to the country, who to turn away, and who to arrest, forcing players to reflect on their choices and better empathize with both immigrants and inspectors alike. Even respected developer Valve is beginning to throw their hat in the ring with their Steam for Schools initiative [54]. Utilizing their engine and mechanics for the popular game *Portal 2* [55], educators can craft levels and scenarios designed to teach students about physics ideas like oscillations, gravity, and terminal velocity [56].

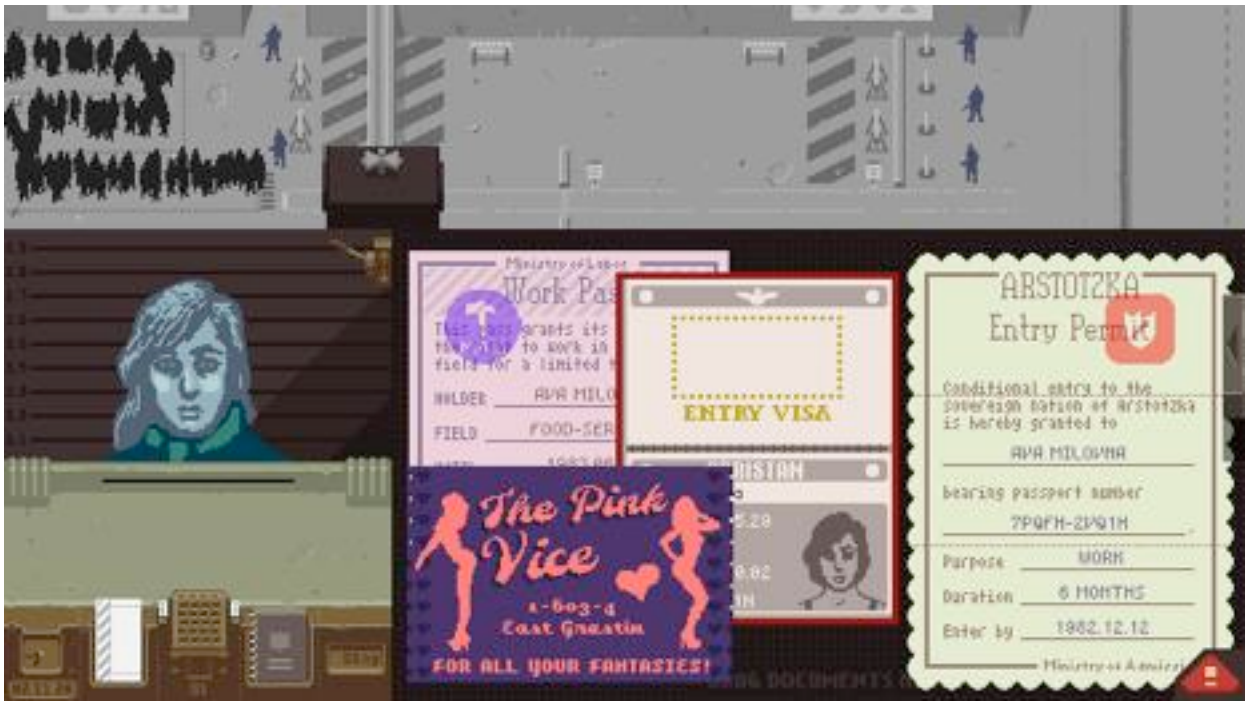
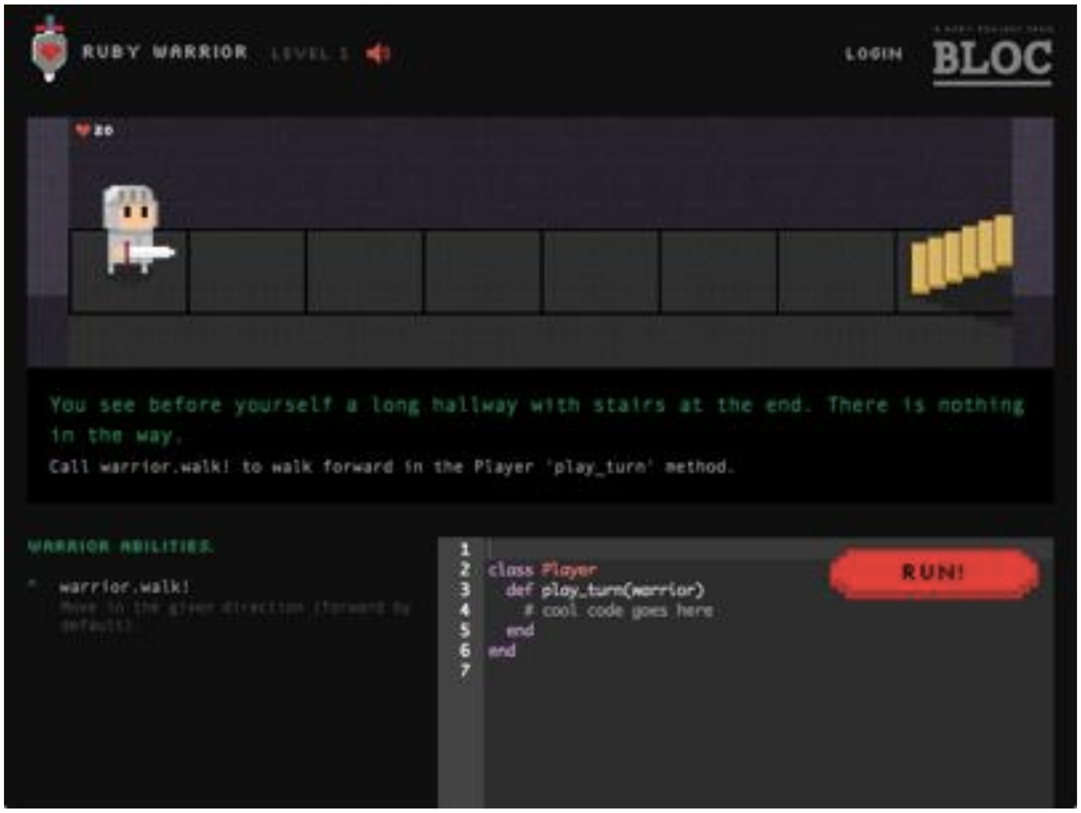


Figure 9: *Papers, Please* puts players in the role of an immigration officer [58]

Even in the author's field of computer science, there are serious games being created with the goal of teaching the fundamentals of programming. *Code Hero* [59] gives players a "code gun" that is used to overcome challenges. Players can either write their own code or edit code copied from the game world. This code is then "fired" at game objects that will then execute the code. For example, take the following line of code: `hitObject.transform.position.y += 4`. "Firing" that code at a platform will cause it to rise four units in the Y direction, creating a bridge needed to cross a short gap. By translating a programming idea into a mechanic that has contextual meaning within the game, players can practice and master specific skills like this.

In a similar vein, *Ruby Warrior* [60] gives players small programming challenges in the form of grid-based puzzles. Using the Ruby programming language, players must write short programs to guide a knight out of a small dungeon filled with monsters and captives that need to be saved. The levels are cleverly designed to continually build upon a player's learned skill set, adding conditionals, loops, and other ideas over time into the domain of mechanics. The game's interactive nature provides immediate feedback, visually demonstrating exactly where faults in logic lie when, say, the knight dies because he did not use an `if` conditional statement to determine if he should heal before attacking. This allows players to iterate and improve their understanding of the game's mechanics (and its corresponding programming knowledge).



You see before yourself a long hallway with stairs at the end. There is nothing in the way.

Call warrior.walk! to walk forward in the Player 'play_turn' method.

warrior.walk!

Move in the given direction (forward by default).

```ruby

class Player
  def play_turn(warrior)
    # cool code goes here
  end
end

```

Figure 10: *Ruby Warrior*, a game that teaches the basics of Ruby [61]

There is clearly potential in serious games as evidenced by the works outlined above. Despite these successes, there is still a lack of critical thought as to how a serious game should be designed and developed. There is a need for a formalized approach to gamifying knowledge, and the framework being proposed is a significant step in that direction.

3. FRAMEWORK [1]

3.1 A Critical Approach to Gamification

The area of gamification is a relatively new field for academia, and as such there has been little research or progress made despite its clear potential (see Section 2.2). When it *is* discussed, it is usually in case examples, rules of thumb, and individual ideas or mechanics that lack an eye towards the overall construction of the work. What is needed is a generalized approach, a way of thinking about gamification in the shape of a more formal framework architecture and methodology that describes how to transform a set of knowledge or ideas into a serious game. The following framework meets that need.

3.2 Framework Steps

The framework is divided into a number of steps that outline the process and principles needed to develop a compelling serious game. Sections 3.2.1 through 3.2.5 represent a preliminary formalization of knowledge, whereas Sections 3.2.6 through 3.2.8 (defining game mechanics and the design space) and Section 3.2.9 (creating art, narrative, and sound) can be considered somewhat concurrent.

[1] While the author is not an expert in education or psychology, he has an extensive knowledge of game development and has been a key architect on both the *Edufitment* and *Mythos Unbound* serious game projects. This section codifies and generalizes the lessons about gamification that he has learned from these projects.

3.2.1 Identify Domain of Knowledge

The very first step in gamification should be a formal identification of the domain of knowledge that is to be taught. It is not enough to simply say, "We are going to gamify Chemistry," or some other arbitrary subject matter. The name of a subject alone does not define specific, concrete ideas. The domain of knowledge must be explicitly identified and articulated in a formal, objective manner. Such enumerated ideas in a subject like Physics I might include velocity, acceleration, momentum, and spring physics.

This may sound obvious or perhaps even trivial, but it is an extremely important step. One of the major issues encountered with the *Edufitment* project was that the domain experts on the team never formally acknowledged the domain of knowledge being taught. Most of the mini-games targeted whatever health-related topics first came to mind as being important, but we didn't have a structured approach to communicating subject matter. As such, the final suite of games lacked a clear and unified progression from one idea to the next, and ultimately they served more to *test* knowledge than to *teach* it.

It is easy to assume that a general umbrella subject is understood by all, that everyone envisions the same ideas represented by that identifier. And it is easy to think of games as being playful, fun, and informal, and thus assume actual development must also be equally lax and freeform. Without prior experience, one might assume that individual levels and mechanics can be developed haphazardly as long as they are enjoyable and have some connection to an arbitrary knowledge item within the subject, and that they will magically all come together and remain cohesive and intuitive in the final product.

The above is certainly not the case. Effective game design, especially for the purposes of gamification, requires a very formal, logical, and structured approach. If the expectation is that a player will acquire a specific array of knowledge items through gameplay, then game development cannot even begin until a critical approach is taken to defining what that body of knowledge consists of.

3.2.2 Decomposition of Knowledge

Once the domain of knowledge has been formally identified, it must then be decomposed into a collection of atomic ideas. What this means is that any higher-level ideas within the body of knowledge must be deconstructed into component ideas, particularly if multiple higher-level ideas share common component ideas. For example, in an art course, one knowledge item might be "impressionism," which can be decomposed to include the lower-level item of "painting," which can be decomposed even further to include the idea of "color theory." The "color theory" knowledge item might be shared with another higher-level idea, such as "drawing."

For certain subject matter, this step might seem redundant or unnecessary, especially depending on how specifically the knowledge items were defined in the first step. There is also an element of subjectivity as to how low-level a knowledge item must be to be considered "atomic." However, the importance of this step is in thinking critically about knowledge content and identifying the underlying ideas that define a subject matter. This step is to help designers define common low-level ideas and start seeing the connections and dependencies between them. By formally stating and acknowledging these items, step three can be better implemented.

3.2.3 Arrange Ideas into a Knowledge Hierarchy

A common convention in role-playing games is to have “skill” or “talent” trees. Players can place points into skills to learn them, but acquiring high-level skills depends upon the mastery of lower-level component skills. Skill Z cannot be learned until skills X and Y have been completed.

Considering this is a framework for gamification, the above example is fitting since that is exactly what this step entails. Taking the previously enumerated atomic ideas and their higher-level parents, a tree of dependencies should be constructed. That is to say, a map or web must be made that links atomic ideas to the higher-level ideas that depend on them. And from there, the higher-level ideas should be linked to the even greater ideas that depend on them, and so on and so forth. The final result can be called a **knowledge hierarchy**, a gamification “skill tree” that explicitly defines which ideas depend upon the mastery of other ideas. Understanding these dependencies is key in constructing game systems that build upon previously established ideas and push a player towards the acquisition of desired higher-level ideas.

Interestingly, this is very similar to how Khan Academy (see Section 2.2.2) arranges its content and determines how students should progress through its teaching materials. Called a “knowledge map,” it applies the same principles by showing how a great many ideas in a subject are connected and depend on the mastery of previous lower-level ideas. Khan Academy uses this map to recommend educational videos and practice problems for students that lack the requisite base knowledge for a desired skill.

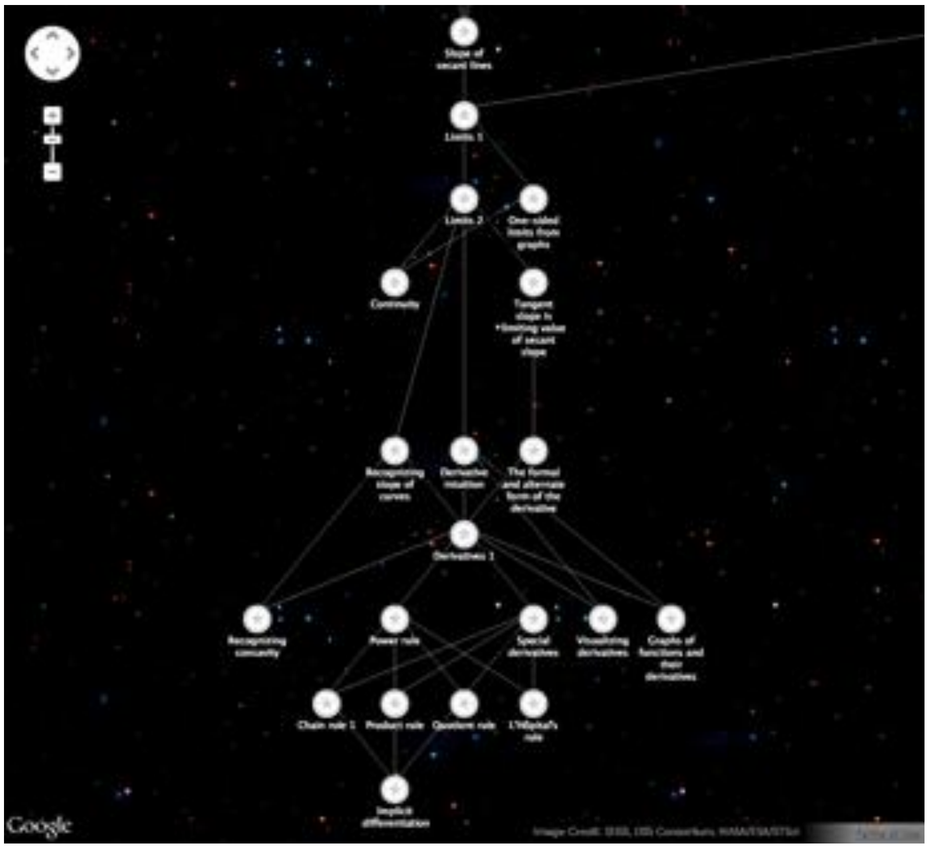


Figure 11: A low-level view of Khan Academy's knowledge map, focusing on Calculus [62]

As seen in the screenshot above, the "Calculus" subject matter is broken down into component ideas. Before "derivatives" can be understood, "limits" and several periphery ideas must be mastered. And further below, "derivatives" feeds into other ideas, like "chain rule" and "quotient rule." If one were to zoom out of the knowledge map even further, it would be seen that the entire group of ideas known collectively as "Calculus" depend upon other groups like "Trigonometry." In fact, if one traces back far enough, the most basic ideas of "Addition" and "Subtraction" can be found at the root of the knowledge tree, atomic ideas that nearly the rest of the field of mathematics depends upon. By understanding and formalizing the relationship of all these ideas, it becomes possible to convert knowledge into a form that a computer can understand and rationalize through gamification.

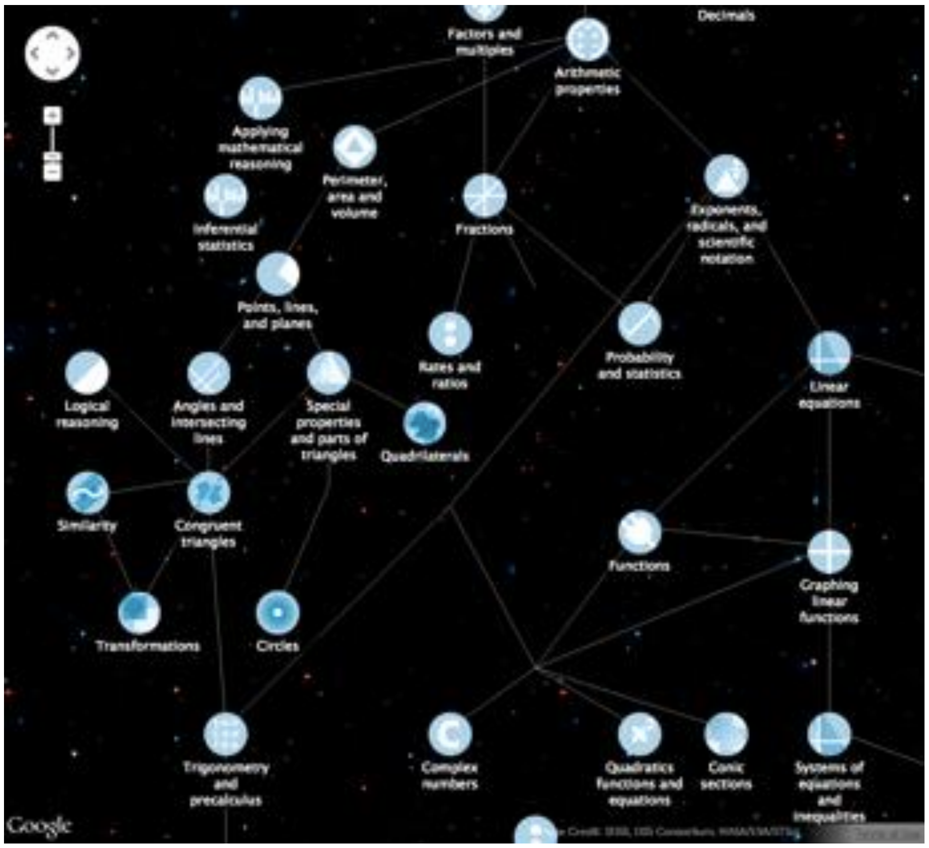

Factors and multiples
Decimals
Arithmetic properties
Applying mathematical reasoning
Perimeter, area and volume
Inferential statistics
Fractions
Exponents, radicals, and scientific notation
Points, lines, and planes
Rates and ratios
Probability and statistics
Linear equations
Logical reasoning
Angles and intersecting lines
Special properties and parts of triangles
Quadrilaterals
Similarity
Congruent triangles
Functions
Graphing linear functions
Transformations
Circles
Trigonometry and precalculus
Complex numbers
Quadratics functions and equations
Conic sections
Systems of equations and inequalities
Google

Figure 12: A high-level view of Khan Academy's Mathematics knowledge map [63]

3.2.4 Identify Branches in the Knowledge Hierarchy

It is very common in a course for later concepts to depend on the understanding of earlier ideas. For example, trigonometry ideas like sine and cosine depend on an understanding of right triangles, which would be learned in a geometry class. Within the previously constructed knowledge hierarchy, these "lines of dependencies," or **branches**, should be identified. If the knowledge map is being given a visual representation, the branches should be arranged vertically, with lower-level base ideas organized near the bottom of the map and higher-level concepts situated closer to the top. Note that it should be common for many branches to join with other branches, or for some items to be a part of multiple branches.

It is important to explicitly acknowledge these hierarchy branches since a game's design must respect these dependencies. Later mechanics cannot be introduced before the mechanics that they depend upon, and the game design must find a way of evaluating or verifying understanding of prior ideas. In other words, any "link" between ideas within the knowledge hierarchy must have an explicit gating mechanic (see Section 3.2.8) embedded into the game's design. Also note that this imposes no restriction on "parallel" branches, or branches that do not share any knowledge items between them, as seen in Section 3.2.5.

Keep in mind that dependencies do not have to be strictly mathematical or skill-based; many branches will consist of conceptual links. For example, learning about ancient Egyptian politics and understanding how the pharaohs derived their power depends upon an understanding of Egyptian religion and knowing that the pharaoh was believed to be a god. These conceptual dependences must be accounted for and organized as best as possible.

Generally, courses in areas like mathematics or science will probably have a larger number of dependencies and thus longer branches. These courses can be described as being "tall" or "vertical" due to their branches. The opposite of this would be "flat" or "horizontal" courses that have shorter branches, but have them in much great quantity. These will affect how the game design can be approached, such as if there will be only a few core systems that evolve over time, or a larger number of smaller mechanics that come and go over the course of gameplay, interlocking where possible.

3.2.5 Identify Tiers in the Knowledge Hierarchy

If knowledge item B depends on knowledge item A in the same branch, then B cannot be introduced until A has been mastered by the player. Similarly, if item Y depends on item X, then Y’s introduction must wait until the player’s understanding of X has been demonstrated. However, because neither A nor X depend on one another, they can be considered to be “parallel” to one another. This means that both ideas could be present concurrently within the gameplay.

In recognition of this, it is possible to arrange knowledge items into **tiers** within their knowledge hierarchies. If branches can be thought of as vertical lines of dependencies, then tiers can be considered horizontal groups of parallel ideas. A tier represents a domain of knowledge items that can potentially all be present in the gameplay at once. It represents a limited pool from which game designers can draw mechanics until the player improves their skills and moves up one or more branches in the knowledge hierarchy, thereby progressing to the next tier and increasing the pool by filling it with new knowledge items.

Explicitly defining tiers is somewhat subjective and fuzzy in practice. Just because some ideas are independent of one another does not necessarily mean they should all be introduced within the same tier, particularly if one of the ideas is an advanced concept that is difficult to understand, or if that tier is already inundated with a large number of mechanics compared to other tiers. It is also difficult to distinguish one branch from another; if a player progresses one knowledge item within a single branch, it’s not clear whether or not that constitutes a new tier. This is especially an issue with flat hierarchies, where branches do not naturally disseminate into obvious tiers based on dependencies. There is just not really an easy, objective way to define tiers in a way that remains relevant to game designers.

Rather, the value of tiers is in describing the general bounds of the gameplay at a given point of progression by identifying all potentially concurrent knowledge items and their mechanical representations. In a sense, a tier represents a subset of the design space afforded by the player skills most recently acquired or introduced. Tiers also provide a high-level view of player progression through content so that designers and educators can get a sense of the rate at which players will be learning skills via their advancement through the tiers. At some level, the labeling of tiers is going to be subjective and up to the discretion of the developers as to what will best lead to an effective game design.

3.2.6 Convert Knowledge into Mechanics

Nintendo game developer, director, and producer Shigeru Miyamoto is well-known for his "form follows function" approach to game design [64, 65, 66]. The philosophy is that gameplay should be at the root of all major design decisions and that a representation of these mechanics can later be crafted to best convey the underlying ideas. For example, Miyamoto's *Pikmin* began as a collection of action-strategy mechanics revolving around ideas of resource management, planning, and efficiency. These ideas later informed the story and aesthetics, representing the mechanics as "tiny space explorers leading groups of miniature plant-men" [67]. The premise may sound strange, but it is incredibly effective at communicating a wide range of ideas and exploring an expansive design space that might not have been possible without the initial focus on mechanics.

As a vehicle for delivering serious content, what gives games an advantage over textbooks and other mediums is their unique ability to communicate complex ideas through gameplay. Critical thought should be given to the purpose of a game's mechanics and their role in transferring knowledge to the player, particularly through the application of the learning principles outlined in Section 2.2.1. A game without mechanics at the foundation of its design risks constricting its design space (and thus its range of explorable ideas) according to elements that might not have direct pedagogical value. Thus, it would seem reasonable for serious games to take a similar approach to game design as Shigeru Miyamoto by prioritizing the construction of mechanics and allowing them to give form to the rest of the game.

Thus, the goal of this step is to examine the knowledge hierarchy and convert its various items into gameplay mechanics and systems. In keeping with the philosophy of "form follows function," it is okay to be somewhat abstract with the mechanics as long as the gameplay is cohesive and effectively communicates its ideas; how these mechanics will be represented and specifically organized will be determined later in the process. The importance of this step is in creating engaging mechanics that are enjoyable and effective in teaching the subject matter, i.e. the resulting design space covers the desired knowledge items.

Just as there are many ways to write an academic textbook, there are many ways to approach the construction of gameplay mechanics, and like the multitude of textbooks covering the same subject matter it is more than reasonable for different developers to have different mechanical interpretations of the same knowledge items. However, there are some key ideas to keep in mind that should lead to improved gamification.

3.2.6.1 Mechanical Representations Across Fields

Knowledge items can range from purely rule- or math-based techniques in fields like science and engineering, or they could be more conceptual as in art or literature. This step's emphasis on mechanics might seem an ill fit for the latter fields, but both can benefit from this approach. For example, a math-dependent field can rely on purely mechanical systems, as seen with Valve's *Portal 2* in Section 2.2.4. However, in a course that depends on the understanding and analysis of complex literary ideas or sociocultural concepts, mechanics can give rise to deliberate, structured narrative and interaction systems, guiding players through events and scenarios that communicate subject matter through procedural rhetoric (see Section 2.1.4). In fact, this is similar to the approach that led to *Mythos Unbound*, as seen in Section 4.3. In either approach, mechanics are the means by which players interact with the game and learn its ideas.

In addition, for many games a narrative might be imperative to communicating conceptual ideas. For example, the narrative in *Mythos Unbound* exposes players to many elements of Roman and Greek culture that could not be easily expressed through pure mechanics. In these cases it is common for a narrative to be developed alongside mechanics during this step, usually with an iterative approach that has the two influencing one another, back and forth. Regardless, the mechanics are still the vehicle by which the narrative is delivered, and so it is important they are continued to be kept at the forefront of development during this step, especially since their implementation will affect other mechanics that might be communicating ideas not dependent on the narrative.

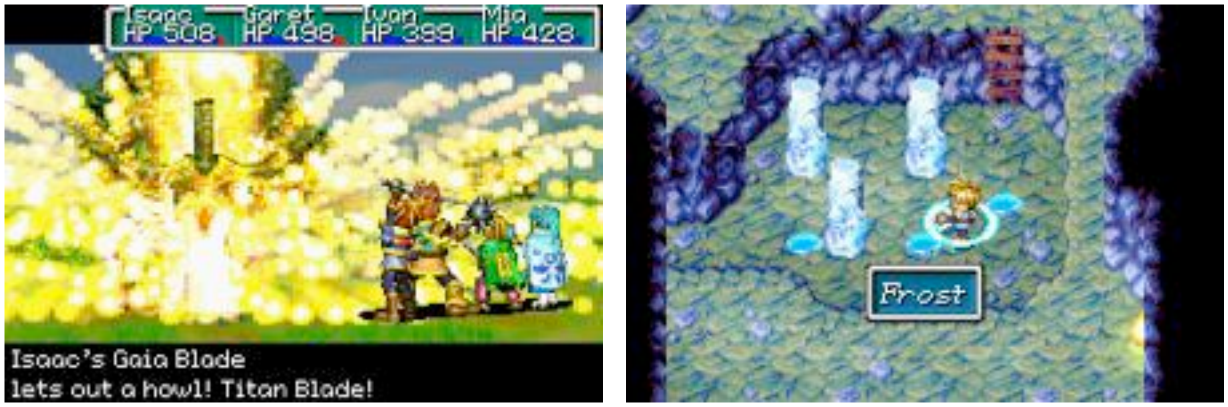
3.2.6.2 Mechanics Adhering to Knowledge Hierarchy Branches

Knowledge items within the same branch of the knowledge hierarchy will, by definition, share common ideas. To reinforce this connection, where possible, branches should embody the same general mechanics or gameplay systems, progressively growing more advanced with the addition of new ideas as players move up the branch through learning and mastery. Not only does this tie together related ideas and help players understand their relationships, it makes the game more elegant by reducing unnecessary complexity. There is no need for the player to have to cope with multiple gameplay systems when the same ideas could be taught with less time and effort on the player's part.

Exploring new gameplay ideas within the context of previously established mechanics is a very common practice in some of the game industry's best-received games. For example, the *Super Mario* series is well-known for its platformer gameplay that has the "jump" mechanic at its core along with associated ideas like gravity and momentum. *Super Mario Galaxy* [68] expands on this mechanic by taking one of its elements (gravity) and building on it in various ways. One stage might have the player capable of reversing gravity in order to navigate environments, while another stage might have the player exploring all sides of a small spherical planetoid by becoming anchored to its gravitational field. Rather than use separate gameplay systems to explore each of these ideas, they are all embedded into the core, established platformer mechanics, building off of previously learned ideas.

3.2.6.3 Mechanics Adhering to Knowledge Hierarchy Tiers

In the role-playing game *Golden Sun* [69], there are two very distinct gameplay systems. When in the overworld, the player navigates the hero through various environments, interacts with NPCs, and solves puzzles. During battle sequences, the gameplay becomes entirely different, relying on turn-based strategy combat and resource management. Even though there is never a mechanical cross-over between the two (i.e. there is no spatial navigation during battle, the overworld operates in real-time, etc.), there are still mechanics at play that relate the two and give context to one another. For example, in the overworld the player can use gold earned from combat to purchase armor and weapons from NPC merchants, thereby affecting the player's performance during battle. Through puzzle-solving and exploration in the overworld, the player can also discover magical Djinn that are the key to mastering the battle system.



Isaac's Gaia Blade
lets out a howl! Titan Blade!

Figure 13: *Golden Sun*'s battle (left) [70] and overworld (right) [71] systems

Despite being separated across parallel branches, knowledge items will usually share relationships of some kind with one another (or else they would not be included within the same subject matter in the first place). It is ideal to reinforce these relationships with connecting mechanics where possible, like in *Golden Sun*'s overworld and battle systems. This ensures that the game remains cohesive and that players understand the high-level connections between knowledge items.

Developers should also be wary of adding too many separated gameplay systems unless necessary. Where feasible or practical, similar knowledge items should be represented under some overarching mechanical umbrella even if they do not have a direct dependence. As mentioned in the previous section, reducing the total amount of complexity is desirable.

3.2.6.4 Relating Traditional Approaches to Mechanics

Similar to a traditional course, different ideas require different approaches to practice and mastery. A calculus course might provide weekly homework with a large quantity of small problems, whereas a history course might require a large research paper written over the course of a couple months. Understanding what knowledge items need to be tested in what ways (and understanding *why* those techniques are effective for those items) will lead to better mechanical representations.

For example, let us consider a traditional role-playing game with battles and exploration. Different traditional approaches might be implemented as follows:

A typical turn-based **battle system** involves many small, regular fights over the course of a game. Each encounter features a particular arrangement of foes with varying attributes and skills that give rise to a unique collective meaning based on the context of the combination of enemies and the player's current state. This is similar to the math questions mentioned above and is effective for ideas that require constant, bite-sized testing with problems that can come in a wide variety of configurations.

The overworld could include **embedded environment puzzles**, or perhaps **treasure chests** that require problem-solving to open. These occur less often than the turn-based battles but take more thought and energy to solve. This is appropriate for medium-sized problems that are not configurable or easily randomized.

Various **mini-games** could be implemented that exist alongside the main game. These are fitting for side concepts that are not explored deeply or do not integrate well with the rest of the material, but are still important to understand or give context to the rest of the course content.

Difficult **bosses** or **challenges** can represent "gates" (see Section 3.2.8) traditionally implemented as exams. They are collections of mechanics that embody a deep understanding of key material and can be integrated into other gameplay systems, such as the battle system. Player progression is blocked until the challenge is surmounted, indicating a mastery of content.

Long-term endeavors can be designed as **metagame** mechanics. They are not represented by multiple smaller instances, but rather one large overarching system that can be addressed a piece at a time over a long period. Perhaps at certain points in the game, the player is prompted to apply what skills they have recently acquired to the metagame.

3.2.7 Recompose Mechanics into Structures Adhering to Knowledge Hierarchy

Although somewhat implicit in the previous step, the game's mechanics and systems should be organized according to the knowledge hierarchy. (In fact, this and the next two sections should be kept in mind when constructing mechanics; they could even be considered to be somewhat concurrent, or at least approached with an eye as to how they will lead into later steps of the process.) Mechanical dependencies should align with knowledge hierarchy branches in that low-level mechanics do not depend on skill or knowledge acquired from higher-level mechanics. Game mechanics across the same tiers should be compatible and synergetic.

Mechanics that do not follow these conventions or do not adhere to the knowledge hierarchy should be modified accordingly. It will likely be the case that little change (if any) will be needed, but this process ensures a structured arrangement of mechanics that is capable of effectively communicating the subject matter. This provides a foundation for the remaining parts of the framework.

3.2.8 Explore Design Space through Level Design

With an underlying mechanical framework, game designers can now begin constructing levels, scenarios, challenges, and other game design elements with the purpose of fully exploring the design space. As to exactly how this is accomplished is fairly open and has a great deal of creative leeway, but there are some important points to keep in mind.

Generally, each level (or whatever gameplay structure is used) should incorporate and explore the mechanics of a given tier. They should include means and resources for conveying mechanical meaning and functionality, and adequate practice to explore the range of ideas afforded by their mechanics.

Appropriate tests and metrics should be embedded into the design to ensure a player's skills and progression are in keeping with the knowledge hierarchy. This can be accomplished through the use of "gating" devices that prevent a player from advancing in the game without mastering pre-conditional ideas. A great traditional example of gating is a "boss" that is extremely difficult and requires a high level of skill to overcome, at which point the game will progress. Another approach might include a resource system whereby a player must expend some sort of in-game resource to attempt high-level challenges; failure requires the player to acquire more resources, which can mechanically equate to more practice with the material. A score system might also block a player from continuing if he has not accumulated enough points via practice with the subject matter.

One of the most important points of this process is *iteration*. Game development necessitates playtesting and understanding how players interpret and learn a game's mechanics and systems. If they are unintuitive or not compelling, designers should take a critical approach to determining which elements are at issue and address them. This leads to more playtesting of the revised content, continuing a cycle that should last the entirety of development.

3.2.9 Adapt Narrative, Art, Audio, and Aesthetics

A mechanical framework cannot exist on its own; without adapting aesthetics like art, audio, and narrative to the gameplay, a game is intangible and meaningless. Applying these elements to the mechanics gives them form and shape, representing abstract ideas in a way that players can understand. Thus, before a serious game can be completed, a team of creators must be assembled to craft this content.

From the perspective of a mechanical framework designed to communicate specific ideas through gameplay, this can be seen as the "last" step. However, in practice, aesthetics must be considered very early in the development process and will often be developed in tandem with the mechanics and game design. In fact, the successful communication of certain subject matter can hinge almost entirely on its narrative or audiovisual representation and cannot be made an afterthought. In these cases it will often be common for the mechanics themselves to be created with the purpose of facilitating these aesthetic elements. This was the case with *Mythos Unbound* and many of its gameplay systems, such as its event and NPC systems (see Section 4.2.2).

Beyond simply giving physical form to the game, these artistic elements should be designed to further reinforce a game's ideas. Communicating an item's function to the player will largely depend on its in-game representation, which will include elements like the item's colors and silhouette, camera lighting and framing, audio cues, etc. How a player should approach a problem can also be taught through suggestions in the environment and narrative.

Another key consideration is a game's messaging through the intersection of gameplay, art, and narrative. In an effort to avoid ludonarrative dissonance, a game's procedural rhetoric should align with its aesthetic themes (see Section 2.1.4). In other words, the ideas being communicated by the gameplay should agree with the ideas being communicated by the narrative and art. This is particularly important for subject matter that consists of conceptual or artistic ideas.

3.3 Pre- and Post-testing

In theory, testing for understanding of the material should not be necessary in a properly designed serious game. Effective gating mechanics should ensure that a player cannot complete a game until a desired level of mastery has been achieved; finishing a game should meet the exact same standards as traditional exams in terms of evaluating player skills. However, until serious games and the entire process of gamification (let alone this framework) are verified and accepted as legitimate means of education, certain outside tests will need to be made to ensure players are actually learning the material.

Asking players to take traditional tests (i.e. handwritten or typed questions and answers) before and/or after playing the game is a great way of objectively determining the effectiveness of gamification. The results can be used to chart how much a player improves their skills during their time with the game, and the data can be more easily compared with student performance from traditional courses. *Mythos Unbound* took a similar approach to this through its short response and essay questions placed at the end of each level (see Section 4.2.5), and more formal pre- and post-tests are being planned for future offerings of the game (see Section 6.3).

4. MYTHOS UNBOUND

4.1 Overview

Beginning development in October 2012 and launching in August 2013 for the fall semester, *Mythos Unbound* is an online serious game developed at the University of Arkansas. Offered as part of an online course through Global Campus, the game course is classified as CLST 2323, Roman and Greek Mythology. This class has previously been offered many times in the past in a more traditional, non-game context, but this is the first time it has been presented in a gamified form.

Mythos Unbound is played through the eyes of a Roman slave named Glaucus and follows his story from bondage to freedom. The primary narrative takes place in the House of Octavius Quartio, which is modeled after a real-world location in Pompeii. Entwined with Glaucus' journey are various "myth levels" that depict events from the assigned reading (which includes mythological texts like the *Odyssey*, *Oresteia*, and *Bacchae*). The game is organized in chapters that usually consist of a myth level with accompanying prelude and epilogue house scenes that together share ideas and themes that contribute to the overarching narrative. Each week students are assigned about one or two chapters with followup wax tablet short response questions or essay prompts (see Section 4.2.5). As the game progresses, Glaucus learns about mythology and its importance to Roman culture. The slave thus becomes a vehicle through which students learn about ancient literature in the context of a first century Pompeiian environment, a unique experience not possible in traditional, non-interactive learning methods.

The author was one of the first developers to be brought onto the team and has been with the project from its inception to its release. It is from his experiences and the lessons learned by the team that the framework described in this thesis was engineered. The game also serves as a prime example of the effectiveness of the framework and the ideas it suggests. Thus, an understanding of *Mythos Unbound* is essential in order to analyze the framework in Chapter 5.

Figure 14: A scene within the House of Octavius Quartio in *Mythos Unbound* [72]

4.2 Technology

A number of technologies were needed to deliver educational content and tell stories in *Mythos Unbound*. The following were all developed entirely or in large part due to the efforts of the author.

4.2.1 Character Controllers

One of the most important aspects of a compelling, immersive game like *Mythos Unbound* is how well the player can control his avatar. Behind what seems like a natural, simple, and intuitive interface for the player is a complex system involving camera control, collision detection, physics, and more. Even more challenging is the fact that nearly every level incorporates a different style of gameplay that often calls for unique implementation of the base character controller code. The team accomplished this task through a combination of adapting several of Unity's base scripts, researching common game industry techniques, and applying a little of its own ingenuity.

4.2.1.1 Input

In order to best implement its character controllers, the team needed to first decide on a common input scheme that would unify the various controllers. This would reduce complexity by requiring players to learn only one basic control scheme. A common means of input also gave the engineers a foundation from which to construct the code by outlining some base assumptions.

The player views the game world through the perspective of an in-game "camera" that can often be moved and manipulated by the player. In the levels where the player can control the camera, the team defined two means of movement: rotating the camera around the player at a certain radius, and zooming the camera in or out by changing that radius. Rotating the camera around the player would be accomplished via the mouse, with left and right mouse movement affecting the camera's position around the player in the XZ plane and up and down mouse movement affecting the height of the camera. Zooming would be performed by scrolling the mouse wheel up or down. These camera controls would be suitable for third-person levels, but the house scenes that took place in the House of Octavius Quartio (see Section 4.3.1) were in first-person and required a slightly different approach; the camera would not zoom in or out, but mouse movement would still "rotate" the camera by rotating the player avatar as well.

Like most computer games, we opted for a WASD movement scheme (with the arrow keys often providing equivalent functionality). Movement can be envisioned as occurring on two axes, the X and Z directions. (The Y, or vertical, direction was relegated to mechanics like "jumping" that would be activated through other key presses; the player never directly moves in the Y direction.) The Z axis is controlled by the W (forward) and S (backward) keys, while X axis motion is controlled by A (left) and D (right). At a high-level, the team said that the Z axis should control "forward" movement while the X axis would be "lateral" movement. However, those descriptions are relative; "forward" does not indicate whether it is from the camera's perspective or the avatar's orientation. Our testing showed that the most intuitive implementation happened to be using the camera's frame of reference.

4.2.1.2 Movement Vector

Once the raw axis input data is received from the player, a "movement vector" needs to be calculated according to the above input conventions. The movement vector represents the character controller's direction and magnitude of movement for a single update of the main game loop.

Since movement needs to be calculated relative to the camera and *not* the player avatar, the first step is to grab the camera's current rotation. Rotation around the X and Z axes is ignored since the player avatar will only move in the XZ plane, meaning only the Y axis rotation matters. Since most computer keyboards are not analogue, only digital inputs of -1, 0, and 1 are processed in each of the X and Z directions according to WASD input, with positive values representing “forward” and “right” and negative values signifying “back” and “left.” Based on the input, vectors in the X and Z direction are constructed relative to the camera’s rotation and then added together to create a single vector. This vector is then normalized, or else diagonal movement will result in an larger vector magnitude.

Depending on the angle, the movement vector is then multiplied by a defined speed value. There are speed values for forward, side, and backward movement that can be adjusted by designers. What this means is that a character could move at a quick forward speed, but moving backwards could result in a much slower speed. Since speed values are defined in units per second, the vector is also multiplied by the current time slice.

Up to this point, movement in the Y direction has been ignored. The Y axis represents “gravity” and is tracked through a separate vertical velocity variable that has a downward acceleration applied to it. The velocity is usually set to a positive value as a result of certain actions (such as jumping) and is reset whenever the character controller touches the ground. In order to curb excessive or uncontrollable falling speeds, there is also a terminal velocity value that the vertical velocity cannot exceed. Once the vertical velocity has been calculated for the current update cycle, it is added to the movement vector.

The movement vector is then transformed into the player avatar’s local space. It now defines both the direction and magnitude the character should move relative to the character itself.

4.2.1.3 Character Movement

With a movement vector in hand, it is now possible to move a character. This is performed with a *collider*, which is a sort of invisible region of a particular shape (such as *box* or *sphere*) that is used with Unity's intersection algorithms to detect collisions. Unity provides a special type of collider conveniently named as *Character Controller*, which is designed to model character movement and simplify certain physics calculations.

Using a capsule-shaped collider, these controllers provide a variety of information. A set of collision flags allow developers to detect when another collider has collided with the controller, and from which direction (such as above, below, to the side, etc.). This is useful for determining when a character has run into, say, a wall as opposed to the floor that it is constantly colliding with as a result of walking. In fact, detecting collisions with the ground is how the vertical velocity value is reset during the calculation of the movement vector. The controller also provides exposed variables for affecting what degree of slopes a character can walk up or down before the slope is considered a wall or cliff.

However, the most valuable function for the purposes of character movement is `CharacterController.Move()`, which moves the controller in a given movement vector (like the one calculated and described in the previous section). The advantage of using this method over simply displacing the character's position manually is that Unity's Character Controller will constrain its movement based on collisions. What this means is that a high-magnitude movement velocity that would have otherwise placed the character in the middle of a wall will instead place the character just before the wall at the point of collision. This collision-detection is all done behind the scenes through Unity's physics engine.

4.2.1.4 Camera Movement

Camera movement is somewhat different to how the character's movement vector is calculated. Given a focus target and the mouse input described in Section 4.2.1.1, a "destination position" and "destination rotation" will be calculated and the camera will interpolate to those values over a short time. The X axis mouse movement will rotate the camera along its Y axis, while Y axis mouse movement will rotate the camera along its X axis. These values provide the destination rotation. Separate from these values, a "distance" value is tracked that can be increased or decreased from zooming via the mouse wheel. To calculate the destination position, the destination rotation's forward vector is multiplied by the distance value and subtracted from the focus target's position. The exact functionality of the camera can be manipulated by a variety of variables that affect interpolation speed, min and max distances and rotations, and more.

One issue the team struggled with was character occlusion. It was often possible for world geometry (such as a wall or a large rock) to get between the camera and the avatar, blocking the player's line of sight. This made controlling a character difficult and unintuitive. A number of solutions were developed depending on the application, but the primary one was using ray casting. Each update, a ray would be cast from the player's position to the camera's destination position. If the cast resulted in a collision before reaching the camera, the camera's actual destination position would be at the point of the ray collision, ensuring no other objects would be blocking the camera's view to the avatar. If the obscuring object were later removed, the camera would then bounce back to its original destination position.

In practice, there were a number of issues with this approach to character occlusion. For one, it resulted in jittery camera behavior when the ray cast would hit uneven geometry. Another problem occurred when the ray cast itself did not hit any geometry, but the camera view was still obscured by objects in the corners of the viewport. The team developed a number of workarounds for each situation as it came up (for example, casting rays at each of the four corners of the camera viewport in the latter issue), but it was still largely a case-by-case situation. Ultimately, most levels called for their own specific implementations.

Many camera issues were avoided altogether in the house scenes due to the necessity of first-person gameplay. Since the camera is always located at the player's eyes (and thus position), there are no issues with character occlusion. We implemented the first-person camera as a child object of the player character controller, thus ensuring it maintained a position and rotation relative to the avatar. As a result of this approach, the camera did not need a distance value for zooming, and mouse X movement was ignored for the purposes of camera movement (lateral movement was instead handled by the character controller turning left and right).

4.2.1.5 Special Considerations

While the above describes the basic functionality of all of *Mythos Unbound*'s character controllers, each level required its own customizations. Some examples are listed below.

Many events and gameplay mechanics necessitated a "lock" function that would prevent the player from being able to move his avatar, the camera, or both.

Cutscenes often had the camera being decoupled from the player avatar in order to show key events. Sometimes this was accomplished through moving the actual player camera, and other times special cutscene cameras were turned on and off when called. Most of this was handled through the workflow system (see Section 4.2.2).

Uneven terrain often resulted in character controllers oscillating between "grounded" and "not grounded" states depending on how their colliders touched the floor. This resulted in a number of complications, such as unresponsive jumps if the player tried to jump just as the controller thought it was in the air for a brief moment. The solution was a buffer that provided a short window during which the controller was considered grounded even if collision detection reported it was off the ground. This buffer was reset every time a "grounded" collision was reported.

Some levels needed special movement options. These ranged from "double jumps" (being able to jump once more in the air) to "air dashes" (being able to move through the air horizontally at a high speed for a short time) and "run" mechanics (being able to move at a higher speed than normal). All of these required specialized input and affected the movement vector in their own ways.

Enemies would often have abilities to knock back or stun players. Methods were made for enemies to tell character controllers the direction and magnitude of impact, during which time input was locked.

Sometimes movement vectors eschewed normal calculation and instead used other algorithms, like *Bacchae*'s flocking algorithms for the maenads that follow Dionysus (see Section 4.3.13).

4.2.2 Events and Workflows

The narrative-heavy nature of *Mythos Unbound* called for the inclusion of many events. By "event," I mean a specific sequence of mechanics that are triggered by specific conditions.

An event might be as simple as listening to an NPC introduce himself, or it might be as complex as coordinating a complex cinematic dinner scene with timed dialogue, moving servants, and NPC interactions. It was common for there to be anywhere from ten to twenty events written into each week's level offering. Clearly, it was not feasible to hand code each event individually for the entire project, so the team sought a more generalized approach.

Initially, the team relied on third-party products and packages. One of the earliest tools we tried was an NPC dialogue editor called *Unity Dialogue Engine* [73]. It was used for about a month before it was agreed that it was inadequate for the purposes of the project. The editor was unwieldy and prone to bugs, and writers and designers had to struggle with learning a very specific writing format in order for scripts to be understood by the editor. Furthermore, it was only suited for NPC conversation trees, i.e. series of strings that would occasionally branch or skip to other strings based on certain variables and flags. However, the team needed a system that went beyond simply verbal communication; the script called for characters to move, to interact with their environment, to pick up items, etc.

What the team found was it needed a way to organize a series of small, well-defined steps in a larger workflow. This realization led to *Okashi* [74], a third-party Unity framework for the construction of RPGs, or role-playing games. For many months this software package met the team's needs, and in fact several of the final levels still contain remnants of its code. It provided an interface for constructing workflow events from a selection of pre-defined steps. These steps included functionality like "playing a sound" or "moving to waypoint." A designer would arrange a series of steps in order and assign them values, such as the audio file to play or the speed at which a character would move to a waypoint. During gameplay, these events would be activated through various means such as stepping into a trigger space, and they would play until they were resolved.

Unfortunately, even *Okashi* proved to fall short of the team's needs. It was simply too rigid, making too many assumptions about the nature of the game. It was designed for creating traditional console RPGs in the vein of *Final Fantasy* or *Dragon Quest*, so its pre-defined steps were designed to recreate those sorts of experiences. There was no step to display a subtitle, or to have the player look at an object of interest, or to mend a broken vase; the team's specific needs were not included in the framework's steps, nor could they have been. In addition, it was impossible to write custom steps, and implementing the team's desired functionality required complex workarounds that were limited, prone to error, and difficult to trace and debug.

Around summer 2013, the team decided that the only way it could realize its vision is if it built its own event system to construct workflows that met its particular needs. Thus, the author began work on building a workflow system with three goals in mind:

- Workflows must be able to chain together any arbitrary series of steps.
- The system must be flexible and easy to extend by writing new steps.
- The steps and interface must be intuitive and meet the needs of designers.

The third point is particularly important; there's no point in building powerful tools if no one is going to use them. By communicating and collaborating with various members of the team, the author was able to anticipate their needs and design the system in such a way that designers and writers felt it was logical and matched how they were internally representing the workflows in their minds. This greatly improved productivity and drastically reduced the amount of support technical support.

The system is actually fairly simple in theory. It consists of two key classes: `Step` and `StepManager`. `Step` is an abstract class with three key methods: `StartStep()`, `UpdateStep()`, and `FinishStep()`. `StartStep()` should initialize a step by resetting variables, obtaining references, etc. `UpdateStep()` should have the step update itself for that time slice, returning `true` if it has completed its objective. For example, a `MoveStep` might finish when the object it is moving has reached a given waypoint. Finally, `FinishStep()` should set all relevant variables and objects to a final exiting state, the state they would be in when the step had completed its objective. In practice, this last method was rarely called seeing as most steps would have already accomplished this during the last iteration of `UpdateStep()`, but it was included in the class's definition in case the team should ever find a need for it. (And, in fact, it appears the team might revisit `FinishStep()` as a means of debugging: designers often find themselves playing the same workflow over and over to test a specific step near the end of that workflow. This is a time-consuming process, but implementing a means to skip steps would be possible by calling `FinishStep()` for each skipped step. By immediately resolving a step and setting all of its associated values to their final states, it would be no different than had the step naturally executed and accomplished its objective via `UpdateStep()`. This would ensure subsequent steps could continue their execution and assume that prior steps had "finished" appropriately.)

The idea behind the `Step` class is to derive other steps from it. Each derived class should represent a single "step" in a larger workflow, similar to the pre-defined steps in *Okashi*. Each step had to be built from scratch, defining some sort of functionality in the context of the game's mechanics and systems. What constituted a "step" depended on what functions would be most reused and what was intuitive and easy for non-technical writers and designers to use. For example, a common step was `DialogueStep`, which originally simply played a sound file with a line of dialogue. However, a `DialogueStep` was often preceded or followed by other steps, such as a `RotateStep` to turn the speaking NPC to the player, or an `EmoteStep` to have the NPC play a specific emote animation. Rather than require designers to repeatedly implement these same step sequences, the functionality of steps like `RotateStep` and `EmoteStep` were rolled into `DialogueStep`. If a designer specified a `rotateTarget` or an `emote`, then the step would smartly perform the corresponding functions. If those values were left null, the step would simply behave as it always had.

Governing the execution of workflows is the `StepManager` class. The `Play()` and `Stop()` methods manage the current `Step` object by continually calling its `UpdateStep()` method until it returns true, indicating it has finished. `StepManager` then initializes the next step in the workflow as defined by the current step's `nextStep` variable. Like links in a chain, a workflow sequentially executes each step one by one until there are none left.

Across the game's many levels and scenes, dozens and dozens of steps were developed. Once the team had shifted over to its own custom code for handling events and workflows, it never hit any hard walls that this approach simply couldn't handle. However, because the levels were fairly linear, some of the shortcomings of this system were never encountered and are functions that could be added in the future. While the game would commonly have multiple workflows running at the same time, "splitting" a single workflow into two or more concurrent threads was not possible, nor was there a means of “joining” multiple workflows so that execution couldn’t continue until two or more threads had reached the same step. The team would have liked to also implement a means of constructing high-level workflows from multiple smaller low-level workflows, or to have workflows be constructed dynamically in-game based on given input or game logic. Instead, all workflows had to be built by hand in the Unity editor. And, extending from that, the system would have been even easier to use had an elegant and intuitive GUI editor been developed, but unfortunately that was beyond the team’s limited time and resources. There will hopefully be opportunities to iterate on this workflow approach in future projects.

4.2.3 Animations

Unity’s Mecanim system provides an approach to model animation using finite-state machines. Each state represents a specific animation or group of related, blended animations (such as “walking” and “running”). Transitions between the states can be defined using boolean conditionals that are defined by Mecanim-specific variables that can be manipulated via scripting. Each finite-state machine is defined in its own *Animator*; typically, each animating object will receive its own animator, although some animators can be reused among objects with the same underlying models and animation logic.

For many of the animators, the corresponding code and logic was custom-made. This was common in most of the myth levels, particularly the ones with gameplay unique to that level. Because the mechanics between these levels were so radically different, it was not practical to attempt a generic system that could be applied to all of them.

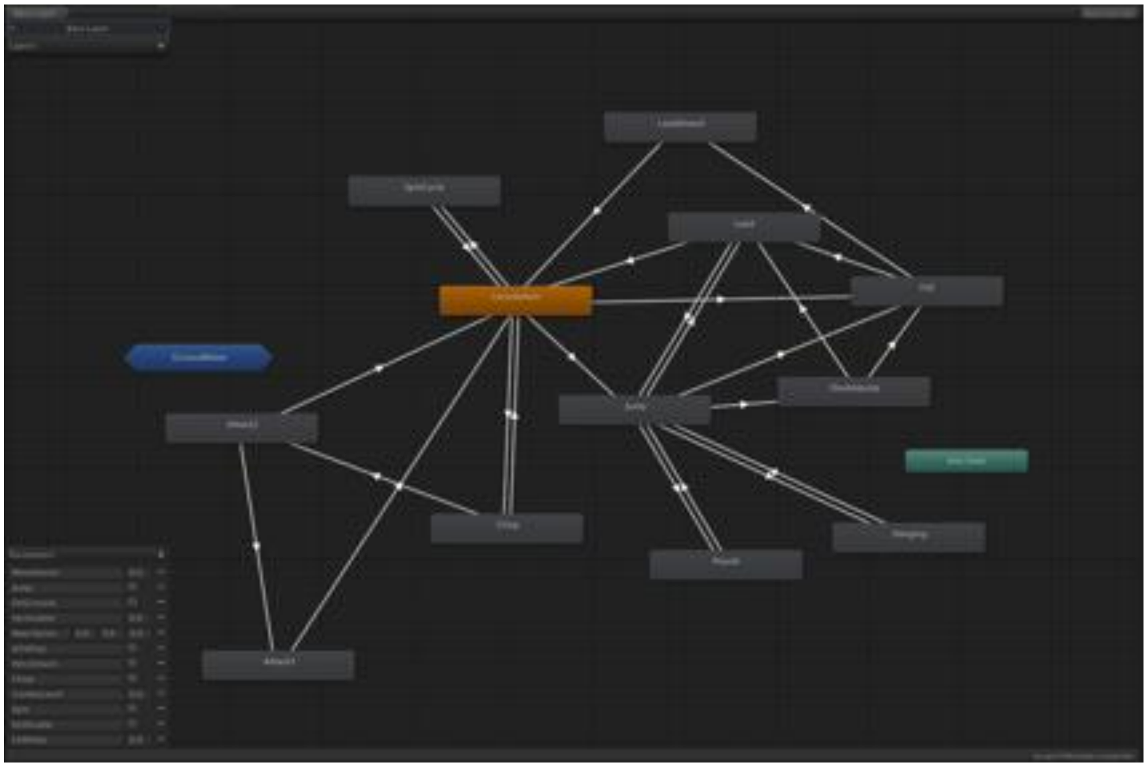


Figure 15: Hercules' custom animator, built specifically for the Hercules model [75]

However, for the house scenes, there were far too many characters to make custom scripts for each of them. In addition, since those scenes were so largely dependent on narrative and workflow events, a common solution needed to be developed that could be reused and made compatible with the various event steps.

A `MythosAnimator` class was developed specifically to interface with cutscenes in a generic way. For movement, a `MoveDelta` variable was updated with the magnitude of the difference between the current and last positions of the character. This was used to trigger transitions to "locomotion" states once it surpassed a certain threshold. Some animators also used this data to determine when to blend between "walk" and "run" states depending on how large the delta value was. In a similar manner, a `RotateDelta` variable tracked the character's rotation between frames, giving a negative value for left turns and a positive value for right turns. This was used for "rotating" animations. In the case of concurrent rotation and movement, movement animations generally took precedence.

The most important feature of `MythosAnimator`, however, was its emote system. A common need was to have characters "emote," or play a one-shot animation such as pointing or waving. These animations weren't dependent on motion, so a means of explicitly calling them was necessary. The solution was an enumeration of all possible emotes that would be used in conjunction with an `EmoteState` animator variable that could be modified through certain workflow steps, such as `EmoteStep` and `DialogueStep`. An array of emote states could be set up in the animator, each with a transition corresponding to a specific integer in the emote enumeration. These transitions originate from an "Any State" state, which means they could be activated at any time regardless of the animator's current state. Upon finishing their animations, the emote steps transition back to a default state, such as "Idle."

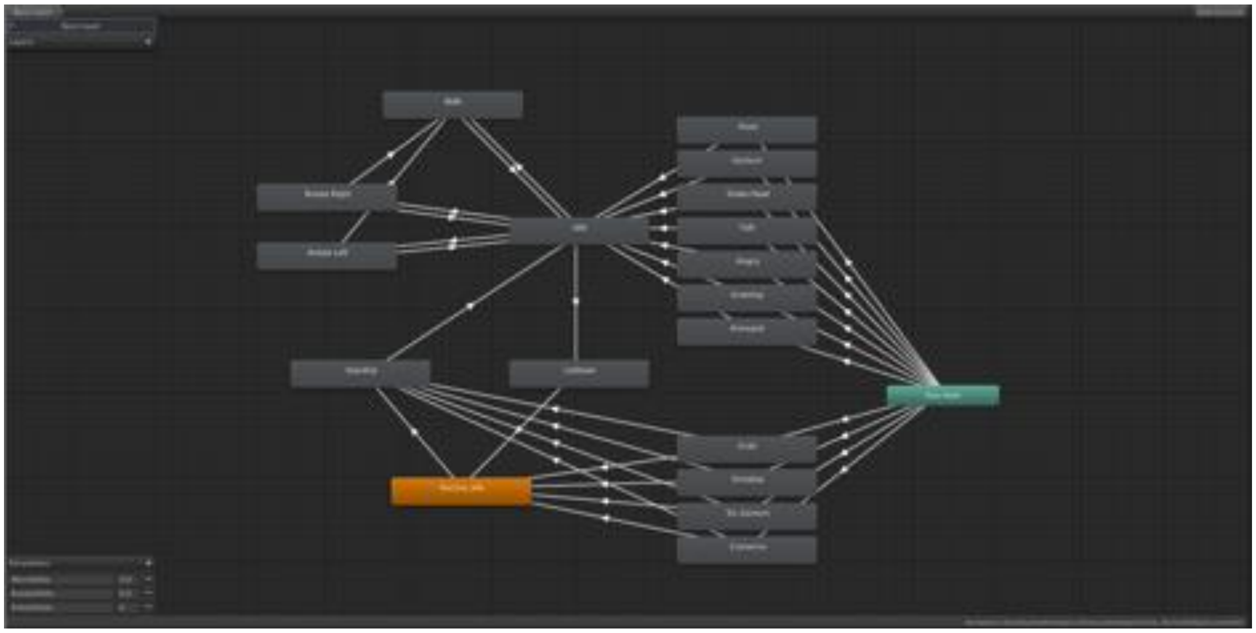


Figure 16: Octavius' animator, made compatible with `MythosAnimator` [76]

While the character animators themselves were sometimes modified for specific needs, `MythosAnimator` ensured that the actual code would always remain consistent among characters in the house scenes. This was convenient for designers since all basic movement animations were handled automatically through code; animations like "walk" and "rotate" did not need to be called explicitly, unlike in prior systems the team had used. The process of calling emote animations was also simplified since the emote enumeration allowed the workflow system steps to include drop-down menus of all available emotes. Designers could simply select the name of the appropriate emote, which would be converted to an integer and submitted to the animator behind the scenes.

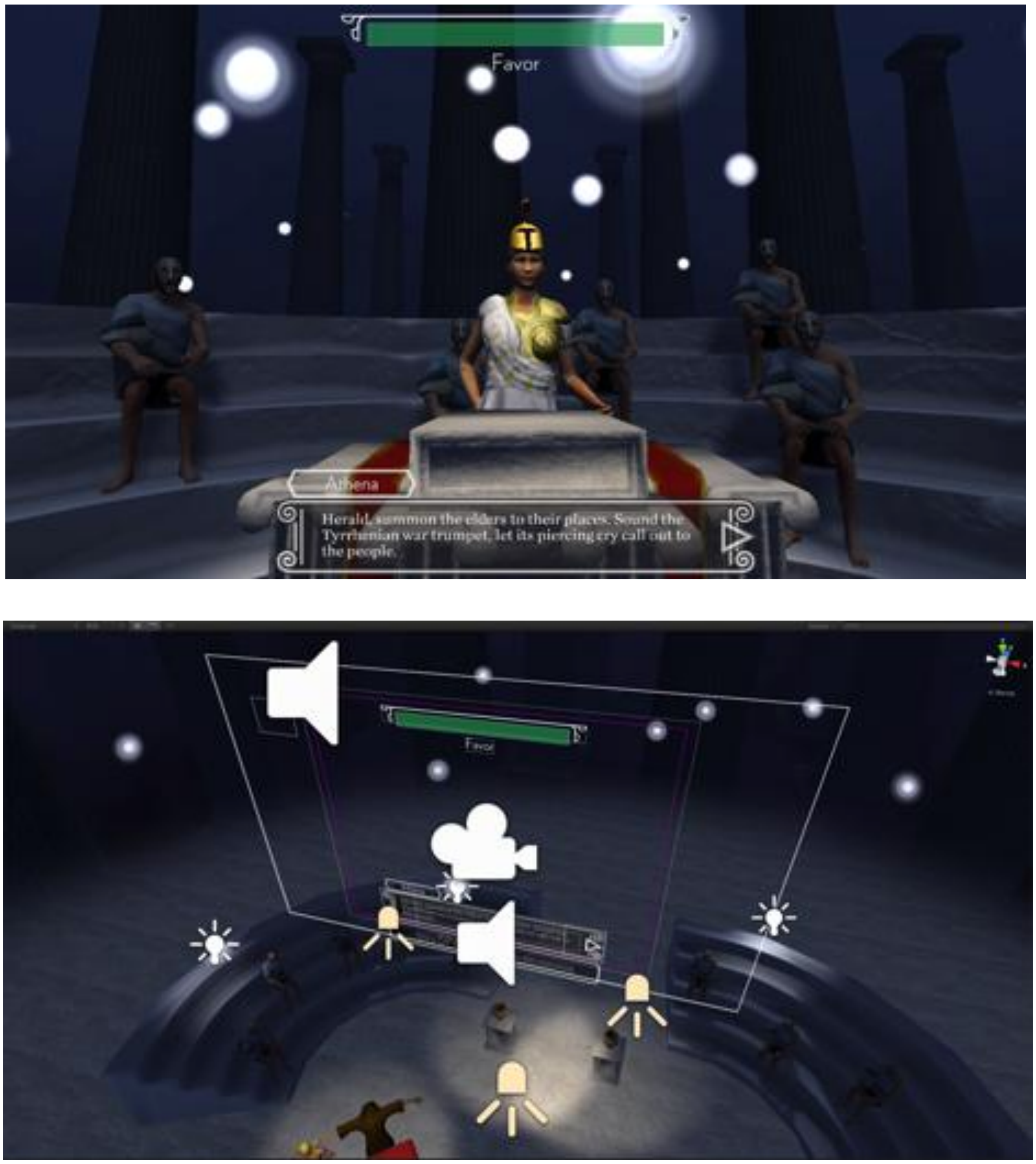
4.2.4 GUI

A graphical user interface, or GUI, was necessary for nearly every scene of *Mythos Unbound*. Unity provides an `OnGUI()` function that will draw 2D graphics and text to the screen. However, it is not ideal since it is actually somewhat unoptimized, has no respect for depth (i.e. GUI elements are drawn over everything and in the order they were drawn), and is difficult to visualize since it is constructed all in code and does not use anchors.

The team sought a third-party solution in the form of NGUI [77]. NGUI provides tools and scripts to construct various GUI elements in the actual 3D scene view. This allows designers to build the GUI more intuitively by being able to see how the items are laid out rather than relying on abstract descriptions like "two hundred pixels to the right of the screen width divided by three." The 3D nature of the GUI also allows items to be easily moved forward or backward to change the order in which they are drawn, even during gameplay. The inclusion of sprite features like 9-slicing and radial fill gave artists more flexibility while increasing efficiency. NGUI also provides anchor functionality, allowing GUI elements to be arranging relative to positions like "lower-right corner," allowing the GUI to easily scale to different screen resolutions.

Since NGUI items exist in the actual scene as 3D objects, they need their own specific camera. The "GUI camera" is set to be orthographic so that the images will be seen as pixel-perfect 2D graphics overlaid onto the screen. The camera and the rest of the GUI are set on their own custom "GUI" layer, which normal cameras are set to exclude from rendering. (This is so that players will not see the GUI floating in the middle of the 3D game world.) Meanwhile, the

GUI camera is set to ignore everything but the "GUI" layer, ensuring that it does not draw any scene elements into the GUI.



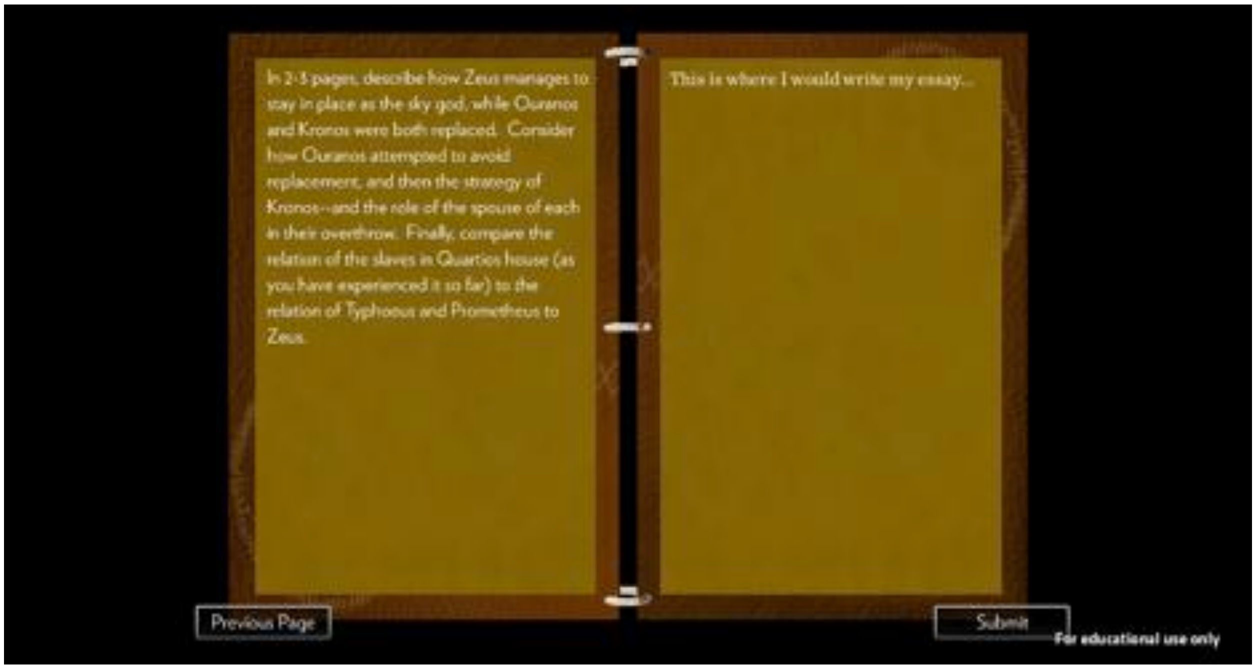
Herald, summon the elders to their places. Sound the Tyrrhenian war trumpet, let its piercing cry call out to the people.

Figure 17: An in-game GUI (top) [78] with its corresponding 3D scene view (bottom) [79]

4.2.5 Wax Tablet

Some knowledge can't be tested solely through traditional gameplay mechanics. Writing is an important skill that is practiced often in past offerings of the course, and this version is no different. The team wanted a way to have students submit short responses and essays related to both the literature as well as the game, but breaking immersion with the game was a concern. Instead of separating the game and the writing content across two separate means of delivery and submission, the team wanted to create an elegant in-game means of submitting content to the professor. The solution was the "wax tablet" system.

A sheet of wood covered with a layer of wax, the wax tablet was an actual writing tool used in ancient Rome. A stylus could be used to engrave words into the wax. They were portable and could be reused by heating the wax and smoothing it out. This provided a perfect backdrop for an in-game text editor which players could use to write answers to question prompts. By hooking this up to a GUI and a server that receives student submissions, the game is able to test students' understanding and writing skills while providing context that makes sense in the game world. Currently, the wax tablet is primarily used in post-level evaluations that ask players questions related to the levels they have just finished. Students are commonly asked to draw parallels between the literature and events in the myth and house levels, particularly in terms of narrative, theme, symbolism, and other literary elements. After submitting a tablet, the student receives an email confirming the submission along with the contents of the submitted wax tablet.


In 2-3 pages, describe how Zeus manages to stay in place as the sky god, while Ouranos and Kronos were both replaced. Consider how Ouranos attempted to avoid replacement, and then the strategy of Kronos--and the role of the spouse of each in their overthrow. Finally, compare the relation of the slaves in Quartios house (as you have experienced it so far) to the relation of Typhoeus and Prometheus to Zeus.

This is where I would write my essay...

Figure 18: The wax tablet with a question (left) and the start of an answer (right) [80]

4.2.6 Web Streaming

One of Unity's strongest features is its ability to export web players. By installing a small plugin, users can play Unity games through their Internet browsers. The course is hosted on Blackboard (an online learning management system), which can be communicated with using the Sharable Content Object Reference Model (SCORM) Web communication standard that was developed by the Advanced Distributed Learning (ADL) Initiative [81]. ADL produced a package of Unity scripts that enabled Unity web players to communicate with SCORM platforms such as Blackboard [82]. This means Blackboard can receive data from Unity games for the purposes of grades and feedback. For an online educational course, this is incredibly useful since it allows users to access all of the course material in one place from most internet-capable platforms.

Unity's default web player includes a standard streaming option that allows content to be streamed as the user plays the game. Proper use of this feature means that users don't need to wait for the entire game to load before playing, shortening wait times and increasing accessibility. This also gives developers more leeway to add higher-quality assets to the game (such as voice acting and higher resolution art) that would otherwise bloat the initial load time.

Unfortunately, the default streaming is not very customizable. Unity levels are arranged into "scenes," or collections of instantiated game objects. Unity will load the initial scene and then begin streaming the following scenes in a preset order one scene at a time. A scene cannot be played until it is fully loaded. This is limiting for larger scenes where the ideal solution would be to have the player start in a partially loaded scene that will stream in the other scene elements as they become necessary. While this isn't possible through standard web streaming, Unity offers a means of streaming additional assets through AssetBundles.

An *AssetBundle* is a collection of assets that can be exported from Unity into a single package. This package can be hosted on a server that Unity player clients can access. By streaming and extracting assets from AssetBundles during gameplay, new content can be added to the scene dynamically. However, this requires custom code to manage the AssetBundles.

4.2.6.1 AssetStreamer

The team's solution was the use of AssetStreamers, which were developed by the author. The idea is that scenes should be divided into a number of logical streaming units based on level design and asset size and then packaged into AssetBundles that would stream in over the course of gameplay.

Determining the streaming units was done on a case-by-case basis and required iteration to decide how best to group the units and in what order to stream them. The interactive, non-linear nature of gameplay necessitated that each scene be given a unique implementation since players can potentially walk into or see any environment that is accessible or visible, and the player's actions might prioritize streaming one unit over another. For example, if the player is walking up a flight of stairs, the upper floor would need to be streamed before, say, the garden. What actions the player can take and where he can go depends on the narrative and environment design, so constant communication between writers, designers, artists, and programmers was a necessity, particularly when technical limitations required alterations in the script or level. Sometimes a pair of cleverly placed visitors would block a certain path or room, or certain textures would have their resolutions lowered if they could only be seen from a distance in a particular scene.

Through a custom editor script, developers could select a group of scene objects representing a logical streaming unit and create an AssetStreamer from them. If one does not already exist, the AssetStreamer creates a Unity prefab from the objects. A *prefab* is a reusable object that is treated like an asset and can have multiple instances within a scene. For example, a door object, with all of its associated art assets (door model, wood textures) and code ("open" script, colliders), could be turned into a prefab that is regularly placed around a scene. The editor script then scours the scene for any other instances of the prefab, keeping track of their positions and rotations.

Developers continue this until they have an ordered list of AssetStreamers, all accessible from the editor. They can then be exported as AssetBundles and uploaded to a server. Then, the

URL addresses of the AssetBundles are copied and pasted into their respective AssetStreamers in the editor. When the scene is finished, the editor has an option to delete AssetStreamer prefabs in the scene. This is necessary because Unity will consider any scene objects as dependencies that must be included in the build, so they need to be removed from the scene since they will be streamed in later.

When the first AssetStreamer is created, an AssetStreamerManager is also instantiated. The manager will keep track of every AssetStreamer that is constructed. During gameplay, the AssetStreamerManager will begin streaming the AssetStreamer AssetBundles in order. As soon as an AssetBundle is loaded, its owning AssetStreamer will instantiate its prefab at the position and location of all of the previously recorded prefab instances. The manager will eventually finish, and the final scene will then be constructed and complete.

One important feature of the AssetStreamerManager is its ability to dynamically prioritize AssetStreamers based on gameplay input. Game objects can call `StreamAssets()` on the AssetStreamerManager to have it prioritize a specific AssetStreamer based on a given name string or index integer. The next time the manager needs to stream an AssetBundle, it will first consult the priority list before returning to the default list. So, extending from the previous example, if the player starts walking up the stairs, a collider trigger might be positioned at the foot of the staircase that will inform the AssetStreamerManager that it needs to start streaming the upper floor at the next possible opportunity. Until then, the door to the upper floor can remain “locked” until it receives a message from the manager that the upper floor has finished streaming, thereby unlocking the door.

4.2.6.2 Occlusion Culling

One of the major consequences of a streaming approach is that the game cannot rely on Unity's built-in occlusion culling. Normally, the Unity engine is smart enough to detect when the player cannot see certain geometry due to occlusion, and will thus not spend resources rendering it. This is called "occlusion culling." However, by streaming in world geometry pieces at a time, Unity has no means of constructing the data structures it needs to determine what to occlude; this sort of geometry needs to be static and pre-existing at the start of the scene. Unfortunately, without occlusion culling, frame rates will rapidly drop for users with weaker machines due to the large amounts of overdraw. This was largely an issue in the House of Octavius Quartio, which had many large textures and complex geometry.

The team had to develop its own occlusion culling solution in the house scenes to maintain efficiency. This was accomplished by segmenting the architecture into distinct rooms separated by doors. Each room had associated with it an array of objects. If a room's doors were closed and the player was not in it, every object owned by that room was turned off. If a room's doors were open (thus opening lines of sight in adjoining rooms) or the player was in the room, that room's objects were turned on. This was not true occlusion culling, but it was effective in increasing frame rates for the machines we tested it on.

4.2.6.3 Streaming Issues

The AssetStreamer code is fully functional and the team had a demo available in June 2013 that featured a streaming version of the House of Octavius Quartio. Unfortunately, the streaming tech was not integrated into the final builds for the Fall 2013 semester offering of the game due to a number of reasons.

While ADL provided a set of Unity scripts that enabled SCORM communication, they were written for Unity 3.x. In November 2012, Unity was updated to 4.x, at which point the SCORM package was no longer compatible. The team communicated the issue to ADL (the developers of the SCORM-Unity package), but a solution was not developed in time for the fall semester. Thus, the team could not offer Blackboard integration for the initial run of the course. Since web streaming through Blackboard was one of the primary reasons for developing the AssetStreamer tech in the first place, it was not quite so imperative that the team spend limited resources on implementing the AssetStreamer tech into its levels.

Another problem was the way the team had arranged the project development structure. Every level was contained in its own Unity project, separate from the house scenes. The intent was that each team could work more easily and independently by not having to depend on one another and manage a single, larger project that would carry a heavy file size and require synchronization (which was difficult at the time since the team lacked the licenses for version control through Unity's Asset Server software). However, this created major conflicts when trying to bookend the house scenes to their respective levels (see Section 4.3.1) in their final builds. For example, different projects would have the same names for different source code files (such as "Helper.cs"). This made streaming assets impossible unless the team spent a large amount of resources resolving these conflicts at a time when many levels were still not finished. The team could not justify allocating its limited engineers to these tasks, so we instead opted to cut streaming between scenes and instead offer separate builds for each house scene and myth level.

But the largest problem was simply that customized web streaming through AssetStreamers is an enormously time-consuming process as described in Section 4.2.1. It requires a lot of extra planning and changing of assets and design elements. The process cannot begin until the scene is finalized, which was often too close to a deadline to integrate AssetStreamers. Streaming also requires a lot of iteration and lengthy waiting times while AssetBundles are being exported and uploaded to servers. There was simply a lack of time and resources, and when the choice came down to either having streaming or having more robust gameplay experiences, the team always chose the latter.

While the AssetStreamer tech has been shelved for the first run of the course, the team intends to revisit it in future semesters. Now that the levels and scenes are nearly finalized, it will be more feasible to spend time optimizing them for web streaming in the future.

4.2.6.4 Future Improvements

While the AssetStreamers are functional, there are many ways they could be improved upon. There are a few features the team would like to introduce when the opportunity presents itself.

The current implementation relies on prefabs, but it is not necessarily the prefabs that need to be streamed. Rather, it is the actual art assets themselves, primarily the textures and the audio files. Instead of identifying a streaming unit as a single prefab, it would be better if the code was able to smartly detect all art dependencies within a streaming unit, compare them with those of other streaming units, and export AssetBundles based on smartly organized art asset dependencies and not the streaming units themselves. This approach would reduce a lot of redundancy caused by multiple streaming units sharing the same art asset across different prefabs since each prefab would spend time loading the shared art asset.

Streaming content based on LOD (level of detail) would allow players to get into the game sooner at the cost of slightly larger overall download sizes. The idea is that an initial wave of low-poly, low-resolution assets would be streamed and instantiated first, providing a fully-functional (if slightly ugly) game environment. As time goes on, higher LOD content would stream in to replace the lower LOD assets.

The team would also like to make streaming tech better integrated with its other software. Currently, no code outside of the AssetStreamer system accounts for streaming, but it would be better if relevant scripts were optimized to work with the asset streaming. For example, voice-acting is managed by audio players that assume all audio files will be available to play at any time. If these audio players were intelligent enough to just display subtitles if the speaker has not had its audio files streamed in yet, then players could start the game sooner while dialogue clips stream in.

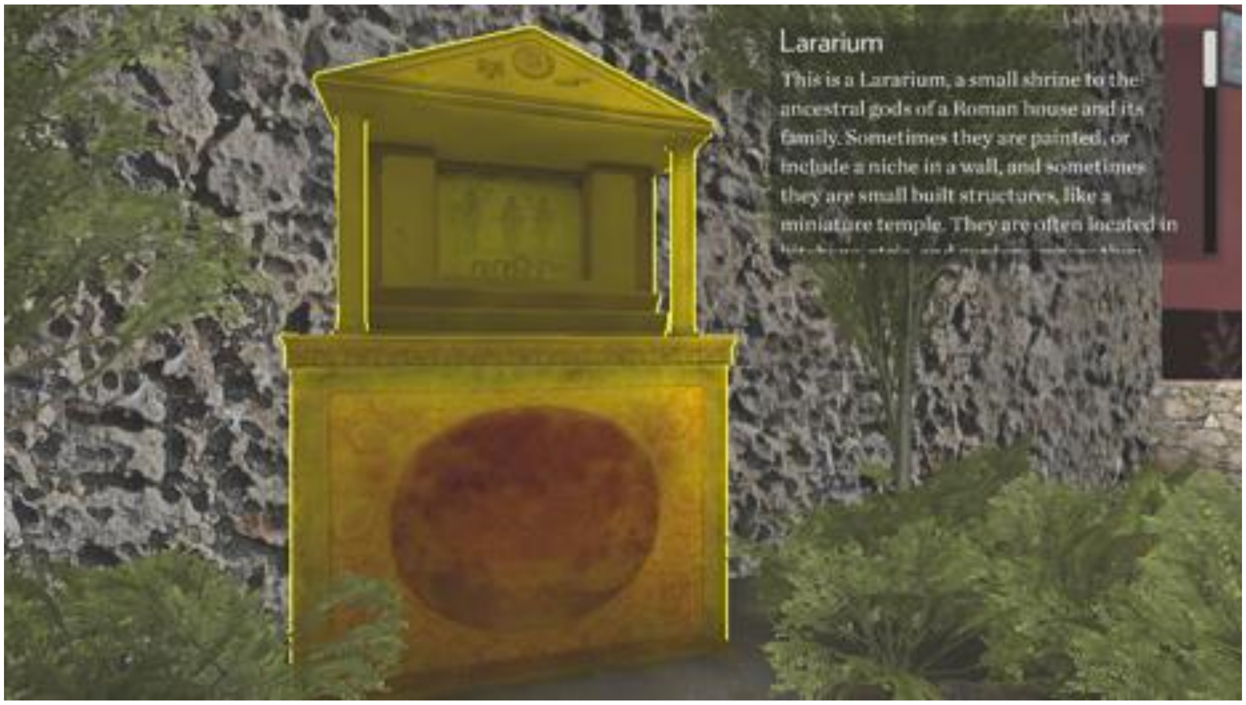
4.2.7 Inspection Mode

Mythos Unbound contains many historically accurate elements in its environments, from looms and urns to graffiti on the walls. Unfortunately, the nature of the gameplay and narrative does not offer many opportunities to acknowledge these items over the course of the game. The team wanted to create a mechanic by which players could explore and learn about these environmental elements even if they were not directly tied to progression.

Inspiration came from the sci-fi game *Metroid Prime*, which featured a “Scan Visor” that granted players vision of contextual icons and silhouettes that could be “scanned” to acquire additional information about the environment. We adopted this mechanic into *Mythos Unbound*, referring to it as “Inspection Mode.”

Through a simple toggle button, players can see the world in a different light when in Inspection Mode. Clicking on highlighted items results in a small window appearing in the corner of the screen, providing the name of the object and a short description. This is done by attaching a script to colliders that flags them as being inspectable. If Inspection Mode is active and the player clicks, a ray is cast that will return the first inspectable item it hits. That item’s information is then displayed in a GUI window. The visual effects of Inspection Mode include shaders and a subtle filter over the camera that give inspectable items a colored silhouette.

Unfortunately, while Inspection Mode was featured in the June 2013 streaming web demo, it was not implemented in the initial run of the course. There were no major technical issues, the team just didn’t have the resources to go back through its many scenes and add the scripts, colliders, and texts needed for Inspection Mode, not when the mechanic was unnecessary for progression. Like web streaming, this is another feature that will be revisited in future versions of the course after its base content has been completed and is fully functional.


Lararium

This is a Lararium, a small shrine to the ancestral gods of a Roman house and its family. Sometimes they are painted, or include a niche in a wall, and sometimes they are small built structures, like a miniature temple. They are often located in

Figure 19: Inspection Mode highlights items of interest [83]

4.3 *Mythos Unbound* Levels

The following is a list of *Mythos Unbound*'s major levels and gameplay content. It provides an overview of what the game has to offer.

4.3.1 The House of Octavius Quartio

The House of Octavius Quartio hosts the main narrative of the game and is where most of the action takes place. Played from a first-person perspective, the house scenes follow Glaucus' story as he rises from a slave to a free man. The player will often interact with other characters, such as the slave cook Bibulus and the magister Epictetus. Glaucus' owner, Octavius himself, is also a major character, as is his wife Cornelia and his son Vibius.

Although some house scenes stand on their own, myth levels are normally bookended by prelude and epilogue scenes in the house that together make a cohesive chapter within the game. These scenes will share similar themes that reinforce key learning goals. For example, during the *Trial of Orestes* (see Section 4.3.11), Glaucus must also stand trial before Octavius after becoming involved in Cornelia's machinations. A result of the continuing friction between Octavius and Cornelia, the event demonstrates the complex social structure and politics of a Roman household as well as its political and economic ties to the outside world. These developments serve to relate the mythological texts to Roman culture, emphasizing their importance and relevance to the people of that time.

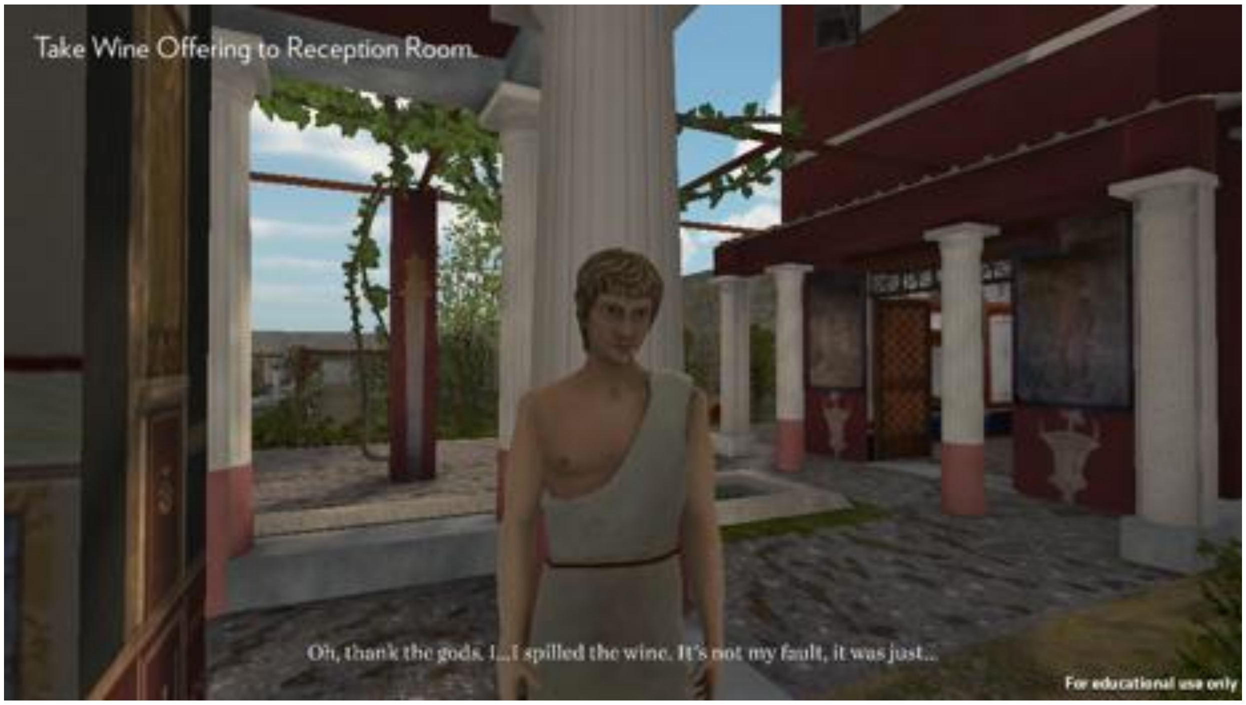

Take Wine Offering to Reception Room.
Oh, thank the gods. I...I spilled the wine. It's not my fault, it was just...
For educational use only

Figure 20: The player interacts with another slave [84]

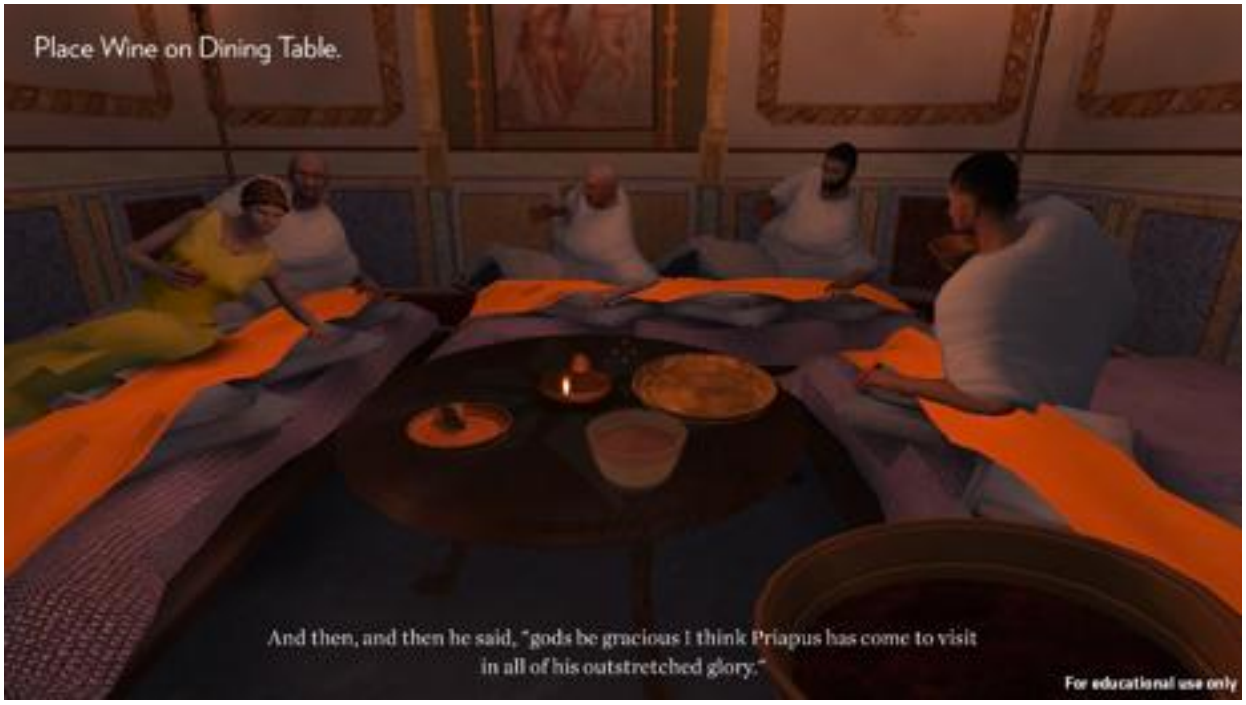

Place Wine on Dining Table.
And then, and then he said, "gods be gracious I think Priapus has come to visit in all of his outstretched glory."
For educational use only

Figure 21: The player serves his owners during an important dinner [85]

The house levels also make use of cinematic cutscenes, which are choreographed events where the player has little or no control. All dialogue in the house is voice-acted, requiring the development of audio player data structures that manage the audio files. These objects organize the order in which audio dialogue lines are played and ensure they sync with subtitles and game events. There was also a great need for executing complex workflows that accurately reflected the script, which led to the development of the step system described in section 4.2.2.

Many custom mini-games or gameplay systems were developed specifically for house events. The following is a short list that details several major examples.

4.3.1.1 Kronos Vase Puzzle

At one point in the story, Glaucus accidentally breaks an Athenian vase and must collect the vase shards scattered by Octavius' son, Vibius. Upon collecting all of the shards, the player engages in a 3D puzzle where the vase pieces need to be placed in their appropriate positions.

The puzzle mechanics were implemented by assigning each piece a "correct" position that it will snap into if the delta between it and the piece's current position is below a certain threshold. The entire puzzle can be rotated around a vertical center axis, thereby defining a new horizontal axis relative to the camera. When a piece is "grabbed" by the mouse (which is determined by ray casts), its motion is limited to the plane defined by the vertical axis and the relative horizontal axis.

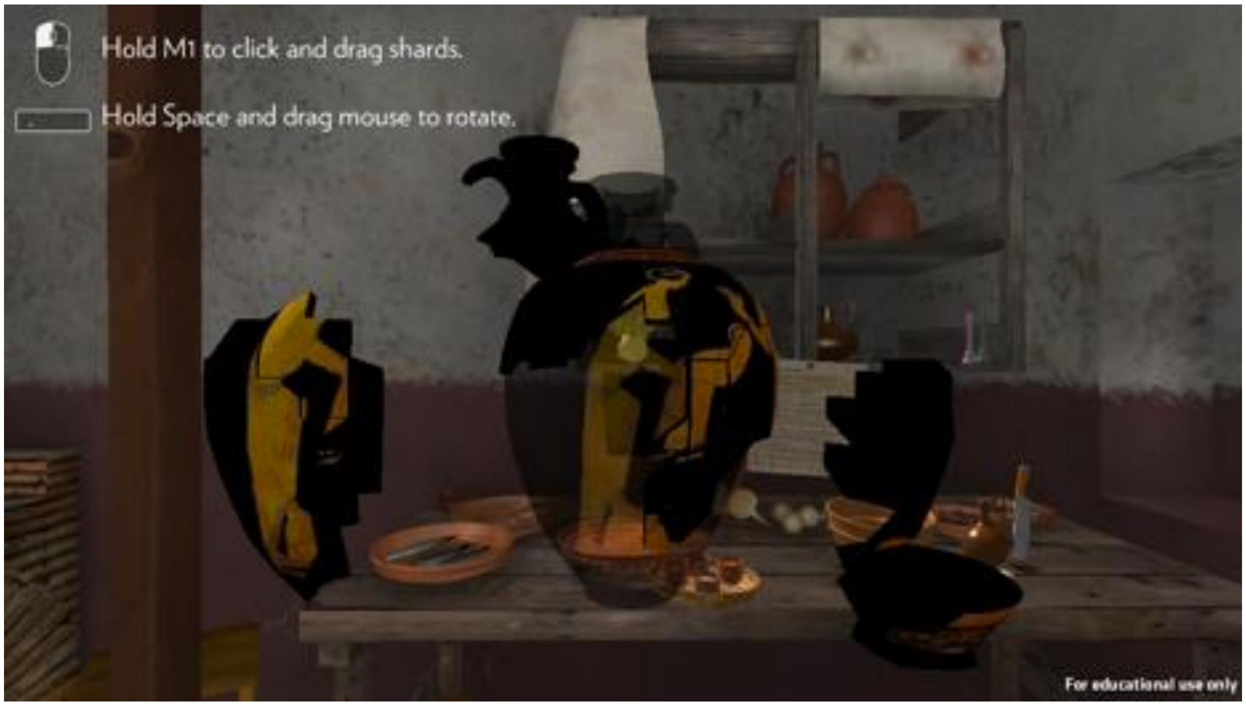
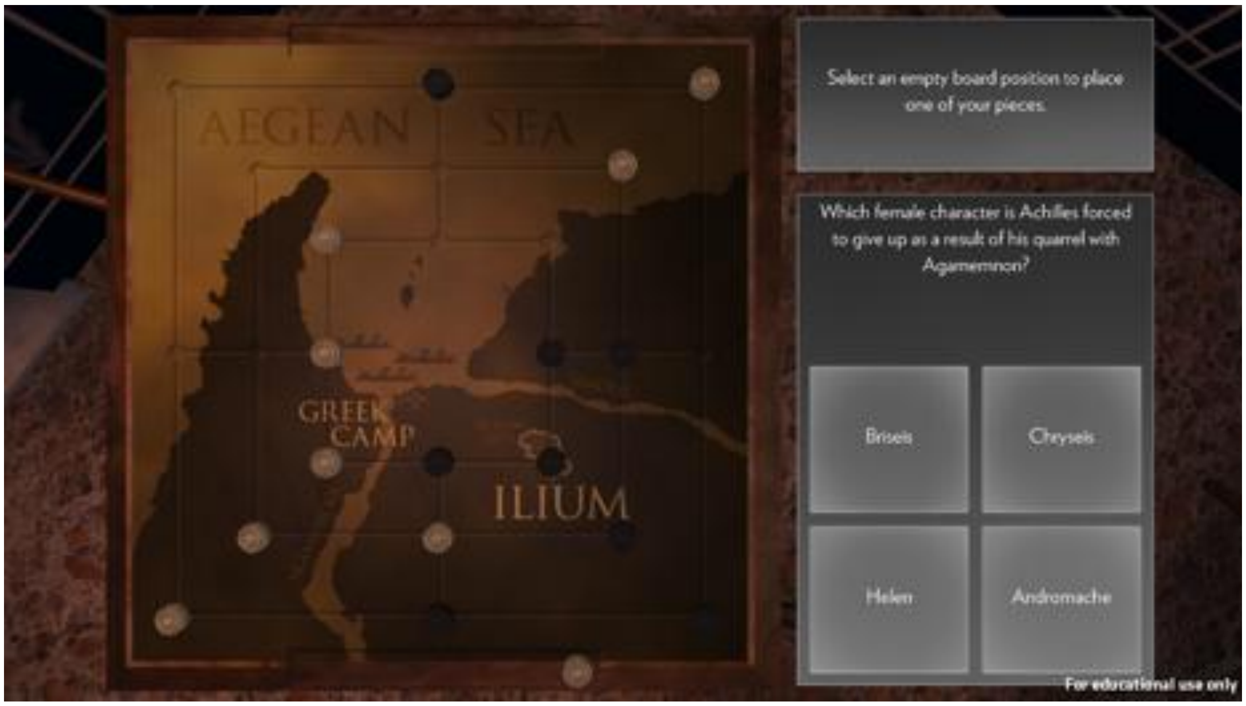

Hold M1 to click and drag shards.
Hold Space and drag mouse to rotate.
For educational use only

Figure 22: The Athenian vase being assembled [86]

4.3.1.2 Nine Men's Morris (Molus)

One house scene has the magister teaching the player about an ancient board game known today as Nine Men's Morris, but referred to in-game by the fictional name of Molus. It plays similarly to a cross between tic-tac-toe and checkers, but in *Mythos Unbound* the board's layout parallels geography taken from the *Iliad*. Each position on the board is tied to a question from the text of the *Iliad* so that the player cannot place a piece on that position unless the question is answered correctly. A simplistic AI opponent challenges the player by making valid moves and removing the player's pieces when given the opportunity. The AI will also take an extra turn when the player misses a question.


Select an empty board position to place one of your pieces.

Which female character is Achilles forced to give up as a result of his quarrel with Agamemnon?

Figure 23: A Molus game in progress with a question being asked of the player [87]

4.3.1.3 Cooking

At several points in the story, the player is asked to perform his slave duties by helping the cook in the kitchen. Given a recipe, the player must follow it accurately by selecting the right tools and ingredients in the right order. While entertaining, the recipes also serve as allegories for myths and events in the story.

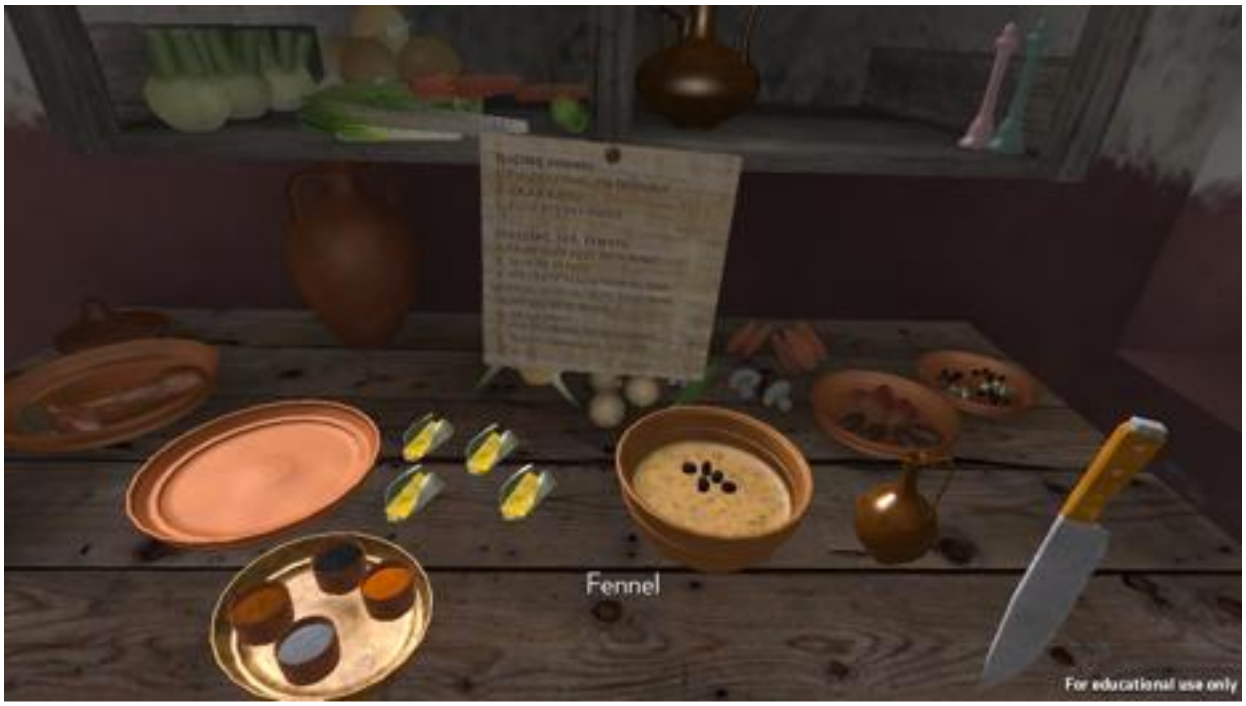


Figure 24: The player is helping the cook make sliced fennel [88]

4.3.1.4 Garden Maze

The House of Octavius Quartio includes a garden that doubles as an environmental maze. Several times over the course of the game, the player must navigate the labyrinth either as a chaser or the one being chased. The first of these events is when Vibius runs off with an important scroll; the child's AI includes evasive behaviors, so the player must guide Vibius to a dead end in order to corner him. A later event involves an assassin dressed as a minotaur that the player must defeat by avoiding it until he can maneuver himself into an advantageous position.

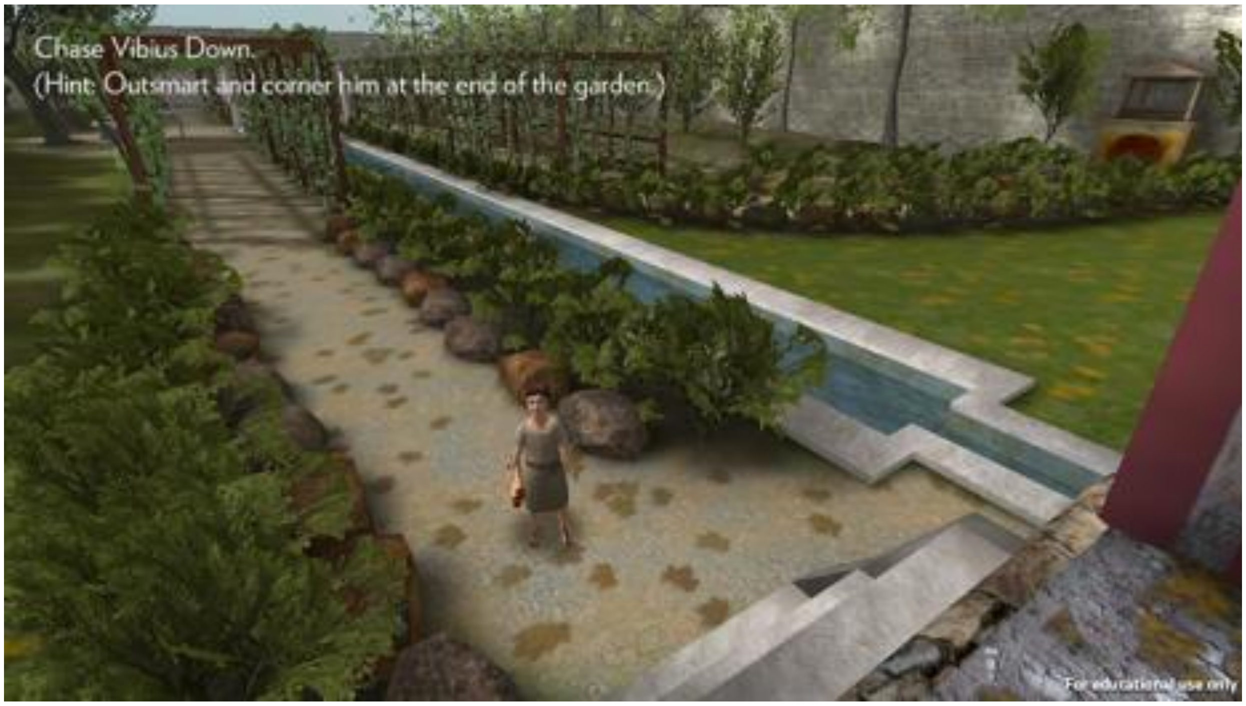


Figure 25: Standing at the entrance to the garden with Vibius [89]

4.3.2 Cosmogony

Mythos Unbound opens with the first *Theogony* myth level. A series of abstracted and narrated events, *Cosmogony* follows the creation of the world. As an amorphous shadow figure, the player witnesses these events by moving through environments such as Gaia and Tartarus.

From a teaching perspective, the goal of this level is to acquaint the player with third-person controls, cameras, and platforming. Later levels depend on a knowledge of these mechanics in order to communicate their own ideas. In other words, these mechanics are at the root of several knowledge hierarchy branches, so it is important that they are established early in the game.

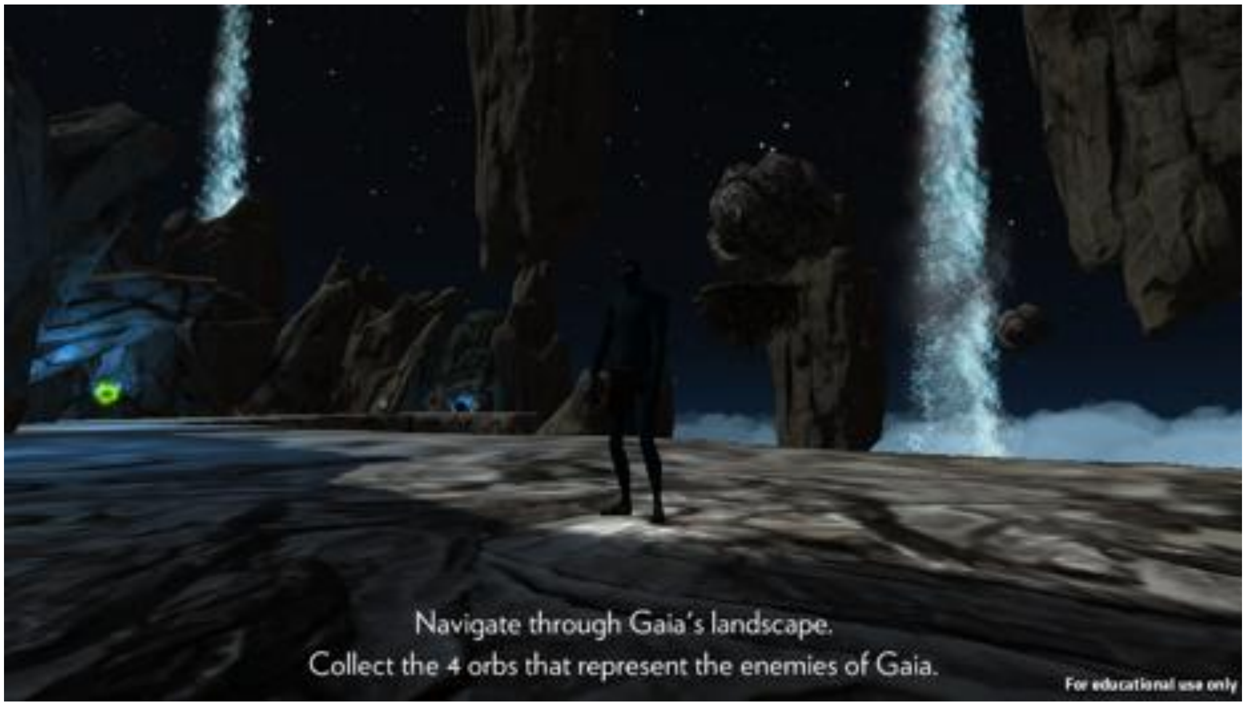

Navigate through Gaia's landscape.
Collect the 4 orbs that represent the enemies of Gaia.
For educational use only

Figure 26: The player's shadowy avatar standing within Gaia [90]

4.3.3 Sinews of Zeus

As Hermes, the player is tasked with retrieving Zeus' stolen sinews in the second and final *Theogony* myth level. In the text, the rebellious dragon Typhoeus hid these tendons in bear-skin bags. The player must use Hermes' abilities to traverse a dangerous environment and find the bags. Along the way there will be bags with skins from animals like wolves and lions, so the player must demonstrate their knowledge of the text by selecting the correct type of bags. Another example of an embedded question comes when Hermes must choose which item to return to Zeus, with "sinews" being the correct answer according to the text.

As a more advanced third-person platformer that builds on the ideas established in *Cosmogony*, *Sinews of Zeus* includes a number of its own mechanics. Hermes is capable of limited bursts of speed in the air to cross certain gaps. The Scepter of Zeus gives Hermes the ability to create shields with a limited energy resource that can be refilled. A final battle with the dragon Delphyne requires mastery of Hermes' movement and shield mechanics.

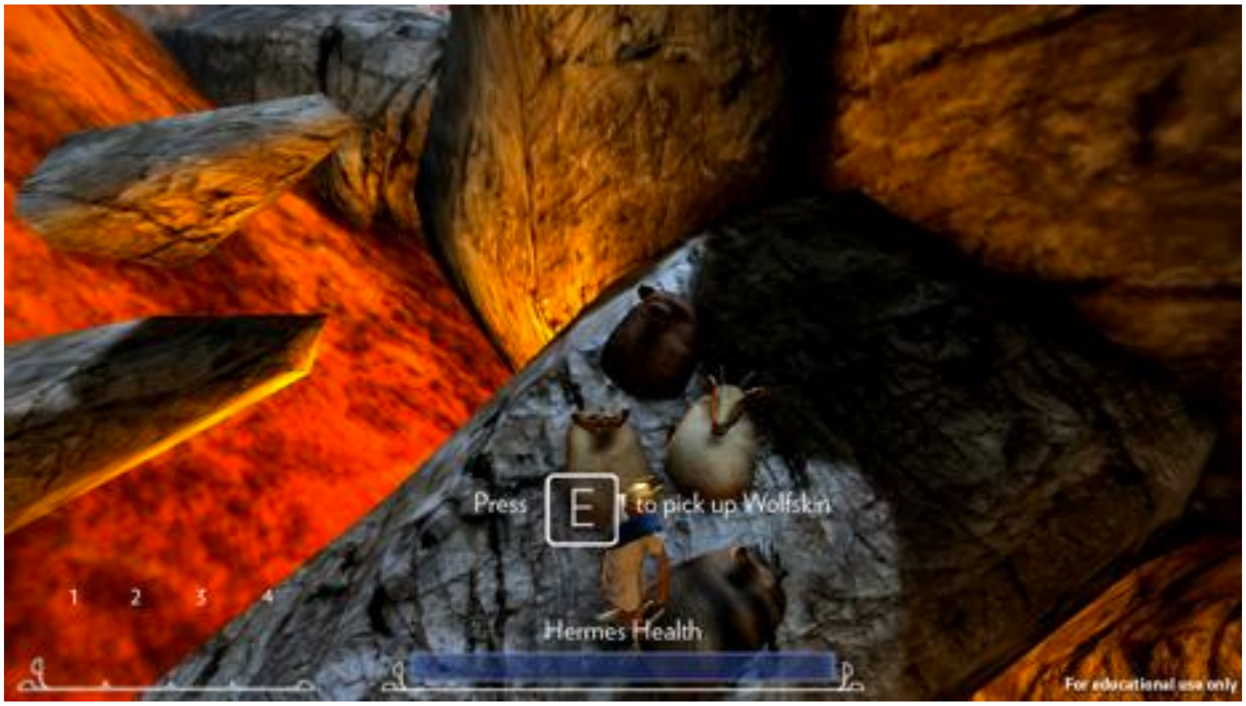


Figure 27: Hermes selects which bag to pick up [91]

4.3.4 Penelope's Deception

The first of three *Odyssey* levels that take advantage of a detailed Mycenaean palace environment, *Penelope's Deception* is from the perspective of the titular character Penelope, wife of Odysseus. While her husband is returning from the Trojan War, Penelope must outwit the hungry suitors that have taken residence in her home by stealthily navigating its labyrinthine halls and gardens.

The beginning of the level features conversations with a number of suitors that were developed with the *Okashi* dialogue tools described in Section 4.2.2. As Penelope, players must provide appropriate responses consistent with the text to avoid suspicion.

Later, the environment introduces stealth mechanics that are revisited in future levels. The suitors are powered by AI systems that patrol regions of the garden, searching for Penelope. If she is found, suitors will pursue Penelope using a navigation mesh pathfinding algorithm.

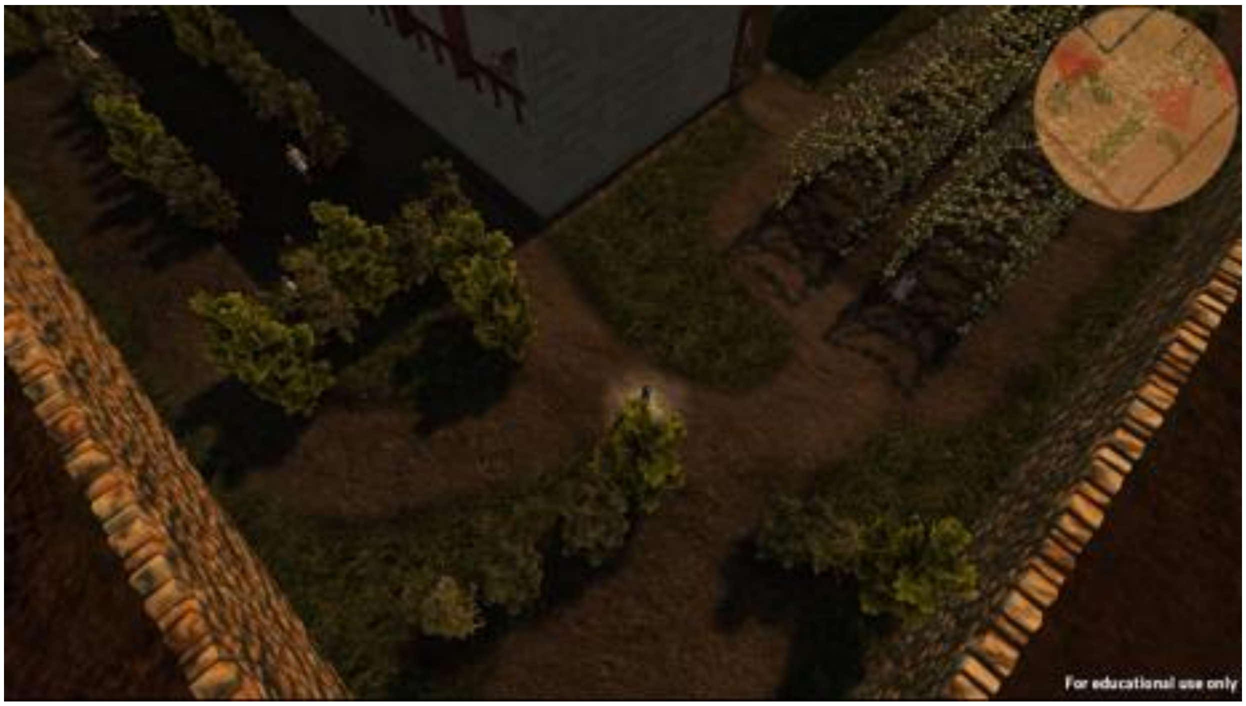


Figure 28: Penelope hides from the suitors in an overhead view of the garden [92]

4.3.5 Calypso's Island

Stranded on the island of Ogygia with the nymph Calypso, Odysseus must find a way to return to his home of Ithaca. While a relatively minor event in the *Odyssey*, the island is divided into sections that serve as allusions to various events within the text. For example, there is a grove of hazardous lotus flowers taken from the island of the lotus-eaters episode, and later the player must utilize a bag of winds inspired from Aeolus' gift to Odysseus' crew to explore a maze-like cavern. The player's ultimate goal is to collect twenty pieces of lumber, the same number Odysseus used to craft a raft in the text. Mechanically, this level also builds on the third-person platformer controls introduced in the previous *Theogony* stages.

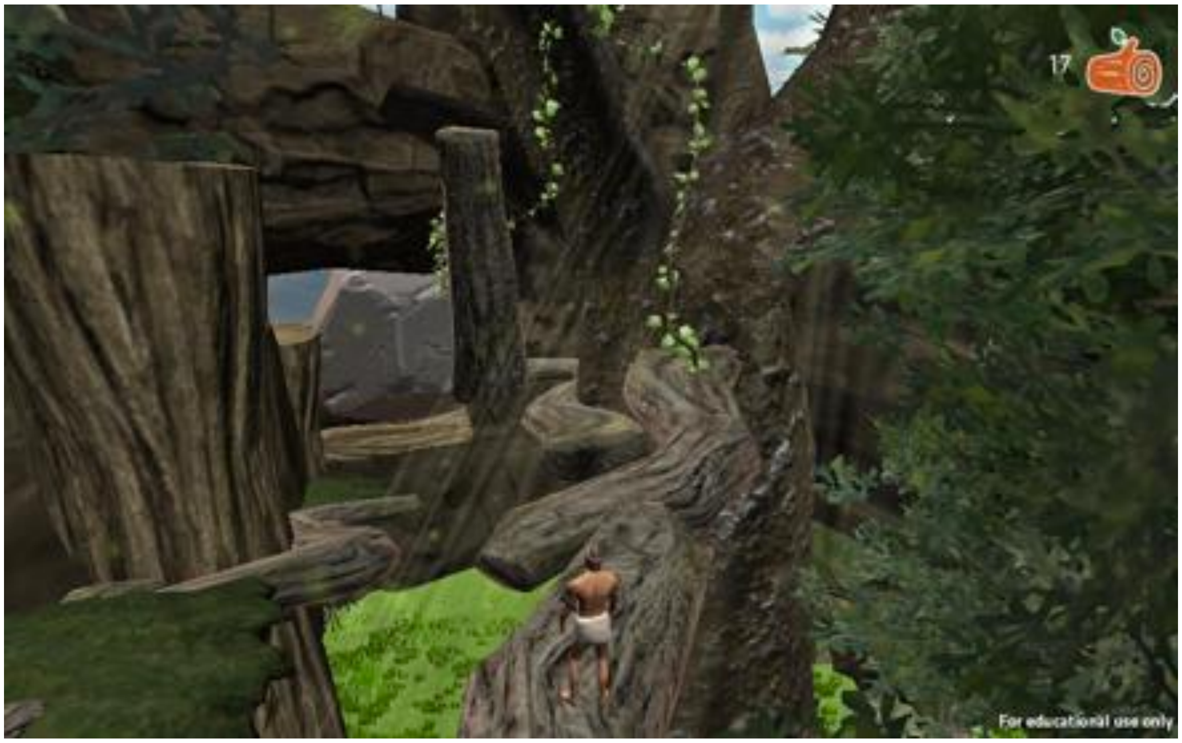


Figure 29: Odysseus explores the treetops of Calypso's Island [93]

4.3.6 Scylla & Charybdis

The third *Odyssey* level has players helm Odysseus' ship to navigate treacherous waters past the Sirens and the sea monsters Scylla and Charybdis. Rife with unique mechanics built specifically for this stage, *Scylla & Charybdis* is one of the larger-scale levels in *Mythos Unbound*.

The player is primarily controlling a large sea vessel as obstacles are avoided and a crew of various class types is managed. Losing shipmates, whether it be due to collisions or sea monsters, will affect the ship's performance based on the crew member's class type. Acceleration, max speed, and turning are all affected by the crew.

When passing by the Sirens, control reverts to a third-person system. As Odysseus, players must follow the text in preparing for the Siren encounter by placing wax in the ears of the crew and requesting to be tied to the mast of the ship. The player must avoid incorrect choices, such as using wool instead of wax or choosing to be tied to the mast before the crew's ears have been protected.

While passing the dual sea monsters, players must adhere to Odysseus' logic that it would be better to lose a number of shipmates to the jaws of Scylla than to sink the entire ship at the bottom of Charybdis' whirlpool. This is represented mechanically in that the pull of Charybdis' whirlpool is inescapable and will always mean certain death, while straying closer to Scylla is survivable despite heavy losses in the crew. This reinforces the themes of the text by integrating the idea of necessary sacrifice into the already established crew system, especially since it is impossible to survive the level unscathed. It is also a great example of procedural rhetoric and maintaining ludonarrative harmony (see Section 2.1.4).

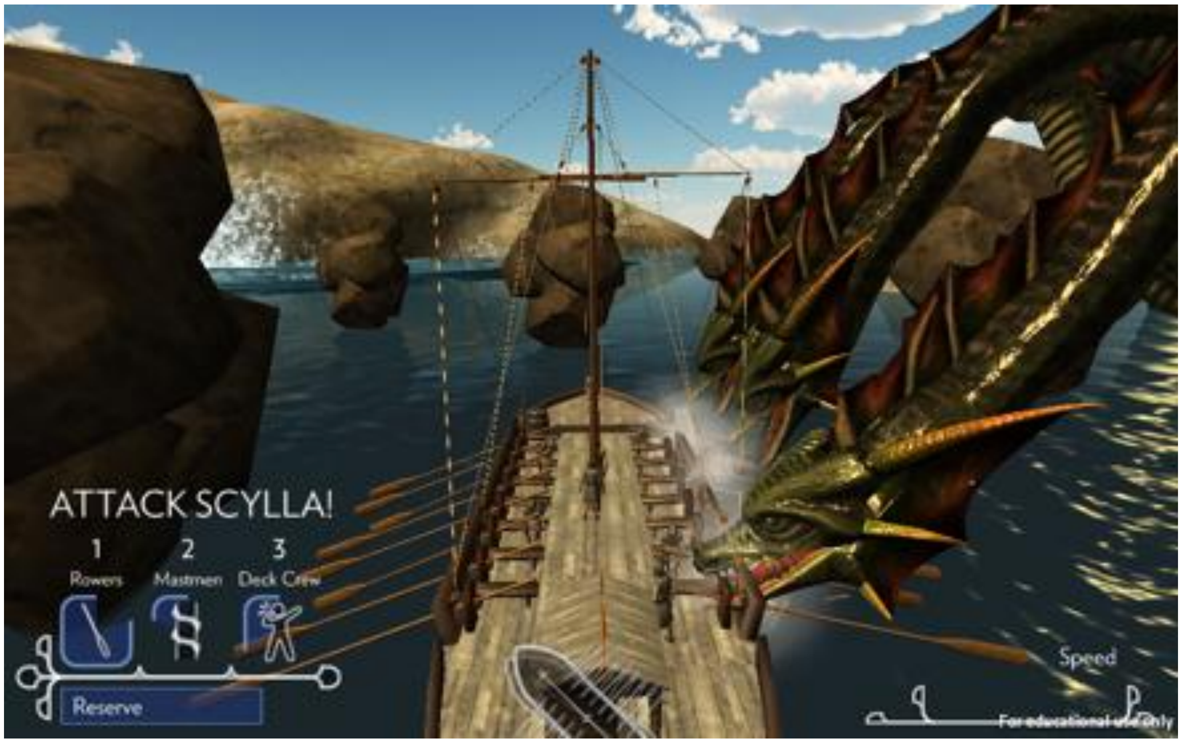


Figure 30: Scylla attacks Odysseus' ship and crew [94]

4.3.7 Telemachus Hides the Weapons

The final two *Odyssey* levels are fairly succinct, aiming to visually represent key events from the text while offering parallels to the narrative in the house scene. They also present great opportunities to tie in writing prompts via the wax tablet mechanic.

In *Telemachus Hides the Weapons*, Odysseus' son, Telemachus, must aid his returned father by hiding the suitors' weapons around the palace. This is a return to the stealth mechanics of *Penelope's Deception* and includes many of the same AI and pathfinding algorithms. A weight system ensures that Telemachus can only carry a few items at a time, requiring multiple trips back to the weapons room and to various locations around the palace.

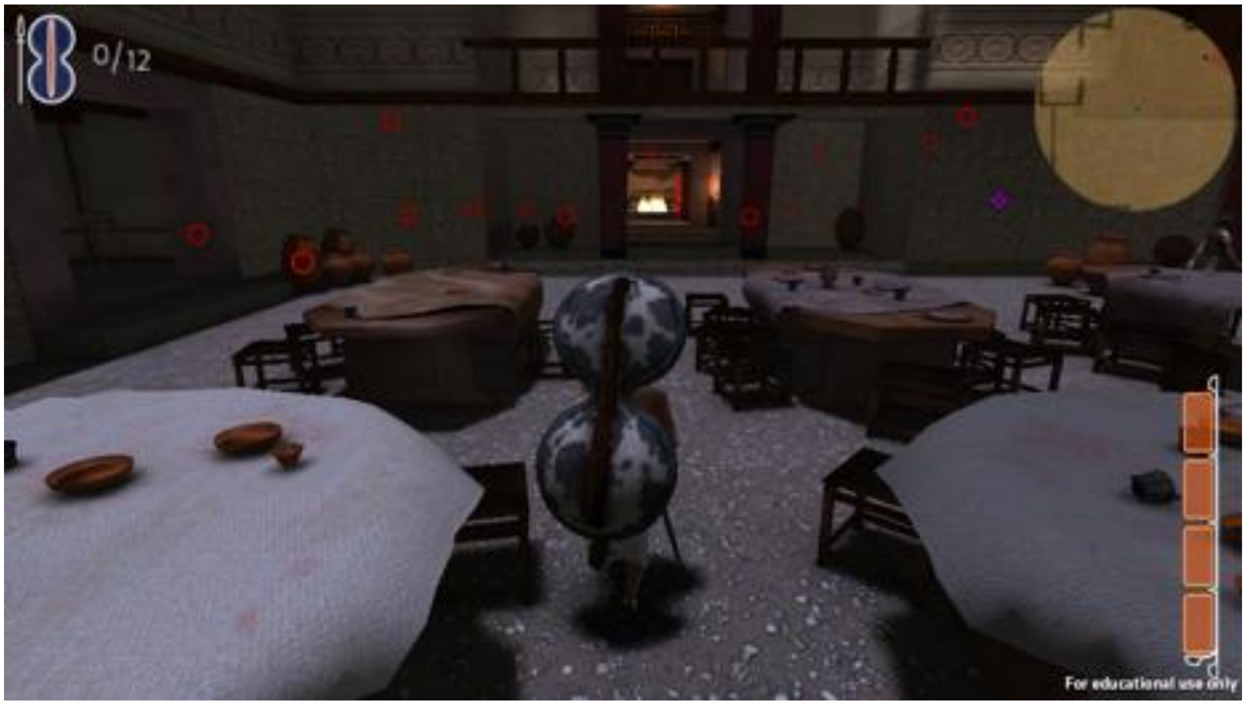


Figure 31: Telemachus evades the suitors and hides the weapons [95]

4.3.8 Odysseus Slays the Suitors

To cap off the *Odyssey* levels, players relive the epic climax by taking control of Odysseus and reclaiming his home from the suitors. Introducing a bow-and-arrow combat system, the player must first win a contest of archery before turning on the suitors. As players have now become familiar with the palace environment, an action sequence has players roaming the environment to slay the suitors and avoid their attacks.

This relates to an important event in the unfolding narrative of the the Roman house. Like Odysseus' surprise attack, the matron Cornelia exposes the nefarious plans of a guest during a tense dinner scene. In doing so, she saves Octavius from a disastrous business agreement that would have ruined him. Cornelia's schemes are part of an ongoing theme that highlights aspects of gender and sex that were prominent in ancient Rome.

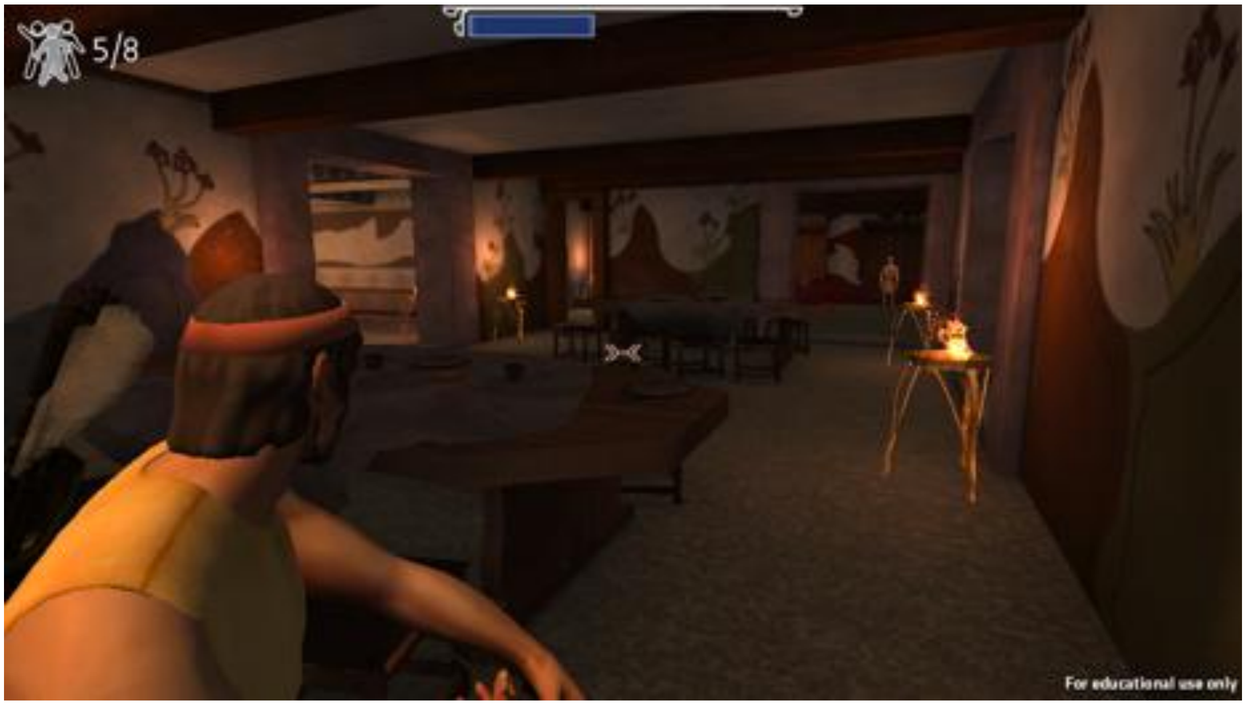


Figure 32: Odysseus searches for the suitors that nearly destroyed his home [96]

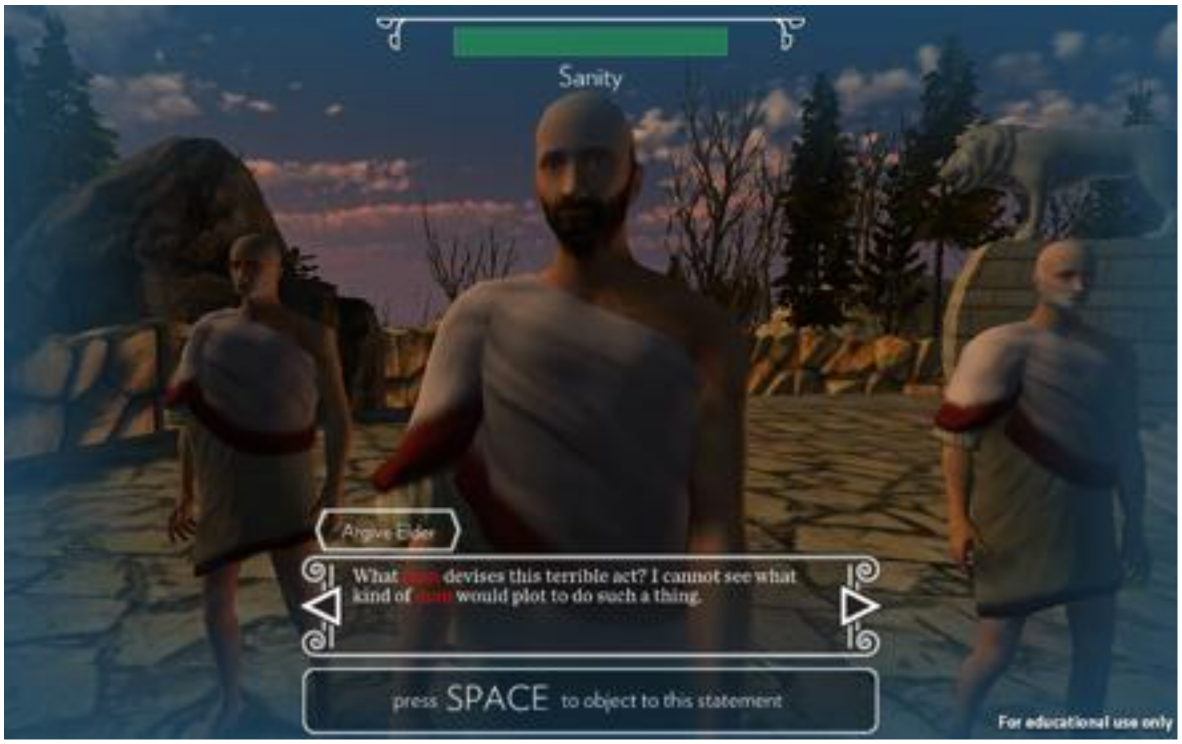
4.3.9 Cassandra’s Prophecy

The play *Oresteia* is explored after the *Odyssey*. Unlike the *Odyssey*, the *Oresteia* lacks an abundance of scenes with action or adventure elements that have obvious gameplay analogues. There is instead an emphasis on dialogue and interactions between characters like Cassandra and Clytemnestra, or Orestes and the Furies. Rather than force some action mechanics into these levels and risk ludonarrative dissonance, the team embraced the idea of conversations as gameplay.

The first *Oresteia* level, *Cassandra’s Prophecy*, introduces a new type of gameplay that relies solely on dialogue between characters. Playing as Cassandra, the player engages in conversation with Clytemnestra and the Argive Elders. The god Apollo has blessed Cassandra with the power of prophecy, but she is cursed in that no one will believe her premonitions. The player must navigate dialogue skillfully by speaking truths and identifying statements that conflict with her prophecies. This is accomplished through two mechanics, the first being simple multiple choices that represent possible statements Cassandra could make. Correct choices will reflect knowledge of the text, such as explaining Cassandra's relationship to Apollo.

The most interesting knowledge-testing mechanic, however, is what the team called "testimony" sections. The player's conversational opponent will sometimes say something contradictory or untrue, allowing the player to roll back the dialogue and reexamine each set of statements separately. Key words relating to the logic of the statement and the events of the story will be highlighted in red, drawing the player's attention to them. The player must then object to the dialogue section that does not align with Cassandra's prophecies. Only by reading the text and understanding the story and the contents of Cassandra's visions can the player make the correct statements and objections, thus completing the level. Making incorrect statements will lower Cassandra's "sanity," and poor performance will result in her going insane and the level restarting.

The mechanics were built on top of a workflow system that eventually led to the full-fledged step system used in the house scenes and described in Section 4.2.2. There were a number of objects that managed the conversations, tying together the workflows, branching paths, health systems, and display of text. In fact, the dialogue box is actually processing the text in the background, looking for certain keywords to trigger certain actions like changing the music or shifting the pace of the dialogue. The team wanted these mechanics to be robust, so it justified the extra development time they needed by using them in the third *Oresteia* level as well.



What man devises this terrible act? I cannot see what kind of man would plot to do such a thing.

Figure 33: Looking for contradictions, Cassandra hears the claims of the Argive Elder [97]

4.3.10 Orestes' Vengeance

This level revisits the stealth mechanics introduced in the first *Odyssey* level as a disguised Orestes sneaks into the palace of his mother, Clytemnestra. The ruling king, Aegisthus, had seduced Clytemnestra while Orestes' father, Agamemnon, was away at war. The two adulterers murdered Agamemnon upon his return, and now Orestes has returned to avenge his father. The player must sneak past the guards and find both Aegisthus and Clytemnestra. These encounters result in cutscenes that end with their respective deaths.

The emphasis in *Orestes' Vengeance* is on the narrative and cutscenes that parallel exactly what occurs in the text. As a result, the murders of Aegisthus and Clytemnestra leave a greater impact on the player and better communicate the seriousness of the content and themes.

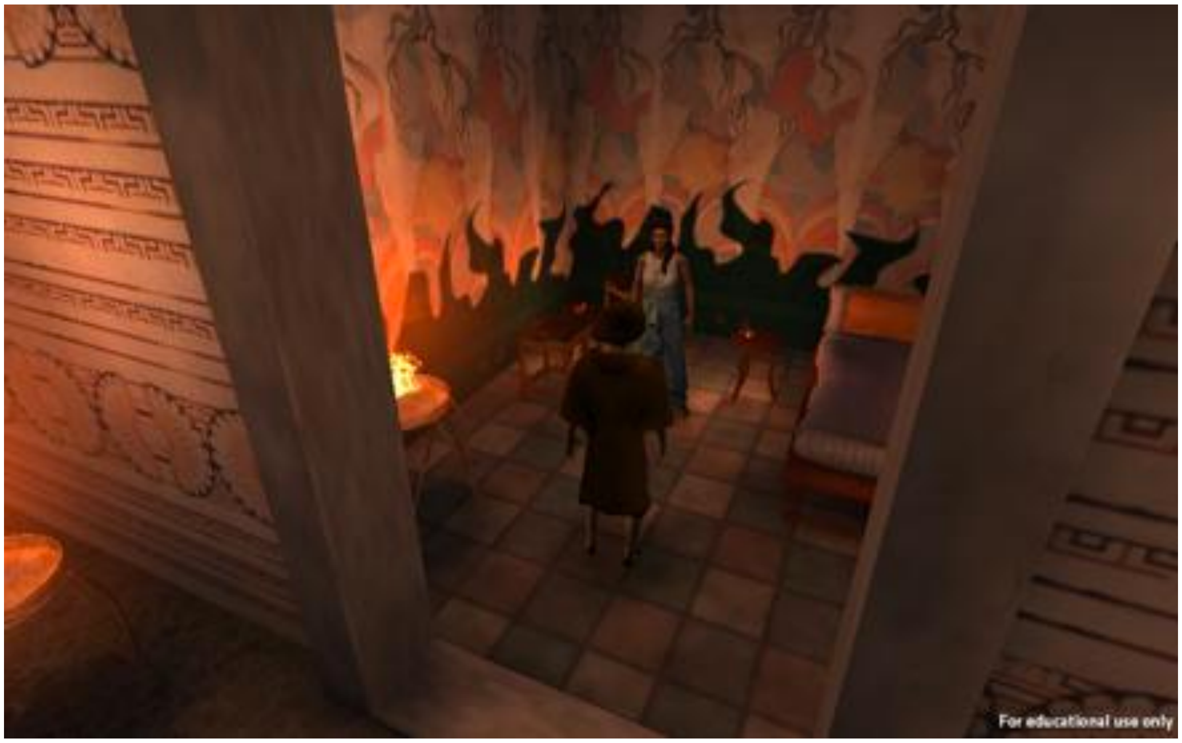


Figure 34: Orestes confronts his traitorous mother, Clytemnestra [98]

4.3.11 The Trial of Orestes

The third and final *Oresteia* level builds off the mechanics established in the first *Oresteia* level. This time, Orestes is under trial for the murder of his mother Clytemnestra, committed in the scenes of the previous level. The Furies have come to convict Orestes while Apollo has come to defend him. Athena arrives as judge for the trial.

Since the player has already learned the base mechanics in *Cassandra's Prophecy*, this level can build on them with more challenging questions and interesting scenarios. There is a new type of question that has the player select multiple statements at the same time, all of which must be true and in agreement with the truths established in the text. A final question has the player see the court from Apollo's perspective, where he must point out a person or item in the courtroom that proves his argument. (The answer is Athena, just as it was in the text.) In general, the conversations are more argumentative and the questions slightly more difficult.

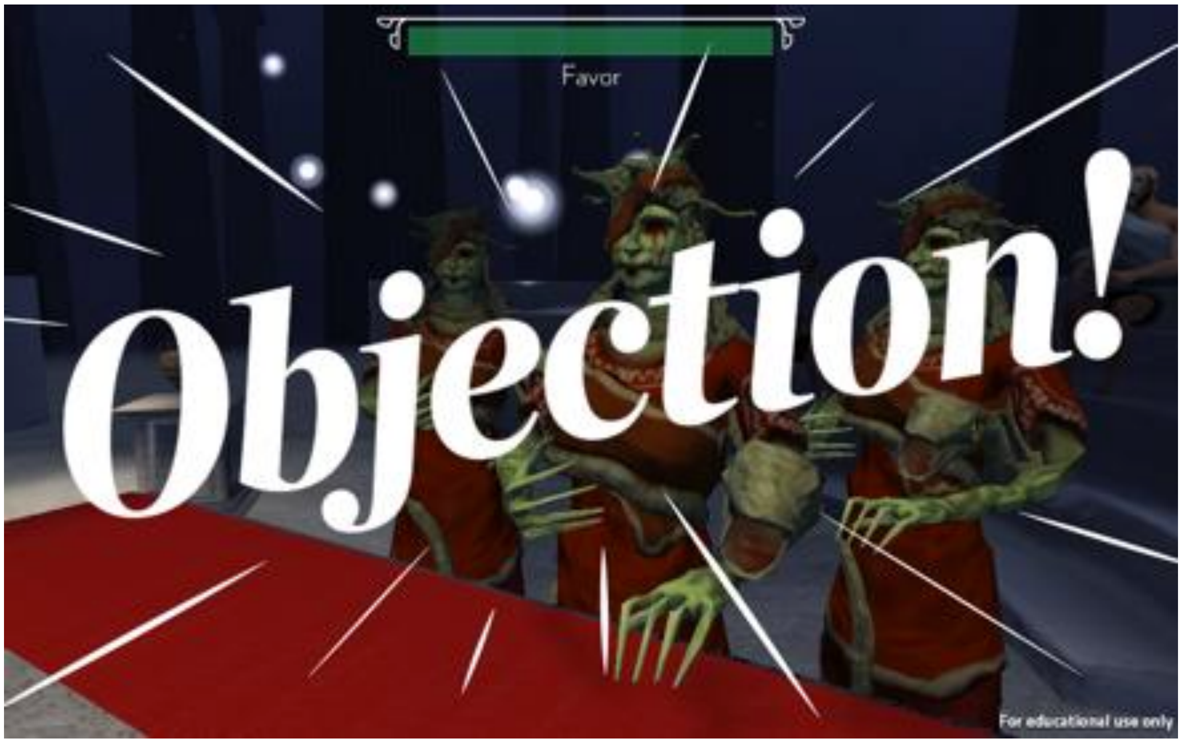


Figure 35: Orestes objects to the Furies' statement [99]

4.3.12 Hercules Wrestles Thanatos

As the sole myth level representing the *Alcestis* play, *Hercules Wrestles Thanatos* has players take on the role of Hercules as he journeys into the underworld to rescue Alcestis from Thanatos. As one might imagine, the level is full of action and incorporates a brawling action gameplay system with various enemy types and attacks.

This level runs alongside a house scene that has the player rescuing Vibius from drowning, directly paralleling Hercules' journey to save Alcestis from her place in the underworld. Taking place during an upper-class dinner party that highlights the social aspects of the Roman elite, the event also pulls inspirations the Satyricon, another ancient text.

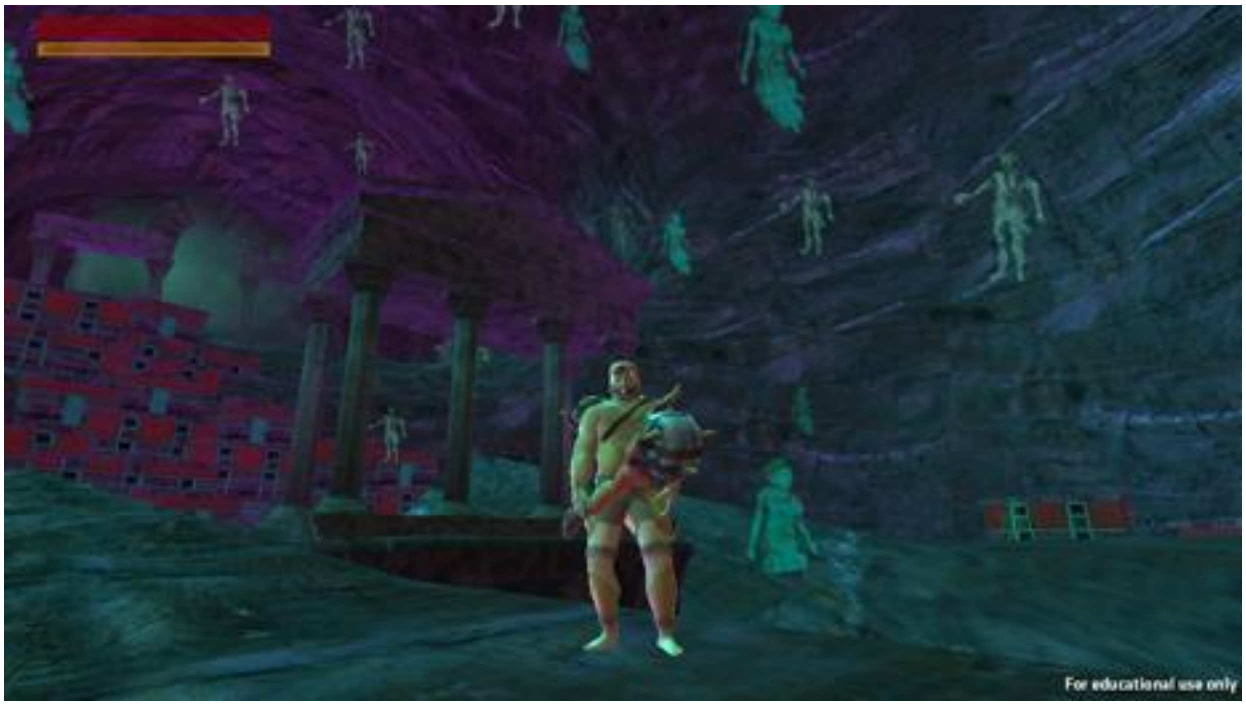


Figure 36: Hercules fights his way through the underworld to rescue Alcestis [100]

4.3.13 Dionysus Comes to Thebes

The first of the two *Bacchae* levels introduces another new gameplay system involving unit and resource management. As the god Dionysus, the player must gather up his maenad followers and spread his cult throughout Thebes. To do this, Dionysus' maenads must be instructed to perform several tasks, such as tearing down gates or destroying archer towers.

The implementation of this level included AI for the maenads that allowed them to intelligently follow Dionysus and perform actions based on his directions. One of the key techniques contributing to this behavior was a flocking algorithm that allowed the maenads to organically move around Dionysus and organize themselves in flowing, natural patterns. It involves maintaining equilibriums between a number of desires (such as *cohesion* and *separation*) that are represented as separate movement vectors.

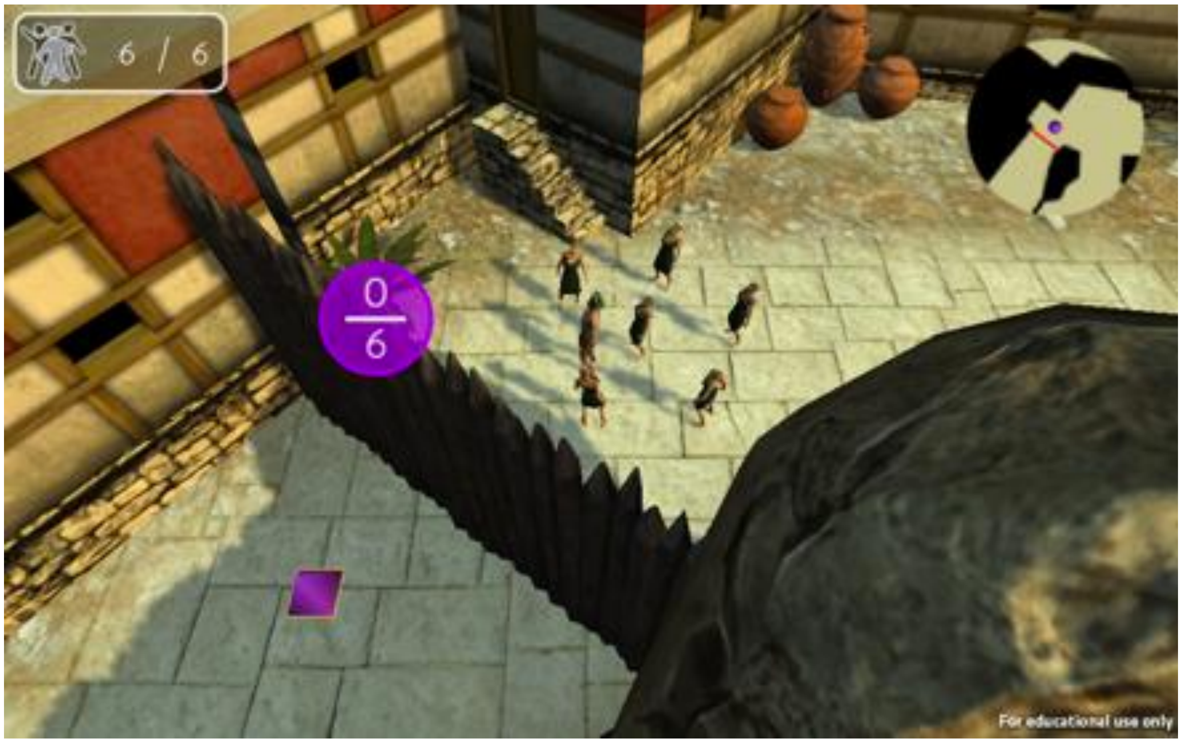


Figure 37: Dionysus leads his maenad followers in Thebes [101]

4.3.14 Pentheus and the Maenads

In the *Bacchae* play, the Theban king Pentheus refused to acknowledge Dionysus and worked against him and his followers. This eventually led to his demise as he was torn limb from limb by Dionysus' maenads. As the slave who will witness the event, the player must stealthily follow Pentheus and Dionysus as they climb a mountain, avoiding Maenads using the same stealth mechanics leveraged in levels like *Penelope's Deception* and *Orestes' Vengeance*.

Having grown arrogant after becoming Octavius' favored slave, the accompanying house scenes depict Glaucus refusing to pay tribute to the gods. This parallels Pentheus' ignorance of Dionysus' strength and leads to the slave's temporary fall from grace. Glaucus' suffering invites students to make comparisons to the *Bacchae* events portrayed in this level.

Figure 38: Hiding in the woods, the player observes Dionysus and Pentheus [102]

4.3.15 Future Levels

At the time of writing, not all of *Mythos Unbound* has been finished. The remaining levels come from Ovid's *Metamorphoses* and are listed below:

- *Diana and Actaeon*
- *Narcissus and Echo*
- *Minotaur Pursuit*

- *Orpheus and Eurydice*
- *Vertumnus and Pomona*

Like the other levels, these will include their own unique mechanics while building off of perviously established ones. The accompanying house scenes will round out Glacus' story as he attains his freedom.

5. ANALYSIS

5.1 Evaluation of the Framework

Chapter 3 described a framework for transforming knowledge into gameplay, but the framework alone does not provide meaningful data with which to evaluate its efficacy. The framework's principles can be applied to *Mythos Unbound*, a serious game described in Chapter 4. The two were developed concurrently and together serve as a representative example of the potential of gamification. Thus, examining the framework in the context of *Mythos Unbound* can provide insight into its effectiveness. [2]

5.2 *Mythos Unbound* Learning Outcomes

In order to evaluate whether or not *Mythos Unbound* has achieved its goals, an explicit list of those learning outcomes were identified. [3]

Primary Learning Outcome

- An understanding of classical myth in a Lived Roman context

Subordinate Learning Outcomes

- Connection of myths to everyday practices
- Connection of myths to fundamental cultural/ideological structures

[2] This is not to suggest that the outcomes of *Mythos Unbound* are an objective metric by which the entirety of the framework can be validated. It should also be recognized that the course was still ongoing in November 2013 at the time of writing, and thus its results are not complete. However, it does provide a real-world example backed by the grades and feedback of actual students. This kind of data is not common and is valuable in evaluating gamification techniques like this one.

[3] It should be noted that these learning outcomes were identified after the fact and the data used to compare with the outcomes was anecdotal, deidentified data provided by the course instructor.

- Connection of myths to Roman socio-political realities
- Connection of myth to education and social advancement

5.3 Applying the Gamification Framework to *Mythos Unbound*

With the learning outcomes in mind, the relationship between the gamification framework and *Mythos Unbound* can be explored. By walking through the tenets of the framework and connecting them to the development of the game, the effectiveness of the framework can be illustrated.

5.3.1 Outlining the Knowledge Hierarchy

Sections 3.2.1 through 3.2.5 describe how to take a body of knowledge and make a formal tree known as a knowledge hierarchy. While such a tree was never explicitly mapped out during the early development stages of *Mythos Unbound* (before the framework was even theorized), the team still took a critical approach to knowledge items and how to best communicate them in a structured way that was consistent with the principles of the knowledge hierarchy.

The domain of knowledge was identified early on through the learning outcomes listed in Section 5.2. From the beginning, the team also knew exactly which classical texts were to be represented within the game: *Odyssey*, excerpts from the *Iliad*, *Oresteia*, *Alcestis*, *Bacchae*, and *Metamorphoses*.

From there, higher-level knowledge items were decomposed further into more atomic units of knowledge. For example, in connecting myths to everyday practices, subjects like cooking and eating, death and birth, and sexuality were identified. In connecting myths to Roman socio-political realities, the team wanted to cover topics like the social mobility of slaves and the relationship between patrons and clients.

The knowledge hierarchy branches were fairly clear to identify. The primary learning outcome largely depended upon the mastery of the subordinate learning outcomes, which in turn depended upon the lower-level knowledge items described above. Clearly, the stories told in the mythological texts had linear lines of dependencies relative to their respective plots, e.g.. the ending of the *Odyssey* and its associated ideas and themes could not be explored until the rest of the text had been addressed. Content from the texts would also be aligned with ideas for the learning outcomes; for example, understanding the importance of household gods (connecting myths to everyday practices) relates to *Pentheus and the Maenads* described in Section 4.3.14. The two items thus share a joint position in their respective branches, strengthening the link between Roman culture and mythology and reinforcing the communication of the primary learning outcome.

Due to the lack of strict mechanical dependencies in the knowledge domain, the knowledge hierarchy tiers were fairly flexible. Knowledge items across learning outcomes could be presented concurrently and often were. For example, in the House of Octavius Quartio, Cornelia's actions and scheming were symbolic of the role of women (part of the fundamental cultural/ideological structures) and of the importance of information management (part of social advancement). The tiers were arranged in such a way as to contribute to an overarching narrative that reinforced the ideas and themes of the course.

5.3.2 Constructing the Mechanics and the Design Space

In developing the gameplay systems, the team needed to create mechanics that would result in a design space that matched the knowledge domain. Because of the emphasis on narrative both in the mythological texts as well as the narrative within *Mythos Unbound* itself, it was imperative that the mechanics were capable of delivering immersive storytelling experiences. The following mechanics and technologies in the House of Octavius Quartio are an example of this.

- **First-Person Controller**, the means by which the player can even interface with the game in the first place. It can be manipulated by player input or game events to communicate ideas through gameplay interactions and cinematic cutscenes.
- **Workflow and Event System**, the tool used to build complex sequences of steps and interactions that tell the story of *Mythos Unbound*. This enables the narrative to depict events from the overarching story that reinforce the role of Roman slaves (connection to cultural/ ideological structures) and the social mobility of slaves (connection to Roman socio-political realities).
- **Graphical User Interface**, used to clearly communicate important information and objectives as well as serve as a means by which players can select responses and answers that can be evaluated for understanding.
- **Mini-games**, of which there were several and each served to communicate important knowledge items. For example, the cooking mini-games explicitly connect myths to cooking and eating (connection to everyday practices), and a garden maze event involving an illusory hermaphrodite ties into the Roman views of sexuality (connection to everyday practices).

- **Wax Tablet**, the means by which players can submit their own critical analyses that demonstrate their understanding of the texts, knowledge items, and their relationship to the gameplay (including the narrative, environments, and mechanics).
- **Audio Player Management**, used to orchestrate the sounds and songs needed to create an immersive environment. It is also used to structure the many voice-acting audio files and ensure they play at the correct times and in sequence with subtitles.
- **Item System**, a mechanic whereby the player can pick up and deliver items of importance. Selecting appropriate items and placing them in correct locations is a tool used to underline learning outcomes.

Outside the house, the myth levels introduce too many mechanics and systems to enumerate here in a comprehensive list, but the following are representative of the breadth of content and demonstrate their connections to the knowledge hierarchy and learning outcomes. (More detailed descriptions of the mechanics in their respective contexts can be found in Section 4.3.)

- **Bag Pickup System**, used in *Sinews of Zeus* (Section 4.3.3) to embed challenges that required an understanding of the text to continue. Players can pick up multiple types of bags, but only by reading the text and knowing which type of bags Hermes picked up can they also select the correct bags.
- **Crew System**, used in *Scylla and Charybdis* (Section 4.3.6) to provide procedural rhetoric that ties the gameplay to the themes of necessary sacrifice and logical problem-solving (connection to education and social advancement). It is also used to evaluate understanding of the text through gameplay-embedded tests.

- **Testimony System**, used in *Cassandra's Prophecy* (Section 4.3.9) and *The Trial of Orestes* (Section 4.3.11) to both evaluate understanding of the text by selecting appropriate responses consistent with the myths. This also plays into logical problem-solving and information management through misdirection and concealment that were important aspects of Roman culture and values.

Many of these mechanics and systems were across separate branches and were often presented concurrently within similar tiers. As the player progresses through *Mythos Unbound*, new mechanical ideas are introduced that build on these established gameplay systems within their respective branches. The third-person controller is one such example.

- **Third-Person Controller**, first introduced in *Cosmogony* (Section 4.3.2) and used in over half of the myth levels. It provides a means for players to interact with and maneuver around their environments. The controller is the foundation from which other mechanics are extended in later levels once the base mechanics have been mastered upon completion of *Cosmogony*.
- **Health System**, implemented in several levels to build on the third-person controller mechanics. As opposed to the player invulnerability in *Cosmogony*, the health system introduces a mechanic wherein the player will be forced to restart a sequence if too many mistakes are made, often in the form of incorrect choices or enemy attacks. It is often used for the purposes of repetition (Section 2.2.1) and gating (Section 3.2.8).
- **Stealth System**, which is several times reintroduced in different mechanical contexts related to third-person movement, such as *Penelope's Deception* (Section 4.3.4) and *Orestes'*

Vengeance (Section 4.3.10). The player must avoid the detection of enemy AI agents while traversing a complex environment.

- **Hide Mechanic**, performed by the third-person controller after being introduced in *Telemachus Hides the Weapons* (Section 4.3.7) to hide certain items in strategic locations around the environment. This is accomplished by navigating to the items, picking them up into an inventory with limited capacity, and then navigating to available locations to deliver the items and progress towards an overall goal.
- **Combat System**, used in *Hercules Wrestles Thanatos* (Section 4.3.12) to attack enemies and manipulate the environment from a third-person controller. This later builds into a *combo system* where the player can use combinations of attacks to increase performance and acquire valuable resources.

These mechanics are all used to facilitate the game's narrative and better communicate its ideas, sometimes in more subtle ways as supporting systems (like the flocking mechanics in *Dionysus Comes to Thebes*, described in Section 4.3.13), other times as items central to key themes (the testimony and crew systems described above). They collectively establish a design space that matches the domain of knowledge identified early in the project and represented by the learning outcomes.

5.3.3 Incorporating Narrative, Art, Audio, and Aesthetics

As a project with an emphasis on humanities content, the story of *Mythos Unbound* was at the foundation of the game. It was written in accordance with the identified knowledge items and provided a structured progression through the game's tiers and branches. Key plot elements were written specifically to parallel the literature at a pace consistent with the order and allotted reading times of the texts, as seen with the level outline in Section 4.3. The narrative also adhered to the hierarchy of mechanics, ensuring that advanced mechanics were not introduced until earlier base mechanics had been mastered in a manner that was consistent with the story. For example, the first house scene largely has the player simply coming to grips with the first-person controller as a newly-inducted slave, whereas later scenes have the player interacting with items and talking with NPCs during complex dinner events.

The artists created visuals that were both functional and also representative of Roman environments and art. The House of Octavius Quartio itself is modeled after a real house in Pompeii and greatly increased immersion. Many characters were custom-made by our largely self-taught modelers and were rigged to animate in a convincing manner. Without characters, a story cannot be told as effectively, and, by extension, the knowledge items would be difficult to communicate in this context. Even art elements like the user interface were designed to be thematically consistent while also excelling in clear, concise communication of information needed to learn and complete the game.

Another important element to a story-intensive project like *Mythos Unbound* is the audio. All of the dialogue in the House of Octavius Quartio is voice-acted and enhanced the experience by adding a sense of weight and realism to the scenes. Each scene's music crafted an atmosphere that matched the tone of the story and its mechanics, such as the eerie sounds of the underworld (Section 4.3.12) or the upbeat excitement of cornering an opponent with a well-crafted argument (Section 4.3.11). The many sound effects, ranging from footsteps to chirping birds, brought a greater sense of immersion to the environments.

5.4 Results

Mythos Unbound, being a game that is currently being played by an actual class of about twenty active students, demonstrates that the principles of the framework can indeed lead to the construction of a serious game. Whether or not this framework will also result in serious games that are consistently effective in communicating their domains of knowledge to players will require more than just one offering of one course developed in this manner. However, even though this semester's offering of *Mythos Unbound* has not finished, early grades and feedback have been positive and can serve as early indicators of the educational potential of both *Mythos Unbound* and the framework itself (based on personal communication from the course instructor).

Over the course of the semester, users elected to provide feedback describing their experiences with the game. The following are deidentified excerpts:

> **Scylla and Charybdis**
>
> This level was very fun. I really loved the dramatic music when driving the ship.
>
> This level was probably one of my favorites, so far! The music was absolutely beautiful and really fit the action of the game. It also had the most variety of instruments (I think) so far, which was awesome because it kept me at the edge of my seat (out of fascination and interest) and I never got bored/annoyed with the tune as in other levels.
>
> I have to agree, this has been one of my favorite levels so far. The actual game level (not the prelude or epilogue) was very easy for me to understand. […] Also, I agree that the music was fantastic. It perfectly captured what I had expected and pictured as a siren song which made the level even more interesting and fun.
>
> **Cassandra's Prophecy**
>
> I loved the music, details, and effects of this level. The pop up that screamed "Listen to Me" was an awesome touch. The typing sounds were a cool effect in my opinion. This is one my favorite levels so far, really enjoyed it!

> The environment of this level was amazing! I couldn't stop looking at the background because there was so much detail. I liked how you got to choose the dialogue but it got kinda boring because it was all talking and you didn't really get to do much but I understand seeing that it is the first level for this book.

However, alongside the positive comments the team also received occasional reports of confusion or even bugs or glitches. These have been saved and will be referenced during future revisions of the game (see Section 6.3).

In terms of grades and student work, according to the course instructor, the gamified version of the course has resulted in above-average grades (usually A's and B's) compared to previous, more traditional offerings. The instructor also indicated that student essays are of higher quality than in the past, often directly connecting ideas and themes from the gameplay to the literature. As an example, the first long essay assignment was related to gender roles, a key element of the learning goal of connecting myths to fundamental cultural/ideological structures. The following essay excerpts from two students (deidentified and provided by the course instructor) make clear the connections between assigned reading and game content:

> Quartio's long time friend, Caecilius, who is attempting to sell him the wine, has a similar concern regarding Cornelia as the suitors do Penelope. She has stolen his wax tablets and accused him of a horrible betrayal. Carson notes that the Greeks expected women to "'let [themselves] go' in emotion or appetite" and that seems to be what Caecilius immediately assumes (Carson 156). His first defense is that Cornelia must be drunk, but when it is clear that she is not, he starts attacking her character. He makes her out to be deceitful and wild—all stereotypes presented of women in the Carson article.

> Cornelia seems "leaky" and "out of place" to several of Gauis' clients, as we saw in Level 3 of the game while serving those clients from Glaucus' point of view. Some of the things those clients were insinuating about Cornelia, by referring to her as a "Clytemnestra," match up with Carson's categorization of women as dangerous (because of their wetness, leakiness, and dirtiness). Despite these insinuations, Cornelia stands out in the last, most recent levels of the game as nobly clever. She is one who beautifully exhibits intellectual finesse.

5.5 Analysis

While it is too soon to objectively conclude that the framework is always effective and reliable in producing serious games, the current data from *Mythos Unbound* suggests that the game elements were an improvement compared to previous offerings. As seen by the above comments, student feedback to the course was largely positive. Students were able to make stronger connections to the learning goals and demonstrated a deeper understanding of the course content, especially when in the context of *Mythos Unbound* scenes and levels (as evidenced by the above essay excerpts). Considering how well-aligned the framework is with *Mythos Unbound* (as seen in Section 5.3), this would suggest that the approach offered by the framework is an effective one.

Most issues users had with the game were related to the team's relative lack of resources and experience. As the team's first game development project, some mistakes were made in clearly communicating mechanical rules and goals. Several desirable features had to be cut from this first iteration, such as asset streaming (Section 4.2.6) and Inspection Mode (Section 4.2.7), and other features could be made more robust or reliable, like some server communication issues with the wax tablet (Section 4.2.5).

Another caveat is that the class is a small sample size of only about twenty fully participating students. More accurate results could be obtained by offering the course to a larger body of students along with pre- and post-tests (Section 3.3).

Regardless, the results were positive overall. At the very least, they suggest that this approach to gamification has potential and merits further study and investment. Future projects should only improve in quality now that the team has established a workflow and has acquired a large amount of experience. The initial funding allowed the team to invest in key tools and technologies that can be reused, so development should be more cost-effective as well.

6. CONCLUSIONS

6.1 Summary

This thesis presented a framework from which serious games can be constructed. By critically analyzing a domain of knowledge and deriving a knowledge hierarchy, a game can be designed that respects knowledge dependencies and includes mechanics that effectively communicate ideas in a structured way. These mechanics can be dynamic and iterative to ensure a player's understanding or mastery of key concepts. The framework's format and methodology also results in gameplay, art, story, and sound that align with the knowledge hierarchy to enhance the learning process.

Beginning in October 2012, the author was part of a team that developed *Mythos Unbound*, a serious game designed to teach players about Greek and Roman culture and literature. Developed alongside the gamification framework, *Mythos Unbound* consisted of over a dozen myth levels and about twice as many scenes in the House of Octavius Quartio, a Roman household modeled after a real location in Pompeii. Gameplay was varied and covered several genres that supported an overarching narrative that reinforced key learning objectives. The game was offered online at the University of Arkansas in the 2013 fall semester, and overall course grades and user feedback anecdotally suggest it was a positive improvement to the learning process. *Mythos Unbound* is representative of the viability of the gamification framework being proposed, and so this serious game indicates that the framework has potential worth investigating further.

6.2 Significance

The contributions of this framework are significant in the following ways.

- A base framework for formalizing serious game development has now been established where previous approaches were insufficient or non-critical in nature.
- With the framework in place, more serious games can be developed that are modeled after it. This should lead to games that are more effective at communicating their ideas.
- Specifically in the field of education, this framework could lead to the development of educational games that improve the learning process and result in better grades and understanding of the knowledge material.
- This framework also represents a means of computationally representing knowledge and understanding that can be applied to other computer science fields.

6.3 Future Work

The relative success of *Mythos Unbound* merits further development of the gamification framework through the creation of more serious games. In fact, because *Mythos Unbound* was considered a success by its funding sources at the University of Arkansas Global Campus, a new serious game project focusing on the history of Rome has been offered to the team with development beginning in the near future.

However, this does not mean *Mythos Unbound* is finished. One of the advantages of gamified content is that it can be offered again and again, and with the initial run of *Mythos Unbound* being considered a success, the game will be offered as part of a course again in 2014.

Whereas this past semester's course consisted of only about twenty actively participating users, the next offering is expected to accommodate a much larger group. This second offering will give the team an opportunity to collect and act on game improvement feedback in an organized manner (see Section 5.4) which will in turn feed back into the framework. The team can also revisit features and mechanics that were cut the first time around, such as streaming (see Section 4.2.6) and Inspection Mode (see Section 4.2.7).

While the team did not take advantage of robust pre- and post-testing in *Mythos Unbound* to further validate its results and give a more objective understanding of its success, next year's offering will allow the team to rectify this. New student team member William Loder will be interviewing students of both the past and upcoming *Mythos Unbound* courses to acquire even more feedback. In addition, questions will be given to students of the next class as part of a new pre- and post-testing process. Loder's year-long research project will be part of a SURF grant.

Beyond the future of the *Mythos Unbound* team, the established framework opens the door for other teams to be organized and begin developing serious games. The current team focuses on subjects related to humanities, but the framework could easily be applied to STEM fields. In particular, the author's field of computer science could easily be enriched by serious games developed with the framework if games like *Code Hero* and *Ruby Warrior* (see Section 2.2.4) are an indication of what is possible.

The framework itself could also be improved upon. While it serves as a good foundation, it could be iterated upon by applying it to more serious game projects to determine improvements. There are also areas where it could expand on certain tenets or provide more concrete means of implementing its ideas. For example, it could use a more explicit representation and external communication of the game's understanding of the player's knowledge.

The development of *Mythos Unbound* also resulted in the creation of several tools and technologies that could be greatly expanded. For example, the workflow system could become the basis of a more robust and generalized means of computationally representing human workflows. These workflows could be modeled in virtual environments to better analyze and understand the underlying behaviors.

This thesis opens the door to many open questions; more understanding of the benefits of gamified learning over conventional learning is needed. For instance, Does it take longer to learn gamified content? Do students learn the material better? Do they enjoy this learning more? How much overhead is involved in gamifying subject matter and can we optimize this process?

REFERENCES

[1] Tesseract Interactive. *Mythos Unbound*. Computer software. University of Arkansas, 2013.

[2] Richards, Chad. *Edufitment*. Computer software. Edufitment. Web. 12 Nov. 2013. <http://www.edufitment.com/>.

[3] Schell, Jesse. *The Art of Game Design: A Book of Lenses*. Burlington: Morgan Kaufmann, 2008.

[4] Perry, David, and Rusel DeMaria. *David Perry on Game Design: A Brainstorming Toolbox*. Boston: Charles River Media, 2009.

[5] Koster, Raph. *A Theory of Fun for Game Design*. Scottsdale, AZ: Paraglyph, 2005.

[6] Kelly, Tadhg. "Ludeme." What Games Are. Web. 11 Nov. 2013. <http://www.whatgamesare.com/ludeme.html>.

[7] Id Software. *Wolfenstein 3D*. Computer software. Apogee Software, 1992.

[8] Wisdom Tree. *Super 3D Noah's Ark*. Computer software. Wisdom Tree, 1994.

[9] Screenshot of *Wolfenstein 3D*. Digital image. Wikipedia. Wikimedia Foundation, 18 Apr. 2009. Web. 9 Apr. 2014. <http://upload.wikimedia.org/wikipedia/en/6/69/Wolf3d_pc.png>.

[10] Screenshot of *Super 3D Noah's Ark*. Digital image. Wikipedia. Wikimedia Foundation, 20 Sept. 2008. Web. 9 Apr. 2014. <http://upload.wikimedia.org/wikipedia/en/6/68/Noah%27s_Ark_3D_SNES_gameplay.gif>.

[11] Kushner, David. *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture*. New York: Random House, 2003.

[12] Schubert, Damion. "Understanding Design Space." Game Developer Magazine Apr. 2008: 37-38.

[13] Sirlin, David. "Balancing Multiplayer Games." Sirlin.Net. 17 Oct. 2008. Web. 11 Nov. 2013. <http://www.sirlin.net/articles/balancing-multiplayer-games-part-2-viable-options.html>.

[14] Terrell, Richard. "For the Feel of Design Space Pt. 2." Critical-Gaming Network. 27 June 2009. Web. 12 Nov. 2013. <http://critical-gaming.com/blog/2009/6/27/for-the-feel-of-design-space-pt2.html>.

[15] Nintendo. *Super Mario Bros.*. Computer software. Nintendo, 1985.

[16] Yust, Taylor. Image of Design Space in Length and Height. Digital Image.

[17] Yust, Taylor. Image of Design Space in Variable Jump Height. Digital Image.

[18] Yust, Taylor. Image of Design Space in Minimum Jump Height. Digital Image.

[19] "Chekhov's Gun." Wikipedia. Wikimedia Foundation, 11 Feb. 2013. Web. 12 Nov. 2013. <http://en.wikipedia.org/wiki/Chekhov's_gun>.

[20] Bogost, Ian. "The Rhetoric of Video Games." The Ecology of Games: Connecting Youth, Games, and Learning. Ed. Katie Salen. Cambridge, MA: MIT, 2008. 117-40.

[21] Blow, Jonathan. "Conflicts in Game Design." Montreal International Game Summit. Montreal. 19 Nov. 2008. Lecture.

[22] Hocking, Clint. "Ludonarrative Dissonance in Bioshock." Web log post. Click Nothing. 7 Aug. 2007. Web. 12 Nov. 2013. <http://clicknothing.typepad.com/click_nothing/2007/10/ludonarrative-d.html>.

[23] Zichermann, Gabe, and Christopher Cunningham. *Gamification by Design*. Sebastopol, CA: O'Reilly Media, 2011.

[24] Unity Technologies. *Unity*. Computer software. Unity Technologies, 2005.

[25] Skinner, B. F. *The Technology of Teaching*. New York: Appleton-Century-Crofts, 1968.

[26] Hattie, John. *Visible Learning: A Synthesis of over 800 Meta-analyses Relating to Achievement*. London: Routledge, 2008.

[27] Epstein, Michael L., Amber D. Lazarus, Tammy B. Calvano, Kelly A. Matthews, Rachel A. Hendel, Beth B. Epstein, and Gary M. Brosvic. "Immediate Feedback Assessment Technique Promotes Learning and Corrects Inaccurate First Responses." *The Psychological Record* (2002): 187-201.

[28] Cotner, Sehoya H., Bruce A. Fall, Susan M. Wick, J. D. Walker, and Paul M. Baepler. "Rapid Feedback Assessment Methods: Can We Improve Engagement and Preparation for Exams in Large-enrollment Courses?" *Journal of Science Education and Technology* 17.5 (2008): 437-43.

[29] *Khan Academy*. Web. 09 Apr. 2014. <https://www.khanacademy.org/>.

[30] "How Did Khan Academy Get Started?" Khan Academy. Khan Academy. Web. 12 Nov. 2013. <http://khanacademy.desk.com/customer/portal/articles/329316-how-did-khan-academy-get-started->.

[31] Khan, Salman. "Let's Use Video to Reinvent Education." TED2011. Long Beach Performance Arts Center, Long Beach. 2 Mar. 2011. Lecture.

[32] "Research Foundations." Khan Academy. Khan Academy. Web. 12 Nov. 2013. <https://www.khanacademy.org/coach-res/KA-in-the-classroom/classroom-vision/a/research-foundations>.

[33] Guskey, Thomas R., and Sally L. Gates. "Synthesis of Research on the Effects of Mastery Learning in Elementary and Secondary Classrooms." *Educational Leadership* 43.8 (1986): 73-80.

[34] "Introducing...the Learning Dashboard." Web log post. Khan Academy. Khan Academy, 15 Aug. 2013. Web. 12 Nov. 2013. <http://www.khanacademy.org/about/blog/post/58354379257/introducing-the-learning-dashboard>.

[35] Yust, Taylor. Screenshot of *Food Group Match Up*. Digital image.

[36] Yust, Taylor. Screenshot of *Calorie Meter: The Card Game*. Digital image.

[37] Yust, Taylor. Screenshot of *NutriHunt*. Digital image.

[38] "Game Innovation Lab." USC Cinematic Arts. University of Southern California. Web. 12 Nov. 2013. <http://interactive.usc.edu/game-innovation-lab/>.

[39] Ruiz, Susana, Ashley York, Mike Stein, Noah Keating, and Kellee Santiago. *Darfur is Dying*. Computer software. mtvU, 2006.

[40] Brinson, Peter and Kurosh ValaNejad. *The Cat and the Coup*. Computer software. Valve Corporation, 2011.

[41] Collegeology Games. *Collegeology.* Computer software. University of Southern California. Web. 12 Nov. 2013. <http://collegeology.usc.edu/>.

[42] USC Cinematic Arts. *Walden, A Game*. Computer software. University of Southern California. Web. 12 Nov. 2013. <http://cinema.usc.edu/interactive/research/walden.cfm>.

[43] Thatgamecompany. *Journey*. Computer software. Sony Computer Entertainment, 2012.

[44] Screenshot of *The Cat and the Coup*. Digital image. The Cat and the Coup. Web. 9 Apr. 2014. <http://www.thecatandthecoup.com/catc_screenShot2a.jpg>.

[45] "Games Learning Society." Games Learning Society. Games Learning Society. Web. 12 Nov. 2013. <http://www.gameslearningsociety.org/index.php>.

[46] Games Learning Society. *Citizen Science*. Computer software.

[47] Games Learning Society. *Trails Forward*. Computer software.

[48] Games Learning Society. *Fair Play*. Computer software.

[49] "Research." Games Learning Society. Games Learning Society. Web. 12 Nov. 2013. <http://www.gameslearningsociety.org/research.php>.

[50] Rohrer, Jason. *Diamond Trust of London*. Computer software. indiePub, 2011.

[51] Rohrer, Jason. *Passage*. Computer software. 2007.

[52] Persuasive Games. *Take Back Illinois*. Computer software. Persuasive Games. Web. 12 Nov. 2013. <http://www.persuasivegames.com/games/game.aspx?game=takebackillinois>.

[53] Pope, Lucas. *Papers, Please*. Computer software. 3909 LLC. Web. 12 Nov. 2013. <http://papersplea.se/>.

[54] "Teach with Portals." Teach with Portals. Valve Corporation. Web. 12 Nov. 2013. <http://www.teachwithportals.com/>.

[55] Valve Corporation. *Portal 2*. Computer software. Valve Corporation, 2011.

[56] "Physics Lesson Plans." Teach with Portals. Valve Corporation. Web. 12 Nov. 2013. <http://www.teachwithportals.com/index.php/category/lesson_plans/physics/>.

[58] Screenshot of *Papers, Please*. Digital image. Papers, Please. 3909 LLC. Web. 9 Apr. 2014. <http://papersplea.se/img/Shot05-Brothel.png>.

[59] Primer Labs. *Code Hero*. Computer software. Web. 12 Nov. 2013. <http://primerlabs.com/codehero>.

[60] Schlensker, Christian, Emelyn Baker, and Roshan Choxi. *Ruby Warrior*. Computer software. Web. 12 Nov. 2013. <https://www.bloc.io/ruby-warrior/#/>.

[61] Yust, Taylor. Screenshot of *Ruby Warrior*. Digital image.

[62] Yust, Taylor. Low-Level Screenshot of Khan Academy's Knowledge Map. Digital image.

[63] Yust, Taylor. High-Level Screenshot of Khan Academy's Knowledge Map. Digital image.

[64] Miyamoto, Shigeru. "Iwata Asks: New Super Mario Bros. Wii." Interview by Satoru Iwata. Iwata Asks. Nintendo. Web. 11 Nov. 2013. <http://iwataasks.nintendo.com/interviews/>.

[65] Miyamoto, Shigeru. "Iwata Asks: Super Mario Galaxy." Interview by Satoru Iwata.Iwata Asks. Web. 12 Nov. 2013. <http://iwataasks.nintendo.com/interviews/>.

[66] Miyamoto, Shigeru. "Exclusive Interview With Nintendo Gaming Mastermind Shigeru Miyamoto." Interview by Seth Porges. Popular Mechanics. 18 Dec. 2009. Web. 11 Nov. 2013. <http://www.popularmechanics.com/technology/gadgets/video-games/4334387>.

[67] Miyamoto, Shigeru. "Nintendo's Miyamoto: All This Talk about Our Earnings Is "silly"" Interview by James Brightman. GamesIndustry International. 12 June 2013. Web. 12 Nov. 2013. <http://www.gamesindustry.biz/articles/2013-06-12-nintendos-miyamoto-all-this-talk-about-our-earnings-is-silly>.

[68] Nintendo. *Super Mario Galaxy*. Computer software. Nintendo, 2007.

[69] Camelot Software Planning. *Golden Sun*. Computer software. Nintendo, 2001.

[70] Screenshot of *Golden Sun*'s Battle System. Digital image. Wikipedia. Wikimedia Foundation, 4 Feb. 2007. Web. 9 Apr. 2014. <http://upload.wikimedia.org/wikipedia/en/3/3b/GoldenSunBattle.png>.

[71] Screenshot of *Golden Sun*'s Overworld Gameplay. Digital image. Wikipedia. Wikimedia Foundation, 13 Mar. 2007. Web. 9 Apr. 2014. <http://upload.wikimedia.org/wikipedia/en/8/82/GoldenSunPsynergy.png>.

[72] Yust, Taylor. Screenshot of House of Octavius Quartio in *Mythos Unbound*. Digital image.

[73] UDE Dialogue Engine for Unity. Computer software. Vers. 1.4. MyBad Studios, 15 Oct. 2013. Web. 12 Nov. 2013. <https://www.assetstore.unity3d.com/#/content/4628>.

[74] Okashi RPG Kit. Computer software. Vers. 1.2.7. Okashi Itsumo, 7 Nov. 2013. Web. 12 Nov. 2013. <http://rpg-kit.com/>.

[75] Yust, Taylor. Screenshot of Hercules' Custom Animator for *Mythos Unbound*. Digital image.

[76] Yust, Taylor. Screenshot of Octavius' MythosAnimator-Compatible Animator for *Mythos Unbound*. Digital image.

[77] Lyashenko, Michael. NGUI: Next-Gen UI Kit. Computer software. Tasharen Entertainment. Vers. 3.0.5. Tasharen Entertainment, 9 Nov. 2013. Web. 12 Nov. 2013. <http://www.tasharen.com/?page_id=140>.

[78] Yust, Taylor. Screenshot of In-Game GUI in *Mythos Unbound*. Digital image.

[79] Yust, Taylor. Screenshot of *Mythos Unbound* GUI in the Unity Editor. Digital image.

[80] Yust, Taylor. Screenshot of Wax Tablet in *Mythos Unbound*. Digital image.

[81] "SCORM." Advanced Distributed Learning. Advanced Distributed Learning. Web. 12 Nov. 2013. <http://www.adlnet.gov/scorm/>.

[82] Unity-SCORM Integration Toolkit. Computer software. Advanced Distributed Learning. Vers. 1.0. Advanced Distributed Learning, 4 Jan. 2012. Web. 12 Nov. 2013. <http://www.adlnet.gov/scorm-unity-integration/>.

[83] Yust, Taylor. Screenshot of Inspection Mode in *Mythos Unbound*. Digital image.

[84] Yust, Taylor. Screenshot of Character Interaction in *Mythos Unbound*. Digital image.

[85] Yust, Taylor. Screenshot of Dinner Event in *Mythos Unbound*. Digital image.

[86] Yust, Taylor. Screenshot of Vase Puzzle Mini-game in *Mythos Unbound*. Digital image.

[87] Yust, Taylor. Screenshot of Molus Mini-game in *Mythos Unbound*. Digital image.

[88] Yust, Taylor. Screenshot of Cooking Mini-game in *Mythos Unbound*. Digital image.

[89] Yust, Taylor. Screenshot of Garden Maze Mini-game in *Mythos Unbound*. Digital image.

[90] Yust, Taylor. Screenshot of Cosmogony Level in *Mythos Unbound*. Digital image.

[91] Yust, Taylor. Screenshot of Sinews of Zeus Level in *Mythos Unbound*. Digital image.

[92] Yust, Taylor. Screenshot of Penelope's Deception Level in *Mythos Unbound*. Digital image.

[93] Yust, Taylor. Screenshot of Calypso's Island Level in *Mythos Unbound*. Digital image.

[94] Yust, Taylor. Screenshot of Scylla & Charybdis Level in *Mythos Unbound*. Digital image.

[95] Yust, Taylor. Screenshot of Telemachus Hides the Weapons Level in *Mythos Unbound*. Digital image.

[96] Yust, Taylor. Screenshot of Odysseus Slays the Suitors level in *Mythos Unbound*. Digital image.

[97] Yust, Taylor. Screenshot of Cassandra's Prophecy Level in *Mythos Unbound*. Digital image.

[98] Yust, Taylor. Screenshot of Orestes' Vengeance Level in *Mythos Unbound*. Digital image.

[99] Yust, Taylor. Screenshot of The Trial of Orestes Level in *Mythos Unbound*. Digital image.

[100] Yust, Taylor. Screenshot of Hercules Wrestles Thanatos Level in *Mythos Unbound*. Digital image.

[101] Yust, Taylor. Screenshot of Dionysus Comes to Thebes Level in *Mythos Unbound*. Digital image.

[102] Yust, Taylor. Screenshot of Pentheus and the Maenads Level in *Mythos Unbound*. Digital image.